Low-Cost Duplicate Multiplication

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

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

Abstract—Rising levels of integration, decreasing component reliabilities, and the ubiquity of computer systems make error protection a rising concern. Meanwhile, the uncertainty of future fault and error modes motivates the design of strong error detection mechanisms that offer fault-agnostic error protection. Current concurrent hardware mechanisms, however, either offer strong error detection coverage at high cost or restrict their coverage to narrow synthetic error models. This paper investigates the potential for duplication using alternate number systems to lower the costs of duplicated multiplication without sacrificing error coverage. Two examples of such low-cost duplication schemes are described and evaluated; it is shown that specialized carry-save or residue number system checking can be used to increase the efficiency of duplicated multiplication.

Index Terms—Low-Cost Duplication, Dual Modular Redundancy, Low-Cost Error Detection, Concurrent Error Detection, Self-testing and Self-checking Circuitry, Computer Arithmetic.

I. Introduction

Shrinking feature sizes and the importance of computer systems make error protection a rising concern. Significant uncertainty remains with respect to the type and rate of errors expected in future systems, however [1], [2], [3], [4], complicating the design and analysis of reliability schemes. The focus of this research is to investigate the costs and benefits of strong, holistic error detection for multiplication that is not restricted to a narrow fault or error model. The concept of *low-cost duplication* is described and evaluated—organizations of duplicate checkers that use alternate number systems and redundant or carry-free arithmetic to avoid the limitations of duplication while preserving its strengths.

This paper is structured as follows. Section I reviews what can go wrong with arithmetic and describes the strengths and weaknesses of simple duplication. Section I-C describes a novel class of arithmetic error detectors based on a principle called low-cost duplication. Section II and Section III present prior work and the experimental methodology of this research, respectively, and Section IV investigates the design and overheads of using specialized carry-save checkers in low-cost duplicate multipliers. It is shown that carry-save duplication lowers the costs of duplication in a straightforward manner without sacrificing error coverage, checking latency, or separability. Section V investigates an alternate low-cost duplicate multiplier that uses the residue number system to good effect. Finally, Section VI explores some exciting future avenues that extend the results of this research.

A. Arithmetic Faults and Errors

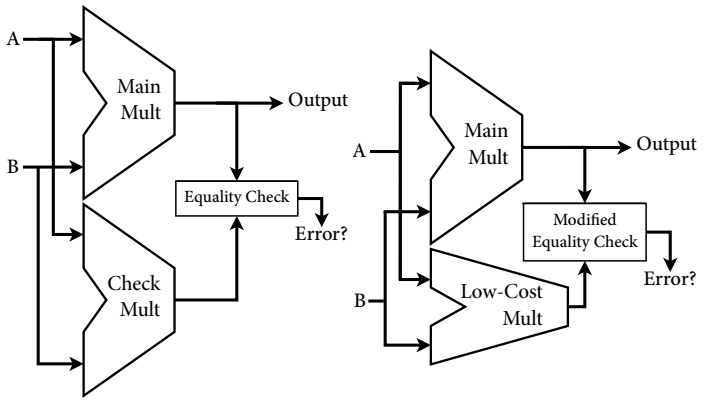
This paper adopts the established terminology that a *fault* is a physical phenomenon or defect that may cause an error or failure, an *error* is a discrepancy between the intended and actual data in a system, and a *failure* is an instance in time when a system displays a behavior that is contrary to its specification [5], [6]. A plethora of faults can cause arithmetic errors; the rate and severity of these faults is unknown in current technologies, and this uncertainty is exacerbated by the unpredictability of future technology challenges and limitations [7], [8], [9], [10], [11].

The faults that can affect arithmetic include, but are not limited to: transient faults caused by high energy particles in the environment [12], [8]; end-of-life gate oxide faults due to electromigration [13], hot carrier degradation [14], or time-dependent oxide breakdown [15], [9]; timing violations due to voltage droop [16], [17] or age and temperature-related slowdown [10], [18]; fabrication-related faults due to manufacturing defects [11] or technology variability [19]; and uncaught design faults due to insufficient testing [20] or insufficient cooling [21].

The typical approach to arithmetic error detection is either to attempt a best-effort mechanism with incomplete error coverage, such as residue checking [22], or to try and select the most prevalent faults and to deal with them using specialized mechanisms. Examples of such specialized mechanisms include error detectors for transient faults [23], [24], gate oxide faults [25], timing violations [26], [27], fabrication-related faults [28], and design faults [29].

While specialized error detection mechanisms boast low area and energy overheads, their use in a comprehensive arithmetic error detection scheme is fraught with difficulty. Choosing the most prevalent fault mode can be costly and difficult, especially given the uncertainty of future technologies, use-cases, environmental conditions, and design constraints. Also, whether and how a fault manifests as an error is strongly design dependent. This design dependence introduces additional development complexity—specialized or best-effort error detection mechanisms must sometimes dictate or react to the arithmetic unit design to have high coverage, and there can be a tradeoff between arithmetic unit optimization and error coverage [30].

In lieu of using such detection mechanisms, this work proposes the use of duplication for holistic, fault-agnostic arithmetic error detection. The high error coverage of duplication avoids the need to tailor error detection towards the most severe and prevalent errors, lessening the onus of fault rate analysis, error modeling, and system-level error propagation and failure modeling on the chip designer.



(a) Strict Duplication (b) Low-Cost Duplication

Fig. 1: A block diagram of a simple, strict duplicate multiplier and a low-cost duplicate multiplier. Simple duplication replicates the logic of the main multiplier, resulting in high area and energy overheads and incomplete coverage against correlated faults. Low-cost duplication makes use of carry-free arithmetic, diversified design, and any extra detection slack to reduce the costs of duplication and possibly increase its error coverage.

