# Performance Evaluation of Multi-Operand Fast Decimal Adders

Jeff Rebacz, Erdal Oruklu, and Jafar Saniie
Department of Electrical and Computer Engineering
Illinois Institute of Technology
Chicago, Illinois 60616-3793
Email: jrebacz@iit.edu, erdal@ece.iit.edu, sansonic@ece.iit.edu

***Abstract*—A renewed interest in decimal arithmetic has introduced several new designs for multi-operand decimal addition. For performance evaluation, this work implements, synthesizes, and compares four unsigned and four signed-digit multi-operand decimal adders on 0.18$\mu$m CMOS VLSI technology. Furthermore, a new architecture for signed-digit decimal adders with fast implementation is proposed. Synthesis results for 2, 4, 8, and 16 operands and 8 decimal digits provide useful statistics in determining each adder's performance and scalability. The results unambiguously highlight the advantages and disadvantages of each approach.**

## I. Introduction

Software packages have been available for most programming languages so that decimal numbers could be evaluated with decimal arithmetic to avoid error [1] [2]. IBM recently departed from this software solution by incorporating a decimal floating-point arithmetic unit in the Power6 and z10 processors [3] [4]. A compelling reason to do such is a report [5] showing that 55% of the numbers stored in the databases of 51 major organizations are decimal. If software implemented decimal arithmetic takes 100 to 1000 times longer than a hardware version [6], then hardware implemented decimal arithmetic can clearly impact performance.

Thus, research into decimal arithmetic has gained momentum. Decimal renditions of binary carry-save [7] [8] and carry-lookahead [9] [10] adders have been proposed. The main motivation behind this paper is (1) to objectively compare several recent unsigned and signed-digit multi-operand adders, and (2) to introduce a new signed-digit multi-operand adder.

Signed-digit decimal adders have the benefit of carry-free addition, although a carry-propagate adder must be used to transform the signed-digit sum into an unsigned sum. For certain applications, such schemes are powerful; the Svoboda signed-digit adder [11] has been used in a recent, novel decimal multiplier using signed-digit partial products [12].

In the next section, we will present the theory and design of a new signed-digit 2-operand decimal adder. Then, multi-operand operation based on this scheme will be introduced. In the subsequent sections, a brief description of the other adders examined in this evaluation will be given: the nonspeculative multi-operand adder [7], mixed binary and BCD adder [13], dynamic decimal CLA [14], Svoboda adder [11], speculative signed-digit adder [15], and Redundant Binary Coded Decimal (RBCD) adder [16] [17]. The synthesis results and conclusion will follow, illustrating the pros and cons of each method.

## II. Signed-Digit Theory

The decimal signed-digit set is -9 to 9 inclusive. In conventional signed-digit implementations, to add two signed-digit decimals $x_i$ and $y_i$, three quantities must be found: the intermediate sum $u_i$, the carry $c_i$ and the correction $-10 * c_i$. The operation to obtain these quantities are expressed in Equations (1) - (3). Note that the corrected sum is between -8 and 8 inclusive, so an input carry of -1 or 1 will not cause a carry to propagate to the next decimal digit.

$$u_i = x_i + y_i \tag{1}$$

$$s_i = u_i - 10 * c_i + c_{i-1} \tag{2}$$

$$c_i = \begin{cases} -1 & \text{if } u_i < -1 \\ 1 & \text{if } u_i > 1 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

An example demonstrates that the signed-digit technique still produces carries, but never propagates them. In other words, a digit's carry out is not a function of its carry in.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 3 | 1 | 3 | 5 | 2 | 8 | |
| + | | | 3 | 4 | 7 | 8 | |
| | 3 | 1 | 6 | 9 | 9 | 16 | u vector |
| | −10 | 0 | −10 | −10 | −10 | −10 | correction vector |
| 1 | 0 | 1 | 1 | 1 | 1 | | c vector |
| 1 | −7 | 2 | −3 | 0 | 0 | 6 | sum vector |

