Hybrid Number Representations with Bounded Carry Propagation Chains

D. S. Phatak, I. Koren and H. Choi

Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA 01003

ABSTRACT

A novel, hybrid number representation is proposed in this paper. It includes the two's complement representation and the signed-digit representation as special cases. Such a hybrid representation is capable of bounding the maximum length of carry propagation chains during addition. Thus, one can select a specific hybrid number representation that limits the maximum length of all the carry propagation chains to any desired value between 1 and the entire word length. The framework reveals a continuum of number representations between the two extremes of two's complement and signed-digit number systems and allows a unified performance analysis of the entire spectrum of adders based on the representation selected.

1. Introduction

A redundant signed digit (SD) number representation makes it possible to perform addition with bounded carry propagation chains, and has been used to speed up arithmetic operations [1]- [5]. The term "carry-free" is used whenever the carry propagation during addition can be limited to at most 2 digit positions. The SD representation also renders most significant digit schemes feasible and has been used in on-line arithmetic and digit-pipelined schemes [4, 5]. In the Redundant Binary Signed Digit (RBSD) number system, each digit can assume any one of the three values $\{\bar{1} = -1, 0, 1\}$. This provides redundancy, i.e., a number can be represented in more than one way. For example, a 1 can be represented as 01 or as $1\bar{1}$. This redundancy can be exploited to limit the length of carry propagation chains to only 2 digit positions [2]. In other words, the carry c_i out of the ith digit position depends only on the operand digits x_i, y_i, x_{i-1}, y_{i-1}. The addition consists of two steps [6]. In the first step, an intermediate sum s_i and a carry c_i are generated, based only on 4 operand digits x_i, y_i, x_{i-1}, y_{i-1}, at each digit position i. This is done *in parallel* for all digit positions i. In the second step, the summation $z_i = s_i + c_{i-1}$ is carried out to produce the final sum digit z_i. The important point is that it is always possible to select the intermediate sum s_i and carry c_{i-1} such that the summation in the second step does not generate a carry. Hence, the second step can also be executed *in parallel* for all the digit positions, yielding a fixed addition time, irrespective of the word length. However, the speed-up in addition time does not come for free, since two bits are needed to represent a binary signed digit instead of the single bit which is sufficient for unsigned binary digits. Also, the basic adder cell that operates on signed digits is more complex than a full adder for unsigned digits. Thus, more area is traded off for the constant addition time. The SD representation is especially useful for multi-operand addition. Signed digit adder trees are easier to lay out and route than Wallace trees [3]. In [3] a 64×64 multiplier based on a RBSD adder tree was shown to yield a smaller critical path delay than the corresponding Wallace tree multiplier.

These number representations are at two extremes. In the SD number system, more bits, switching devices and routing are required per digit. In return, the carry propagation is limited to 2 digit positions. In the conventional number systems on the other hand, less bits, switches and routing are needed per digit, but the carry propagates across the entire word length. We introduce a hybrid number representation where the maximum carry propagation length can be set to any desired value between the extremes of 2 and full word length. The area required decreases in proportion with the length of the carry propagation chain. Such a representation reveals a continuum of possible realizations that trade off area for speed. This framework permits a unified analysis of performance measures based on area (A) and execution time (T).

2. Hybrid Number Representation

In the hybrid number representation, instead of insisting that every digit be a signed digit, we let some digits be signed and leave others unsigned. For example, every alternate or every third or fourth digit can be signed; all the remaining ones are unsigned. We refer to this representation as a Hybrid Signed Digit (HSD) representation. It turns out that such a representation can limit the (maximum) length of carry propagation chains to any desired value. In the follow-

1063-6404/93 $03.00 © 1993 IEEE

ing, we demonstrate that the (maximum) length of a carry propagation chain equals $[1 + d]$, where d is the (longest) distance between neighboring signed digits.

It can be verified that operations with such a representation are feasible only if the carry in between digit positions (signed or unsigned) is allowed to take any value in the set $\{-1, 0, 1\}$ as in the SD system. Without loss of generality, assume that the radix is $r = 2$ and that every alternate digit is a signed digit, for the purpose of illustration. Then, operations in the signed–digit position are the same as those in the SD case. For instance, let x_i and y_i be radix–2 signed digits to be added at the ith digit position, and c_{i-1} be the carry into the ith digit position. Each of these numbers takes any of the three values $\{-1, 0, 1\}$. Hence $-3 \leq x_i + y_i + c_{i-1} \leq +3$. This sum can be represented in terms of a signed digit output z_i and a signed carry c_i as follows:

$$x_i + y_i + c_{i-1} = 2c_i + z_i \quad \text{where } c_i, z_i \in \{-1, 0, 1\} \qquad (1)$$

In practice, the signed digit output z_i is not produced directly. Instead, the carry c_i and an intermediate sum s_i are generated in the first step. In the second step, the summation $z_i = s_i + c_{i-1}$ is performed.

