Low-Cost Scan-Chain-Based Technique to Recover Multiple Errors in TMR Systems

Mojtaba Ebrahimi, *Student Member, IEEE*, Seyed Ghassem Miremadi, *Senior Member, IEEE*, Hossein Asadi, *Member, IEEE*, and Mahdi Fazeli, *Student Member, IEEE*

***Abstract*— In this paper, we present a scan-chain-based multiple error recovery technique for triple modular redundancy (TMR) systems (SMERTMR). The proposed technique reuses scan-chain flip-flops fabricated for testability purposes to detect and correct faulty modules in the presence of single or multiple transient faults. In the proposed technique, the manifested errors are detected at the modules' outputs, while the latent faults are detected by comparing the internal states of the TMR modules. Upon detection of any mismatch, the faulty modules are located and the state of a fault-free module is copied into the faulty modules. In case of detecting a permanent fault, the system is degraded to a master/checker configuration by disregarding the faulty module. FPGA-based fault injection experiments reveal that SMERTMR has the error detection and recovery coverage of 100% and 99.7% in the presence of single and two faulty modules, respectively, while imposing negligible area and performance overheads on the traditional TMR systems.**

***Index Terms*— Fault-tolerant design, roll-forward error recovery, scan chain, triple modular redundancy (TMR).**

I. Introduction

TODAY, the use of embedded systems in safety-critical applications such as avionics, process control, and patient life-support monitoring has become a common trend [1], [2]. Such a system often has both timing constraints and fault-tolerance requirements. To meet the reliability requirement, such embedded systems should be equipped with appropriate error detection and correction mechanisms. However, achieving a high level of reliability and meeting the timing requirements are conflicting objectives, i.e., the reliability enhancement may have a negative impact on timing constraints. For example, in a rollback recovery-based system, the overall reliability is improved; however, since the expected response time increases, the probability of missing deadlines also increases for certain applications. Generally, improving system reliability without considering its real-time constraints is not justifiable for safety-critical applications. Consequently, providing fault-tolerant techniques with minimum performance overhead in embedded processors is of decisive importance.

Manuscript received October 19, 2011; revised May 26, 2012; accepted July 21, 2012. Date of publication September 19, 2012; date of current version July 22, 2013.

The authors are with the Department of Computer Engineering, Sharif University of Technology, Tehran 11155-9517, Iran (e-mail: mojtaba_ebrahimi@ce.sharif.edu; miremadi@sharif.edu; asadi@sharif.edu; m_fazeli@ce.sharif.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2012.2213102

One of the well-known and widely used fault-tolerant techniques in safety-critical applications is triple modular redundancy (TMR) [3], [4]. A traditional TMR system consisting of three redundant modules and a voter at the modules outputs has some shortcomings that should be addressed in order to be employed in safety-critical applications. A major shortcoming of the traditional TMR is its inability to cope with TMR failures. TMR failure refers to a failure in a TMR system caused by multiple faulty modules or a faulty voter [5]. Although the probability that two particles hitting two replica flip-flops in a TMR system is very low, the probability that two energetic particles hitting two modules of a TMR system is not very rare when the system is running in a harsh environment for long periods. In case of independent fault arrivals in two different modules, if neither of the faults is overwritten, it may result in a TMR failure. For long-term applications, the absence of appropriate recovery mechanisms significantly increases the probability of TMR failure occurrence [6], [7]. To address this issue, TMR should be equipped with a transient error recovery technique. Most of the previous TMR-based error recovery techniques proposed so far exploit retry mechanisms [1], [5], [7]–[9]. These techniques, however, are not suitable for tight deadline applications, as the recomputation may result in a task completion after its deadline [10].

In contrast to retry based error recovery mechanisms, roll-forward recovery mechanisms are efficient to be used in tight deadline applications as no recomputation is needed. A roll-forward recovery technique for TMR-based systems has been proposed in [6]. This technique, however, is not applicable for general-purpose circuits such as processors, as it requires detailed information about the function of all registers of TMR modules. A TMR-based technique applicable to general-purpose circuits has been proposed in [11]. The proposed technique, called ScTMR, provides recovery for both transient and permanent errors in TMR systems [11]. ScTMR uses a roll-forward approach and employs the scan chain implemented in the circuits for testability purposes to recover the system fault-free state. Although ScTMR significantly reduces the probability of TMR failures, it suffers from two major shortcomings. First, ScTMR cannot recover a single faulty module in the TMR system in the presence of latent faults. A fault is referred to as latent if it is not propagated to the system outputs but does cause a mismatch between the states of the TMR modules. Note that, in the presence of a mismatch between the states of the TMR modules, once an error is detected at the output of either of the modules, the system will fail to restore its fault-free state.

Second, ScTMR is unable to recover the system if multiple faults are simultaneously manifested to the outputs of two modules.

In this paper, we present a scan-chain-based roll-forward error recovery technique for TMR-based systems, which addresses the shortcomings of ScTMR. The proposed technique, called scan chain-based multiple error recovery TMR (SMERTMR), has the ability to locate and remove latent faults in TMR modules as well as to recover the system from multiple faults affecting two TMR modules. To the best of our knowledge, SMERTMR is the first roll-forward error recovery technique for a TMR-based system that has the capability of error recovery in the presence of multiple latent faults as well as two faulty modules. The main idea behind SMERTMR is to reuse the available scan chains devoted for testability purposes in order to compare the internal states of TMR modules to locate and restore the correct state of faulty modules using the state of nonfaulty modules. In case of permanent faults, the faulty module is disregarded and the system is degraded to a master/checker (M/C) configuration. Nevertheless, the offline testability characteristics of the system are preserved. As compared to other TMR-based recovery techniques, SMERTMR has negligible area overhead, as it reuses the available resources within the circuit.

The SMERTMR technique has been analytically and experimentally evaluated and compared with the state-of-the-art techniques. As a case study, the proposed technique has been implemented on the Leon2 processor [12]. The proposed analytical study shows that, in the presence of multiple errors, SMERTMR improves the reliability of TMR systems up to five orders of magnitude as compared to the recently suggested techniques. Additionally, the results of FPGA-based fault injection experiments demonstrate that SMERTMR can detect and correct 100% and 99.7% of multiple faults affecting single and two faulty modules, respectively.

The rest of this paper is organized as follows. Section II describes related work. Section III reviews the ScTMR technique. Section IV presents our proposed SMERTMR architecture. Section V presents a reliable processor design using the proposed SMERTMR technique and provides experimental results. Section VI evaluates the SMERTMR technique using an analytical study. Finally, Section VII concludes this paper.

II. Related Work

Traditional TMR voter masks the faults affecting only one module. In addition, the faulty module cannot be recovered in a traditional TMR system, as the system cannot identify the faulty module. The techniques proposed in [1], [5]–[8], and [13] use modified voters to diagnose the faulty module. The voters presented in [1], [5]–[7], and [13] are hardware-based, while the technique proposed in [8] uses a software-based method for voting and fault diagnosis resulting in negative impact on the system performance. Some of these voters [1], [5], [7], [8], [13] keep the history of faulty modules and, whenever the number of consecutive recovery operations caused by one module exceeds a predefined number, the error is then identified as a permanent error.

Transient and permanent errors in a voter can be masked by employing multiple voters and a disagreement detector [1], [7], [9]. A disagreement detector that compares the values from different voters of a TMR system can detect a single fault, but a faulty detector circuit may lead to failure.

Most of the previous work [1], [5], [7]–[9] has used retry mechanisms to recover from transient errors in TMR systems. In a retry mechanism, once an error is detected, the faulty module will re-execute the entire process. Retry mechanisms impose significant performance overhead to the system and cannot be used in tight deadline applications. An analytical study presented in [10] compares the traditional TMR and TMR with roll-forward, or retry mechanism. The study reveals that roll-forward mechanisms have lower performance overhead and are more reliable than retry mechanisms. In a roll-forward recovery, unlike retry recovery schemes, the correct state is copied from a fault-free redundant module to the faulty module to avoid recomputation.

A transient error roll-forward recovery method for TMR systems has been proposed in [6]. In this method, the system registers are divided into different categories based on the importance of their values before and after checkpoints. Upon detection of an error, the error recovery mechanism copies the voted value of three corresponding registers into the faulty register. The main shortcoming of the method is that its implementation would require detailed information about the module function and cannot be applied to general-purpose systems such as processors.

To address the issue of latent faults in TMR systems, several methods have been proposed suggesting TMR partitioning and voter insertion techniques [4], [14]–[17]. In these methods, a system is partitioned into several blocks and then each block is protected using the TMR technique. Increasing the number of voters in the circuit will reduce the average latency of fault masking [4]. In the technique presented in [14], voters are inserted at the output of each flip-flop and, consequently, a fault occurring inside a flip-flop will be masked in the next clock cycle. Hence, this technique reduces the fault-masking latency to one clock cycle. However, such a technique has several shortcomings. First, protecting a circuit including thousands of flip-flops using such voter insertion scheme imposes significant area overhead to the circuit. Second, inserting a voter right after each flip-flip will increase the delay of the critical path, leading to increased performance overhead. Third, combinational logic used for voter insertion circuit increases the susceptibility of the system to single-event transients.

A method for diagnosing permanent faults in TMR systems with spare modules has been proposed in [9]. The authors propose the testing of all possible combinations of three modules to find the faulty module. Another technique for detecting permanent faults using dual modular redundancy (DMR) has been presented in [18]. In this technique, each module of the system runs in the DMR mode with a spare for a small time interval in order to detect possible permanent faults.

Detect–diagnose–reconfigure is a recovery method for handling permanent faults in TMR systems [1], [5], [7], [9]. In this method, after detecting a permanent error, the system will diagnose the faulty module and will replace it with a spare module. However, this method cannot be used in systems

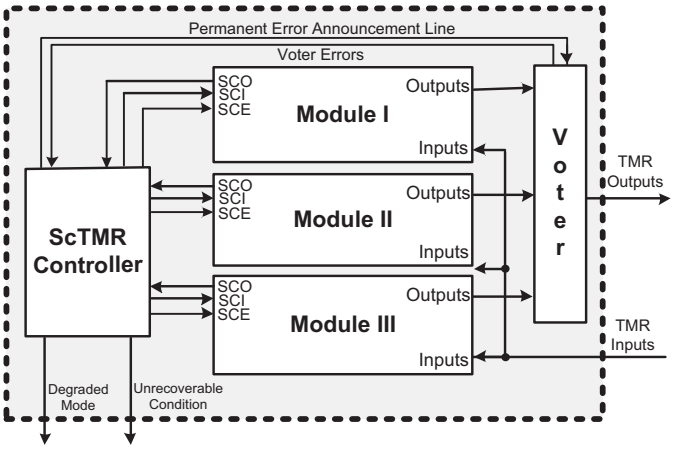

Fig. 1. ScTMR block diagram [11].

