Hybrid Residue Generators for Increased Efficiency

Michael B. Sullivan
School of Electrical and
Computer Engineering
University of Texas at Austin
Austin, Texas 78712
Email: mbsullivan@mail.utexas.edu

Earl E. Swartzlander, Jr.
School of Electrical and
Computer Engineering
University of Texas at Austin
Austin, Texas 78712
Email: eswartzla@aol.com

***Abstract*—In order for residue checking to effectively protect computer arithmetic, designers must be able to efficiently compute the residues of the input and output signals of functional units. Low-cost, single-cycle residue generators can be readily formed out of two's complement adders in two ways, which have area and delay tradeoffs. A residue generator using adder-incrementers for end-around-carry adders is small but slow, and a design using carry-select adders is fast, but large. It is shown that a hybrid combination of both approaches is more efficient than either.**

***Index Terms*—Residue checking, end-around-carry adder, low-cost residue generation, hybrid residue generator.**

I. Introduction

Rising soft-error rates in combinational logic make arithmetic error protection increasingly important [1], [2]. Residue checking is a popular error coding technique for protecting the arithmetic datapath in a microprocessor [3], [4]. In order for residue checking to protect against computer errors, designers must be able to efficiently compute the residues of the input and output signals of functional units.

Low-cost, single-cycle residue generators can be formed by a tree of modular end-around-carry (EAC) adders [5]. While such adders are common in computers which use a one's complement or residue-based representation of numbers, efficient EAC adders may not be well known or available to designers that use two's complement arithmetic system-wide. End-around-carry adders can be readily formed out of commodity two's complement adders in multiple ways, which have area and delay tradeoffs.

This study shows that a low-cost residue generator implemented out of two architecturally distinct EAC adders is more area efficient than any implementation consisting of a single constituent adder. Such a residue generator, with multiple types of EAC adders, is referred to as a *hybrid residue generator*, and is the focus of this work.

Section II describes the hybrid residue generator in detail and illustrates when and why it has the potential to increase residue generation efficiency. The empirical performance of the hybrid residue generator is examined in Section III for a range of system sizes and speeds.

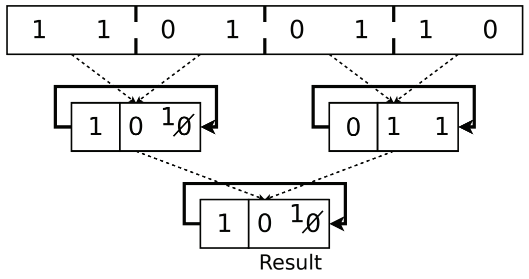
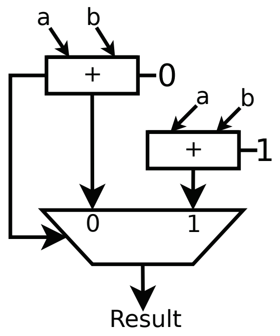


Fig. 1. The low cost residue of an unsigned number, using a tree of end-around-carry adders. $1 \equiv |214|_3$ is shown.

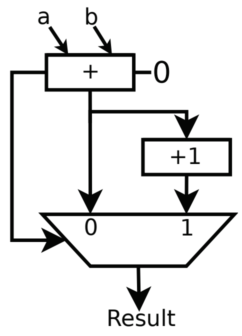
A. Low-Cost Residue Codes

Before describing the concept of the hybrid residue generator, some basic properties and definitions of low-cost residue codes are reviewed. Often, residue checking relies on a restricted class of residue codes, in the form $[2^a - 1; a \in \mathbb{N}]$, in order to reduce the circuit complexity of generator circuits [5]. This class of residue codes is commonly referred to as the *low-cost residue codes*, because of its relative simplicity of residue generation. The low-cost residue of an n-bit number X, $|X|_{2^a\text{-}1}$, can be generated by the addition of non-overlapping a-bit slices of X under modulo-(2^a-1) arithmetic. This procedure is often implemented in a single cycle as a tree of EAC adders. Figure 1 demonstrates how such a tree of EAC adders can generate a low-cost residue code.

