connected and the total number of paths in the network remains constant.

The maximum path length is, of course, $k - 1$ and therefore, if $k$ is a fixed fraction of $n$, grows linearly. Thus, the parameter $k$ can be used to trade off maximum message delay (in terms of hops) against connection cost. The relatively expensive linear connection cost is the price to be paid for the exceptional fault-tolerant properties of this topology.

In a bus implementation, passing a message from any processor to any other requires traversing only a single bus. In this situation, $k$ measures the number of processors connected to each bus and thus measures bus contention. The number of buses to which a processor is connected is $r$ and therefore $r$ ports are required for each processor. The number of buses in the system is given by $b$. Thus, in a bus implementation, the parameter $k$ can be used to trade off increased bus contention (larger $k$) against increased number of ports per processor and total number of buses.

## V. Conclusions

We have presented a new family of static interconnection networks that are suitable for distributed multiprocessors, particularly those in which reliability and fault tolerance are important. These networks have superior flexibility because of the existence of a parameter that can be used for cost/performance tradeoffs. They have extremely high fault tolerance and degrade gracefully from the fully operational state as far down as any remaining connected topology.

## References


[1] B. W. Arden and H. Lee, "Analysis of chordal ring network," *IEEE Trans. Comput.*, vol. C-30, pp. 291–294, Apr. 1981.

[2] L. Bhuyan and D. P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Trans. Comput.*, vol. C-33, pp. 323–343, Apr. 1984.

[3] A. M. Despain and D. A. Patterson, "X-Tree—A tree structured multiprocessor computer architecture," in *Proc. 5th Annu. Symp. Comput. Architecture*, 1978, pp. 144–151.

[4] A.-H. Esfahanian and S. L. Hakimi, "Fault-tolerant routing in De Bruijn communication networks," *IEEE Trans. Comput.*, vol. C-34, pp. 777–788, Sept. 1985.

[5] R. A. Finkel and M. H. Solomon, "Processor interconnection strategies," *IEEE Trans. Comput.*, vol. C-29, pp. 360–371, May 1980.

[6] J. R. Goodman and C. H. Sequin, "Hypertree: A multiprocessor interconnection strategy," *IEEE Trans. Comput.*, vol. C-30, pp. 923–933, Dec. 1981.

[7] H. Hanani, "Balanced incomplete block designs and related designs," *Discrete Math.*, vol. 11, pp. 255–369, 1975.

[8] E. Horowitz and A. Zorat, "The binary tree as an interconnection network: Applications to multiprocessor systems and VLSI," *IEEE Trans. Comput.*, vol. C-30, pp. 247–253, Apr. 1981.

[9] D. K. Pradhan, "Fault-tolerant multiprocessor link and bus network architectures," *IEEE Trans. Comput.*, vol. C-34, pp. 33–45, Jan. 1985.

[10] F. P. Preparata and J. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," *Commun. ACM*, pp. 300–309, May 1981.

[11] C. S. Raghavendra, M. Gerla, and A. Avizienis, "Reliable loop topologies for large local computer networks," *IEEE Trans. Comput.*, vol. C-34, pp. 46–55, Jan. 1985.

[12] D. A. Reed and H. D. Schwetman, "Cost-performance bounds for multimicrocomputer networks," *IEEE Trans. Comput.*, vol. C-32, pp. 83–95, Jan. 1983.

[13] D. B. Skillicorn and R. Dawes, "Chordal rings as computer networks," in *Proc. 14th Manitoba Conf. Numer. Math. Comput., Cong. Numerantium*, vol. 46, 1985, pp. 79–90.

[14] A. P. Street and W. D. Wallis, *Combinatorial Theory: An Introduction,* Charles Babbage Research Centre, 1977.

[15] L. D. Wittie, "Message routing in mega-microcomputer networks," in *Proc. 3rd Symp. Comput. Architecture*, Jan. 1976, pp. 136–140.

[16] ——, "Communication structures for large networks of microcomputers," *IEEE Trans. Comput.*, vol. C-30, pp. 264–273, Apr. 1981.

[17] S. B. Wu and M. T. Liu, "A cluster structure as an interconnection network for large multimicrocomputer systems," *IEEE Trans. Comput.*, vol. C-30, pp. 254–263, Apr. 1981.


# Carry-Free Addition of Recoded Binary Signed-Digit Numbers

BEHROOZ PARHAMI

*Abstract*—**Signed-digit number representation systems have been defined for any radix $r \geq 3$ with digit values ranging over the set $\{-\alpha, \cdots, -1, 0, 1, \cdots, \alpha\}$, where $\alpha$ is an arbitrary integer in the range $r/2 < \alpha < r$. Such number representation systems possess sufficient redundancy to allow for the annihilation of carry or borrow chains and hence result in fast, propagation-free addition and subtraction. The original definition of signed-digit arithmetic precludes the case of $r = 2$ for which $\alpha$ cannot be selected in the proper range. Binary signed-digit numbers are known to allow limited-carry propagation with a somewhat more complex addition process. In this paper, we show that a special "recoded" representation of binary signed-digit numbers not only allows for carry-free addition and borrow-free subtraction but also offers other important advantages for the practical implementation of arithmetic functions. The recoding itself is totally parallel and can be performed in constant time, independent of operand lengths. It is also shown that binary signed-digit numbers compare favorably to other redundant schemes such as stored-carry and higher radix signed-digit representations.**

*Index Terms*—**Binary signed-digit numbers, computer arithmetic, number representation, recoding algorithm, redundant number systems, signed-digit arithmetic, stored-carry representation, string recoding.**

## I. Introduction

For any radix $r \geq 3$, there are one or more signed-digit (SD) number representation systems [2]–[4]. These *ordinary* SD (OSD) number systems correspond to different values of $\alpha$ in the range $r/2 < \alpha < r$, from the minimum redundancy system ($\alpha = \lfloor r/2 + 1 \rfloor$) to the maximum redundancy one ($\alpha = r - 1$), where $\alpha$ determines the set $\{-\alpha, \cdots, -1, 0, 1, \cdots, \alpha\}$ of the $2\alpha + 1$ digit values used. The most important property of OSD number representation systems is the possibility of performing carry-free addition and (by changing all the digit signs in the subtrahend) borrow-free subtraction.

The carry-free addition property of OSD number systems is best understood by a conceptual "recoding" process which replaces each $\pm\alpha$ value in the digit-by-digit interim sum of two operands by $\pm(\alpha - r)$ and an outgoing transfer digit of $\pm 1$. The new digit value, which has a magnitude $r - \alpha$ in the range $0 < r - \alpha < r/2 < \alpha$, always absorbs an incoming transfer digit of $\pm 1$, thus stopping its propagation.

Whenever long sequences of computations are to be performed on particular pieces of data, the one-time conversion and reconversion effort to/from the OSD representation is more than compensated for by the gain in computation speed. This is especially true for maximally redundant OSD representations (for which $\alpha = r - 1$), since conventional radix-$r$ numbers can be interpreted as maximally redundant OSD numbers, with no need for the initial conversion [7].

Manuscript received December 16, 1985; revised February 5, 1987, and September 4, 1987. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grants G1140 and A5515.

The author is with the School of Computer Science, Carleton University, Ottawa, Ont., Canada K1S 5B6.

IEEE Log Number 8718429.

0018-9340/88/1100-1470$01.00 © 1988 IEEE

However, maximally redundant OSD number systems may consume more storage space than the corresponding minimally redundant or intermediate OSD number systems. Potential applications of SD number representation include real-time signal processing [11], [13], high-precision scientific computation [12], [23], and computation-intensive modeling problems.