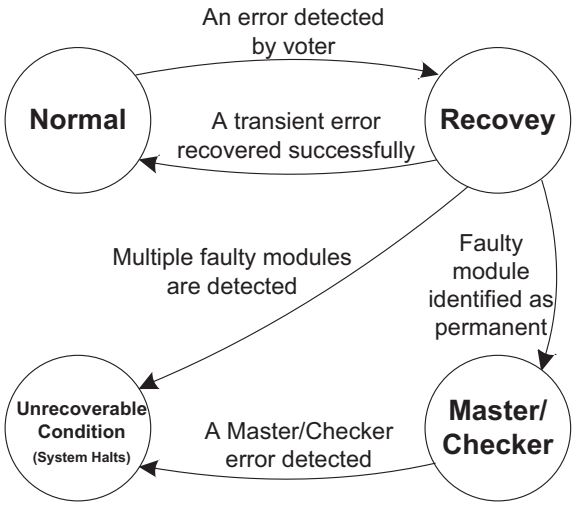

Fig. 2. ScTMR state diagram.

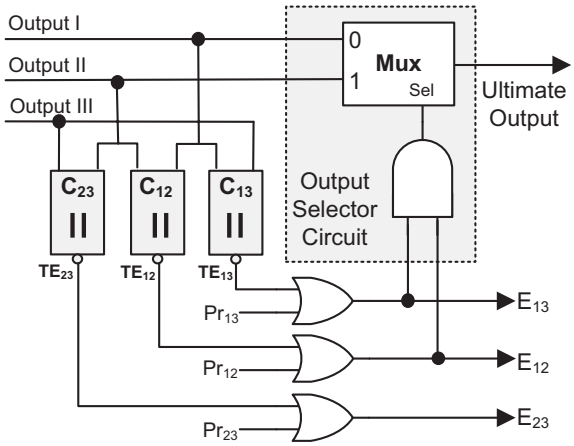

Fig. 3. Proposed voter [11].

without spare modules. To address this issue, a technique has been proposed in [8] to degrade a TMR system to an M/C system in case of having one permanent faulty module.

There are also several methods suggesting the protection of a system at the circuit level [19]–[22]. The area and power overheads imposed by circuit-level techniques are typically much lower than those of modular redundancy techniques; however, such techniques only minimize the vulnerability of the system against soft errors and do not completely remove the effect of all types of soft errors affecting combinational and sequential logic. Additionally, employing circuit-level methods are highly technology-dependent and should be integrated in the design flow of digital systems.

III. Overview of the ScTMR Technique

Fig. 1 shows the block diagram of the ScTMR technique. As shown in this figure, the ScTMR includes: 1) three redundant modules; 2) a voter; and 3) a controller. In this architecture, once an error is detected by the voter, the ScTMR controller identifies the error type (transient or permanent) and triggers an appropriate recovery mechanism to eliminate the error from the system. This is achieved by copying the state of a fault-free module to the detected faulty module using the scan-chain circuitry. The recovery process is done through the scan-chain input (SCI), scan-chain output (SCO), and scan-chain enable signals instructed by the ScTMR controller.

Fig. 2 shows the state diagram of a system protected by the ScTMR technique. Primarily, the system is in the normal state and, upon detection of an error by the voter, the recovery process is initiated. During the recovery process, the ScTMR controller detects the faulty module as well as the fault type (permanent or transient). If a permanent fault is detected in one of the modules, the system is degraded to a M/C configuration. In case of detecting a transient fault, the recovery process is performed to bring the system to the fault-free, i.e, the normal, state. Upon detection of multiple transient faults, the recovery process is terminated and the system will be halted immediately to provide a fail-safe state.

A. Proposed Voter

In a TMR system, detection and correction of a faulty module is a challenging issue and is still an ongoing research topic [18], [23], [24]. In particular, a wrong detection or inability to locate the faulty module can significantly affect the system reliability. To address this issue, we have presented a voter that can identify the faulty module. Additionally, the proposed voter can also detect possible faults occurring in the comparators. The proposed voter can be used in both ScTMR and SMERTMR techniques. The architecture of the proposed voter is depicted in Fig. 3. As shown in the figure, three comparators (C_{12}, C_{13}, and C_{23}) are used to represent any mismatch between TMR modules. As an example, TE_{23} signal is activated once a mismatch between Outputs II and III is detected. If one of the modules generates an erroneous output (e.g., Output I), two of the comparators (here, C_{12} and C_{13}) will activate the mismatch signals (here, TE_{12} and TE_{13}) and only one of the comparators (here, C_{23}) will not activate the corresponding mismatch signal (here, TE_{23}). In case of a faulty comparator (e.g., C_{13}), only the corresponding signal (here, TE_{13}) is activated and the other signals (here, TE_{12} and TE_{23}) are deactivated.

This voter can also detect and recover from permanent faults. In order to detect permanent faults, the proposed voter employs three input signals (named Pr_{12}, Pr_{13}, and Pr_{23}), which are derived by the ScTMR controller. In the normal state and during transient error recovery process, these three

TABLE I
IDENTIFYING FAULTY MODULE AND SELECTING CORRECT VOTER OUTPUT USING ERROR SIGNALS [11]

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

signals are deactivated ($Pr_{12} = Pr_{13} = Pr_{23} = 0$) and as a result, the values of E_{12}, E_{13}, and E_{23} become equal to TE_{12}, TE_{13}, and TE_{23}, respectively. Upon detection of a permanent fault, Pr_{12}, Pr_{13}, and Pr_{23} will be activated by the ScTMR controller as will be detailed in Section III-B.

In the proposed voter, an output selector circuit is used to route the error-free output to the ultimate output signal. As shown in Fig. 3, the output selector circuit uses E_{12} and E_{13} signals as inputs of a logical AND gate to generate the select signal for a 2×1 multiplexer. The value of error signals shown in Table I identifies the faulty module or faulty component and selects the correct voter output. For instance, if $E_{12}E_{13}E_{23} = 101$, module II is identified as the faulty module and Output I is selected as an error-free output. Briefly, according to Table I, if one of the comparators, module II, or module III becomes faulty, the output of module I is selected as the error-free output. If module I becomes faulty, Output II will be selected as the error-free output. Based on this specification, the output selector can be implemented by a 2×1 multiplexer.

B. Transient and Permanent Error Recovery Mechanisms

The ScTMR controller is used for both transient and permanent fault recovery. As mentioned in Section III-A, once an error is detected by the voter, it alerts the ScTMR controller using an error signal. The ScTMR controller then changes the system state from the normal operation mode to the recovery mode to restore the correct state of the system using the states of fault-free modules. Fig. 4 shows a simple block diagram of the ScTMR controller configuration when it is in the recovery mode. During the recovery process, the flip-flop values of fault-free modules are shifted out using the scan chain and are copied to the corresponding flip-flops in the faulty module. To do this, the ScTMR controller enables modules scan chains and configures the multiplexers as follows: 1) the SCI signal of each fault-free module is connected to its SCO and 2) the SCI signal of faulty module is connected to the SCO of one of the fault-free module (see Fig. 4). Using this configuration, the state of a fault-free module can be copied into the state of the faulty module after shifting the fault-free module by L_{sc} clock cycles (L_{sc} is the length of the scan chains). While shifting the flip-flop values, a counter is used to enumerate the number of clock cycles. Upon activation of the recovery mode, the counter is first loaded by L_{sc}. Afterwards, the counter is decremented by one at each clock cycle. Once the counter reaches zero, the recovery process will be completed.

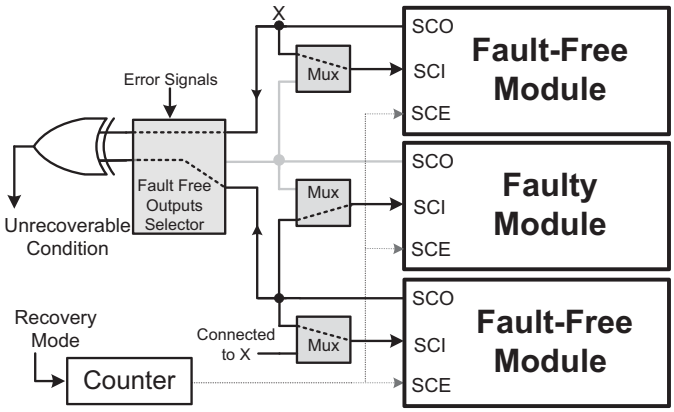

Fig. 4. ScTMR in recovery mode [11].

ScTMR has also the ability of distinguishing between permanent and transient faults. ScTMR employs two internal registers named most recent faulty module (MRFM) and number of consecutive faults (NCF). MRFM holds the faulty module number detected most recently. As an example, if module II becomes faulty, MRFM is equal to 2. Upon detection of another faulty module, the faulty module number is compared with the previous faulty module number stored in MRFM. If these two numbers are equal, the ScTMR controller increments the NCF register by 1; otherwise, it resets the NCF. Whenever, the value of NCF exceeds a predefined threshold value, the module is considered as a permanently faulty module. In this case, the faulty module is disregarded and the system is degraded to an M/C configuration. For example, if a permanent fault is detected within module I, Pr_{12} and Pr_{13} will be permanently activated by the ScTMR controller and consequently E_{12} and E_{13} signals will be activated as well. In this case, based on Table I, the output of module II will be routed to the voter output. From this time on, modules II and III will act as an M/C system and C_{23} will be responsible for comparing the outputs of these two modules. In the M/C configuration, any mismatch will result in an unrecoverable condition.

IV. ARCHITECTURE OF SMERTMR

The ScTMR technique introduced in the previous section can recover from a transient fault only if it manifests in the module outputs. This is because only modules outputs are compared by the voter. As will be shown in the experimental results, it is quite likely that a fault remains latent in a module for a long time without propagating to the module outputs. During this period, it is likely that a second fault occurs in the other modules, resulting in an unrecoverable condition.

In the ScTMR technique, if a fault occurs in one of the TMR modules while there is a latent fault in the other modules, ScTMR fails to recover the correct state of the system because of having two faulty modules. In this case, during the recovery operation, ScTMR detects that there are two faulty modules and enters the unrecoverable condition. In all previous studies, it has been assumed that a fault occurring in one of the modules immediately manifests itself in the module outputs, i.e., the time between a fault occurrence and its manifestation in the module outputs is small enough that the probability of having a second fault in the other modules during this time interval can be neglected. However, our fault injection

results show that a remarkable percentage of faults remain latent for a long period. Therefore, latent faults should be considered during the design of fault-tolerant systems. To this end, we have enhanced the ScTMR technique by employing a comparison mode in order to locate latent faults. In the comparison mode, the internal state of each TMR module is compared with the other modules in order to extract the number of mismatches for each comparison pair. Using the results of pair comparisons, faulty modules will be detected and located. Unlike the ScTMR technique that only compares the modules outputs, the SMERTMR technique compares the internal states of the modules to detect possible latent faults.

