International Journal of Electrical, Electronic and Communication Sciences
ISSN: 2517-9438
Vol:2, No:2, 2008

# Flagging Critical Components to Prevent Transient Faults in Real-Time Systems

Muhammad Sheikh Sadi, D. G. Myers, and Cesar Ortega Sanchez

***Abstract*—**This paper proposes the use of metrics in design space exploration that highlight where in the structure of the model and at what point in the behaviour, prevention is needed against transient faults. Previous approaches to tackle transient faults focused on recovery after detection. Almost no research has been directed towards preventive measures. But in real-time systems, hard deadlines are performance requirements that absolutely must be met and a missed deadline constitutes an erroneous action and a possible system failure. This paper proposes the use of metrics to assess the system design to flag where transient faults may have significant impact. These tools then allow the design to be changed to minimize that impact, and they also flag where particular design techniques – such as coding of communications or memories – need to be applied in later stages of design.

***Keywords*—**Criticality, Metrics, Real-Time Systems, and Transient Faults.

## I. Introduction

A transient fault is a temporary unintended change of state within a logic circuit that lasts for a few state transitions only. A transient fault, also known as soft error or Single Event Upset (SEU), is a rare phenomenon and usually not catastrophic. 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) [2]. Though transient faults do not affect the internal structure of the semiconductor, they nevertheless lead to malfunctions and even failures of the circuit.

Transient faults are a great concern for designing high availability systems or systems used in electronic-hostile environments such as outer space. These errors are also severe in those systems where reliability is a great concern [8, 9, and 11]. 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. 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. Mission critical embedded applications, and even the execution of simple programs, where a corrupted intermediate value can corrupt all subsequent computations, are vulnerable to transient faults [4], [12].

Muhammad Sheikh Sadi is with Curtin University of Technology, Australia (e-mail: muhammad.sadi@postgrad.curtin.edu.au).

Traditional approaches to prevent transient faults focus on recovery after detection. Almost no research has been done on 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 [13], 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 prevention 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 prevent soft errors. Hardware solutions for soft error mitigation 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 [10]. Logic level solutions [1], [7] 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 use 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) [19], and block-level duplication used in IBM Z-series machines [6].

Software based approaches include redundant programs to detect and/or recover the problem [18], duplicating instructions [16], task duplication [12], and by dual use of super scalar data paths [17]. EDDI [16] duplicate instructions and program data to detect soft errors which in turn create higher memory requirements and increase register pressure. Hardware and software co-design approaches [3], [5], and [14] use the parallel processing capacity of chip multiprocessors (CMPs) and redundant multi threading to detect and recover the problem.

One of the more interesting of these approaches [14] is 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.

Others [3], [15], and [18] have followed similar approaches. However, in all cases the system is vulnerable to soft error problems in key areas. Further, the complex use of threads presents a difficult programming model.

## 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, Unified Modeling Language (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, prevention is needed against transient faults. Fig. 1 symbolizes the plan in short.

The proposed metrics are outlined briefly in the following paragraphs.

### A. Structural Vulnerability Factor

This metric is the composition of 'Fan In' and 'Fan Out'; 'Functional distance'; and 'Feedback or recursion of components' metrics. It measures the structural vulnerability of the UML model. The constituents of this metric are described as follows.

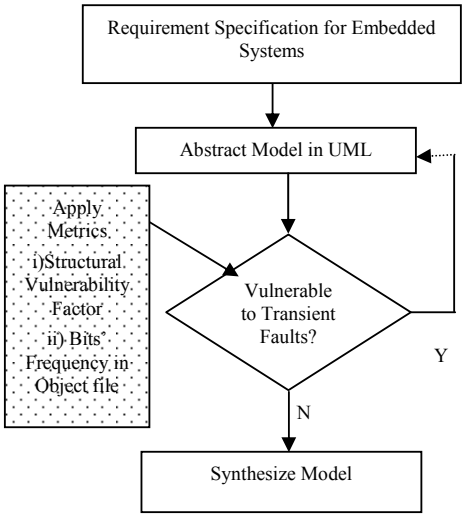


Fig. 1 Methodology of the proposed research

*1) 'Fan In' and 'Fan Out':* These 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. The metric classifies the set of components affected by any transient fault arising within that particular component. The larger the number, the more critical the component is, and the greater the probability is of interruption by a transient fault.

*2) Functional distance:* These metric measures the functional distance of a node from the starting node. Usually, if a transient fault arises within a component then its effect continues to all following connected components. A fault in the initial components will therefore tend to have a more devastating effect serious loss of system functionality or system failure proportionately increases with the increase of closeness of the nodes with the starting node.

*3) Feedback or recursion of components:* Feedback or recursion within a system model sees the impact of a transient fault continue for a possibly significant duration. Further, the larger the number of iterations, the more serious the potential problem is. It is important therefore, to identify feedback or iteration within the system model.

### B. Bits' Frequency in Object File

UML does not presently offer a simulation capability and so does not allow such measurements to be made. For that reason, the UML model was mapped into C++ to gain an executable program. The generated C++ code from Rhapsody tool was further processed to have a clear representation of the whole model including all states and transitions between states. Since transient faults only alter a bit value from 1→0 or 0→1, the object file of the C++ program was taken to inject a transient fault and for further tests. Binary Editor was used to open the object file where every state and transition was clearly monitored. Faults were injected at different states in

this object file. The frequency of faults at a particular state depends on the frequency of that state's bits in the object file. The more the frequency of any state's bits in the object file, the more susceptible the state is towards transient faults. This frequency is the ratio between the number of a state's bits in the object file and the total number of bits in the object file.

