# CLASS: Combined Logic and Architectural Soft Error Sensitivity Analysis

Mojtaba Ebrahimi[1] Liang Chen[1] Hossein Asadi[2] Mehdi B. Tahoori[1]

[1]Department of Computer Science
Karlsruhe Institute of Technology
Karlsruhe, Germany
e-mails: {mojtaba.ebrahimi, liang.chen, mehdi.tahoori}@kit.edu

[2]Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
e-mail: asadi@sharif.edu

**Abstract— With continuous technology downscaling, the rate of radiation induced soft errors is rapidly increasing. Fast and accurate soft error vulnerability analysis in early design stages plays an important role in cost-effective reliability improvement. However, existing solutions are suitable for either regular (a.k.a address-based such as memory hierarchy) or irregular (random logic such as functional units and control logic) structures, failing to provide an accurate system level analysis. In this paper, we propose a hybrid approach integrating architecture-level and logic-level techniques to accurately estimate the vulnerability of all regular and irregular structures within a microprocessor. It carefully handles error propagation and masking scenarios among these structures. We have evaluated the vulnerability of the OR1200 processor using the proposed approach. Comparison with statistical fault injection shows an average inaccuracy of less than 7% with five orders of magnitude improvement in runtime.**

## I. Introduction

Feature size shrinking in nanoscale VLSI circuits results in a reduction of capacitance per transistor, and as a consequence particles with lower energy can generate sufficient charge to cause soft errors [1, 2]. In the absence of protection techniques, the soft error rate of the system will grow in direct proportion to the number of transistors in the design which increases exponentially according to Moore's law. Due to various masking factors such as electrical, timing, logic, architecture, and application, the vulnerability of different components of the system are quite non-uniform. Therefore, a fast and accurate soft error vulnerability analysis is essential to identify the most vulnerable components and apply selective protection techniques [3, 4].

*Soft Error Rate* (SER) estimation techniques can be generally categorized as fault injection [5–7] and analytical techniques [8–14]. Fault injection techniques are based on Monte-Carlo simulations and their accuracy can be improved by increasing the number of iterations [15, 16] at the expense of huge runtime. In contrast, fast analytical techniques can provide limited accuracy as they use simplified models of error propagation to expedite SER estimation.

Analytical techniques compute SER of a system at either architecture- or circuit- level. Most of the architecture-level techniques rely on the *Architecturally Correct Execution* (ACE) analysis [8] to compute the vulnerability of a structure. In the ACE analysis, the lifetime of a bit is divided into ACE and un-ACE intervals. A bit is un-ACE for an interval if a transient error at its value does not affect the final output of the program otherwise it is ACE. The *Architectural Vulnerability Factor* (AVF) of a structure is the fraction of time that this structure is in an ACE state. Since time intervals not proven as un-ACE are assumed to be ACE, the vulnerability of the target structure can be significantly overestimated. Rigorous fault injection experiment revealed that ACE analysis overestimates the vulnerability of some structures up to 7x [17]. The ACE analysis cannot consider error masking within complex logic structures, and therefore, this technique is mostly suitable for *regular structures* (a.k.a address-based structures) such as cache, register file, reorder buffer, and translation lookaside buffer (TLB) in microprocessors [9]. Moreover, the ACE analysis neglects circuit-level masking factors such as electrical and timing masking.

Most analytical techniques at circuit level are based on either *Boolean/Algebraic Decision Diagram* (BDD/ADD) [10–12] or *Error Propagation Probability* (EPP) [13, 14]. BDD/ADD-based techniques employ decision diagrams in order to find the failure rate of each output node. The accuracy of these techniques is very high, however, they are not scalable [18]. EPP techniques propagate the error from each error site to the primary outputs using a path based analysis and calculate the failure rate of each node. These techniques are very fast and offer reasonable accuracy for identification of vulnerable parts of the design. However, circuit level techniques are only applicable to *irregular structures* such as controllers and functional units. Such techniques are used to measure the error effect at a limited timing window consisting of few clock cycles. Moreover, they cannot consider application and system level masking factors.

Since microprocessors contain both types of regular and irregular structures, using either an architecture or a circuit level technique leads to an unacceptable inaccuracy. Additionally, independent analysis of these two types of structures using separated techniques results in significant inaccuracy as interactions among them causing error propagation and/or masking are completely ignored. Therefore, a unified approach which carefully considers the error propagation and masking due to interaction of different types of structures is crucial to obtain the accurate soft error vulnerability of the entire system.