In the SMERTMR technique, the comparison mode is activated in two cases: 1) when an error is detected by the voter and 2) when the checkpoint signal is activated. In the latter case, the checkpoint signal is employed to intentionally trigger the comparison mode in order to eliminate latent faults. The checkpoint signal can be activated during slack times; however, if there is not enough slack time, the comparison mode is activated only in the former case, i.e., the comparison mode is activated once a fault in a module propagates to the module outputs and is detected by the voter. Therefore, when the comparison mode is not regularly activated by the checkpoint signal, latent faults are detected and located once a second fault is propagated to the module outputs and detected by the voter. This can increase the probability of having multiple faulty modules since there is more opportunity for the second fault to occur. To reduce the delay of locating latent faults, the comparison mode can regularly be triggered by activating the checkpoint signal in predefined time intervals. It should be noted that having slack time is quite common in real-time embedded systems. In SMERTMR, we have exploited the available slack time in such systems to increase the reliability of the SMERTMR system.

As mention earlier, in the SMERTMR technique the internal states of TMR modules are compared together, while in the ScTMR technique only the module outputs are compared by the voter. This results in the following advantages for the SMERTMR technique.

1) If there is enough slack time to regularly perform the comparison process, the probability of having multiple faulty modules is significantly reduced.
2) Two faulty modules can be efficiently detected and located. Therefore, the faulty modules can be recovered using the states of the fault-free module.

Fig. 5 shows the state diagram of SMERTMR. In the normal mode, upon detection of an error by the voter or activation of the checkpoint signal, the system switches to the comparison mode. In the comparison mode, the internal states of the TMR modules are compared with each other to locate faulty modules and to determine the fault type. If no mismatch is found between all comparison pairs of the modules, the system returns to its normal mode. Otherwise, the system switches to the recovery mode and the recovery process is started. Finally, if the recovery process finishes successfully, the system continues to operate in the normal mode. Otherwise, it enters the unrecoverable condition state resulting in a system halt. SMERTMR can also detect permanent faults in one module during the comparison mode. If it detects a permanent fault, the system enters the M/C mode. In the M/C mode, any fault in the master or the checker modules results in an unrecoverable condition. In this case, other methods such as functional testing could be exploited to locate and identify the faulty module.

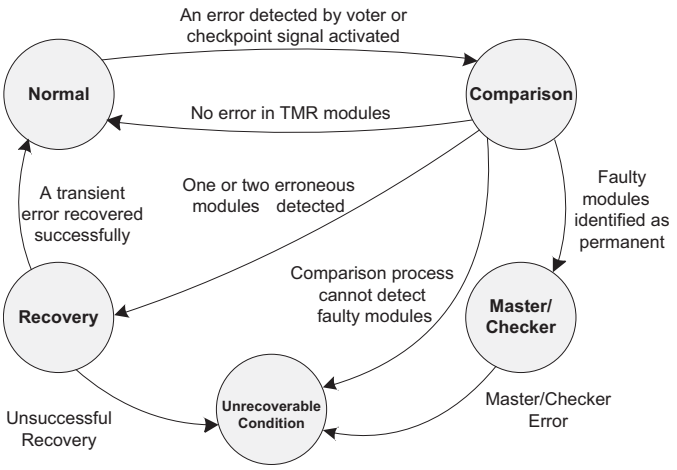

Fig. 5. SMERTMR state diagram.

The voter used in the SMERTMR technique is similar to that explained in Section III-A. In the following subsections, we will explain the architecture of SMERTMR in detail.

A. Comparison Process

In the SMERTMR technique, whenever the voter detects an error, it activates an error signal to alert the SMERTMR controller. Upon activation of the error signal, the SMERTMR controller switches from the normal operation to the comparison mode to locate the faulty modules. After locating the faulty modules, SMERTMR switches to the recovery mode to recover the faulty modules using the state of one of the fault-free modules.

Fig. 6 shows a simplified block diagram of the SMERTMR controller circuit working in the comparison mode. In this mode, the internal states of all TMR modules are shifted out using the scan chains and all module pairs (I/II, I/III, and II/III) are compared. As shown in Fig. 6, there are three counters, namely, counter_{12}, counter_{13}, and counter_{23}, to store the number of mismatches between each module pairs. For example, counter_{12} stores the number of mismatches between modules I and II. To this end, the SMERTMR controller enables scan chains of the SMERTMR modules and configures the multiplexers in such a way that the SCO signal in each module is connected to the SCI signal of the same module. During the shift operation, the internal states of the modules are compared using XOR gates. Whenever a mismatch is detected, the corresponding counter is incremented by one unit. Using this configuration, counter_{ij} will contain the number of mismatches between modules i and j after L_{sc} clock cycles.

In the following, we will show how an SMERTMR system can detect and locate faulty modules by comparing the internal states of the system modules. Suppose that there is an SMERTMR system including three modules named i, j, and k. Basically, the system may be in the following four situations.

1) All modules are fault-free: In this case, all three counters will be equal to zero.

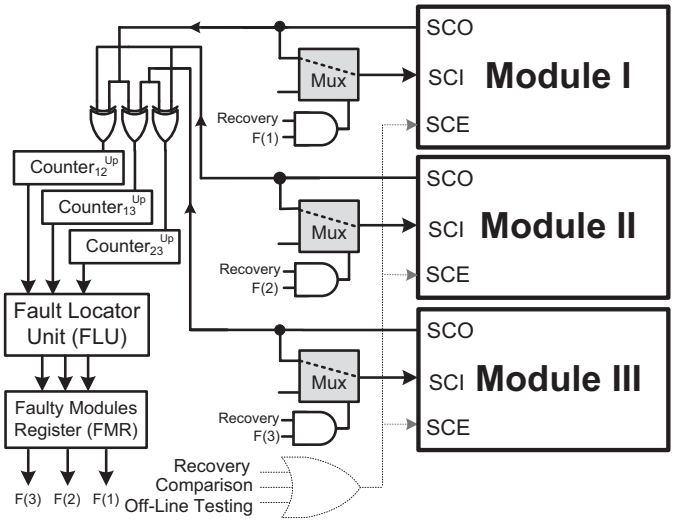

Fig. 6. SMERTMR in comparison mode.

2) There is only one faulty module: Let us assume that module i is faulty and it contains x erroneous flip-flops and the other modules, i.e., modules j and k are fault-free. In this case, we will have $\text{counter}_{ij} = \text{counter}_{ik} = x$. Note that, since both modules j and k are fault-free, counter_{jk} will be equal to zero (i.e., $\text{counter}_{jk} = 0$). After extracting the number of mismatches, the system enters the recovery mode and the state of module i is recovered using the state of either modules j or k.
3) There are two faulty modules: Suppose that there are two faulty modules (e.g., modules i and j) and one fault-free module (here, module k). Let A and B be the sets of erroneous flip-flops in modules i and j, respectively. Here, the faulty modules may have either no common erroneous flip-flops ($A \cap B = \emptyset$) or at least one common erroneous flip-flop ($A \cap B \neq \emptyset$). Assume that the number of erroneous flip-flops in modules i and j are denoted with x and y, respectively. In case $A \cap B = \emptyset$, $\text{counter}_{ik} = x$, $\text{counter}_{jk} = y$, and $\text{counter}_{ij} = x + y$. In case $A \cap B \neq \emptyset$, the counters will have the following values: $\text{counter}_{ik} = x$, $\text{counter}_{jk} = y$, and $0 < \text{counter}_{ij} < x + y$. By comparing the values of the counters, SMERTMR can effectively detect and locate the faulty modules when there is no common erroneous flip-flop in the faulty modules ($A \cap B = \emptyset$). In the latter case, where there is at least one common erroneous flip-flop ($A \cap B \neq \emptyset$), SMERTMR is not able to locate the faulty modules.
   This is because this case is not distinguishable from a case in which there are three faulty modules or in which there are two faulty modules with common flip-flops. For further clarity, the following example demonstrates that the number of mismatches in case of two and three faulty flip-flops are the same and not distinguishable. Suppose that there are 10 flip-flops in each processor core ($FF0 - FF9$). Also, let us assume we have the following cases (case A and case B). In case A, the first core has four erroneous FFs: $FF1$, $FF2$, $FF3$, and $FF4$; the second core does not have any erroneous FF and the third core has an error in $FF2$, $FF3$, and $FF7$. In this case, $\text{Counter}_{12} = 4$, $\text{Counter}_{23} = 3$, and $\text{Counter}_{13} = 3$. In case B, the first core has four erroneous FFs: $FF1$, $FF2$, $FF3$, and $FF4$; the second core has an error in $FF2$ and $FF8$ and the third core has an error in $FF2$, $FF3$, and $FF7$. Note in these two cases, we will have $\text{Counter}_{13} < \text{Counter}_{12} + \text{Counter}_{23}$. As demonstrated in this example, in the case of having overlapping faults between modules, just having the number of flip-flop mismatches between three modules is not sufficient to locate the faulty modules. However, it should be noted that the probability of having common erroneous flip-flops in two modules is very low. Our fault injection experiments will prove this claim, as will be detailed in Section V-D.
4) All modules are faulty: In this case, SMERTMR is not able to locate the faulty modules and it enters the unrecoverable condition.

In the SMERTMR technique, upon completion of the comparison mode, the fault locater unit (FLU) will determine the faulty modules. Algorithm 1 outlines how faulty modules are detected by the FLU. As can be seen, if all counters become zero, there is no faulty module and consequently the system returns to its normal mode. The condition statement in line 3 checks the existence of one faulty module. As discussed earlier, if there is only one faulty module, two out of three counters will have the same non-zero value while the third counter will be equal to zero. The condition statement in line 6 checks the existence of two faulty modules with no common faulty flip-flops. In the last two cases, the system enters the recovery mode to restore the correct state of the faulty modules using the state of the fault-free modules. If none of the previous conditions is valid, the system enters the unrecoverable condition (line 10). The FLU stores the faulty module numbers in a register named the faulty modules register (FMR) (see Fig. 6). As an example, if FMR is equal to 110, it means that modules I and II are faulty. This information is used by the SMERTMR controller during the recovery mode.

It is worth mentioning that in SMERTMR, instead of directly comparing and voting the output of the three scan chains, we first make sure that we have correctly identified the fault-free module. If one directly compares and votes the outputs of the three scan chains, it is possible that two out of three replica flip-flops are erroneous and a wrong state is written back to all three modules. In this case, the system will continue to work in a wrong state. Such a condition is not acceptable in safety-critical applications.