## III. Proposed Signed-Digit Decimal Adder

### A. Methodology

In the proposed signed-digit architecture, the digit set used is -9 to 9 inclusive and is represented using a conventional 5-bit, two's complement vector. For the digit at position $i$, $x_i$ and $y_i$ are added to yield a 6-bit wide intermediate sum, $u_i$ (the carry-propagate adder (CPA) chosen in these designs uses carry lookahead logic). Then, two levels of simple logic determine the carry $c_i$, which uses positive and negative magnitude components to represent {-1, 0, 1}: $c_i^-$ and $c_i^+$.

$$c_i = c_i^+ - c_i^- \tag{4}$$

978-1-4244-4480-9/09/$25.00 ©2009 IEEE

TABLE I
CORRECTION VECTOR GENERATION

| $C_i^+$ | $C_i^-$ | $C_{i-1}^+$ | $C_{i-1}^-$ | Correction Vector |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | $00000_2$ ($0_{10}$) |
| 0 | 0 | 0 | 1 | $11111_2$ (-$1_{10}$) |
| 0 | 0 | 1 | 0 | $00001_2$ ($1_{10}$) |
| 0 | 1 | 0 | 0 | $01010_2$ ($10_{10}$) |
| 0 | 1 | 0 | 1 | $01001_2$ ($9_{10}$) |
| 0 | 1 | 1 | 0 | $01011_2$ ($11_{10}$) |
| 1 | 0 | 0 | 0 | $10110_2$ (-$10_{10}$) |
| 1 | 0 | 0 | 1 | $10101_2$ (-$11_{10}$) |
| 1 | 0 | 1 | 0 | $10111_2$ (-$9_{10}$) |

Simplicity is owed to a rule set for $c_i$ that differs from (3). The main advantage is the reduced logic complexity. The new rule set can be implemented as a boolean function of 3 variables with 4 minterms as opposed to a boolean function of 6 variables with 9 minterms (for the rule set in 3). This ruleset is defined in Equation (5).

$$c_i = \begin{cases} -1 & \text{if } u_i < -8 \\ 1 & \text{if } u_i > 7 \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Each digit's positive and negative carry signals are calculated with Equations (6) - (7).

$$c_i^+ = \overline{u_i[5]}\ AND\ (u_i[4]\ OR\ u_i[3]) \quad (6)$$

$$c_i^- = u_i[5]\ AND\ (u_i[4]\ NAND\ u_i[3]) \quad (7)$$

After $u_i$ and $c_i$ are found, the correction vector must be found and added. The correction vector corresponds to $-10{*}c_i$ from Equation (2). However, since the incoming carry from the previous digit, $c_{i-1}$, must eventually be added to $u_i$ as well, it is convenient to think of the correction vector as $-10{*}c_i{+}c_{i-1}$ (only for this 2-operand adder). The correction vector (which is simply the constant term to be added) is tabulated for every possible input combination in Table I. Addition between $u_i$ and this correction vector will result in $s_i$.

The proposed SD decimal adder for one digit and 2, 3, 4, or 5 operands is shown in Figure 1a. The main operation is finding $u_i$, $c_i$, and $-10{*}c_i$, then adding $u_i{+}c_{i-1}{-}10{*}c_i$ to get the final sum. For multiple digits, there is no carry-ripple effect since carry generation does not depend on the carry in. The carry signal is the only signal transfered between consecutive digits.

### B. Parallel Addition of the Correction Vector

The previous proposed adder's correction generation and correction addition blocks can be replaced by a parallel structure to reduce delay since the bits of $u_i$ are available before the carries. In this parallel adder, fast addition of the eight constants (see Table I) is performed by two stages of logically minimized constant addition. The terminology and concept of this optimization were derived from the flag inversion cell (*fic*) sequences in [18]. The method in [18] has been simplified for adding one variable and a constant.

The addition of $r + m$ ($r$ is a variable and $m$ is a constant) is performed by inverting the 'flagged' bits in $r$. The flag at position $i$, $f[i]$, is a function of $f[i-1]$ and $r[i-1]$. A ripple effect is created but is tolerated for two reasons: the logic being rippled through is small and the ripple only travels through a few bits.