Efficient low-cost residue generation requires the use of modular or EAC adders. For a system which uses two's complement arithmetic system-wide, this may mean designing these structures out of binary adders. Simply connecting the carry-out to the carry-in of a binary adder turns the adder into an asynchronous sequential circuit, which can suffer from long and unpredictable race conditions [6]. However, EAC adders can be formed out of binary adders in other ways, which vary in their area-delay tradeoffs. Two simple, single-cycle implementations of EAC addition using binary adders are shown in Figure 2. The two designs exhibit different area-delay characteristics: the carry select EAC adder (Figure 2a) is fast but has replicated logic which adds to its area and power overheads, and the adder-incrementer (Figure 2b) is small but has a relatively long critical path.



(a) Carry-Select (Fast)



(b) Add-Increment (Small)

Fig. 2. Two simple designs for stable single-cycle EAC adders which can utilize commodity two's complement adders.

II. Hybrid Residue Generators

The hybrid residue generator (HRG) is a low-cost residue generator that can combine multiple architecturally distinct EAC adders in order to increase the total implementation efficiency. Figure 3 shows an HRG which uses small EAC adders for all but the last level; such an HRG would be appropriate for a design subspace with very efficient adder-incrementer residue generators that cannot completely satisfy the timing constraints. This study investigates the potential benefits of dual adder HRGs as a method of architectural optimization fit for the synthesis of low-cost residue generators as a part of standard ASIC flow.

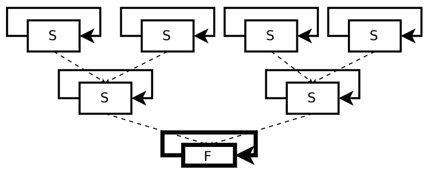


Fig. 3. An HRG with one fast adder (F). Small adders (S) are used to increase efficiency, while the fast adder preserves timing.

VLSI designs suffer from decreasing marginal area efficiency with diminishing delays as the fundamental speed limit of a design is approached. This asymptotic behavior is unavoidable, but effectively optimized circuits can often result in more efficient behavior for a given time budget [7]. Figure 4 compares the area-delay behavior of residue generators made out of the two EAC adders chosen for this study. All synthesized designs are enumerated by a script under finely varying timing constraints. The small residue generator requires significantly less area, but asymptotes to a speed limit sooner. Because of its slower design, there are time targets which only the faster generator can reach. Also, there are delays which the small adder can achieve, but only with enough cost that the fast adder becomes the more efficient design.

As Figure 4 shows, the small and fast residue generators are suitable for different target delays. The hybrid residue generator combines the strengths of the small and fast residue generators—a properly formed HRG is never less efficient than either constituent generator, and can outperform either for

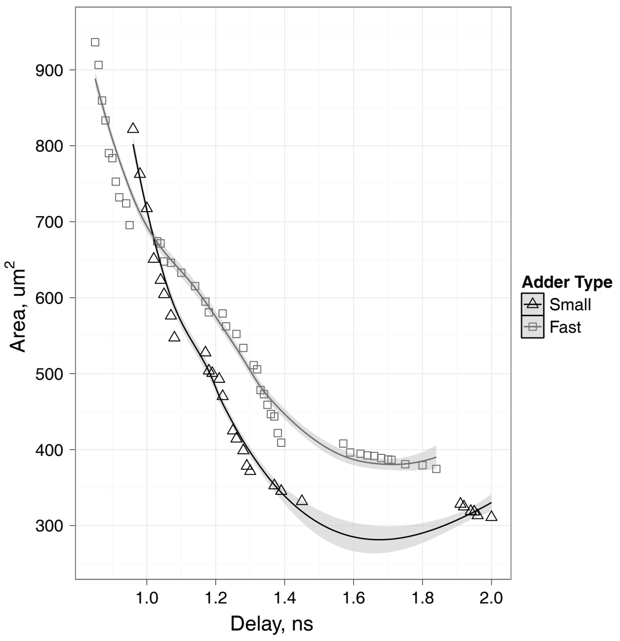


Fig. 4. The area required to achieve a given max-delay for a carry-select (fast) 32-bit residue generator ($a = 4$), and an adder-incrementer (small) one. A LOESS curve is shown with a 95% confidence interval to better visualize trends in the data.

some delays. Formation of the proper hybrid residue generator for a given time budget entails a search of the HRG space to select the most efficient design. In general, the best HRG is similar to the small generator at slower delays, and similar to the fast one under stricter time limitations.

III. Hybrid Residue Generator Evaluation

This study analyzes hybrid residue generators with two types of end-around-carry-adders. All empirical findings were carried out using the adder configurations in Figure 2, though the methodology may be applied to other types of EAC adders.