B. Transient and Permanent Error Recovery Mechanisms

After the identification of fault-free and faulty modules by the FLU unit at the end of the comparison process, the system enters the recovery mode if there is one or two faulty modules in the system. Otherwise, it returns to the normal mode. In the recovery mode, the state of the faulty module is recovered by the state of fault-free modules using the employed scan chains. Fig. 7 shows a simplified block diagram of the SMERTMR controller circuit in the recovery

Algorithm 1 Faulty Modules Detection Algorithm

1: **if** $Cntr_{ij} = Cntr_{ik} = Cntr_{jk} = 0$ **then**
2: $next\_state \leftarrow Normal$
3: **else if** $\exists\ i, j, k$: $(Cntr_{ij} = Cntr_{ik})$ & $(Cntr_{jk} = 0)$ **then**
4: $next\_state \leftarrow Recovery$
5: $FMR \leftarrow i$
6: **else if** $\exists\ i, j, k : (Cntr_{jk} = x)$ & $(Cntr_{ik} = y)$ & $(Cntr_{ij} = x + y)$ **then**
7: $next\_state \leftarrow Recovery$
8: $FMR \leftarrow i, j$
9: **else**
10: $next\_state \leftarrow Unrecoverable\ condition$
11: **end if**

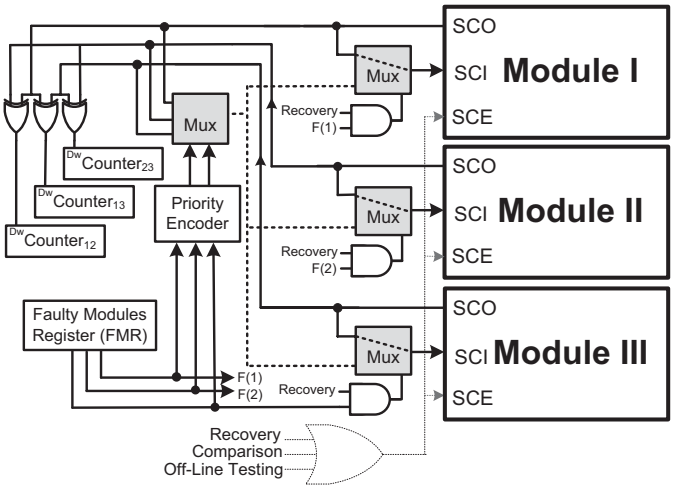

Fig. 7. SMERTMR in recovery mode.

mode. In this mode, the SMERTMR controller enables the scan chains of the SMERTMR modules and configures the multiplexers as follows: The SCI signal of fault-free modules is connected to the SCO signal of the same module. In addition, the SCI signal of the faulty module is connected to the SCO of one of the fault-free modules. As shown in Fig. 7, the value of the FMR register is used in the recovery mode to select the incoming driver of the appropriate signal driver for the SCI signals. Using the configuration shown in Fig. 7, the state of one of the fault-free modules is copied into the faulty modules after L_{sc} clock cycles.

While shifting out the states of the modules in the recovery mode, similar to the comparison mode, they are also compared to find any mismatch due to faults occurring in the recovery process. During the recovery process, whenever a mismatch is detected, the corresponding counter value containing the number of mismatches is decreased by one unit. At the end of the recovery process, all counters should be zero. This is because, for each mismatch, the corresponding counter is incremented by one unit during the comparison process and is decremented by one unit during the recovery process. If either of the counters is nonzero at the end of the recovery process, it is indicative of another fault occurrence during the recovery process. In this case, the SMERTMR system would enter the unrecoverable condition since such faults cannot be located.

Permanent error detection in SMERTMR is similar to that in the ScTMR technique explained in the previous section. Briefly, SMERTMR exploits the history of faulty modules to detect permanent faults. The only difference between ScTMR and SMERTMR in permanent error detection is that ScTMR stores the module status (faulty or fault-free) based on the voter results in the MRFM register, whereas SMERTMR stores the comparison process results (saved in FMR) in the MRFM register. The permanent error detection mechanism for the ScTMR technique was explained in Section III-B.

The proposed history-based permanent fault detection technique, however, can misdetect a permanent fault as a transient fault or vice versa. If consecutive transient faults occur in a short period in one module, the SMERTMR misdetects transient faults as a permanent fault and the system is degraded to the M/C configuration. However, the occurrence probability of consecutive transient faults in one module while no fault occurs in the other modules is extremely low. If the NCF in a module exceeds a predefined threshold (TR), the module is assumed to be permanently faulty. In this case, the probability that consecutive transient faults are detected as a permanent fault is equal to $1/3^{TR-1}$. For example, if TR is set to 10, the probability that consecutive transient faults are mistakenly regarded as a permanent fault is equal to 0.000050. TR can be adapted based on the circuit size and the environment condition. On the other hand, if the time for a permanent fault to manifest in the module outputs is comparable with the time interval of occurrence of consecutive transient faults, SMERTMR is not able to distinguish the permanent fault from transient faults. Although permanent faults in flip-flops can be detected during checkpointing, such faults in combinational logic are not detectable in checkpointing and during the recovery process.

C. Protection of SMERTMR Controller

The SMERTMR controller is a sequential circuit including 42 flip-flops and few logic gates. Previous mitigation techniques presented in [2] and [19]–[21] can be exploited to protect the sequential part. In the proposed technique, we have used the flip-flop triplication technique introduced in [2] to protect flip-flops in the SMERTMR controller. Using this technique, any bit-flip in flip-flops can be corrected in one clock cycle. Since the number of flip-flops within the SMERTMR controller is limited, the triplication technique does not impose significant area overhead to the overall circuit. Note that, because of very limited number of logic gates used in the SMERTMR controller, the combinational logic has not been protected in the case study presented in Section V. However, one can use either gate resizing techniques or SEU mitigation techniques to further improve the reliability of the controller [19], [20].

D. SMERTMR Offline Testing Capability

As mentioned earlier, the scan chains used in a TMR system to detect fabrication defects are reused for transient error recovery purposes while preserving the testability of the design. In the offline testing phase, SMERTMR facilitates communications between an external tester and the scan chains

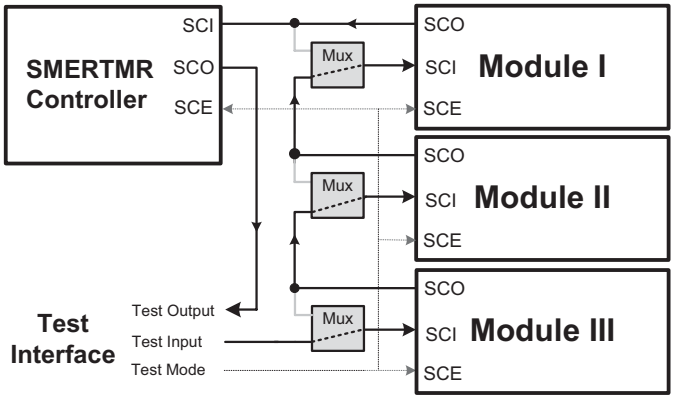

Fig. 8. SMERTMR in test mode.

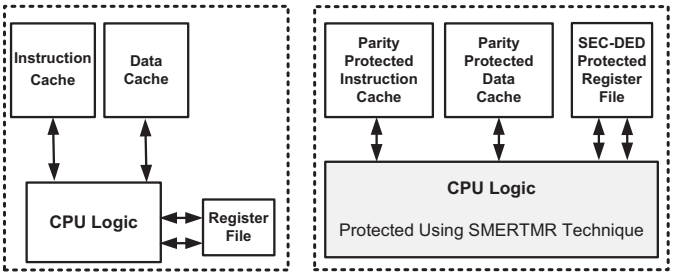

Fig. 9. Leon2 versus the SMERTMR processor.

in TMR modules and the SMERTMR controller. SMERTMR and the external tester communicate through the test interface, as shown in Fig. 8. As can be seen, three signals named test input, test output, and test mode are used to control the test process. During the offline testing process, test vectors are applied to the test input port and propagated to the test output port through the scan chains of modules and the SMERTMR controller. As shown in Fig. 8, in the offline testing mode, a scan chain is formed by flip-flops of all modules and the SMERTMR controller.

V. Experimental Evaluation

In order to evaluate the efficiency of SMERTMR, we have implemented a Leon2 processor core using the proposed technique. In the following subsections, the architecture and the implementation details of unprotected and protected Leon2 processors are described first. Then, the area and performance overheads of the SMERTMR technique are compared with those of the traditional TMR and ScTMR techniques. Finally, the error detection and recovery capability of SMERTMR is evaluated using an FPGA-based fault injection technique.

A. Case Study: Leon2

Both ScTMR and SMERTMR techniques can be employed in any design equipped with scan chains for testability purpose. To validate these techniques, we have applied both techniques to a processor design. As memory arrays such as cache and register file can be efficiently protected by means of error correction codes, ScTMR, and SMERTMR have been applied only to the processor core excluding the register file. As a case study, we have used a Leon2 processor IP core [12] to implement the ScTMR and SMERTMR techniques. In the Leon2 processor protected by ScTMR and ScTMR processor and SMERTMR processor hereafter (SMERTMR), the register file is protected by the single error correction double error detection (SEC-DED) code and it is shared between all three redundant cores. Since the registers in the Leon2 processor have 32-bit width, seven additional bits for error detection and correction are added to each entry in the register file. In the ScTMR and SMERTMR processors, to protect the write-through cache, a parity bit scheme is used, which is an appropriate and widely used solution [25]. To protect the main CPU core, we have used the ScTMR and SMERTMR techniques. Hereafter, the CPU core excluding the register file will be called the CPU logic. Fig. 9 shows Leon2 and the SMERTMR processor block diagrams.

B. Implementation Details

We have applied both ScTMR and SMERTMR techniques to the VHDL model of the Leon2 processor. In the first step, we have used parity and SEC-DED codes to protect the cache memories and register file, respectively. Then, the Leon2 processor with protected memory units is synthesized using synopsys design compiler [26]. Using this tool, we have added multiple scan chains to the processor core. Each module of the ScTMR/SMERTMR processor includes 2096 flip-flops. The scan chain used in each module is a multiple scan chain containing 16 parallel scan chains with a same length of 131 flip-flops ($L_{sc} = 131$). Note that, if the number of flip-flops in all chains is not same, the ScTMR/SMERTMR controller adds additional flip-flops to the smaller chains to equalize the length of all chains. In short, the implemented ScTMR/SMERTMR-protected processor consists of three redundant processor cores, a controller unit, and a voter. To verify the efficiency of the proposed architectures, different MiBench programs have been executed on both the Leon2 processor and the proposed ScTMR/SMERTMR processors. The selected benchmark programs are Qsort, Basicmath, and Bitcount. These three programs belong to automotive category MiBench [27].