In this paper, we propose an approach called *Combined Logic and Architectural Soft error Sensitivity analysis (CLASS)* to compute the soft error vulnerability of the entire microprocessor system consisting of regular and irregular structures. This hybrid approach uses an EPP analysis to model the propagation of errors in logic structures (i.e. combinational

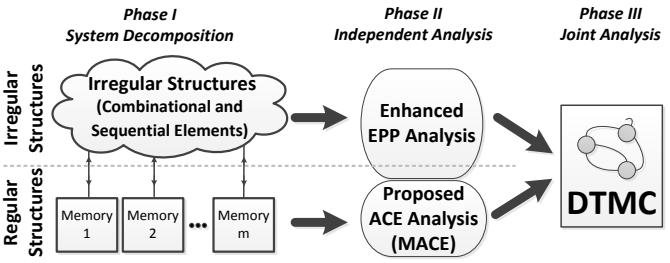


Fig. 1. Conceptional Block Diagram of CLASS

and sequential elements) and an ACE analysis to compute the vulnerability of memories. By considering the error rate of the interconnecting signals between regular and irregular structures, a *Discrete Time Markov Chain* (DTMC) is used to estimate the contribution of each structure to the overall system SER. Evaluation of the OR1200 processor using the CLASS approach is five orders of magnitude faster than *Statistical Fault Injection* (SFI) while its inaccuracy is less than 7%.

The rest of this paper is organized as follows: The CLASS approach is explained in Section II. The experimental results are presented in Section III. Finally, Section IV concludes the paper.

## II. The Proposed Approach (CLASS)

All of the previously proposed analytical techniques for SER estimation [8–14] are only suitable for either regular or irregular structures. In microprocessors, there are intensive interactions between these two types of structures. Consequently, an error in an irregular structure may propagate to a regular structure and contaminates it or the other way around. Additionally, some of the errors propagating from irregular structures to regular structures and vice versa are masked in the secondary structure. Considering these two effects, separate and independent treatment of regular and irregular structures in SER estimation results in significant inaccuracy (underestimation or overestimation) of system SER. To address this issue, we propose the CLASS approach that integrates an EPP analysis (for irregular structures typically represented at gate-level) and an ACE analysis (for regular structures, represented at micro-architecture level) in order to estimate the vulnerability of the entire system.

The key idea of the CLASS approach is the propagation of errors from regular to irregular structures and vice versa in a unified framework. The conceptional block diagram of this approach is shown in Figure 1. In the first phase, the system is decomposed into memories and logic blocks (i.e. combinational and sequential), representing regular and irregular structures, respectively. In the second phase, the proposed *Memory ACE* (MACE) analysis is used to calculate the propagation probability of an error in a memory cell to the memory output, the so called *Memory Vulnerability Factor* (MVF). On the other hand, an enhanced EPP analysis provides the probability of error propagation from combinational or sequential logic elements to the primary outputs of the system and inputs of each memory unit. Errors at the input ports of the memory may immediately propagated to its output (*short-term effect*) and/or affect its content (*long-term effect (a.k.a latent error)*), which may reside there for several cycles before getting masked or propagated to system outputs. Different error propagation scenarios between memory and logic are illustrated in Figure 2.

In CLASS, short-term effects are modelled by EPP analysis and in this case, similar to combinational and sequential elements, the error is propagated through memories. Modelling of long-term effect is more challenging. Since in most cases, latent errors in a memory unit are activated much later, they cannot be considered by EPP analysis. In this case, the latent error with probability of MVF is propagated to the memory output and then using an EPP analysis these errors are propagated through the logic. Note that it is quite possible that an error in a memory output results in another long-term effect in memories of the system. To address this issue, we have used a Discrete Time Markov Chain (DTMC) model to accurately consider such scenarios (see Figure 1).

### A. Analysis of Short-term and Long-term Effects

According to our investigation, an error at an input port of a memory unit may have short-term and/or long-term effects. If it affects the memory outputs in the next clock cycle, it has a short-term effect. For example, an error at the address port of a memory causes a wrong data (from another address) to be sent to the memory outputs. In contrast, if an error at an input port changes the contents of the memory, it has a long-term effect. For instance, an error at input data of a memory during write access contaminates the contents of the memory and the effect of this error remains in the memory unit until the next write access in the same address. The error in the memory contents will be propagated to the memory outputs during the following read accesses from the erroneous address.

Short-term and long-term effects of errors at input ports of a memory have been evaluated using an analytical approach which will be explained in the following. In this analysis, we assume that only one of the input ports of the memory is erroneous. For the sake of simplicity, consider a simple memory block as depicted in Figure 3.a which has four input ports named clock, address, write enable (WEn), and Data-in, and one output port named Data-out. As shown in the *Verilog* HDL description of the memory (see Figure 3.b), when the WEn signal is active, new data from the Data-in port is written into the memory at the rising edge of the clock signal.

