Long Residue Checking for Adders

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

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

***Abstract*—As system sizes grow and devices become more sensitive to faults, adder protection may be necessary to achieve system error-rate bounds. This study investigates a novel fault detection scheme for fast adders, *long residue checking* (LRC), which has substantive advantages over all previous separable approaches. Long residues are found to provide a ~10% reduction in complexity and ~25% reduction in power relative to the next most efficient error detector, while remaining modular and easy to implement.**

***Index Terms*—Adder, long residue checker (LRC), residue checking, lazy checker, standard cell synthesis, self-testing and self-checking circuitry.**

I. Introduction

Adders are of fundamental importance to computer systems, and as such have been studied extensively. Addition is a highly utilized instruction and is used for data manipulation, memory addressing, and control flow—this means that an error in addition can manifest in many ways, ranging from silent data corruption to catastrophic system failure. The wide range of maladies that can result from adder errors makes the protection of addition important across many problem domains, including scientific computing and system software.

While detected errors may be economically corrected at the architectural level through hardware sparing and re-execution, dynamic error detection in adders remains expensive. In part, the high relative cost of error detection in adders comes from the efficiency of adder designs. An error detection mechanism is often evaluated relative to the unit it protects, putting adder protection under stringent area and power constraints. Also, any arithmetic error code must be closed under addition, which limits the separable codes to those in the residue class. While residue codes are well suited elsewhere in the computer system, they are *not* typically ideal for protecting a single adder against error. This study investigates a *modified* residue checker that is able to provide strong, low latency error detection for a single fast adder at less cost than any other separable design.

A. Residue Checking

Before describing the main contribution of this paper, some basic properties and definitions of low-cost residue codes are reviewed. Addition can be checked by testing the equality of Equation 1, where $|N|_A = N \bmod A$ and $+$ denotes modular addition. If both sides of Equation 1 are equal, it is likely that no error has occurred. If they are not equal, then some error *has* occurred.

$$|a \oplus b|_A \stackrel{?}{=} \big||a|_A + |b|_A\big|_A \tag{1}$$

Often, residue checking relies on a restricted class of residues in the form $A = [2^a - 1; a \in \mathbb{N}]$, in order to simplify operations.The error coverage of a residue code depends on the width, a, of its checking modulus. Given the current trend towards reliability-constrained devices, strong error protection will become increasingly valuable. This paper demonstrates that a modified residue checker can enable low-cost error detection with large checking moduli. In fact, it is shown that the modified checker with the largest possible residue width ($a = n$) is the least complex, most power efficient, has the highest error coverage, and the lowest latency. Such a checker can detect a fault in any single component, providing complete coverage against single event upsets (SEUs). Furthermore, experiments demonstrate that this *long residue checker* has significant efficiency advantages over all other techniques for separable error detection in fast adders.

II. Long Residue Codes

Long residue codes are based upon a residue checking algorithm which operates on the modular carry-save representation of the main adder result subtracted from its inputs. If $\big||a|_A + |b|_A\big|_A = |c|_A$ (the traditional residue checking equality), then $\big||a|_A + |b|_A - |c|_A\big|_A = 0$. This equality holds if and only if the sum and carry terms cancel. Such cancellation is easily detected without carry-propagation, such that the carry-propagate adder from may be eliminated from residue generation.

Due to the fact that the modified checking algorithm has no carry dependence, its efficiency does not strongly

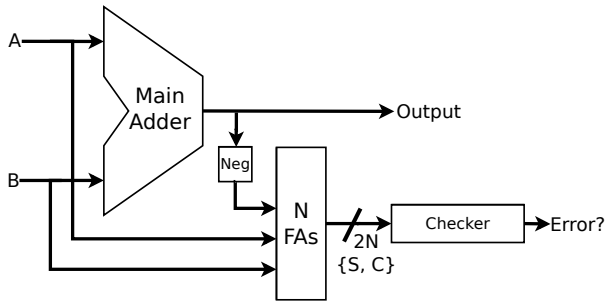


Fig. 1. The long residue checker.

depend on the checking modulus. Further simplifications to the system are possible when the checking modulus is equal to the word size ($a = n$), as shown in Figure 1. This *long residue checker* (LRC) eliminates the need for a general carry-save multi-operand modular adder (CS-MOMA), and instead uses a single carry-save adder followed by a full-length checking tree. Later, in Section IV, it is shown that the efficiency gained by eliminating the CS-MOMA outweighs any added checking costs, making the long residue checker the most efficient residue checking implementation for adders.

