FARHAD: a Fault-Tolerant Power-Aware Hybrid Adder for Add Intensive Applications

Mohammad Hossein Hajkazemi[1], Amirali Baniasadi[1], Hossein Asadi[2]

[1] Department of Electrical and Computer Engineering, University of Victoria, Victoria, Canada

[2]Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

mhkazemi@ece.uvic.ca, amirali@ece.uvic.ca, asadi@sharif.edu

***Abstract*— This paper introduces an alternative Fault-Tolerant Power-Aware Hybrid Adder (or simply FARHAD). FARHAD is a highly power efficient protection solution against errors in application with high number of additions. FARHAD, similar to earlier studies, relies on performing add operations twice to detect errors. Unlike previous studies, FARHAD uses an aggressive adder to produce the initial outcome and a low-power adder to generate the second outcome, referred to as the checker. FARHAD uses checkpointing, a feature already available to high-performance processors, to recover from errors. FARHAD achieves the high energy-efficiency of time-redundant solutions and the high performance of resource-redundant adders. We evaluate FARHAD from power and performance points of view using a subset of SPEC'2K benchmark. Our evaluations show that FARHAD outperforms an alternative time-redundant solution by 20%. FARHAD reduces the power dissipation of an alternative resource-redundant adder by 40% while maintaining performance.**

***Keywords*— fault-tolerant adders; low-power design; power-aware reliability**

I. INTRODUCTION

Achieving high-performance while maintaining power within acceptable limits, continues to serve as a major challenge for IC designers. Higher performance may require more transistors, higher working frequency, lower supply voltage and smaller devices. All these trends make digital circuits more vulnerable to transient errors.

Recent studies show that combinational logics including ALU are highly susceptible to transient errors [1]. Similar to a memory cell which may be flipped by a transient voltage caused by neutron or alpha particles, any combinational logic node may confront a transient fault. This may result in a soft error as the fault propagates through the logic gates and gets latched by a sequential logic somewhere in the circuit [2].

Conventionally, there are two main approaches in building fault-tolerant designs. The first approach relies on modular redundancy [4]. This approach uses three equal modules and a majority voter to recover from possible errors. Modular redundancy comes with significant power and area overhead. The second approach relies on temporal redundancy, i.e., it takes three executions of a task to prepare fault-free results. Although this approach has less area and hardware complexity, it suffers from performance overhead.

The adder is one of the most frequently used ALU components. Many CPU operations including addition, subtraction, comparison and address calculation use the adder.

Modern applications come with different instruction distribution. While addition constitutes a big portion of instruction in some applications, it accounts for a small share in others. Fig. 1 shows the frequency of ADD operations in a subset of SPEC'2k applications as measured by SimpleScalar 3.0 toolset [3]. In addition to integer and floating-point instructions we also include operations relying on additions (e.g., address calculations).

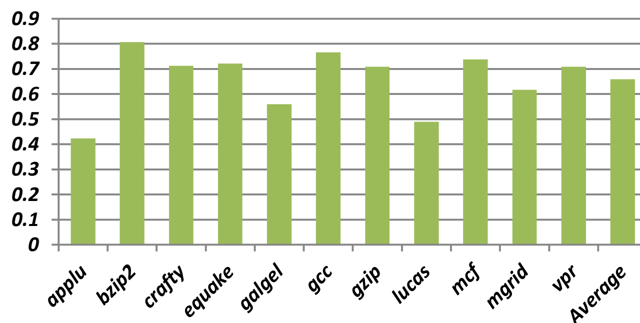

Figure 1. Frequency of add operations in a subset of SPEC'2K application.

As we show in Fig. 1, in some applications, including bzip2, gcc and mcf, addition accounts for a significant share of operations. Thus, developing a low-power, fast and fault-tolerant adder can impact the ALU's power, performance and reliability significantly in this group of applications.

With aggressive technology scaling, radiation-induced soft errors can lead to Multiple Event Transients (METs) in combinational logic and Multiple Bit Upsets (MBUs) in sequential logic [13-16]. An energetic particle hitting a combinational logic can affect two or multiple physically adjacent logic gates leading to multiple transients, also called METs. If an energetic particle hits multiple flip-flops, it can invert multiple values, leading to MBUs. Previous protection techniques consider the effect of Single Event Transients (SETs) and Single Event Upsets (SEUs) in combinational and sequential logic [11, 17]. In this work, we present a fault-tolerant architecture to protect adders in the presence of METs and MBUs with minimal power and area overheads.

Previous studies on fault-tolerant adders have introduced many designs including TMR (i.e. triple modular redundancy) [4], QTRA [10], TEPS [11], and lazy error detection [18].

