Towards an Algorithm-based Approach for Soft Error Tolerance using Interval Arithmetic

Larry Tang, Varun Kumar, Matthew Ngaw, Siddharth Singh, Devdutt Nadkarni,
Lohith Tummala, Ken Mai, and Franz Franchetti
Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA
{lawrenct, varunk2, mngaw, ssingh4, dnadkarn, lctummal, kenmai, franzf}@andrew.cmu.edu

***Abstract*—Soft errors pose a critical reliability concern for modern electronics both in space and terrestrial applications. The traditional hardware redundancy approaches such as triple modular redundancy or dual modular redundancy introduce significant overheads, especially in large modern SoCs which must consider tradeoffs between power, performance, area, and reliability requirements. We propose a new approach for hardware redundancy based on the ideas behind algorithm-based fault tolerance (ABFT). Rather than replicating entire logic modules in the design, we propose using floating-point interval arithmetic to realize redundancy in computational datapaths. Error detection is then performed by leveraging the guarantees provided by interval arithmetic and forward error analysis of the specific algorithm. We demonstrate the technique for protection of a hardware FFT datapath and a systolic array. To evaluate the approach, a silicon test chip is fabricated in a 28nm process and post-PnR/simulation results demonstrate up to 2-3 times savings in area.**

***Index Terms*—Triple-Modular Redundancy; interval arithmetic; radiation-hardening; fault tolerance**

I. Introduction

Modern semiconductor technology scaling has led to an increased vulnerability in electronic systems to radiation-induced soft errors [1], [2]. High energy particles, such as heavy ions, protons, or neutrons, can cause a temporary change of state known as a soft error upset (SEU) upon striking and interacting with the integrated circuit. Protecting against soft errors is crucial for circuits operating in safety-critical space/upper atmosphere applications where radiation environments are harsh. Even large scale terrestrial systems, such as hyperscale data centers deploying integrated circuits (ICs) at high volume, are facing increasing reliability concerns due to effects from terrestrial radiation [3]–[5].

One of the most common design techniques to mitigate the effects of soft errors is to incorporate system-level hardware redundancy for error detection and correction. Using triple modular redundancy (TMR) or dual modular redundancy (DMR) schemes, the designer triples/doubles modules within the design and includes additional logic for error correction or detection. Both of these approaches can be leveraged at varying levels of granularity (e.g. at the register or functional unit-level) at around 2-4X area and power overheads. However the cost of TMR/DMR-based approaches is significant and becomes overwhelming, especially as power constraints and energy efficiency drive the performance scaling of new computing systems. In designing large modern SoCs, it has become critical to analyze tradeoffs between reliability, power, performance, and area so that only critical components are actually replicated [6], [7]. Hence strategies to reduce the overheads while still providing tolerance against soft errors is increasingly important.

Another method to protect against soft errors is to refactor computations such that they take advantage of properties of the algorithm itself for error detection or correction, an approach typically referred to as algorithm-based fault tolerance (ABFT) [8]. Exploiting algorithmic properties, such as transform invariants or matrix checksums, provides opportunities to detect and correct errors at low overhead without necessarily relying on system-level redundancy techniques. Inspired by the principles of ABFT, in this work we propose a new hardware redundancy approach for soft error tolerance in datapath modules. Our insight is that redundancy can be realized through floating-point interval arithmetic where lower precision interval arithmetic hardware reduces overheads compared to fully replicating modules. Error detection then relies on the guarantees provided by interval arithmetic and forward error analysis of the specific computation.

Contributions. This paper makes the following contributions:

- A novel algorithm-based approach using interval arithmetic and forward error analysis for soft error tolerance.
- The design and implementation of the approach for an FFT datapath and systolic array on a test chip fabricated in a 28nm process.
- A quantitative evaluation of the approach against traditional hardware redundancy-based techniques.

We begin with a discussion of related work and relevant background. Section IV details the idea of using an algorithm-based interval arithmetic approach for soft error tolerance. Section V outlines the design of a silicon prototype and Section VI presents measured results. Finally we conclude with a discussion and avenues for future work.

II. Related Work

In this section we discuss some of the common approaches for soft error protection.

A. Redundancy-Based Techniques

Hardware redundancy-based schemes have been widely employed in fault tolerant systems. In Triple Modular Redundancy (TMR), the module to be protected is replicated three times and a majority voter circuit can then detect and correct errors assuming a single event upset fault model. Similarly, Dual Modular Redudancy (DMR) duplicates the module but only guarantees error detection using comparator logic between the two outputs. Both schemes are relatively intuitive and generalizable, but incur significant power and area overheads due to entire module replication.

