A Design Approach for Soft Errors Protection in Real-Time Systems

Muhammad Sheikh Sadi, D. G. Myers, and Cesar Ortega Sanchez
Curtin University of Technology, Perth, Australia

***Abstract*—This paper proposes the use of metrics to refine system design for soft errors protection in system on chip architectures. Specifically this research shows the use of metrics in design space exploration that highlight where in the structure of the model and at what point in the behaviour, protection is needed against soft errors. As these metrics improve the ability of the system to provide functionality, they are referred to here as reliability metrics. Previous approaches to prevent soft errors focused on recovery after detection. Almost no research has been directed towards preventive measures. But in real-time systems, deadlines are performance requirements that absolutely must be met and a missed deadline constitutes an erroneous action and a possible system failure. This paper focuses on a preventive approach as a solution rather than recovery after detection. The intention of this research is to prevent serious loss of system functionality or system failure though it may not be able to eliminate the impact of soft errors completely**.

***Index Terms*—Reliability Metrics, Real-Time Systems, and Soft Errors Protection**

I. INTRODUCTION

Soft error is a temporary unintended change of state within a logic circuit that lasts for a few state transitions only. Soft error, also known as transient faults or Single Event Upset (SEU), is a rare phenomenon and usually not catastrophic [9], [18]. These types of faults could be induced by alpha particles from the naturally occurring radioactive impurities, high energy neutrons induced by cosmic rays, and low-energy cosmic neutron interactions with 10B found in boro-phospho-silicat glass (BPSG) [3], [17]. Though soft errors do not affect the internal structure of the semiconductor, they nevertheless lead to malfunctions and even failures of the circuit [7], [9], and [10].

Soft errors are a great concern for designing high availability systems or systems used in electronic-hostile environments such as outer space [14], [22]. These errors are also severe in those systems where reliability is a great concern [13]. Space programs, where a system cannot afford malfunction while in flight, are vulnerable to transient faults. Banking transactions, where a momentary failure may cause a huge difference in balance, are also threatened by transient faults [6]. For example, if a soft error causes $1 \rightarrow 0$ bit flips in the most significant bit of the register storing the amount of money deposited in a bank account then the effect might be an unexpected change in balance. Mission critical embedded applications, and even the execution of simple programs, where a corrupted intermediate value can corrupt all subsequent computations, are vulnerable to soft errors [19], [27].

Technology scaling, drastic shrinking in device sizes, associated with reduction in operating voltages and increase in clock frequencies result in increased susceptibility to soft errors [1], [2], [4], [14], [16], and [21]. Manufacturing design at advanced technology nodes-such as 90 nm, 65 nm, and onward-system level soft errors are much more frequent than in the previous generations [18]. Soft errors have traditionally been associated with the corruption of computer memory content. This phenomenon was reported as early as 1954. Since 1978, dense memory circuits, both DRAM and SRAM, have been known to be susceptible to soft errors caused by alpha particles from IC packaging and cosmic rays [15]. In the nanometer era, soft errors are no longer confined to memory cell upsets and can impact field-level product reliability for logic and latches. Additionally, with deeper pipelining, the number of logic stages between latches becomes smaller, increasing the probability of transient faults into latch [5]. Individual registers in a processor require an exorbitant amount of overhead due to these errors. Attack on a microprocessor, which is the core part of computer systems, makes the problem more critical. Exponential growth in the number of on-chip transistors, coupled with reduction in voltage levels, has made microprocessors extremely vulnerable to transient faults. The high clock rate of modern processors exacerbates the problem by increasing the probability that an incorrect signal in a combinational circuit is latched by a flip-flop [15]. This places designers in an unfamiliar realm in which logically correct implementations alone cannot ensure correct program execution. As a result, soft errors in flip-flops need immediate attention and due to technological advancements, solutions to handle combinational logic errors are essential.

