1. $\{(0,x,0)|x\in V_0^0\}$
2. $\{(0,0,x)|x\in V_0^0\}$
3. $\{(0,0,x)|x\in V_1^0\}$
4. $\{(0,x,\bar{x})|x\in V_1^0\}$
5. $\{(1,x,0)|x\in V_0^1\}$
6. $\{(1,x,x)|x\in V_1^1\}$

Fig. 6. A minimum test set for $(2, p)$ adders under MCFM.

1. $\{(0,x,0,x,0,x,0,...)|x\in V_0^0\}$
2. $\{(0,0,x,0,x,0,x,...)|x\in V_0^0\}$
3. $\{(0,0,x,\bar{x},0,x,\bar{x},...)|x\in V_1^0\}$
4. $\{(0,x,\bar{x},0,x,\bar{x},0,...)|x\in V_1^0\}$
5. $\{(1,\bar{x},0,x,\bar{x},0,x,...)|x\in V_1^0\}$
6. $\{(1,x,x,x,x,x,x,...)|x\in V_1^1\}$

Fig. 7. A minimum test set for (N, p) adders under MCFM.

V. Conclusion

We proved that the minimum test size of arbitrary ripple carry adders under multiple cell fault model is 11. Furthermore, the minimum test size of general (N, p) adders was also proved to be $3 \times 2^{2p} - 1$. The test set is generated from the truth table description of a full adder cell; therefore, the implementation details can be ignored for testing purposes. We also showed some guidelines to find a minimum test set. Those could be used to find minimum test sets of other circuits. The general procedure to find minimum test sets of arbitrary iterative logic arrays is still undetermined.

References

[1] W. H. Kautz, "Testing for faults in cellular logic arrays," in *Proc. 8th Annu. Symp. Switching, Automata Theory*, 1967, pp. 161–174.

[2] A. D. Friedman, "Easily testable iterative systems," *IEEE Trans. Comput.*, vol. C-22, pp. 1061–1064, Dec. 1973.

[3] T. Sridhar and J. P. Hayes, "Testing bit-sliced microprocessor," in *Dig. 9th Symp. Fault-Tolerant Comput.*, Madison, WI, June 1979, pp. 211–218.

[4] R. Parthasarathy and S. M. Reddy, "A testable design of iterative logic array," *IEEE Trans. Comput.*, vol. C-30, pp. 833–841, Nov. 1981.

[5] B. A. Prasad and F. G. Gray, "Multiple fault detection in arrays of combinational cells," *IEEE Trans. Comput.*, vol. C-24, pp. 794–802, Aug. 1975.

[6] F. J. O. Dias, "Truth-table verification of an iterative logic array," *IEEE Trans. Comput.*, vol. C-25, pp. 605–613, June 1976.

[7] W.-T. Cheng and J. H. Patel, "Multiple-fault detection in iterative logic arrays," in *Proc. 1985 Int. Test Conf.*, Philadelphia, PA, Nov. 1985, pp. 493–499.

[8] P. R. Menon and A. D. Friedman, "Fault detection in iterative logic arrays," *IEEE Trans. Comput.*, vol. C-20, pp. 524–535, May 1971.

[9] O. C. Boelens, A. D. Friedman, and P. R. Menon, "Fault location in iterative logic arrays," in *Dig. 4th Symp. Fault-Tolerant Comput.*, June 1974, pp. 1.2-1.7

[10] J. A. Abraham and D. D. Gajski, "Design of testable structure defined by simple loops," *IEEE Trans. Comput.*, vol. C-30, pp. 875–884, Nov. 1981.

[11] J. P. Shen and F. J. Ferguson, "The design of easily testable VLSI array multipliers," *IEEE Trans. Comput.*, vol. C-33, pp. 554–560, June 1984.

[12] S. C. Seth and K. L. Kodandapani, "Diagnosis of faults in linear tree networks," *IEEE Trans. Comput.*, vol. C-26, pp. 29–33, Jan. 1977.