B. The Benefits and Limitations of Duplication

Concurrent error detection through duplicated execution and equality testing (also known as dual modular redundancy [DMR]) is a simple and well-known mechanism with many advantages. Figure 1a shows a block diagram of a simple DMR organization (called *strict duplication*) that uses a duplicate multiplier to check the results of computation. Duplication provides strong, low-latency error detection while maintaining complete design modularity and separability—no interaction is needed with the main arithmetic unit, and the duplicate error detector does not dictate or alter the on-chip storage or data movement sub-systems. The simplicity and separability of duplication facilitates its design and implementation, and its low detection latency can simplify higher-level error recovery mechanisms [31], [32].

Despite its various benefits, duplication suffers from some serious limitations. First and foremost, the high area and energy overheads of full duplication preclude it from being a realistic error detection scheme for most computers. Duplication has only been employed in specialized processors where reliability is of paramount importance, surpassing the need for efficiency [33], [34], [35], [36], [37]; even IBM uses alternative arithmetic error detection mechanisms in some of its notoriously reliable mainframe computers for increased efficiency [38], [39], [40].

In its simplest form, full duplication can also have incomplete coverage against certain faults. Timing violations, design faults, and fabrication faults can possibly affect both the main arithmetic unit and duplicate check unit, escaping error detection and possibly resulting in silent data corruption or a system failure.

C. Low-Cost Duplication

The goal of low-cost duplication is to preserve the various strengths of DMR while lessening its limitations. This paper investigates the potential of duplicate multipliers to employ alternate number representations that achieve superior efficiency. Because the output of the duplicate unit is discarded after error detection, many of the costs traditionally associated with non-standard number systems (such as the data movement and storage costs of redundant arithmetic or the costly conversion back to a weighted format) can largely be ignored. Figure 1b shows the organization of a low-cost duplicate checker. A low-cost duplicate multiplier employs an alternate number system to perform efficient multiplication, and the equality checker is modified to compare the output of the main multiplier with the alternate encoding of the low-cost unit.

While the primary goal of this research is to reduce the overheads of duplication, it is noted that low-cost duplication is amenable to closing the potential coverage holes of strict duplication[1]. A well-known way to detect design and fabrication faults is to ensure design diversity between the main arithmetic unit and checker [5]; this approach has been recognized since the beginning of mechanical computation [41], [5] and has been used to target hardware bugs [29]. Since the most efficient main arithmetic unit should be employed, however, diversified duplication normally implies some inefficiency. Low-cost duplication makes use of redundancy and alternate number representations in the checker, and is naturally diversified by construction. As the results of the duplicate low-cost checker are in an alternate encoding, low-cost duplication can be more efficient than the main arithmetic unit despite this diversification. Timing violations can be correctly diagnosed by keeping all checking circuitry well off of the circuit's critical path [25]. With traditional duplication, this implies some additional checking latency. By utilizing alternate number systems, low-cost duplication may be able to operate faster than the main arithmetic unit and thus be non-critical without additional checking latency.

Low-cost duplicate units can be thought of in two ways. First, low-cost duplication can be considered as a more efficient replacement for DMR in systems demanding high reliability and availability—in fact, its substantive cost savings may even make low-cost duplication a viable approach in systems where DMR is prohibitively costly. Alternatively, low-cost duplication can be thought of as a more optimized baseline for the evaluation of specialized or best-effort error detection mechanisms. Currently, the assumption is often made that concurrent, low-latency duplication requires >100% implementation overheads. This research suggests, however, that with careful design this may not always be the case.

II. Prior Work

A. Low-Cost Duplicate Addition

Redundant carry-save addition has been used to check the result of a fast adder [42]. This prior work can be thought of as low-cost duplication for addition; this research extends the concept to check the results of multiplication. The modified equality checkers for that are used in this work for carry-save numbers are similar to the carry-free circuits used elsewhere [43], [44], [45], [42] for carry-save arithmetic.

[1] While the efficacy of low-cost duplication for problematic faults is noted, a full error coverage evaluation is left for future work.

B. Lazy Duplication

Prior research has employed a simplified, long-latency duplicate unit to detect errors in arithmetic [29], [46]. This work differs from low-cost duplication in two respects. First, these duplicate units perform arithmetic in the same number representation as the main arithmetic unit, and do not fit the definition of a specialized low-cost duplicate unit considered in this paper. Also, this research aims to maintain the low-latency error detection of strict duplication in order to simplify implementation, cheapen higher-level recovery, and avoid a dependence on aggressive latency-tolerant microarchitectural features to lessen performance loss. In order to evaluate the merits of lazy duplication in the context of low-latency error detection, experiments include baselines that are allowed to utilize any available slack to lower their overheads.

C. Residue Checking

Any study of error detection for multiplication would be remiss if it did not mention residue checking. Addition, subtraction, and multiplication can be checked by testing the equality of Equation 1, where $|X|_m = X \bmod m$ and $\oplus$ denotes the protected operation [22]. 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.

$$|X \oplus Y|_m \stackrel{?}{=} ||X|_m \oplus |Y|_m|_m \tag{1}$$

The error coverage of a residue code depends on the width of its checking modulus. In general, large checking moduli are prohibitively expensive, such that residue checking suffers from some coverage holes [47], [30]. It is likely that low-cost duplication will have a lower latency[2] and higher error coverage than residue checking, at the expense of more checking hardware.

It is possible to use multiple co-prime moduli in a multi-residue code for higher error coverage [48], [49]. Low-cost RNS duplication (Section V) can be thought of as a fully parallel multi-residue code with a dynamic range large enough to have complete error coverage. In this context, it is obvious that (multi) residue checking can be less expensive than RNS duplication, but with incomplete error coverage against severe single-component errors.

