ScTMR: A Scan Chain-Based Error Recovery Technique for TMR Systems in Safety-Critical Applications

Mojtaba Ebrahimi Seyed Ghassem Miremadi Hossein Asadi
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
Email: mojtaba_ebrahimi@ce.sharif.edu, {miremadi,asadi}@sharif.edu

***Abstract*—We propose a roll-forward error recovery technique based on multiple scan chains for TMR systems, called Scan chained TMR (ScTMR). ScTMR reuses the scan chain flip-flops employed for testability purposes to restore the correct state of a TMR system in the presence of transient or permanent errors. In the proposed ScTMR technique, we present a voter circuitry to locate the faulty module and a controller circuitry to restore the system to the fault-free state. As a case study, we have implemented the proposed ScTMR technique on an embedded processor, suited for safety-critical applications. Exhaustive fault injection experiments reveal that the proposed architecture has the error detection and recovery coverage of 100% with respect to Single Event Upset (SEU) while imposing a negligible area and performance overhead as compared to traditional TMR-based techniques.**

I. Introduction

The reliance of today's embedded processors on deep sub-micron technologies has brought serious reliability challenges due to soft errors in the safety-critical applications. Soft errors are caused by high energy particle strikes in sensitive regions of an electronic device. Traditionally, soft errors were regarded as the main reliability threat only for space applications, but due to technology down scaling toward nanometer era, these errors have become a crucial concern in ground-level applications as well [1].

To meet the reliability requirement of safety-critical applications, embedded processors should be equipped with error detection and correction mechanisms. In addition to the reliability requirement, performance is another important issue in these applications as most of them have real-time constraints. Thus, providing fault-tolerant techniques with minimum performance overhead in embedded processors is of decisive importance.

TMR is a well-known and widely used fault-tolerant technique. However, the traditional TMR is unable to recover the system state when one of its modules becomes faulty. A TMR fails if it does not properly obtain a majority decision among its modules outputs due to multiple faulty modules or a faulty voter [2]. Although the TMR failure rate due to coincident faults is usually small enough to be ignored in a well-manufactured system, independent fault arrivals in different modules is no longer negligible. Independent fault arrivals in two different modules may lead to a TMR failure if neither of the faults is overwritten by the system.

Without an appropriate recovery mechanism, there is a high probability of having TMR failures, when running an application for a long period of time [3], [4]. To address this issue, a transient error recovery technique can be employed along with TMR technique. Normally, an error recovery technique restores the state of the system to a fault-free state once an error is detected. A majority of previous error recovery techniques in TMR systems exploit *retry* mechanism [2], [4], [5], [6]. In a retry mechanism, once an error is detected, the faulty module will re-execute the entire process. Retry techniques are not suitable for tight deadline applications as they impose significant performance overhead to the system.

Rollback recovery is an alternate approach to recover from multiple fault arrivals. In rollback recovery techniques [7], the system state is backed up in some points during a program execution, which are called *checkpoints*. Whenever an error is detected, the system is restored to the previous fault-free checkpoint. Checkpointing and re-computation impose significant performance overhead which may violate the real-time requirements of safety-critical applications. In contrast, *roll-forward* recovery mechanisms are efficient to be used in tight deadline applications as they do not rely on re-computation. A roll-forward recovery technique has been presented in [3]. The proposed technique, however, would require detailed information about the module function and cannot be applied to general purpose circuits such as processors.

In this paper, we present a general purpose technique to provide recovery for both transient and permanent errors in TMR systems. Our recovery scheme, called ScTMR, uses a roll-forward approach to avoid re-computation and to meet tight deadlines in real-time systems. The proposed technique reuses the *scan chain* implemented in the processors for testability purposes to recover the system fault-free state. Compared to other TMR-based error detection and recovery techniques [2], [3], [4], [5], [6], ScTMR imposes a negligible area overhead to the system as it reuses the available resources within the chipset. Once a transient error is detected, the state of one of the fault-free modules is copied to the faulty module using the scan chains. As no re-computation is needed, ScTMR has lower performance overhead compared to roll-back recovery techniques. In case of permanent faults, the faulty module is disregarded and the system will be degraded to the *master/checker* (M/C) configuration.

The remaining of this paper is organized as follows. Sec. II introduces the ScTMR technique. Sec. III presents a case study in which the proposed ScTMR technique is implemented on an embedded processor. Sec. IV gives the experimental setup and the simulation results. Finally, Sec. V concludes the paper.