C. Area Overhead

To extract the area overhead, we have used the synopsys design compiler [26] and UMC memory maker [28] toolsets. The results of area overheads for different processor architectures (Leon2, Core-TMR, the ScTMR processor, and the SMERTMR processor) are reported in Table II. Note that no protection has been used in Leon2 implementation. In the ScTMR and SMERTMR implementations, only the CPU logic is triplicated. The register file and the cache memories are, however, shared between all three modules. In the traditional TMR implementation, the CPU logic as well as the register file and the cache memories are fully triplicated. In order to have a fair comparison, the register file and the cache memories are shared in the Core-TMR implementation and are protected using SEC-DED and parity codes, respectively.

In Table II, the first row reports the area for the cache memories and the register file. The second row reports the area for the CPU logic which includes all components of the processor core excluding the caches and the register file. The third and

TABLE II
AREA OVERHEAD (μm^2) (65-nm STANDARD CELL LIBRARY)

Architecture	Leon2	Core-TMR	ScTMR	SMERTMR
Cache and register file	115 218	120 683	120 683	120 683
CPU logic	37 995	3 × 39 608	3 × 39 608	3 × 39 608
Voter	—	2801	4083	4083
Controller	—	—	1084	2273
Total area	153 213	239 507	244 674	245 863
Core area overhead versus core-TMR	—	0%	1.9%	2.9%

TABLE III
EFFECT OF 30 000 FAULTS INJECTED INTO THE CPU LOGIC: RESULTS ARE REPORTED IN PERCENT (*: SMERTMR-0.01%)

	Unprotected	Protected architectures		
Architecture	Leon2	Core-TMR	ScTMR	SMERTMR*
Failure	$8.6_{\pm 0.5}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$
Latent	$39.9_{\pm 0.8}$	$43.5_{\pm 0.8}$	$31.2_{\pm 0.7}$	$0.3_{\pm 0.1}$
Overwritten	$51.5_{\pm 0.8}$	$56.5_{\pm 0.8}$	$37.5_{\pm 0.8}$	$35.0_{\pm 0.8}$
Corrected	$0.0_{\pm 0.0}$	$0.0_{\pm 0.0}$	$31.3_{\pm 0.7}$	$64.7_{\pm 0.8}$

fourth rows report the area for the voter and the controller, respectively. The last two rows in Table II report the total area and the core area overhead for all architectures as opposed to the Core-TMR implementations. The area used by the CPU logic, the voter, and the controller in the ScTMR processor increases from (3 ×39 608 + 2801) μm^2 in the Core-TMR processor to (3 × 39 608 + 4083 + 1084) μm^2 in the ScTMR processor. Hence, the ScTMR technique has less than 2% area overhead compared to the traditional TMR implementation. In similar way, the SMERTMR technique imposes less than 3% area overhead compared to the traditional TMR system. Note that the area overhead reported in this table already includes the area overhead imposed to the SMERTMR controller by the flip-flop triplication technique.

D. Fault Injection

In order to evaluate the detection and recovery coverage of the ScTMR and SMERTMR processors, we have carried out statistical fault injection experiments. The experiments have been performed using the FPGA-based fault injection technique presented in [29], where faults are injected in the ScTMR and SMERTMR processors implemented on an FPGA platform. In fault injection experiments, we have used single event upset (SEU) as the fault model.

To analyze the fault injection results, the effect of each fault injection is classified as follows.

1) *Overwritten*: A fault is overwritten before it is propagated to the module outputs. Thus, the fault has no effect either on the output of the running workload or on the processor states at the end of the workload execution.
2) *Latent*: A fault does not affect the workload output but it does cause a mismatch in the processor state at the end of the workload execution.
3) *Corrected*: An error is detected, located, and corrected by the employed technique.

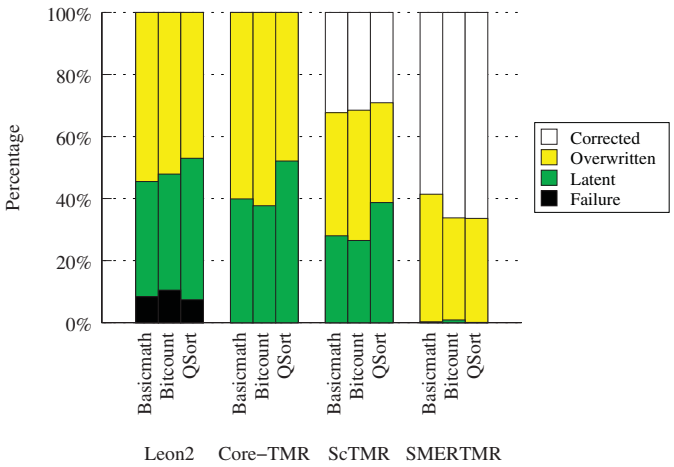

Fig. 10. Effect of SEU injection in CPU logic for different testbenches.

4) *Failure*: An error causes a wrong result or leads to the system halt.

The results of fault injection experiments are reported in Table III. A total of 30 000 fault injection experiments have been performed on each architecture. All fault injection results are reported with 95% confidence level. The sampling error is calculated using the following equation [30]:

$$\text{Sampling Error} = Z_{1-a/2}\sqrt{\frac{p(1-p)}{n}}. \quad (1)$$

In this equation, n and p are the number of samples and the occurrence probability of the fault injection effect (overwritten, latent, corrected, or failure), respectively. The other parameter, $Z_{1-a/2}$, is a function of the confidence level (here, $a = 95\%$ and $Z_{1-a/2} = 1.96$).

As can be seen from Table III, all injected faults into the CPU logic of the Core-TMR processor are either overwritten or latent. Since there is no recovery scheme in the Core-TMR implementation, the percentage of detected, located, and corrected faults is equal to zero. The ScTMR processor detects, locates, and corrects 31.3% of faults, while 31.2% of faults remain latent. As discussed earlier, this remarkable ratio of latent faults can cause unrecoverable condition upon occurrence of a second fault in the subsequent clock cycles. In the SMERTMR technique, we can activate a checkpoint signal such a way that the performance overhead of the system due to running comparison process becomes less than 0.01%. In our experiments, SMERTMR with less than 0.01% performance overhead is denoted by SMERTMR-0.01%. Similarly, SMERTMR with less than 1.0% and 0.0001% performance overhead is denoted by SMERTMR-1% and SMERTMR-0.0001%, respectively. The SMERTMR-0.01% architecture results in a considerable decrease of about 30.9% (31.2%-0.3%) in the total number of latent faults. The remaining 0.3% of latent faults is due to injection of faults after the last checkpoint. If a checkpoint is added at the end of the testbench, this percentage will become zero. The results of FPGA-based fault injection experiments for different benchmark programs running on different processor architectures (Leon2, Core-TMR, the ScTMR processor, and the SMERTMR processor) are also reported in Fig. 10. Note that the results reported in Table III are the average percentages over three benchmark

TABLE IV
EFFECT OF 10 000 FAULTS INJECTED INTO TWO MODULES IN EACH ARCHITECTURE (RESULTS REPORTED IN PERCENT)

Architecture	ScTMR	SMERTMR-0%	SMERTMR-0.01%
Failure	$0.20_{\pm 0.10}$	$0.05_{\pm 0.04}$	$0.01_{\pm 0.02}$
Unrecoverable	$34.81_{\pm 0.93}$	$0.23_{\pm 0.09}$	$0.14_{\pm 0.07}$
Latent	$22.59_{\pm 0.82}$	$19.98_{\pm 0.78}$	$0.31_{\pm 0.11}$
Overwritten	$26.49_{\pm 0.86}$	$28.10_{\pm 0.88}$	$13.28_{\pm 0.67}$
Corrected	$15.91_{\pm 0.72}$	$51.64_{\pm 0.98}$	$86.26_{\pm 0.67}$

programs, i.e., Qsort (424 Mcycles), Bitcount (320 Mcycles), and Basicmath (745 Mcycles).

As reported in Table III, the number of overwritten faults in Leon2 is greater than that in the ScTMR and SMERTMR processors. This can be understood by the way errors are treated in these two architectures. In the ScTMR/SMERTMR processors, a byte or a word is written to the cache if all three CPU cores produce the same result. Therefore, it is likely that an erroneous data is prevented from being written to the cache while the erroneous data would be possibly overwritten and be masked by the cache memory. This means that the masking factor of the cache memory is excluded from the overall system masking factor in the ScTMR/SMERTMR processor architecture. That is why the number of overwritten faults in the ScTMR/SMERTMR processors is less than that of the basic Leon2 CPU core.

As mentioned in Section IV, SMERTMR has the detection and correction capability of two faulty modules using the proposed comparison process. In order to evaluate the recovery coverage of the SMERTMR technique in the presence of two faulty modules, we have injected two faults in two different modules. The results of fault injection experiments are shown in Table IV. As can be seen from Table IV, 34.81% of faults lead to the unrecoverable condition in the ScTMR architecture. In this case, ScTMR detects that there are inconsistencies between all three modules, but it cannot locate the fault-free module. Fault injection results also reveal that 0.20% of injected faults into ScTMR leads to a system failure. In this case, ScTMR misdetects the faulty module as both faulty modules create the same faulty output at the same clock cycle. Consequently, ScTMR considers the fault-free module as faulty and copies the state of one of the faulty modules into the fault-free module.

As reported in Table IV, in the SMERTMR-0% architecture, only 0.23% of faults causes unrecoverable condition. As discussed in Section IV, when two faulty modules have common flip-flops in the SMERTMR architecture, it enters the unrecoverable condition and cannot recover the system to its fault-free state. In SMERTMR-0%, 51.64% of faults are detected and corrected while 19.98% of faults remain latent. By employing an appropriate checkpointing scheme, latent faults can be completely removed from the system. As can be seen from Table IV, in SMERTMR-0.01%, the percentage of latent faults decreases to 0.31%. Another remarkable point in the fault injection results is that the failure percentage in SMERTMR and SMERTMR-0.01% is 0.05% and 0.01%, respectively.

There is only one situation where the SMERTMR technique cannot recover two faulty modules. Suppose that module i and module j are faulty and module k is fault-free. Let A and B be the sets of erroneous flip-flops in modules i and j, respectively. If $A \subseteq B$ or $B \subseteq A$, SMERTMR cannot detect the fault-free module correctly. We explain this situation with an example. Let x and y be the number of faulty flip-flops in modules i and j, respectively. If $A \subseteq B$, after the comparison process, we will have $\text{Counter}_{ij} = y - x$, $\text{Counter}_{ik} = x$, and $\text{Counter}_{jk} = y$. In this case, Counter_{jk} is equal to ($\text{Counter}_{ij} + \text{Counter}_{ik}$) and consequently SMERTMR identifies modules i and k as faulty modules and module j as fault-free module. However, if Counter_{jk} is equal to $\text{Counter}_{ij} + \text{Counter}_{ik}$, it will be similar to the cases where $A \cap B = \emptyset$. Consequently, SMERTMR misdetects module i and module k as faulty modules and module j as a fault-free module. This misdetection will cause a failure in the SMERTMR technique.

