Performance Evaluation of a Sum Error Detection Scheme for Decimal Arithmetic

Clara Schaertl Short and Earl E. Swartzlander, Jr.
Department of Electrical and Computer Engineering
University of Texas at Austin, Austin, TX 78712
Email: clarity@utexas.edu, eswartzla@aol.com

***Abstract*—Vázquez and Antelo's modified long-residue checker (LRC) for detecting errors in mixed binary/decimal addition and subtraction is evaluated for performance and correctness. The proposed design is shown not to detect a certain class of errors, and a corrected design is presented. The first available post-route area and delay figures for this design are obtained in a 32/28nm technology. The proposed checker's area savings and delay penalty (compared to a dual-modular redundant adder) are much smaller than predicted, suggesting that the digit corrections required to make an LRC scheme compatible with decimal addition are roughly as complex as decimal addition itself.**

***Index Terms*—Adders, CMOS integrated circuits, digital arithmetic, integrated circuit reliability, logic design.**

I. Introduction and Related Work

As CMOS logic densities increase and transistor sizes decrease, online error detection is becoming more and more important for acceptable reliability. The long residue checker (LRC) circuit [1] detects all errors in a fixed-point adder with a simple carry-save addition between the operands and the one's complement of the result. For two addends X and Y:

$$\begin{aligned} SV &= X \oplus Y \oplus \overline{S} \\ CV &= XY \vee (X \vee Y)\overline{S} \end{aligned}$$

The carry term $2 \times CV$ is equal to the one's complement of the sum term SV if and only if the adder output is correct:

$$\begin{aligned} X + Y &= S \\ X + Y + \overline{S} &= S + \overline{S} \\ SV + 2 \times CV &= 111 \cdots 1_2 \\ SV + 2 \times CV &= SV + \overline{SV} \\ 2 \times CV &= \overline{SV} \end{aligned} \tag{1}$$

The identity in (1) holds for all values of X and Y, resulting in robust error detection, and can be checked using only bitwise operations (i.e. an array of full adders with no carry propagation required), reducing the area needed for error checking compared to a dual-modular redundant (DMR) adder.

A. Two's-Complement Binary Error Checking

(1) may be extended to support two's-complement arithmetic by conditionally complementing Y at the input of the carry-save adder and by appending a carry-in bit to CV at the output of the carry-save adder:

$$\begin{aligned} Y^B &= Y \oplus sub \\ SV &= X \oplus Y^B \oplus \overline{S} \\ CV &= XY^B \vee (X \vee Y^B)\overline{S} \end{aligned}$$

For two's-complement subtraction, i.e. $sub = 1$:

$$\begin{aligned} X - Y &= S \\ X - Y - S &= 0 \\ X + (\overline{Y} + 1) + (\overline{S} + 1) &= 0 \\ (X + Y^B + \overline{S}) + 1 &= -1 \\ (SV + 2 \times CV) + sub &= 111 \cdots 1_2 \\ SV + 2 \times CV + sub &= SV + \overline{SV} \\ 2 \times CV + sub &= \overline{SV} \end{aligned} \tag{2}$$

For two's-complement addition, $sub = 0$, and (2) becomes equivalent to (1).

B. Ten's-Complement Decimal Error Checking

Vázquez and Antelo [2] describe an LRC-based error checker modified to support ten's-complement binary-coded decimal (BCD) arithmetic. Let F be the (pseudo)-BCD value whose binary representation is all ones, i.e. $F = \sum_i 15 \times 10^i$. Ten's-complement addition may be checked with:

$$\begin{aligned} X + Y &= S \\ X + Y - S &= 0 \\ X + Y + (F - S) &= F \\ X + Y + \overline{S} &= F \end{aligned} \tag{3}$$

and continuing as in (1). Subtraction may be checked with:

$$\begin{aligned} X - Y &= S \\ X - Y - S &= 0 \\ X - Y + \overline{S} &= F \\ X + (Y^D + sub) + \overline{S} &= F \end{aligned} \tag{4}$$

and continuing as in (2), where Y^D is the conditional nine's complement of Y:

$$Y_i^D = \begin{cases} Y_i, & sub = 0 \\ \overline{Y_i + 6}, & sub = 1 \end{cases} \tag{5}$$