A. Two's Complement Numbers and Subtraction

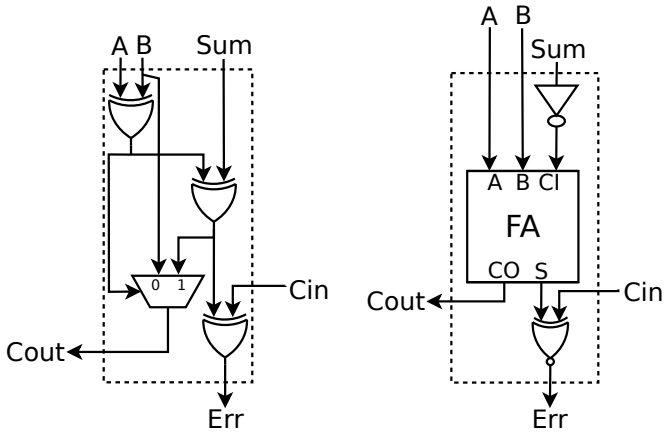
The LRC checker is easily adapted to work for two's complement numbers and to support subtraction. For two's complement arithmetic, the final check slice carry-out is checked against the complemented carry-out of the main adder. In order to support subtraction, the subtrahend must be inverted (not shown), and the main adder carry-in propagated into the first checking stage.

III. Relationship to Existing Work

An error code is *separable* if it can be split into a data and check portion such that the checking procedure does not change the main datapath. Separability is highly desirable for arithmetic error detection, because it increases the modularity of a design and allows for error detection to be implemented off of the critical arithmetic path. Residue checking is the only separable arithmetic error code for adders, apart from full duplication [1]. This limits the number of competing designs; the competing mechanisms used in this study are described below.

Coarse duplication (also known as dual modular redundancy [DMR]) is simple, intuitive, strong, separable, general, and may be applied to addition. The area and power costs of duplication, however, generally keep it from being competitive with coding approaches.

Figure 2(a) shows one bit-slice of the lazy error checker proposed by Yilmaz *et al.* [2]. Each slice checks one output bit using a modified full adder and an XOR gate; the error signal from each bit is ORed together to determine if an error has occurred. The checker has



(a) Lazy Adder Checker (b) Long Residue Checker

Fig. 2. A 1-bit slice of the lazy adder checker [2] and of the long residue error checker proposed in this study.

no carry dependence, such that the checker delay (apart from the OR tree) does not directly depend on the word width. Careful inspection shows that the functionality of the long residue and lazy checkers are essentially equivalent[1]; each is a bit-sliced design, and the two have a similar error coverage. Figure 2(b) shows the equivalent bit-slice implementation of long residue checking.

While long residue checking is functionally similar to the lazy checker, the LRC has the potential to significantly reduce error detection overheads using standard cell synthesis. An important advantage of the LRC is that it uses a full adder as a fundamental unit. There exist efficient full adder cells [3] in many standard cell libraries, which can increase the efficiency of the LRC checker without resorting to custom cell design.

To demonstrate the efficiency advantages of the LRC, this study makes use of the mirror adder cell found in the Nangate 45nm standard cell library [4]. Isolated analysis of both the checkers in Figure 2 shows that an LRC cell consumes 10.73% less area and 19.66% less power than its lazy-checker counterpart. The following section evaluates all separable adder protection approaches, and finds that the full LRC has efficiency advantages equal to or greater than these initial estimates.

IV. Long Residue Evaluation

To evaluate the LRC, long residue checking is first shown to be the most efficient residue checking implementation for addition. Having established this, the LRC is evaluated relative to lazy checking and DMR. Dual modular redundancy is evaluated using two configurations: one with a second fast parallel prefix adder, and one with a serial prefix checking adder. Unless mentioned otherwise, each mechanism is given an extra cycle of detection latency for completion, to mimic

[1]With a different carry interface; the two designs cannot be mixed.

