An Energy Efficient Circuit Level Technique to protect Register File from MBUs and SETs in Embedded Processors

M. Fazeli, A. Namazi, S.G. Miremadi
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
{m_fazeli, namazi}@ce.sharif.edu, miremadi@sharif.edu

Abstract

This paper presents a circuit level soft error-tolerant-technique, called RRC (Robust Register Caching), for the register file of embedded processors. The basic idea behind the RRC is to effectively cache the most vulnerable registers in a small highly robust register cache built by circuit level SEU and SET protected memory cells. To decide which cache entry should be replaced, the average number of read operations during a register ACE time is used as a criterion to judge. In fact, the victim cache entry is one which has the maximum read count. To minimize the power overhead of the RRC, the clock gating technique is efficiently exploited for the main register file resulting in significantly low power consumption. The RRC is able to protect the register file not only against Single Bit Upsets (SBUs) but also against Multiple Bit Upsets (MBUs) and Single Event Transients (SETs). The RRC is experimentally evaluated using the LEON processor. The experimental results show that, if the cache size is selected properly, the Architectural Vulnerability Factor (AVF) of the register file becomes about 1% while it imposes low power, area and performance overheads to the processor.

1. Introduction

High energy particle strikes have severely challenged the reliability of today's embedded processors [19][20]. Until recently, Single Bit Upsets (SEUs) in memory elements were considered as the main effect of the particle strikes. However, as technology shrinks to nanometer era, Single Event Transients (SETs) in combinational parts and Multiple Bit Upsets (MBUs) are becoming serious reliability concerns [1][2][3].

The register file is a critical part of an embedded processor from reliability point of view [6][7]. This is because, the register file mostly holds useful data and is frequently accessed by the processor. This implies that, an error occurring in a register most likely propagates to other parts of the processor. Several error handling techniques such as parity or error correction codes (ECC) are used to protect the entire register file against SEUs in different processors [8][9]. However, protecting all of the register by ECC or other techniques such as the TMR are not viable solutions in embedded applications due to high power consumption overhead. In addition, the use of parity bits can just leads to error detection but not error correction. To correct errors, a recovery mechanism such as checkpointing and rollback recovery is needed to recover the correct state of the processor. Using such recovery mechanisms might violate real-time requirements of embedded applications if the recovery happens after the tasks deadlines.

[6][7] have shown that only some registers are vulnerable to SEUs simultaneously. Thus, protecting only the most vulnerable registers of the register file with SECDED code can significantly reduce the power and area overheads of such techniques. Such classes of techniques as provided in [6][7][26] that utilize the architectural vulnerability characteristics of the register file for the protection are regarded as architecture level techniques in this paper. However, even though such techniques are power efficient to protect the register file against SEUs, they cannot be employed to cope with MBUs and SETs as they use SECDED code as the main protection technique.

One effective way to combat MBUs and SETs in the register file is to use circuit level protected SEU/SET-tolerant memory cells in the design of the register file [11][12][13]. However, such techniques impose rather high power and area overheads when they are used to protect the entire register file.

In short, architecture level SEU-tolerant techniques have low power and area overheads but are not able to protect the register file against MBUs and SETs. On the other hand, circuit level techniques are effective in combating such errors, but they suffer from high power and area overheads. To bridge the gap, this paper proposes a technique, called Robust Register Caching (RRC), which efficiently combines the advantages of both circuit and architecture level techniques to mask MBUs and SETs in the register file with low power, performance and area overheads.

The RRC technique is based on four ideas: 1) employing a small register cache along the register file as an architecture level technique to store the most vulnerable registers; 2) building the register cache by circuit level protected SEU and SET memory elements. As each memory cell of the register cache is built by such memory elements, it is highly robust against MBUs and SETs; 3) using an efficient replacement policy to increase the cache residency time of a register during its ACE time. The number of read operations from registers is used in the replacement policy as a criterion for replacement, such that a victim entry is the one that has the most read count; and 4) using the RRC

978-1-4244-4421-2/09/$25.00

with a proper register cache size significantly reduces the number of write operations to the register file. Based on this fact, the clock gating is efficiently used to prevent power consumed due to unnecessary transitions in the input lines of the register file during the write operations.

The rest of the paper is organized as follows: Section 2 presents motivations of the paper. In Section 3, some preliminaries are introduced. Section 4 describes the proposed soft error-tolerant technique. The experimental evaluations and comparisons are reported in Section 5. Finally, Section 6 concludes the paper.

2. Motivations

In following, we will discuss about MBUs and SETs and their importance in challenging the reliability of today's digital systems, as well as the shortcomings of the existing techniques when confronting such events.

2.1. Multiple Bit Upsets (MBUs)

As mentioned earlier, while technology shrinks toward deep sub-micron, the probability of MBUs increases. This is because the distance between two memory cells in a word line decreases [14]. It has been reported in [14] that the probability of MBUs exceeds 10% for technology sizes below 50nm. In addition some studies have shown that 1–5% of the SEUs can cause MBUs [1]. MBUs can be classified into three types: 1) an energetic particle passes through multiple memory cells; 2) multiple particles strikes multiple memory elements at different times; and 3) a particle strike produces secondary particles that have enough energy to cause bit flips.

Using error correction codes (ECCs) is a well-known technique to protect memory against SEUs. However, the imposed area and power overheads to enhance the code to detect and correct MBUs increases quickly as the code is strengthened [10]. Moreover, reading and calculating ECC bits in each read operation imposes performance and power consumption overheads. The scenario gets even worse for register files with multiple read ports. One effective technique for MBU detection and correction is the joint use of bit interleaving and simple SEC-DED code. Bit interleaving is a memory layout in which physically adjacent memory cells are assigned to different logical words. For example in a k-way interleaving, k consecutive errors in a row or a column appear as k single errors in k different words. Therefore, using SEC-DED code can efficiently detect and correct these k bits errors. However, this technique imposes power overhead due to unnecessary reads from undesired words in a row. In addition, the use of long word lines and multiplexers for columns imposes additional area and performance overheads to the system. These overheads get worse as the degree of interleaving increases [10]. In some cases, the bit interleaving technique may negatively impact floor planning [15]. Moreover, all ECC based techniques correct the errors after the erroneous data is read. It means that it may take a long time between the occurrence of an error and the correction time. This may increase the occurrence of type 2 errors.