Since the effect of errors at input ports are different during write and read accesses, the effect of errors at each input port when the write enable (WEn) signal is active or inactive should be analyzed separately. Let's assume that there is an error at the address port of the memory. When the write enable signal is inactive, data is read from a wrong address and consequently the output of the memory becomes erroneous (only short-term effect). However, when the write enable signal

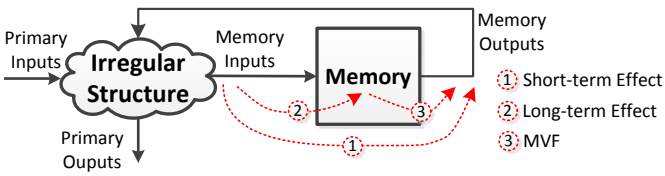


Fig. 2. Error Propagation Scenarios Between Irregular and Regular Structures in CLASS

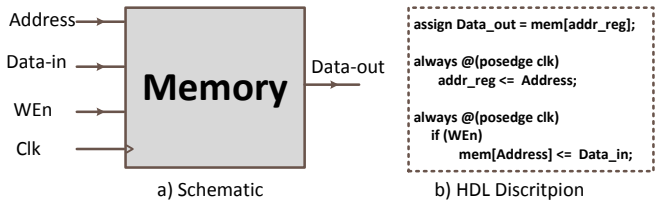


Fig. 3. a) Schematic and b) HDL Description of the Memory

is active, data is written to a wrong address and still the valid (correct) data appears at the output, meaning there is no error at the output while the memory contents is erroneous (only long-term effect). Briefly, an error at the address port with probabilities of $1 - SP(WEn)$ and $SP(WEn)$ has short-term and long-term effect, respectively, where $SP(WEn)$ is WEn signal probability. This analysis is summarized in Table I.

Now let us consider that there is an error at the data-in port. While the write enable signal is inactive, this error will be masked and it has no effect neither on contents nor on the output of the memory. If write enable signal is active, at the rising edge of clock, erroneous data is written into the memory and then this erroneous data appears at the output of the memory, meaning both memory contents and memory outputs are affected by the error. So, an error at the data-in port with probability of $SP(WEn)$ has both short-term and long-term effects and with probability of $1 - SP(WEn)$ has no effect.

If the write enable signal is inactive and due to an error it becomes active, data is written on the current address of the memory (write instead of read) and then erroneous data is propagated to the output. So, both memory contents and output are affected. On the other hand, if write enable signal becomes erroneous while it is active, a write operation is ignored. In the error-free case, this write operation changes the contents and the output of memory. In case of an error, these effects are ignored and both contents and output of the memory is different from the error-free case. Briefly, an error at the write enable signal has both short-term and long-term effects. According to Table I, if we assume that the error probability of input port $i$ is $P_e(i)$, the probabilities of having short-term ($P_{ST}$) and long-term ($P_{LT}$) effects are extracted using Equations (1) and (2), respectively.

$$P_{ST} = [1 - SP(WEn)] \times P_e(Address) + SP(WEn) \times P_e(Data\text{-}in) + P_e(WEn) \quad (1)$$

$$P_{LT} = SP(WEn) \times P_e(Address) + SP(WEn) \times P_e(Data\text{-}in) + P_e(WEn) \quad (2)$$

### B. Memory ACE (MACE) Analysis

Although ACE analysis techniques for memory consider the masking effect of other parts of the processor [9, 19, 20], as we discussed in Section I, these techniques cannot completely model this effect and in some cases there are significant differences between fault injection and ACE analysis results [17]. In CLASS, we use a life time analysis called Memory ACE (MACE) analysis to estimate the probability of error propagation from memory contents to its output (instead of program output which is used in conventional ACE analysis). The MACE analysis looks at the memory boundaries for the entire runtime of the program and profiles all accesses to the memory and according to access types detects ACE and un-ACE states. In CLASS, MACE is just used to propagate the error from memory contents to its outputs while propagation of the error from memory outputs to the other parts of the system will be considered by the EPP analysis.

Now, we show how MACE analysis can be done for the simple memory depicted in Figure 3. This method can be extended for other memory structures as it is much simpler than the conventional ACE analysis techniques which consider the masking effect of other components. In MACE, it is assumed that there is an error in the contents of the memory and the propagation probability of this error to memory output is calculated. This error is due to either an error occurred in the memory unit or propagation of an error from other parts of the system to the memory. Figure 4 shows an example of some serial accesses to a specific address of the memory during program runtime. In case that the next access to the specific address is a read, the error will be propagated to the memory output, while a write access overwrites the error. This means that all intervals leading to a read and a write access are ACE and un-ACE, respectively. Vulnerability of each address of the memory is the fraction of time that it is in the ACE state. The average vulnerability of all addresses is called Memory Vulnerability Factor (MVF). In CLASS, given a long-term effect due to an error at the input ports of the memory, the error will be propagated from memory contents to its output with its MVF probability.

### C. EPP Analysis

Our CLASS approach requires an EPP analysis for propagation of errors in combinational and sequential structures of the design. It should be able to calculate the probability of error propagation from each error site to all primary outputs as well as inputs of each memory unit. Additionally, this analysis should consider the short-term effect of errors at the input ports of the memory (Equation (1)).