On-the-Fly Conversion of Redundant into Conventional Representations

MILOŠ D. ERCEGOVAC AND TOMAS LANG

Abstract—**An algorithm to convert redundant number representations into conventional representations is presented. The algorithm is performed concurrently with the digit-by-digit generation of redundant forms by schemes such as SRT division. It has a step delay roughly equivalent to the delay of a carry-save adder and simple implementation. The conversion scheme is applicable in arithmetic algorithms such as nonrestoring division, square root, and on-line operations in which redundantly represented results are generated in a digit-by-digit manner, from most significant to least significant.**

Index Terms—**Conditional sum addition, conversion, most significant digit first algorithms, on-line algorithms, redundant number representation, square-root, SRT division, 2's complement.**

I. Introduction

We consider an algorithm and implementation to convert a signed integer from a redundant (signed digit) into a conventional range-complement representation. Such a conversion is necessary, for example, in SRT division or square root algorithms that produce this redundant result [1]–[5], or to convert results produced by on-line algorithms into the equivalent conventional forms [6]–[8]. The standard approach for the conversion is to separate the digit vector into two, one formed by the positive digits and the other by the negative ones, and add them in a carry-propagate adder. However, this adds the delay of the adder to the total operation time. To reduce this delay we propose here an algorithm that has the following characteristics:

- it performs the conversion on the fly, as the digits of the result (e.g., of the division or square root) are obtained in a serial fashion from most to least significant,
- it uses two conditional forms of the current result, similar to the conditional sum technique [9], and
- it has a delay which is compatible with one step of a fast division algorithm [4]. This delay is roughly of a carry-save adder.

We begin with a description of the algorithm for a radix-2 representation and then generalize to radix r.

II. Radix-2 Conversion Algorithm and Implementation

We want to convert the redundant digit-vector P (with $p_i \in \{-1, 0, 1\}$) representing the normalized fraction p into the digit-vector Q (with $q_i \in \{0, 1\}$). That is,

$$p = \sum_{i=1}^{m} p_i 2^{-i}, \qquad p_i \in \{-1, 0, 1\}$$

is to be converted into its conventional 2's complement form

$$q = -q_0 + \sum_{i=1}^{m} q_i 2^{-i}, \qquad q_i \in \{0, 1\}.$$

As mentioned before, the transformation is to be performed as the digits of P are produced, from most significant to least significant. A simple algorithm would be to form $q[k]$ such that

$$q[k] = q[k-1] + p_k 2^{-k}$$

which results in $q = q[m]$.

However, this algorithm requires the propagation of a carry when $p_k = -1$. The propagation of this carry is up to the previous bit of Q having value 1. That is, if for example,

$$q_i, q_{i+1}, \cdots, q_{k-2}, q_{k-1} = 1, 0 \cdots 0, 0$$

then after adding -2^{-k} we get

$$q_i, q_{i+1}, \cdots, q_{k-2}, q_{k-1}, q_k = 0, 1, \cdots, 1, 1, 1.$$

The approach we follow is to avoid the propagation of the carry by keeping two conditional forms, one ($A[k-1]$) expecting $p_k = 1$ or

Manuscript received September 13, 1985. This work was supported in part by the Office of Naval Research under Contract N00014-85-K-0159.

The authors are with the Department of Computer Science, University of California, Los Angeles, 90024.

IEEE Log Number 8713391.

$p_k = 0$ and the other ($B[k-1]$) expecting $p_k = -1$, and selecting the correct one when p_k is known.

That is,

$$q[k] = \begin{cases} A[k-1] + 2^{-k} & \text{if } p_k = 1 \\ A[k-1] & \text{if } p_k = 0 \\ B[k-1] + 2^{-k} & \text{if } p_k = -1. \end{cases} \quad (1)$$

To obtain this result it is necessary that

$$A[k-1] = -q_0 + \sum_{i=1}^{k-1} q_i 2^{-i} = q[k-1] \quad (2)$$

and

$$B[k-1] = A[k-1] - 2^{-(k-1)}. \quad (3)$$

The final result is obtained as

$$q = A[m].$$

We now present the recurrence to compute $A[k]$ and $B[k]$ forms. Since p is normalized, the initial conditions are

$$A[1] = \begin{cases} +1/2 \quad (0.1) & \text{if } p > 0 \quad (p_1 = +1) \\ -1/2 \quad (1.1) & \text{if } p < 0 \quad (p_1 = -1) \end{cases}$$

$$B[1] = \begin{cases} +0 \quad (0.0) & \text{if } p > 0 \quad (p_1 = +1) \\ -1 \quad (1.0) & \text{if } p < 0 \quad (p_1 = -1). \end{cases}$$