To calculate the left-hand side of (4) without carry propagation, [2] describes three correction steps. First, the carry-in signal to each digit is eliminated by approximating the carry function

$$C_{i+1} = (X_i + Y_i^D + C_i \geq 10)$$

with the alive (i.e. carry-propagate) function

$$A_i = (X_i + Y_i^D \geq 9). \tag{6}$$

Second, the carry-out signal from each digit is eliminated by adding 6 to each decimal digit where $A_i = 1$. Finally, the error introduced by substituting A_i for C_{i+1} is corrected by subtracting 6 from each decimal digit $\overline{S_i}$ where $A_i = 1$ and $C_{i+1} = 0$. Since the only possible values for S_i when $A_i = 1$ are 9 (when $C_{i+1} = 0$) and 0 (when $C_{i+1} = 1$), this last correction is implemented as a simple conditional bitwise operation [2, Fig. 5(c)]:

$$S_i^B = \overline{S_i} \wedge \begin{cases} 1001_2, & A_i = 1 \text{ and } S_{i,3} = 1 \\ 1111_2, & A_i = 0 \text{ or } S_{i,3} = 0 \end{cases} \tag{7}$$

Continuing from (4), the carry-save addition becomes:

$$\begin{aligned} X + (Y^D + sub) + S^B &= F \\ SV + 2 \times CV + sub &= F \\ 2 \times CV + sub &= \overline{SV} \end{aligned} \tag{8}$$

which is identical to (2).

The proposed architecture is presented along with a set of favorable area and delay estimates. However, these estimates are hand-calculated and given in terms of FO4 inverter delays and NAND2 cell areas[1] for a generic CMOS technology. No reference implementation is provided.

II. Methods

All code described in this section will be made available under the MIT License at https://arith.cshort.io/ following publication.

A. Implementation

A parameterized Verilog model is developed for the "mixed binary/BCD addition/subtraction error checker" described above and in [2, Sec. II-C] . The checker itself is fairly straightforward (less than 50 lines of code), but the adder being checked (and, by extension, the error checker's testbench) requires additional effort, as Verilog lacks native support for BCD arithmetic. The `$sformat` and `$sscanf` system tasks [4, Clause 17.2] can convert an integer to a hexadecimal string and read it back as a decimal value in testbench code, but are obviously not synthesizable. Instead, a ten's-complement binary/BCD adder similar to [5] is implemented, consisting of a digit-wise addition stage, a carry-propagation stage, and a final digit-correction stage.

[1]One FO4 delay is the delay of a minimum-size inverter with a fan-out of four minimum-size inverters; one NAND2 area is the area of the standard cell that implements a minimum-size two-input NAND gate. The Synopsys 32/28nm educational library [3] has a nominal FO4 delay of 0.03 ns and a NAND2 area of 2.54 μm^2.

This poses another implementation problem: while the first and last stages are simple (again, only about 50 lines of code), the available synthesis tools only infer efficient carry-lookahead circuits as part of their proprietary datapath IP, which lacks integer BCD arithmetic blocks. An additional Verilog model is implemented that generates synthesizable Ladner-Fischer trees is implemented and used in its minimum-depth configuration for both the binary and BCD adders. This model improves on previously available implementations such as [6] by supporting any power-of-two bit width, providing an adjustable tradeoff between gate count and logic depth, and having a simple recursive structure that corresponds directly to the original figures in [7].

B. Validation

The adders are validated through exhaustive simulation at an input width of 16 bits (4 decimal digits), which is sufficient to exercise all the bitwise and digit-wise logic in the adder, as well as the base case and both recursive cases for the Ladner-Fischer tree generator. Validating both addition and subtraction this way requires $(10^4)^2 \times 2 = 2 \times 10^8$ BCD test cases (or about 20 minutes of simulation time) and $(2^{16})^2 \times 2 \approx 9 \times 10^9$ binary test cases (or an overnight simulation run). The error checkers are validated during the same simulation runs; in each test case, the input to the error checker is the correct sum XORed with a random error value, which is zero (i.e. no error) in 50% of the test cases and uniformly distributed in the other 50%.

C. Performance Evaluation

The design is synthesized, placed, and routed using the Synopsys 32/28nm Educational Design Kit [3] at widths from 32 to 512 bits. Mixed binary/BCD adders are synthesized with no error checking, with dual-modular redundancy, and with the proposed error checker. For comparison purposes, a non-redundant, minimum-depth Ladner-Fischer adder [7] is also synthesized, as well as a fast parallel-prefix adder from Synopsys's DesignWare library [8]. BCD support is not implemented in the latter two adders.