2.2. Single Event Transients (SETs)

Soft errors have long been the major concern for memory elements. But due to smaller feature sizes, lower voltage levels, higher operating frequencies, and reduced logic depth of today's digital systems, the soft error rate due to particle strikes in combinational parts i.e. SETs is dramatically increasing [2][13]. A study shows that the soft error rate per chip of logic circuits will increase nine orders of magnitude from 1992 to 2011 [3]. When a high energy particle strikes a sensitive node in combinational logics, it causes a voltage glitch, the so called SET. An SET may results in a soft error if it reaches input lines of system latches or flip flops at the edge of the clock pulse. The conventional error correcting codes cannot be used for masking SETs. This is because, when the ECC of a word is being computed, the SETs occurring in the combinational logic may also be incorporated in the computation. Therefore, the code is computed for the erroneous data and no error will be detected when decoding. In addition, hardware redundancy based techniques, such as conventional duplication or TMR cannot tolerate SETs, since SETs reaching the input lines of the memory cells will also propagate to the redundant copies of the cells. Using time redundancy based methods are effective to mask SETs. For example, sampling the outputs of the combinational parts at three different times and voting will efficiently mask SETs. Although such techniques are efficient, they suffer from a high performance overhead due to time redundancy and also high area and power overheads due to use of additional hardware components to support multiple samplings.

3. Preliminaries

In this section, the preliminary concepts that help in better understanding the proposed technique and the employed reliability evaluation method. In addition, the circuit level techniques for protecting latches and flip flop (FFs) against SEUs and SETs will be discussed.

3.1. Architectural Vulnerability Factor (AVF)

The errors are classified as either Silent Data Corruption (SDC) or Detected Unrecoverable Errors (DUE). The SDCs are those errors that cannot be detected and the DUEs are the ones that can be only detected and there is no mechanism to recover from them.

In order to estimate the reliability of the proposed register file architecture, the AVF (Architectural Vulnerability Factor), a widely used metric [6][7][17], is employed in our experiments. The AVF of a part is the probability that a fault occurring in the part results in an error. The AVF can be measured for both SDCs and DUEs separately, and are called SDC AVF and DUE AVF [6]. To measure the AVF of a register, we should first extract the fraction of time in which the register is vulnerable to faults, dubbed ACE time (Architecturally Correct Execution) [17]. If a fault occurs in each bit of a register when it is in its ACE time, it will produce an error. In contrast, the un-ACE time of a register is the fraction of time in which a faulty bit in the register will not result in an error.

Finally, the AVF of a register is the percentage of time in which the register is in its ACE time.

Figure 1.a represents a typical register ACE and Un-ACE times. As shown in Figure 1.a, when a write operation is performed on the register, it enters its ACE time until the last read operation from the register. The time duration between the last read from the register and the next write to the register is considered as the Un-ACE time. If there is no read operation between two consecutive write operations (Figure 1.b), the register is always in its Un-ACE time. This means that a value is written to a register but never used.

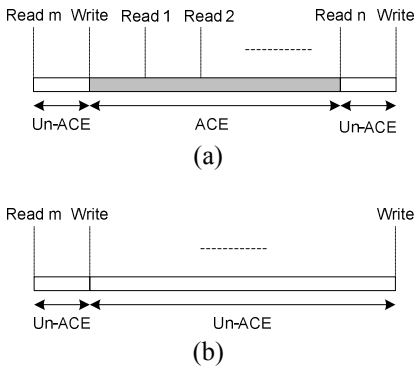


Figure 1. The ACE and Un-ACE time of a register

3.2. SEU/SET-Tolerant Latches & Flip Flops

Circuit level techniques for soft errors protection can effectively mask the effect of particle strikes i.e. SEUs and SETs. At this level of abstraction, extra circuitry is utilized to mask the transient voltage caused by particle strikes in the time of occurrence. The circuit level schematic of the SEU/SET tolerant latch proposed in [11][18] is shown in Figure 2. To mask the SEUs occurring in the internal nodes of the latch, a redundant feedback path and three filtering circuits called C-Element are used. The redundant feedback path provides a copy of the stored value inside the latch and the C-Element examines if the two values stored in the main and the redundant feedback paths are equal. The C-element is a state holding element that holds its previous state if its inputs are different and inverts its inputs if they are of identical logic value. To mask the SETs occurring in the combinational parts (which in turn can propagate to the input line of the latch), a CMOS delay element is utilized in the latch. To do this, the main input and its delayed version are compared using a C-Element. To minimize the cost of the latch, the already available C-Element used for the SEU tolerance purpose is also utilized for masking SETs.

It should be noted that the use of delay elements is a common practice in the design of reliable latches [19][20]. The amount of delay that should be applied to the input line of the latch depends on the amount of the required reliability. According to some studies [21], the SET pulse widths may be even more than few nanoseconds depending on the used technology and the particle energy. But the probability of a particle strike with a specific amount of charge decreases exponentially as the amount of deposited charge increases.

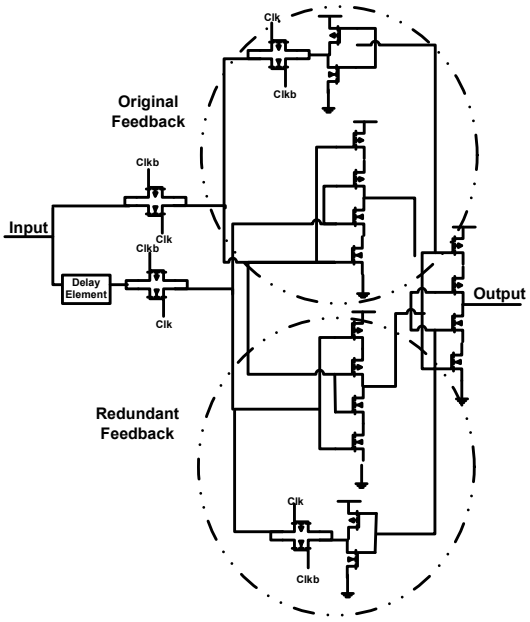


Figure 2. The circuit level schematic of the SEU/SET tolerant latch [11][18]

It means that the occurrence of a wide pulse is significantly less frequent than the narrow ones. In other words, if it is required that the latch filters out all possible SETs, the amount of the delay should be greater than the possible longest SET. Figure 3 shows a situation in which the SET pulse width is less than the amount of the applied delay. In this situation, the SET is masked by the C-element as it has occurred in the latch input and has not overlapped its delayed version.