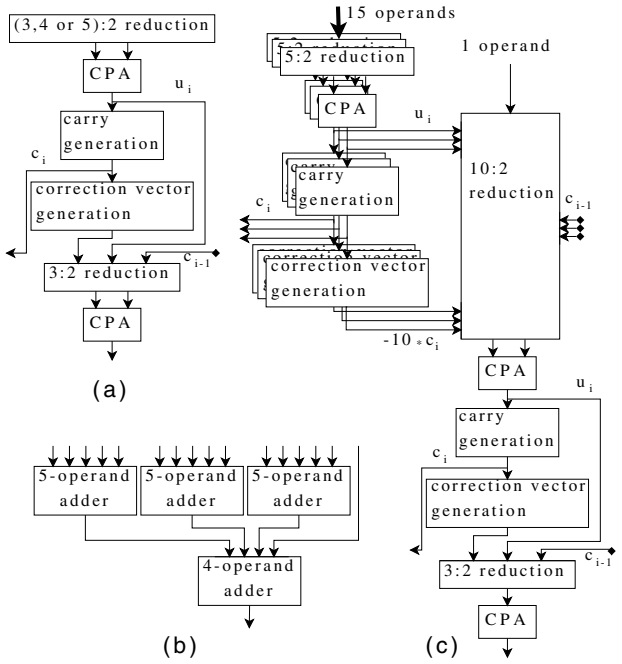


Fig. 1. In (a) the proposed signed-digit multi-operand scheme for one digit is shown. For 2-operand addition, the first n:2 reduction block does not exist and the previous digit's carry together with the current digit's carry is inputted into the correction vector generation block to find $-10 * c_i + c_{i-1}$ and eliminate the need for the 3:2 reduction block near the end. In (b), a 16-operand adder is shown constructed with 4-operand and 5-operand adders from (a). In (c), an improved 16-operand adder is shown that combines the reduction step in the first parts of the three 5-operand adders with the last parts of the 4-operand adder.

In the parallel adder, the intermediate sum, $u_i$, is fed to two *fic* sequences for adding and subtracting 10. Only the flags are calculated. A 3-to-1 multiplexer will select the flag vector for adding -10, 0 or 10 according to the selects: $c_i^+$ and $c_i^-$. The multiplexer's output is XORed with $u_i$ to invert the flagged bits. This inversion yields $u_i - 10{*}c_i$. An additional but similar stage adds the incoming carry.

## IV. MULTI-OPERAND ADDITION

Two-operand signed-digit decimal adders can be used to add multiple numbers if arranged in a tree. In this way, corrections are made after every addition. However, it is apparent that immediate correction is not necessary. The correction step can be postponed in a similar manner as shown in [7] with the addition of new constraints to the problem (see Table II).

Note that 6 or more operands cannot be added without an intermediate correction since adding 5 carries restricts $u_i - 10 * c_i$ to $[-4, 4]$. This condition cannot hold for such $u_i$ as $-5$, 5, 15. Therefore, only 2, 3, 4 and 5-operand signed-digit adders are possible with the present scheme, see Figure 1a. Furthermore, only the 2-operand adder can make use of the *fic* parallel addition technique; delay cannot be improved

TABLE II
RANGES FOR MULTI-OPERAND ADDITION

| Number of Operands | Range of $u_i$ | Maximum $u_i - 10 * c_i$ |
|---|---|---|
| 2 operands | [-18,18] | [-8,8] |
| 3 operands | [-27,27] | [-7,7] |
| 4 operands | [-36,36] | [-6,6] |
| 5 operands | [-45,45] | [-5,5] |

since the greater number of parallel additions increases the multiplexer size and the load.

However, for multi-operand addition requiring parallel modules, combining reduction where possible can yield better performance. The lower part of the SD adder that reduces and adds $u_i$, $c_{i-1}$ and $-10 * c_i$ can be combined with the earlier part of the next SD adder that reduces and adds N input operands. The operation for these parts is addition, so commutativity and associativity can be exploited.

