On Separable Error Detection for Addition

Michael B. Sullivan
The Cockrell School
of Engineering
The University of Texas at Austin
Austin, TX 78712
Email: mbsullivan@utexas.edu

Earl E. Swartzlander, Jr.
The Cockrell School
of Engineering
The University of Texas at Austin
Austin, TX 78712
Email: eswartzla@aol.com

***Abstract*—Addition is ubiquitous in computer systems, and rising error rates make error detection within adders increasingly important. This paper considers the best way to introduce strong, non-intrusive error detection to fixed-point addition within an existing, optimized machine datapath. A flexible family of separable error detection techniques called carry-propagate/carry-free (CP/CF) duplication is presented that offer superior error detection efficiency for a variety of adders.**

***Index Terms*—Adder, duplication, long residue checker (LRC), residue checking, lazy checker, low-cost error protection, self-testing and self-checking circuitry, computer arithmetic.**

I. Introduction

Adders are ubiquitous in computer systems, and a wide and diverse assortment of fixed-point adders are needed for different purposes. Addition is utilized for data manipulation, memory addressing, and control flow. Accordingly, the constraints on adders vary based on their purpose and the overall architecture. As such, extensive study has led to a wide variety of adder designs that vary in speed, area consumption, and power dissipation [1].

Prior research has investigated low-cost, strong, separable error detection mechanisms for *fast, unpipelined* fixed-point addition [2]. However, no cohesive, low-cost solution exists to provide separable protection for different fixed-point adder designs in an arithmetic pipeline. This study presents a flexible family of separable error detection techniques called *carry-propagate/carry-free (CP/CF) duplication* that provide superior error detection efficiency for a variety of adders.

One observation of this study is that separability *only* dictates the separation of designs—a separable error detector preserves the modularity of the checked adder, and lies completely off of the critical path of the protected design. This does not mean, however, that one can select the most efficient separable error detector irrespective of the design of the protected adder, as *the separable checker with the lowest overhead depends on the design of the checked adder*.

It is desirable that the selection of the most efficient separable checker be simple—chip designers already deal with enough complexity without having to consider interactions between the adder and checker. (After all, modularity is one of the compelling advantages of separable checkers in the first place!) With this in mind, the family of separable error detectors presented by this work is designed to be simple and parameterized. This simple and parameterized design makes the selection of the most aggressive separable checker easily automated or guided by simple heuristics.

The rest of the paper is as follows. Section II reviews some basic notation and concepts, and surveys the state-of-the-art in separable error detection for adders. Section III develops a parameterized family of strong error detection mechanisms that can adapt to the parameters of an adder design. Section IV-A draws upon prior literature to develop a set of baseline adders that represent an efficient design across a variety of time budgets. Using these baseline adders, Section IV-B evaluates the efficacy of different separable adder designs, including the novel family of detectors considered in this study. Matching a separable error detection mechanism to each adder achieves superior area and energy efficiency without impacting the design or timing of the main adder. Findings show that, CP/CF duplication offers a ~2–28% decrease in checking circuitry area and a ~14–66% reduction in energy consumption relative to the next most efficient separable mechanism.

II. Separable Error Detection

Before describing the main contribution of this paper, some basic error detection terminology and notation are reviewed. The current state-of-the-art separable error detectors for addition are also surveyed.

A. Notation and Error Model

Unless otherwise noted, the two operands for each addition are denoted A and B, and the output Z. The input word size of each operation is represented by N. An arithmetic error occurs when an addition fails to produce the correct result.

It is assumed that an error is caused by a single event transient (SET) that can propagate to affect one or more connected components in the adder or checker[1]. Due to the separability of every considered scheme, no common-mode errors can simultaneously affect both the adder and checker.

Unless otherwise noted, the target error coverage of each scheme is to detect any SET. This coverage represents a strong level of error protection (akin to the coverage of duplication) against a single erroneous event.