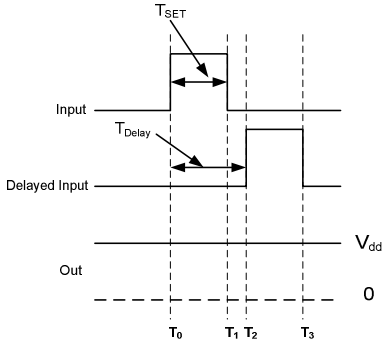


Figure 3. An example of an SET occurring in the input line of the latch ($T_{SET}<T_{Delay}$)

In contrast, when the SET pulse width is longer than the amount of the delay (Figure 4), the SET occurred in the input line of the latch overlaps its delayed version, which may result in a bit flip. However, detailed experimental evaluation of the latch for both SEUs and SETs can be found in [18]. A robust FF can be simply designed by the use of this latch. Figure 5 shows a master slave FF consisting of two robust latches. Using this robust FF, one can design an SEU/SET tolerant register.

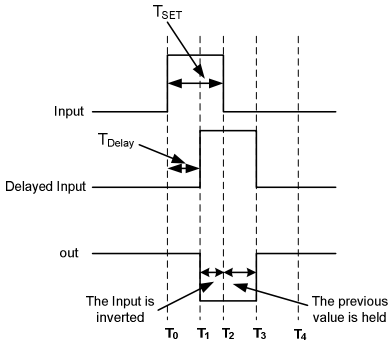


Figure 4. An example of an SET occurring in the input line of the latch (T_{SET}>T_{Delay})

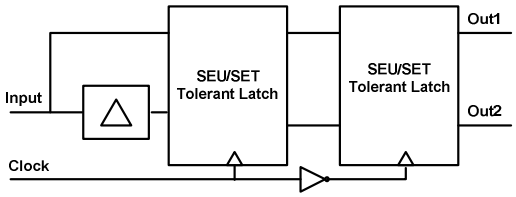


Figure 5. An SEU/SET tolerant FF

4. The Proposed MBU/SET Technique for Register Files

As mentioned in Section 2, the need for a fault tolerant technique that efficiently tolerates both SETs and MBUs while satisfying embedded system constraints, such as power consumption, is severely magnified. Circuit level techniques for protecting digital systems against particle strikes are very effective. In these techniques extra circuitry are used in the design of a memory element such as SRAM cells, latches or flip flops to prevent the voltage transients caused by particle strikes in sequential or combinational parts to turn into soft errors. In fact, they mask the effect of particle strikes in the time of occurrence. The main problem is that, exploiting robust latches to protect entire register file is not a viable solution as it imposes high power and area overheads. In the following section, we will show that how we can utilize the advantages of circuit level techniques to protect the register file from MBUs and SETs while providing attractive tradeoffs between reliability and power consumption.

4.1. The Simulation Platform and Benchmarks

Since the main target of this paper is embedded processors, A synthesizable VHDL model of the LEON2 processor that is designed for embedded application is used. LEON2 is a 32-bit processor conforming to the IEEE-1754 (SPARC V8) architecture [22]. The LEON register file has one 32-bits write port and two 32-bits read ports. LEON exploits register windowing. Number of registers depends on the configured number of register windows. The standard configuration is 8 windows requiring 136 registers. In this paper, two register windows consisting 64 registers are used. Some of the benchmarks of MiBench, An embedded benchmark suite [25], are used as the workload programs in our experiments.

4.2. Clock Gating

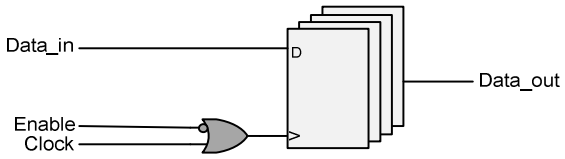
The two main source of power dissipation in CMOS circuits are static current caused by nominally off state transistors and dynamic power due to voltage transitions or switching activities. The dynamic power consumption is modeled as Eq.1:

$$P_{dynamic} = \alpha . C . {V_{dd}}^2 . f$$

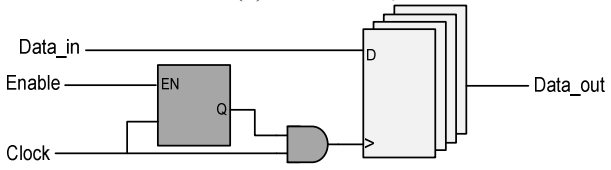
Where α is the switching activity, C is the total load capacitance, Vdd is the supply voltage and f is the operational frequency of the circuit. Different low power design techniques try to decrease one of these factors to reduce the dynamic power. For example, in dynamic voltage scaling technique (DVS), the frequency and supply voltage are scaled down. Although this technique significantly reduces the total power consumption, it has a negative effect on soft error rate of the system [5]. This is so since reducing the supply voltage results in an exponential increase in soft error rate [12][13].

Other low power techniques such as clock gating try to reduce the switching activity (α) by decreasing the unnecessary transitions. In the clock gating technique, the clock signal of the circuit is gated when there is no need to write a value to a memory element. These techniques have no negative effect on system reliability so that they are useful to reduce the power consumption in safety critical systems.

There are two styles for clock gating in memory cells, namely OR-based (Figure 6.a) which is employed in the proposed technique and Latch based (Figure 6.b). In both styles, the "enable" signal determines when the clock should be gated. The clock gating technique results in considerable power reduction in the processor if it is used in components that are frequently accessed and containing many memory cells such as the register file.



(a) Or-based style



(b) Latch Based style

Figure 6. Different styles for clock gating techniques

4.3. Register Caching

The basic concept of register caching is to use a small cache memory along the register file to decrease the register access time. In [23], it has been reported that there is a substantial locality in register values such that when a new value is written to a register, it takes a few cycles before the first read operation is performed. This small distance between writing a new value and its

first use suggests that the register caching could be beneficial.

In [23] a register caching technique is proposed for high performance processors exploiting Out-of-Order Execution technique. The register cache contains register values bypassed within a processor pipeline. If the cache size is properly selected, the register cache supplies most of the bypassed values. In [24] a use-based register caching technique is proposed to enhance the performance of the register caching. This is achieved by not caching the register values whose predicted consumers are satisfied by the bypass network. To achieve this goal, a use predictor is designed to predict the probable number of times a register value is used. Using this predictor prevents caching registers whose consumers are satisfied. These techniques are designed to reduce the register access time resulting in performance gain. Although register caching for enhancing the performance is a viable solution, it is not profitable in embedded applications. This is because, it imposes power overheads due to use of a cache memory and the additional hardware needed to manage the insertion policy, replacement policy and register use prediction. Moreover, using cache memory increase the occupation area both in combinational and sequential components resulting in higher soft error susceptibility. In fact, performance is not a first order concern in embedded systems design. Therefore, performance gain at the cost of power consumption increase or reliability decrease is not acceptable in such applications.