As an example of multi-operand as well as this optimization technique, a 16-operand adder is shown in Figure 1b. Three 5-operand adders add the first 15 input operands. The $16^{th}$ input operand is added on the second level with the three sums from the first level using a 4-operand adder.

The circuit can be improved by considering the three parallel 5-operand adders together, and summing their internal $u_i$, $c_{i-1}$, and $-10 * c_i$ terms together instead of separately. Basically, instead of performing three separate 3:2 reductions for each 5-operand adder, one large 9:2 reduction is performed. Adding the $16^{th}$ input at this stage only involves growing the reduction circuit to 10:2. Best of all, one CPA is needed to produce $u_i$ for the 4-operand circuit instead of four (one at the end of each 5-operand adder, and one near the beginning of the 4-operand adder).

## V. PRIOR WORK IN UNSIGNED-DIGIT DECIMAL ADDITION

### A. Nonspeculative Multi-operand Adder

In [7], a nonspeculative multi-operand adder is presented that can sum up to 16 operands. For each digit column, $M$ operands are added using a linear array of $M-2$ binary carry-save adders. Carries outside the 4-bit range for each digit are transfered to the next column. The output of the linear array and the outside carries are inputs to combinational logic that generates a correction term to be added to obtain the final sum. Finally, the carry-outs and the corrected sum must be added with a word-wide carry-propagation adder.

### B. Mixed Binary and BCD Multi-operand Adder

This adder, proposed in [13], presents a scalable scheme to realize any N-operand addition. First, for each digit column, the N operands are added with [N:2] reduction and a CPA. Each column sum is then converted into a decimal number with a network of binary to decimal converter cells. Multiple stages of binary addition and binary to decimal conversion ensues until the number of decimal digits per column after conversion is 2.

### C. The Reduced Delay BCD Adder

In [10], the 2-operand decimal adder is composed of a 4-bit binary adder, an analyzer circuit, a carry network, and another 4-bit binary adder to add the correction vector. Using the intermediate sum from the first 4-bit binary adder, the analyzer circuit finds the generate and propagate signals for that digit. The signals from all digits are passed to a Kogge-Stone carry network to find the carries. Appropriately wiring the generated carries to the final 4-bit adder will add 0, 6, 1 or 7 based on the carries.

### D. The Dynamic Decimal Adder using Carry Lookahead

The dynamic decimal CLA adder [14] is a 2-operand adder that finds digit propagate and generate signals to be used in a carry lookahead scheme for fast carry propagation. These signals are generated per digit using combinational logic on the bit propagate and generate signals of the two input BCD digits. The sum bits are calculated as a function of the bit generate and propagate signals, the carry in, and the carry out. We have synthesized a CMOS version of the design for fairer comparison because it has been proposed as a fast dynamic logic circuit.

## VI. PRIOR WORK IN SIGNED-DIGIT DECIMAL ADDITION

### A. Svoboda Adder

The Svoboda adder [11] is a 2-operand design that adds digits from -6 to 6 inclusive. The Svoboda adder used an SD code where 5-bits represented numbers as multiples of three. This code helped simplify decimal addition. On the other hand, converting BCD operands into the Svoboda code and back requires overhead. The Svoboda adder has been used in a recent, novel decimal multiplier using signed-digit partial products [12].

### B. Speculative SD adder

This architecture [15] is rather complex as it implements a clever speculation technique that greatly facilitates the addition of input carries. The input operands are two SD decimal numbers with each digit represented by one positive 4-bit vector $x^+$ and one negative 4-bit vector $x^-$.

### C. RBCD Adder

The 2-operand Redundant Binary Coded Decimal adder (RBCD) [16] [17] adds 4-bit wide signed-digits between 7 and $-7$ inclusive. The reduced digit set dramatically simplifies carry detection at the expense of some initial overhead required to conform BCD operands to the adder's digit range. The architecture is similar to the proposed with differences in the digit set, the carry detection circuit, and the correction method.

For multi-operand addition, these 2-operand SD adders are arranged in a parallel tree. Conversions from and to BCD and the adder's internal representation are made for a fairer comparison. This way, signed and unsigned adders can be compared.