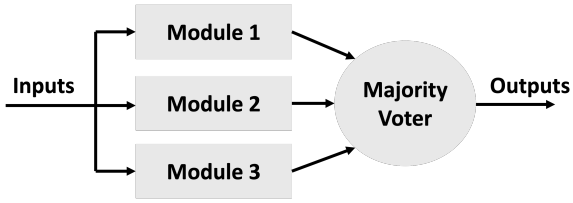

Fig. 1. TMR technique

As modern systems attempt to continue scaling performance the key constraint lies in power consumption and energy efficiency, making it challenging to realize the high overhead of hardware redundancy-based schemes. Prior work has investigated methods for replicating only critical portions of the hardware [6], [7] as well as different low-overhead algorithmic approaches.

B. Algorithm-Based Fault Tolerance

The idea of exploiting algorithm specific properties for fault tolerance was first demonstrated for matrix operations [8]. The key idea behind ABFT is to encode the data such that the encoding relation still holds after the algorithm is applied, enabling error detection or correction. The algorithm can then be modified to operate on the encoded data and produce encoded outputs, and finally the computation is distributed across processing units such that faults affect only certain parts of the data. Error detection and correction can be performed based on the outputs of the original computation and the encoded outputs.

The approach has been extended to a number of other domains such as digital signal processing [9], [10], machine learning [11], [12], and HPC scientific computing applications [13]–[16]. Properties of signal processing transforms make them particularly well suited for ABFT techniques and have been applied to digital filters [17], [18] and fast Fourier Transforms [19]–[21]. In linear algebra subroutines based on matrix multiplication, row and column checksums of the matrices are often used to encode the matrices [8], [22]. ABFT has also been extended to the context of HPC parallel distributed machines with low performance overheads for error detection and correction [13].

III. Background: Interval Arithmetic

Interval arithmetic [23] is an approach for performing sound floating-point arithmetic as it provides the guarantee that the resulting interval must contain the real number that is the result of the mathematical expression. Operations on real numbers represented in floating point instead take place on intervals with floating point boundaries containing the real number. Concretely, an interval $\hat{x} = [x_l, x_u]$ consists of floating point bounds that guarantee the exact number is contained within the interval $\hat{x}$. Interval arithmetic operations for addition, subtraction, and multiplication are then defined as follows:

$$\hat{x} + \hat{y} = [\mathrm{RD}(x_l + y_l),\ \mathrm{RU}(x_u + y_u)] \tag{1}$$

$$\hat{x} - \hat{y} = [\mathrm{RD}(x_l - y_u),\ \mathrm{RU}(x_u - y_l)] \tag{2}$$

$$\begin{aligned}\hat{x} \cdot \hat{y} = [&\min(\mathrm{RD}(x_l \cdot y_l),\ \mathrm{RD}(x_l \cdot y_u),\\ &\mathrm{RD}(x_u \cdot y_l),\ \mathrm{RD}(x_u \cdot y_u)),\\ &\max(\mathrm{RU}(x_l \cdot y_l),\ \mathrm{RU}(x_l \cdot y_u),\\ &\mathrm{RU}(x_u \cdot y_l),\ \mathrm{RU}(x_u \cdot y_u))]\end{aligned} \tag{3}$$

where $\mathrm{RD}(x)$ rounds x towards negative infinity and $\mathrm{RU}(x)$ rounds x towards positive infinity.

The guarantee of sound floating point arithmetic in interval arithmetic is valuable for safety critical applications in which round-off errors can be catastrophic [24], [25]. Recent work has thus studied methods for automatically producing efficient interval arithmetic programs with higher certified accuracy [26], [27]. In this work we use this guarantee for designing an algorithm-based approach to soft error tolerant hardware.

IV. Interval Arithmetic-based Soft Error Tolerance

In this section we assume the single event upset fault model in which exactly one internal node in the datapath can have its logic state flipped at a given clock period. The focus of this work lies in the protection of logic or datapath modules which implement a specific computation. Memories are assumed to be protected through ECCs and scrubbing logic. The following section discusses how we adopt interval arithmetic for lower overhead hardware redundancy and perform algorithm specific forward error analysis for error detection.

A. Hardware Redundancy via Interval Arithmetic

Fig. 2 shows the block diagram of the proposed technique. Module 1 contains the original hardware which executes its computation on the unmodified inputs. A lower precision version of the module executes the same computation in interval arithmetic using floating-point, providing hardware redundancy for error detection.