For $k > 1$

$$A[k+1] = \begin{cases} A[k] + 2^{-(k+1)} & \text{if } p_{k+1} = 1 \quad (4.1) \\ A[k] & \text{if } p_{k+1} = 0 \quad (4.2) \\ B[k] + 2^{-(k+1)} & \text{if } p_{k+1} = -1 \quad (4.3) \end{cases}$$

$$B[k+1] = \begin{cases} A[k] & \text{if } p_{k+1} = 1 \quad (4.4) \\ B[k] + 2^{-(k+1)} & \text{if } p_{k+1} = 0 \quad (4.5) \\ B[k] & \text{if } p_{k+1} = -1. \quad (4.6) \end{cases}$$

Note that none of these requires a propagation of carries.

Proof: By induction. We assume that (2) and (3) are true for k and show that they are satisfied for $k+1$.

Basis: Both (2) and (3) are satisfied for $k = 2$ by the initial conditions.

Induction Step: We consider all possible values of p_{k+1}.

For $p_{k+1} = 1$, we have

$$\begin{aligned} A[k+1] &= A[k] + 2^{-(k+1)} \text{ by (4.1)} \\ &= q[k] + p_{k+1} 2^{-(k+1)} \quad \text{by induction hypothesis} \\ &= q[k+1] \end{aligned}$$

and

$$\begin{aligned} B[k+1] &= A[k] \text{ by (4.4)} \\ &= A[k+1] - 2^{-(k+1)} \quad \text{by (4.1).} \end{aligned}$$

For $p_{k+1} = 0$ we have

$$\begin{aligned} A[k+1] &= A[k] \text{ by (4.2)} \\ &= q[k] + p_{k+1} 2^{-(k+1)} \quad \text{by induction hypothesis} \\ &= q[k+1] \end{aligned}$$

and

$$\begin{aligned} B[k+1] &= B[k] + 2^{-(k+1)} \quad \text{by (4.5)} \\ &= A[k] - 2^{-k} + 2^{-(k+1)} \quad \text{by induction hypothesis} \\ &= A[k+1] - 2^{-(k+1)} \quad \text{by (4.2).} \end{aligned}$$

For $p_{k+1} = -1$

$$\begin{aligned} A[k+1] &= B[k] + 2^{-(k+1)} \quad \text{by (4.3)} \\ &= A[k] - 2^{-k} + 2^{-(k+1)} \quad \text{by induction hypothesis} \\ &= A[k] - 2^{-(k+1)} \\ &= q[k] + p_{k+1} 2^{-(k+1)} \quad \text{by induction hypothesis} \\ &= q[k+1] \end{aligned}$$

and

$$\begin{aligned} B[k+1] &= B[k] \quad \text{by (4.6)} \\ &= A[k+1] - 2^{-(k+1)} \quad \text{by (4.3).} \end{aligned}$$

Consequently, the recurrence satisfies the conditions of the conversion. □

As an example consider the conversion of $P = 0.1101\bar{1}00\bar{1}1010$ where $\bar{1} = -1$.

k	p_k	$A[k]$	$B[k]$	$q[k]$
0	0			
1	1	0.1	0.0	0.1
2	1	0.11	0.10	0.11
3	0	0.110	0.101	0.110
4	1	0.1101	0.1100	0.1101
5	−1	0.11001	0.11000	0.11001
6	0	0.110010	0.110001	0.110010
7	0	0.1100100	0.1100011	0.1100100
8	−1	0.11000111	0.11000110	0.11000111
9	1	0.110001111	0.110001110	0.110001111
10	0	0.1100011110	0.1100011101	0.1100011110
11	1	0.11000111101	0.11000111100	0.11000111101
12	0	0.110001111010	0.110001111001	0.110001111010

The converted fraction is $q = 0.110001111010$.

Similarly, for a negative case such as $P = 0.\bar{1}0\bar{1}100\bar{1}1$ the conversion is

k	p_k	$A[k]$	$B[k]$	$q[k]$
0	0			
1	−1	1.1	1.0	1.1
2	0	1.10	1.01	1.10
3	−1	1.011	1.010	1.011
4	1	1.0111	1.0110	1.0111
5	0	1.01110	1.01101	1.01110
6	0	1.011100	1.011011	1.011100
7	−1	1.0110111	1.0110110	1.0110111
8	1	1.01101111	1.01101110	1.01101111

The converted fraction is $q = 1.01101111$.