Since ScTMR and SMERTMR cannot recover the system in which an unrecoverable conditions is detected, the unrecoverable condition signal is activated and the system halts. Note that the unrecoverable condition is a fail-safe state, while failure is a fail-unsafe state. Considering both these cases as failures, SMERTMR-0% and SMERTMR-0.01% only fail in 0.28% (0.05% + 0.23%) and 0.15% (0.01% + 0.14%) of cases, respectively, where there are two faulty modules.

E. Performance Overhead

Two important factors that influence the performance of the ScTMR- and SMERTMR-based systems are the increase in the critical path delay and the time required for error recovery. Additionally, in SMERTMR, the time required to run the comparison process at checkpoints imposes performance overhead to the system. Our simulation results show that the critical path delay in both ScTMR and SMERTMR processors is increased by 0.44% compared to that of Leon2. Therefore, the minimum clock cycle is increased by 0.44%, resulting in 0.44% performance overhead. The simulation results also show that the minimum clock cycle in the TMR architecture is increased by 0.32%.

The other factor influencing the performance of the ScTMR- and SMERTMR-based systems is recovery time, which depends on the number of flip-flops and the width of multiple scan chains. The following equation can be used to measure the recovery time in a ScTMR-based system:

$$\text{Recovery Time} = L_{\text{sc}} \times \text{ Clock Period.} \tag{2}$$

The time required for a comparison or recovery process in the SMERTMR technique is equal to the recovery time of the ScTMR technique. This is because the scan chain is fully circulated in both techniques during the recovery process. However, the SMERTMR technique imposes twice the performance overhead to the system in the presence of a fault as compared to the ScTMR technique since both comparison and recovery processes are required for locating and correcting the faulty module in the SMERTMR technique. As can be inferred from (2), the recovery time decreases when the width of the multiple scan chains increases. In our simulations, by employing 16 parallel scan chains, L_{sc} is equal to 131. Therefore, it takes 131 and 262 clock cycles in ScTMR and SMERTMR, respectively, to recover the system from a transient error. In case multiple scan chains

are not affordable, employing only a single scan chain will impose 2096 and 4192 clock cycles performance overhead during the recovery process in ScTMR and SMERTMR, respectively.

Another parameter that affects the performance of a system protected by SMERTMR is the frequency of checkpointing. Checkpointing imposes a negligible impact on the system performance. As an example, if the checkpoint signal is triggered at every 10 million clock cycles, the performance overhead of the system due to checkpointing is equal to 0.042% ($= 4192/10M$) and 0.0026% ($= 262/10M$) for single and 16-parallel scan chains, respectively. It should be noted that checkpointing is used to further enhance the reliability of the system protected by SMERTMR. As will be shown in the next section, SMERTMR with checkpointing rate of 0% (denoted as SMERTMR-0%) improves the reliability by orders of magnitudes compared to other techniques. Additionally, as reported in Table III, employing a checkpointing rate that imposes only 0.01% performance overhead (denoted as SMERTMR-0.01%) results in a significant reduction in the percentage of latent faults and a remarkable increase in the percentage of corrected faults.

Note that, because of the negligible performance overhead of the recovery process, we have not included the task deadline violation failure mechanism in the results provided in Tables III and IV. As an example, injecting five transient faults when running the Qsort program imposes less than 0.0003% performance overhead to the normal execution time of this program in the SMERTMR architecture. Task deadline violation failure mechanism can be considerable when either the recovery process is time consuming or the rate of transient faults is extremely large.

F. Limitations of SMERTMR

Although SMERTMR offers appealing features such as detection and correction of multiple bit errors and fast error recovery, it suffers from few limitations. First, SMERTMR can be applied only to TMR systems where the three replicas are synchronized at any clock cycle and share exactly the same micro-architecture. For instance, in our case study, all three replicas have the same implementation and exactly the same number of flip-flops. There are cases where three replicas are designed and fabricated in different ways to reduce common failure modes. As such, SMERTMR is not applicable to those TMR systems. Additionally, it is assumed that the state of the three replicas is the same at any clock cycle in the fault-free state. There are few cases in which an error threshold between the state of three replicas is acceptable. As an example, when the three replicas receive their inputs from sensors, the state of three replicas are not exactly the same in the fault-free state. As a result, SMERTMR is not applicable to such TMR systems because of possible mismatch between fault-free states.

Another shortcoming of SMERTMR as compared to the traditional TMR is that SMERTMR introduces a performance penalty for transient faults that do not result in any effect in the final outputs. As reported in Table III, a significant number of transient faults in the unprotected Leon2 processor is overwritten in the next clock cycles and do not need any recovery mechanism. As an example, a transient fault in a register may never be read and eventually it will be overwritten by error-free values. The proposed technique is not able to distinguish between such overwritten faults and the faults that can lead to system failure. However, as demonstrated in Section V-E, the recovery process introduces only a negligible performance overhead to the system.

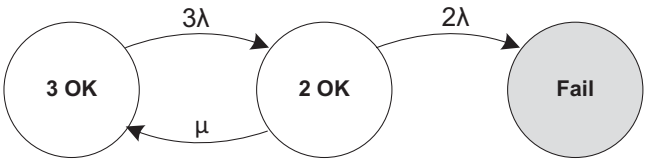

Fig. 11. Markov diagram of ScTMR.

VI. Analytical Assessment

In this section, the reliability of the ScTMR and SMERTMR techniques is evaluated using an analytical approach. The ScTMR and SMERTMR techniques are designed for both permanent and transient fault recovery. We have used a Markov model to evaluate the reliability of the proposed techniques. In the proposed analytical study, we have made the following assumptions.

1) Module failure is statistically independent.
2) Fault occurrence follows a Poisson process with a failure rate of λ.
3) The failure rate of the voter and the controller is negligible and is therefore ignored.

In the next subsections, the reliability of the proposed architectures is modeled in two cases: 1) in the presence of transient errors and 2) in the presence of permanent errors. Note that the reliability modeling of the proposed architectures in the presence of both transient and permanent errors makes this analysis complicated and possibly intractable. Additionally, there is no accurate field data reporting the ratio of transient and permanent errors [6].

A. ScTMR Reliability in the Presence of Transient Errors

Unlike the traditional TMR system, in the ScTMR technique, the faulty module can be recovered if another module does not fail before the completion of the recovery process. Fig. 11 shows the Markov diagram of the ScTMR technique, where μ is the recovery rate. The Markov model of ScTMR has three states.

1) 3OK: all three modules are fault-free.
2) 2OK: there is only one faulty module.
3) Fail: there is more than one faulty module and, therefore, ScTMR cannot recover the system into its fault-free state.

Originally, the system is in the 3OK state. When one of the modules becomes faulty, it enters the 2OK state where only two modules are operational. The rate at which the system leaves the 3OK state and enters the 2OK state is 3λ, as three modules may become independently faulty at the rate of λ. In the 2OK state, while the system continues to work by two operational modules, the recovery process for the faulty module is started. If another module becomes faulty during the recovery process, the system enters the Fail state. Otherwise,

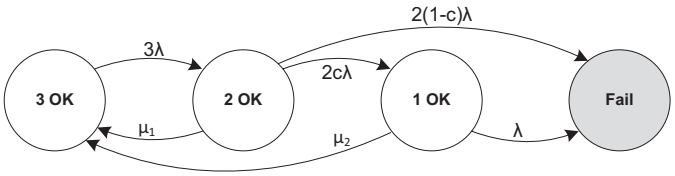

Fig. 12. Markov diagram of SMERTMR.

the recovery process is successfully completed and the system returns to the 3OK state. It should be noted that, when the system enters the Fail state, the ScTMR technique cannot recover the faulty modules. Therefore, in this state, there is no returning edge either to the 2OK or to the 3OK states. Under this condition, the reliability expression of ScTMR can be written as follows:

$$R^{T}_{\mathrm{ScTMR}}(t) = P_{3\mathrm{OK}}(t) + P_{2\mathrm{OK}}(t) \tag{3}$$

where $P_i(t)$ is the probability of the system being in the state of i at time t.

B. SMERTMR Reliability in the Presence of Transient Errors

Fig. 12 shows the Markov diagram of the SMERTMR technique in the presence of transient errors. In this diagram, μ_1 and μ_2 are the recovery rates when there are one or two faulty modules, respectively. As mentioned in Section IV, SMERTMR cannot recover from all cases in which there are two faulty modules. Parameter c in Fig. 12 is the recovery coverage when there are two faulty modules. In other words, when one of the operational modules becomes faulty in the 2OK state, the probability that the system can be recovered to its fault-free state is c. Consequently, the system enters the 1OK state at the rate of $2c\lambda$ and it enters the Fail state at the rate of $2(1-c)\lambda$.

The Markov diagram of SMERTMR has four states.

1) 3OK: all three modules are fault-free.
2) 2OK: there is only one faulty module.
3) 1OK: there is only one fault-free module and the system can be restored to its fault-free state.
4) Fail: SMERTMR cannot recover the system.

During the normal operation of the system, i.e., when the system is in the 3OK state, whenever a fault occurs, the system enters the 2OK state and the recovery process is started. If another module becomes faulty during the recovery process, the system enters the 1OK state. Otherwise, the system is recovered to the fault-free state by activating the recovery process. Similar to ScTMR, the Fail state is an absorbing state in SMERTMR, i.e., no recovery mechanism exists in the Fail state. Based on the Markov diagram shown in Fig. 12, the reliability of SMERTMR can be extracted using (4)

$$R_{\mathrm{ScTMR}}(t) = P_{3\mathrm{OK}}(t) + P_{2\mathrm{OK}}(t) + P_{1\mathrm{OK}}(t). \tag{4}$$

C. ScTMR and SMERTMR Reliability in the Presence of Permanent Errors

In case of permanent faults, ScTMR and SMERTMR behave like the traditional TMR technique. This means that, after occurrence of the first permanent fault in one of the TMR modules, the other modules are used to generate the system outputs. In this case, if a second permanent fault occurs, the system fails to obtain the majority between TMR modules because of having two permanently faulty modules.

Markov diagram of ScTMR and SMERTMR in the presence of permanent faults is very similar to that of the ScTMR in the presence of transient faults (depicted in Fig. 11). The main difference here is that μ is equal to zero as there is no recovery mechanism for permanent faults in the proposed techniques. By considering $\mu = 0$ in the ScTMR reliability equation (3), it can be simplified as follows:

$$R^{P}_{\mathrm{ScTMR}}(t) = R^{P}_{\mathrm{SMERTMR}}(t) = 3e^{-3\lambda t} - 2e^{-3\lambda t}. \tag{5}$$

D. Reliability of NMR and SPMR

In order to show the efficiency of ScTMR and SMERTMR, we have compared the reliability of the proposed techniques with two well-known techniques, namely N-tuple modular dedundancy (NMR) and self-purging modular redundancy (SPMR) [31] with N modules. In NMR, a module is replicated N times ($N = 2n + 1$, n is an integer) and it is operational if at least $(N + 1)/2$ (or, $n + 1$) modules are fault-free. As shown in [7], since there is no recovery or synchronization mechanism in the traditional NMR, transient faults would have a permanent effect on faulty modules. Therefore, modules affected by transient faults are considered to have a permanent effect. In [32], it has been proved that the reliability of an NMR system with an ideal voter is

$$R_{\mathrm{NMR}}(t) = \sum_{i=0}^{n} \binom{N}{i} (1 - R_M(t))^i (R_M(t))^{N-i}. \tag{6}$$

In the above equation, $N = 2n + 1$ is the number of modules and $R_M(t)$ is the reliability of a single module.