Some of these adders achieve reliability at the expense of power and area overhead (e.g., TMR and lazy error detection). Such solutions employ high performance and power hungry units to reproduce execution units' outputs. This aggressive approach is inefficient from the energy point of view as errors occur rarely making calculating the second output often unnecessary. One way to reduce the associated overhead is to use a simple, slower and hence more power efficient unit to reproduce the execution unit output. This approach can provide the same level of reliability while imposing much less power overhead.

To this end, in this work we address both power and performance issues by introducing a **Fa**ult-tole**r**ant Power-Aware **H**ybrid **Ad**der (FARHAD). FARHAD uses two adders to produce the add operation result twice. Both outcomes are compared and a mismatch detects an error. This is achieved by using a power-efficient adder (i.e., ripple carry adder) alongside a conventional fast adder (i.e., carry look-ahead adder). The fast adder calculates the outcome which will be immediately used by the processor. The slow adder calculates the outcome and has it compared to the outcome produced earlier. The processor ignores matching outcomes without interrupting program flow. A mismatch triggers a recovery process.

Our two underlying adders come with different latencies. As the ripple carry adder's (RCA) input carry should propagate through full adders to produce the output carry, RCA is slower than carry look-ahead adder. In this work we address timing complexities and show that the associated performance cost is negligible.

It should be noted that we use RCA and carry look-ahead (CLA) adder as examples of low-power, but slow and fast but power hungry adders. FARHAD, however, can be applied to any pair of adders with characteristics similar to ripple carry and CLA adders.

In this study we take into account the power, area and performance overhead associated with FARHAD and compare to conventional methods. In summary our contributions are:

- We introduce FARHAD as an alternative to conventional fault-tolerant adders. Our proposed method builds on the observation that faults happen rarely, making aggressive reliability solutions too costly.
- Motivated by the above observation, we use a small RCA in parallel to a CLA adder to detect possible errors. We show that by using checkpointing (already available to high-performance processors), it is possible to recover from errors without compromising performance [5].
- We study and address the impact of infrequent consecutive additions on FARHAD.
- We evaluate FARHAD from different aspects and show that FARHAD comes with higher reliability but dissipates 59% less power compared to TMR . We also show that FARHAD outperforms QTRA by 20%.

The rest of the paper is organized as follows. . In Section II we review the related works. In Section III we review FARHAD. In Section IV we report methodology. In Section V we present the results. In Section VI we offer our conclusions.

II. Related Work

Using multiple copies of the target module is a simple physically redundant solution for both detecting and correcting errors but requires extra logic and considerable overhead [7]. One way to reduce this overhead is to reuse the module (time redundancy) for error detection and correction. Re-computing with Swapped Operands (RESWO) [8] and Re-computing with Rotated Operands (RERO) [9] are two examples. They come with considerable performance overhead.

In RESWO, operands are applied to the module and the result is saved. Then, the operands upper and lower haves are swapped and the computation is done again. Finally the results are swapped back and compared with the primary result to detect any probable errors. The same technique is used in RERO. The difference in RERO is that the operands are not swapped; they are rotated by a specific number of bits instead.

Re-computing with Duplication with Comparison (REDWC) [23] improves performance in RESWO as it uses two half sized replicas in two consecutive iterations. In the first iteration the first half of the operand is fed to the replicas. In the second iteration the replicas do the computations for the second half. In both iterations the results are compared.

By adding another replica and dividing the operands to three portions, HPTR extends REDWC's error detection capability to error correction ability [24]. HPTR suffers from two problems. First, often operand sizes are not dividable by three, so the module should be padded, which results in more resources, area and more power dissipation. Second, not all resources including the multiplexer control signals are utilized in all iterations.

QTRA overcomes the utilization and resource overhead problem of HPTR [10] as operands are divided to four parts. Therefore four iterations are needed for the operation to complete. While each iteration can be done in a shorter time, the extra number of iterations results in performance loss.

An alternative approach to detecting errors in arithmetic operations (e.g., multiplication) is residue codes [7, 20]. This method utilizes smaller arithmetic units as the replica to detect errors. On the negative side there is a chance that both checker results and the primary results match even in the presence of an erroneous primary result [7]. One way to address this is to increase the modulus operands size, which results in more hardware and power dissipation.

TEPS (Transient Error Protection Utilizing Sub-word Parallelism) [11] and Lizard [12] are two recent techniques, which take advantage of narrow-width operands to protect the ALU from transient and permanent errors respectively. While TEPS comes with little area and power dissipation overhead, it imposes significant performance penalty. Lizard uses an

approach similar to TEPS. In lizard, the 32-bit adder is divided to four 8-bit adders. As most additions require less than eight bits, three sub-adders can adequately correct any occurring fault. Lizard comes with time overhead, as it requires two cycles for non-narrow-width operations.