III. Experimental Methodology

The remainder of this paper focuses on the design and implementation of low-cost duplicate checkers. Before these contributions, the experimental methodology used throughout the paper is described. Gate-level design space exploration is used to examine the area and energy required for each circuit. The Synopsys toolchain is used for synthesis, targeting the 40nm TSMC standard cell library [50], [51]. All circuits are compiled using the Synopsys Design Compiler with high mapping effort and optimization options consistent with an area-optimized implementation. Structural Verilog descriptions of each circuit are used throughout, with compressor-based multipliers and minimum-depth parallel prefix adders. The Synopsys Design Compiler provides area (in μm^2) and timing (in ns) estimates for each design. A TSMC wire model is used for timing estimates, and it is assumed that each circuit is driven by, and drives, a pipeline register. Dual-rail encoded equality checkers are used at the output of error detection to create totally self-testing designs.

[2] The higher latency of residue checking relative to low-cost duplication comes from the need to generate the residue of the result before checking. The latency of residue generation is inversely proportional to the modulus width, such that the checking latency is greatest for the smallest residue codes.

TABLE I: The baseline multipliers selected for this work.

Width (N)	Critical Latency [ns]	AT-Efficient Latency [ns]	AT-Efficient Area [μm^2]
16	1.21	1.25	3547.8
32	1.70	1.73	12937.9
64	2.30	2.34	38179.8

A. The Baseline Multipliers

Three efficient unsigned binary multipliers are selected to serve as the main arithmetic unit baselines at 16, 32, and 64 bits. The Pareto-optimal (over area and time) post-synthesis design which minimizes the AT metric is chosen at each word length through a search of the design space. Table I gives the properties of the selected baselines.

B. Strict and Lazy Duplication

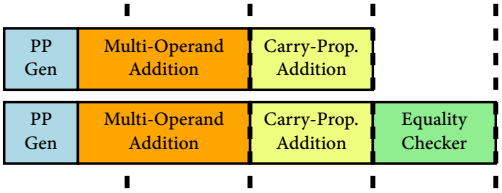
Two baseline DMR organizations are considered: strict duplication, which uses a mirrored multiplier for checking, and lazy duplication, which uses any extra detection latency to reduce the cost of the duplicate multiplier. Figure 2 visualizes both strict and lazy duplication. A typical parallel fixed-point multiplier goes through three steps of computation: partial product generation, the multi-operand addition of the partial products, and the carry-propagate addition of the redundant carry-save product. The steps of a multiplier are shown over time along with the steps of a duplicated checker. Strict duplication proceeds in lockstep with the main multiplier; lazy duplication utilizes the extra checking latency for modest cost savings.

Strict duplication requires a 30–20% detection latency and 102.8%, 102.3%, and 101.2% area overheads to protect the 16, 32, and 64-bit multipliers, respectively. This reduction in relative overheads with increasing word length is due to the quadratic scaling behavior of parallel multipliers—the dual-rail equality checker scales linearly, such that its contribution to the total area becomes smaller.

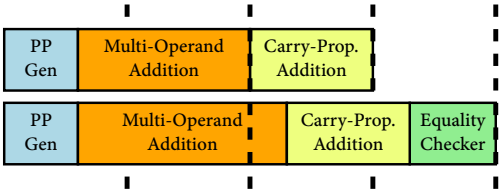
The >100% implementation overheads of strict duplication are consistent with many prior assumptions about DMR organizations, but the overheads of duplication can be easily decreased by utilizing extra detection latency. Later, Table II gives the estimated overheads for lazy duplication. Lazy duplication considerably lowers the overheads of strong error detection by utilizing any available slack, but it may still require a prohibitive amount of overhead for many applications. Conversely, lazy duplication may demand too much detection latency to achieve a sufficient efficiency; Section IV shows that carry-save duplication amends this deficiency.

IV. Carry-Save Low-Cost Duplication

A simple and straightforward scheme for low-cost duplicate multiplication is to eliminate the final carry-propagate adder

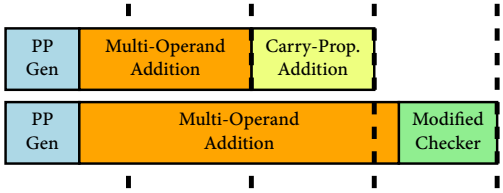


(a) Strict Duplication Visualization

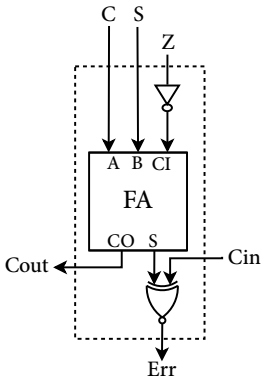


(b) Lazy Duplication Visualization

Fig. 2: Strict duplication uses a mirrored multiplier for checking, whereas lazy duplication utilizes any extra detection latency to reduce the complexity of the duplicate multiplier. Simple timing diagrams are shown for a multiplier; the relative time spent in each step of computation is not accurate and is for visualization purposes.



(a) Lazy Carry-Save Duplication Visualization



(b) A Slice of the Modified Carry-Save Checker

Fig. 3: Carry-propagate addition can be eliminated in the duplicate multiplier by employing a modified equality checker. One slice of the modified equality checker is shown.

in the duplicate multiplier, checking its result directly in the redundant carry-save representation. This scheme is referred to as *lazy carry-save duplication*.

Lazy carry-save duplication can increase efficiency over traditional lazy duplication in two ways. First, the cost of a carry-propagate adder is replaced with a slight increase in the complexity of the modified checker; the cost increase of the checker is strictly equal to or less than that of the carry-propagate adder, leading to some savings [42]. Second, and more notably, the latency of carry-propagate addition is avoided; this additional checking slack may be used to reclaim checking efficiency in a manner similar to lazy duplication. The latency of this carry-propagation is significant, taking roughly a third of the time for a logarithmic-time multiplication[3]. A visualization of the carry-save duplication process is shown in Figure 3a. A bit-sliced modified checker (Figure 3b) is used to check the results of multiplication, similar to its application in [42], [52].