Inputs to the interval arithmetic block are lower precision intervals constructed such that the original number is guaranteed to be within the interval. The input values are first cast to a lower precision floating point representation and then rounded towards $+\infty$ and towards $-\infty$ to get the upper and lower endpoint values. Thus the constructed input intervals are small—the error margin is at most one unit in least precision (ulp), the smallest difference between two adjacent floating point numbers. Constants within the module are also cast to intervals using the same procedure and in the case that the number is exactly representable in floating point will

have the same endpoints. The interval outputs and the high precision outputs from the original module all feed into the error detection block.

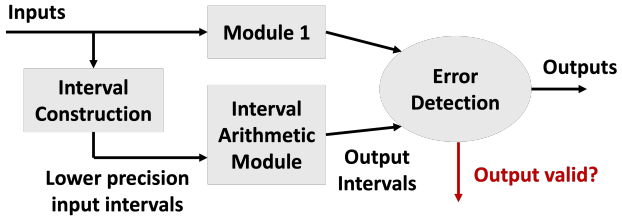

Fig. 2. Interval arithmetic based approach for soft error tolerance.

Interval arithmetic addition and subtraction is implemented using two adders to compute the upper and lower intervals. For subtraction the second operands are swapped and the sign bit is flipped. In interval arithmetic multiplication, a naive implementation following Eq. 3 may use four multipliers for each of the operand combinations in computing one endpoint. Instead we perform sign bit logic to swap operands on the fly and require only two multipliers for interval multiplication. Constant multiplication is even simpler since the sign of one operand is known at design time. The arithmetic unit for the upper endpoint implements rounding towards $+\infty$ and for the lower endpoint rounding towards $-\infty$. Fig. 3 shows block diagrams of the interval arithmetic units.

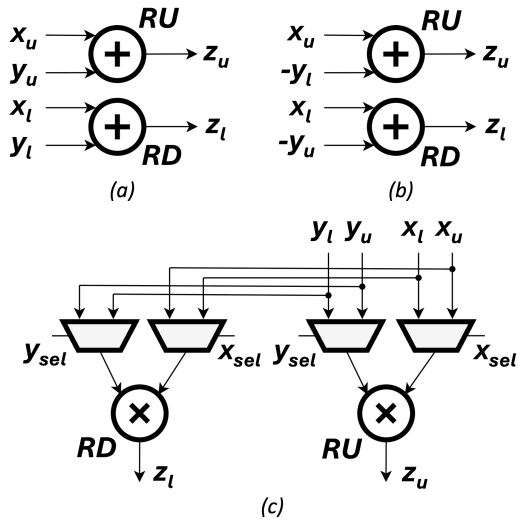

Fig. 3. Interval arithmetic hardware block diagram for (a) addition, (b) subtraction, and (c) multiplication.

B. Error Detection

The guarantee of sound floating-point arithmetic and forward error analysis of the computation enable error detection. There are four general cases depicted in Fig. 4 which we must consider for the output interval and middle value. Fig. 5 shows a block diagram of the steps for error detection. For any interval $\hat{x} = [x_l, x_u]$, we first check that $x_u > x_l$ and $d(x_u, x_l) < T$ where $d(x, y)$ is the distance between two floating point numbers, x and y, in terms of ulp. The threshold value, T, is determined in terms of ulp through forward error analysis, which is discussed in the following section. If both conditions are satisfied then the interval is accepted as valid. Otherwise we declare that there is an interval error.

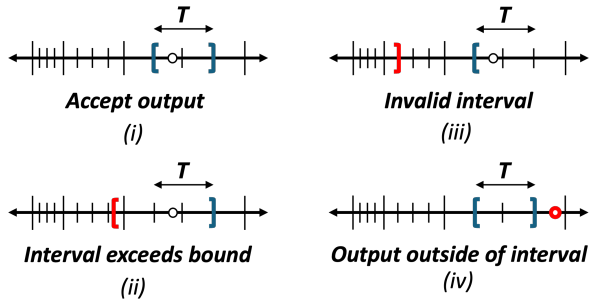

Fig. 4. The four potential cases for an output interval: (i) no error, middle value accepted (ii) interval grows beyond bound, (iii) interval endpoints flipped, and (iv) high precision middle value outside of interval.