A. *Experimental Methodology*

The Synopsys DesignWare IP parallel prefix adder is used as the basic building block in the EAC adders to provide flexibility across different word lengths and speeds. Parallel prefix adders can efficiently span the range of area-delay trade-offs, allowing for design space exploration across a large range of delays without changing the underlying binary adder [8].

Synthesis is performed using the Synopsys toolchain, targeting the 45nm Nangate Open Cell Library [9], [10]. All designs are compiled using the Synopsys Design Compiler with high mapping effort and optimization options consistent with an area-optimized implementation.

The analyses in this study examine the behavior of residue generation across a spectrum of word sizes, modulo widths, and delay budgets. The population of synthesized circuits is studied by repeatedly performing synthesis with finely varying delay constraints. Analyses use the Pareto frontier of the generated solution set—only designs with the lowest area and least delay which satisfy timing are retained. By eliminating

TABLE I
MEAN AREA-TIME AND AREA-TIME-SQUARED SAVINGS FROM THE HRG.

16-bit				
	vs. Fast		vs. Slow	
Mod Width (a)	AT	AT^2	AT	AT^2
2	-0.151	-0.157	-0.096	-0.142
4	-0.098	-0.226	-0.034	-0.010

32-bit				
	vs. Fast		vs. Slow	
Mod Width (a)	AT	AT^2	AT	AT^2
2	-0.098	-0.143	-0.097	-0.024
4	-0.091	-0.213	-0.038	-0.011
8	-0.153	-0.239	-0.012	-0.017

64-bit				
	vs. Fast		vs. Slow	
Mod Width (a)	AT	AT^2	AT	AT^2
2	-0.113	-0.117	-0.126	-0.110
4	-0.105	-0.209	-0.048	-0.033
8	-0.154	-0.179	-0.005	-0.021
16	-0.216	-0.338	-0.002	-0.009

128-bit				
	vs. Fast		vs. Small	
Mod Width (a)	AT	AT^2	AT	AT^2
2	-0.065	-0.035	-0.247	-0.245
4	-0.127	-0.195	-0.046	-0.025
8	-0.173	-0.214	-0.023	-0.034
16	-0.227	-0.360	-0.003	-0.002
32	-0.283	-0.388	-0.008	-0.017

plateaus of similar, locally sub-optimal designs, the Pareto frontier spreads analyses over the full, diverse design space.

The most effective HRG for a given time budget is the most efficient implementation. Figure 5 shows the algorithm used for forming the HRG frontier; this process was carried out at all time budgets. The search space of all HRGs is heuristically pruned using the knowledge that the number of EAC adders per-level decreases geometrically in a residue generator. However, the critical path contribution of each level stays the same. Therefore, the globally most efficient HRG is likely to be made of small adders towards the input nodes and fast adders towards the output and levels with mixed adder designs need not be considered. This results in $O(L)$ possible HRGs which must be searched at every time step. The fitness of HRGs is evaluated using Pareto optimality over area and time; as such, the results of the study are not skewed towards any specific metric of efficiency.

```
Input: Tree with FAST EAC Adders
Output: Hybrid Residue Generator
foreach Time Step do
    foreach Level $L$ from the Input to Output do
        Replace all FAST Adders with SMALL Ones;
        Synthesize;
    end
end
Retain all Pareto efficient designs;
```

Fig. 5. The algorithm used for HRG selection.