[1]This notation is consistent with [3]. In some work, this error model is referred to as a *multiple error* [4] or a *single distributed fault* [5].

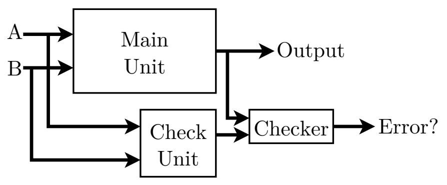


Fig. 1. The design of a separable error checker.

B. Separability

Systematicity and separability are two important classifying properties of error detecting codes. An error detecting code is *systematic* if the data information bits and check bits are distinct such that the data can be extracted from the codeword without first passing through a decoder. A code is *separable* if the checking procedure can operate without any communication from checked unit. By construction, all separable codes must be systematic.

Separability is a desirable property for an error detection scheme for several reasons. First and foremost, separable codes make for modular designs, as shown by Figure 1, and allow chip designers to implement arithmetic units and checkers independently. This modularity lessens the design burden of reliable execution. Also, separable designs can operate completely off of the critical path (during error-free execution) and allow pre-existing and highly optimized arithmetic units to be used without the error detector introducing new design constraints and unwanted timing effects. For these reasons, this paper focuses only on separable error detection. Residue checking has been proven to be the only separable error code for adders (apart from duplication) [6], which limits the number of competing designs. The current state-of-the-art separable error detectors for addition are reviewed below.

C. Duplication

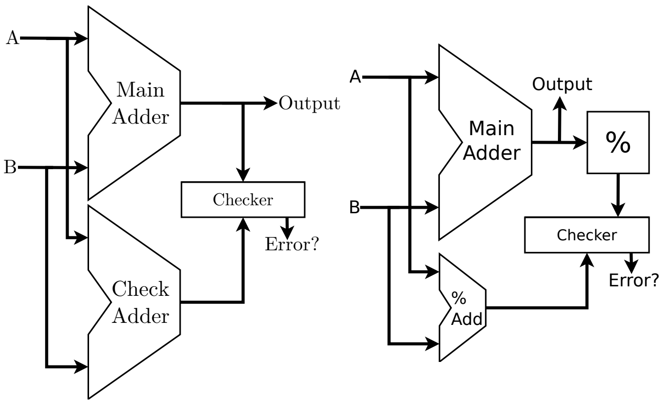
Coarse-grained duplication (also known as dual modular redundancy [DMR]) is simple, intuitive, strong, separable, and general, and it may be applied to addition (as shown in Figure 2a). The area and power costs of duplication generally keep it from being competitive with code-based approaches for fast adders. One observation of this study, however, is that careful duplication is an efficient error detection technique for slow, pipelined adders. The fact that duplication has advantages in some parts of the adder design space is utilized in Section III during the creation of a flexible family of error detectors.

D. Residue Checking

Addition can be checked by testing the equality of Equation (1), where $|N|_A = N \bmod A$ [6]. 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.

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

This scheme, shown in Figure 2b, is called residue checking. Residue checking represents an attractive error detection option for large fixed-point multipliers (the code is also closed under multiplication) [7], [8]. For addition, however,



(a) Duplication (b) Residue Checking

Fig. 2. Arithmetic error detection through duplication and residue checking. The % unit represents residue generation, and the % adder performs residue generation and modular arithmetic.

the scheme has some notable drawbacks [2]. First, residue checking offers incomplete error coverage against SETs, providing less error detection capability than other separable error detectors. Also, the generation of a residue following addition ($|Z|_A$) makes the error detection latency of residue checking longer than that of other separable schemes. Finally, the area and power efficiency of residue checking is not competitive with other error detection approaches for addition.