We then check whether the high precision middle value, x_m, is contained within the interval. If $x_l < x_m < x_u$, then we accept the high precision computation of the original module and can write the value to a protected memory. If the condition does not hold, then an error must have occurred in the high precision block and we mark the computation invalid. The output control signal can be used to indicate to a controller that the computation must be redone. We assume that the algorithms to be protected are numerically stable and bit flips that affect the lowest order mantissa bits while remaining within the bound are tolerable.

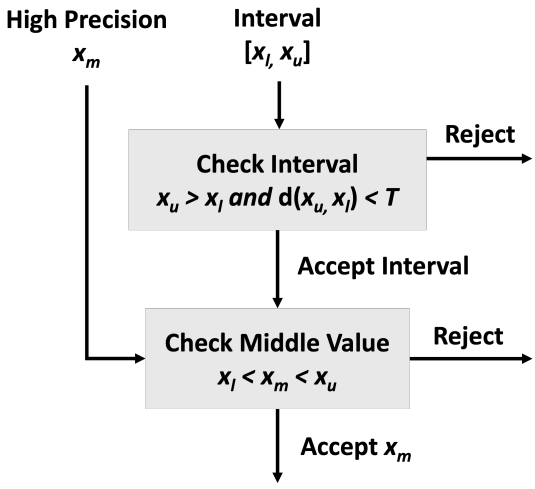

Fig. 5. High level block diagram of the proposed method for error detection.

C. Forward Error Analysis

Using forward error analysis we can analyze the sequence of operations through the computation to provide a meaningful

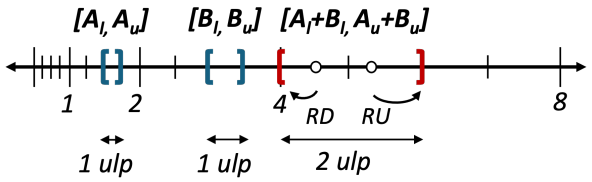

Fig. 6. Example of interval growth after floating point operations. Each interval begins at 1 ulp distance but the resulting interval grows to 2 ulp due to rounding.

bound on the growth of the interval. We also assume that the dynamic range of the inputs and outputs are known either by properties of the application or through static analysis. All computations begin with intervals that are constructed such that the width is at most one ulp. In interval addition, rounding to $+\infty$ introduces an absolute roundoff error of at one ulp. Rounding down incurs the same error, so the growth of the interval is bounded by at most two ulp for a single interval addition. Fig. 6 shows an example of how the resulting interval width can grow following interval addition. In multiplication, we can bound the growth of the interval by constraining multiplicand range. For example when computing FFTs, multiplication is always with twiddle factors so it is guaranteed that one multiplicand is in the range [0, 1] and the growth is fixed to 2 ulp.

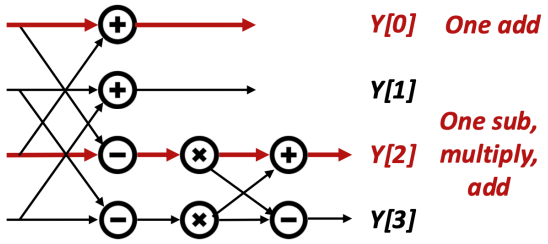

Fig. 7. Analysis of an example 2-point FFT butterfly datapath to bound the output interval growth.

For each output of the module we analyze the critical path of additions/subtractions and multiplications to compute the bound, T, on the growth of each interval. The bound is computed in terms of ulp whose absolute value depends on the range of the number. To avoid tracking dynamic range throughout the computation, we use a worst case bound by performing static analysis to determine the maximum ulp across the entire computation. The block diagram in Fig. 8 shows how we implement the interval check in hardware. The error detection module uses the exponent associated with the maximum ulp value to first normalize and shift the interval mantissas. Then, the mantissas can be subtracted and compared against the bound, T. For every output interval we perform forward error analysis as shown in Fig. 7 for a 2-point butterfly and implement the corresponding error checking module in Fig. 8.

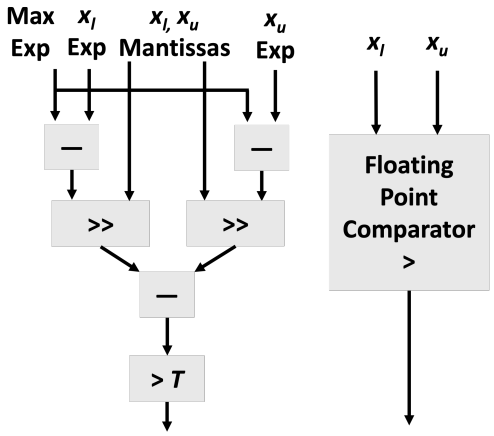

Fig. 8. Hardware block diagram for interval checking using forward error analysis.