We use the 4-Value EPP technique presented in [13, 21] in our framework. In this work, only logic masking has been considered as we want to compare the results of CLASS with logic-level SFI. Otherwise, an EPP technique with all three masking (i.e. logic, timing and electrical) factors can be

| Error at Port | Probability of Error | | | |
|---|---|---|---|---|
| | None | Output (Short-term) | Contents (Long-term) | Both |
| Address | 0 | 1 − SP(WEn) | SP(WEn) | 0 |
| Data-in | 1 − SP(WEn) | 0 | 0 | SP(WEn) |
| WEn | 0 | 0 | 0 | 1 |

TABLE I
EFFECT OF ERROR AT THE INPUT PORTS OF THE MEMORY

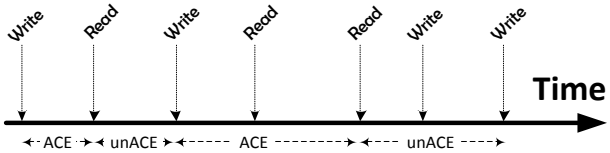


Fig. 4. Life Time Analysis of the Simple Memory

employed.

4-value EPP has the capability of propagating error probabilities in combinational and sequential elements during multi-cycle operation. For a given error site, in the first cycle, it propagates the error through combinational logic until it reaches to the flip-flops and primary outputs. In the next cycle, error probabilities are propagated from flip-flops through combinational logic. This operation is repeated for several cycles until error probabilities become very small. The experimental results in [21] revealed that most of the errors are propagated to the output within 10 cycles. More details about 4-value EPP can be found in [13, 21].

Since the original 4-value EPP technique does not include memories, a model for the memory is employed to consider the short-term effect of the errors (Equation (1)) at the input ports of the memory during multi-cycle error propagation. Additionally, using this model, the occurrence probability of long-term effects at each memory unit is calculated. Note that only the probability of having long-term effects is calculated in the EPP analysis while its effect will be considered in the DTMC, as we will explain later. In this regard, during each cycle of the multi-cycle analysis, probability of error propagation to each input port of the memory is calculated. As there might be a correlation between the errors propagated to the primary outputs ($P_{PO}$) and those propagated to memory inputs ($P_{Mem}$), we have used a simple method to calculate the joint probability of error propagation to memory inputs when it is not propagated to the primary output ($P_{Mem \cap \bar{PO}}$): ($P_{Mem \cap \bar{PO}} = P_{Mem \cup PO} - P_{PO}$).

After computation of uncorrelated error probabilities at the memory input ports, the probabilities of having short- and long-term effects are estimated using Equation (1) and (2). In case of short-term effect, there is an error at the memory output in the next cycle and it should be propagated through combinational logic. Summation of long-term effects for all cycles is reported as long-term effect of error after multi-cycle analysis. By this enhancement, 4-value EPP provides the probability of error propagation to primary outputs as well as the probability of having long-term effects in each memory unit.

### *D. Combining EPP and ACE Analysis using DTMC*

For each possible *error site* (combinational logic cell, sequential element, or memory cell), our CLASS approach can calculate the probability of error propagation to the primary outputs. In this work, if an error propagates to any one of the primary outputs of the system, it is assumed to be a failure. In CLASS, the propagation probability of an error from memory contents to its output is calculated using the MACE analysis while all the other probabilities (propagation to primary outputs, becoming long-term effect in each memory, etc) are estimated using the EPP analysis. The EPP analysis is applied to the entire design while MACE considers only the memory boundaries. Finally, a DTMC is employed to combine the results of MACE and EPP analysis and calculate the failure probability of each error site.

Generally, if a system has $m$ memory units, its DTMC has $2m + 3$ states. These states are as follows:

- **Error at error site:** Error just occurs at the error site.

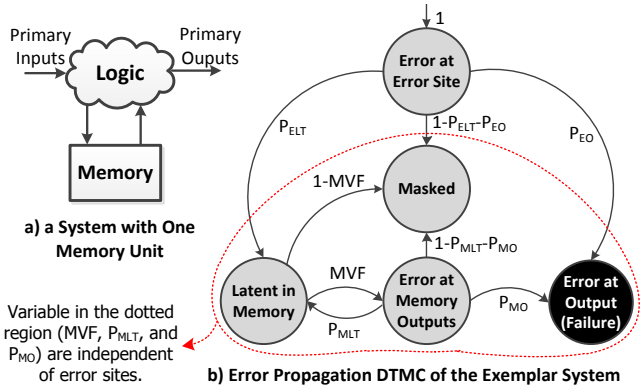


Fig. 5. A System with One Memory Unit and Its Error Propagation DTMC Diagram

| Computed by | Variable | Description |
|---|---|---|
| EPP | $P_{EO}$ | Probability of propagation of error to primary outputs |
| | $P_{ELT}$ | Probability of having long-term effect in the memory unit while it does not propagate to the primary outputs |
| | $P_{MO}$ | Probability of propagation of error from memory outputs to primary outputs |
| | $P_{MLT}$ | Probability of having long-term effect in the memory due to error propagation back from memory output |
| MACE | $MVF$ | Memory vulnerability factor |

TABLE II
DESCRIPTION OF VARIABLES IN DTMC DIAGRAM

- **Masked:** Error is masked.
- **Failure:** Error propagates to at least one of the primary outputs of the system.