E. Long Residue Checking and Lazy Checking

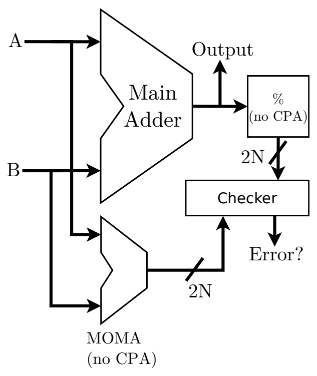
While traditional residue checking does not represent an efficient option for error detection in an adder, an extension of the scheme is the the state-of-the-art separable error detector for fast adders [2]. The cost of a modified residue checking design (shown in Figure 3a) that manipulates the test from Equation (1) to Equation (2) is inversely proportional to the residue width, $\log_2(A)$. Thus, the modified residue checking design with the largest possible residue width $(\log_2(A) = n)$ is the least complex, most power efficient, has the highest error coverage, and the lowest latency. Such a design is called the long residue checker.

$$\big||a|_A + |b|_A - |c|_A\big|_A \overset{?}{=} 0 \tag{2}$$

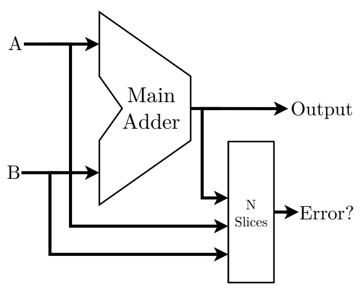
Careful inspection shows that the long residue checker can be implemented by a bit-sliced design (shown in Figure 3b) using slices composed of a full-adder with complemented carry-in and an XNOR gate (slice shown in Figure 3c). Functionally, this is similar to another scheme called the lazy adder checker [9]; both the LRC and lazy checking are bit-sliced designs, and the two have an equivalent error coverage against SETs. Figure 3d shows the bit-slice used by lazy checking. The LRC has been shown to be more efficient than lazy checking using standard cell synthesis, with most efficiency gains due to the ability of long residue checking to leverage an efficient full adder design while using only standard library cells [2].

III. A family of error detectors for addition

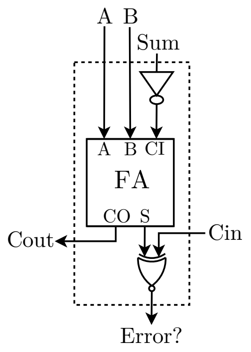
While many varied adder designs are in use, no existing separable mechanism provides the most efficient detection



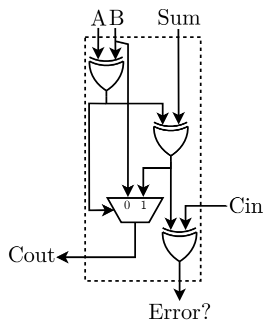
(a) Modified Residue Checking



(b) Bit-Sliced SET Checker

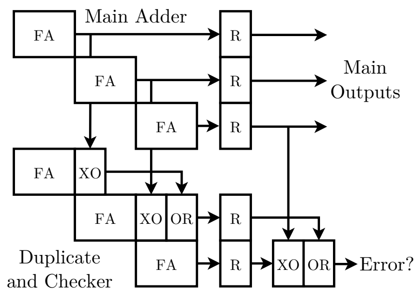


(c) Long Residue Checker

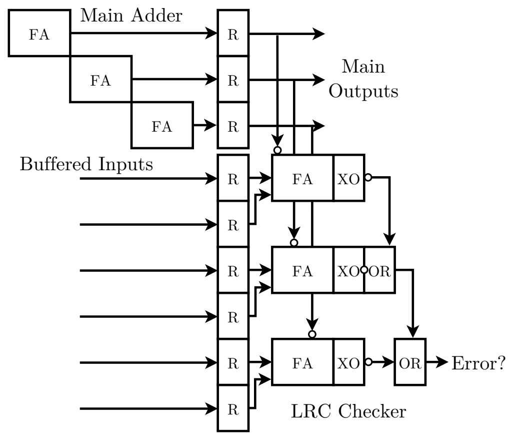


(d) Lazy Adder Checker

Fig. 3. The modified residue checking scheme used for long residue checking, and the bit-sliced design used for both the LRC [2] and the lazy adder checker [9]. The modified % unit finds the residue in a carry-free format, and the MOMA unit represents a multi-operand modular adder with no final carry-propagate adder (such that its output is in a carry-free format). Internals of a 1-bit slice of each checker are also shown.