Traditional approaches to prevent transient faults focus on recovery after detection. Almost no research has done on

Corresponding Author's e-mail: muhammad.sadi@postgrad.cutrtin.edu.au

preventive measures. Avionic systems, or any real time applications, cannot even tolerate recovery delay when there is a fault. In real-time systems, hard deadlines are performance requirements that absolutely must be met. A missed deadline constitutes an erroneous action and a possible system failure. In these systems, late data is bad data. For example, it would be awkward to have to reset the flight control computer because of a fault while the plane is in the air. Measures are needed to maintain functionality at all times. Past research has mostly considered using redundant hardware or software, or both, but this does not guarantee that real-time criteria can be met.

The aim of this paper is to examine a preventive approach as a solution rather than recovery after detection. Focusing on a preventive approach means that it is first necessary to consider what changes could affect the functionality desired, and then relate that to a demand for more robustness in the systems model. That requires some detailed assessment of both the functions to be provided and the structure and behaviour of the model. Whatever protection is nominated will flow through to the remaining stages and eventually end as some form of hardware or software, or both.

Clearly, testing conclusively across all structure-behaviour coordinates is a near-impossible task. Simplification is needed. This paper proposes the usage of metrics to reduce the size of the test space. These metrics are simply heuristics that are used to scan the system model and flag at what structure-behaviour coordinate a problem can arise. The aim is not to scan all points but look for key indicators that highlight particular conditions that need to be addressed. Thus the metric output will be some priority or it will be a measure of how long the impact of a transient fault may last, and so on.

II. Related Work

Researchers have evolved several measures to protect soft errors. Current approaches to soft error mitigation are shown in Figure 1.

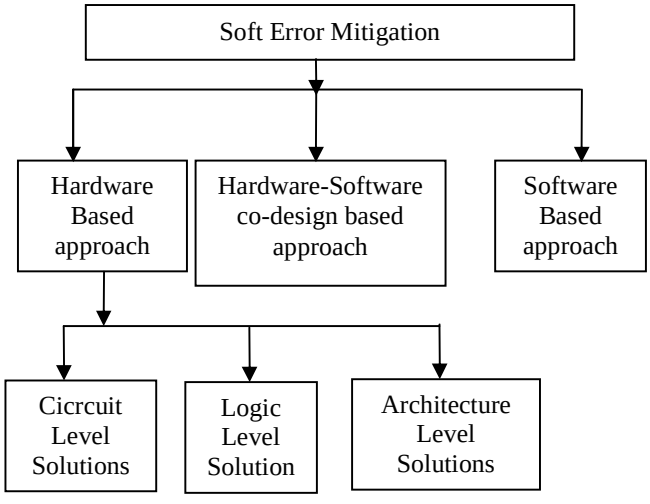

Figure 1. Soft Error Mitigation

Hardware solutions on soft error mitigation techniques mainly emphasize circuit level solutions, logic level solutions and architectural solutions. At the circuit level, the solution is mainly to increase the critical charge of a circuit node [8]. Logic level solutions [3], [4], [6], [11], and [18] mainly propose detection and recovery in combinational circuits by using redundant or self-checking circuits. For example, to validate the output of combinational circuits, these techniques used output parity generation. Validation of flip-flops is done by providing redundant latches or by using scan flip-flops to hold redundant copies of flip-flop data. Architectural solutions include dynamic implementation verification architecture (DIVA) [29], and block-level duplication used in IBM Z-series machines [17].

Software based approaches include redundant programs to detect and/or recover the problem [28], task duplication [20], by allocating, binding and scheduling resources in some definite ways [9] and by dual use of super scalar data paths [27]. Hardware and software co-design approaches [1], [2], [12], [23], [24], [25], and [26] use the parallel processing capacity of chip multiprocessors (CMPs) and redundant multi threading to detect and recover the problem.

Cazeaux et al. [8] showed circuit level fault tolerance/hardening techniques by keeping the critical charge (Q_{crit} which is a function of transistor sizing) of the circuit node as high as possible. They did not consider other issues of transient faults, such as cross section of the node, and charge collection efficiency of the node.