III. Results

A. Validation

Although the 16-bit (4-digit) simulation does not exhaustively verify the error checkers, it does identify one type of error that is not detected by the proposed architecture. Specifically, errors that change a sum digit S_i from 9 to an invalid BCD value, e.g., flipping a bit from $1001_2 = 9_{10}$ to $1101_2 = 13_{10}$, are masked by the digit-wise sum correction step. This class of errors can be detected by replacing the AND operation in (7) with an exclusive-OR, ensuring that S_i^B is unique for each S_i:

$$S_i^B = \overline{S_i} \oplus \begin{cases} 0110_2, & A_i = 1 \text{ and } S_{i,3} = 1 \\ 0000_2, & A_i = 0 \text{ or } S_{i,3} = 0 \end{cases}$$

A second run of the 16-bit simulation described in Section II-B, followed by an exhaustive simulation of the checker

TABLE I
POST-ROUTE DELAY (ns)

Adder	Checker	Width (bits) 32	64	128	256	512
Ladner-Fischer	None	0.59	0.68	0.77	0.89	1.01
DesignWare	None	0.55	0.66	0.74	0.86	1.00
Binary/BCD	None	0.99	1.12	1.20	1.33	1.47
	DMR	1.26	1.41	1.57	1.75	2.01
	LRC	1.43	1.55	1.76	1.93	2.16

TABLE II
POST-ROUTE CELL AREA (μm^2 × 1000)

Adder	Checker	Width (bits) 32	64	128	256	512
Ladner-Fischer	None	0.92	1.93	4.16	8.30	17.85
DesignWare	None	1.07	2.38	4.61	9.29	20.02
Binary/BCD	None	1.49	3.03	5.81	11.9	24.06
	DMR	3.16	6.25	12.09	24.36	49.95
	LRC	2.93	6.04	11.65	23.84	48.77

alone at a width of 8 bits, shows no remaining false negatives and no false positives.

B. Performance Evaluation

Tables I and II show the post-route delay (in nanoseconds) and cell area (in thousands of square microns) of each adder; Figures 1 and 2 show the delay and area of each adder relative to a non-redundant binary/BCD adder of the same width. Solid lines indicate actual results, while dashed lines indicate predicted results from [2]. Filled markers indicate binary/BCD adders, while outlined markers indicate binary-only adders.

Adding DMR error checking to a mixed binary/BCD adder increases cell area by 105%–113% and delay by 26%–37%, while adding LRC error checking increases cell area by 98%–103% and delay by 39%–47%. Replacing a DMR checker with an LRC checker saves 5%–15% of the original (non-redundant) adder's area, at the cost of an additional 10%–17% increase in delay. For non-redundant binary adders, the minimum-depth Ladner-Fischer adder uses 10%–19% less area than the fast `pparch` implementation of the DesignWare adder, at the cost of only 1%–9% greater delay.

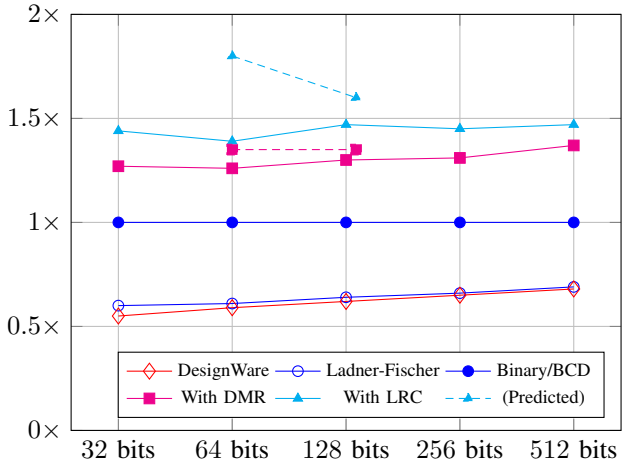

Fig. 1. Delay results relative to a non-redundant binary/BCD adder.

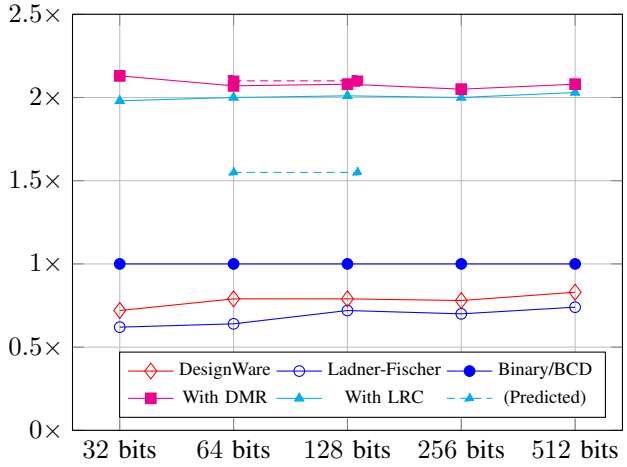

Fig. 2. Area results relative to a non-redundant binary/BCD adder.