(a) Ripple-Carry Duplication



(b) Long Residue Checking

Fig. 4. Ripple-carry duplication and long residue checking for a 3-bit ripple-carry adder. Cells labeled FA denote full adders, R denote registers, XO denote XOR gates, and OR denote OR gates. While both ripple-carry duplication and long residue checking use the same number of logic cells, duplication uses fewer pipeline registers due to some overlapped execution. For simplicity, two-rail checking logic is not shown.

of SETs in all adder designs. As an example, long residue checking is the most efficient known strong error detector for protecting fast, unpipelined adder designs, but the careful application of duplication may be the most efficient strong, separable error detector available for pipelined ripple-carry adders. This is because both ripple-carry duplication and long residue checking use the same number of logic cells, but ripple-carry duplication can overlap some checking with the main adder if the least-significant result bits are produced first. This overlapped execution can reduce the number of pipeline registers that are required for error detection precipitously. Figure 4 illustrates these savings through the example of a 3-bit ripple-carry main adder (two-rail checking logic is not shown).

Based on the above observations, this study investigates a parameterized family of error detectors that combines the advantages of the long residue checking with those of duplication. The family is characterized by three parameters: N (the input width), lw (the *logic width* of duplication), and cw (the *checking width* of duplication). Functionally, the lw least-significant bits of addition are computed using partial duplication, with cw of these bits being fully checked before the main addition completes. In the pipeline stage following the main addition, the remaining $(lw - cw)$ *duplicated-but-unchecked* result bits are equality checked, and the $(N - cw)$ *unduplicated* bits are checked via long residue checking. This scheme is referred to as carry-propagate/carry-free (CP/CF) duplication owing to the fact that a carry-propagate adder is used to check the least significant bits of the result while a carry-free adder is used to check the most-significant bits.

Figure 5 shows a block diagram of CP/CF duplication. It should be noted that the parameters of CP/CF duplication must be chosen in a reactive manner in order to stay off of the critical path and maintain separability. Because of the constraint of separability, carry-propagate/carry-free duplication degenerates to long residue checking for protecting a fast adder with perfectly balanced outputs but can operate as ripple-carry duplication for protecting a very slow adder. Figure 4 can be used to demonstrate this—Figure 4a can be thought of as CP/CF duplication with ($lw=3$, $cw=2$) and Figure 4b as a design with (lw=0, cw=0). Table I qualitatively summarizes the properties of carry-propagate/carry-free duplication relative to other separable error detectors. Designers can tailor the behavior of CP/CF duplication to match the delay of an adder, achieving low-cost separable error detection regardless of the speed and design of the protected circuit.

IV. Evaluation

The efficiency and flexibility of CP/CF duplication is demonstrated by protecting a range of adders with different speeds. Section IV-A describes the experimental methodol-