B. Results

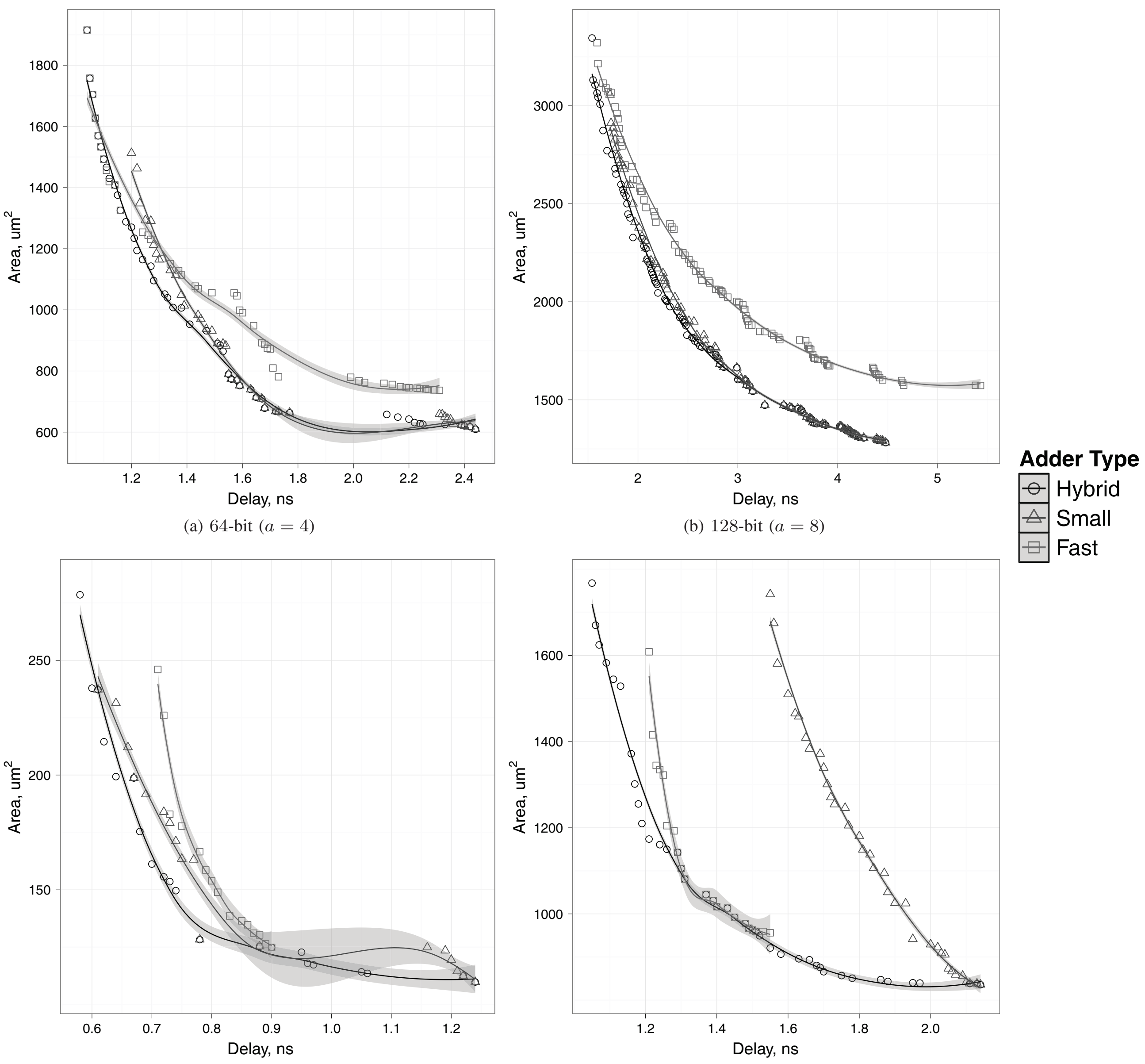
The area-delay efficiency of an HRG relative to the fast and small residue generators is analyzed using the combined area-time (AT) and area-time-squared (AT^2) metrics. The mean AT and AT^2 savings from the hybrid residue generator are shown in Table I. Comparisons are made using the HRG designs with achievable delays for each respective residue generator. The HRG increases the efficiency of residue generation at all timescales.

In general, the largest efficiency gains due to hybridization are in the low delay design subspace where the (small) adder-incrementer cannot form the entire residue generator. This range of delays is particularly important for designers, as it is the region where performance-conscious designs are likely to fall. Figure 6a shows the relative area-delay behavior of the fast, small, and hybrid 64-bit residue generators ($a = 4$). The HRG shows improvement over either uniform design in the region where the small residue generator asymptotes to its maximum speed. As the time budget continues to decrease, the HRG degenerates into a fast residue generator in order to scale to higher speeds.

In Table I, the performance-centric efficiency gains due to hybridization can be seen from the efficiency of the HRG relative to the carry-select adder. The HRG improves AT efficiency considerably, from 10 to 20% on average. Weighted efficiency (AT^2) is improved further. This indicates that the HRG can improve the efficiency of performance oriented designs without violating timing constraints.

