International Journal of Innovative Research in Engineering
Volume 7, Issue 4 (July - August 2026), PP: 01-06.
https://www.doi.org/10.59256/ijire.20260704001
www.theijire.com
ISSN No: 2582-8746

Hardware-Efficient Triple Modular Redundancy via Shared Arithmetic Resources

B. Gayathiri[1], N. Peramlatha[2]
[1]*VLSI Design, Vandayar Engineering College, Thanjavur, Tamilnadu, India.*
[2]*Assistant Professor, Vandayar Engineering College, Thanjavur, Tamilnadu, India.*

How to cite this paper:
B. Gayathiri[1], N. Peramlatha[2], "Hardware-Efficient Triple Modular Redundancy via Shared Arithmetic Resources", IJIRE-V7I4-01-06.

Abstract: *Conventional Triple Modular Redundancy (TMR) provides robust fault masking by triplicating functional logic and applying a two-out-of-three majority voter. Although this approach is effective against single faults and transient upsets, full replication of multiplier-intensive datapaths results in large area, power, and thermal overhead. This paper proposes a hardware-efficient resource-sharing TMR architecture that preserves the voting semantics of TMR while replacing three dedicated multipliers with a single low-power shared Booth-Wallace multiplier. The three redundant datapaths retain independent local logic, registers, and control state, while a round-robin arbiter and time-division multiplexing scheduler coordinate access to the shared arithmetic resource through request, grant, and done handshaking. Analytical evaluation indicates that the design can reduce normalized area from approximately 3.0x to 1.4-1.5x and normalized power from approximately 3.0x to 1.1-1.3x compared with conventional full TMR, while preserving deterministic masking of one faulty replica. The proposed approach is suitable for energy-constrained and mission-critical systems such as nanosatellites, portable avionics, embedded sensor processors, and implantable medical electronics.*

Key Words: *Triple Modular Redundancy, resource sharing, shared multiplier, low-power VLSI, fault tolerance, FPGA, DSP accelerator, majority voter.*

I.INTRODUCTION

Fault-tolerant digital design is essential in systems that must continue operating under radiation, soft errors, transient faults, aging-induced failures, and harsh environmental conditions. TMR is one of the most widely used architectural mitigation techniques because it executes the same computation in three replicas and forwards the final output through a two-out-of-three voter [1]. A single faulty replica can therefore be masked when the other two replicas agree. FPGA and space-grade design guides have also emphasized TMR as a practical approach for protecting logic implemented in reprogrammable devices [2], [3].

The main limitation of conventional TMR is its direct hardware overhead. A complete TMR implementation requires three copies of the datapath, three copies of the arithmetic units, and majority voting logic. In DSP-oriented circuits, the multiplier is often one of the dominant sources of silicon area, switching activity, and leakage. Triplicating this block can make the design impractical for size- and power-constrained platforms. Therefore, reducing multiplier redundancy while retaining TMR-level single-replica fault masking is a meaningful design objective.

This work proposes a resource-sharing TMR architecture in which the local datapath portions of the three replicas remain triplicated, while multiplication is executed by a single shared Booth-Wallace multiplier. A round-robin arbiter serializes multiplication requests, returns each product to the requesting replica, and allows the final majority voter to compare three independently stored results. The key contribution is an analytical architecture that lowers multiplier resource overhead without abandoning TMR voting semantics.

II.RELATED WORK

Lyons and Vanderkulk introduced the classical TMR concept as a two-out-of-three voting method for improving computer reliability [1]. Since then, TMR has become a core method for fault-tolerant embedded systems, avionics, and FPGA-based radiation mitigation. Xilinx and AMD design guides describe practical methods for inserting TMR logic, voter placement, and state-machine synchronization in FPGA devices [2], [3]. Kastensmidt et al. showed that voter insertion strategy strongly affects robustness in SRAM-based FPGA TMR designs, especially because routing upsets can violate independence among redundant domains [4]. NASA studies on single-event effects also discuss the vulnerability of FPGA devices and the importance of mitigation schemes such as TMR in radiation-prone environments [5].

Multiplier design is another important research area for low-power VLSI and DSP accelerators. Booth multiplication reduces partial-product complexity for signed multiplication [6], while Wallace tree compression improves multiplication

speed by reducing the partial-product summation depth [7]. Recent low-power multiplier studies continue to optimize Wallace-tree and approximate multiplier structures for reduced energy and area [8], [9]. However, many fault-tolerant datapath designs still rely on full arithmetic triplication. The proposed work addresses this gap by combining TMR semantics with shared multiplier scheduling.