In [26], a technique called register value caching is proposed to enhance the reliability of the register file for ARM based embedded processors. In this technique, a duplicate version of register values is stored in a small register cache memory. The register cache is protected by the CRC codes. In each read operation, the register value in the register file is compared with its possible stored duplicate value in the register cache. If a mismatch occurs, the register value stored in the register cache is checked. If it is erroneous, the register value stored in the main register file is considered as the correct value otherwise the content of register cache is selected to perform the operation. This technique has two shortcomings: 1) it is not power efficient since in each read operation both register file and register cache are accessed and a comparison is performed; 2) similar to ECC based techniques, this technique cannot cope with SETs in the combinational part of the register file. In the following sections, firstly, we will investigate how the register caching can be used for protecting the register file against particle strike effects such as MBUs and SETs. Secondly, we will show that how the additional power overhead can be minimized with the use of register caching.

4.4. Robust Register Caching Technique (RRC)

In this section, we will show that how the register caching can be employed: 1) to store the most vulnerable registers in the register cache; 2) to protect the register file against MBUs and SETs; and 3) to provide conditions in which the clock gating technique can be efficiently used to decrease the imposed power overhead. For the sake of clarity, the proposed technique is explained by answering to the following two questions.

1) Can we efficiently employ register caching for reliability enhancement of the register file?

The registers are vulnerable to soft errors when they are containing useful data. In other words, the registers are vulnerable to soft errors when they are in their ACE time. In addition, not all allocated registers are simultaneously in their ACE time. To prove this claim, a set of simulations has been carried out to extract the average number of live registers for different bechmarks. To do this, the total number of registers which are in their ACE time are counted in five parts of the program execution i.e. when 20%, 40%, 60%, 80% and 100% of the program is executed. Finally, the average of the extracted values are shown in Figure 7. As shown in Figure 7, in the average case, maximum number of 25 registers are simultaneously in their ACE time. This implies that, it is not required to protect the entire register file during the whole program execution, but rather it is quite enough to protect only the registers when they are in their ACE time.

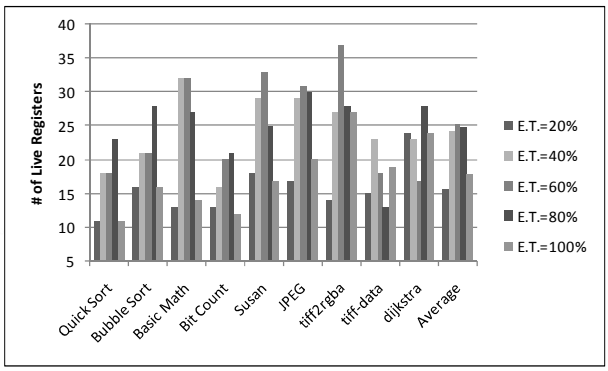


Figure 7, the number of live registers during program execution time (E.T=Execution Time)

Therefore, if we cache the register file in a way that most of the live registers are stored in the cache and if we protect the register cache with a proper protection technique, we can increase the reliability of the register file with lower power overhead as compared to protection of the entire register file. This is because; most of the live registers are read from the protected cache instead of non-protected register file.

2) How can we employ the cache to provide high level of reliability?

It was shown that using register caching is a viable solution to protect the register file. However, two questions still exist: 1) which protection technique should be employed in the register cache to efficiently protect it against MBUs and SETs?; 2) how can we guarantee that most of the registers are stored in the register cache during their ACE time?

As mentioned in section 3, the circuit level soft error protection techniques are most efficient techniques to cope with particle strikes as 1) they mask the effect of particles at the time of strike; 2) they can protect the circuit both from MBUs and SETs. The main problem is that the circuit level techniques suffer from high area and power overhead when they are used in power consuming parts of the processor such as the register

file. Since the register cache is considerably smaller than the register file, using the SEU/SET tolerant memory elements in the register cache is power and area efficient. In the RRC technique, the register cache is a fully associative cache built by SEU/SET-tolerant flip flops introduced in Section 3.

The main part of a caching technique is the replacement policy that determines the victim entry when the cache is full. As in the RRC technique, the main goal is to hold the registers in the cache when they are in their ACE time, the replacement policy should give the high priority to those registers which are less probable to reach their UN-ACE times. In other words, the victim entry is one which is containing a register value that most of its consumers are satisfied. Therefore, the number of read operation from a register value is a proper criterion to determine the victim cache entry. Consequently, the cache entry containing a register value with highest read count is selected as the victim entry. To reach the aim, a read counter is assigned to each entry of the cache and incremented in each read operation. The question arises here is how to determine the best size for the read counter.

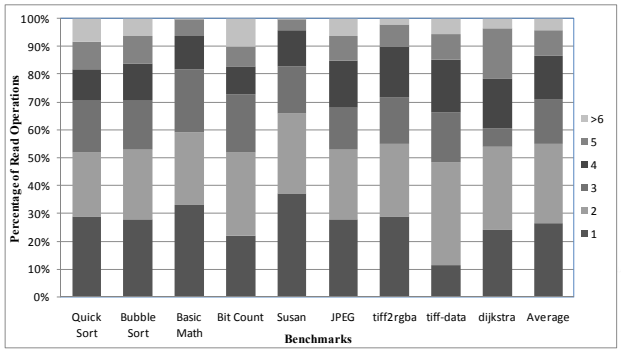


Figure 8, the average number of read operations from a register value

To determine the size of the read counter, a set of simulations has been done to extract the average number of read operations from a register during its ACE time. The simulation results shown in Figure 8 reveal that more than 90% of live registers have at most five read operations during their ACE time. This means that a three-bits read counter is sufficient for determination of the victim entry. The schematic of a register file protected by the RRC technique is depicted in Figure 9. For the sake of simplicity, the RRC technique for write to a register and read from a register operation is explained separately.