Two other studies [18-19] use lazy error detection method and take advantage of processor speculation mechanism. In these methods, the result of the add operation is tested by a *checker*. Error detection is done between the execution time and commit time.

Our work is different from lazy error detection in several ways. First, FARHAD and lazy error detection employ different logic circuits. While lazy error detection utilizes a checker cell, FARHAD uses a simple full adder. We simulated the overhead associated with both solutions. Our measurements show that FARHAD comes with 12.5% less power overhead and 11% less area overhead compared to lazy detection.

Furthermore, in this work we provide deeper and more accurate analysis of performance and timing issues. We evaluate FARHAD under more advanced technology and investigate scenarios that may slow down CPU's performance including consecutive additions. We also compare FARHAD to alternative fault-tolerant solutions and report power, performance, delay, area overhead and reliability. Moreover, and in addition to the quantitative results, we offer qualitative analysis of FARHAD's vulnerability. For qualitative analysis we consider various conditions under which SEU, SET, MBU and MET could occur.

In addition to the differences mentioned above, FARHAD can also be potentially used to recover from permanent adder errors. This is not available to lazy detection.

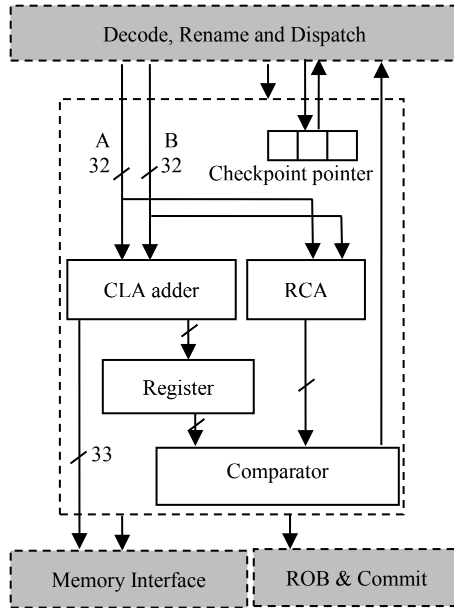

Figure 2. FARHAD architecture.

III. FARHAD

A. Overview

Fig. 2 shows FARHAD's architecture. FARHAD uses CLA adder as a fast but power hungry adder. In order to detect possible errors FARHAD uses an RCA to execute the operation twice. Performing an operation twice and comparing the two independent outcomes enhances reliability significantly.

We refer to the two outcomes as the *actual outcome* (produced by CLA adder) and the *checker* (produced by RCA). While the actual outcome has to be produced fast to maintain performance, the checker can be decided slowly (and hence more power efficiently). Fig. 3 shows the average power dissipation per addition for the CLA adder and the RCA in a subset of SPEC'2K benchmark. As Fig. 3 shows the power dissipation of the CLA adder is seven times higher than RCA.

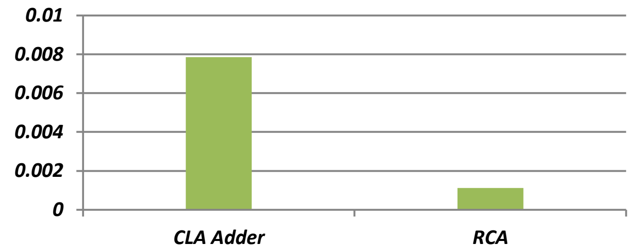

Figure 3. Power dissipation of CLA adder vs. Ripple Carry adder (mW).

In order to maintain reliability, the actual outcome has to be compared to the checker prior to the instruction's retirement. Upon a mismatch (which occurs rarely), the operation must be redone and the processor must recover from the mistake. Current modern processors come with effective recovery and checkpointing mechanisms which we employ at a low extra expense (i.e., area cost of a 3-bit register). The slower checker has to be produced before the instruction commits to avoid performance penalty. In our system, the checker is produced early enough and therefore there is no performance penalty associated with accurate outcomes (see Fig. 2).

FARHAD comes with the overhead of four extra components, including an *RCA* adder, a comparator, a 3-bit checkpoint pointer and a 32-bit register.

B. Microarchitecture

- ***RCA, CLA adder and Comparator.*** The two results generated by the two adders are compared using a comparator. The comparator result decides whether the associated instruction can retire. As Fig. 2 shows, CLA adder result is bypassed to the next stage to assure that CPU is not stalled while error detection is in process. More details are explained in the sub-section C.