The original definition of SD arithmetic uses a symmetric digit set and precludes the case of $r = 2$, since such a *binary signed-digit* (BSD) number system possesses insufficient redundancy for the general carry-free algorithm to be applicable. This is also the reason behind the requirement that $\alpha$ be greater than $r/2$, even though $\alpha = r/2$ is a viable selection for a redundant number system if $r$ is even (see below). BSD numbers have been in practical use for representing intermediate values in 2's complement and high-speed multiplication schemes ever since multiplier recoding was introduced by Booth [6]. They have also been used in redundant quotient representation for the S-R-T division algorithm which was proposed independently by Sweeney [14], Robertson [20], and Tocher [21] and which derives its name from their initials. Limited-carry addition of BSD numbers was noted by Avizienis [2] and practically realized by Chow and Robertson [8]. In this paper, we derive a carry-free addition algorithm for specially recoded BSD numbers, thus making this special case more applicable to the design of high-speed arithmetic processors.

For the sake of our subsequent discussion, we define a *generalized signed-digit* (GSD) number system utilizing the digit set $\{-\alpha, -\alpha + 1, \cdots, \beta - 1, \beta\}$ with the conditions $\alpha \geq 0$, $\beta \geq 0$, and $\alpha + \beta + 1 > r$, where $r$ is the number representation radix [19]. The condition $\alpha + \beta + 1 = r$ results in nonredundant number representation systems which include the conventional radix-$r$ system with $\alpha = 0$ and $\beta = r - 1$ as a special case. The *redundancy index* of a GSD number system is defined as $\rho = \alpha + \beta + 1 - r$. GSD number systems cover the following systems as special cases:

binary stored-carry (BSC) number system: $r = 2$, $\alpha = 0$, $\beta = 2$, $\rho = 1$

BSD number system: $r = 2$, $\alpha = \beta = 1$, $\rho = 1$

binary stored-carry-or-borrow (BSCB) number system: $r = 2$, $\alpha = 1$, $\beta = 2$, $\rho = 2$

OSD number systems: $r/2 < \alpha = \beta < r$, $2 \leq \rho < r$.

The BSCB number system is equivalent to a redundant number representation system proposed for the design of systolic binary counters [9], [18].

## II. Binary Signed-Digit Numbers

A BSD number is a vector of digits from the set $\{\bar{1}, 0, 1\}$, where $\bar{1}$ denotes the digit value $-1$. In two-valued logic, each binary signed digit can be represented by two bits, using several possible encodings. Two natural encodings are the following.

1) The $\langle s, v\rangle$ encoding, consisting of "sign" and "value" bits for each digit, whereby $\bar{1}$, 0, and 1 are represented by $\langle 1, 1\rangle$, $\langle 0, 0\rangle$, and $\langle 0, 1\rangle$, respectively.

2) The $\langle n, p\rangle$ encoding, consisting of "negative" and "positive" flags for each digit, whereby $\bar{1}$, 0, and 1 are represented by $\langle 1, 0\rangle$, $\langle 0, 0\rangle$, and $\langle 0, 1\rangle$, respectively.

It is also possible to use a 1-out-of-3 $\langle n, o, p\rangle$ encoding to provide complete unidirectional error detection capability with a relatively low overhead in terms of added hardware complexity. This encoding is similar to the $\langle n, p\rangle$ encoding, except that the middle flag denotes the value zero.

Here, we limit our discussion to $\langle s, v\rangle$ and $\langle n, p\rangle$ encodings. If a digit $d$ is represented as $\langle d^s, d^v\rangle$ with the first and as $\langle d^n, d^p\rangle$ with the second encoding, then the following equalities hold:

$$d = (1 - 2d^s)d^v = d^p - d^n.$$

In subsequent sections of the paper, designs for circuits of interest will be given for each of these two encodings. Both $\langle s, v\rangle$ and $\langle n, p\rangle$ encodings allow the implementation of normalized significant digit arithmetic [15] if the extra combination ($\langle 1, 0\rangle$ in the $\langle s, v\rangle$ encoding and $\langle 1, 1\rangle$ in the $\langle n, p\rangle$ encoding) is used to denote nonsignificant zeros. In the remainder of this paper, we will treat the unused combinations $\langle 1, 0\rangle$ of the $\langle s, v\rangle$ encoding and $\langle 1, 1\rangle$ of the $\langle n, p\rangle$ encoding as DON'T CARE conditions to obtain simpler designs.

Any conventional binary number represented as a signed value is also a BSD number with the $\langle s, v\rangle$ encoding if the common sign is attached to each nonzero digit. In general, conversion from complement notation to BSD representation requires a complementation step. For the special case of 2's complement, attaching a $\bar{1}$ digit to the left of a negative number will produce its correct BSD representation. To reconvert from BSD to conventional representation, we proceed to eliminate the $\bar{1}$ digits as follows. Each $\bar{1}$ digit is changed to 1, generating a borrow (carry of $\bar{1}$) for the next higher position. The borrow propagates over 0's (changing them to 1's) and stops at either $\bar{1}$ or 1 (changing it to 0). A nonzero borrow coming out of the most significant position indicates that the number is negative. Using the outgoing borrow as the sign bit, we obtain the correct 2's-complement representation of the original BSD number. With an end-around borrow, the 1's-complement representation is obtained which is easily convertible to a signed value.

*Example:* Consider a BSD representation of the number —26:

| $z_6$ | $z_5$ | $z_4$ | $z_3$ | $z_2$ | $z_1$ | $z_0$ |
|---|---|---|---|---|---|---|
| 0 | $\bar{1}$ | 0 | 1 | $\bar{1}$ | 1 | 0 |

According to the above procedure, $z_2$ and $z_5$ change to 1's and generate borrows. The first borrow converts $z_3$ to 0 and stops, while the second borrow transforms $z_6$ to 1 and exits to become the sign bit. Thus, we obtain the 8-bit number

| $x_7$ | $x_6$ | $x_5$ | $x_4$ | $x_3$ | $x_2$ | $x_1$ | $x_0$ |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |

which is the correct 2's-complement representation of—26. ●

Hardware implementation of the above procedure can be based on simple ripple-borrow principle or the more advanced borrow-skip or borrow-lookahead schemes, depending on the speed desired. If ripple borrow is used, we have for the $\langle s, v\rangle$ and the $\langle n, p\rangle$ encodings

$$b_{i+1} = z_i^s z_i^v + \bar{z}_i^v b_i = z_i^n + \bar{z}_i^p b_i$$

where $b_i$ is the borrow into the $i$th position ($b_0 = 0$).

Alternatively, one can reconvert by first separating the positive and negative components of the BSD number through bit-by-bit logical operations

$$z^n \leftarrow z^s z^v$$

$$z^p \leftarrow \bar{z}^s z^v$$

and then subtracting the negative magnitude $z^n$ from the positive part $z^p$. These steps can be accomplished by conventional machine instructions or through specialized hardware (consisting of a row of $k$ single-bit two-way demultiplexors connected to the inputs of a regular adder/subtractor), again depending on the speed desired. Note that with the $\langle n, p\rangle$ encoding, $z^n$ and $z^p$ are directly available and subtraction of the two values yields the correct result even if we relax the assumption that $\langle 1, 1\rangle$ is never used to represent 0. A discussion of the various reconversion procedures for OSD numbers (which also apply to BSD numbers) can be found in [4].