Write to a Register- in the RRC technique, all new values are stored in the register cache. It means that the new values are not written to the main register file. As shown in Figure 9, there is no data and address path between the processor core and the main register file. When a write operation to a register is issued by the processor, the new data and its delayed version are fed to the register cache. The delay elements are used since the robust FFs require both input and its delayed version for SET tolerance purpose (see Section 3.2). If there is an empty entry in the cache, the new value will be stored in that entry, otherwise one of two following scenarios occurs: 1) the new value is written to a register already cached with an old value, e.g. the new value is written to R0 while R0 already exists in the cache; and 2) there is no empty entry and no entry containing the same register is cached. In the first case, the new value is overwritten with the old one and there is no need to store the old value in the main register file. In the second case, a replacement should be done. In the employed replacement policy, the entry with maximum read count is selected as the victim entry. The value of the victim entry is then stored in a robust buffer. This buffer contains the data, register address, a validation bit that determines if the buffer is containing a victim value and a parity bit. The content of the buffer is copied to the main register file in the beginning of each read and write operations. In fact, the victim value is copied into the main register file in the next read or write operation. This prevents extra cycle imposed for writing the victim entry to the main register file.

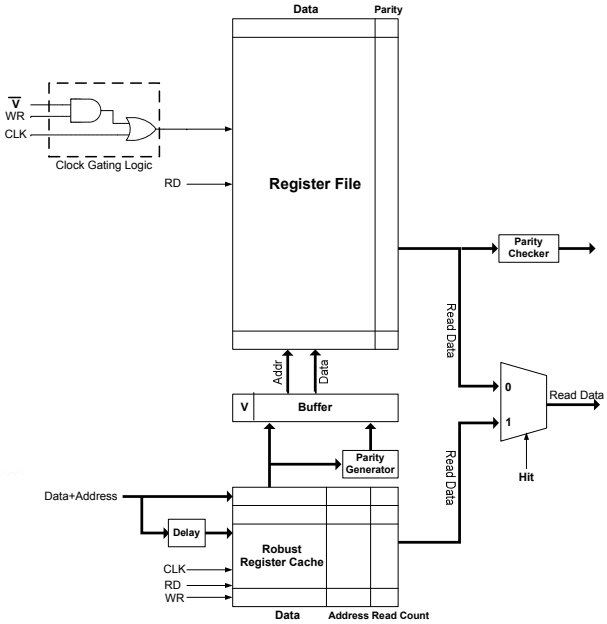


Figure 9. the RRC architecture

When a victim register is copied to the main register file, it is probable that it is still in its ACE time so that a parity bit is added to each register in the main register file. Therefore, when a victim value is copied to the buffer, the parity bit of the register value is computed and stored in the corresponding bit. In fact, one- bit parity is used for the main register file for protection of those registers that are transferred from the register cache to the main register file and are still in their ACE time.

To decrease the power consumption of the register file, the clock gating technique is utilized. As mentioned, all new values are written to the register cache, and the main register file is accessed only in read operations and when the buffer is containing valid data. Therefore, the clock of the main register file can be gated when a new data is written to the register cache and the buffer is not containing a valid data. In the next section, we will show that if the cache size is properly selected, clock gating technique can be efficiently employed to significantly reduce the total power consumption.

978-1-4244-4421-2/09/$25.00 ©2009 IEEE

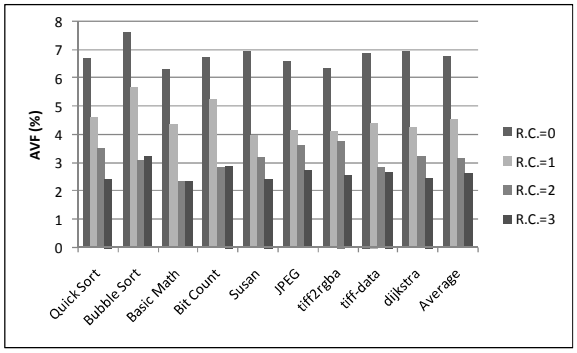


Figure 10. Register File AVF using cache size of 14 for different number of Read Count bit (R.C.)

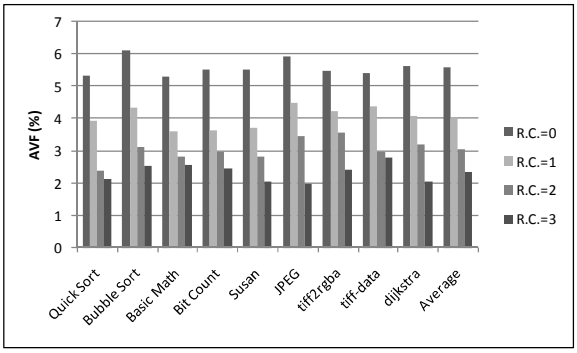


Figure 11. Register File AVF using cache size of 16 for different number of Read Count bit (R.C.)

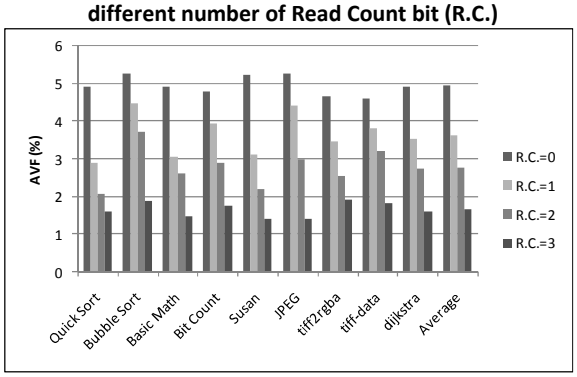


Figure 12. Register File AVF using cache size of 18 for different number of Read Count bit (R.C.)

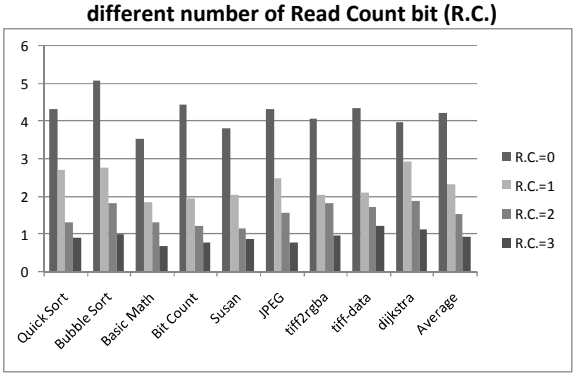


Figure 13. Register File AVF using cache size of 20 for different number of Read Count bit (R.C.)