Table II evaluates the cost of both lazy and lazy carry-save duplication. The benefits of lazy carry-save duplication are significant and robust; several trends are of note. Carry-save duplication adds some additional latency relative to the fastest strict or lazy duplicate designs due to the need to perform modified checking following the main multiplication. For any achievable detection latency, carry-save duplication shows superior efficiency to lazy duplication, achieving roughly the same area efficiency as a lazy duplicate design with 30% additional detection slack.

The cost savings of carry-save duplication saturate quickly, such that it is neither necessary nor profitable to increase the checking latency past 30–40%. This is because there is sufficient slack in the absence of duplicate carry-propagate addition to use the least expensive standard cells at this point. For this

[3] The 16, 32, and 64-bit baselines spend roughly 28%, 37%, and 29% of their time performing the carry-propagate addition, respectively.

TABLE II: The overheads of lazy and lazy carry-save duplication. Lazy carry-save duplication avoids the need for carry-propagate addition in the duplicate unit, increasing efficiency.

Width (N)	Detection Latency (%)	Lazy Area [μm^2] (+%)	Lazy CS Area [μm^2] (+%)
16	40	2448.4 (69.0)	1869.8 (52.7)
	50	2273.8 (64.1)	1665.2 (46.9)
	60	2093.2 (59.0)	1670.9 (47.1)
32	30	9672.9 (74.8)	7306.8 (56.5)
	40	8380.9 (64.8)	6989.8 (54.0)
	50	7855.8 (60.7)	6899.5 (53.3)
	60	7299.6 (56.4)	6899.5 (53.3)
64	30	33760.2 (88.4)	28118.6 (73.6)
	40	30698.8 (80.4)	27625.9 (72.4)
	50	29834.2 (78.1)	27625.9 (72.4)
	60	28153.1 (73.7)	27625.9 (72.4)

reason, lazy duplication and carry-save duplication asymptote to the same efficiency; carry-save duplication just gets there more quickly. The use of a slower and more area-efficient design (or the use of a flexible delay-proportional multiplier, such as the DesignWare PPArch multiplier [53]) would almost certainly allow for carry-save duplication to provide detection latency-proportional savings.

A. Carry-Save Karatsuba Duplication

The implementation of carry-save multiplication is straightforward for a fully parallel, tree-based multiplier like the considered baselines. There are alternative multiplier architectures, however, where the adoption of carry-save checking is slightly more nuanced. With careful design, carry-save checking can apply to a wide class of parallel multipliers. This section demonstrates

TABLE III: The overheads of lazy and carry-save Karatsuba duplication (relative to a baseline Karatsuba multiplier).

Width (N)	Detection Latency (%)	Lazy Area [μm^2] (+%)	Lazy Carry-Save Area [μm^2] (+%)
32	20	2651.1 (66.7)	1988.9 (50.0)
	30	2234.1 (56.2)	1799.1 (45.3)
	40	1834.9 (46.2)	1697.5 (42.7)
	50	1752.9 (44.1)	1693.6 (42.6)
	60	1712.0 (43.1)	1693.6 (42.6)
	10	9888.4 (89.8)	N/A
	20	9652.1 (87.7)	6800.7 (61.8)
	30	7531.7 (68.4)	6183.3 (56.2)
	40	6532.1 (59.3)	6044.7 (54.9)
	50	6221.2 (56.5)	5908.5 (53.7)
	60	6088.2 (55.3)	5888.2 (53.5)
64	10	32499.5 (91.0)	N/A
	20	30662.8 (85.9)	24355.4 (68.2)
	30	27025.9 (75.7)	23073.2 (64.6)
	40	24932.4 (69.8)	22947.2 (64.3)
	50	23534.6 (65.9)	22827.1 (63.9)
	60	23146.7 (64.8)	22812.8 (63.9)

the flexibility of carry-save duplication through its application to a Karatsuba multiplier.

Karatsuba multiplication (originally attributed to [54]) is a divide-and-conquer scheme that is able to perform N-bit fixed-point multiplication using three $\frac{N}{2}$-bit multipliers by exploiting Identity 2 (where a_H, a_L, b_H, and b_L represent the high and low halves of the input operands a and b, respectively) [55].

$$\left(2^{\frac{N}{2}}a_H + a_L\right)\left(2^{\frac{N}{2}}b_H + b_L\right) = 2^N a_H b_H + a_L b_L + 2^{\frac{N}{2}}\left((a_H + a_L)(b_H + b_L) - a_H b_H - a_L b_L\right) \quad (2)$$

Karatsuba multiplication can be somewhat area efficient at large word lengths, since the area of parallel multiplication tends to scale quadratically and it replaces a full N-width multiplier with just three smaller $\frac{N}{2}$ multipliers [55]. However, hidden in the $2^{\frac{N}{2}}\left((a_H + a_L)(b_H + b_L) - a_H b_H - a_L b_L\right)$ term of Karatsuba multiplication is a lengthy adder carry propagation before the results of $(a_H + a_L)(b_H + b_L)$ can be determined. This additional latency negatively impacts the overall efficiency of the scheme.

An optimized Karatsuba baseline is used that avoids many of the latency issues with the $(a_H + a_L)(b_H + b_L)$ term. By slightly modifying the multiplier, the inner subtractions $(-a_H b_H - a_L b_L)$ can be performed without any carry propagation. This is done as follows: a regular CSA is used with complemented inputs from the $a_H b_H$ and $a_L b_L$ multipliers. The two incrementations necessary for two's complement arithmetic are achieved in a carry-free manner by (1) setting the empty carry bit in the least-significant position of the CSA and (2) placing an extra bit in the partial product generation of the $(a_H + a_L)(b_H + b_L)$ multiplier. Neither incrementation has any impact on the latency or complexity of the resultant duplicate multiplier.