III.CONVENTIONAL TMR AND PROBLEM DEFINITION

A conventional TMR datapath instantiates three identical processing paths, commonly denoted as Module A, Module B, and Module C. Each module receives the same input and produces one output. The majority voter forwards the value that appears in at least two outputs. If one module is affected by a transient or permanent fault, the two remaining correct modules determine the final output.

The reliability benefit comes at the cost of increased resource usage. If a non-redundant datapath has normalized area A and power P, a full TMR datapath has an approximate normalized cost of 3A and 3P before voter overhead. For a multiplier-heavy datapath, the multiplier dominates the marginal cost. The problem addressed in this paper is therefore: how can multiplier overhead be reduced while preserving the ability to mask a single faulty local replica?

IV.PROPOSED RESOURCE-SHARING TMR ARCHITECTURE

The proposed architecture retains three independent local datapath replicas but removes the dedicated multiplier from each replica. Multiplication requests are routed to one shared Booth-Wallace multiplier. The local datapath logic, product registers, and final voter remain triplicated or independently stored so that the voter still compares three replica-level results.

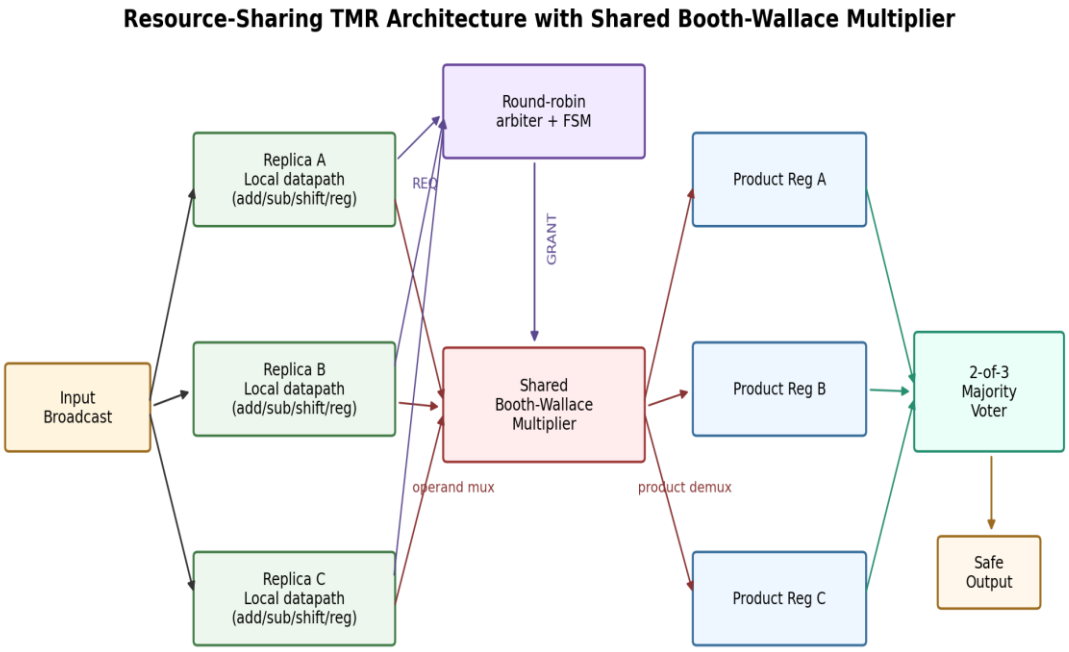

Figure 1. Proposed resource-sharing TMR architecture with triplicated local datapaths and one shared multiplier

4.1 Operating Principle

- Input operands are broadcast to the three redundant local datapath replicas.
- Non-multiplication operations such as addition, subtraction, shifting, comparison, and register transfer are executed locally inside each replica.
- When multiplication is required, the requesting replica asserts a request signal to the arbiter.
- The round-robin arbiter grants access in the order A -> B -> C, or according to the next pending valid request.
- The shared Booth-Wallace multiplier computes the product and returns it to the requesting replica product register.
- After the three replica results are available, the two-out-of-three majority voter produces the final safe output.

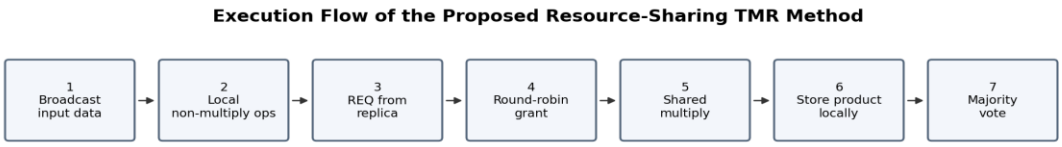

Figure 2. Methodology flow for serialized multiplication and majority voting.