The operations in the unsigned digit position are as follows. Let a_{i-1} and b_{i-1} be the bits to be added at the $(i-1)$th digit position; $a_{i-1}, b_{i-1} \in \{0, 1\}$. The carry into the $(i-1)$th position, is signed and can be $-1, 0$ or 1. The output digit denoted by e_{i-1} is restricted to be unsigned, i.e., $e_{i-1} \in \{0, 1\}$. Hence the carry out of the $(i-1)$th place must be allowed to assume the value -1 as well. In particular

if $\quad (a_{i-1} = b_{i-1} = 0 \text{ and } c_{i-2} = -1) \quad$ then

$$c_{i-1} = -1 \text{ and } e_{i-1} = 1$$

else

$$a_{i-1} + b_{i-1} + c_{i-2} = 2c_{i-1} + e_{i-1}$$
$$\text{where } c_{i-1}, e_{i-1} \geq 0$$

endif $\qquad (2)$

We next demonstrate that the carry propagates only between the signed digits. The addition consists of two steps:

Step 1 : The signed digit positions generate a carry–out and an intermediate sum based only on the two input signed digits and the two bits at the neighboring lower order unsigned digit position. For example, let x_i and y_i be the signed digits to be added in the ith position and a_{i-1} and b_{i-1} be the unsigned digits (bits) in the $(i-1)$th position. The carry c_i and intermediate sum s_i at the ith (signed) digit position are determined based on only four operands x_i, y_i, a_{i-1} and b_{i-1} according to Table 1.

x_i+y_i	x_iy_i	a_{i-1} , b_{i-1}	c_{i-1}	s_i	c_i
-2	$\bar{1}\,\bar{1}$	$\times$	$\times$	0	-1
-1	$\bar{1}\,0$	$a_{i-1} = b_{i-1} = 0$	$\{-1, 0\}$	$+1$	-1
-1	$0\,\bar{1}$	at least one of a_{i-1}, b_{i-1} is 1	$\{+1, 0\}$	-1	0
0	$\bar{1}\,1$	$\times$	$\times$	0	0
0	$1\,\bar{1}$	$\times$	$\times$	0	0
0	0 0	$\times$	$\times$	0	0
$+1$	0 1	$a_{i-1} = b_{i-1} = 0$	$\{-1, 0\}$	$+1$	0
$+1$	1 0	at least one of a_{i-1}, b_{i-1} is 1	$\{+1, 0\}$	-1	$+1$
$+2$	1 1	$\times$	$\times$	0	$+1$

Table 1: Rules to determine the carry c_i and intermediate sum s_i based on x_i, y_i, a_{i-1} and b_{i-1}.

In this table, a $\bar{1}$ denotes -1 and $\times$ denotes a "don't care". The first column of Table 1 indicates all possible values of the sum $(x_i + y_i)$. The second column indicates the individual digit values that lead to the sum in column 1. The third column indicates the possible values of a_{i-1} and b_{i-1}. Together, these columns cover all possible inputs. The fourth column indicates the possible values of c_{i-1} which is the carry into the ith (signed) digit position. This carry into the signed digit position affects the carry out of the signed digit position (viz., c_i). Note that if $(a_{i-1} = b_{i-1} = 0)$ then c_{i-1} is *non positive*, i.e., $c_{i-1} \in \{0, -1\}$. If at least one of a_{i-1} and b_{i-1} is 1, then c_{i-1} is *non negative*, i.e., $c_{i-1} \in \{0, +1\}$. The polarity of c_{i-1} as defined by these mutually exclusive conditions (both a_{i-1}, b_{i-1} are zero and at least one of them is nonzero) is valid irrespective of what values x_i and y_i assume. The last two columns indicate the values of s_i and c_i, respectively, for each possible combination of x_i, y_i, a_{i-1} and b_{i-1}.