## III. A Recoding Algorithm for BSD Numbers

We now derive a recoding algorithm which transforms any BSD number $x = x_{k-1}x_{k-2} \cdots x_0$ into an equivalent *recoded* BSD (RBSD) number $z = z_k z_{k-1} \cdots z_0$, such that $z_j \times z_{j-1} \neq 1$ ($1 \leq j \leq k$). This recoding is essential to the carry-free addition process, as we shall see in Section IV. It also offers other advantages which will be discussed in Section V.

Given the BSD number $x$, we start by converting it to the BSD number $y$ which has no two consecutive $\bar{1}$ digits. This can be

TABLE I
RECODING TO ELIMINATE STRINGS OF $\bar{1}$'s

| $x_i$ | $x_{i-1}$ | $y_i$ |
|---|---|---|
| $\bar{1}$ | $\bar{1}$ | 0 |
| $\bar{1}$ | 0 | 1 |
| $\bar{1}$ | 1 | 1 |
| 0 | $\bar{1}$ | $\bar{1}$ |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | $\bar{1}$ | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

TABLE II
RECODING TO ELIMINATE STRINGS OF 1's

| $y_i$ | $y_{i-1}$ | $y_{i-2}$ | $z_i$ |
|---|---|---|---|
| $\bar{1}$ | 0 | X | $\bar{1}$ |
| $\bar{1}$ | 1 | $\bar{1}$ | $\bar{1}$ |
| $\bar{1}$ | 1 | 0 | 0 |
| $\bar{1}$ | 1 | 1 | 0 |
| 0 | $\bar{1}$ | X | 0 |
| 0 | 0 | X | 0 |
| 0 | 1 | $\bar{1}$ | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | $\bar{1}$ | X | 1 |
| 1 | 0 | X | $\bar{1}$ |
| 1 | 1 | $\bar{1}$ | $\bar{1}$ |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

accomplished by the initial recoding of Table I. The resulting BSD number $y$ is then recoded according to Table II, yielding the BSD number $z$. The recoding of Table II has been constructed in such a way that the reduction of strings of 1's does not recreate adjacent $\bar{1}$'s. One of the two recodings (but never both) may increase the number of digits by 1. Thus, we write $z$ as $z_k z_{k-1} \cdots z_0$. The two recodings can easily be combined into the single-step recoding of Table III. The following two theorems show that the recoding of Table III produces the desired result ($z_j \times z_{j-1} \neq 1$), while leaving the numerical value of the original BSD number unchanged.

*Theorem 1:* If the BSD number $x = x_{k-1}x_{k-2} \cdots x_0$ is recoded according to Table III, the resulting BSD number $z = z_k z_{k-1} \cdots z_0$ has the property that $z_j \times z_{j-1} \neq 1$ for $1 \leq j \leq k$.

*Proof:* Suppose that $z_{j-1} = 1$. We show that $z_j$ cannot be 1. Let $S_h$ be the set of four-digit vectors such that if $x_i x_{i-1} x_{i-2} x_{i-3} \in S_h$, then $z_i = h$. From Table III, we have

$$S_1 = \{\bar{1}\bar{1}00,\ \bar{1}\bar{1}01,\ \bar{1}\bar{1}1X,\ \bar{1}0\bar{1}X,\ 0100,\ 0101,$$
$$011X,\ 1\bar{1}00,\ 1\bar{1}01,\ 1\bar{1}1X,\ 10\bar{1}X\}$$
$$S_{\bar{1}} = \{\bar{1}00X,\ \bar{1}01X,\ \bar{1}1\bar{1}X,\ \bar{1}10\bar{1},\ 0\bar{1}\bar{1}X,$$
$$0\bar{1}0\bar{1},\ 100X,\ 101X,\ 11\bar{1}X,\ 110\bar{1}\}.$$

By definition, $z_{j-1} = 1$ implies that $x_{j-1}x_{j-2}x_{j-3}x_{j-4} \in S_1$. Now, if $z_j = 1$, we must also have $x_j x_{j-1} x_{j-2} x_{j-3} \in S_1$. Therefore, $S_1$ must contain two vectors so that the first three elements of one are identical to the last three elements of the other. We see by inspection that this is not the case. Thus, $z_j \neq 1$. Similarly, for $z_{j-1} = \bar{1}$, we can show that $z_j \neq \bar{1}$ by examining the set $S_{\bar{1}}$. ●

*Theorem 2:* If the BSD number $x = x_{k-1}x_{k-2} \cdots x_0$ is recoded according to Table III, the resulting BSD number $z = z_k z_{k-1} \cdots z_0$ has the same numerical value as $x$; that is,

$$\sum_{i=0}^{k-1} 2^i x_i = \sum_{j=0}^{k} 2^j z_j.$$

TABLE III
THE FINAL COMBINED RECODING

| $x_i$ | $x_{i-1}$ | $x_{i-2}$ | $x_{i-3}$ | $z_i$ |
|---|---|---|---|---|
| $\bar{1}$ | $\bar{1}$ | $\bar{1}$ | X | 0 |
| | | 0 | $\bar{1}$ | 0 |
| | | 0 | 0 | 1 |
| | | 0 | 1 | 1 |
| | | 1 | X | 1 |
| $\bar{1}$ | 0 | $\bar{1}$ | X | 1 |
| | | 0 | X | $\bar{1}$ |
| | | 1 | X | $\bar{1}$ |
| $\bar{1}$ | 1 | $\bar{1}$ | X | $\bar{1}$ |
| | | 0 | $\bar{1}$ | $\bar{1}$ |
| | | 0 | 0 | 0 |
| | | 0 | 1 | 0 |
| | | 1 | X | 0 |
| 0 | $\bar{1}$ | $\bar{1}$ | X | $\bar{1}$ |
| | | 0 | $\bar{1}$ | $\bar{1}$ |
| | | 0 | 0 | 0 |
| | | 0 | 1 | 0 |
| | | 1 | X | 0 |
| 0 | 0 | X | X | 0 |
| 0 | 1 | $\bar{1}$ | X | 0 |
| | | 0 | $\bar{1}$ | 0 |
| | | 0 | 0 | 1 |
| | | 0 | 1 | 1 |
| | | 1 | X | 1 |
| 1 | $\bar{1}$ | $\bar{1}$ | X | 0 |
| | | 0 | $\bar{1}$ | 0 |
| | | 0 | 0 | 1 |
| | | 0 | 1 | 1 |
| | | 1 | X | 1 |
| 1 | 0 | $\bar{1}$ | X | 1 |
| | | 0 | X | $\bar{1}$ |
| | | 1 | X | $\bar{1}$ |
| 1 | 1 | $\bar{1}$ | X | $\bar{1}$ |
| | | 0 | $\bar{1}$ | $\bar{1}$ |
| | | 0 | 0 | 0 |
| | | 0 | 1 | 0 |
| | | 1 | X | 0 |

*Proof:* We prove this result by induction on $k$, the number of digits in $x$. For $k = 1$, there are three possible values of $x$: $\bar{1}$, 0, and 1. The recoded versions $0\bar{1}$, 00, and $1\bar{1}$ have the same values as the original numbers. Now, suppose that the numerical values of $x = x_{i-1}x_{i-2} \cdots x_0$ and its recoded version $z = z_i z_{i-1} \cdots z_0$ are the same. Consider $u = u_i x_{i-1} x_{i-2} \cdots x_0$ and its recoded version $w = w_{i+1} w_i z_{i-1} \cdots z_0$. Note that the last $i$ digits of $x$ and $u$, and thus of $z$ and $w$, are identical. To show that the numerical values of $w$ and $u$ are the same, it is sufficient to prove that $u_i = 2w_{i+1} + w_i - z_i$. The exhaustive enumeration of Table IV shows that the two sides are indeed equal for all the 37 cases of Table III. ●