In the SPMR scheme, all N modules are active and the system output is fault-free as long as there are at least two operational modules. In this scheme, whenever a module output disagrees with the outputs of the other modules, the module is switched out and it will no longer contributes in the voting. The general reliability expression for a SPMR system with ideal voter and switches is as follows [31]:

$$R_{\mathrm{SPMR}}(t) = 1 - \sum_{i=0}^{1} \binom{N}{i} (R_M(t))^i (1 - R_M(t))^{N-i}. \tag{7}$$

E. Case Study

In this subsection, we compare the reliability of our proposed technique, i.e., SMERTMR, with the reliability of the conventional TMR, 51MR (NMR with $N = 51$), 5SPMR, 51SPMR, and ScTMR as a function of operating time using the equations presented in the previous subsections. Because of the large amount of redundancies, 51MR and 51SPMR are not affordable in real applications. However, to show the efficiency of our proposed technique, we have chosen 51MR and 51SPMR in our analysis as representatives of NMR and NSPMR systems with aggressive redundancies.

Note most parameters used in our analytical assessment (such as recovery coverage, number of chains, number of flip-flops) are directly extracted from the experimental results presented in Section V. The transient and permanent failure rates reported in previous paper are used as inputs to

this analytical study. In this paper, we have assumed the following.

1) Each module contains 2000 flip-flops connected together through a single scan chain. Based on this assumption, the recovery process in ScTMR and SMERTMR will take 2000 and 2 × 2000 clock cycles, respectively.
2) Failure rates of each module for transient and permanent faults are 10^{-3} and 10^{-5}/h, respectively.
3) The error recovery coverage (c) for SMERTMR is 0.99, which is in agreement with the fault injection experimental results.

In order to simplify this analysis, we have ignored the effect of latent faults in the results provided in Fig. 13(a). The effect of latent faults has been considered in the results provided in Fig. 13(b). As can be seen in Fig. 13(a), the reliability of the conventional TMR sharply decreases with the operating time. This is because, when a fault occurs in one of the TMR modules, the TMR system degrades to a serial system with two modules. In this case, since there is no recovery mechanism, upon occurrence of a second fault in the remaining fault-free modules, the system will fail. Note that, after the occurrence of the first fault, the TMR system becomes less reliable than a simple system including just one module. This issue also exists for 51MR system. An interesting point in the comparison of 51MR and TMR is that, after a specific time, the reliability of 51MR becomes less than that of TMR. In a 51MR system, once 25 out of 51 modules become faulty, the system becomes less reliable than a TMR system with two operational modules. As shown in Fig. 13(a), the reliability of 5SPMR will eventually become greater than that of 51MR since, upon occurrence of a fault in one of the 5SPMR modules, the faulty module is switched out and it no longer contributes to the overall voting. As shown in Fig. 13(a), the reliability of the ScTMR and SMERTMR techniques decreases only very slightly with operating time. This is mainly due to the ability of the proposed techniques to detect and recover the faulty modules. Since we have ignored the effect of latent faults in the results provided in Fig. 13(a), there is no difference between the reliability of SMERTMR with and without checkpointing.

In order to include the effect of latent faults in this paper, we need first to compute the average fault latency. Fault latency is the time between the occurrence of a fault and either its manifestation as an error at the module outputs or its elimination from the system. We have used our fault injection experiments to extract the average fault latency for each architecture. To this end, we have monitored the fault propagation for 500 faults injected into each architecture to measure the average fault latency. Let assume that the recovery time and the average fault latency of a system are denoted by t_{rec} and t_{lat}, respectively. The recovery rate of the system considering latent faults is equal to $\mu = t_{\text{lat}}/(t_{\text{rec}} + t_{\text{lat}})$. Fig. 13(b) shows the reliability of each architecture considering latent faults. The reliability of SMERTMR with no checkpointing (SMERTMR-0%), SMERTMR-1%, SMERTMR-0.01%, and SMERTMR-0.0001% is shown in this figure. As can be seen, checkpointing has a significant impact on the reliability of the SMERTMR system. As can be inferred from the results provided in Fig. 13(b), increasing the checkpointing rate results in a remarkable improvement on the overall system reliability. However, it also increases the performance overhead of the SMERTMR system.

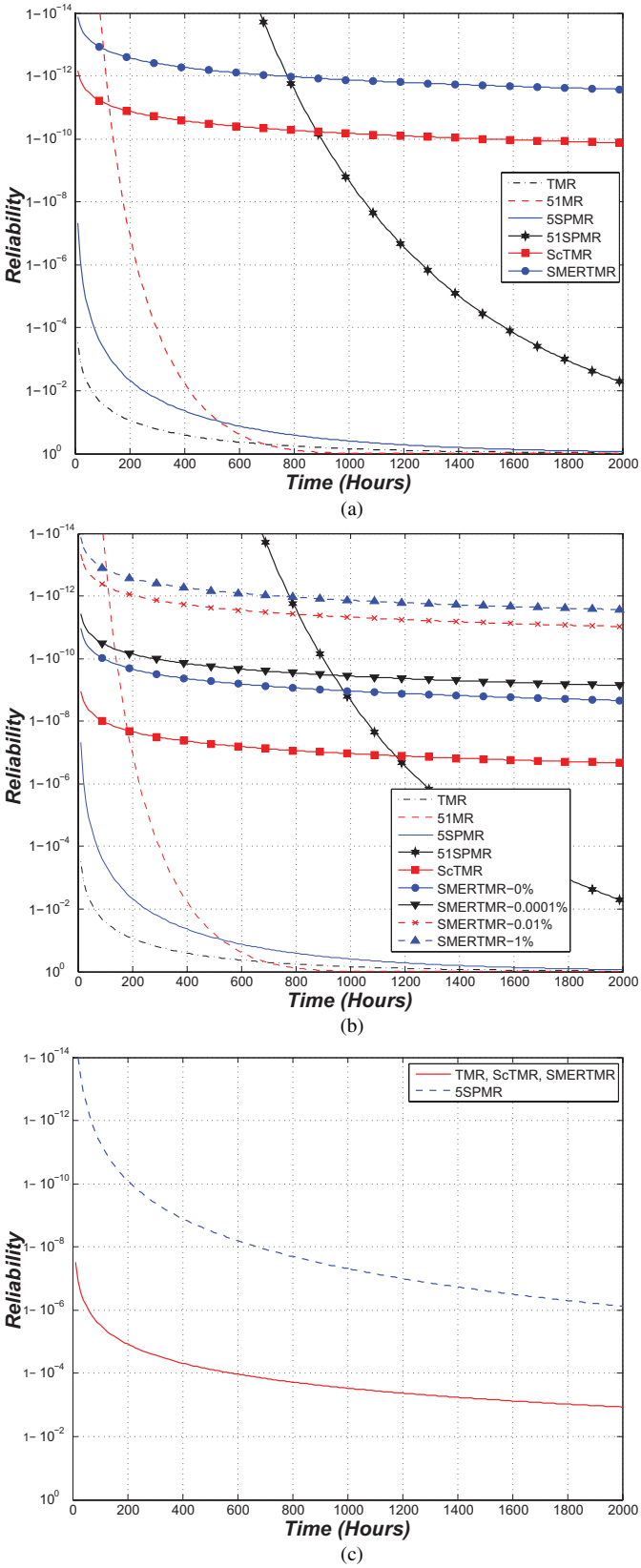

Fig. 13. Reliability in the presence of (a) transient faults without consideration of latent faults, (b) transient faults with consideration of latent faults, and (c) permanent faults.

TABLE V
MTTF FOR DIFFERENT HARDWARE REDUNDANCY TECHNIQUES (HOURS)

Technique	MTTF transient	MTTF permanent	Failure type
TMR	833	8.33×10^4	SDC
51MR	702	7.02×10^4	SDC
5SPMR	1283	1.28×10^5	DUE
51SPMR	3518	3.51×10^5	DUE
ScTMR	9.23×10^9	8.33×10^4	DUE
SMERTMR-0%	9.21×10^{11}	8.33×10^4	DUE
SMERTMR-0.0001%	2.76×10^{12}	8.33×10^4	DUE
SMERTMR-0.01%	2.11×10^{14}	8.33×10^4	DUE
SMERTMR-1%	7.32×10^{14}	8.33×10^4	DUE

Fig. 13(c) shows the reliability of SMERTMR, ScTMR, the traditional TMR, and 5SPMR in the presence of permanent faults. As can be inferred from this figure, SMERTMR, ScTMR, and TMR provide the same reliability level. However, 5SPMR has higher reliability than the other techniques while imposing much more area overhead to the system. Briefly, SMERTMR has the best transient error detection capability as compared to other well-known techniques, whereas it has almost the same reliability level as the traditional TMR technique in the presence of permanent faults.

To have a more accurate comparison, we have extracted the mean time to failure (MTTF) of all discussed techniques. MTTF is calculated as $\text{MTTF} = \int_0^\infty R(t)dt$. Here, $R(t)$ is the reliability of a system. The results of MTTF analysis are summarized in Table V. As reported in Table V, the MTTF of SMERTMR in the presence of transient faults is 11 orders of magnitude greater than that of TMR, while its MTTF with respect to permanent faults is the same. Although the MTTF of SMERTMR is similar to that of TMR, the failure types are different. In TMR, a failure is not typically detected, i.e., silent data corruption (SDC) is very likely to occur in the traditional TMR technique. However, permanent failures are detected in SMERTMR even though such failures are not recoverable. Therefore, in SMERTMR, the failure types are degraded from SDCs to detected unrecoverable errors (DUEs). Note that DUE failures are preferable to SDC failures in safety-critical applications.

Finally, it should be noted that, as opposed to SMERTMR, the static redundancy techniques with no error recovery mechanism such as NMR do not cause any interruption to the normal operation of the system in order to mask the effect of transient and permanent faults. This key characteristic of static redundancy techniques, i.e., fault masking without interruption in the system functionality, is only valid for the first few failure events. As an example, a TMR system without a recovery mechanism will fail upon occurrence of the second component failure. Hence, it is a well-known fact that static redundancy techniques such as TMR, NMR, and NSPMR would have lower MTTF than a redundancy-based technique equipped with a recovery mechanism. As a result, such techniques are not suitable for long-time safety-critical missions.