978-3-9810801-7-9/DATE11/©2011 EDAA

II. The Proposed Technique (ScTMR)

The ScTMR technique reuses scan chains for recovering the state of the faulty module. Scan chain [8] is a cost-effective technique used in *Design for Testability* (DFT) to provide a simple way for testing combinational and sequential circuits. In this technique, flip-flops are chained together through a long shift register circuit and a multiplexer is used in front of each flip-flop to switch between the normal and testing operations. In order to reduce the observation and loading time in large designs multiple scan chains are used. Multiple scan chains include parallel chains, which consist of approximately the same number of flip-flops. The number of flip-flops in each scan chain is called scan chain length (L_{sc}) and the number of scan chains in parallel is named scan chain width (W_{sc}).

Figure 1 shows the ScTMR block diagram. As shown in this figure, the ScTMR architecture consists of three redundant modules, a voter and a ScTMR controller. The proposed voter detects errors and reports them to the ScTMR controller. The ScTMR controller detects the error type and exploits an appropriate mechanism to remove the error effects from the system. The ScTMR controller then uses scan chains to copy the state of a fault-free module into the faulty module. The scan chain signals including *Scan Chain Input* (SCI), *Scan Chain Output* (SCO), and *Scan Chain Enable* (SCE) are controlled by the ScTMR controller.

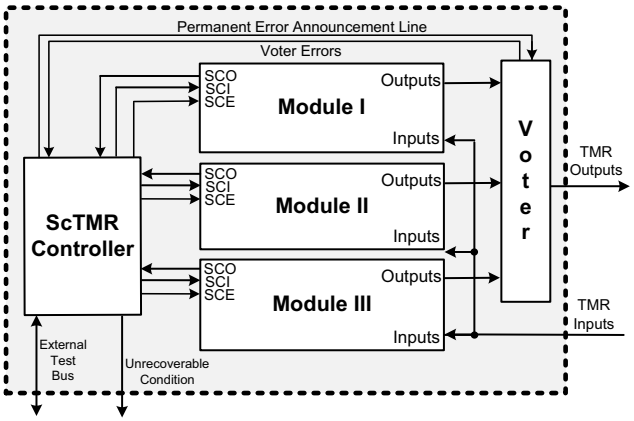

Fig. 1. ScTMR Technique Block Diagram

A. ScTMR Voter

The first concern in a highly reliable TMR system is finding the faulty module. To address this issue, we present a novel voter that has the capability of locating the faulty module. The proposed voter has the capability of detecting and locating faults within the comparators, too. Figure 2 shows the proposed voter architecture. In this voter, three comparators (C_{12}, C_{13}, and C_{23}) are used to compare the outputs of the TMR modules. Three error signals showing mismatch between TMR modules are generated accordingly (TE_{12}, TE_{13}, and TE_{23}). If one of the TMR modules becomes faulty and the fault is not overwritten within the module, an error is manifested to the output of the faulty module and the corresponding output becomes erroneous. Since the output of each module

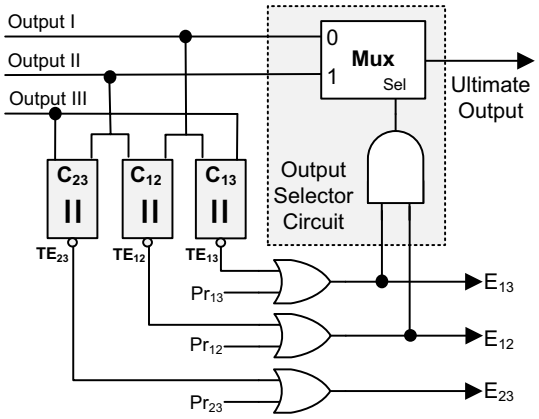

Fig. 2. ScTMR Voter

is compared by the outputs of the other modules, the error is detected by two comparators. As an example, if output I becomes erroneous, the error is detected by C_{12} and C_{13} and consequently both TE_{12} and TE_{13} are activated accordingly. On the other hand, if one of the comparators becomes faulty, only the corresponding error signal is activated.