4.2 Design Rationale

The central idea is to distinguish between replica-level independence and arithmetic-resource duplication. Complete TMR duplicates all resources, including the multiplier. The proposed architecture keeps the three logical result paths independent at the register and voting level, but uses temporal sharing for one large arithmetic unit. This approach is most attractive when the target workload has moderate multiplication demand or when energy saving is more important than peak throughput.

V.HARDWARE IMPLEMENTATION MODEL

The implementation can be organized into four RTL modules: the top-level integration module, three redundant local datapath modules, one shared multiplier module, and the majority voter. The arbiter can be implemented as a small finite-state machine that cycles through A, B, and C request states.

Parameter	Selected implementation
Datapath word size	32-bit
Shared multiplier	Radix-4 Booth encoder with Wallace-tree reduction
Multiplier latency	3 clock cycles, analytically assumed
Scheduling policy	Round-robin time-division multiplexing
Handshake signals	REQ, GRANT, DONE
Local protection	Triplicated datapath registers and local control
Final protection	2-of-3 majority voter
Shared-resource protection	Parity/error-detection wrapper; optional hardened multiplier

Table 1. Hardware implementation parameters of the proposed architecture.

5.1 Arbiter State Model

Let R_A, R_B, and R_C represent pending multiplication requests from replicas A, B, and C. At each scheduling opportunity, the arbiter selects one valid request and asserts the corresponding grant G_i. A simple round-robin state machine can be expressed as:

NextGrant = A if previous grant was C and R_A = 1; otherwise the first pending request after the previous grant in cyclic order A -> B -> C.

This policy prevents starvation and keeps the scheduling logic small. Additional priority policies may be used in future versions if the application has deadline-sensitive tasks.

5.2 Majority Voter Model

For a single output bit, the majority voter is implemented as:

$$Y = (A \cdot B) + (B \cdot C) + (A \cdot C)$$

For a word-level datapath, the expression is applied bitwise. This voter returns the value agreed upon by any two replicas. Therefore, if one local replica is faulty but the other two are correct and synchronized, the final output remains correct.

VI.FAULT-TOLERANCE ANALYSIS

The proposed architecture preserves single-replica fault masking in the final output path. If one local datapath replica produces an incorrect stored result, the two correct replicas determine the voter output. Table 2 illustrates representative cases.

Replica A	Replica B	Replica C	Voter output	Interpretation
5	5	3	5	Replica C faulty; masked
7	2	7	7	Replica B faulty; masked
4	4	4	4	All replicas agree
9	8	7	Undefined / flagged	No majority; requires error handling

Table 2. Representative majority-voting outcomes.

A shared multiplier introduces an important reliability consideration because it can become a common-mode failure point. This paper therefore treats the shared multiplier as a protected arithmetic service rather than an unprotected block. The

recommended mitigation options are parity or residue checking on multiplier outputs, hardened cell implementation, temporal recomputation for critical operations, and optional duplication of the shared multiplier in ultra-high-reliability deployments. This limitation should be explicitly evaluated during FPGA synthesis and fault-injection experiments.

Fault location	Effect in conventional TMR	Effect in proposed design	Mitigation
One local replica	Masked by voter	Masked by voter	Maintain independent local registers and voter
Voter	Potential single point of failure	Potential single point of failure	Hardened or duplicated voter
Shared multiplier	Not present as shared point	Potential common-mode fault	Parity/residue checking, hardening, temporal recomputation
Arbiter FSM	Not required or local	Can affect scheduling	FSM hardening, watchdog, state encoding
Routing/upsets	Can break domain isolation	Can break domain isolation	Floorplanning, voter insertion, scrubbing

Table 3. Fault-location analysis and mitigation strategy.

VII.ANALYTICAL EVALUATION AND RESULTS

The current evaluation is analytical and RTL-oriented. It assumes a 32-bit datapath and a multiplier-heavy design in which replacing three multipliers with one multiplier provides the dominant reduction. The values should be interpreted as design estimates until confirmed by FPGA synthesis reports from tools such as Vivado or Quartus.