The following logic equations specify a minimal two-level AND–OR (NAND-NAND) hardware realization of the recoding algorithm, defined in Table III, with the $\langle s, v \rangle$ encoding

$$\pi_0 = \bar{x}_i^v x_{i-1}^s x_{i-2}^s$$
$$\pi_1 = \bar{x}_i^v x_{i-1}^s \bar{x}_{i-2}^v x_{i-3}^s$$
$$\pi_2 = x_i^v \bar{x}_{i-1}^s x_{i-1}^v x_{i-2}^s$$
$$\pi_3 = x_i^v \bar{x}_{i-1}^s \bar{x}_{i-2}^v x_{i-3}^s$$
$$z_i^s = \pi_0 + \pi_1 + \pi_2 + \pi_3 + x_i^v \bar{x}_{i-1}^v \bar{x}_{i-2}^s$$
$$z_i^v = \pi_0 + \pi_1 + \pi_2 + \pi_3 + x_i^v \bar{x}_{i-1}^v + x_i^v x_{i-1}^s \bar{x}_{i-2}^s x_{i-2}^v$$
$$+ x_i^v x_{i-1}^s \bar{x}_{i-2}^s \bar{x}_{i-3}^s + \bar{x}_i^v \bar{x}_{i-1}^s x_{i-1}^v \bar{x}_{i-2}^s x_{i-2}^v + \bar{x}_i^v \bar{x}_{i-1}^s x_{i-1}^v \bar{x}_{i-2}^s \bar{x}_{i-3}^s.$$

A minimal two-level OR–AND (NOR–NOR) realization requires 15 gates with 58 input lines. Thus, the above AND–OR realization with 12 gates, having a total of 52 input lines, is optimal.

Less complex designs are possible by going to a three-level or four-level realization. For example, taking advantage of the particu-

TABLE IV
EXHAUSTIVE ENUMERATION FOR THE PROOF OF THEOREM 2

| $u_i$ | $x_{i-1}$ | $x_{i-2}$ | $x_{i-3}$ | $z_i$ | $w_i$ | $w_{i+1}$ | $2w_{i+1}+w_i-z_i$ |
|---|---|---|---|---|---|---|---|
| $\bar{1}$ | $\bar{1}$ | $\bar{1}$ | X | $\bar{1}$ | 0 | $\bar{1}$ | $\bar{1}$ |
| | | 0 | $\bar{1}$ | $\bar{1}$ | 0 | $\bar{1}$ | $\bar{1}$ |
| | | 0 | 0 | 0 | 1 | $\bar{1}$ | $\bar{1}$ |
| | | 0 | 1 | 0 | 1 | $\bar{1}$ | $\bar{1}$ |
| | | 1 | X | 0 | 1 | $\bar{1}$ | $\bar{1}$ |
| $\bar{1}$ | 0 | $\bar{1}$ | X | 0 | $\bar{1}$ | $\bar{1}$ | $\bar{1}$ |
| | | 0 | X | 0 | $\bar{1}$ | 0 | $\bar{1}$ |
| | | 1 | X | 0 | $\bar{1}$ | 0 | $\bar{1}$ |
| $\bar{1}$ | 1 | $\bar{1}$ | X | 0 | $\bar{1}$ | 0 | $\bar{1}$ |
| | | 0 | $\bar{1}$ | 0 | $\bar{1}$ | 0 | $\bar{1}$ |
| | | 0 | 0 | 1 | 0 | 0 | $\bar{1}$ |
| | | 0 | 1 | 1 | 0 | 0 | $\bar{1}$ |
| | | 1 | X | 1 | 0 | 0 | $\bar{1}$ |
| 0 | $\bar{1}$ | $\bar{1}$ | X | $\bar{1}$ | $\bar{1}$ | 0 | 0 |
| | | 0 | $\bar{1}$ | $\bar{1}$ | $\bar{1}$ | 0 | 0 |
| | | 0 | 0 | 0 | 0 | 0 | 0 |
| | | 0 | 1 | 0 | 0 | 0 | 0 |
| | | 1 | X | 0 | 0 | 0 | 0 |
| 0 | 0 | X | X | 0 | 0 | 0 | 0 |
| 0 | 1 | $\bar{1}$ | X | 0 | 0 | 0 | 0 |
| | | 0 | $\bar{1}$ | 0 | 0 | 0 | 0 |
| | | 0 | 0 | 1 | 1 | 0 | 0 |
| | | 0 | 1 | 1 | 1 | 0 | 0 |
| | | 1 | X | 1 | 1 | 0 | 0 |
| 1 | $\bar{1}$ | $\bar{1}$ | X | $\bar{1}$ | 0 | 0 | 1 |
| | | 0 | $\bar{1}$ | $\bar{1}$ | 0 | 0 | 1 |
| | | 0 | 0 | 0 | 1 | 0 | 1 |
| | | 0 | 1 | 0 | 1 | 0 | 1 |
| | | 1 | X | 0 | 1 | 0 | 1 |
| 1 | 0 | $\bar{1}$ | X | 0 | $\bar{1}$ | 0 | 1 |
| | | 0 | X | 0 | $\bar{1}$ | 1 | 1 |
| | | 1 | X | 0 | $\bar{1}$ | 1 | 1 |
| 1 | 1 | $\bar{1}$ | X | 0 | $\bar{1}$ | 1 | 1 |
| | | 0 | $\bar{1}$ | 0 | $\bar{1}$ | 1 | 1 |
| | | 0 | 0 | 1 | 0 | 1 | 1 |
| | | 0 | 1 | 1 | 0 | 1 | 1 |
| | | 1 | X | 1 | 0 | 1 | 1 |

lar structure of Table III, we can decompose the circuit into two stages. The first stage generates auxiliary signals designating the equivalence classes of $x_ix_{i-1}$ with respect to value patterns of $z_i$

$$a_i=1 \quad \text{iff } x_ix_{i-1} \in \{\bar{1}\bar{1}, 01, 1\bar{1}\}$$
$$b_i=1 \quad \text{iff } x_ix_{i-1} \in \{\bar{1}0, 10\}$$
$$c_i=1 \quad \text{iff } x_ix_{i-1} \in \{\bar{1}1, 0\bar{1}, 11\}$$

and distinguishing between the different $z_i$ values within each equivalence class:

$$f_i=1 \quad \text{iff } x_{i-2}=\bar{1}$$
$$g_i=1 \quad \text{iff } x_{i-2}x_{i-3} \in \{\bar{1}X, 0\bar{1}\}.$$

Referring to Table III, we note that the signals $a_i$, $b_i$, and $c_i$ designate the row groups with identical value patterns for $z_i$ (e.g., $a_i = 1$ corresponds to the value pattern 00111 in five successive rows of Table III), while $f_i$ and $g_i$ subdivide the rows within row groups (e.g., $f_i = 1$ singles out the first row in three-row groups having the value pattern $1\bar{1}\bar{1}$ for $z_i$). The resulting four-level design, which is not necessarily optimal, is as follows:

$$a_i=\bar{x}_i^v\bar{x}_{i-1}^s x_{i-1}^v+x_i^v x_{i-1}^s x_{i-1}^v$$
$$b_i=x_i^v\bar{x}_{i-1}^v$$
$$c_i=\bar{x}_i^v x_{i-1}^s x_{i-1}^v+x_i^v\bar{x}_{i-1}^s x_{i-1}^v$$
$$f_i=x_{i-2}^s$$
$$g_i=x_{i-2}^s+\bar{x}_{i-2}^v x_{i-3}^s$$
$$z_i^s=b_if_i+c_ig_i$$
$$z_i^v=b_i+a_i\bar{g}_i+c_ig_i.$$

This four-level AND–OR realization requires 14 gates with 33 input lines. However, by rewriting $a_i$ and $c_i$ as

$$a_i=x_{i-1}^v\overline{(x_i^v \oplus x_{i-1}^s)}$$
$$c_i=x_{i-1}^v(x_i^v \oplus x_{i-1}^s)$$

and using an XOR gate, a simpler four-level design with 11 gates and 23 input lines is obtained. Finally, omitting the AND gates which generate $a_i$ and $c_i$ (substituting the expressions for $a_i$ and $c_i$ in those of $z_i^s$ and $z_i^v$), we obtain a four-level design with nine gates and 21 input lines.