The proposed voter takes three other input signals denoted with Pr_{12}, Pr_{13}, and Pr_{23}. These input signals are used to recover from permanent faults. These signals, which are derived by the ScTMR controller, are set to zero before a fault is identified as a permanent fault. In this case, E_{13}, E_{12}, and E_{23} are equal to TE_{13}, TE_{12}, and TE_{23}, respectively.

The error signals (E_{13}, E_{12}, and E_{23}) are connected to both the output selector circuit and the ScTMR controller. The output selector circuit, shown in Figure 2, selects the error-free output. The faulty module and the voter output are identified using different values of error signals according to Table I. As shown in this table, if either one of the comparators (C_{13}, C_{12}, and C_{23}), module II, or module III becomes faulty, output I is selected as the error-free output of the system. Otherwise, if module I becomes faulty, output II will be selected as the error-free output of the system. Therefore, the output selector circuit can be simply implemented by a 2-to-1 multiplexer whose selector signal can be designed as depicted in Figure 2. Note since we assume a single faulty module throughout this analysis, simultaneous activation of all error signals is illegal and is not supported by the proposed voter scheme. This has been shown as *Unrecoverable Condition* in the last row of Table I.

TABLE I
IDENTIFYING FAULTY MODULE AND SELECTING CORRECT VOTER OUTPUT USING ERROR SIGNALS

E_{12}	E_{13}	E_{23}	Faulty Module	Output
0	0	0	—	Output I
0	0	1	C_{23}	Output I
0	1	0	C_{13}	Output I
0	1	1	Module III	Output I
1	0	0	C_{12}	Output I
1	0	1	Module II	Output I
1	1	0	Module I	Output II
1	1	1	Unrecoverable	X

B. Transient and Permanent Error Recovery Mechanisms

The ScTMR controller is used for both transient and permanent fault recovery. As mentioned earlier in the previous subsection, when the voter detects an error, it activates an error signal to alert the ScTMR controller. Upon activation of the error signal, the ScTMR controller switches from the normal operation to the recovery mode to restore the state of the faulty module using the state of one of fault-free modules.

Figure 3 shows a simplified block diagram of the ScTMR controller circuit in the error recovery mode. In the recovery mode, the internal states of the two fault-free modules are shifted out using the scan chains and then the state of one of the fault-free modules is copied into the faulty module. For this purpose, the ScTMR controller enables scan chains of the ScTMR modules and configures the multiplexers as follows. The SCI signal of fault-free modules is connected to the SCO signal of the same module. Also, the SCI signal of the faulty module is connected to the SCO of one of the fault-free modules. This has been shown in Figure 3. Using this configuration, the state of one of the fault-free modules will be copied into the faulty module after L_{sc} clock cycles. Upon activation of the recovery mode, the counter within the ScTMR controller is loaded with L_{sc} and it starts counting down on each clock cycle. The recovery mode is terminated when the counter becomes zero.

While shifting out the states of the fault-free modules, both states are compared with each other to discover possible latent errors. A latent error is an error that changes the state of a module but the effect of the error has not manifested to the outputs of the module yet. Note the voter considers a module as a fault-free module in the presence of a latent error if the error is not manifested to the outputs of the module when voting is being performed. If the states of fault-free modules do not match, the ScTMR will indicate *Unrecoverable* since there would be two faulty modules within the system. Otherwise, the faulty module is recovered by the state of one of the fault-free modules and the system will be returned to its normal operation mode.

In order to distinguish between transient and permanent faults, the ScTMR controller uses two internal registers, called MRFM (*Most-Recent Faulty Module*) and NCF (*Number of Consecutive Faults*). MRFM records the most recent faulty module number (e.g., 1, 2 or 3). If a new faulty module is detected by the voter, the module number is compared with the previous module number saved in MRFM. If the faulty module is the same as the previous module, NCF is incremented. Otherwise, it is reset. If NCF exceeds a predefined number, MRFM is considered as permanent faulty module. Otherwise, the fault is considered as transient.