Tosun et al. [9] described the reliability model by allocating, binding and scheduling resources in a data flow graph based on resources' reliability. This approach provides no solutions when all proposed versions were tried and it still could not meet the performance bound. But for real-time systems, solution(s) must be there to avoid system failure or serious system loss.

Mitra et al. [18] used the inherent redundancy of latches with the Muller C-element to detect and tolerate soft errors in latches and flip-flops. C-element has two inputs and one output. If the two inputs match, the C-element acts as an inverter. If the input doesn't match then the previous value is retained. C-element tolerates any errors in the latches and flip-flops when the clock is 0, i.e. the latch is vulnerable to error. They did not cover bus lines, memory and register files.

Xie et al. [20] proposed the reliability-aware co-synthesis paradigm where they improved reliability of the system via task duplication. In their system, only a few tasks are duplicated that are defined by them as critical. Soft error may arise in any one of the tasks and its consequence can affect others as well.

Gomaa et al. [24] has developed a chip level redundantly threaded multiprocessor with recovery (CRTR) scheme for transient fault detection and recovery. There are certain faults from which CRTR cannot recover. If a register value is written prior to committing an instruction, and if a fault corrupts that register after the committing of the instruction then CRTR fails to recover that problem. Since CRTR commits the leading thread before checking and the trailing thread after

checking, and uses the trailing thread state for recovery, if any fault arises in the trailing thread itself, then the recovery may be wrong.

Chip level Redundant Threading (CRT) [25] used a load Value Queue (LVQ) such that redundant executions can always see an identical view of memory. Although LVQ produced an identical view of memory for both leading and trailing threads, integrating this into the chip multiprocessor environment requires significant changes.

DIVA [29] in its method of fault protection assumed that the checker is always correct and it proceeds using the checker's result in case of a mismatch. Faults in the checker itself must be detected through alternative techniques.

Xinping Zhu et al. [2] proposed a prototype of a fault tolerant multiprocessors system on chip. In their proposed work, they did not discuss the issue of bandwidth requirement and latency during inter-processor communication, and they also did not discuss the protection over memory and register files.

In the Simultaneously and Redundantly Threaded processors with Recovery (SRTR) scheme [26], there is a probability of a fault corrupting both threads since the leading thread and trailing thread execute on the same processor. SRTR checks speculative values to detect faults and its scheme of Dependence-Based checking Elision (DBCE) encounters problem with masking instructions, which may mask a fault in its inputs by producing the correct output even if an input is faulty.

Reinhardt et al. [28] described the concept of sphere of replication for aiding the design of fault tolerant simultaneously and Redundantly Threaded (SRT) processors. In short, the parts of the processor that fall outside the sphere are not replicated and must be protected by other means, such as information redundancy.

III. Methodology of the Research

Modern embedded systems design begins by constructing a single abstract model that captures the functionality demanded in the requirements specifications. In this research, UML has been chosen as a modeling tool. This research assumes that such a model might be created without considering the effect of transient faults. Specifically, this research will examine the use of metrics in design space exploration that highlights where in the structure of the model, and at what point in the behaviour, protection is needed against transient faults. Figure 2 symbolizes the plan in short.

The plan of using these metrics is outlined briefly in the following paragraphs.

A. Fan In and Fan Out

This metric measures the number of components connected to and from a particular component. 'Fan In' represents the number of connections to that component and 'Fan Out' represents the number of connections from that component. This metric finds out a set of components which could be interrupted if any transient fault arises in that component. The larger the number, the more critical the component is, and the greater the probability is of interruption by transient fault for the whole system. The next phase is to alter the UML representation of the system model by replacing the critical component with any other component(s) where Fan In and Fan Out will be lesser and no larger than a user defined threshold value. So, the process is iterated until all of the components do not cross the threshold value for Fan In and Fan Out.