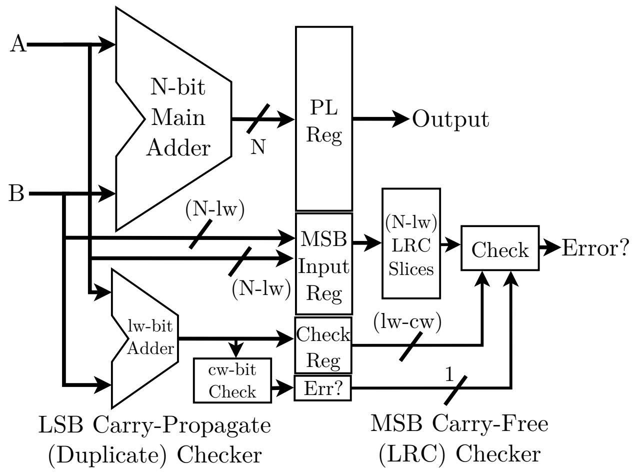


Fig. 5. A block diagram of carry-propagate/carry-free duplication. A partial adder duplicates the lw least-significant bits of addition, cw of which are checked in the first pipeline stage (in parallel with the main addition). In the following pipeline stage, the $(lw-cw)$ duplicated-but-unchecked bits are equality checked, and the $(N-cw)$ unduplicated bits are checked via long residue checking. The carry-in bit (which is passed through a register to the LRC checker) is not shown.

TABLE I
SUMMARY OF SEPARABLE ADDER ERROR DETECTORS.

Scheme	FAs, XOs, ORs	Latches	Checking Latency
Ripple-Carry Duplication	N	Few	Short–Very Long
Long Residue Checking	N	Many	Short
CP/CF Duplication	N	Few	Short

ogy used for this study and presents the selected baseline adders. Section IV-B analyzes the performance of different separable error detectors in a pipelined design with the different baseline adders and demonstrates the cost savings of CP/CF duplication.

A. Experimental Methodology

In-house circuits are used for error detection; the baseline adders mix in-house designs (for flexible addition) with those from a cell-based arithmetic unit library [10] (for Brent-Kung addition [11]) and those from the DesignWare library [12] (for ripple-carry addition). Synthesis is performed using the Synopsys toolchain, targeting the 40nm TSMC standard cell library. All baseline adders are compiled with high mapping and area optimization efforts. Synthesis is performed in a bottom-up, hierarchical manner such that no changes are made to the main adder or to the clock period. Energy calculations rely on gate-level power estimates and assume that the latency of the main adder dictates the clock frequency. Two-rail equality checkers are used to generate dual error outputs.

Following the mixed serial/parallel prefix methodology used for flexible parallel prefix addition [1], five different 32-bit baseline adders are considered. The adders use slow, ripple-carry (RC) propagation for the least-significant bits of addition and Brent-Kung (BK) parallel prefix addition for all remaining

TABLE II
BASELINE ADDERS WITH VARYING DELAY BUDGETS.

Design	Delay (ns)	Area (μm^2)	Energy pJ/op
BK-32	0.29	1008	0.64
{BK-24, RC-8}	0.45	842	0.59
{BK-16, RC-16}	0.75	792	0.58
{BK-8, RC-24}	1.01	737	0.56
RC-32	1.21	719	0.56

TABLE III
AREA AND ENERGY COSTS OF THE BASELINES.

Design	Area		Energy	
	% Reg	% Logic	% Reg	% Logic
BK-32	39	61	65	35
{BK-24, RC-8}	47	53	71	29
{BK-16, RC-16}	50	50	73	27
{BK-8, RC-24}	54	46	75	25
RC-32	55	45	75	25

bits. Table II gives the delay of each selected baseline, along with its area and energy demands. It can be seen that adder costs go down with decreasing speeds, such that an error detection mechanism must change proportionally in order to be competitive across all time scales. Table III gives the relative cost of pipeline registers and logic for each baseline design. It is apparent that the relative cost of pipeline registers is inversely proportional to the speed of addition, which indicates that a pipelining-cognizant approach is needed at slow speeds.

B. Results

The costs of three different separable error detectors are investigated using the above methodology. Table IV shows the area and energy required for traditional DMR (as shown in Figure 2a, with the checker in the pipeline stage following addition), long residue checking (as shown in Figure 3b, with the checker in the following pipeline stage[2]), and carry-propagate/carry-free duplication. The area and energy cost of registers, logic, and the total cost (including both registers and logic) are shown. The design with the lowest total cost is highlighted in bold.

In addition to the absolute hardware costs of separable error detection for addition, Table V shows the area and energy overheads required relative to the cost of each baseline adder. These data represent the percent area and energy costs required to protect each baseline design from error.