Carry-save Karatsuba duplication uses a carry-save checker for the final addition of the constituent subcomponents, as shown in Figure 4. Table III evaluates the overheads of both lazy and carry-free Karatsuba duplication relative to 16, 32, and 64-bit baseline Karatsuba multipliers. Again, carry-save duplication

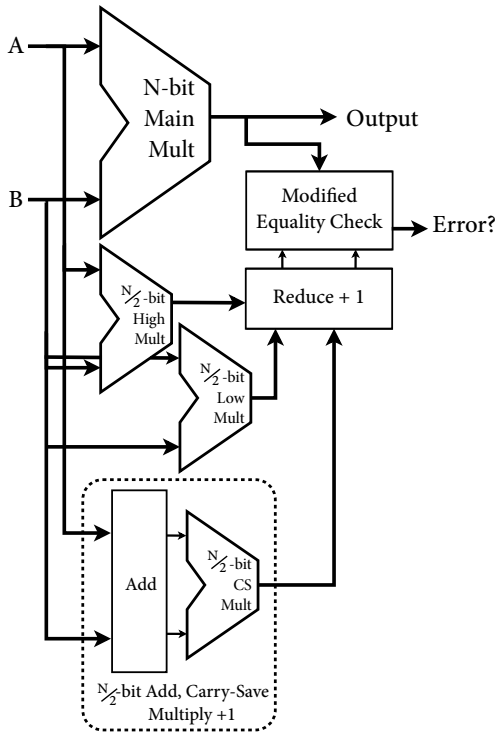


Fig. 4: Karatsuba multiplication can perform an N-bit fixed-point multiplication using three $\frac{N}{2}$-bit multipliers. Carry-free duplication uses a modified checker to eliminate the carry-propagate addition of the sub-components.

shows consistent, robust efficiency improvements. By avoiding the latency of the final carry-propagate addition, carry-save duplication is able to reach the efficiency of the asymptotic lazy checker with about a 20% detection latency; it takes lazy duplication 20–30% more slack to compete.

V. Residue Number System Duplication

A compelling low-cost duplication alternative using the residue number system (RNS) is described and evaluated. Before delving into the details of RNS duplication, the basics of the residue number system are reviewed.[4]

The residue number system represents integer values using a small number of non-weighted digits. An RNS number is formed from a weighted number, X with respect to a set of n co-prime bases, $(m_0|m_1|...|m_{n-1})$. To convert to the RNS representation, the residue of X is formed with respect to each base. In general, the formation of an arbitrary residue $\left(|X|_m ; m \in \mathbb{N}\right)$ is expensive; for efficiency, designs often restrict themselves to specialized moduli in the form $m = 2^a \pm 1, a \in \mathbb{N}$. Common arithmetic operations can be performed without carry propagation between the digits of RNS numbers, significantly increasing the speed. Operations within each digit are carried out in a modular manner such that the arithmetic result for the RNS digit corresponding to

[4] This short summary is felt to be sufficient in the context of low-cost duplication. For a more formal introduction, the reader is referred to [56]. For a comprehensive treatment of RNS arithmetic, see [57], [58], [59].

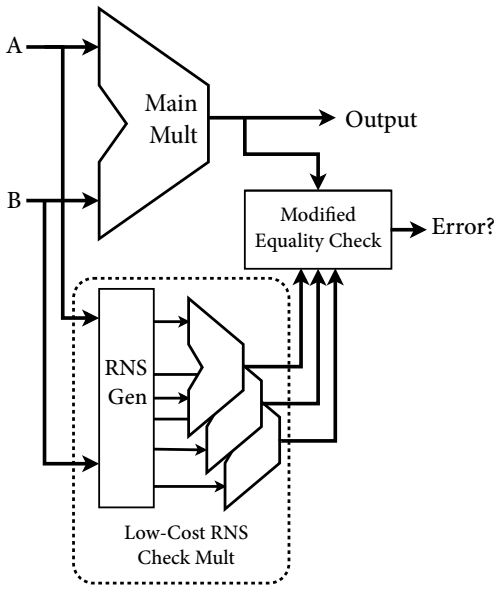


Fig. 5: A block diagram of RNS duplication. Multiplication on an RNS-encoded number can proceed more quickly than its fixed-point counterpart; this extra slack can lead to overall cost savings.

a modulus m is computed as $|X \oplus Y|_m = ||X|_m \oplus |Y|_m|_m$, where $\oplus \in \{+, -, *\}$.

Despite their fast arithmetic speed, the general-purpose usefulness of RNS numbers is greatly limited by practical concerns: bitwise logical operations, truncation, division, sign detection and magnitude comparison are all expensive in this representation, as is the conversion back to a fixed-point format. Low-cost duplicate RNS multiplication is able to exploit the superior speed and diversified design of the residue number system while avoiding the aforementioned limitations. Because the output of the duplicate RNS unit is discarded after error detection, no expensive operations are performed nor is backwards conversion necessary. Figure 5 shows an organization of RNS duplication—the input operands are converted to the RNS representation, and the modular arithmetic for each RNS modulus proceeds in parallel.

To evaluate the idea of RNS duplication, an RNS duplicate multiplier is formed. The residue generator circuitry from [60] is used along with the parallel modular multipliers from [61] (mod 2^a–1) and [62] (mod 2^a+1). Following the recommendations of [63], a moduli set in the form $\{2^a - 1,\ 2^a + 1,\ 2^b\}$ is employed. Table IV shows the overheads of RNS duplication, along with the moduli sets used[5]. The experimental results indicate that the speed advantages of RNS arithmetic provide modest cost savings in the duplicate multiplier through increased design slack. Also, because RNS arithmetic is faster than the main multiplication, there is sufficient slack to saturate these benefits at 30–40% detection latencies and further detection latency does not significantly lessen the overheads of detection.

Despite the increased speed of RNS multiplication, no drastic simplification of the duplicate multiplier is seen. A nuanced

[5] These moduli sets were chosen through a brief computer-guided search and are expected to be aggressive but not optimal.

TABLE IV: The overheads of RNS duplication (relative to the compressor-based fully parallel multiplier).