With the $\langle n, p\rangle$ encoding, the minimal two-level AND–OR realization of the recoding function requires 18 gates with 78 input lines. The simpler two-level OR–AND realization is as follows:

$$\begin{aligned}z_i^n=&(\bar{x}_i^n+\bar{x}_{i-1}^n)(\bar{x}_i^p+\bar{x}_{i-1}^n)(\bar{x}_{i-1}^n+\bar{x}_{i-2}^p)\\&\cdot(\bar{x}_{i-1}^p+\bar{x}_{i-2}^p)(x_i^n+x_i^p+x_{i-1}^n)(\bar{x}_{i-1}^n+x_{i-2}^n+x_{i-3}^n)\\&\cdot(x_{i-1}^n+x_{i-1}^p+\bar{x}_{i-2}^n)(\bar{x}_{i-1}^p+x_{i-2}^n+x_{i-3}^n)\\z_i^p=&(\bar{x}_i^n+\bar{x}_{i-1}^p)(\bar{x}_i^p+\bar{x}_{i-1}^p)(\bar{x}_{i-1}^n+\bar{x}_{i-2}^n)\\&\cdot(\bar{x}_{i-1}^p+\bar{x}_{i-2}^n)(x_i^n+x_i^p+x_{i-1}^p)(\bar{x}_{i-1}^n+x_{i-2}^p+\bar{x}_{i-3}^n)\\&\cdot(x_{i-1}^n+x_{i-1}^p+x_{i-2}^n)(\bar{x}_{i-1}^p+x_{i-2}^p+\bar{x}_{i-3}^n).\end{aligned}$$

This two-level OR–AND realization requires 18 gates with 56 input lines. Examining the expressions for $z_{i+1}$ and $z_{i-1}$, we note that ten sum terms in the expressions for $z_i^n$ and $z_i^p$ can be shared with adjacent recoder stages. Thus, the effective complexity is 13 gates with 45 input lines for each digit position.

A four-level realization with the $<n, p>$ encoding can be obtained in a manner similar to that of the $<s, v>$ encoding. However, the resulting circuit has 15 gates with 45 input lines and thus provides no improvement over the two-level design.

## IV. ADDITION OF RECODED BSD NUMBERS

Suppose that we have applied the recoding of Section III to a pair of BSD numbers, obtaining the recoded versions $x = x_{k-1}x_{k-2}\cdots x_0$ and $y = y_{k-1}y_{k-2}\cdots y_0$, with the properties $x_j \times x_{j-1} \neq 1$ and $y_j \times y_{j-1} \neq 1$ for $1 \leq j < k$. We now prove that the addition of $x$ and $y$ entails no carry propagation.

*Theorem 3:* In adding two BSD numbers $x = x_{k-1}x_{k-2}\cdots x_0$ and $y = y_{k-1}y_{k-2}\cdots y_0$ which satisfy the properties $x_j \times x_{j-1} \neq 1$ and $y_j \times y_{j-1} \neq 1$ for $1 \leq j < k$, the $i$th digit $s_i$ of the sum $s = s_ks_{k-1}\cdots s_0$ depends only on $x_i$, $y_i$, $x_{i-1}$, and $y_{i-1}$.

*Proof:* There are five possible values for the sum $x_{i-1} + y_{i-1}$. We prove that the carry $c_i$ entering the $i$th position is uniquely determined by $x_{i-1} + y_{i-1}$ in every case. Obviously, $c_i = 0$ when $x_{i-1} + y_{i-1} = 0$, since in this case the $(i - 1)$th position stops its incoming carry. Also, $c_i = \bar{1}$ when $x_{i-1} + y_{i-1} = -2$ and $c_i = 1$ when $x_{i-1} + y_{i-1} = 2$. We now show that $c_i = 0$ for the remaining cases of $x_{i-1} + y_{i-1} = \pm 1$, completing the proof. The two cases are similar. Thus, we consider only the case of $x_{i-1} + y_{i-1} = 1$. Clearly, in this case we have $c_i \neq \bar{1}$. To show that $c_i \neq 1$, we note that for $c_i$ to be 1, we must have $c_{i-1} = 1$. But this is impossible, since $x_{i-2} + y_{i-2} = 2$ violates either $x_{i-1} \times x_{i-2} \neq 1$ or $y_{i-1} \times y_{i-2} \neq 1$ and $x_{i-2} + y_{i-2} = 1$ with $c_{i-2} = 1$ leads to a recursive requirement which is eventually unsatisfiable for $c_0$. ●

As a result of Theorem 3, we can construct the addition table for recoded BSD numbers (Table V). The addition table contains 49 possible combinations of $x_i$, $y_i$, $x_{i-1}$, and $y_{i-1}$. The other 32 combinations are ruled out by the fact that $x$ and $y$ have been recoded. These 32 combinations can be used as DON'T CARE conditions to simplify the hardware implementation.

When an arithmetic operation is performed on recoded BSD numbers, the result does not necessarily have the $x_i \times x_{i-1} \neq 1$

property. In other words, the class of RBSD numbers is not closed under the addition algorithm which was just described. Two approaches can be used for dealing with this problem. In one approach, the numbers are recoded prior to entering the adder. Thus, either two recoding circuits are required or twice as much time will be spent for recoding. A better approach is to recode each number initially (upon entering the system) and after it emerges from the arithmetic unit. The two-stage operation of addition followed by recoding lends itself perfectly to pipelining for the purpose of increasing system throughput.

The following logic equations specify a minimal two-level OR–AND hardware realization of the addition function with the $\langle s, v\rangle$ encoding:

$$\sigma_0 = \bar{x}_i^s + y_i^s + \bar{y}_i^v$$

$$\sigma_1 = \bar{x}_i^v + y_i^v + \bar{x}_{i-1}^s + \bar{y}_{i-1}^s$$

$$\sigma_2 = x_i^v + \bar{y}_i^v + \bar{x}_{i-1}^s + \bar{y}_{i-1}^s$$

$$\sigma_3 = \bar{x}_i^s + y_i^s + \bar{x}_{i-1}^v + y_{i-1}^s + \bar{y}_{i-1}^v$$

$$\sigma_4 = x_i^v + \bar{y}_i^s + x_{i-1}^s + \bar{x}_{i-1}^v + \bar{y}_{i-1}^v$$

$$s_i^s = \sigma_0\sigma_1\sigma_2\sigma_3\sigma_4(\bar{x}_i^v + \bar{y}_i^s)(x_i^s + y_i^s + x_{i-1}^s)(x_i^s + y_i^s + y_{i-1}^s)$$

$$s_i^v = \sigma_0\sigma_1\sigma_2\sigma_3\sigma_4(x_i^s + \bar{x}_i^v + \bar{y}_i^s)(\bar{x}_i^v + \bar{y}_i^v + x_{i-1}^v)$$
$$\cdot\ (\bar{x}_i^v + \bar{y}_i^v + y_{i-1}^v)(x_i^v + y_i^v + x_{i-1}^v)(x_i^v + y_i^v + y_{i-1}^v)$$
$$\cdot\ (x_i^v + y_i^v + \bar{x}_{i-1}^s + y_{i-1}^s)(x_i^v + y_i^v + x_{i-1}^s + \bar{y}_{i-1}^s).$$

A minimal two-level AND–OR realization requires 18 gates with 78 input lines. Thus, the OR–AND realization with 17 gates, having a total of 72 input lines, is optimal.