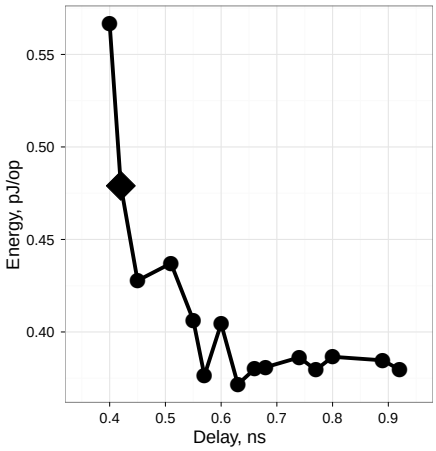


Fig. 3. Pareto-efficient 16-bit adder designs. The highlighted adder minimizes the ED^2 metric and is used as a baseline.

operation in a pipelined design. Pipeline registers are not inserted or evaluated, due to their similar impact on every design and because it is difficult to pipeline the DMR designs for every considered configuration. Every error detection mechanism is examined across a range of adder widths, to study how it scales with input size.

A. Experimental Methodology

RTL-level design space exploration is used to examine the area and energy properties of each error detection mechanism. The Synopsys toolchain is used for synthesis, targeting the 45nm Nangate Open Cell Library [5], [4]. All circuits are compiled with high mapping effort and options consistent with an area-optimized implementation. Dual-rail encoded checkers are used to create totally self-testing designs. In-house circuits are used for residue checking; the baseline adders are taken from a high-performance cell-based arithmetic unit library [6]. Gate-level (pre-layout) area and power estimates are used for all analyses; dynamic power is determined using a pair of random test vectors every cycle. Energy calculations assume that the latency of the main adder dictates the clock frequency. It is assumed that all circuitry is driven by (and drives) pipeline latches.

All LRC overheads are given relative to an efficient 2's complement adder design. A Pareto-optimal (over area and energy) post-synthesis design which minimizes the ED^2 metric [7] is chosen at each word length through a search of the design space. Figure 3 shows a simplified Pareto frontier of possible 16-bit adder designs, and highlights the reference adder used in this study. Table I gives the area, power, and delay of each selected design. Most results are normalized relative to the baseline adder or the LRC; absolute area and power consumption estimates can be derived accordingly.

B. Results

Figure 4 shows the area and power consumption of traditional and modified residue checking for a 32-bit adder across different residue widths. The LRC is denoted by a rhombus. The CS-MOMA for modified residue checking is implemented using a tree carry-save adders with an end-around-carry at each level. Any such CS-MOMA requires a constant number of full adders regardless of the modulus width. Wiring complexity increases drastically at low residue widths, however, making the long residue configuration ($a = n$) more area and power efficient than any alternative. Increasing CS-MOMA depths conspire to make residue checking at short residue widths ($a \ll n$) a long latency operation. As such, all designs in Figure 4 use a fixed detection latency of three cycles, despite the fact that long residue checking easily completes earlier.

Table II gives the area and power of the long residue checker relative to the baseline adder. The relative power overheads of the LRC are slightly higher than its area costs. This trend is consistent with prior work [2]; it is likely that the main adder exacerbates the difference, as it is selected to be energy efficient. Due to the continual utilization of the circuit, dynamic power dominates.

Table III shows the additional area and power overheads incurred by competing error detection mechanisms (normalized relative to the area and power of the LRC). Long residue checking has significant efficiency gains relative to all other designs. The lazy checker consumes about 10% more area and 25% more power than the LRC. The power advantages of the LRC slightly outweigh the savings of a single LRC cell relative to a lazy checker cell, due to reduced switching activity at its error checking tree. The LRC is the lowest latency design of those considered. The longer latency of lazy checking causes a significant loss in its relative efficiency at 16-bits—in order to satisfy timing, the lazy checker must

TABLE I
BASELINE ADDER PROPERTIES.

Adder Width	Delay (ns)	Area (μm^2)	Power (mW)
16	0.42	381.35	0.657
32	0.55	848.79	1.133
64	0.67	1546.06	1.616
128	0.7	3247.85	3.161

TABLE II
THE OVERHEAD OF LONG RESIDUE CHECKING.

Adder Width	% Area Overhead	% Energy Overhead
16	38	69
32	33	70
64	36	84
128	34	86

TABLE III
ADDITIONAL OVERHEADS RELATIVE TO THE LRC.