Width (N)	Checking Latency (%)	Modulus 1 ($2^a \pm 1$)	Modulus 2 (2^b)	Area [μm^2] (+%)
16	30	5	23	1982.2 (55.9)
	40	5	23	1974.4 (55.7)
	50	9	15	1959.6 (55.2)
	60	9	15	1913.9 (53.9)
32	30	18	29	8119.9 (62.8)
	40	18	29	7323.4 (56.6)
	50	17	31	7065.0 (54.6)
	60	17	31	6957.2 (53.8)
64	30	41	47	33170.1 (86.9)
	40	41	47	31410.9 (82.3)
	50	41	47	29938.6 (78.4)
	60	41	47	29040.2 (76.1)

fact about the RNS representation hints at why this is the case. RNS multipliers typically take N-bit inputs and provide an N-bit output, whereas two's complement multipliers take N-bit inputs and produce a $2N$-bit result. This means that the duplicate RNS multiplier providing $2N$ bits of dynamic range is somewhat overdesigned, and actually could accept $2N$ bit inputs (though in the duplicate RNS organization such inputs will never occur).

These experimental results demonstrate that RNS duplication can provide diversified, low-latency duplication that is completely off of the critical path. This allows RNS duplication to provide strong, fault-agnostic error detection. The efficiency of RNS duplication is on par with that of lazy carry-save duplication (Section IV). Furthermore, it is possible that the RNS organization and experimental methodology used in this study could under-represent the potential efficiency of RNS duplication; Section VI-A describes some of the future research that these initial results warrant.

VI. Discussion

Low-cost duplication presents the opportunity for several exciting avenues of future research. Some discussion of this potential future work follows.

A. Further RNS Duplication Evaluation

There are several future experiments that may better represent and analyze the potential advantages of RNS duplication. First, it has been noted that some of the efficiency advantages of RNS arithmetic come from its increased circuit regularity relative to two's complement arithmetic units [55]; these layout advantages are not taken into account in this work for methodological reasons. Also, initial experiments indicate that an RNS duplicate multiplier can be faster than the main arithmetic unit, garnering modest area savings. This high speed could be better exploited for increased efficiency by more flexible modular multiplier designs or by using a multi-Vth design flow where small and low power (but slow) high Vth cells are mixed with their standard Vth equivalents.

B. Alternate Number Systems and Organizations

The low-cost duplicate checkers described by this work are based on carry-save and RNS arithmetic and by no means exhaust

the search space. Alternate low-cost duplicate checkers and different organizations of the described checkers undoubtedly exist. Some avenues of future research include other RNS organizations, including the use of pseudo-residues [64] or redundancy through non-coprime moduli [65], [66]. Also promising is the investigation of other number systems where multiplication is inexpensive, such as the logarithmic number system [67] (or approximate binary logarithms [68]) and the use of index calculus for inexpensive multiplication [69].

C. Security Applications

This study focuses on providing complete protection against single component errors—any feasible arithmetic fault can be detected so long as it is confined to either the main arithmetic unit or the low-cost duplicate checker. This error model represents complete protection against the rare, random, independent faults that typically impact system reliability[6]. There is some interest in the security community in detecting arbitrary component errors to guard against laser fault injection, where a targetted fault is induced by a nefarious agent [70], [71], [72]. While an analysis of the security potential of low-cost duplication is outside the scope of this study, it seems that the scheme may be a amenable to such applications. This may be especially true for RNS duplication, as the residue number system has a long pedigree of highly fault-tolerant behavior [57], [55]—by adding an additional redundant residue (and further modifying the equality checking circuit) it should be possible to tolerate more than one component failure.

D. Stochastic or Timing-Speculative Computing

It may be possible to exploit the benefits of low-cost duplication—strong, separable, low-latency error detection—to operate with unreliable hardware [19], [73] or with reduced timing and voltage margins [26], [27], increasing common-case efficiency while preserving correctness. The strength of low-cost duplication is crucial in such an approach, as it is necessary to guarantee correctness in the presence of errors[7]. Separability is important, as well, to allow for the most optimized and efficient main arithmetic unit to be used. Finally, low-latency error detection is important in this application, as the arithmetic error rate will be artificially inflated and low-latency detection cheapens fast microarchitectural replay [31].

VII. Conclusion

This paper describes low-cost duplication for error detection in multiplication—a class of duplicate error detection mechanisms that use specialized checkers to perform duplicate arithmetic using alternative number systems. Two such low-cost duplicate multipliers are evaluated that check results in a carry-save and RNS representation. These low-cost duplicate multipliers achieve superior efficiency in a straightforward manner without sacrificing error coverage, error detection latency, or separability, opening up exciting new avenues of future research.

[6] As discussed in Section I-C, design faults, fabrication faults, and timing violations may not be random and independent. Established techniques can be used to handle such faults using low-cost duplication.

[7] Some prior work investigates the use of residue checking for such a purpose [74], but it cannot guarantee correctness.

Acknowledgments

This work is supported in part by the following organizations: The National Science Foundation under Grant #0954107 and the DOE FastForward 2 Program.

References

[1] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward Exascale resilience," *International Journal of High Performance Computing Applications*, vol. 23, no. 4, pp. 374–388, 2009.

[2] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir, "Toward Exascale resilience: 2014 update," *Supercomputing frontiers and innovations*, vol. 1, no. 1, pp. 5–28, 2014.

[3] J. Shalf, S. Dosanjh, and J. Morrison, "Exascale computing technology challenges," in *High Performance Computing for Computational Science*. Springer Berlin Heidelberg, Jan. 2011, no. 6449, pp. 1–25.

[4] X. Yang, Z. Wang, J. Xue, and Y. Zhou, "The reliability wall for Exascale supercomputing," *IEEE Transactions on Computers*, vol. 61, no. 6.

[5] A. Avizienis and J. Kelly, "Fault tolerance by design diversity: Concepts and experiments," *Computer*, vol. 17, no. 8, pp. 67–80, Aug. 1984.