The implementation of the algorithm requires two registers to hold $A[k]$ and $B[k]$, respectively. These registers can be shifted one bit left with insertion in the least significant bit depending on the value of p_k. They also require parallel loading to load $A[k]$ with $B[k]$ and vice versa. This implementation is shown in Fig. 1.

III. Radix-r Conversion

We now generalize the previous algorithm to the radix-r case. That is, we want to convert

$$p = \sum_{i=1}^{m} p_i r^{-i}, \; p_i \in \{-a, \cdots, 0, \cdots, a\} \qquad r/2 \le |a| \le r-1$$

into

$$q = -q_0 + \sum_{i=1}^{m} q_i r^{-i}, \qquad q_i \in \{0, \cdots, r-1\}.$$

To keep the characteristics of the algorithm, the expression (1) generalizes to

$$q[k] = \begin{cases} A[k-1] + p_k r^{-k} & \text{if } p_k \ge 0 \\ B[k-1] + (r - |p_k|) r^{-k} & \text{if } p_k < 0. \end{cases}$$

Consequently, to obtain this we need that

$$A[k-1] = q[k-1]$$

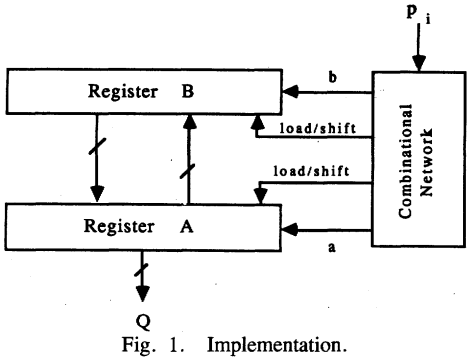

Fig. 1. Implementation.

and

$$B[k-1]=A[k-1]-r^{-(k-1)}.$$

The initial conditions extend to

$$A[1]=\begin{cases}+p_1 r^{-1}\ (0.p_1) & \text{if } p>0\\ -|p_1|r^{-1}\ (1.(r-|p_1|)) & \text{if } p<0\end{cases}$$

$$B[1]=\begin{cases}+(p_1-1)r^{-1}\ (0.(p_1-1))^* & \text{if } p>0\\ -(|p_1|+1)r^{-1}\ (1.(r-1-|p_1|)) & \text{if } p<0\end{cases}$$

*$p_1 > 0$ since p is normalized.

The recurrence presented for the radix-2 case is transformed as follows. For $k > 1$

$$A[k+1]=\begin{cases}A[k]+p_{k+1}r^{-(k+1)} & \text{if } p_{k+1}\geq 0\\ B[k]+(r-|p_{k+1}|)r^{-(k+1)} & \text{if } p_{k+1}<0\end{cases}$$

$$B[k+1]=\begin{cases}A[k]+(p_{k+1}-1)r^{-(k+1)} & \text{if } p_{k+1}>0\\ B[k]+((r-1)-|p_{k+1}|)r^{-(k+1)} & \text{if } p_{k+1}\leq 0.\end{cases}$$

The proof follows the same scheme as that for the radix-2 case.

The implementation is also similar. It requires a one-digit adder to compute $r - |p_{k+1}|$.

IV. Conversion from Radix-4 Redundant to Conventional 2's Complement

As a detailed example, we now apply the conversion algorithm to the radix-4 case, compatible with the radix-4 division scheme [4]. The conversion rules from a radix-4 redundant representation into the conventional binary 2's complement are specified in Table I. The digit appended to bit-vector $A[k]$ is $a = (a_1, a_0)$ with value $2a_1 + a_0 \in \{0, 1, 2, 3\}$. Similarly, $b = (b_1, b_0)$ is appended to bit-vector $B[k]$.