- ***Checkpoint Pointer.*** FARHAD relies on the checkpointing feature available to modern high-performance processors [5]. Conventional processors use checkpoints to recover from branch mispredictions. Checkpoint pointer is in charge of saving the checkpoint for the add instruction being checked. This checkpoint is the same checkpoint associated with the most recent

branch instruction. We use eight checkpoints requiring a 3-bit checkpoint pointer. Upon an error, FARHAD takes advantage of this feature, flushing the pipeline back to the most recent branch instruction. This comes with the cost of re-executing the accurately executed instructions between the branch and the add operation. The cost is negligible as such errors occur rarely.

- ***Register.*** As RCA is slower than the CLA adder, the CLA adder result is saved so it can be compared against the checker. A 32-bit register is responsible to latch the CLA result (more details in the sub-section C).

C. Timing

- **CLA adder and RCA latency.** The actual outcome and the checker are not produced at the same time. To make comparison possible, FARHAD requires an extra register. As FARHAD uses a bypass path, the CPU does not need to wait for the fault detection system to verify the correctness of the operation. While the fault detection system is working, a copy of the CLA adder result is sent to the next pipeline stage. Accurately produced results, therefore , face no additional penalty.

- **Issue to commit time interval**. It is also essential that the fault detection system produces the checker early enough to compare against the actual outcome before the instruction leaves the pipeline. In a typical processor, as well as the one used in this work, there is a *write-back* stage between issue and commit stages, taking at least two cycles. Moreover, instructions reaching the commit stage sometimes have to wait for additional cycles so earlier instructions can commit first. As a result, the checker is produced in time to avoid any additional stalls. Fig. 4 shows the timing diagram of FARHAD.

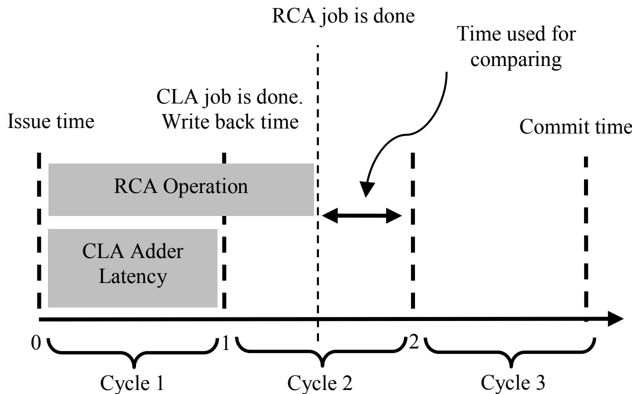

Figure 4. FARHAD's timing diagram.

As Fig. 4 illustrates, while CLA adder operation is done in one cycle, the RCA needs an additional cycle. As RCA delay is less than two cycles, sufficient time remains for comparing the results.

- **Consecutive additions.** One of the challenges FARHAD faces is dealing with consecutive additions. RCA cannot execute a new additions when busy performing the previous addition. To address this issue, FARHAD stalls dispatching instructions to ALU upon seeing an immediate second addition. FARHAD keeps the second addition in the ready list and issues it in the next cycle. Our study shows that consecutive additions happen infrequently. According to our observations, just four of the 11 studied benchmarks in this paper have consecutive additions [i.e., *bzip2*, *crafty*, *gcc* and *vpr*]. The performance penalty associated with the required stalls is less than 0.1%.

IV. Methodology

We use SimpleScalar 3.0 toolset with ALPHA configuration to model our processor [3]. We use a subset of SPECK'2K benchmarks compiled for ALPHA [21]. Table 1 reports the processor configuration used in this study. We run benchmarks for one billion instructions.

TABLE I. SIMULATED PROCESSOR CONFIGURATION.

Processor Core	
Fetch Queue/LSQ/RUU Size Decode/Issue/Commit Width Int. & FP Functional Unit	4/32/64 Instruction 4/4/4Instruction per cycle 4 ALU, 1 mult/div
Memory Hierarchy	
Memory Latency L1 D-cache, L1 I-cache L1 hit Latency Unified L2 cache L2 hit Latency	First_chuck: 512, Inter_chunck: 4 16 KB, 4-way 3 cycles 1 MB, 8-way 32 cycles
Branch Prediction	
Combinational Bimodal 2level BTB Misprediction penalty	meta size: 1024 2048 8 bits history and 1024 array 512, 4-way 3 cycles

All methods studied in this paper are implemented using Verilog. We use Synopsys tool chain to synthesize the designs and measure their delay, area and power dissipation [22]. In order to have the best efficiency we use Synopsys DesignWare Building Block IPs to implement adders [25]. While for CLA adder we use "cla" implementation, for RCA we use "rpl" implementation.