Additionally, this DTMC has two states for each memory unit including:

- **Latent error in the memory:** Due to either the long-term effect or occurrence of an error in memory, memory contents are erroneous.
- **Error at the memory outputs:** There are errors at the memory outputs. This is due to the propagation of errors from memory contents to its outputs.

For the sake of clarity, we explain the propagation scenarios for a simple design with one memory unit and an arbitrary number of sequential and combination elements (depicted in Figure 5.a). Figure 5.b shows the error propagation DTMC diagram for this system. All variables used in this diagram, their meanings and how each variable is obtained are listed in Table II. As shown in Figure 5, in the first step an error occurs at the error site. In the next step, this error with probability of $P_{EO}$ is propagated to the primary outputs of the system and results in a failure. Also, an error at the error site may propagate to input ports of a memory and lead to a latent error in contents of the memory. These probabilities are extracted using the EPP analysis. In case the error is propagated to neither system outputs nor the memory inputs, it is masked (with probability of $1 - P_{EO} - P_{ELT}$). In case of a latent error in a memory unit, this error propagates to the output of the memory with probability of $MVF$. Next, the error propagates through logic circuit to system output or memory inputs with probabilities of $P_{MO}$ and $P_{MLT}$, respectively. In CLASS, it is

quite possible that an error propagates several times between different memory units and then it propagates to the system outputs.

This DTMC diagram can be easily solved as it has two dead states. Equation (3) shows the steady state solution of this diagram:

$$P_f = P_{EO} + \frac{P_{ELT} \times MVF \times P_{MO}}{1 - MVF \times P_{MLT}} \quad (3)$$

Using this equation, we can estimate the vulnerability of each error site of the design against soft errors. Note, most of the variables used in the DTMC diagram have the same value for all error sites with fixed system structure and application. For example in Figure 5.b, probabilities inside the dotted region (MVF, PMLT, PMO) are independent of the error sites and are calculated only once in advance. In this regard, we run an application on the system and extract the MVF for each memory unit using MACE analysis. Additionally, for each memory unit, by considering its output as error site in the EPP analysis, the probability of error propagation from there to primary outputs and other memory units is calculated. After computation of these fix variables, for each error site the propagation probabilities from the error site to primary outputs and all memory units are calculated using EPP analysis. Briefly, for vulnerability estimation of individual cell in the system, with $m$ memory units and $l$ combinational and sequential elements, MACE analysis runs just once for calculation of all MVFs, while multi-cycle EPP analysis is used $m + l$ times ($m$ times in advance for propagation of error from each memory outputs to other parts and $l$ times for each individual error sites).

In this work, we focus on *single bit flip* error model. However, CLASS can easily be extended to handle multiple bit flips. For this purpose, the *multiple bit upset* (MBU) implementation of the 4-value EPP [22] should be exploited. Moreover, the DTMC should be modified to consider more general cases in which multiple errors are present (latent) in the system, in various memory and logic blocks.

## III. Experimental Results

In order to show the efficiency and accuracy of CLASS compared to various alternative techniques, we have evaluated the vulnerability of the OpenRISC 1200 (OR1200) processor using this approach.

### A. OR1200 Processor and Benchmarks

OR1200 [23] is an open source 32-bit embedded processor with a five stage pipeline and single precision floating point unit. It implements Harvard memory architecture with separate data and instruction buses connected to data and instruction caches, respectively, and controlled by two independent cache controllers. Synthesis results using Design Compiler [24] and SAED 90nm library reveal that OR1200 has 44480 combinational gates, 4960 flip-flops and four single-port memory units and a three-port register-file. So, there are 49,444 possible error sites including all combinational logic cells, flip-flops, and single-port memory units in OR1200. Memory units are instruction cache (IC), data cache (DC) and their tags (ICTAG, DCTAG) which has the same implementation as the

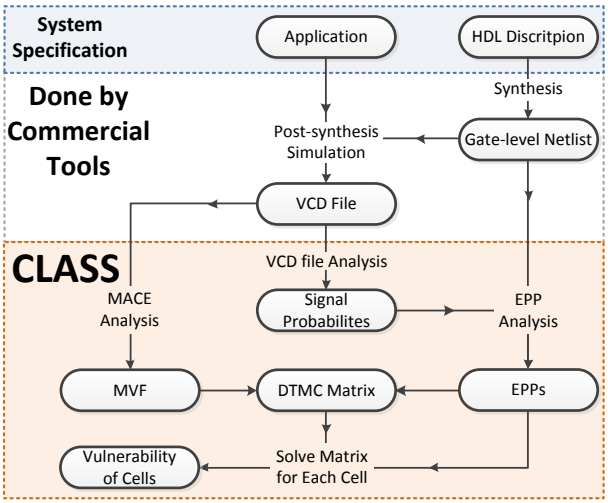


Fig. 6. Work Flow of Evaluation of a System Using CLASS

memory shown in Figure 3. Register-file is excluded form the memory as we do not have a model for propagation of errors through register-file. In this work, all errors propagated to register file and output pins of the processor are assumed to be a failure. Six programs from the Mibench [25] benchmark suite named Basicmath, Bitcount, CRC32, FFT, QSort, and Stringsearch are running on the processor during our experiments.