[6] I. T. C. on Real-Time Systems", "Terminology and Notations," http://tcrts.org/education/terminology-and-notation/, 2014.

[7] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, and L. Alvisi, "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2002, pp. 389–398.

[8] 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.

[9] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The impact of technology scaling on lifetime reliability," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2004, pp. 177–186.

[10] D. K. Schroder and J. A. Babcock, "Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing," *Journal of Applied Physics*, vol. 94, no. 1, pp. 1–18, Jul. 2003.

[11] J. W. McPherson, "Reliability challenges for 45nm and beyond," in *Proceedings of the Design Automation Conference (DAC)*, 2006, pp. 176–181.

[12] 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, June 2003.

[13] C.-K. Hu, D. Canaperi, S. Chen, L. Gignac, B. Herbst, S. Kaldor, M. Krishnan, E. Liniger, D. Rath, D. Restaino *et al.*, "Effects of overlayers on electromigration reliability improvement for cu/low k interconnects," in *Proceedings of the International Reliability Physics Symposium*, 2004, pp. 222–228.

[14] E. Takeda, R. Izawa, K. Umeda, and R. Nagai, "AC hot-carrier effects in scaled MOS devices," in *Proceedings of the International Reliability Physics Symposium*, 1991, pp. 118–122.

[15] J. H. Stathis, "Reliability limits for the gate insulator in CMOS technology," *IBM Journal of Research and Development*, vol. 46, no. 2.3, pp. 265–286, 2002.

[16] P. Larsson, "Power supply noise in future IC's: A crystal ball reading," in *Proceedings of the Conference on Custom Integrated Circuits*, 1999, pp. 467–474.

[17] V. J. Reddi, S. Kanev, W. Kim, S. Campanoni, M. D. Smith, G.-Y. Wei, and D. Brooks, "Voltage noise in production processors," *IEEE micro*, vol. 31, no. 1, pp. 20–28, 2011.

[18] V. Reddy, A. T. Krishnan, A. Marshall, J. Rodriguez, S. Natarajan, T. Rost, and S. Krishnan, "Impact of negative bias temperature instability on digital circuit reliability," *Microelectronics Reliability*, vol. 45, no. 1, pp. 31–38, Jan. 2005.

[19] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, Nov. 2005.

[20] H. Sharangpani and M. Barton, "Statistical analysis of floating point flaw in the Pentium processor," Intel Corporation, Tech. Rep., 1994.

[21] "AMD reports potential heat problem with some Opteron chips," *Information Week*, April 2006.

[22] W. W. Peterson, "On checking an adder," *IBM Journal of Research and Development*, vol. 2, pp. 166–168, 1958.

[23] P. Ndai, A. Agarwal, Q. Chen, and K. Roy, "A soft error monitor using switching current detection," in *Proceedings of the International Conference on Computer Design (ICCD)*.

[24] A. Narsale and M. Huang, "Variation-tolerant hierarchical voltage monitoring circuit for soft error detection," in *Proceedings of the Symposium on Quality of Electronic Design (ISQED)*.

[25] K. Constantinides, S. Plaza, J. Blome, B. Zhang, V. Bertacco, S. Mahlke, T. Austin, and M. Orshansky, "BulletProof: a defect-tolerant CMP switch architecture," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2006, pp. 5–16.

[26] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw, "RazorII: In situ error detection and correction for PVT and SER tolerance," *IEEE Journal of Solid State Circuits (JSSC)*, vol. 44, no. 1, pp. 32–48, Jan. 2009.

[27] J. Tschanz, K. Bowman, S. Walstra, M. Agostinelli, T. Karnik, and V. De, "Tunable replica circuits and adaptive voltage-frequency techniques for dynamic voltage, temperature, and aging variation tolerance," in *Proceedings of the Symposium on VLSI Circuits*.

[28] E. Böhl, T. Lindenkreuz, and R. Stephan, "The fail-stop controller AE11," in *Proceedings of the International Test Conference (ITC)*, 1997, pp. 567–577.

[29] T. Austin, "DIVA: A reliable substrate for deep submicron microarchitecture design," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 1999, pp. 196–207.

[30] I. A. Noufal and M. Nicolaidis, "A CAD framework for generating self-checking multipliers based on residue codes," in *Proceedings of the Conference on Design, Automation, and Test in Europe (DATE)*, 1999.

[31] M. de Kruijf and K. Sankaralingam, "Idempotent processor architecture," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2011, pp. 140–151.

[32] J. Chung, I. Lee, M. Sullivan, J. H. Ryoo, D. W. Kim, D. H. Yoon, L. Kaplan, and M. Erez, "Containment Domains: A scalable, efficient, and flexible resilience scheme for Exascale systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, Nov. 2012, pp. 1–11.

[33] J. Bartlett, J. Gray, and B. Horst, "Fault tolerance in Tandem computer systems," in *The Evolution of Fault-Tolerant Computing*. Springer Vienna, Jan. 1987, no. 1, pp. 55–76.

[34] M. Mueller, L. Alves, W. Fischer, M. Fair, and I. Modi, "RAS strategy for IBM S/390 G5 and G6," *IBM Journal of Research and Development*, vol. 43, no. 5.6, pp. 875–888, 1999.

[35] L. Alves, M. Fair, P. Meaney, C. Chen, W. Clarke, G. Wellwood, N. Weber, I. Modi, B. Tolan, and F. Freier, "RAS design for the IBM eServer z900," *IBM Journal of Research and Development*, vol. 46, no. 4.5, pp. 503–521, 2002.

[36] M. Fair, C. Conklin, S. Swaney, P. Meaney, W. Clarke, L. Alves, I. Modi, F. Freier, W. Fischer, and N. Weber, "Reliability, availability, and serviceability (RAS) of the IBM eServer z990," *IBM Journal of Research and Development*, vol. 48, no. 3.4, pp. 519–534, May 2004.