All designs are compiled by Design Power Compiler with medium mapping and area effort using TSMC 90nm CMOS process technology. In order to measure power dissipation, we feed all methods operands obtained from the application benchmarks using SimpleScalar.

V. Evaluation

In this section we evaluate FARHAD from power, performance, area and vulnerability points of view. We compare FARHAD with two conventional methods i.e., TMR and QTRA. To provide better understanding, we also compare to an alternative processor, which uses two CLA adders in parallel referred to as *Dual-CLA*.

A. Power reduction per addition

In Fig. 5 we report the power reduction for FARHAD relative to other studied methods. As reported, on average, FARHAD comes with 41% and 60% power reduction compared to Dual-CLA and TMR respectively. Moreover FARHAD dissipates 44% more power compared to QTRA.

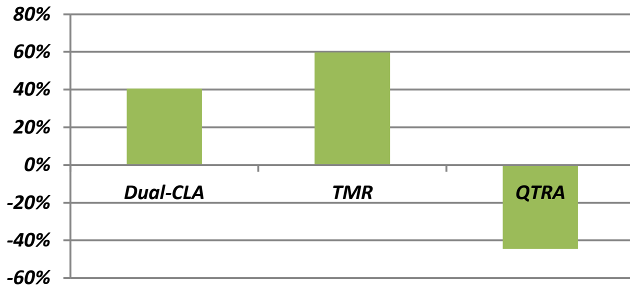

Figure 5. Power reduction of FARHAD relative to Dual-CLA, TMR and QTRA.

QTRA employs fewer resources in comparison to FARHAD and therefore dissipates less power. This, however, is achieved at the expense of 20% performance loss compared to FARHAD. This is due to the fact that QTRA needs four cycles to execute each addition while FARHAD needs only one cycle.

Dual-CLA employs more resources compared to FARHAD. While FARHAD uses an RCA to generate the checker, Dual-CLA employs a CLA adder to do so. As already reported in Fig. 3, RCA dissipates much less power compared to CLA adder. As a result Dual-CLA comes with more power dissipation. Similarly, because TMR utilizes more resources (i.e., an extra CLA adder compared to Dual-CLA) it has the highest power dissipation among all methods.

B. Latency

In Fig. 6 we report relative latency for the adders studied in this paper. Latency is important as it can impact CPU's clock frequency. As almost all conventional methods add extra resources to ALU's critical path, they can increase ALU latency if the additional delay requires an extra cycle.

Figure 6. Relative Delay of FARHAD, TMR and Dual-CLA and QTRA compared to CLA adder.

As Fig. 6 reports, both TMR and QTRA have the least latency. FARHAD is the slowest adder. TMR uses three parallel CLA adders which are connected to a voter serially. As voter delay is equal to two connected gates i.e., an XOR gate and OR gate, TMR's latency is slightly more than a CLA adder. QTRA uses a small size adder but requires extra resources including a multiplexer and a voter making it as slow as TMR.

FARHAD includes a carry propagation chain and a voter, and therefore has the longest delay among all the methods. But as Fig. 2 depicts, the CLA adder result is bypassed to the next stage and error detection is done while the result is being used. Dual-CLA uses a bypass path to the next stage too. Therefore both FARHAD and Dual-CLA's effective latencies are close to a CLA adder and slightly more than TMR and QTRA latencies. As a result, in our simulations we consider one clock cycle for FARHAD, Dual-CLA and TMR. Additionally as QTRA needs four iterations, and its latency is almost equal to a CLA adder, QTRA's latency is four clock cycles.

C. Performance

Fig. 7 reports the performance loss for processors using FARHAD and QTRA relative to an unprotected processor. Since the delay overheard of TMR is negligible, we assume that its performance is equal to an unprotected CPU. Therefore we do not include it in Fig. 7..

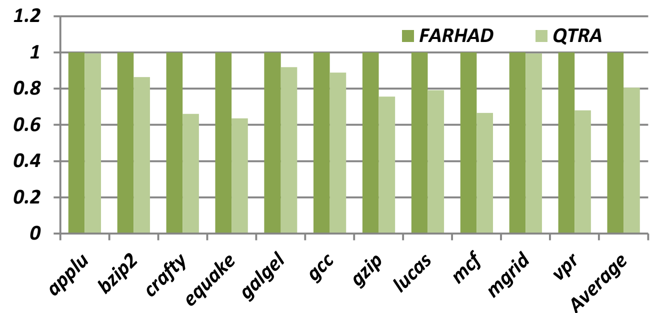

Figure 7. Relative performance of FARHAD and QTRA compared to an unprotected CPU.