The implementation consists of two left-shift registers (A and B) that contain $A[k]$ and $B[k]$, respectively. The operations on these registers are

$$A \leftarrow \begin{cases}\text{Shift } A \text{ with insert } (a_1, a_0) & \text{if } C_{SHA}=1\\ \text{Shift } B \text{ with insert } (a_1, a_0) & \text{if } C_{LDA}=1\end{cases}$$

$$B \leftarrow \begin{cases}\text{Shift } B \text{ with insert } (b_1, b_0) & \text{if } C_{SHB}=1\\ \text{Shift } A \text{ with insert } (b_1, b_0) & \text{if } C_{LDB}=1.\end{cases}$$

By definition, $C_{LDA} = C'_{SHA}$ and $C_{LDB} = C'_{SHB}$.

For the minimally redundant case $p_i \in \{-2, -1, 0, 1, 2\}$ the radix-4 digit values are represented by the following sign-and-magnitude code:

p_i	s	m_1	m_0
0	0	0	0
1	0	0	1
2	0	1	0
−1	1	0	1
−2	1	1	0

TABLE I
RADIX-4 CONVERSION

p_{k+1}	$A[k+1]$	$B[k+1]$
0	$(A[k], 0, 0)$	$(B[k], 1, 1)$
1	$(A[k], 0, 1)$	$(A[k], 0, 0)$
−1	$(B[k], 1, 1)$	$(B[k], 1, 0)$
2	$(A[k], 1, 0)$	$(A[k], 0, 1)$
−2	$(B[k], 1, 0)$	$(B[k], 0, 1)$
3	$(A[k], 1, 1)$	$(A[k], 1, 0)$
−3	$(B[k], 0, 1)$	$(B[k], 0, 0)$

where s is the sign bit, and m_1 and m_0 are the magnitude bits. Using Table I we determine the following switching expressions for the rightmost radix-4 digits $a = (a_1, a_0)$ and $b = (b_1, b_0)$ where $a, b \in \{0, 1, 2, 3\}$:

$$a_1 = s + m_1$$

$$a_0 = m_0$$

$$b_1 = m'_0 m'_1 + s m'_1$$

$$b_0 = m'_0.$$

The shift and load control signals for the registers are also obtained from Table I:

$$C_{LDA} = s$$

$$C_{SHA} = s'$$

$$C_{LDB} = (s + m'_0 m'_1)' = C'_{SHB}$$

$$C_{SHB} = s + m'_0 m'_1.$$

V. Summary

An algorithm for converting redundant forms into range-complement conventional forms concurrently with digit-by-digit generation is presented. The algorithm has a simple implementation and a delay independent of the working precision, roughly equal to two logic levels plus a register shift/load time. It is applicable in nonrestoring division and square root algorithms, and in producing conventional results in on-line algorithms.

Acknowledgment

We thank Dr. J. G. Nash of Hughes Research Laboratories for his interest and support, and one of the referees for helpful comments.

References

[1] J. E. Robertson, "A new class of digital division methods," *IRE Trans. Electron. Comput.*, vol. EC-7, pp. 218–222, Sept. 1958.

[2] D. E. Atkins, "Higher-radix division using estimates of the divisor and partial remainders," *IEEE Trans. Comput.*, vol. C-17, pp. 925–934, Oct. 1968.

[3] G. Metze, "Minimal square rooting," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 181–185, Feb. 1965.

[4] M. D. Ercegovac and T. Lang, "A division algorithm with prediction of quotient digits," in *Proc. 7th IEEE Symp. Comput. Arithmetic*, Urbana, IL, 1985, pp. 51–56.

[5] G. S. Taylor, "Radix 16 SRT dividers with overlapped quotient selection stages," in *Proc. 7th IEEE Symp. Comput. Arithmetic*, Urbana, IL, 1985, pp. 64–71.

[6] M. D. Ercegovac, "A general hardware-oriented method for evaluation of functions and computations in a digital computer," *IEEE Trans. Comput.*, vol. C-26, pp. 667–680, July 1977.

[7] K. S. Trivedi and M. D. Ercegovac, "On-line algorithms for division and multiplication," *IEEE Trans. Comput.*, vol. C-26, pp. 667–680, July 1977.

[8] M. D. Ercegovac, "On-line arithmetic: An overview," in *Proc. SPIE 1984, Vol. 495—Real Time Signal Processing VII*, 1984, pp. 86–93.

[9] J. Sklansky, "Conditional sum addition logic," *IRE Trans. Electron. Comput.*, vol. EC-9, pp. 226–231, June 1960.