From the table, it is clear that the carry out of the signed digit (c_i) is independent of the carry into the previous unsigned $(i-1)$th digit position, viz., c_{i-2}. Hence, the carries out of, and the intermediate sums at *all* the signed digits positions can be calculated *in parallel* in the first step. Furthermore, from the table, it is seen that whenever the carry c_{i-1} to be generated at the $(i-1)$th position is expected to be non negative

(i.e., 0 or +1), s_i is selected to be non positive (i.e., 0 or −1) and vice versa. In other words, s_i and c_{i-1} are guaranteed to have opposite polarity. Consequently, the addition $z_i = s_i + c_{i-1}$ can never generate a new carry. Thus, the carry propagation stops at the signed digit(s). The most important point is that it is possible to predict when c_{i-1} will be non positive and when it will be non negative, just by looking at the operand digits a_{i-1} and b_{i-1}. It is not necessary to wait until the actual value of c_{i-1} becomes available; which makes it possible to break the carry propagation chain.

Step 2 : In the second step, the carries generated out of the signed digit positions ripple through the unsigned digits all the way up to the next higher order signed digit position, where the propagation stops as described above. The second step can also be carried out in parallel, i.e., all the (limited) carry propagation chains between the signed digit positions are executed in parallel.

The most significant digit must be a signed digit in order to incorporate negative numbers. All the other digits can be unsigned. For example, if the word length is 32 digits, then, the 32nd digit is a signed digit. The rest of the digits are at the designer's disposal. If regularity is not necessary, one can make, for example, the 1st, 7th, 9th, and 31st (and 32nd) digits signed and let all the remaining digits be unsigned digits (bits). The addition time for such a representation is determined by the longest possible carry–propagation chain between consecutive signed digit positions (22 digit positions; from the 9th to the 31st digit in this example).

In [7] Parhami presented a unified treatment of several signed digit schemes under a general framework that was called the GSD (Generalized Signed Digit) number representation. In the GSD formulation, each digit in a radix r positional number system can take any value in the interval $[-\alpha, +\beta]$. Conditions on r, α and β that are necessary in order to perform carry–free addition were presented, and equations to perform the carry–free addition were derived. Besides the radix–r SD representation, this formulation also includes stored–borrow/stored–carry type representations as special cases. However, all the digit positions in GSD are symmetric, i.e., the range of values a digit can assume and the way the digit-wise operations are carried out is *the same* for all digit positions. Our representation, on the other hand, deliberately introduces asymmetry in the digit positions in order to reduce the transistor count (and area). Thus, some digits are allowed to be signed and others are left unsigned which makes the range of values a digit can assume non uniform. Also, the operations performed at signed and unsigned digit positions are quite different as illustrated above. The goal is to reduce the transistor count and if possible, to reduce the critical path delay as well. This is feasible because the cells in the unsigned digit positions are simpler than those in the signed digit positions and the number of bits required to represent a number is also reduced (compared to the number of bits required for the SD representation).

The special case where the number of unsigned digits between any two signed digits is the same, say d, our HSD representation can be considered to be a special case of the GSD representation with radix $r = 2^{d+1}$ [8]. When the distance between the signed digits is non-uniform, however, the HSD representation is no longer a special case of the GSD representation. Non uniform distances between signed digits, in some sense, correspond to using different radicii for different digit positions, a concept clearly beyond the scope of the GSD framework. Moreover, even in the case when the distance between the signed digits is uniform, performing the carry-free addition according to the GSD rules will lead to far more complex logic and larger area as well as a higher critical path delay. For instance, if the group of $d+1$ adjacent digits $i,\ i-1, \cdots, (i-d+1)$ is interpreted as a GSD digit with radix 2^{d+1}, then the carry out of the ith (binary) signed digit position (c_i) depends on the *values* of this and the previous (radix 2^{d+1}) digits [9]. In other words, the carry depends on all of the operands in (binary) digit positions $i-1, \cdots, i-d+1$ *as well as those in digit positions* $i-d, \cdots, i-2d+1$. The carry generation logic in such a case could be enormous. In contrast, adopting our representation and the use of Table 1 shows that the carry out c_i depends only on 4 operands, viz., those in digit positions i and $i-1$.

Static CMOS implementations of the HSD adder cells are presented in [9]. Area and Delay tradeoffs for the whole spectrum of adders (from ripple carry to full Signed Digit adder) based on the HSD representation have been analyzed in [9] and are illustrated in Figure 1. As seen in the figure, the full SD adder ($d = 0$) takes maximum area and least time while the ripple carry adder (d = word length) takes maximum time and minimum area. The HSD adders offer a continuum of choices between these two extremes: increasing d trades off execution delay for a smaller area. From the figure it is seen that the SD adder is AT optimal, while the HSD adder with $d = 1$ is A^2T optimal. Note that A^2T can be considered to be a rough estimate of [area × power × time delay] because power consumed ≈ (overall capacitance × voltage^2) and the capacitance usually grows with area, while the supply voltage is constant.