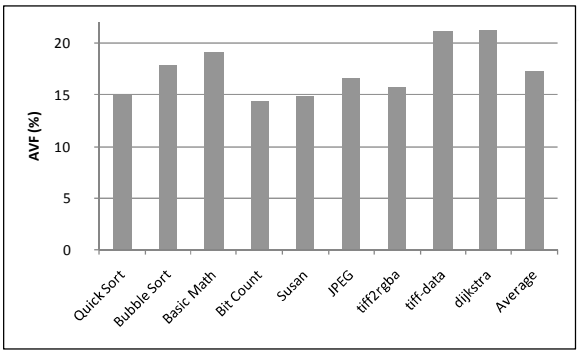


Figure 14. Non-protected Register File AVF

Read from a Register- in each read operation, both the register cache and the main register file are accessed. As shown in Figure 9, a multiplexer is used to select the data. If the value is in the register cache, the hit signal is set and the multiplexer selects the data from the cache. Otherwise, the read data from the main register file is connected to the multiplexer output. It should be noted that SETs occurring in the output lines of the multiplexer may be captured by the next pipeline stage of the processor resulting in a soft error. However, the proposed technique does not address these errors, but if we employ the introduced robust latch in the pipeline stages, theses errors can be simply masked.

When a miss occurs and the data is read from the main register file, the parity checking is done to check if an error exists in the read value. In the case of detecting an erroneous value, a rollback recovery mechanism can be used. After a value is read from a cache entry, the corresponding read counter is incremented. When the read counter reaches the maximum value, it is not incremented anymore until it is selected as a victim entry.

5. Experimental Results and Comparisons

As mentioned in Section 3.1, the AVF of the register file is considered as a metric for reliability estimation of the RRC technique. The two important parameters in the RRC technique affecting the register file AVF are the size of the register cache and the number of bits assigned for the read counter. In Figures 10, 11, 12, and 13, the register file AVF of the RRC technique with cache size of 20, 18, 16, 14 entries and the number of read count bits of 0, 1, 2, and 3 bits are reported. In Figure 14, the AVF of the non-protected register file is also shown. As it can be seen from these five figures, as the cache size increases for a fixed number of read counts, the register file AVF decreases. For example, for three bits of read count (R.C.=3), as the cache size changes from 14 (See Figure 10) to 20 entries (See Figure 13), the AVF decreases from about 3% to about 1%. This trend is still true, but with a higher amount, for a specific cache size with different sizes of the read count bits. For example, for the cache size of 14 (See Figure 10), as the number of read count bits increases, the AVF decreases from about 7% to 3%. This means that to gain more AVF improvement, not only the cache size is an important parameter, but also the number of read count bits considerably affects the improvement. The other important thing inferred from the above figures is that, using register caching even with zero read count bits, results in considerable improvement in register file AVF. For example for

the cache size of 20 and the read count size of zero bit, the register file AVF is 4% for the average case while it is about 15% in the Non-protected register file (See Figure 14). However, the AVF of the register file becomes less than 1% if we use three bits read count.

An MBU or SBU error may occur in the main register file or the register cache. When an MBU or SBU occurs, three situations exist: 1) the erroneous register value is in the main register file and is still in its ACE time. In this case, the MBU results in an SDC error while the SBU results in a DUE error as a parity bit is used for the main register file; 2) the erroneous register value is in the register file and it is in its UN-ACE time. In this case, the both MBU and SBU will not cause any failure; and 3) the erroneous register value is in the register cache. In this case, the error will be masked whether it is an MBU or SBU error, as it is built by robust memory elements. Based on these three cases, in the RRC technique, two kinds of error may exist: 1) SDC errors for MBUs; 2) DUE errors for SBUs. In fact, The SDC AVF for SBUs is zero in the RRC technique due to the use of a parity bit for all registers in the main register file. Therefore, the AVF results shown in Figure 10, 11, 12, and 13 are SDC AVF for MBUs and DUE AVF for SBUs. According to the simulation results, using the RRC technique with proper cache size and read count bits can results in significant reduction for MBU SDC and SBU DUE errors. This is because, the probability that a register value is in the main register file and is still in its ACE time significantly low. In other words, the situations 1 and 2 rarely happen.

To extract the power consumption of the RRC technique, we use the Synopsis power compiler tool for 180nm technology. As the register cache consists of circuit level protected memory elements, the robust register cache is simulated by HSPICE tool and its power consumption for read and write operations are measured for 180nm technology size using CMOS predictive transistor model [4]. The total power consumption of the RRC is then calculated by multiplying total number of both read and write operations by the amount of power consumed by a single read and a single write operation. Finally the total power consumed by the processor is measured. The power overheads reported in this section are the amount of power overheads imposed to the processor not to the register file. In Figure 15, the amount of power overheads of the RRC technique for cache sizes of 14, 16, 18, and 20 and three bits for the read count are depicted.

Based on Figure 15, the average processor power overhead increases from about 10% to about 15%, as the cache size increases from 14 to 20. This means that a 5% power overhead is imposed to the processor for reducing the average AVF of the register file from about 3% to 1%. It should be noted that, the main causes of the power overhead are: 1) SEU/SET-tolerant FFs used in the design of the register cache; 2) the read counter used for each entry of the register cache; and 3) word matching needed for the fully associative register cache.

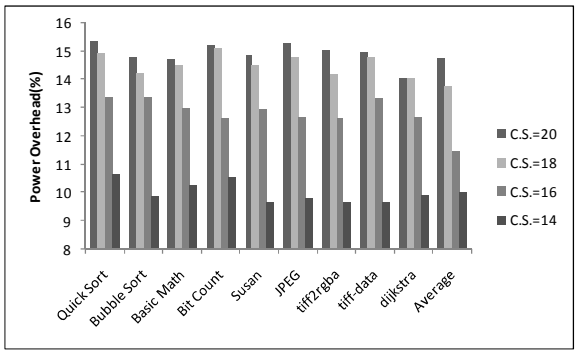


Figure 15. The RRC Power overhead for different cache sizes

To decrease the amount of power overhead, we have utilized the well known clock gating technique introduced in Section 4.2. As mentioned in Section 4, in the RRC technique, the new value is never written to the main register file. In fact, a write to the main register file occurs only when a victim value is transferred to the buffer. A value in the cache becomes victim if its cache entry is selected for storing the new computed value. It should be noted that, if the new value is written to a register already cached with an old value, the new value is overwritten to that entry and the old value is not considered as a victim value. For example, if R0 value is already stored in the cache and the processor executes a new write operation to R0, the new value is just written to the entry storing the old R0 value i.e. a victim case does not happen. Figure 16 shows the percentage of victim values for different cache sizes in different benchmarks. In other words, it shows the probability that a write operation results in a victim value. Figure 16 shows that, if the cache size is 20 (C.S. =20), only about 15% of all register write operations result in a victim value. In other words in about 85% of register write operations, the new value is written to the register that is already stored in the cache. Based on this observation, using clock gating to gate the clock of the main register file when there is no need to write a value, prevents unnecessary transitions in register file inputs resulting in considerable power reduction.