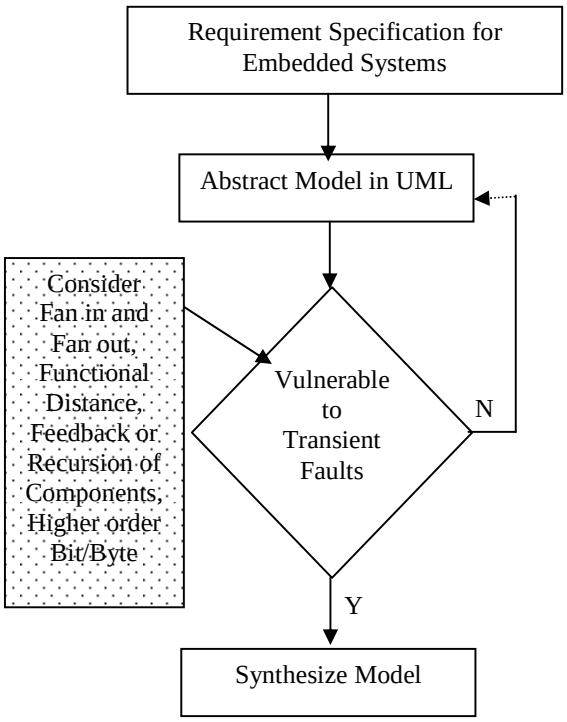

Figure 2: Methodology of Proposed Research

B. Functional Distance

This metric measure the functional distance of a node from starting node. Usually, if any transient fault arises at any hardware or software component then its effect continues to the next hardware or software module(s). If the fault starts at the beginning, then a devastating effect is almost certain. So, to prevent serious loss of system functionality or system failure, preventive measure(s) should be proportionately increased with the increase of closeness of the nodes with the starting node.

C. Feedback or Recursion of Components

The above metric measures the number of feedback or recursion with a component. If any component or any section is recursively used in the system model, then that particular component or section is more critical than a non-recursive one. And the number of iterations performed on any node obviously represents the seriousness of the necessity of precaution measure at that component/section.

D. Higher order Bit/Byte

A change in the most significant bit is more damaging than a change in the least significant bit of the register. For example, if a transient fault causes $1 \rightarrow 0$ bit flips in the most significant

bit of the register storing the amount of money deposited in a bank account, then the effect might be an unexpected change in balance; whereas, a change in the least significant bit is not as serious as in the most significant bit. So, the higher order byte/bit attracts more precaution than the lower one. This paper tries to highlight those portion(s) in the UML model, where this type of risk arises and then it tries to alter the model to such an extent where the risk is minimal.

IV. EXPERIMENTAL ANALYSIS

To investigate the performance of the planned metrics, several small example programs were written in Turbo C++ environment. Binary Editors was used to open the binary equivalence of source code and executable files. Faults were injected manually. Results were verified for the two cases: one is where metrics were considered and another one is where metrics were not considered. An error free example was used for the reference.

The first experiment was done using 'Fan In' and 'Fan Out' metric. This experiment can be described as follows:

Suppose X is a large unit that is composed by A, and B. Example 1 (some statements) uses X as an undividable component and if Y needs to use A then A must be represented by X-B.

Example 1

Y=(X-B) +Z; here X-B represents A since X is undividable.
W=Y+Z

If there is an error in B (but A is error free) then it will certainly affect Y, and W. And if there is an error in A then it will also affect Y, and W. But in Example 2 (some statements) if X can be divided into two separate components A, and B then an error in B will not affect Y, and W. Only an error in A can affect Y, and W.

Example 2

Y=A+Z
W=Y+Z

The codes were written in Turbo C++ language, and faults were injected using binary editors which can show the bit combination of codes. And the results show that when X is undividable then for error in B, it will affect the value of Y, and W. But when X is divided into A, and B then there is no change in the value of Y, and W for the error in B. This result verifies that the components with higher Fan in and Fan out should be divided into effective sub-components provided it will not affect other constraints such as processing time, power requirement or device size.

The next experiment was done using 'Functional Distance' metric. Example 3 (some statements) shows the following statements from a normal program.