TABLE III
AREA, DELAY AND POWER COMPARISONS FOR SIGNED MULTI-OPERAND ADDERS

| Operands | Delay (ns) | Area ($\mu m^2$) | Cells | Power (mW) | Area-Delay (ns$*\mu m^2$) |
|---|---|---|---|---|---|
| Svoboda Adder | | | | | |
| 2 | 3.93 | 83716 | 2894 | 47.26 | 329003.88 |
| 4 | 5.73 | 169867 | 5973 | 69.09 | 973337.91 |
| 8 | 7.45 | 345457 | 12246 | 107.28 | 2573654.65 |
| 16 | 10.31 | 624729 | 20104 | 186.00 | 6440955.99 |
| Speculative SD Adder | | | | | |
| 2 | 3.11 | 51530 | 1574 | 41.83 | 160258.30 |
| 4 | 5.14 | 107947 | 3461 | 56.67 | 554847.58 |
| 8 | 7.00 | 235472 | 7566 | 91.62 | 1648304.00 |
| 16 | 9.96 | 466763 | 14635 | 136.80 | 4648959.48 |
| RBCD Adder | | | | | |
| 2 | 2.75 | 37989 | 1302 | 30.42 | 104469.75 |
| 4 | 4.10 | 59789 | 1982 | 33.26 | 245134.90 |
| 8 | 5.13 | 127335 | 4300 | 56.20 | 653228.55 |
| 16 | 6.54 | 258120 | 8813 | 89.57 | 1688104.80 |
| Proposed SD Adder | | | | | |
| 2 | 2.26 | 39196 | 1218 | 26.56 | 88582.96 |
| 4 | 3.28 | 60395 | 1809 | 47.70 | 198095.60 |
| 8 | 4.70 | 119497 | 3789 | 65.99 | 561635.90 |
| 16 | 5.70 | 241436 | 7883 | 115.60 | 1376185.20 |

TABLE IV
AREA, DELAY AND POWER COMPARISONS FOR UNSIGNED MULTI-OPERAND ADDERS

| Operands | Delay (ns) | Area ($\mu m^2$) | Cells | Power (mW) | Area-Delay (ns$*\mu m^2$) |
|---|---|---|---|---|---|
| Nonspeculative Adder | | | | | |
| 4 | 2.94 | 53653 | 1726 | 45.21 | 157739.82 |
| 8 | 4.56 | 105864 | 3494 | 60.86 | 482739.84 |
| 16 | 7.70 | 190356 | 6232 | 65.63 | 1465741.20 |
| Mixed Binary and BCD Adder | | | | | |
| 4 | 2.73 | 43709 | 1374 | 38.93 | 119325.57 |
| 8 | 3.70 | 71325 | 2291 | 49.52 | 263902.50 |
| 16 | 5.28 | 129503 | 4182 | 64.99 | 683775.84 |
| Reduced Delay BCD Adder | | | | | |
| 2 | 1.58 | 21690 | 628 | 28.51 | 34270.20 |
| 4 | 2.93 | 53958 | 1755 | 41.37 | 158096.94 |
| 8 | 4.31 | 117052 | 3999 | 64.04 | 504494.12 |
| 16 | 5.41 | 234625 | 8154 | 104.85 | 1269321.25 |
| Dynamic Decimal using CLA Adder | | | | | |
| 2 | 1.20 | 19736 | 573 | 29.58 | 23683.20 |
| 4 | 2.30 | 46060 | 1587 | 38.09 | 105938.00 |
| 8 | 3.28 | 98229 | 3629 | 58.73 | 322191.12 |
| 16 | 4.06 | 196183 | 7343 | 99.23 | 796502.98 |

## VII. RESULTS AND CONCLUSION

All designs were written in Verilog HDL. Synthesis results for area, timing and power were obtained from Synopsys Design Compiler using TSMC 0.18$\mu$m standard cell technology. Each design was compiled as an 8-digit (decimal) adder. Every adder inputs decimal digits in the BCD representation, and outputs an unsigned BCD sum vector.