Metric	Conventional full TMR	Proposed shared-multiplier TMR
Multiplier count	3	1
Local datapath replicas	3	3
Majority voting	2-of-3 voter	2-of-3 voter
Normalized area	~3.0x	~1.4-1.5x
Normalized power	~3.0x	~1.1-1.3x
Single local-replica fault masking	Yes	Yes
Control complexity	Low	Moderate
Multiplier throughput	Parallel	Sequential / scheduled
Best use case	Maximum reliability with sufficient resources	Energy-constrained reliable embedded DSP

Table 4. Comparison between conventional TMR and the proposed architecture

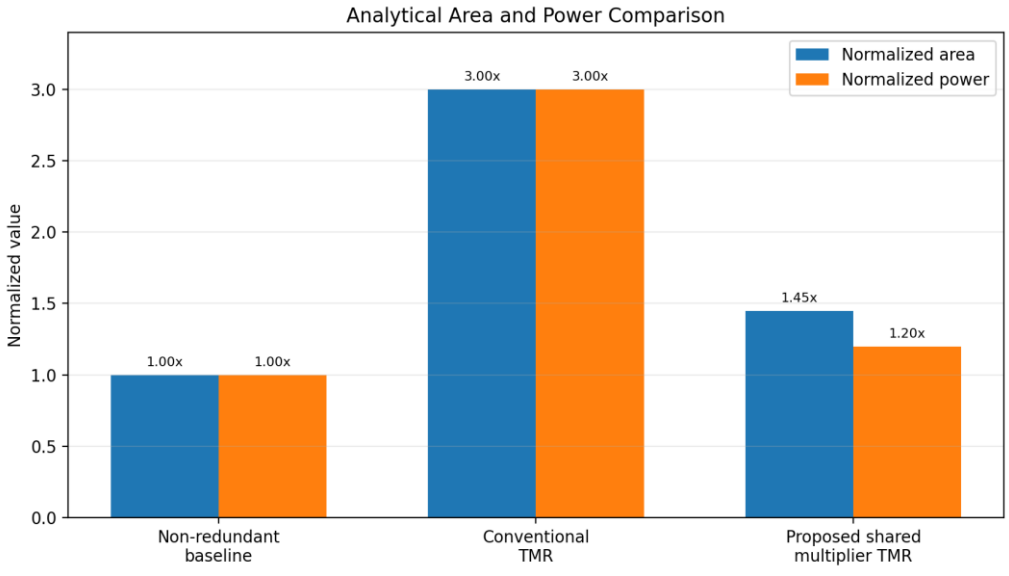

Figure 3. Analytical normalized area and power comparison.

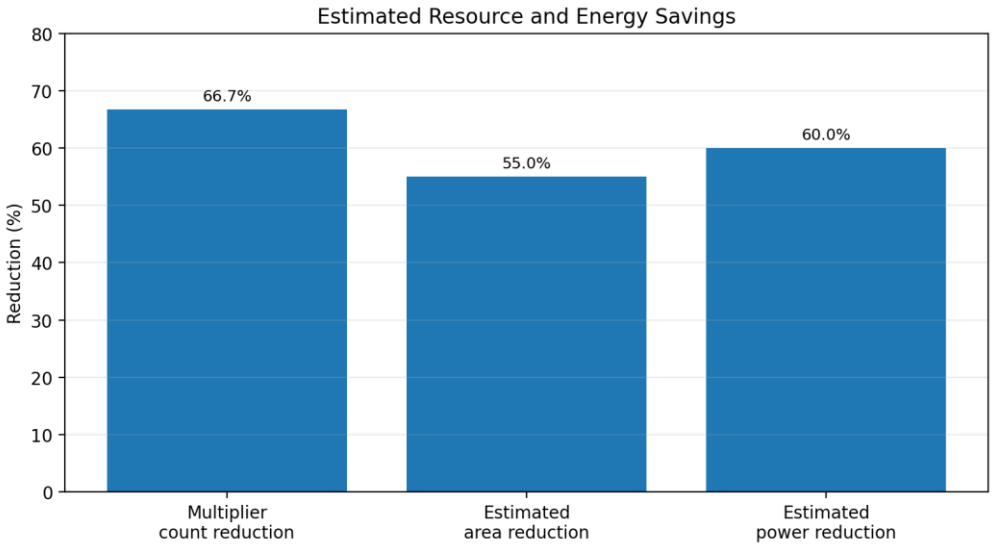

Figure 4. Estimated multiplier count, area, and power reductions.

7.1 Power-Saving Derivation

If P_m is the power consumed by one multiplier, conventional TMR uses approximately 3P_m for multiplier operation. The proposed architecture uses approximately P_m plus a small scheduling overhead P_ctrl. Ignoring the control overhead, the ideal multiplier power saving is:

Saving = ((3P_m - P_m) / 3P_m) x 100 = 66.7%

With arbiter, multiplexing, and register overhead included, the expected net power saving is reported as 55-65%. Similarly, because the largest arithmetic block is reduced from three instances to one instance, the expected area reduction is approximately 50-60%, depending on the proportion of the multiplier in the original datapath.