Traditional duplication uses a constant number of registers, such that its register costs remain invariant across designs. Meanwhile, the absolute logic costs of full duplication scale with the speed (and cost) of the checked adder. As expected, duplication requires $> 100\%$ overhead for logic, because a duplicate main adder and a two-rail checker are used to detect an error. The two duplicate adders share the input registers, and duplication naturally compresses the amount of state that needs to be propagated to the checker. These two factors conspire to give duplication modest register costs, resulting

[2]Prior work describes the LRC for an unpipelined adder [2]. As such, this pipelining scheme is based upon that used by the lazy adder checker [9].

TABLE IV
THE ABSOLUTE COST OF SEPARABLE ERROR DETECTION FOR AN ADDER.

Duplication						
Design	Area (μm^2)			Energy pJ/op		
	Reg.	Log.	Tot.	Reg.	Log.	Tot.
BK-32	134	749	883	0.174	0.318	0.492
{BK-24, RC-8}	134	584	718	0.176	0.264	0.440
{BK-16, RC-16}	134	534	668	0.177	0.249	0.426
{BK-8, RC-24}	134	479	613	0.178	0.233	0.410
RC-32	134	461	594	0.180	0.231	0.411

Long Residue Checking						
Design	Area (μm^2)			Energy pJ/op		
	Reg.	Log.	Tot.	Reg.	Log.	Tot.
BK-32	264	254	518	0.263	0.207	0.471
{BK-24, RC-8}	264	254	518	0.262	0.208	0.470
{BK-16, RC-16}	264	254	518	0.262	0.208	0.470
{BK-8, RC-24}	264	254	518	0.263	0.208	0.471
RC-32	264	254	518	0.262	0.208	0.470

Carry-Propagate/Carry-Free Duplication						
Design	Area (μm^2)			Energy pJ/op		
	Reg.	Log.	Tot.	Reg.	Log.	Tot.
BK-32	247	260	**508**	0.246	0.160	**0.407**
{BK-24, RC-8}	213	269	**482**	0.206	0.150	**0.356**
{BK-16, RC-16}	138	302	**440**	0.136	0.130	**0.266**
{BK-8, RC-24}	113	281	**394**	0.110	0.116	**0.226**
RC-32	49	323	**372**	0.041	0.097	**0.138**

TABLE V
THE RELATIVE OVERHEADS OF SEPARABLE ADDER ERROR DETECTION.

Duplication						
Design	Area (%)			Energy (%)		
	Reg.	Log.	Tot.	Reg.	Log.	Tot.
BK-32	13.3	74.4	87.7	27.1	49.6	76.7
{BK-24, RC-8}	15.9	69.3	85.2	29.7	44.6	74.3
{BK-16, RC-16}	16.9	67.4	84.3	30.8	43.2	74.0
{BK-8, RC-24}	18.2	65.0	83.1	31.7	41.5	73.2
RC-32	18.6	64.1	82.7	32.0	40.9	73.0

Long Residue Checking						
Design	Area (%)			Energy (%)		
	Reg.	Log.	Tot.	Reg.	Log.	Tot.
BK-32	26.2	25.2	51.4	41.0	32.4	73.4
{BK-24, RC-8}	31.3	30.2	61.5	44.4	35.2	79.5
{BK-16, RC-16}	33.3	32.1	65.4	45.5	36.0	81.5
{BK-8, RC-24}	35.8	34.5	70.2	46.9	37.1	84.1
RC-32	36.7	35.3	72.0	46.5	36.9	83.4

Carry-Propagate/Carry-Free Duplication						
Design	Area (%)			Energy (%)		
	Reg.	Log.	Tot.	Reg.	Log.	Tot.
BK-32	24.6	25.8	**50.4**	38.4	25.0	**63.4**
{BK-24, RC-8}	25.4	31.9	**57.3**	34.9	25.4	**60.2**
{BK-16, RC-16}	17.4	38.1	**55.5**	23.6	22.5	**46.1**
{BK-8, RC-24}	15.4	38.1	**53.5**	19.6	20.7	**40.2**
RC-32	6.82	44.9	**51.7**	7.31	17.2	**24.5**

in ~82–88% area and ~73–77% total energy overheads. Duplication is slightly more efficient for slower adders, because the relative cost of data movement dominates at slow speeds (see Table III).