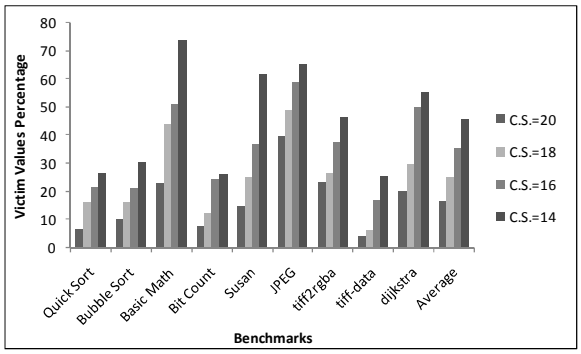


Figure 16. The average percentage of victim values as a function of Cache Size (C.S.)

The results of power overhead of the RRC technique when using clock gating are shown in Figure 17. It can be inferred from this figure that the power overhead considerably decreases by applying the clock gating technique. For example, in the cache size of 20, the power overhead is reduced from about 15% to about 9% using clock gating, i.e. 6% power overhead reduction. The interesting point that can be inferred from Figures 15, 16, and 17 is that the amount of reduction in power consumption decreases as the cache size decreases.

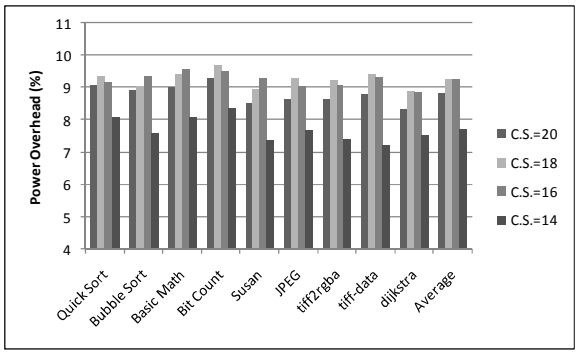


Figure 17. The RRC power overhead for different cache sizes using clock gating

This means that due to the use of clock gating technique, the amount of power saving in the cache size of 20 is more than that in the cache size of say, 18. This is because; when the cache size decreases, the frequency of writing to the main register file increases (See Figure 16) resulting in lower chance of clock gating and higher dissipated power by the register file.

To show the power efficiency of the RRC technique, its power overhead is compared to other well known SEU-tolerant techniques including SEC-DED, duplication with parity and TMR techniques. In SEC-DED technique, the entire register file is protected by SEC-DED codes. In duplication with parity technique, the entire register file is duplicated and a one-bit parity is used for each register. When a value is written to a register, its parity is computed and stored in the corresponding bit. In read operation, the parity is recomputed and compared with the previously computed one. If a mismatch occurs, the duplicated version is used; otherwise the original register is utilized. In the TMR technique, the entire register file is triplicated and a voter is utilized to compare the redundant register values. Figure 18 shows processor power overheads for the mentioned techniques as well as the RRC technique with cache size of 20. As shown in this Figure, RRC with the cache size of 20 has considerably lower power overhead as compared to other mentioned techniques.

The delay of the critical path increase is used as a metric for measuring performance overheads of the RRC technique. The slowest stage of the processor pipeline is considered as the critical path. To extract the delay of the critical path of the processor equipped with the RRC technique, the Synopsis design compiler tool is used. The performance overheads are extracted for the RRC technique with cache sizes of 20, 18 and 16 as well as the other SEU-tolerant techniques including SEC-DED, duplication with parity and the TMR techniques.

Figure 19 shows that the RRC technique has the lowest performance overheads compared to other soft error-tolerant techniques. It is noticeable that, the RRC technique with a cache size of 20 has negligible performance overhead of about 1%. In Figure 20, the area overhead of the RRC technique as well as the other mentioned techniques are shown. It should be noted that the area overhead is the overhead imposed to the processor not to the register file. Based on Figure 20, the RRC technique also has lower processor area overhead as compared to the other techniques.

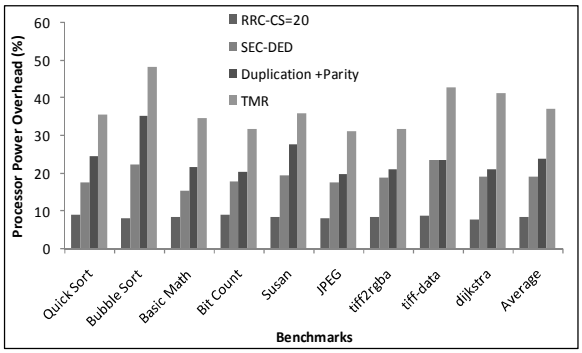


Figure 18. The power overhead comparison results

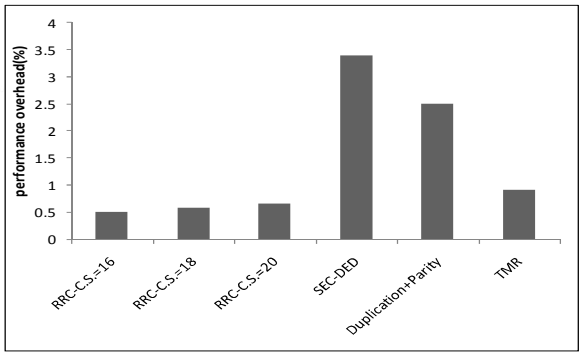


Figure 19. The performance overhead comparison results

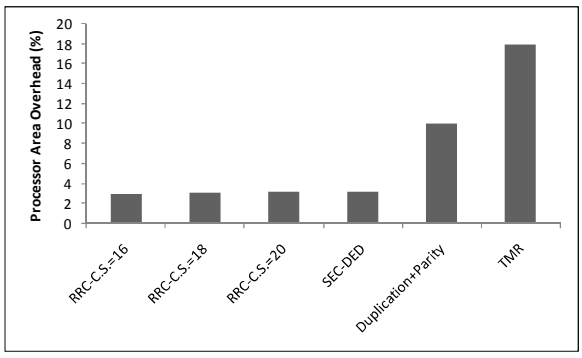


Figure 20. The area overhead comparison results

6. Conclusions

In this paper, a technique called Robust Register Caching (RRC) that effectively utilizes circuit level and architecture level techniques to protect the register file against MBUs and SETs is proposed. In the RRC technique, the most vulnerable registers are cached in a small highly robust memory built by

circuit level protected memory elements. The number of read operations from a register is used as a criterion for the cache replacement policy such that the victim cache entry is one which has the maximum read count. In addition, to minimize the power overhead of the RRC, the clock gating technique for the register file is efficiently exploited. The experimental results of RRC implemented on the LEON processor showed that it can significantly reduce the AVF of the register file while having low overheads in terms of power, area and performance.