### B. Work Flow

The work flow of CLASS is shown in Figure 6. This approach takes the gate-level netlist of the microprocessor and a *Value Change Dump (VCD)* file as inputs, and calculates the soft error sensitivity of each error site in the design regarding the application running on it (see Figure 6). The gate-level netlist is extracted by synthesizing HDL description of the design. The netlist is simulated on a logic simulator (e.g. Modelsim) while an application is running on it and a VCD file is generated. VCD is an ASCII-based format dump file generated by logic simulation tools and includes the values of all system signals during simulation time.

First, signal probabilities of all signals in the system are extracted from the VCD file. Next, by analyzing the values of the signals connected to address and write enable pins of each memory in the VCD file, MVF of all memory units are calculated. Then, the probability of error propagation from each node of the design (i.e. error site) to primary outputs and memories inputs is calculated using 4-value EPP. Finally, for each error site, by combining EPP and MVF results, the DTMC Matrix is constructed. By solving the Markov model,

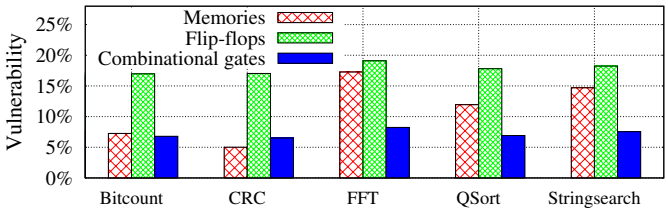


Fig. 7. Vulnerability of Mibench Benchmarks running on OR1200 obtained by CLASS

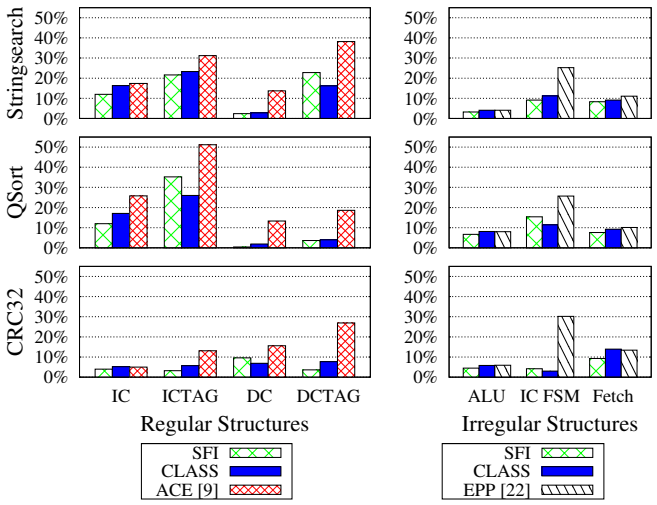


Fig. 8. Vulnerabilities reported by CLASS with SFI, ACE, and EPP

the vulnerability of each node is extracted.

Figure 7 shows the results of the vulnerability analysis extracted by CLASS. In this figure, the average vulnerability of all memories, flip-flops and combinational gates are reported separately. Note that if electrical and timing masking factors are considered in CLASS, vulnerability of combinational gates become lower than the numbers reported in this figure.

### C. Accuracy Analysis

In order to evaluate the accuracy of CLASS, we have compared it with SFI. Fault injection experiments are done using the Modelsim simulator [26]. Also, in order to verify that independent analysis is less accurate than CLASS, we have evaluated the OR1200 memories using ACE analysis presented in [9] and other logic elements using 4-value EPP presented in [21]. The fault model in fault injection experiments is bit-flip for duration of one cycle, so the results become comparable with those of CLASS. The results of evaluation using different approaches for three benchmarks are reported in Figure 8. All SFI results reported in this figure have at least confidence level of 97%. As it can be seen from this figure, results of CLASS in memory units (IC, ICTAG, DC, and DCTAG) are much closer to SFI results than those of ACE analysis. ALU and instruction cache FSM (ICFSM) and Fetch unit of the pipeline are selected as representative of irregular structures. ALU does not have direct interaction with memory units while ICFSM controls the IC and ICTAG memories and most of the errors in this component are propagated to these memories. Also fetch unit regularly accesses to the ICFSM during its operation meaning it has indirect access to the memory units. The results of vulnerability analysis for EPP and CLASS are very close in case of ALU, while there is a huge difference in the vulnerability results of ICFSM. Actually, the original EPP technique cannot handle the memories and all errors propagated to memory inputs are assumed to be a failure while CLASS has the capability of error propagation in memories, so it gives more realistic results. On average, CLASS has less than 3.4% inaccuracy for memories while for ACE analysis it is about 11.7%. With respect to maximum inaccuracy, CLASS is also better than ACE analysis as its maximum inaccuracy in case of memories is 9.2% while for ACE it is 23.3%. For ALU, ICFSM, CLASS has about 2% inaccuracy (max. inaccuracy 4.7%) in evaluation of logic elements while for EPP it is more than 7% (max. 26.3%). Although CLASS has 2% inaccuracy in evaluation of these components, analysis of results for individual error sites reveals that it has inaccuracy of less than 7% for each error site. However, as it overestimates or underestimates the vulnerability of different error sites, the overall inaccuracy is less than the inaccuracy of individual error sites.