Because of its structure and efficient implementation, long residue checking uses significantly less logic area than duplication across all adder designs. However, due to the need to pass both inputs (and the carry-in bit) to the checker, the register overhead of the LRC is roughly double that of full duplication. The low logic and substantial register costs of the LRC add up to a ~51–72% total area overhead. The relative cost of pipeline registers increases with the delay budget, making the LRC less energy efficient than duplication at protecting slow adders. Note that long residue checking requires <100% area overheads to protect ripple-carry addition, despite the fact that an LRC slice contains a full adder cell. This is because a DesignWare ripple-carry adder was used as the baseline, which (at the delay budget used) alternates full adder cells with more expensive logic in order to increase speed. Long residue checking requires a constant number of registers and check slices, such that the absolute cost of the LRC never changes. Due to the scaling behavior of the baseline adders, the LRC is relatively more efficient at protecting faster designs.

Carry-propagate/carry-free duplication overlaps some checking with execution, significantly decreasing the amount of pipelined state (and therefore register costs) relative to LRC checking. Meanwhile, the second-stage carry-free checker decreases the logic costs relative to duplication. When the costs of both logic and data movement are taken into account, CP/CF duplication achieves superior area and energy efficiency across all designs and time budgets. Carry-propagate/carry-free duplication offers a ~2–28% decrease in total checking area and a ~14–66% reduction in total checking energy consumption relative to the next most efficient separable mechanism. Table VI shows the logic depth and checking depth parameters chosen for the carry-propagate/carry-free checkers used in Table IV and Table V. These parameters were determined through an automated computer search; due to the simplicity and parameterization of the error detection scheme, the time required for this search is not prohibitive.

V. DISCUSSION

Carry-propagate/carry-free duplication relies on the least-significant sum bits to be produced first in order to overlap some duplication with the main addition. This early production of the lower result bits is consistent with 2's complement addition. However, there are some designs (such as end-around-carry adders used for 1's complement addition [13]) that produce perfectly balanced outputs. For these designs, CP/CF duplication will degenerate to long residue checking.

For simplicity, this study assumes that the CP/CF carry-propagate (duplicate) adder is implemented using a ripple-carry adder. While preliminary results indicate that this duplicate adder is sufficient for many designs, using a more general form of carry-propagate duplication (such as one based

TABLE VI
THE CP/CF PARAMETERS USED IN TABLE IV AND TABLE V.

Design	CP/CF Parameters	
	Logic Width (lw)	Check Width (cw)
BK-32	4	2
{BK-24, RC-8}	8	7
{BK-16, RC-16}	18	16
{BK-8, RC-24}	21	19
RC-32	29	28

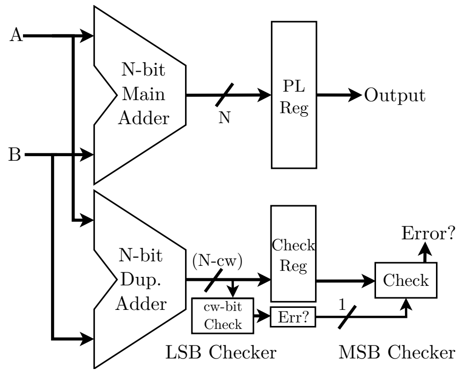


Fig. 6. An alternate error detection scheme that employs a split two-rail checker along with duplication.

on flexible parallel-prefix addition [1]) could give modest efficiency increases, especially for main adders with arbitrarily unbalanced outputs. Analysis of CP/CF duplication with a more flexible and aggressive duplicate adder is left for future work.