As we explained in Section III, there are only four benchmarks including *bzip2*, *crafty*, *gcc* and *vpr* for which consecutive additions occur. Our observations show that for these benchmarks stalling the ALU can result in negligible performance loss (less than 0.1%).

QTRA needs three more cycles to perform additions compared to FARHAD. Therefore, QTRA shows a significant average performance loss of 20%. As Fig. 7 reports, *applu, galgel and mgrid* have the least performance loss. These benchmarks have the lowest number of additions. On the other hand, *equake*, *crafty* and *vpr* have the highest performance loss under QTRA. They have the most number of additions among their operations.

D. Area

Almost all fault-tolerant adders come with area overhead. Fig. 8 reports the area overhead for methods studied in this work compared to a regular CLA adder.

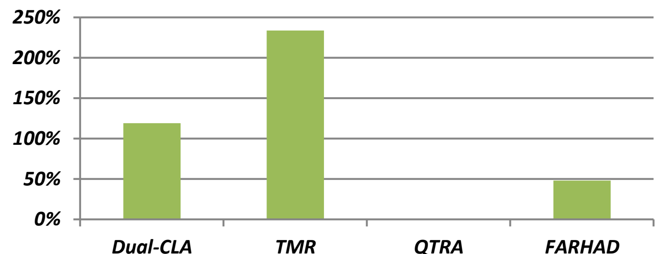

Figure 8. Area overhead of FARHAD, Dual-CLA and conventional methods compared to CLA.

As Fig. 8 depicts, QTRA has the lowest area overhead. QTRA imposes extra resources including three multiplexers, two voters and a register; at the same time it utilizes three small adders (i.e. 8-bit) instead of a full size adder (i.e., 32-bit). Therefore its overall overhead is about zero. FARHAD on average shows 48% area overhead, which is lower than TMR and Dual-CLA. The area overhead of Dual-CLA is about 120%. TMR employs three CLA adders and therefore has the worst area overhead (i.e., 233%).

E. Vulnerability analysis

In this section we present vulnerability analysis from both qualitative and quantitative views

Qualitative. In our qualitative analysis we study different scenarios in which SEU, SET, MBU, and MET happen. Since the probability of having two simultaneous particles hitting combinational or sequential logic is extremely low, in the rest of this analysis, we ignore the effect of having two particles hitting a single device at the same time. However, in our analysis, the effect of METs and MBUs caused by a single event (or an energetic particle) is taken into account.

- **TMR**. SETs and METs may happen in any of TMR's components including the adders and voter. The voter can correct errors occurring in adders. An error happening in the voter, however, can propagate to the next logic block and may result in a system failure.
- **QTRA**. QTRA has several components prone to SETs, METs and SEUs including multiplexers, adders, voter, and a 1-bit register. As Fig. 9 depicts, the multiplexers responsible to select the operand bit slice (4:1 multiplexers) use the same select nets. Therefore any SET or MET in these nets and multiplexer logic can result in feeding the adders wrong but similar inputs. As adders receive the same inputs, they produce the same results. Hence, the voter detects no soft errors and propagates wrong data to the next stage. The same happens for the 2:1 multiplexer if any SET or MET occurs. As the register output is connected to the 2:1 multiplexer input, any SEU in registers passing through the adders and voter propagates to the next stage without being corrected.

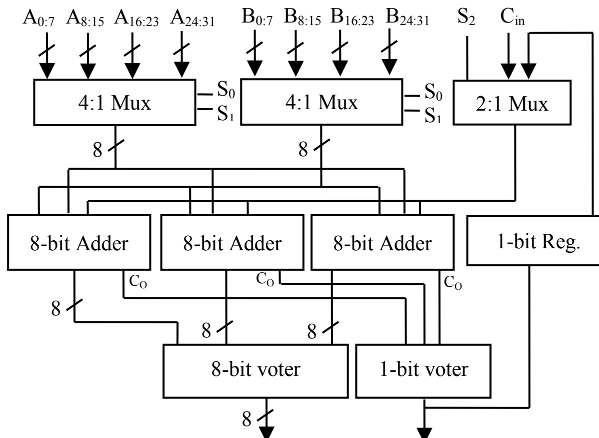

Figure 9. 32-bit QTRA.

The voter corrects any SET or MET initiated in the adders blocking the error from reaching the next stages. However, SETs and METs occurring in the voter can transmit a wrong data to the next logic path.

- **FARHAD**. FARHAD has components prone to soft errors including CLA adder, RCA, register, comparator and checkpoint pointer. The comparator identifies any SETs or METs occurring in CLA adder and RCA. SEUs and MBUs in registers are detectable by the comparator. The comparator detects faults in any of the above three components.