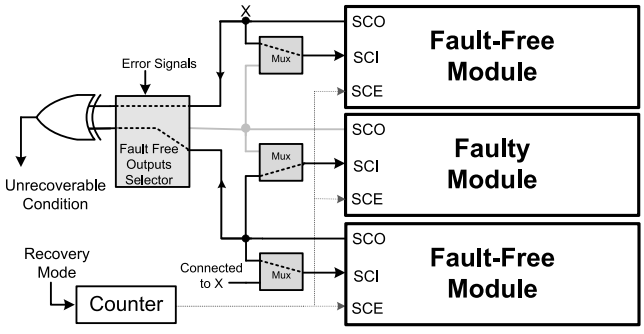

Fig. 3. Recovery Mode

In case a module is identified as permanent faulty module, the system is degraded to the M/C configuration by disregarding the module indicated by the MRFM register. For this purpose, the ScTMR controller enables the corresponding Pr error signals. As an example, if the controller detects a permanent fault within module I, it permanently activates Pr_{12} and Pr_{13} signals. This means that E_{12} and E_{13} will be always activated. According to Table I, the output of module II is selected as the voter output. Note in this example, hereafter module II and III will act as the master and the checker, respectively. In this configuration, C_{23} will compare the results of these two modules and any further mismatch will be reported to the ScTMR controller and will result into an unrecoverable condition.

III. A Case Study: A ScTMR Processor

The ScTMR technique can be applied to any arbitrary design that utilizes the scan chains for testability purpose. In this section, we apply our proposed ScTMR technique to a processor design to validate this technique. Since it is relatively easy to protect cache memory and communication interfaces using coding techniques such as parity or *Error Correction Codes* (ECC), we apply the ScTMR technique to the CPU core.

As a case study, we implement the proposed ScTMR technique on a Leon2 IP core [9]. Hereafter, the protected Leon2 processor using the ScTMR technique is called the *ScTMR processor*. Also, the CPU core excluding the register file is referred as the *CPU logic*. To implement the ScTMR processor, the CPU logic is protected by the proposed ScTMR technique.

The Leon2 processor implements the Harvard architecture with separate instruction and data caches. Both of these caches use write-through policy. Note a write-through cache is significantly more reliable than a write-back cache [10]. To protect a write-through cache against SEUs, we have exploited parity bit scheme in the ScTMR processor. In case an error is detected on a cache read, the fault-free block can be retrieved from the main memory.

To protect the register file, we use *Single Error Correction Double Error Detection* (SEC-DED) code. Since registers are 32-bit width, seven additional bits for SEU detection and correction are needed. The ScTMR controller is also protected against SEUs using the flip-flop triplication technique presented in [11].

IV. Experimental Results

We have implemented the ScTMR processor by modifying the original VHDL code of Leon2 as follows. First we add

parity and SEC-DED coding techniques to protect the caches and the register file, respectively. Then we synthesize the Leon2 CPU logic using Synopsys Design Compiler® [12], which has the capability of adding DFT components in digital circuits. Multiple scan chains are also added to the CPU logic flip-flops using this toolset. Each module of the ScTMR processor includes 2096 flip-flops. We use multiple scan chains with a width equal to 16 (W_{sc}=16) and a length equal to 131 (L_{sc}=131). The ScTMR technique is then implemented using three redundant CPU logic cores with multiple scan chains, a ScTMR controller unit, and a voter.

To extract the area overhead, we used Synopsys Design compiler® [12] and UMC Memory Maker® [13] toolsets. The results of area overheads for Leon2 and the ScTMR processor are reported in Table II. Experemental results show that the area used by the CPU logic, voter, and the ScTMR controller in the SCTMR processor is increased from (1.785+0.02) mm^2 in the traditional TMR to (1.785 + 0.036) mm^2 in the ScTMR processor. Hence, the proposed ScTMR technique has less than 2% area overhead compared to the traditional TMR system. The total area overhead of the ScTMR processor compared to the unprotected Leon2 processor is about 71.7%, which is significantly less than the area overhead of the TMR implementation. The smaller area overhead of the ScTMR processor is also an indicative of lower power consumption of the ScTMR processor compared to the TMR implementation.

TABLE II
AREA OVERHEAD (mm^2)

Architecture	Leon2	TMR	ScTMR
Cache & Register File	1.197	3.591	1.256
CPU Logic	0.595	1.785	1.785
Voter, Controller	—	0.002	0.036
Total Area	1.792	5.378	3.077
Total Area Overhead	—	200.0%	71.7%

In our simulation-based fault injection experiments, we use SEU as our major fault model. Some programs of MiBench automotive benchmarks [14] including *Bitcount*, *Basicmath* and *Qsort*, were used as our benchmark programs during fault injection experiments. Since fault injection experiments are very time-consuming, we have limited the benchmark programs execution to approximately 1M clock cycles to have tractable experiments in this case study. A total of 12,000 fault injection iterations have been performed on each architecture (Leon2 and the ScTMR Processor). Our results indicate that all injected faults into the CPU logic of the ScTMR processor are either overwritten or successfully corrected using the proposed ScTMR technique. The results also show that 45.9% of faults are latent in the TMR implementation. In other words, 45.9% of injected faults are masked by the TMR voter but could not be corrected. The latent faults may lead to a TMR failure if another fault occurs within the TMR modules during program execution. Lastly, the fault injection experiments indicate that 9.5% of faults injected into the Leon2 CPU logic result in the system failure while no failure has been observed in the TMR and the ScTMR processor.