V. CosmIC: A Silicon Prototype

We implement the algorithm-based approach using interval arithmetic for a radix-8 fast Fourier Transform datapath and a systolic array. We also design test structures to facilitate fault injections and error evaluation for each of the modules. Fig. 10 shows a block diagram of the test chip.

A. Fast Fourier Transform

The fast Fourier Transform (FFT) datapath shown in Fig. 9 implements a fully unrolled, complex radix-8 FFT supporting 32-bit single precision floating point arithmetic. In the interval arithmetic version of the module each arithmetic unit is replaced with a 16-bit half precision interval arithmetic unit. Twiddle factor constants within the datapath are also converted to intervals. Both the original module and the interval arithmetic version are pipelined to 13 stages to ensure all outputs are aligned. Both single precision and half precision interval FFT hardware are automatically generated using the SPIRAL [28] code generation system.

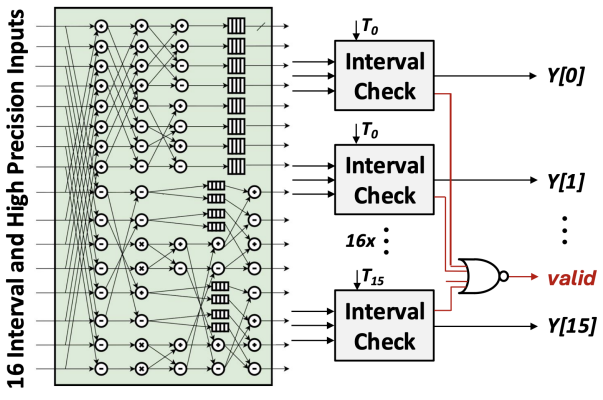

Fig. 9. Architecture of the radix-8 FFT datapath and interval error detection.

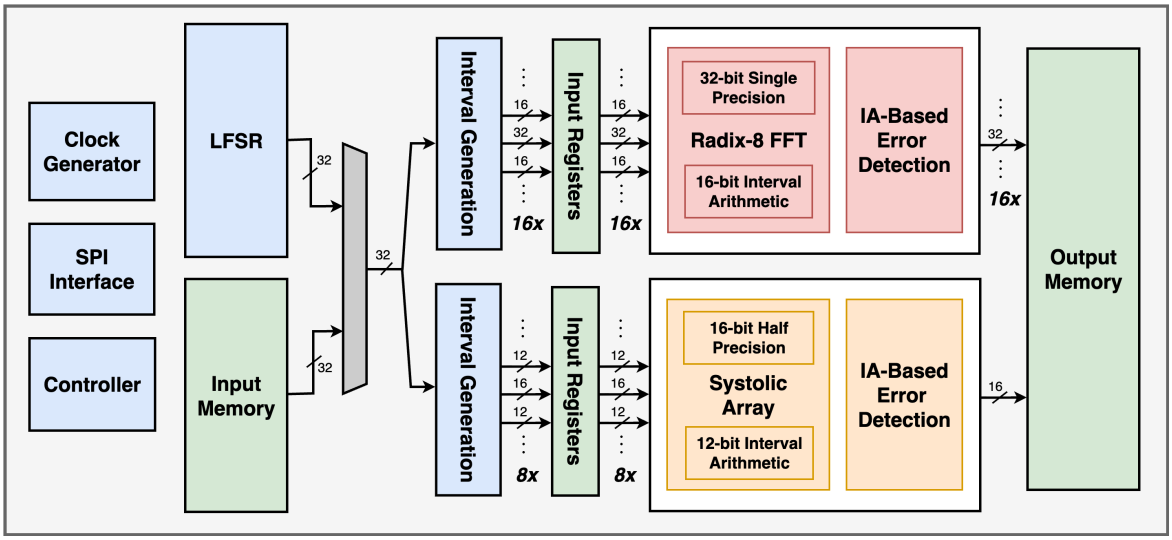

Fig. 10. System diagram of the CosmIC test chip. The datapaths include a complex radix-8 FFT and a systolic array, each replicated in interval arithmetic. Input data can be generated through the LFSRs or specific tests can be loaded into the on-chip memory. Additional test structures enable fault injection tests and error tracking.

Each 32-bit output and 16-bit interval feeds directly to the error detection block to check validity of the outputs. A final reduction step across all interval outputs indicates whether the computation can be accepted as valid. If invalid, batched FFT computations [29] can be restarted by a controller.