At large word sizes, the cost of residue generation dominates the overhead of residue checking. Another clear result is that the hybrid residue generator can scale to large word sizes—the 128-bit HRG shows efficiency improvements akin to those at smaller word sizes. Although the HRG will not change the asymptotically linear area of a low-cost residue tree, its ability to improve efficiency at large word sizes may be especially important.

While most results are consistent with the hypothesis, there are some anomalies. HRG efficiency relative to small generators with large moduli shows little increase; in particular, there does not seem to be much benefit from hybridization relative to the small design when $a \geq 8$. Although further analysis remains to be done, the explanation seems to be that the fast adder is inefficient at large moduli, especially at slower speeds. The small residue generator ($a \geq 8$) is close to the most efficient design over all of its reachable delays. Figure 6b shows the 128-bit residue generators with $a = 8$, where the fast generator is much less efficient than the small one over almost all delays. Accordingly, the HRG is only able to make modest efficiency improvements over the small design. Even so, the HRG provides notable improvements over either constituent approach: it gracefully scales to high speeds, offering a single solution which combines the best aspects of both small and fast generators.



(a) 64-bit ($a = 4$)

(b) 128-bit ($a = 8$)

(c) 16-bit ($a = 2$)

(d) 128-bit ($a = 2$)

Fig. 6. Hybrid Residue Generator behavior across different word lengths and moduli.

Another notable anomaly from Table I is the lower reduction in AT and AT^2 for the HRG ($a = 2$) relative to the fast design. The 128-bit residue generator, in particular, shows little improvement due to hybridization. Lessened efficiency improvement at $a = 2$ might be problematic, as this is a common design choice among residue checking implementations [11], [12]. However, close inspection shows the AT and AT^2 metrics for $a = 2$ to be misleading. Figure 6c and Figure 6d give the behavior of the 16-bit and 128-bit residue generators ($a = 2$), respectively. Because of their deep adder trees with many levels, both the fast and small residue generators asymptote quickly. The HRG shows only modest efficiency improvements over the fast generator before it asymptotes, which is reflected in the tabulated results. However, with $a = 2$, the HRG is also able to scale to higher speeds than either uniform generator, lending additional utility to the designer. At 16-bits, an HRG with one small level is able to decrease the critical delay of the faster design by 4%, and at 128-bits, an HRG with three small levels can operate 13% faster than the fastest uniform design. The 32-bit and 64-bit residue generators show similar results with a modulo width of 2.

As expected, Pareto efficient HRGs generally resemble the small residue generator at slower speeds, and the fast generator at higher speeds. For time targets in between these extremes, most HRG frontiers show gradual, predictable behavior. Figure 7 shows a 16-bit HRG ($a = 4$), with the number

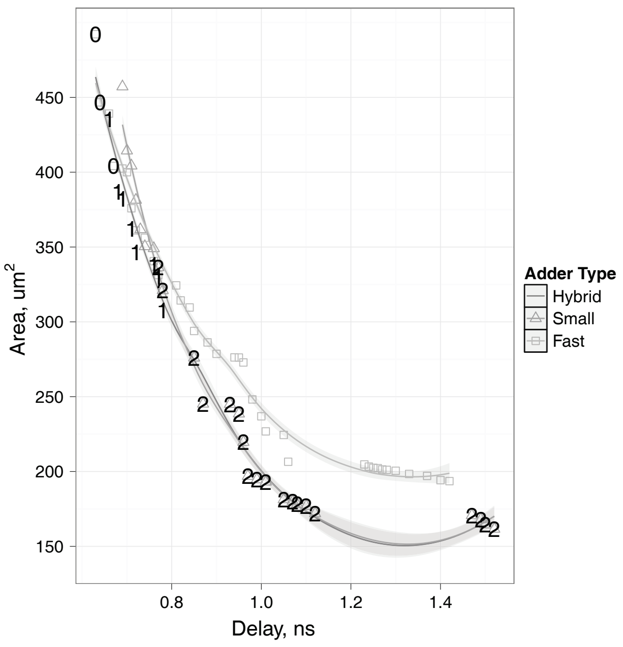


Fig. 7. A 16-bit HRG ($a = 4$) with the composition of the HRG labeled.

of small levels at each design point labeled. The EAC tree for this design has two levels, such that any HRG labeled "2" is the same as the uniform small tree, and "0" is the same as the uniform fast tree. Any HRG labeled "1" has small adders at the input and a fast one at the output. The gradual increase in fast EAC levels is similar across all parts of the HRG design space, although some subspaces (such as where $a = 2$) show greater variability and have fewer uniform HRGs.