The performance overhead of the proposed technique has been measured in the ScTMR processor. Two important factors that influence the performance of an ScTMR-based system are clock cycle increase due to critical path delay and time required for error recovery. Our simulation results show that the ScTMR processor critical path delay is increased by 0.89% compared to the critical path delay of Leon2. Therefore, the minimum clock cycle is only increased by 0.89%. Recovery time (T_R) in the ScTMR technique depends on the number of flip-flops in each scan chain (L_{sc}) and the clock period (T_p) and it can be calculated as: $T_R = L_{SC} \times T_p$. In our simulations L_{SC} is equal to 131. So, it takes 131 clock cycles to recover from a transient error.

V. CONCLUSIONS

In this paper, we proposed a roll-forward error recovery technique based on multiple scan chains for TMR systems. The proposed technique, called ScTMR, reuses the scan chain flip-flops employed for testability purposes to restore the correct state of a TMR system in the presence of soft errors. The fault injection experiment results revealed that the ScTMR technique has the error detection and recovery coverage of 100% with respect to the SEU fault model while offering the benefits of low area (less than 2%) and performance (less than 1%) overheads compared to the traditional TMR systems.

REFERENCES

[1] F. Wrobel, F. Saigne, M. Gedion, J. Gasiot, and R. Schrimpf, "Radioactive nuclei induced soft errors at ground level," *IEEE Transactions on Nuclear Science*, vol. 56, no. 6, pp. 3437–3441, dec. 2009.

[2] H. Kim and K. Shin, "Design and analysis of an optimal instruction-retry policy for TMR controller computers," *IEEE Transactions on Computers*, vol. 45, no. 11, pp. 1217–1225, nov. 1996.

[3] S. Yu and E. McCluskey, "On-line testing and recovery in TMR systems for real-time applications," in *Proceedings of International Test Conference*, 2001, pp. 240–249.

[4] K. Shin and H. Kim, "A time redundancy approach to TMR failures using fault-state likelihoods," *IEEE Transactions on Computers*, vol. 43, no. 10, pp. 1151–1162, oct. 1994.

[5] A. Hopkins, T. Smith, and J. Lala, "FTMP: A highly reliable fault-tolerant multiprocess for aircraft," *Proceedings of the IEEE*, vol. 66, no. 10, pp. 1221–1239, oct. 1978.

[6] S. D'Angelo, C. Metra, and G. Sechi, "Transient and permanent fault diagnosis for FPGA-based TMR systems," in *Proceedings of International Symposium on Defect and Fault Tolerance in VLSI Systems*, nov. 1999, pp. 330–338.

[7] D. Pradhan and N. Vaidya, "Roll-forward and rollback recovery: performance-reliability trade-off," in *Proceedings of 24th International Symposium on Fault-Tolerant Computing*, jun. 1994, pp. 186–195.

[8] L. Wang, C. Wu, and X. Wen, *VLSI Test Principles and Architectures: Design for Testability*. Morgan Kaufmann Publishers Inc., 2006.

[9] "The Leon2 Processor User Manual, http://www.gaisler.com," 2007.

[10] H. Asadi, V. Sridharan, M. B. Tahoori, and D. Kaeli, "Balancing performance and reliability in the memory hierarchy," in *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software*, mar. 2005, pp. 269–279.

[11] J. Gaisler, "A portable and fault-tolerant microprocessor based on the SPARC v8 architecture," in *Proceedings of International Conference on Dependable Systems and Networks*, 2002, pp. 409–415.

[12] "Synopsys Design Compiler, www.synopsys.com," 2010.

[13] "UMC Memory Maker, www.umc.com," 2010.

[14] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of IEEE International Workshop on Workload Characterization*, dec. 2001, pp. 3–14.