Less complex designs are possible by going to a three-level or four-level realization. For example, if we decompose the circuit into two stages, so that the first stage computes $c_i$ from $x_{i-1}$ and $y_{i-1}$ as

$$\rho_{i-1}^s = x_{i-1}^s \oplus y_{i-1}^s$$

$$c_i^s = x_{i-1}^s y_{i-1}^s$$

$$c_i^v = x_{i-1}^v y_{i-1}^v \bar{\rho}_{i-1}^s$$

and the second stage generates $s_i$ as a function of $x_i$, $y_i$, and $c_i$ (six logic variables)

$$\rho_i^v = x_i^v \oplus y_i^v$$

$$s_i^s = c_i^s\bar{\rho}_i^v + x_i^s\bar{y}_i^v\bar{c}_i^v + \bar{x}_i^v y_i^s\bar{c}_i^v$$

$$s_i^v = c_i^v \oplus \rho_i^v,$$

then nine gates (including three XOR's) with 22 input lines will suffice. This realization can easily be transformed into a three-level circuit by noting that

$$\bar{c}_i^v = \bar{x}_{i-1}^v + \bar{y}_{i-1}^v + \rho_{i-1}^s$$

and substituting in the equations for $s_i^s$ and $s_i^v$ to obtain

$$s_i^s = x_i^s\bar{y}_i^v\bar{x}_{i-1}^v + x_i^s\bar{y}_i^v\bar{y}_{i-1}^v + x_i^s\bar{y}_i^v\rho_{i-1}^s + \bar{x}_i^v y_i^s\bar{x}_{i-1}^v + \bar{x}_i^v y_i^s\bar{y}_{i-1}^v + \bar{x}_i^v y_i^s\rho_{i-1}^s + x_{i-1}^s y_{i-1}^s\bar{\rho}_i^v$$

$$s_i^v = \rho_i^v \oplus x_{i-1}^v y_{i-1}^v \bar{\rho}_{i-1}^s.$$

This three-level realization uses 12 gates with 37 input lines.

With the $\langle n, p\rangle$ encoding, the minimal two-level AND–OR realization of the addition function requires 14 gates with 60 input lines. The simpler two-level OR–AND realization is specified by the following logic equations:

$$s_i^n = (\bar{x}_i^p + y_i^p)(x_i^p + \bar{y}_i^p)(\bar{x}_i^n + \bar{y}_i^n)(\bar{x}_{i-1}^p + \bar{y}_{i-1}^p)$$
$$\cdot\ (x_i^n + y_i^n + x_{i-1}^n)(x_i^n + y_i^n + y_{i-1}^n)$$

$$s_i^p = (\bar{x}_i^n + y_i^n)(x_i^n + \bar{y}_i^n)(\bar{x}_i^p + \bar{y}_i^p)(\bar{x}_{i-1}^n + \bar{y}_{i-1}^n)$$
$$\cdot\ (x_i^p + y_i^p + x_{i-1}^p)(x_i^p + y_i^p + y_{i-1}^p).$$

This two-level OR–AND realization requires 14 gates with 40 input lines. Examining the expressions for $s_{i+1}$ and $s_{i-1}$, we note that four of the sum terms in the expressions for $s_i^n$ and $s_i^p$ can be shared with adjacent adder stages. Thus, the effective complexity is 12 gates with 36 input lines for each digit position. This can be reduced to ten gates with 30 input lines if the product of the first two sum terms in each of the above expressions is realized by a two-input XNOR gate.

It is possible to realize a simpler adder in three levels by utilizing "transfer genrate" signals $g_{i-1}^n = x_{i-1}^n y_{i-1}^n$ and $g_{i-1}^p = x_{i-1}^p y_{i-1}^p$. Then, with the above-mentioned XNOR modification, the logic equations become

$$s_i^n = (\overline{x_i^p \oplus y_i^p})(\bar{x}_i^n + \bar{y}_i^n)(\bar{x}_{i-1}^p + \bar{y}_{i-1}^p)(x_i^n + y_i^n + g_{i-1}^n)$$

$$s_i^p = (\overline{x_i^n \oplus y_i^n})(\bar{x}_i^p + \bar{y}_i^p)(\bar{x}_{i-1}^n + \bar{y}_{i-1}^n)(x_i^p + y_i^p + g_{i-1}^p).$$

This three-level realization of the addition function requires eight gates with 22 input lines, provided that the shared sum terms between adjacent adder positions are taken into account.

## V. Advantages of BSD and RBSD Numbers

The carry-free addition property is not unique to BSD and RBSD representations. It is therefore necessary to compare BSD and RBSD numbers to other redundant schemes.

BSD and RBSD numbers require the same amount of storage as the widely used BSC representation (2 bits per digit position). Assuming two-level realization of BSD recoding and RBSD addition functions, the overall addition process is of equal speed in the two systems, since two BSC numbers are added by a two-stage carry-save adder requiring four logic levels. It is natural to ask whether BSD and RBSD numbers offer any advantage over the BSC system. The answer is positive for the following reasons:

1) ease of multiplication, division, and other arithmetic operations
2) ease of zero detection
3) suitability for use with arithmetic error codes.

Each of these points will be explained briefly.

The stored-carry representation was devised in connection with the design of fast multioperand adders [16]. Even though other arithmetic operations such as multiplication and division can be performed on stored-carry numbers, the required hardware is much more complex than those for binary, BSD, or RBSD numbers. In fact, RBSD numbers are already in the proper form for high-speed multiplication in radix 4 [22], [1], while multiplying two stored-carry numbers is much more complex unless the lower-speed radix-2 implementation is used. These claims are substantiated by the digit sets encountered in radix-4 multiplication:

Binary: $\{0, 1, 2, 3\}$

BSD: $\{\bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3\}$

RBSD: $\{\bar{2}, \bar{1}, 0, 1, 2\}$

BSC: $\{0, 1, 2, 3, 4, 5, 6\}$.

Negation of a number (change of sign) can be performed in parallel for BSD and RBSD numbers, while it is nontrivial for any redundant GSD representation with $\alpha \neq \beta$. Similar observations apply to division and other arithmetic operations such as square rooting.