IV. Conclusion

Because of its nature, the hybrid residue generator will degenerate to a uniform EAC adder tree if that is the most efficient choice available. As such, the intelligent application of hybrid residue generation should never decrease system efficiency. In addition, results show that the HRG can combine two different EAC adders to create a design that is more efficient than any uniform generator. Also, some HRGs can increase the speed of residue generation, operating at a faster critical frequency than either the fast or small design.

There exist alternative low-cost residue checking implementations which may give increased efficiency. An example of such an approach is the use of carry-save adder networks for residue generation through multi-operand modular addition [13]. This study makes no direct comparison to alternative residue checkers, but shows that the hybrid residue generator can improve EAC adder tree efficiency with little design effort.

The design of a low-cost residue generator can encompass more than the two simple EAC adders shown in Figure 2. Residue checking hardware is a relatively small component of the entire computer system; accordingly, this study focuses on EAC adders which minimize design effort. Hybrid residue generators are useful in this context, because they provide a mechanism to increase the efficiency of residue checking, while keeping design overhead low. More complex modular adder designs exist, which could alternatively be used for residue generation [14], [15]. These designs are orthogonal to the concept of an HRG and can be used with hybrid residue generation if they are available.

Using the algorithm from Figure 5 to generate an HRG requires $O(L)$ repeated syntheses. More aggressive heuristic optimization could possibly reduce the search space further while retaining high accuracy, but this is left for future work.

This study is an initial attempt to analyze the feasibility and potential benefits of hybrid residue generators. The results are encouraging and show that simple EAC adders can be combined to create a generator which is fast, efficient, and minimizes designer effort. Future work remains to better understand the underlying mechanisms and limitations of the HRG, and to extend it beyond the dual adder configuration considered in this study.

Acknowledgments

Michael Sullivan's research was supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate Fellowship (NDSEG) Program. Earl Swartzlander is supported in part by a grant from AMD, Inc.

References

[1] T. Karnik and P. Hazucha, "Characterization of soft errors caused by single event upsets in CMOS processes," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 2, pp. 128–143, 2004.

[2] G. Saggese, A. Vetteth, Z. Kalbarczyk, and I. Ravishankar, "Microprocessor sensitivity to failures: control vs. execution and combinational vs. sequential logic," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2005, pp. 760–769.

[3] T. Rao, "Error-checking logic for arithmetic-type operations of a processor," *IEEE Transactions on Computers*, vol. C-17, pp. 845–849, 1968.

[4] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, 2nd ed. Oxford University Press, 2010.

[5] A. Avizienis, "Arithmetic error codes: Cost and effectiveness studies for application in digital system design," *IEEE Transactions on Computers*, vol. C-20, pp. 1322–1331, 1971.

[6] J. Shedletsky, "Comment on the sequential and indeterminate behavior of an end-around-carry adder," *IEEE Transactions on Computers*, vol. C-26, pp. 271–272, 1977.

[7] D. M. Markovic, "A power/area optimal approach to VLSI signal processing," Ph.D. dissertation, EECS Department, University of California, Berkeley, May 2006.

[8] S. Knowles, "A family of adders," in *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, 2001, pp. 277–281.

[9] Synopsys Inc., "Design Compiler," 2010.

[10] Nangate, "Open Cell Library v1.3," 2009.

[11] I. Sayers and D. Kinniment, "Low-cost residue codes and their application to self-checking VLSI systems," *IEE Proceedings-E in Computers and Digital Techniques*, vol. 132, pp. 197–202, 1985.

[12] U. Sparmann and S. Reddy, "On the effectiveness of residue code checking for parallel two's complement multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, pp. 227–239, 1996.

[13] S. Piestrak, "Design of residue generators and multioperand modular adders using carry-save adders," *IEEE Transactions on Computers*, vol. 43, pp. 68–77, 1994.

[14] C. Efstathiou, D. Nikolos, and J. Kalamatianos, "Area-time efficient modulo $2^n - 1$ adder design," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 41, pp. 463–467, 1994.

[15] R. Zimmermann, "Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication," in *Proceedings of the 14th IEEE Symposium on Computer Arithmetic*, 1999, pp. 158–167.