The bound on interval growth for the FFT is calculated by tracking the number of operations that occurs for each output. We also assume that the input dynamic range is bounded to values in the range of [0, 1]. For example, the first output in Fig. 9 incurs a maximum growth of 6 ulp from the three addition operations and we know that the maximum output dynamic range by properties of the FFT is [0, 8]. Thus the maximum exponent and the threshold bound is determined and can be used for error detection.

B. Systolic Array

We also implement a parametrized systolic array architecture composed of 16-bit IEEE-compliant Fused-MAC (FMA) units. The interval version implements a custom 12-bit floating point format where the exponent is 5 bits, mantissa is 6 bits, and the final bit is reserved for the sign. This design implements a small 4x1 systolic array.

A similar checker is used for the systolic array outputs. The bounds on interval growth for the systolic array are calculated based on the dimensions of the array, where each FMA unit contributes to the growth.

C. Interval Generation

We design a block for generating intervals on the fly as floating point numbers enter each of the computations. The numbers are rounded towards $+\infty$ and towards $-\infty$ for the desired lower precision representation: 16-bit half precision for the FFT and the custom 12-bit format for the systolic array. This creates intervals which are exactly one ulp distance away from each other. The logic is entirely combinational for each input going into the respective block. This also enables the protected hardware to act as a drop-in replacement since the interface to the data does not change from the original hardware implementation.

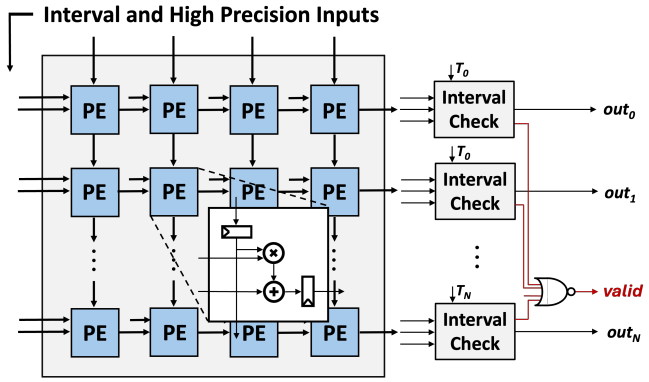

Fig. 11. Systolic array architecture and error detection block. Each PE consists of an FMA unit that computes in interval arithmetic and a higher precision floating point representation.

D. Test Infrastructure

We implement linear feedback shift registers (LFSRs) to generate pseudo-random inputs for both the FFT and systolic array modules. For each of the modules we would also like to bound the input dynamic range, so we add additional post-processing logic to ensure the generated floating point inputs are within the fixed range. We also design synthesized memories so that specific test vectors can be loaded for test and evaluation post-silicon.

To facilitate fault injection tests, we insert a dedicated scan chain for all flops in the core datapaths. This allows us

to scan out the state of the computation, flip any bit, and scan the modified state back in to resume computation. The error detection block also incorporates additional logic for tracking error counts and location across the output intervals. A ring oscillator-based on-die clock generator can provide frequencies up to the target 500 MHz core frequency.

VI. Experimental Results

A. Silicon Prototype

The 1mm x 1mm test chip is fabricated in a TSMC 28nm process and has a core area of 0.7mm x 0.7mm. A micrograph of the die and the layout of the core components is shown in Fig. 12.

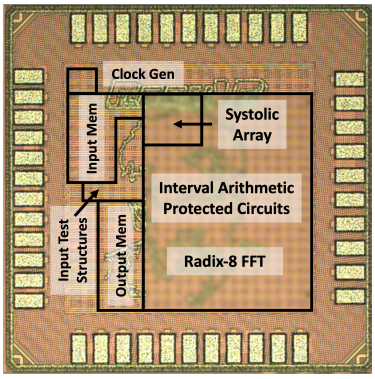

Fig. 12. 1mm x 1mm die micrograph. There are 44 IO pads and three power domains. The protected accelerators are on the right side of the die. Additional test infrastructure including LFSRs, memories, interval generation, and clock generator are on the left side of the die.

The design targets a core frequency of 500 MHz. There are three power domains for the clock generator, FFT/systolic array, and other control/test logic. The chip can be operated through the LFSRs which feed data directly to the datapaths or through specific test vectors that are loaded to the input memories. The input/output memories are large enough to saturate the pipeline of the datapaths.