3. Conclusion

A novel, hybrid number representation has been proposed and was shown to lead to a limited carry propagation during addition. The system uses a mixture of unsigned and signed digits to represent a number. It was demonstrated that the maximum length of a carry propagation chain in such a system is limited to the (longest) distance between consecutive signed digits and can therefore be set to any desired value from 1 (or 2 if the radix is 2) to the entire word length by selecting the position(s) of the signed digits. This reveals a continuum of number representations from two's complement on one hand to the completely signed digit system system on the other. This framework was used to analyze the area and time (delay) tradeoffs associated with each representation. The framework permits a unified performance analysis of a whole spectrum of adders based on these number systems.

Several new implementations presented in [9] clearly demonstrate the feasibility of synthesizing fast and compact circuits for more complex operations by employing the proposed HSD representation. SD trees for partial product accumulation are known to be easier to lay out than Wallace trees. HSD trees could be even easier to lay out and could take less area because the number of wires to be routed at each level is smaller. In an HSD tree with $d = 1$, for instance, the number of wires at any level is only $\frac{3}{4}$th that of SD tree because every alternate digit is unsigned and uses only 1 bit instead of 2. Operations such as multiplication, division, square root extraction and elementary function evaluation by CORDIC have been accelerated by using SD representation. For these operations, the HSD representation could provide a continuum of choices between ordinary and full signed digit, that trade off increasing area for higher speed.

References

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

[2] Takagi, N., Yasuura, H., and Yajima, S., "High-speed VLSI multiplication algorithm with a redundant binary addition tree," *IEEE Transactions on Computers*, vol. C-34, pp. 789–796, Sep. 1985.

[3] Kuninobu, S., Nishiyama, T., Edamatsu, H., Taniguchi, T., and Takagi, N., "Design of high speed MOS multiplier and divider using redundant binary representation," *Proc. of the 8th Symp. on Computer Arithmetic*, pp. 80–86, 1987.

[4] Irwin, M. J. and Owens, R. M., "Digit–Pipelined Arithmetic as Illustrated by the Paste–Up System: A Tutorial ," *IEEE COMPUTER*, pp. 61–73, Apr. 1987.

[5] Ercegovac, M.D. and Lang, T., "Redundant and on-line CORDIC: application to matrix triangularization and SVD," *IEEE Transactions on Computers*, vol. C-39, pp. 725–740, Jun. 1990.

[6] Koren, I., *Computer Arithmetic Algorithms*. Prentice-Hall Inc., Englewood Cliffs, NJ, 1993.

[7] Parhami, B., "Generalized signed-digit number systems: a unifying framework for redundant number representations," *IEEE Transactions on Computers*, vol. C-39, pp. 89–98, Jan. 1990.

[8] Parhami, B., Personal Communication.

[9] Phatak, D. S., Koren I., and Choi, H., "Hybrid Signed–Digit Number Systems: A Unified Framework for Redundant Number Representations with Bounded Carry Propagation Chains," Tech. Rep. TR–93–CSE–2, ECE Dept., Univ. of Massachusetts, Amherst, Jan. 1993.

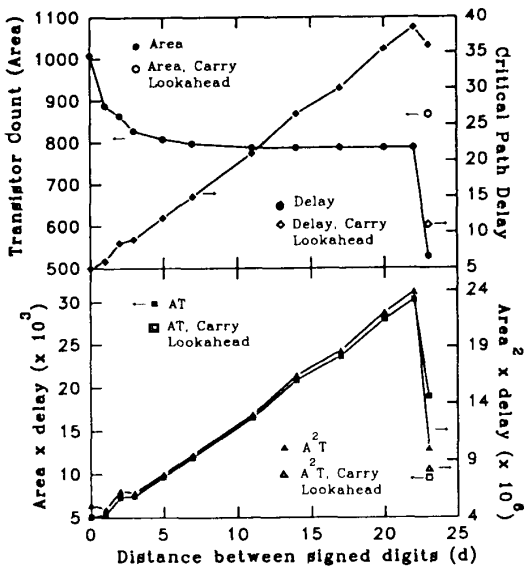

Figure 1: Area (A), Delay (T), AT and A^2T as a function of the distance d between two consecutive signed digits