References

[1] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong, "Characterization of Multibit Soft Error Events in Advanced SRAMs", Proc. of IEEE Int'l Electronic Device Meeting, pp. 519-522, Dec. 2003.

[2] F. X. Ruckerbauer and G. Georgakos, "Soft Error Rates in 65nm SRAMs--Analysis of new Phenomena," Proceedings of the 13th IEEE International On-Line Testing Symposium, pp. 203-204, 2007.

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

[4] A.J. Drake, A. KleinOsowski, A.K. Martin, "A Self-Correcting Soft Error Tolerant Flop-Flop", 12th NASA Symposium on VLSI Design, Coeur d'Alene, Idaho, USA, Oct. 4-5, 2005.

[5] A. Ejlali, B.M. Al-Hashimi, M.T. Schmitz, P. Rosinger, S.G. Miremadi, "Combined Time and Information Redundancy for SEU-tolerance in Energy-Efficient Real-Time Systems," IEEE Trans. on Very Large Scale Integration Systems, Vol. 14, No. 4, pp. 323-335, April 2006.

[6] P. Montesinos, W. Liu, J. Torrellas, "Using Register ACE time Predictions to Protect Register Files Against Soft Errors," IEEE Transactions on Dependable and Secure Computing (IEEE TDSC), To Appear, 2008.

[7] P. Montesinos, W. Liu, J. Torrellas, "Using Register ACE time Predictions to Protect Register Files Against Soft Errors," Proc. of 37th International Conference on Dependable Systems and Networks (DSN), June 2007.

[8] J. Gaisler, "Evaluation of a 32-bit Microprocessor with Built-In Concurrent Error-Detection," in International Symposium on Fault-Tolerant Computing, 1997.

[9] T. Slegel, I. Averill, R.M., M. Check, B. Giamei, B. Krumm, C. Krygowski, W. Li, J. Liptay, J. MacDougall, T. McPherson, J. Navarro, E. Schwarz, K. Shum, and C. Webb, "IBM's S/390 G5 Microprocessor Design," IEEE Micro, vol. 19, 1999.

[10] J. Kim, N. Hardavellas, K. Mai, B. Falsafi, J. C. Hoe, "Multi-bit Error Tolerant Caches Using Two-Dimensional Error Coding", Proceedings of the 40th Annual ACM/IEEE International Symposium on Microarchitecture (MICRO-40).

[11] M. Fazeli, A. Patooghy, S. G. Miremadi, A. Ejlali, "Feedback Redundancy: A Power-Aware SEU-Tolerant Latch Design in DSM Technologies," Proc. of the IEEE/IFIP Int'l. Conference on Dependable Systems and Networks, Edinburg, UK, June 2007, pp. 276-285.

[12] M. Omana, D. Rossi, C. Metra, "Latch Susceptibility to Transient Faults and New Hardening Approach," IEEE Trans. on Computers, Vol. 56, No. 9, pp. 1255-1268, September 2007.

[13] P. Hazucha, T. Karnik, S. Walstra, B. Bloechel, J. Tschanz, J. Maiz, K. Soumyanath, G. Dermer, S. Narendra, V.De, S. Borkar, "Measurements and analysis of SER tolerant latch in a 90-nm dual-Vt CMOS process", IEEE Custom Integrated Circuits Conference, Page(s):617 – 620, 2003.

[14] N. Seifert, P. Slankard, M. Kirsch, B. Narasimham, V. Zia, C. Brookreson, A. Vo, S. Mitra, B. Gill1, J. Maiz, "Radiation-Induced Soft Error Rates of Advanced CMOS Bulk Devices", Proceedings of the IEEE International Physics Symposium, pp. 217-225, 2006.

[15] A. Dutta, N. A. Touba, "A Low Cost Code-Based Methodology for Tolerating Multiple Bit Upsets in Memories," IEEE Workshop on System Effects of Logic Soft Errors, Apr. 2007.

[16] J. Patel, "Characterization of Soft Errors Caused by Single Event Upsets in CMOS Processes," IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 2, pp. 128-143, 2004.

[17] S.S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, T. Austin, "A Systematic Methodology to Compute the Architectural Vulnerability Factors of a High-Performance Microprocessor," Proc. of 36th Int'l Symposium on Microarchitecture (MICRO-36), IEEE CS Press, 2003.

[18] M. Fazeli, S. G. Miremadi, A. Ejlali, A. Patooghy, "A Low Energy SEU/SET-Tolerant Latch for Deep Sub Micron Technologies", To appear in the Journal of IET Computers & Digital Techniques.

[19] S. Mitra, M. Zhang, N. Seifert, T.M. Mak, S. K. Kee, "Soft Error Resilient System Design through Error Correction," Proc. of IFIP Int'l. Conference on Very Large Scale Integration, October 2006, PP. 332-337.

[20] W. Wang, H. Gong, "Edge Triggered Pulse Latch Design With Delayed Latching Edge for Radiation Hardened Application," IEEE Trans. on Nuclear Science, Vol. 51, No. 6, pp. 3626-3630, December 2004.

[21] J. M. Benedetto, P. H. Eaton, D. G. Mavis, M. Gadlage, T. Turflinger, "Variation of Digital SET Pulse Widths and the Implications for Single Event Hardening of Advanced CMOS Processes", IEEE Trans. on Nuclear Science, Vol. 52, No. 6, pp. 2114-2119, December 2005.

[22] Gaisler Resarch. Leon2 Processor User's Manual, Version 1.0.30, XST Edition. July 2005.

[23] R. Yung, N.C. Wilhelm, "Caching processor general registers", Proceedings of International Conference on Computer Design (ICCD,95), PP. 307-312, Oct. 1995.

[24] J. A. Butts, G. S. Sohi, "Use-based register caching with decoupled indexing," Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04), PP. 302-313, June 2004.

[25] Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, Richard B. Brown, "MiBench: A free, commercially representative embedded benchmark suite", *IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, December 2001.

[26] J. Blome, S. Gupta, S. Feng, S. Mahlke, and D. Bradley, "Cost efficient soft error protection for embedded microprocessors," Proc. of the Int'l conference on Compilers, architecture, and synthesis for embedded systems, 2006, pp. 421-431.