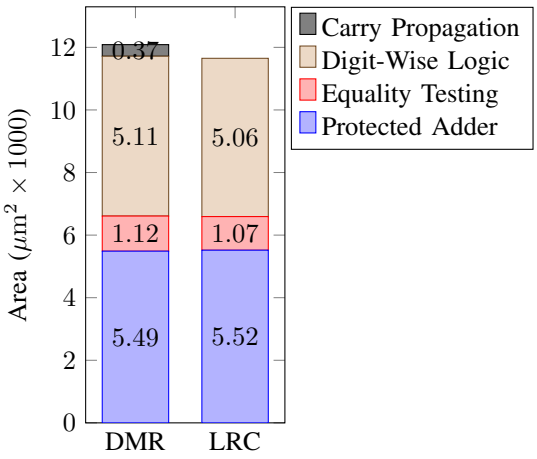

Fig. 3. Area distribution for 128-bit redundant adders.

Figure 3 shows the distribution of cell area in the DMR and LRC versions of a 128-bit adder. The area required for digit-by-digit computations (e.g. SV, CV, Y^D, S^B) is almost identical between the two adders, suggesting that most of the area savings may be due to the lack of a carry tree in the LRC version.

IV. DISCUSSION

The area and delay ratios between non-redundant, DMR, and modified LRC adders are compared against the predicted ratios of [2, Tbl. II], rather than converting the predictions to absolute terms using the standard cell library's FO4 delay and NAND2 area. The predicted area and delay penalties for DMR are 110% and 35% respectively, which fall within the range of observed values. However, the predicted area and delay penalties for a modified LRC error checker are 55% and 60%–80% respectively, which Figures 1 and 2 show to be

pessimistic (in the case of delay) and optimistic (in the case of area) by almost a factor of two, bringing the modified LRC adder's performance closer to that of a DMR adder.

Figure 3 suggests that digit-wise addition and decimal correction dominate the area requirements for mixed binary/decimal addition, limiting the total area that can be saved by eliminating carry propagation. Furthermore, the digit corrections described in Section I-B are roughly as area-intensive as decimal addition itself. These two factors combine to make an LRC design less practical in decimal or mixed applications than in binary applications.

References

[1] M. B. Sullivan and E. E. Swartzlander, Jr., "Long residue checking for adders," in *Proc. IEEE 23rd Int. Conf. Application-Specific Systems, Architectures and Processors (ASAP)*, Jul. 2012, pp. 177–180. DOI: 10.1109/ASAP.2012.31.

[2] Á. Vázquez and E. Antelo, "A sum error detection scheme for decimal arithmetic," in *Proc. IEEE 24th Symp. Computer Arithmetic (ARITH)*, Jul. 2017. DOI: 10.1109/arith.2017.34.

[3] R. Goldman, K. Bartleson, T. Wood, K. Kranen, V. Melikyan, and E. Babayan, "32/28nm Educational Design Kit: Capabilities, deployment and future," in *Proc. IEEE Asia Pacific Conf. Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*, Dec. 2013. DOI: 10.1109/PrimeAsia.2013.6731222.

[4] "IEEE standard Verilog hardware description language," IEEE Std 1364-2001, Sep. 2001. DOI: 10.1109/IEEESTD.2001.93352.

[5] D. P. Agrawal, "Fast B.C.D./binary adder/subtractor," *Electron. Lett.*, vol. 10, no. 8, Apr. 1974. DOI: 10.1049/el:19740093.

[6] V. Naganathan, "A comparative analysis of parallel prefix adders in 32nm and 45nm static CMOS technology," M.S. report, University of Texas at Austin, May 2015. DOI: 10.15781/T2TP6P.

[7] R. E. Ladner and M. J. Fischer, "Parallel prefix computation," *JACM*, vol. 27, no. 4, pp. 831–838, Oct. 1980. DOI: 10.1145/322217.322232.

[8] "DW01_addsub adder/subtractor," Synopsys, v. K-2015.06-SP4, Jun. 2015. [Online]. Available: https://www.synopsys.com/dw/ipdir.php?c=DW01_addsub.