[37] D. Bernick, B. Bruckert, P. Vigna, D. Garcia, R. Jardine, J. Klecka, and J. Smullen, "NonStop advanced architecture," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, Jun. 2005, pp. 12–21.

[38] M. Hsiao, W. Carter, J. Thomas, and W. Stringfellow, "Reliability, Availability, and Serviceability of IBM Computer Systems: A Quarter Century of Progress," *IBM Journal of Research and Development*, vol. 25, no. 5, pp. 453–468, 1981.

[39] D. K. Pradhan, Ed., *Fault-tolerant computing: Theory and technique*. Prentice Hall Inc., Old Tappan, NJ, 1986, vol. I.

[40] W. Clarke, L. Alves, T. Dell, H. Elfering, J. Kubala, C. Lin, M. Mueller, and K. Werner, "IBM System z10 design for RAS," *IBM Journal of Research and Development*, vol. 53, no. 1, 2009.

[41] D. Lardner, "Babbage's calculating engine," *Edinburgh Review*, vol. 59, no. 120, pp. 263–327, 1834.

[42] 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)*, July 2012, pp. 177–180.

[43] J. Cortadella and J. Llaberia, "Evaluation of A+B=K conditions without carry propagation," *IEEE Transactions on Computers*, vol. 41, no. 11, 1992.

[44] W. L. Lynch and G. R. Lauterbach, "Low-latency memory indexing method and structure," U.S. Patent US5 754 819 A, May, 1998. [Online]. Available: http://www.google.com/patents/US5754819

[45] P. Kornerup and D. W. Matula, *Finite Precision Number Systems and Arithmetic*. Cambridge University Press, 2010.

[46] M. Hajkazem and A. Baniasadi, "A power-aware alternative for fault-tolerant multipliers."

[47] U. Sparmann and S. Reddy, "On the effectiveness of residue code checking for parallel two's complement multipliers," *IEEE Transactions on VLSI Systems*, vol. 4, no. 2, pp. 227–239, 1996.

[48] T. R. Rao and O. Garcia, "Cyclic and multiresidue codes for arithmetic operations," *IEEE Transactions on Information Theory*, vol. 17, no. 1, pp. 85–91, 1971.

[49] "Non-linear residue codes for robust public-key arithmetic," in *Fault Diagnosis and Tolerance in Cryptography*, no. 4236.

[50] Synopsys Inc., "Design Compiler I-2013.12-SP5-2."

[51] Taiwan Semiconductor Manufacturing Company, "40nm CMOS Standard Cell Library v120b," 2009.

[52] M. Sullivan and E. Swartzlander, Jr., "On separable error detection for addition," in *Proceedings of The AsiloMar Conference on Signals and Systems*, 2013, pp. 2181–2186.

[53] A. H. Syed, "Performance of different multipliers in the DesignWare building block IP," Synopsys Inc.

[54] A. Karatsuba, "Multiplication of multidigit numbers on automata," in *Soviet Physics Doklady*, vol. 7, 1963, pp. 595–596.

[55] B. Parhami, *Computer arithmetic: algorithms and hardware designs*, 2nd ed. Oxford University Press, 2010.

[56] H. L. Garner, "The residue number system," *IEEE Transactions on Electronic Computers*, vol. EC-8, pp. 140–147, 1959.

[57] N. S. Szabó and R. I. Tanaka, *Residue Arithmetic and its Applications to Computer Technology*. New York: McGraw-Hill, 1967.

[58] P. A. Mohan, *Residue number systems: algorithms and architectures*. Springer, 2002.

[59] A. Omondi and B. Premkumar, *Residue number systems*. World Scientific, 2007.

[60] C. Efstathiou, N. Moschopoulos, K. Tsoumanis, and K. Pekmestzi, "On the design of configurable modulo $2^n \pm 1$ residue generators," in *Proceedings of the Euromicro Conference on Digital System Design (DSD)*, 2012, pp. 50–56.

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

[62] C. Efstathiou, H. T. Vergos, G. Dimitrakopoulos, and D. Nikolos, "Efficient diminished-1 modulo $2^n + 1$ multipliers," in *IEEE Transactions on Computers*, vol. 54, 2005, pp. 491–496.

[63] B. Parhami, "On equivalences and fair comparisons among residue number systems with special moduli," in *Proceedings of The Asilomar Conference on Signals and Systems*, 2010.

[64] ——, "RNS representations with redundant residues," in *Proceedings of The AsiloMar Conference on Signals and Systems*, vol. 2.

[65] M. Abdallah and A. Skavantzos, "On the binary quadratic residue system with noncoprime moduli," *IEEE Transactions on Signal Processing*, vol. 45, no. 8.

[66] A. Skavantzos and M. Abdallah, "Implementation issues of the two-level residue number system with pairs of conjugate moduli," *IEEE Transactions on Signal Processing*, vol. 47, no. 3.

[67] E. E. Swartzlander, Jr. and A. G. Alexopoulos, "The sign/logarithm number system," *IEEE Transactions on Computers*, vol. 24, no. 12.

[68] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, vol. EC-11, no. 4.

[69] A. S. Fraenkel, "The use of index calculus and Mersenne primes for the design of a high-speed digital multiplier," *Journal of the ACM*, vol. 8, no. 1.

[70] E. Trichina and R. Korkikyan, "Multi fault laser attacks on protected CRT-RSA," in *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*.

[71] Z. Wang, M. Karpovsky, and A. Joshi, "Secure multipliers resilient to strong fault-injection attacks using multilinear arithmetic codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 6.

[72] D. Karaklajic, J. Schmidt, and I. Verbauwhede, "Hardware designer's guide to fault attacks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 12.

[73] K. Palem, "Energy aware computing through probabilistic switching: a study of limits," *IEEE Transactions on Computers*, vol. 54, no. 9.

[74] M. Neagu, G. Mois, and L. Miclea, "On-line error detection for tuning dynamic frequency scaling," in *Proceedings of the International Conference on Automation Quality and Testing Robotics (AQTR)*.