B. Comparison against TMR

To evaluate our approach we implement a TMR version of the modules as a baseline comparison against the fabricated test chip. All results are extracted from post place-and-route designs and include the logic area required for error handling.

The proposed interval arithmetic approach achieves a 2 times reduction in area for the 8-point complex FFT and 3.3 times reduction for the systolic array compared to the traditional TMR approach. The greater area improvement in the systolic array design can be attributed to its higher fraction of multipliers which scale down faster with decreased bitwidth. We also note the potential for further savings by lowering the interval precision.

C. Efficacy Results

We discuss some results from fault injection simulation to demonstrate that our method effectively detects bit flips occurring in the datapath. The test cases presented below are extracting the 11th output of the FFT datapath and each one corresponds to a scenario presented in Fig. 4. For each result we present the lower, middle, and upper $\{x_l, x_m, x_u\}$ values with the affected value in bold.

1) Result: $\{\mathbf{0.6528}, 0.6505, 0.6509\}$. The interval check immediately catches that $0.6528 > 0.6509$ and signals that the result is invalid.
2) Result: $\{0.6504, \mathbf{0.6507}, 0.6509\}$. The center value's lower mantissa bit is affected, resulting in a value of 0.6507. However the interval result $[0.6504, 0.6509]$ is tight and still bounds the center value. Thus the error is tolerable and the result can be accepted.
3) Result: $\{0.6504, 0.6505, \mathbf{0.9004}\}$. There is an error in the MSBs of the upper endpoint of the interval. The interval distance is much larger than the allowable bound and the result is declared as invalid.

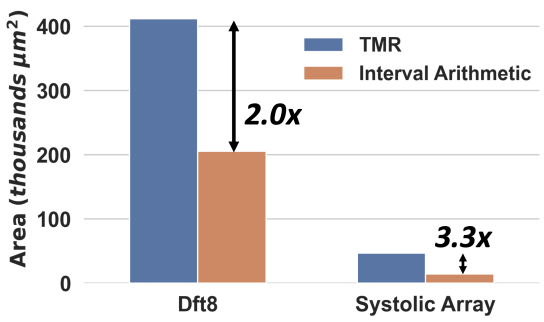

Fig. 13. Comparison of total area between TMR and interval arithmetic protected implementations.

VII. Conclusion

Modern technology scaling poses an increased reliability concern for soft errors in space and terrestial applications. In this paper we proposed a novel algorithm-based method for hardening by design using interval arithmetic. We presented a first look through the design of a test chip protecting an FFT and systolic array datapath. Our measured post-PnR results show a lower area overhead than the traditional TMR method for radiation-hardening. Through fault injection RTL simulation, we also show that the approach successfully detects problematic bit flips in the datapaths.

There are many avenues for future work. It would be interesting to evaluate tradeoffs between reliability and precision of the interval arithmetic and high precision hardware. Ultimately, we look towards building a fully hardened-by-design system that operates with protected memories and control logic to facilitate recomputation in an end-to-end application.

Acknowledgments

This work was supported in part by PRISM, one of seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. We also thank Apple's New Silicon Initiative for academic tapeout support.

REFERENCES

[1] D. M. Fleetwood, "Radiation effects in a post-moore world," *IEEE Transactions on Nuclear Science*, vol. 68, no. 5, pp. 509–545, 2021.

[2] D. Kobayashi, "Scaling trends of digital single-event effects: A survey of seu and set parameters and comparison with transistor performance," *IEEE Transactions on Nuclear Science*, vol. 68, no. 2, pp. 124–148, 2021.

[3] H. D. Dixit, S. Pendharkar, M. Beadon, C. Mason, T. Chakravarthy, B. Muthiah, and S. Sankar, "Silent data corruptions at scale," 2021. [Online]. Available: https://arxiv.org/abs/2102.11245

[4] R. Bonderson, "Training in turmoil: Silent data corruption in systems at scale," in *International Test Conference Silicon Lifecycle Management Workshop*, 2021.

[5] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, "Cores that don't count," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, 2021, pp. 9–16.

[6] O. Atli, "Soft error analysis and design space exploration of radiation-hardened system-on-a-chip platforms," Ph.D. dissertation, Carnegie Mellon University, 2024.

[7] J.-M. Daveau, A. Blampey, G. Gasiot, J. Bulone, and P. Roche, "An industrial fault injection platform for soft-error dependability analysis and hardening of complex system-on-a-chip," in *2009 IEEE International Reliability Physics Symposium*, 2009, pp. 212–220.