Lazy Checker		
Word Width	Area (%)	Power (%)
16	36	48
32	9	24
64	10	23
48	10	25
Duplication (Serial Prefix)		
Word Width	Area (%)	Power (%)
16	65	28
32	92	37
64	97	41
48	100	38
Duplication (Sklansky)		
Word Width	Area (%)	Power (%)
16	193	111
32	205	98
64	186	59
48	188	48

increase the size and power consumption of its cells.

Duplication consumes significantly more area and power than the LRC or lazy checker. Apart from the reduced efficiency of lazy checking at 16-bits, Table III shows a clear ordinal rank among the four detectors.

V. DISCUSSION

The majority of efficiency gains from the long residue checker come from its ability to leverage efficient full adders while using only standard library cells. Custom cell design may be used to improve the efficiency of the lazy checker. One of the main strengths of the LRC is that it is able to avoid such custom design while providing superior implementation efficiency.

Some standard cells (such as D flip-flops) offer both an inverted and non-inverted output with little additional complexity or power usage. As such, a pipeline register with dual outputs may be used after the main adder to avoid an explicit inversion of the sum signal in the LRC.

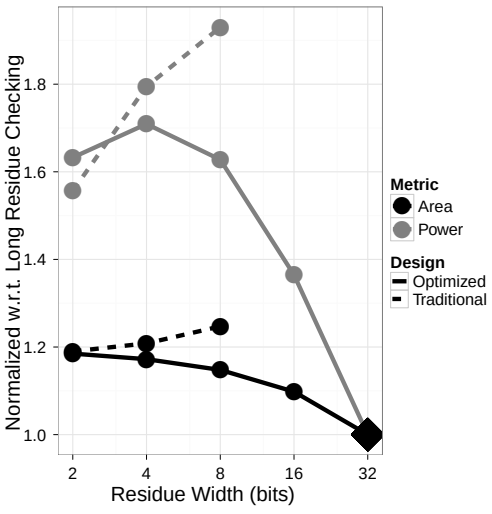


Fig. 4. The area and power of traditional and modified residue checking, normalized w.r.t. long residue checking.

Preliminary experiments show that this optimization may give ∼6% area and ∼12–14% power savings over the long residue checker used in this study.

Careful inspection shows that the relative overheads presented for the LRC (Table II) are higher than those claimed in prior work for lazy checking [2], despite the fact that this study finds long residue checking to be more efficient. It is likely that this difference is mainly due to differences in the baseline adders. However, some methodological decisions could also have an impact. All error protection designs in this study also employ a dual-rail encoded checker (unlike prior work). This is consistent with totally self-testing circuit implementations, and is intended to be correctly diagnose permanent checker errors. Experiments indicate that a single checker would reduce the area and power of the LRC by ∼8% and ∼13–15%, respectively.

VI. CONCLUSION

The long residue checker is a novel error detection scheme for fast adders which is based on a straightforward modification of residue checking. The LRC provides levels of SEU coverage comparable to duplication, while drastically reducing the area and power overheads of error detection. The LRC is shown to be the most efficient separable design for protecting fast adders using standard-cell synthesis. The error coverage, efficiency, low detection latency, modularity, and separability of the LRC make it a valuable error detection technique for fast adders, and motivate its use towards the reliability of future computer systems.

ACKNOWLEDGMENTS

Michael Sullivan's research was supported by the Temple Foundation. Earl Swartzlander is supported in part by a grant from AMD, Inc.

REFERENCES

[1] W. W. Peterson, "On checking an adder," *IBM Journal of Research and Development*, pp. 166–168, 1958.
[2] M. Yilmaz, A. Meixner, S. Ozev, and D. Sorin, "Lazy error detection for microprocessor functional units," in *IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems*, 2007, pp. 361–369.
[3] M. Alioto and G. Palumbo, "Analysis and comparison on full adder block in submicron technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 10, no. 6, pp. 806–823, 2002.
[4] Nangate, "Open Cell Library v1.3," 2009.
[5] Synopsys Inc., "Design Compiler," 2010.
[6] R. Zimmermann, "VHDL library of arithmetic units," in *The International Forum on Design Languages*, 1998, pp. 267–272. [Online]. Available: http://www.iis.ee.ethz.ch/~zimmi/arith_lib
[7] A. J. Martin, "Towards an energy complexity of computation," *Information Processing Letters*, vol. 77, pp. 181–187, 2001.