There are three important sources of inaccuracy in CLASS. The most important one is the inaccuracy of 4-value EPP. Our experiments on OR1200 show that it has average inaccuracy of 2-3% for each error site and it may under- or overestimate the failure rate of different error sites. The second source is the simultaneous propagation of one error to several memory units or several input ports of the same memory. In this work, for such cases the most vulnerable memory unit/input port is considered and the effects on other memory units/input ports are ignored. Actually, considering such scenarios need more complicated DTMC which requires more runtime. The third source of inaccuracy is the multiple propagations of an error from a memory unit to its output. Basically, MACE only considers the probability of each address being in an ACE state. The read frequency between two write accesses for a specific address is also another factor which may affect the failure rate of the memory. Actually, it results in multiple propagations of an error to the memory outputs. These errors cannot be considered as independent errors as all of them propagate through the same path in the logic. So these errors should have similar behavior (propagation or masked) after being propagated to the memory output. However our experiments show that in some cases this assumption of similar behavior is not true and vulnerability of memory is a little bit higher than the case that the error propagates only once to the memory output. Resolving these sources of inaccuracy is in our future work.

### D. Runtime Analysis

All simulations are done on a workstation with Intel Xeon E5540 2.53GHz and 16GB RAM. On average, using CLASS approach, the evaluation of the entire OR1200 with all possible error sites for each benchmark takes less than one hour and it occupies less than 50MB memory (0.3%). The runtime of CLASS, ACE, EPP, and simulation-based SFI for three benchmarks are compared in Table III. The ACE analysis is done online while benchmark is running (being simulated) on the processor. So, its runtime is just a little higher than the runtime of the benchmarks on the processor. Although ACE analysis is very fast, it suffers from high inaccuracy. For EPP and CLASS, as shown in Table III, VCD file extraction and analysis is the most time consuming part. Since EPP does not include MVF and DTMC calculations, its overall runtime is slightly less than that of CLASS. Note most of the EPP and CLASS runtime is used for extraction and analysis of VCD files and less than 7 minutes is used for error propagations and solving DTMCs. In the SFI technique, fault injection into the OR1200 processor and simulating system (using Modelsim [26]) per each fault takes on average about 87 seconds. To achieve inaccuracy below 7% for each error site, according to

| Benchmark | Length (Cycles) | ACE [9] | EPP [21] | | CLASS | | Simulation-based SFI |
|---|---|---|---|---|---|---|---|
| | | | VCD Extraction and Analysis | Error Propagation | VCD Extraction and Analysis | Error Propagation and DTMC | |
| CRC32 | 671,455 | 69 | 1,827 | 261 | 2,031 | 379 | $49,444 \times 196 \times 59$ |
| QSort | 2,251,482 | 198 | 4,643 | 270 | 5,047 | 390 | $49,444 \times 196 \times 182$ |
| Stringsearch | 194,179 | 24 | 437 | 260 | 470 | 385 | $49,444 \times 196 \times 21$ |
| Average (Seconds) | | **97** | 2303 | 264 | 2518 | 385 | $49,444 \times 196 \times 87$ |
| | | | **2,567** | | **2,903** | | **843,119,088** |

TABLE III
COMPARISON OF RUNTIME OF ACE, EPP, SFI. AND CLASS (IN SECONDS)

[27], at least 196 faults should be randomly injected into each error site . This means evaluating all 49,444 error sites using SFI takes $49,444 \times 196 \times 87$ seconds $>$ 27 years. Hence, we have evaluated only some representative components. While our CLASS approach can provide the same level of accuracy in only 2903 seconds, it is five order of magnitudes faster than simulation-based SFI.

## IV. CONCLUSIONS

In this paper, we proposed a fast soft error vulnerability estimation approach, called CLASS, that has the capability of handling both regular and irregular structures. Experimental results revealed that the inaccuracy of CLASS is less than 7% for individual cells while it is five order of magnitudes faster than simulation-based statistical fault injection. Moreover, our results show that the existing approaches that can only handle regular (ACE) or irregular (EPP) structures have huge inaccuracies at system level. As CLASS can calculate the vulnerability of different cells, it provides the capability of identifying the most vulnerable cells of the processor and can be used along with selective protection techniques in order to increase the reliability of the system with minimum overhead.

## REFERENCES


[1] 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. In *Proceedings of Dependable Systems and Networks (DSN)*, pages 389–398, 2002.

[2] V. Chandra and R. Aitken. Impact of technology and voltage scaling on the soft error susceptibility in nanoscale cmos. In *In Proc. of the 2008 IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems (DFT)*, pages 114–122, 2008.