Some of the benefits of CP/CF duplication are due to its split two-rail checker, which opportunistically checks the least-significant bits of the duplicate result in parallel with the main adder. This split result checking can sometimes decrease the amount of registered state that needs to be propagated to the next pipeline stage. In addition to its use for CP/CF duplication, such a split checking scheme can also be employed with full duplication, as shown by Figure 6. Such a design offers superior register costs relative to traditional duplication, but does not reduce the significant costs of fully duplicated logic (as does CP/CF duplication). Initial experiments indicate that duplication with split checking operates as expected, providing a modest improvement over full duplication but failing to achieve the superior efficiency of CP/CF duplication. For brevity, a full analysis of duplication with a split checker is left for future work.

For reasons explained in Section II, traditional residue checking is not competitive at protecting adders. In some computer organizations, such as in a scalar computer or one that uses multiply-accumulate operations for both multiplication and addition, some residue checking circuitry could possibly be shared between an adder and multiplier without impacting performance. This paper focuses on the protection of a single adder and assumes that no protection circuitry can be shared among different functional units; cost studies under alternative computer organizations are left as future work.

This study focuses on arithmetic error detection in a generative context, where operands arrive to the adder in an unencoded format (or use a different separable encoding scheme). An alternative organization is to protect addition in an end-to-end fashion, where on-chip memory and data movement are protected by the same error code as addition. Such an strategy is not considered in this study, as it pushes complexity to the memory and data movement subsystems (the cost of which often surpass that of arithmetic in control-intensive architectures). Also, other operations that do not preserve the error code suffer an increase in complexity under end-to-end arithmetic error detection. This diffusion of implementation costs makes evaluating end-to-end schemes difficult without fixed knowledge of the microarchitecture in which a protected adder operates.

VI. Conclusion

Strong and efficient separable error detection mechanisms are needed to ensure reliable addition without prohibitive design costs. Meanwhile, a plethora of adder designs exist and the separable checker with the lowest overhead depends on the design of the checked adder. This study investigates a flexible family of techniques called carry-propagate/carry-free (CP/CF) duplication that provide efficient, strong, separable error detection for a variety of adders. Preliminary results indicate that CP/CF duplication achieves superior area and energy efficiency across a multiplicity of designs and time budgets.

Acknowledgments

Michael Sullivan's research was supported by the Temple Foundation through an MCD Fellowship in Engineering. Earl Swartzlander is supported in part by a grant from AMD, Inc. The authors would like to thank Mehmet Başoğlu for helping to prepare this paper for publication.

References

[1] R. Zimmermann, "Binary adder architectures for cell-based VLSI and their synthesis," Ph.D. dissertation, Swiss Federal Institute of Technology (ETH), 1998.

[2] M. B. Sullivan and E. E. Swartzlander, Jr., "Long residue checking for adders," in *Proceedings of the International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2012, pp. 177–180.

[3] P. Dodd and L. Massengill, "Basic mechanisms and modeling of single-event upset in digital microelectronics," *IEEE Transactions on Nuclear Science*, vol. 50, no. 3, pp. 583–602, 2003.

[4] D. K. Pradhan, Ed., *Fault-tolerant computing: Theory and technique*. Englewood Cliffs, NJ: Prentice Hall, 1986, vol. I.

[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] W. W. Peterson, "On checking an adder," *IBM Journal of Research and Development*, pp. 166–168, 1958.

[7] 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.

[8] M. Yilmaz, D. R. Hower, S. Ozev, and D. J. Sorin, "Self-checking and self-diagnosing 32-bit microprocessor multiplier," in *Proceedings of the International Test Conference (ITC)*, 2006, pp. 1–10.

[9] 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.

[10] 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

[11] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Transactions on Computers*, vol. 100, no. 3, pp. 260–264, 1982.

[12] S. Inc., "Designware IP library." [Online]. Available: http://www.synopsys.com/products/designware/

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