Example 3

Y=A-B+Z
X=A+C
W=Y+Z
P=W+Z

In above examples, if there is a soft error in the first Instruction (say a bit change in B) then all other instructions that are using Y will be affected by the error. The result was checked by injecting soft error at different level of position of the instructions. This example not only proves that beginning instructions need much care to protect soft error but also beginning modules need extra care to protect soft error.

Thirdly the effect of 'Feedback or Recursion of Components' were verified by injecting soft error in those variables that are recursively used and/or feed backed several times and the frequent variables of a example program. This experiment finds that these variables spread error more than those other variables do.

Table 1 describes the experiment in short. First column shows the serial number of examples taken into account for

Table 1. Effect of Soft Error in several examples

Examples' Identification	How Soft Errors were Injected	Number of affected data
1 (Considering Fan In and Fan Out)	Fault injected into B	2
2 (Considering Fan In and Fan Out)	Do	Nil
3	Fault injected into A at the first statement	4
3	Fault injected into Z when it was used at first	3

the experiment. Second column shows where the soft error (only a bit change; either 1→0 or 0→1) were made. Third column shows the total number of variables that were affected due to only one bit change. At first fault was injected into B of Example 1. This fault affects 2 more data as a consequence. Second row describes that fault in B has no impact in Example 2 which is derived by considering Fan In and Fan Out. Third row shows that if the fault occur at the beginning of the program (here in A at the first statement) then it is affecting 4 more data consecutively. It emphasized that the beginning modules of any hardware or software model need extra precaution than others. And similarly in fourth case, since fault in the most frequent variable causing more damaging, any feedback or recursion of components of a model attracts more attention than others.

V. CONCLUSIONS

The literature on transient fault protection is remarkably small in spite of its increasing importance. As gate size decreases, so its incidence increases. At present, a number of fabrication plants are operating at 65 nm line widths, with new ones announced being of even smaller dimensions. Almost no research has focused on preventive measure(s) to tackle this problem. Though recovery after detection may offer a

temporary solution, for real time safety critical applications research must focus on preventive measure(s). By highlighting the key points where protection is needed, this paper shows the methodology to prevent serious loss of system functionality or system failure.

REFERENCES

[1] Smolens, J.C., et al.,"Reunion: Complexity-Effective Multicore Redundancy," in *IEEE/ACM International Symposium on Microarchitecture, MICRO-39*, Dec. 2006, pp. 223 - 234

[2] Xinping, Z. and Q. Wei.,"Prototyping a fault-tolerant multiprocessor SoC with run-time fault recovery," in *43rd ACM/IEEE Design Automation Conference*, 24-28 July. 2006, pp. 53 - 56.

[3] Zhang, M.,"Analysis and design of soft-error tolerant circuits," *Ph.D. Thesis*, University of Illinois at Urbana-Champaign, United States -- Illinois. 2006.

[4] Zhang, M., et al.,"Sequential Element Design With Built-In Soft Error Resilience," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions* on, vol. 14, no. 12, pp. 1368-1378, 2006.

[5] Narayanan, V. and Y. Xie,"Reliability concerns in embedded system designs," in *Computer*, vol. 39, no. 1, pp. 118-120, 2006.

[6] S. Mitra, M.Z., N. Seifert, TM Mak and K. Kim. Soft and IFIP,"Soft Error Resilient System Design through Error Correction," in *VLSI-SoC*, 2006.

[7] Mukherjee, S.S., J. Emer, and S.K. Reinhardt. ,"The soft error problem: an architectural perspective," in *11th International Symposium on High-Performance Computer Architecture*, San Francisco, CA, USA: IEEE (Comput. Soc.), 2005, pp. 243 - 247.

[8] Cazeaux, J.M., et al.,"On transistor level gate sizing for increased robustness to transient faults," in *11th IEEE International On-Line Testing Symposium, IOLTS 2005*, 6-8 July. 2005, pp. 23 - 28.

[9] Tosun, S., et al.,"Reliability-centric high-level synthesis," in *Proceedings Design, Automation and Test in Europe*, 2005, pp. 1258 - 1263.