[3] 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 Transaction on VLSI Systems (TVLSI)*, 14(12):1368–1378, 2006.

[4] Q. Zhou and K. Mohanram. Gate sizing to radiation harden combinational logic. *IEEE Transaction on CAD of Integrated Circuits and Systems (TCAD)*, 25(1):155–166, 2006.

[5] M-L. Li, P. Ramachandran, U. R. Karpuzcu, S. K. S. Hari, and S. V. Adve. Accurate microarchitecture-level fault modeling for studying hardware faults. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, pages 105–116, 2009.

[6] A. Mohammadi, M. Ebrahimi, A. Ejlali, and S. G. Miremadi. SCFIT: A FPGA-based fault injection technique for SEU fault model. In *Proceedings of Design, Automation and Test in Europe Conference (DATE)*, pages 1–4, 2012.

[7] L. Entrena, M. García-Valderas, R. F. Cardenal, A. Lindoso, M. Portela-García, and C. López-Ongil. Soft error sensitivity evaluation of microprocessors by multilevel emulation-based fault injection. *IEEE Transaction on Computers (TC)*, 61(3):313–322, 2012.

[8] S. S. Mukherjee, C. T. Weaver, J. S. Emer, S. K. Reinhardt, and T. M. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of International Symposium on Microachitecture (MICRO)*, pages 29–42, 2003.

[9] A. Biswas, P. Racunas, R. Cheveresan, J. Emer, S. S. Mukherjee, and R. Rangan. Computing architectural vulnerability factors for address-based structures. In *Proceedings of the 32nd annual international symposium on Computer Architecture (ISCA)*, pages 532–543, 2005.

[10] B. Zhang, W. Wang, and M. Orshansky. Faser: Fast analysis of soft error susceptibility for cell-based designs. In *Proceedings of the 7th International Symposium on Quality Electronic Design (ISQED)*, pages 755–760, 2006.

[11] G. Norman, D. Parker, M. Z. Kwiatkowska, and S. K. Shukla. Evaluating the reliability of nand multiplexing with prism. *IEEE Transaction on CAD of Integrated Circuits and Systems (TCAD)*, 24(10):1629–1637, 2005.

[12] D. Bhaduri, S. K. Shukla, P. Graham, and M. Gokhale. Scalable techniques and tools for reliability analysis of large circuits. In *Proceedings of IEEE International Conference on VLSI Design (VLSID)*, pages 705–710, 2007.

[13] G. Asadi and M. B. Tahoori. An analytical approach for soft error rate estimation in digital circuits. In *Proceedings of International Symposium on Circuits and Systems (ISCAS)*, pages 2991–2994, 2005.

[14] L. Chen and M. B. Tahoori. An efficient probability framework for error propagation and correlation estimation. In *Proceedings of 18th IEEE International On-Line Testing Symposium (IOLTS)*, pages 1–6, 2012.

[15] P. Ramachandran, P. Kudva, J. W. Kellington, J. Schumann, and P. Sanda. Statistical fault injection. In *Proceedings of Dependable Systems and Networls (DSN)*, pages 122–127, 2008.

[16] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert. Statistical fault injection: quantified error and confidence. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 502–506, 2009.

[17] N. J. George, C. R. Elks, B. W. Johnson, and J. Lach. Transient fault models and avf estimation revisited. In *Proceesings of Dependable Systems and Networks (DSN)*, pages 477–486, 2010.

[18] N. Miskov-Zivanov and D. Marculescu. Modeling and optimization for soft-error reliability of sequential circuits. *IEEE Transaction on CAD of Integrated Circuits and Systems (TCAD)*, 27(5):803–816, 2008.

[19] N. J. Wang, A. Mahesri, and S. J. Patel. Examining ace analysis reliability estimates using fault-injection. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 460–469, 2007.

[20] S. Wang, J. Hu, and S. G. Ziavras. On the characterization and optimization of on-chip cache reliability against soft errors. *IEEE Transaction on Computers (TC)*, 58(9):1171–1184, 2009.

[21] M. Fazeli, S. G. Miremadi, H. Asadi, and S. N. Ahmadian. A fast and accurate multi-cycle soft error rate estimation approach to resilient embedded systems design. In *Proceedings of IEEE/IFIP International Confrences on Dependable Systems and Networks (DSN)*, pages 131–140, 2010.

[22] M. Fazeli, S. N. Ahmadian, S. G. Miremadi, H. Asadi., and M. B. Tahoori. Soft error rate estimation of digital circuits in the presence of multiple event transients (mets). In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 70–75, 2011.

[23] OpenRISC processor, www.opencores.org, july 2012.

[24] Synopsys Design Compiler, www.synopsys.com, 2012.

[25] 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 *Proceedings of IEEE International Workshop on Workload Characterization, WWC-4*, pages 3–14, 2001.

[26] Modelsim Simulator, www.model.com, 2012.

[27] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert. Statistical fault injection: quantified error and confidence. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 502–506, 2009.