Zero has a unique BSD and RBSD representation and can thus be easily detected when needed for proper sequencing of arithmetic algorithms. Zero detection is also possible with the stored-carry

TABLE V
ADDITION TABLE FOR RECODED BSD NUMBERS

| $x_i$ | $y_i$ | $x_{i-1}$ | $y_{i-1}$ | $s_i$ |
|---|---|---|---|---|
| $\bar{1}$ | $\bar{1}$ | 0 | 0 | 0 |
| | | 0 | 1 | 0 |
| | | 1 | 0 | 0 |
| | | 1 | 1 | 1 |
| $\bar{1}$ | 0 | 0 | X | $\bar{1}$ |
| | | 1 | $\bar{1}$ | $\bar{1}$ |
| | | 1 | 0 | $\bar{1}$ |
| | | 1 | 1 | 0 |
| $\bar{1}$ | 1 | 0 | $\bar{1}$ | 0 |
| | | 0 | 0 | 0 |
| | | 1 | $\bar{1}$ | 0 |
| | | 1 | 0 | 0 |
| 0 | $\bar{1}$ | X | 0 | $\bar{1}$ |
| | | $\bar{1}$ | 1 | $\bar{1}$ |
| | | 0 | 1 | $\bar{1}$ |
| | | 1 | 1 | 0 |
| 0 | 0 | $\bar{1}$ | $\bar{1}$ | $\bar{1}$ |
| | | X | 0 | 0 |
| | | $\bar{1}$ | 1 | 0 |
| | | 0 | X | 0 |
| | | 1 | $\bar{1}$ | 0 |
| | | 1 | 1 | 1 |
| 0 | 1 | $\bar{1}$ | $\bar{1}$ | 0 |
| | | X | 0 | 1 |
| | | 0 | $\bar{1}$ | 1 |
| | | 1 | $\bar{1}$ | 1 |
| 1 | $\bar{1}$ | $\bar{1}$ | 0 | 0 |
| | | $\bar{1}$ | 1 | 0 |
| | | 0 | 0 | 0 |
| | | 0 | 1 | 0 |
| 1 | 0 | $\bar{1}$ | $\bar{1}$ | 0 |
| | | $\bar{1}$ | 0 | 1 |
| | | $\bar{1}$ | 1 | 1 |
| | | 0 | X | 1 |
| 1 | 1 | $\bar{1}$ | $\bar{1}$ | $\bar{1}$ |
| | | $\bar{1}$ | 0 | 0 |
| | | 0 | $\bar{1}$ | 0 |
| | | 0 | 0 | 0 |

representation but requires more hardware, expecially for 2's-complement numbers (check that every 0 or 2 is followed by a 0 and that every 1 is followed by a 1 or a 2).

Finally, the BSD system's suitability for use with arithmetic error codes is a direct result of the compatibility of BSD representation with the binary system. In particular, effectiveness studies performed for arithmetic error codes in connection with the binary system [5], [17] carry over with little change to the BSD system. As an example of possible difficulties with error coding in the stored-carry system, consider unidirectional multiple faults affecting a single-digit position. With the $\langle n, p\rangle$ encoding of BSD numbers, such a fault may result in the error magnitude $\pm 2^j$ which is the same as the set of possible error magnitudes for binary numbers. With stored-carry representation, the set of possible error magnitudes for digit position $j$ is $\{-2^{j+1}, -2^j, 2^j, 2^{j+1}\}$ which is different from the assumption used in existing effectiveness studies and leads to a larger number of undetectable error combinations.

Higher radix SD numbers generally consume less storage space due to the fact that they need fewer extra bits to accommodate the digit signs. In the following discussion, we consider only radices of the form $r = 2^a$, since such radices allow more efficient implementation of arithmetic functions. In the extreme case of $a = k$, we have the conventional representation of $k$-bit numbers with a single sign bit.

Let us first consider the case of $r = 4$. The required digit set is $\{\bar{3}, \bar{2}, \bar{1}, 0, 1, 2, 3\}$ in OSD representation which can be represented using a 3-bit encoding for each digit. The recoding of Section III can be applied to the binary digit representations in radix-4 SD numbers to reduce the digit set to $\{\bar{2}, \bar{1}, 0, 1, 2\}$. This can be done by simply attaching the common sign to both bits and recoding as usual. Such an unconventional radix-4 SD representation is superior to both the conventional radix-4 SD and BSD representations in that it results in simple multiplication while consuming less space than BSD.

TABLE VI
THE COMPLEXITY OF VARIOUS HARDWARE REALIZATIONS*

| Encoding | BSD Recoding | | | RBSD Addition | | | Addition + Recoding | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | L | G | I | L | G | I | L | G | F | I |
| <s,v> | 2 | 12 | 52 | 2 | 17 | 72 | 4 | 29 | 12 | 124 |
| | | | | 3 | 12 | 37 | 5 | 24 | 9 | 89 |
| | | | | 4 | 9 | 22 | 6 | 21 | 9 | 74 |
| | 4 | 9 | 21 | 2 | 17 | 72 | 6 | 26 | 12 | 93 |
| | | | | 3 | 12 | 37 | 7 | 21 | 7 | 58 |
| | | | | 4 | 9 | 22 | 8 | 18 | 3 | 43 |
| <n,p> | 2 | 13 | 45 | 2 | 10 | 30 | 4 | 23 | 8 | 75 |
| | | | | 3 | 8 | 22 | 5 | 21 | 8 | 67 |

* L = Levels, G = Gates, I = Input lines, F = Maximum fan-in

Conversion from such a radix-4 representation to BSD representation is trivial.

For $r = 8$, the recoding (when applied as discussed for $r = 4$) converts the SD number to minimally redundant form, since the largest resulting digits are $\pm 5$ which correspond to RBSD patterns $10\bar{1}$ and $\bar{1}0\bar{1}$. However, this type of recoding does not simplify the multiplication process considerably for $r = 8$ and all higher radices. It is interesting to note that even though addition with limited carry propagation is possible with the digit set $\{\overline{r/2}, \cdots, \bar{1}, 0, 1, \cdots, r/2\}$, such a choice offers neither better storage efficiency nor easier multiplication compared to OSD representation for $r = 2^a$ when $a > 2$.

## VI. Conclusions

We have presented a recoding for BSD numbers which not only simplifies the addition process by eliminating carries altogether, but also speeds up the multiplication process at no extra cost in terms of circuit complexity (area used in VLSI implementation). Even though this technique can be applied to intermediate computation results in order to speed up conventional binary computers, its main application area is in the design of special-purpse arithmetic "engines" [10] which deal with long sequences of computations and/or long operands. Since conventional binary numbers are easily converted to RBSD numbers, and since reconversion can be accomplished by a single binary subtraction, compatibility between the two (conventional and RBSD) computational systems can easily be achieved.

Several hardware realizations were considered for the $\langle s, v\rangle$ and $\langle n, p\rangle$ encodings of BSD numbers. Table VI summarizes the complexity results for recoding and addition, as well as the overall complexity for addition followed by recoding. An immediate conclusion is that the $\langle n, p\rangle$ encoding is superior to the $\langle s, v\rangle$ encoding form a complexity point of view. For the sake of comparison, one might note that the addition of two BSC numbers by two stages of full adders (four logic levels) requires 18 gates with 50 input lines (maximum fan-in of 4) for each digit position. This represents a lower complexity than the four-level realizations in Table VI. However, despite this higher overall complexity for addition, BSD and RBSD representations are still useful in view of the advantages discussed in the preceding section.

## References


[1] S. F. Anderson *et al.*, "The IBM System/360 Model 91: Floating-point execution unit," *IBM J. Res. Develop.*, pp. 34-53, Jan. 1967.

[2] A. Avizienis, "Signed-digit number representation for fast parallel arithmetic," *IRE Trans. Comput.*, vol. EC-10, pp. 389-400, 1961.

[3] ——, "On a flexible implementation of digital computer arithmetic," in *Inform. Processing '62.* Amsterdam: North-Holland, 1963, pp. 664-670.