[8] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Transactions on Computers*, vol. C-33, no. 6, pp. 518–528, 1984.

[9] A. Reddy and P. Banerjee, "Algorithm-based fault detection for signal processing applications," *IEEE Transactions on Computers*, vol. 39, no. 10, pp. 1304–1308, 1990.

[10] B. Shim and N. R. Shanbhag, "Energy-efficient soft error-tolerant digital signal processing," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 14, no. 4, p. 336–348, Apr. 2006. [Online]. Available: https://doi.org/10.1109/TVLSI.2006.874359

[11] K. Zhao, S. Di, S. Li, X. Liang, Y. Zhai, J. Chen, K. Ouyang, F. Cappello, and Z. Chen, "Ft-cnn: Algorithm-based fault tolerance for convolutional neural networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1677–1689, 2021.

[12] J. Kosaian and K. V. Rashmi, "Arithmetic-intensity-guided fault tolerance for neural network inference on gpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476184

[13] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou, "Algorithm-based fault tolerance applied to high performance computing," *Journal of Parallel and Distributed Computing*, vol. 69, no. 4, pp. 410–416, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731508002141

[14] Z. Chen and J. Dongarra, "Algorithm-based fault tolerance for fail-stop failures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 12, pp. 1628–1641, 2008.

[15] L. Narmour, S. Derrien, and S. Rajopadhye, "Automatic algorithm-based fault tolerance (aabft) of stencil computations," in *2023 32nd International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2023, pp. 187–198.

[16] Z. Chen, "Online-abft: an online algorithm based fault tolerance scheme for soft error detection in iterative methods," in *Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 167–176. [Online]. Available: https://doi.org/10.1145/2442516.2442533

[17] S. Pontarelli, G. Cardarilli, M. Re, and A. Salsano, "Totally fault tolerant rns based fir filters," in *2008 14th IEEE International On-Line Testing Symposium*, 2008, pp. 192–194.

[18] P. Reviriego, C. J. Bleakley, and J. A. Maestro, "Structural dmr: A technique for implementation of soft-error-tolerant fir filters," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 58, no. 8, pp. 512–516, 2011.

[19] J.-Y. Jou and J. Abraham, "Fault-tolerant fft networks," *IEEE Transactions on Computers*, vol. 37, no. 5, pp. 548–561, 1988.

[20] Y.-H. Choi and M. Malek, "A fault-tolerant fft processor," *IEEE Transactions on Computers*, vol. 37, no. 5, pp. 617–621, 1988.

[21] S.-J. Wang and N. Jha, "Algorithm-based fault tolerance for fft networks," *IEEE Transactions on Computers*, vol. 43, no. 7, pp. 849–854, 1994.

[22] P. Wu, Q. Guan, N. DeBardeleben, S. Blanchard, D. Tao, X. Liang, J. Chen, and Z. Chen, "Towards practical algorithm based fault tolerance in dense linear algebra," in *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 31–42. [Online]. Available: https://doi.org/10.1145/2907294.2907315

[23] R. E. Moore, *Methods and applications of interval analysis*. SIAM, 1979.

[24] D. Delmas, E. Goubault, S. Putot, J. Souyris, K. Tekkal, and F. Védrine, "Towards an industrial use of fluctuat on safety-critical avionics software," in *Formal Methods for Industrial Critical Systems*, M. Alpuente, B. Cook, and C. Joubert, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 53–69.

[25] F. Franchetti, T. M. Low, S. Mitsch, J. P. Mendoza, L. Gui, A. Phaosawasdi, D. Padua, S. Kar, J. M. Moura, M. Franusich, J. Johnson, A. Platzer, and M. M. Veloso, "High-assurance spiral: End-to-end guarantees for robot and car control," *IEEE Control Systems Magazine*, vol. 37, no. 2, pp. 82–103, 2017.

[26] J. Rivera, F. Franchetti, and M. Püschel, "An interval compiler for sound floating-point computations," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 52–64.

[27] ——, "A compiler for sound floating-point computations using affine arithmetic," in *2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2022, pp. 66–78.

[28] F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Püschel, J. C. Hoe, and J. M. F. Moura, "Spiral: Extreme performance portability," *Proceedings of the IEEE*, vol. 106, no. 11, pp. 1935–1968, 2018.

[29] L. Tang, S. Chen, K. Harisrikanth, G. Xu, K. Mai, and F. Franchetti, "A high throughput hardware accelerator for fftw codelets: A first look," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, 2022, pp. 1–7.