  The effect of SETs and METs in the comparator can be studied under two different conditions: 1) CLA adder and RCA produce similar results but the comparator mistakenly reports a mismatch. This results in the processor flushing the pipeline back to the most recent branch instruction unnecessarily. Consequently we pay a negligible performance penalty. Unlike TMR and QTRA, the produced soft error does not propagate to the next stage. 2) RCA and CLA adders produce different results but the comparator mistakenly reports a match due an SET or MET. Therefore, erroneous data propagates to the next stage. The second scenario is extremely unlikely, as it is very rare to have two particles hitting a single device at the same time. Moreover, it is very unlikely for two SETs (i.e., the one in the adder and the one in the comparator) to occur at the logic locations associated with the same bit.

  As we save the checkpoint associated with the last branch instruction in the checkpoint pointer, in case of any SEUs and MBUs, the checkpoint associated with an earlier branch instruction is saved in checkpoint pointer. So even if an SET or MET happens to other components and are detected by the comparator, no failure occurs: the processor flushes instructions and loses small performance.

In the light of the above observations, QTRA is the most vulnerable method. TMR comes second. Dual-CLA, while having an architecture similar to FARHAD, has higher number of gates and therefore is more vulnerable. We conclude from our qualitative analysis that FARHAD has the lowest vulnerability among all studied methods.

Quantitative. Reliability shows how probable it is for a component to survive till time t [7]. Reliability depends on a parameter called failure rate, λ, which is defined as follows:

$$R(t) = e^{-\lambda t} \quad (1)$$

Failure rate is expressed using equation (2) where π_P, π_Q, π_L, π_T and π_E are the pin, quality, learning, temperature and environment factors, respectively. C_1 and C_2 are the complexity factors which depend on the number of gates and pins used in the device. More details can be found in [12].

$$\lambda = (C_1\pi_T + C_2\pi_T)\pi_P\pi_Q\pi_L \quad \text{Failures}/10^6 \text{ Hours} \quad (2)$$

In order to estimate reliability we assume that beside C_1 and C_2 all other factors are the same for all methods. Fig. 10 shows reliability estimate for FARHAD along with other methods using (1). While the Y axis shows reliability, the X axis shows time in 10^6 hours.

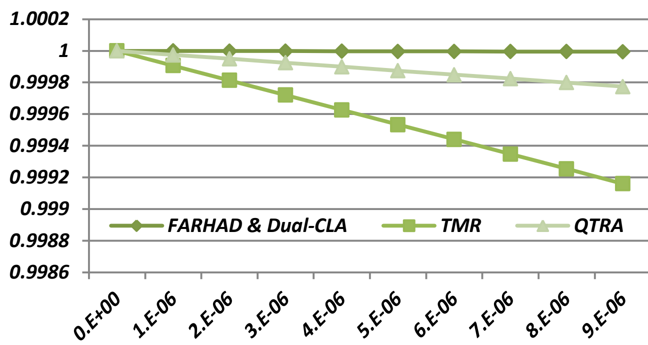

Figure 10. Reliability estimation.

As Fig. 10 reports, FARHAD and Dual-CLA have the highest reliability. QTRA and TMR come next. Since the complexity factors of FARHAD and Dual-CLA are similar, their failure rates and thus reliabilities are equal.

VI. CONCLUSION

In this work we introduced FARHAD as a low-power fault-tolerant adder to protect applications with high number of additions. FARHAD uses dual modular redundancy for error detection. Unlike previous methods, which use the same adder as the replica, FARHAD uses an energy-efficient adder.

For error recovery, FARHAD relies on checkpointing, also used by high-performance processors to recover from branch misprediction. In case of any errors, FARHAD flushes the pipeline back to the most recent branch instruction.

As the error detection process of FARHAD is performed in parallel to execution, FARHAD comes with no performance penalty. FARHAD reduces power overhead significantly compared to modular redundant solutions.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions. This work was supported by the Natural Sciences and Engineering Research Council of Canada, Discovery Grants Program and Canada Foundation for Innovation, New Opportunities Fund.