[4] ——, "Binary-compatible signed-digit arithmetic," in *AFIPS Conf. Proc., 1964 Fall Joint Comput. Conf.*, pp. 663-672.

[5] ——, "Arithmetic error codes: Cost and effectiveness studies for applications in digital system design," *IEEE Trans. Comput.*, vol. C-20, pp. 1322-1331, Nov. 1971.

[6] A. D. Booth, "A signed binary multiplication technique," *Quarterly J. Mech. Appl. Math.*, vol. 4, part 2, pp. 236-240, 1951.

[7] T. C. Chen, "Maximal redundancy signed-digit systems," in *Proc. Symp. Comput. Arithmetic*, Urbana, IL, June 1985, pp. 296-300.
[8] C. Y. Chow and J. E. Robertson, "Logical design of a redundant binary adder," in *Proc. Symp. Comput. Arithmetic,* 1978, pp. 109-115.
[9] L. J. Guibas and F. M. Liang, "Systolic stacks, queues, and counters," in *Proc. Conf. Advanced Res. VLSI*, MIT, 1982, pp. 155-164.
[10] M. J. Irwin and R. M. Owens, "Digit-pipelined arithmetic as illustrated by the paste-up system: A tutorial," *Computer*, vol. 20, pp. 61-73, Apr. 1987.
[11] L. B. Jackson, *Digital Filters and Signal Processing*. Boston, MA: Kluwer, 1986.
[12] R. W. Hockney and C. R. Jesshope, *Parallel Computers*. Bristol, England: Hilger, 1981.
[13] S-Y. Kung, R. E. Owen, and J. G. Nash, Eds., *VLSI Signal Processing II*. New York: IEEE Press, 1986.
[14] O. L. MacSorley, "High-speed arithmetic in binary computers," *Proc. IRE*, vol. 49, pp. 67-91, Jan. 1961.
[15] N. Metropolis and R. L. Ashenhurst, "Significant digit computer arithmetic," *IRE Trans. Electron. Comput.*, vol. EC-7, pp. 265-267, 1958.
[16] G. Metze and J. E. Robertson, "Elimination of carry propagation in digital computers," in *Inform. Processing '59, Proc. UNESCO Conf., June 1959,* 1960, pp. 389-396.
[17] B. Parhami and A. Avizienis, "Detection of storage errors in mass memories using low-cost arithmetic error codes," *IEEE Trans. Comput.*, vol. C-27, pp. 302-308, Apr. 1978.
[18] B. Parhami, "Systolic up/down counters with zero and sign detection," in *Proc. Symp. Comput. Arithmetic,* Como, Italy, May 1987, pp. 174-178.
[19] ——, "A general theory of carry-free and limited-carry computer arithmetic," in *Proc. Canadian Conf. VLSI (CCVLSI-87),* Winnipeg, Canada, Oct. 1987, pp. 167-172.
[20] J. E. Robertson, "A new class of digital division methods," *IEEE Trans. Electron. Comput.*, vol. C-7, pp. 218-222, Sept. 1958.
[21] K. D. Tocher, "Techniques for multiplication and division for automatic binary computers," *Quarterly J. Mech. Appl. Math.*, vol. 11, part 3, pp. 364-384, 1958.
[22] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 14-17, Feb. 1964.
[23] V. Zakharov, "Parallelism and array processing," *IEEE Trans. Comput.*, vol. C-33, pp. 45-78, Jan. 1984.

# A Distributed Algorithm for Fault Diagnosis in Systems with Soft Failures

CHE-LIANG YANG AND GERALD M. MASSON

*Abstract*—**The problem of diagnosis of soft failures at the system level in large and fully distributed networks of processors (or units) is considered. A system model in which each of the network's units is assumed to possess the ability to test (or evaluate) certain other units for the presence of failures is employed. Using this model and assuming that the total number of faulty units does not exceed a given bound, a distributed algorithm is presented which allows all the fault-free units to independently converge to correct and consistent diagnoses of the system status. This algorithm is also shown to be applicable to bounded fault situations where both units and communication links can be faulty.**

Manuscript received July 9, 1986; revised May 29, 1987 and July 31, 1987. This work was supported by the National Science Foundation under Grant ECS-8412245.

C.-L. Yang is with GTE Laboratories, Inc., Waltham, MA 02254.
G. M. Masson is with the Department of Computer Science, The Johns Hopkins University, Baltimore, MD 21218.

IEEE Log Number 8718950.

*Index Terms*—**Byzantine faults, distributed fault diagnosis, hard faults, PMC models, soft faults, syndrome, test assignment.**

## I. Introduction

The problem of diagnosing soft failures, at the system level, in large and truly distributed networks of processors (*units*) will be considered. For the purpose of fault diagnosis, it is assumed that no central facility is involved, and units in the system can exchange information with one another only through normal communication links of the network. The well-known PMC model [1], in which the network units are assumed to possess the ability to test (or evaluate) certain other units for the presence of failures, will be used as the framework of our approach. Using this model and assuming that the total number of faulty units does not exceed a given bound, say $t$, a distributed algorithm will be considered for fault diagnosis such that

1) each fault-free unit in the system is able to independently achieve correct (but, perhaps, incomplete) diagnosis of the condition (faulty or fault-free) of all the units,

2) the diagnoses performed by all the fault-free units will converge to a consistent evaluation of the system as testing proceeds,

3) when the system is inflicted with only hard failures, the diagnosis performed by each fault-free unit will be correct and complete.

Using a soft-fail model (that is, a model which assumes that it is possible for faulty units to pass tests of fault-free units), as opposed to the hard-fail model used in the earlier research efforts in this area [4]-[7], has the following advantages and therefore makes system-level fault diagnosis theory more practical for fault-tolerant distributed network design.

1) A soft-fail model can account for the behaviors of intermittent and transient faults.

2) A soft-fail model can account for the situation where units can become faulty during testing as long as the number of faulty units accumulated does not exceed the upper bound $t$.

3) A soft-fail model includes the hard-fail model as a special case.

However, regarding faulty unit identification, test outcomes with a soft-fail model in general give less information than with a hard-fail model, and the consequence of this is that while the correctness of diagnosis is desired, incompleteness sometimes is inevitable. We choose correct diagnosis (that is, a diagnosis in which no fault-free unit will be diagnosed as faulty but, perhaps, a faulty unit can remain unidentified due to insufficient information) because, otherwise, a fault-free unit can conclude that some other fault-free units are faulty, and consequently refuse to interact with them. This could result in a situation where the majority of units with which a fault-free unit interacts being faulty. This clearly would lead to the collapse of the system.

## II. Preliminaries

### A. System Diagnosis

A system $S$ is composed of $n$ units, denoted by the set $U = \{u_1, u_2, \cdots, u_n\}$. Each unit $u_i \in U$ is assigned a particular subset of the remaining units in $S$ to test. In the following, it will be assumed that no unit tests itself. The complete collection of tests in $S$ is called the *test assignment* [1] and is represented by a directed graph $G(U, E)$, where each unit $u_i \in U$ is represented by a vertex and each edge $(u_i, u_j)$ is in $E$ if and only if $u_i$ tests $u_j$. With each unit $u_i \in U$, we associate the set

$$\Gamma^{-1}(u_i) = \{u_j : (u_j, u_i) \in E\}.$$

For a set of units $X \subseteq U$, we define

$$\Gamma^{-1}(X) = \bigcup_{u_i \in X} \Gamma^{-1}(u_i) - X.$$

0018-9340/88/1100-1476$01.00 © 1988 IEEE