[10] Crouzet, Y., J. Collet, and J. Arlat.,"Mitigating soft errors to prevent a hard threat to dependable computing," in *11th IEEE International On-Line Testing Symposium, IOLTS. 2005*, pp. 295-298.

[11] Krishnamohan, S.,"Efficient techniques for modeling and mitigation of soft errors in nanometer-scale static CMOS logic circuits," *Ph.D. Thesis*, Michigan State University, United States -- Michigan. 2005.

[12] Rashid, M.W., et al.,"Power-efficient error tolerance in chip multiprocessors," *Micro, IEEE*, vol. 25, no. 6, pp. 60-70, 2005.

[13] Gold, B.T., et al.,"TRUSS: a reliable, scalable server architecture," *Micro, IEEE*, vol. 25, no. 6, pp. 51-59, 2005.

[14] Saggese, G.P., et al.,"An experimental study of soft errors in microprocessors," *Micro, IEEE*, vol. 25, no. 6, pp. 30-39, 2005.

[15] Iyer, R.K., et al.,"Recent advances and new avenues in hardware-level reliability support," *Micro, IEEE*, vol. 25, no. 6, pp. 18-29, 2005.

[16] Borkar, S.,"Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Micro, IEEE*, vol. 25, no. 6, pp. 10-16, 2005.

[17] Meaney, P.J., et al.,"IBM z990 soft error detection and recovery," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, pp. 419-427, 2005.

[18] Mitra, S., et al.,"Robust system design with built-in soft-error resilience," *Computer*, vol. 38, no. 2, pp. 43-52, 2005.

[19] Tosun, S.,"Reliability-centric system design for embedded systems," *Ph.D. Thesis*, Syracuse University, United States -- New York, 2005.

[20] Xie, Y., et al.,"Reliability-aware co-synthesis for embedded systems," in *15th IEEE International Conference on Application-Specific Systems*, Architectures and Processors. 2004, pp. 41 - 50.

[21] Srinivasan, J., et al.,"The case for lifetime reliability-aware microprocessors," in *31st Annual International Symposium on Computer Architecture*, 19-23 June. 2004, pp. 276- 287.

[22] Wang, N.J., et al.,"Characterizing the effects of transient faults on a high-performance processor pipeline," in *International Conference on Dependable Systems and Networks*, 28 June-1 July. 2004, pp. 61 - 70.

[23] Kosovets, N.A. and L.N. Kosovets,"A fault-tolerant real-time multiprocessor with a built-in recovery mechanism," *Cybernetics and Systems Analysis*, vol. 40, no. 5, pp. 772, 2004.

[24] Mohamed, A.G., et al.,"Transient-fault recovery for chip multiprocessors," IEEE Micro, vol. 23, no. 6, pp. 76, 2003.

[25] Mukherjee, S.S., M. Kontz, and S.K. Reinhardt.,"Detailed design and evaluation of redundant multi-threading alternatives," in *29th Annual International Symposium on Computer Architecture*, 2002, pp. 99-110.

[26] Vijaykumar, T.N., I. Pomeranz, and K. Cheng.,"Transient-fault recovery using simultaneous multithreading," in *29th Annual International Symposium on Computer Architecture*, 2002, pp. 87-98.

[27] Ray, J., J.C. Hoe, and B. Falsafi.,"Dual use of superscalar datapath for transient-fault detection and recovery," in *34th ACM/IEEE International Symposium on Microarchitecture*, 1-5 Dec. 2001, pp. 214 - 224.

[28] Reinhardt, S.K. and S.S. Mukherjee.,"Transient fault detection via simultaneous multithreading," in *27th International Symposium on Computer Architecture*, 2000, pp. 25- 36.

[29] Austin, T.M.,"DIVA: a reliable substrate for deep submicron microarchitecture design," in *32nd Annual International Symposium on Microarchitecture*, 16-18 Nov. 1999, pp. 196 - 207.