7.2 Fault-Injection Evaluation Plan

For a complete journal submission, the analytical evaluation should be extended using simulation and FPGA synthesis. Table 5 provides a recommended experimental plan.

Experiment	Purpose	Expected output
RTL simulation	Verify correct scheduling and voting	Waveforms for REQ/GRANT/DONE and voter output
Single-replica fault injection	Validate masking of A, B, or C errors	Fault coverage table
Shared-multiplier transient fault	Check parity/residue detection behavior	Detection-rate report
FPGA synthesis	Measure LUT, FF, DSP, BRAM, and Fmax	Vivado/Quartus utilization report
Power analysis	Compare dynamic and static power	Estimated power report under equal clock conditions

Table 5. Recommended validation plan before final journal submission.

VIII.DISCUSSION

The major benefit of the proposed architecture is that it reduces the most expensive arithmetic resource while preserving TMR-style output voting. This is particularly useful for applications where multiplications are frequent but do not require all three replicas to execute multiplication in the same clock cycle. The trade-off is increased latency because multiplication requests are serialized. For high-throughput pipelines, the design may require pipelining, buffering, or partial duplication of the shared multiplier.

A second important design issue is common-mode failure. In full TMR, each replica has its own multiplier, so a single multiplier fault affects only one replica. In the proposed design, the multiplier is shared and therefore must be protected. The paper therefore recommends parity or residue checking, hardened implementation, watchdog-controlled arbiter logic, and optional duplication for extremely strict dependability levels. These mitigations should be selected according to the target reliability requirement and energy budget.

IX.CONCLUSION

This paper presented a hardware-efficient resource-sharing TMR architecture that replaces three dedicated multipliers with a single shared Booth-Wallace multiplier while preserving majority-vote fault masking at the replica-output level. The design uses triplicated local datapaths, round-robin arbitration, time-division multiplexed multiplication, local

product storage, and a two-out-of-three voter. Analytical results indicate approximately 50-60% area reduction and 55-65% power reduction compared with conventional full TMR, with the main cost being sequential multiplier latency and the need to protect the shared arithmetic block. Future work should include FPGA synthesis, post-place-and-route power analysis, timing closure, and systematic fault-injection experiments to quantify robustness under realistic radiation and transient-fault models.

References

1. R. E. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," IBM Journal of Research and Development, vol. 6, no. 2, pp. 200-209, Apr. 1962, doi: 10.1147/rd.62.0200.
2. Xilinx, "Triple Module Redundancy Design Techniques for Virtex FPGAs," Application Note XAPP197, Rev. 1.0.1, July 2006.
3. Xilinx/AMD, "TMRTool Software Version 13.2 User Guide," UG156, Rev. 3.1.2, June 2017.
4. [4] F. L. Kastensmidt, L. Sterpone, L. Carro, and M. S. Reorda, "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs," in Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE), 2005, pp. 1290-1295, doi: 10.1109/DATE.2005.229.
5. M. D. Berg, "Single Event Effects in FPGA Devices," NASA Electronic Parts and Packaging Program, 2015.
6. A. D. Booth, "A signed binary multiplication technique," Quarterly Journal of Mechanics and Applied Mathematics, vol. 4, no. 2, pp. 236-240, 1951, doi: 10.1093/qjmam/4.2.236.
7. C. S. Wallace, "A suggestion for a fast multiplier," IEEE Transactions on Electronic Computers, vol. EC-13, no. 1, pp. 14-17, Feb. 1964, doi: 10.1109/PGEC.1964.263830.
8. K. B. Jaiswal, N. Kumar V., P. Seshadri, and G. Lakshminarayanan, "Low power Wallace tree multiplier using modified full adder," in Proc. 3rd International Conference on Signal Processing, Communication and Networking (ICSCN), 2015, pp. 1-4, doi: 10.1109/ICSCN.2015.7219880.
9. V. Solanki, A. D. Darji, and H. Singapuri, "Design of low-power Wallace tree multiplier architecture using modular approach," Circuits, Systems, and Signal Processing, vol. 40, pp. 4407-4427, 2021, doi: 10.1007/s00034-021-01671-3.
10. Y. Wu, C. Chen, W. Xiao, X. Wang, C. Wen, J. Han, X. Yin, W. Qian, and C. Zhuo, "A survey on approximate multiplier designs for energy efficiency: From algorithms to circuits," ACM Transactions on Design Automation of Electronic Systems, vol. 28, no. 6, pp. 1-37, 2023, doi: 10.1145/3610291.