VII. CONCLUSION

In this paper, we presented a roll-forward technique, called SMERTMR, to recover multiple errors in TMR systems. In the proposed technique, we employed built-in design for testability resources available in digital circuits to recover faulty modules in the presence of multiple transient faults. To validate the proposed technique, we performed fault injection experiments on the Leon2 processor by running selected benchmark programs from the automotive category MiBench. The fault injection experiments demonstrated that SMERTMR detects, locates, and corrects 100% and 99.7% of multiple faults affecting single and two faulty modules, respectively. The results also show that the area and the performance overheads of SMERTMR, as compared to the traditional TMR system, are less than 3% and 1%, respectively. An analytical assessment also demonstrated that SMERTMR improves the reliability of TMR systems by several orders of magnitude compared to the state-of-the-art techniques.

REFERENCES

[1] A. L. J. Hopkins, T. B. Smith, and J. H. Lala, "FTMP—A highly reliable fault-tolerant multiprocess for aircraft," *Proc. IEEE*, vol. 66, no. 10, pp. 1221–1239, Oct. 1978.

[2] J. Gaisler, "A portable and fault-tolerant microprocessor based on the SPARC v8 architecture," in *Proc. Int. Conf. Depend. Syst. Netw.*, 2002, pp. 409–415.

[3] 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 Autom. Test Eur. Conf. Exhibit.*, 2005, pp. 1530–1591.

[4] P. K. Samudrala, J. Ramos, and S. Katkoori, "Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs," *IEEE Trans. Nucl. Sci.*, vol. 51, no. 5, pp. 2957–2969, Oct. 2004.

[5] H. Kim and K. G. Shin, "Design and analysis of an optimal instruction-retry policy for TMR controller computers," *IEEE Trans. Comput.*, vol. 45, no. 11, pp. 1217–1225, Nov. 1996.

[6] S. Yu and E. J. McCluskey, "On-line testing and recovery in TMR systems for real-time applications," in *Proc. Int. Test Conf.*, 2001, pp. 240–249.

[7] K. G. Shin and H. Kim, "A time redundancy approach to TMR failures using fault-state likelihoods," *IEEE Trans. Comput.*, vol. 43, no. 10, pp. 1151–1162, Oct. 1994.

[8] P. K. Chande, A. K. Ramani, and P. C. Sharma, "Modular TMR multiprocessor system," *IEEE Trans. Ind. Electron.*, vol. 36, no. 1, pp. 34–41, Feb. 1989.

[9] S. D'Angelo, C. Metra, and G. Sechi, "Transient and permanent fault diagnosis for FPGA-based TMR systems," in *Proc. Int. Symp. Defect Fault Tolerance VLSI Syst.*, 1999, pp. 330–338.

[10] J. Yoon and H. Kim, "Time-redundant recovery policy of TMR failures using rollback and roll-forward methods," *IEE Proc. -Comput. Digit. Tech.*, vol. 147, no. 2, pp. 124–132, Mar. 2000.

[11] M. Ebrahimi, S. G. Miremadi, and H. Asadi, "ScTMR: A scan chain-based error recovery technique for TMR systems in safety-critical applications," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, 2011, pp. 1–4.

[12] *The Leon2 Processor User Manual*. (2007) [Online]. Available: http://www.gaisler.com

[13] G. Latif-Shabgahi and S. Bennett, "Adaptive majority voter: A novel voting algorithm for real-time fault-tolerant control systems," in *Proc. 25th EUROMICRO Conf.*, vol. 2. 1999, pp. 113–120.

[14] O. Ruano, J. A. Maestro, and P. Reviriego, "A methodology for automatic insertion of selective TMR in digital circuits affected by SEUs," *IEEE Trans. Nucl. Sci.*, vol. 56, no. 4, pp. 2091–2102, Aug. 2009.

[15] J. M. Johnson and M. J. Wirthlin, "Voter insertion algorithms for FPGA designs using triple modular redundancy," in *Proc. 18th Annu. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2010, pp. 249–258.

[16] J. A. Abraham and D. P. Siewiorek, "An algorithm for the accurate reliability evaluation of triple modular redundancy networks," *IEEE Trans. Comput.*, vol. 23, no. 7, pp. 682–692, Jul. 1974.

[17] X. Wang, "Partitioning triple modular redundancy for single event upset mitigation in FPGA," in *Proc. Int. Conf. E-Product E-Service E-Entertain.*, 2010, pp. 1–4.
[18] S. Nomura, M. D. Sinclair, C.-H. Ho, V. Govindaraju, M. de Kruijf, and K. Sankaralingam, "Sampling + DMR: Practical and low-overhead permanent fault detection," in *Proc. 38th Int. Symp. Comput. Arch.*, 2011, pp. 201–212.
[19] M. Zhang, S. Mitra, T. M. Mak, N. Seifert, N. J. Wang, Q. Shi, K. S. Kim, N. R. Shanbhag, and S. J. Patel, "Sequential element design with built-in soft error resilience," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 12, pp. 1368–1378, Dec. 2006.
[20] R. R. Rao, D. Blaauw, and D. Sylvester, "Soft error reduction in combinational logic using gate resizing and flipflop selection," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2006, pp. 502–509.
[21] M. Fazeli, A. Patooghy, S. G. Miremadi, and A. Ejlali, "Feedback redundancy: A power-aware SEU-tolerant latch design in DSM technologies," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw.*, Jun. 2007, pp. 276–285.
[22] L. Wang, N. Touba, Z. Jiang, S. Wu, J. L. Huang, and J. C.-M. Li, "CSER: BISER-based concurrent soft-error resilience," in *Proc. 28th IEEE VLSI Test Symp.*, Apr. 2010, pp. 153–158.
[23] G. Rui, C. Wei, L. Fang, D. Kui, and W. Zhiying, "Modified triple modular redundancy structure based on asynchronous circuit technique," in *Proc. 38th Int. Symp. Defect Fault Tolerance VLSI Syst.*, 2006, pp. 184–196.
[24] X. She and K. S. McElvain, "Time multiplexed triple modular redundancy for single event upset mitigation," *IEEE Trans. Nucl. Sci.*, vol. 56, no. 4, pp. 2443–2448, Aug. 2009.
[25] H. Asadi, V. Sridharan, M. B. Tahoori, and D. Kaeli, "Balancing performance and reliability in the memory hierarchy," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Mar. 2005, pp. 269–279.
[26] *Synopsys Design Compiler*. (2010) [Online]. Available: http://www.synopsys.com
[27] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. IEEE Int. Workshop Workload Character.*, Dec. 2001, pp. 3–14.
[28] *UMC Memory Maker*. (2010) [Online]. Available: http://www.umc.com
[29] A. Mohammadi, M. Ebrahimi, A. Ejlali, and S. G. Miremadi, "SCFIT: A FPGA-based fault injection technique for SEU fault model," in *Proc. Design Autom. Test Eur. Conf.*, 2012, pp. 586–589.
[30] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert, "Statistical fault injection: Quantified error and confidence," in *Proc. Design Autom. Test Eur. Conf. Exhibit.*, 2009, pp. 502–506.
[31] J. Losq, "A highly efficient redundancy scheme: Self-purging redundancy," *IEEE Trans. Comput.*, vol. 25, no. 6, pp. 569–578, Jun. 1976.
[32] I. Koren and S. Y. H. Su, "Reliability analysis of N-modular redundancy systems with intermittent and permanent faults," *IEEE Trans. Comput.*, vol. 28, no. 7, pp. 514–520, Jul. 1979.

Mojtaba Ebrahimi (S'12) received the B.Sc. degree from Shahed University, Tehran, Iran, and the M.Sc. degree from Sharif University, Tehran, in 2008 and 2010, respectively, both in computer engineering. He is currently pursuing the Ph.D. degree with the Chair of Dependable and Nano Computing, Karlsruhe Institute of Technology, Karlsruhe, Germany.

He was a Research Assistant with the Dependables System Laboratory, Sharif University, from 2010 to 2011.

Seyed Ghassem Miremadi (SM'07) received the M.Sc. degree in applied physics and electrical engineering from the Linköping Institute of Technology, Linköping, Sweden, and the Ph.D. degree in computer engineering from the Chalmers University of Technology, Gothenburg, Sweden, in 1984 and 1995, respectively.

He is a Professor of computer engineering with the Sharif University of Technology (SUT), Tehran, Iran. He has established and has been the Chair of the Dependable Systems Laboratory, SUT, since 1996. He was the Education Director from 1997 to 1998, the Head from 1998 to 2002, the Research Director from 2002 to 2006, and the Director of the Hardware Group from 2009 to 2010, with the Computer Engineering Department, SUT. From 2003 to 2010, he was the Director of the Information Technology Program, SUT International Campus, Kish Island, Iran. He has authored or co-authored several scientific articles and conference papers. He has been involved in research on physical, simulation-based and software-implemented fault injection, dependability evaluation using HDL models, fault-tolerant embedded systems, fault-tolerant NoCs, and fault-tolerant real-time systems.

Dr. Miremadi is an Editor of the *Scientia Journal on Computer Science and Engineering*. He is a Senior Member of the IEEE Computer Society and the IEEE Reliability Society. He is currently the Vice-Chairman of Academic Affairs with the SUT.

Hossein Asadi (M'08) received the B.Sc. and M.Sc. degrees in computer engineering from the Sharif University of Technology (SUT), Tehran, Iran, in 2000 and 2002, respectively, and the Ph.D. degree in electrical and computer engineering from Northeastern University, Boston, MA, in 2007.

He has been with the Department of Computer Engineering, SUT, since 2009, where he is currently an Assistant Professor. He was with EMC Corporation, Hopkinton, MA, as a Research Scientist and Senior Hardware Engineer, from 2006 to 2009. From 2002 to 2003, he was a member of the Dependable Systems Laboratory, SUT, where he researched hardware verification techniques. From 2001 to 2002, he was a member of the Sharif Rescue Robots Group. He has authored or co-authored more than 50 technical papers in reputed journals and conference proceedings. His current research interests include data storage systems and networks, solid-state disk drives, and reconfigurable and dependable computing.

Dr. Asadi was a recipient of the Technical Award for the Best Robot Design from the International RoboCup Rescue Competition, organized by AAAI and RoboCup, and the Distinguished Lecturer Award from the SUT in 2010, one of the most prestigious awards in the university.

Mahdi Fazeli (S'10) received the Ph.D. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2011.

He is currently an Assistant Professor with the Department of Computer Engineering, Iran University of Science and Technology (IUST), Tehran. He has established and has been the Chair of the System Reliability and Test Laboratory, IUST, since 2011. He has authored or co-authored more than 40 papers in reputable journals and conferences. His current research interests include reliable nanoscale systems design, dependable embedded systems, and low-power and fault-tolerant computer architectures.