Table III shows that the new SD adder design proposed in this work outperforms the existing SD designs' area-delay product for all operands. The RBCD adder occupies the least area for 2 and 4 operands, but falls second to the proposed adder for 8 and 16 operands. For hardware designs that can benefit from the two's complement and SD BCD representation, the proposed architecture should be considered.

Among the unsigned adders in Table IV, the mixed binary and BCD approach yields the best area and area-delay product, while the dynamic decimal adder using CLA yields the best delay. It may be speculated that the mixed binary and BCD approach will scale best, since the main growing component is the fast tree of binary adders.

## REFERENCES


[1] (2008) BigDecimal. [Online]. Available: http://java.sun.com/products

[2] (2008) alphaworks: decnumber. [Online]. Available: http://www.alphaworks.ibm.com/tech/decnumber

[3] E. Schwarz and S. Carlough, "Power6 decimal divide," in *Application -specific Systems, Architectures and Processors, 2007. ASAP. IEEE International Conf. on*, July 2007, pp. 128–133.

[4] C. Webb, "IBM z10: The next-generation mainframe microprocessor," *Micro, IEEE*, vol. 28, no. 2, pp. 19–29, March-April 2008.

[5] A. Tsang and M. Olschanowsky, "A study of database 2 customer queries," IBM Santa Teresa Laboratory, San Jose, CA, USA, Tech. Rep. TR-03.413, Apr. 1991.

[6] M. Cowlishaw, "Decimal floating-point: algorism for computers," in *Computer Arithmetic, 2003. Proceedings. 16th IEEE Symposium on*, June 2003, pp. 104–111.

[7] R. Kenney and M. Schulte, "High-speed multioperand decimal adders," *Computers, IEEE Transactions on*, vol. 54, no. 8, pp. 953–963, Aug. 2005.

[8] I. D. Castellanos and J. E. Stine, "Compressor trees for decimal partial product reduction," in *GLSVLSI '08: Proceedings of the 18th ACM Great Lakes symposium on VLSI*. New York, NY, USA: ACM, 2008, pp. 107–110.

[9] I.S. Hwang, "High speed binary and decimal arithmetic unit," *United States Patent*, no. 4,866,656, September 1989.

[10] A. Bayrakci and A. Akkas, "Reduced delay BCD adder," in *Application -specific Systems, Architectures and Processors, 2007. ASAP. IEEE International Conf. on*, July 2007, pp. 266–271.

[11] A. Svoboda, "Decimal adder with signed digit arithmetic," *Computers, IEEE Transactions on*, vol. C-18, no. 3, pp. 212–215, March 1969.

[12] M. Erle, E. Schwarz, and M. Schulte, "Decimal multiplication with efficient partial product generation," in *Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on*, June 2005, pp. 21–28.

[13] L. Dadda, "Multioperand parallel decimal adder: A mixed binary and bcd approach," *Computers, IEEE Transactions on*, vol. 56, no. 10, pp. 1320–1328, Oct. 2007.

[14] Y. You, Y. D. Kim, and J. H. Choi, "Dynamic decimal adder circuit design by using the carry lookahead," in *Design and Diagnostics of Electronic Circuits and systems, 2006 IEEE*, 0-0 2006, pp. 242–244.

[15] J. Moskal, E. Oruklu, and J. Saniie, "Design and synthesis of a carry-free signed-digit decimal adder," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, May 2007, pp. 1089–1092.

[16] B. Shirazi, D. Yun, and C. Zhang, "RBCD: redundant binary coded decimal adder," in *Computers and Digital Techniques, IEE Proceedings E*, vol. 136, no. 2, Mar 1989, pp. 156–160.

[17] ——, "VLSI designs for redundant binary-coded decimal addition," in *Computers and Communications, 1988. Conference Proceedings., Seventh Annual International Phoenix Conference on*, Mar 1988, pp. 52–56.

[18] V. Dave, E. Oruklu, and J. Saniie, "Design and synthesis of a three input flagged prefix adder," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*, May 2007, pp. 1081–1084.