REFERENCES

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

[2] F. Wang, Y. Xie, R. Rajaraman, B. Vaidyanathan, "Soft error rate analysis for combinational logic using an accurate electrical masking model," *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on* , pp.165-170, 6-10 Jan. 2007

[3] D. Burger, T.M. Austin, S. Bennett, Evaluating Future Microprocessors: The Simple Scalar Tool Set. Technical Report, University of Wisconsin-Madison, New York, 1996

[4] B.W. Johnson, Design and Analysis of Fault-Tolerant Digital Systems. Reading, MA: Addison-Wesley Publishing Company, 1989

[5] A. Moshovos, "Checkpointing alternatives for high-performance, power-aware processors," *Low Power Electronics and Design, 2003. ISLPED '03. Proceedings of the 2003 International Symposium on* , pp. 318- 321, 25-27 Aug. 2003

[6] I. Koren and C.M. Krishna, Fault-Tolerant Systems, Morgan-Kaufman, 2007

[7] D. Sorin, Fault Tolerant Computer Architecture, Morgan & Claypool Publishers, 2009

[8] B.W. Johnson, "Fault-tolerant microprocessor-based systems," *Micro, IEEE* , vol.4, no.6, pp.6-21, Dec. 1984

[9] J. Li, E.E. Swartzlander, "Concurrent error detection in ALUs by recomputing with rotated operands," *Defect and Fault Tolerance in VLSI Systems, 1992. Proceedings., 1992 IEEE International Workshop on* , pp.109-116, 4-6 Nov 1992

[10] W.J. Townsend, J.A. Abraham, E.E. Swartzlander, "Quadruple time redundancy adders [error correcting adder]," *Defect and Fault Tolerance in VLSI Systems, 2003. Proceedings. 18th IEEE International Symposium on* , pp. 250- 256, 3-5 Nov. 2003

[11] S. Hong, S. Kim, "TEPS: Transient error protection utilizing sub-word parallelism," *VLSI, 2009. ISVLSI '09. IEEE Computer Society Annual Symposium on* , pp.286-291, 13-15 May 2009

[12] S. Hong, S. Kim, "Lizard: Energy-efficient hard fault detection, diagnosis and isolation in the ALU," *Computer Design (ICCD), 2010 IEEE International Conference on* , pp.342-349, 3-6 Oct. 2010

[13] D. Rossi, M. Omana, F. Toma, C. Metra, "Multiple transient faults in logic: an issue for next generation ICs?," *Defect and Fault Tolerance in VLSI Systems, 2005. DFT 2005. 20th IEEE International Symposium on* , pp. 352- 360, 3-5 Oct. 2005.

[14] C. Rusu, A. Bougerol, L. Anghel, C. Weulerse, N. Buard, S. Benhammadi, N. Renaud, G. Hubert, F. Wrobel, T Carriere, R Gaillard, "Multiple Event Transient Induced by Nuclear Reactions in CMOS Logic Cells," *On-Line Testing Symposium, 2007. IOLTS 07. 13th IEEE International* , pp.137-145, 8-11 July 2007

[15] J.A. Maestro, P. Reviriego, "Study of the effects of MBUs on the reliability of a 150 nm SRAM device," *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE* , pp.930-935, 8-13 June 2008

[16] D. Falguere, S. Petit, "A statistical method to extract MBU without scrambling information," *Nuclear Science, IEEE Transactions on* , vol.54, no.4, pp.920-923, Aug. 2007

[17] V. Sridharan, H. Asadi, M.B. Tahoori, D. Kaeli, "Reducing data cache susceptibility to soft errors," *Dependable and Secure Computing, IEEE Transactions on* , vol.3, no.4, pp.353-364, Oct.-Dec. 2006

[18] M. Yilmaz, A. Meixner, S. Ozev, D.J. Sorin, "Lazy error detection for microprocessor functional units," *Defect and Fault-Tolerance in VLSI Systems, 2007. DFT '07. 22nd IEEE International Symposium on* , pp.361-369, 26-28 Sept. 2007

[19] A. Meixner, M.E. Bauer, D.J. Sorin, "Argus: Low-cost, comprehensive error detection in simple cores," *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on* , pp.210-222, 1-5 Dec. 2007

[20] I. Lee, M. Basoglu, M. Sullivan, D. Hyun Yoon, L. Kaplan, M. Erez, Survey of error and fault detection mechanism, University of Texas, April 2011

[21] Standard Performance Evaluation Corporation, http://www.spec.org

[22] Synopsys.com, http://www.synopsys.com/home.aspx

[23] B.W. Johnson, J.H. Aylor, H.H. Hana, "Efficient use of time and hardware redundancy for concurrent error detection in a 32-bit VLSI adder," *Solid-State Circuits, IEEE Journal of* , vol.23, no.1, pp.208-215, Feb. 1988

[24] S.A. Al-Arian, M.B. Gumusel, "HPTR: Hardware partition in time redundancy technique for fault tolerance," *Southeastcon '92, Proceedings., IEEE* , pp.630-633 vol.2, 12-15 Apr 1992

[25] http://www.synopsys.com/dw/buildingblock.php