## IV. Experimental Analysis

To test the validity of the metrics, the following statechart diagram (Fig. 2) was used. The statechart diagram shows an on-board missile evasion system. There are six states (Idle, FoeDetected, Evading, CMsDeployed, FiringWeapons, and TimingOut); four different types of triggers (evFoeDetected, evEvade, evFire, and tm (delay time)) that cause change of states from one to another; two guards; and two actions in this statechart diagram. 'Idle' state is the initial state and the final state as well. At the 'Idle' state, if 'evFoedetected' signal is detected then it is transitioning to the 'FoeDetected' state. In the next phase, depending on the signals 'evEvade' or 'evFire', the transition is occurred to either 'Evading' state or 'FiringWeapons' state. At the 'FiringWeapons' state the value of the attribute 'wpnsToFire' is assigned the number 2 and the recursion continues until the value becomes zero. There are some other transitions where there is only time elapsing. At the end, the 'Idle' state is achieved again.

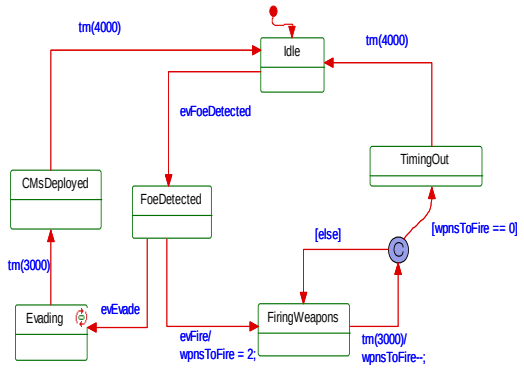


Fig. 2 Statechart diagram of on-board missile evasion system

Table I shows the characteristics of different states in the diagram based on different metrics. The description of the metrics is given in previous section. 'Idle' state is the initial and final state, and for testing purposes, it was not taken into consideration.

As stated, the UML model was mapped into C++ to gain an executable program. The object file of the C++ program was taken to inject transient faults and for further tests. Binary Editor was used to open the object file where every state and transition was clearly monitored. Table II shows the maximum effects of injected faults at different states. Two types of effects (number of affected states and number of operations) are shown in the table. The table shows that due to the fault injected at 'FoeDetected' state, the maximum number of affected states is 5 and the maximum number of affected operations is 6. In both cases, these values are the maximum in their corresponding columns.

TABLE I
The Characteristics of the Diagram Based on Different Metrics

| States | Fan In | Fan Out | Number of Iterations if Used Recursively | Number of States (Next to It) | Number of Operatio-ns (Next to It) |
|---|---|---|---|---|---|
| Foe Detected | 1 | 2 | - | 5 | 6 |
| Evading | 1 | 1 | - | 2 | 2 |
| CMs Deployed | 1 | 1 | - | 1 | 1 |
| Firing Weapons | 2 | 1 | 2 | 2 | 2 |
| Timing Out | 1 | 1 | - | 1 | 1 |

Several tests were performed where syntactical errors were occurred due to random transitioning of bits. Other than those errors, the rest of the bit transitions introduced faults which affected incoming states and operations. Table II summarizes the maximum effects that were caused by fault injections.

TABLE II
The Effect of Injected Faults at Different States

| States where faults were injected | Maximum Number of affected states | Maximum Number of affected operations |
|---|---|---|
| FoeDetected | 5 | 6 |
| Evading | 2 | 2 |
| CMsDeployed | 1 | 1 |
| FiringWeapons | 2 | 3 |
| TimingOut | 1 | 1 |

From the above tests, it can be observed that the structural vulnerability of a state in this model depends on the number of 'Fan In', 'Fan Out', iterations (if it is used recursively), possible incoming states next to it, and possible incoming operations next to it. If in a state we define the number of 'Fan In' as $F_i$, the number of 'Fan Out' as $F_o$, the number of iterations (if it is used recursively) as $I_r$, the number of possible incoming states next to it as $N_s$, and the number of possible incoming operations next to it as $N_o$, then the Structural Vulnerability Factor (SVF) of individual states can be defined as (1).

$$SVF_S = \frac{\sum_S (F_i, F_o, I_r, N_s, N_o)}{\sum_{j=1}^{n} (F_i{}^j + F_o{}^j + I_r{}^j + N_s{}^j + N_o{}^j)} \quad (1)$$

Where n is the number of states.

From (1):

$$SVF_{FoeDetected} = \frac{14}{39} = 0.39,$$

$$(\text{where } \sum_{FoeDetected} (F_i, F_o, I_r, N_s, N_o) = 14$$

$$and \sum_{j=1}^{n} (F_i{}^j + F_o{}^j + I_r{}^j + N_s{}^j + N_o{}^j) = 39)$$

The values of SVF for 'Evading', 'CMsDeployed', 'FiringWeapons', and 'TimingOut' states were calculated according to this formula and plotted in Table III.

In the next phase of the test, the frequency of a state's bits ($F_S$) in the object file, which is the ratio between the number of a state's bits in the object file and the total number of bits in the object file, was calculated. If this frequency for 'FoeDetected' state is $F_{FoeDetected}$ then its definition can be shown as (2).

$$F_{FoeDetected} = \frac{\text{number of bits in the object file for FoeDetected state}}{\text{total number of bits in the object file}} \quad (2)$$

$$= \frac{131}{485} = 0.27$$

And the values of ratio for 'Evading', 'CMsDeployed', 'FiringWeapons', and 'TimingOut' states were calculated according to this formula and plotted in Table III.

The Cumulative Vulnerability Factor (CVF) of a particular state $CVF_S$ is defined as (3).

$$CVF_S = \sum_S (SVF_S, F_S) \quad (3)$$

From (3):

$$CVF_{FoeDetected} = \sum_{FoeDetected} (0.39, 0.27) = 0.66$$

Similarly the CVF for 'Evading', 'CMsDeployed', 'FiringWeapons', and 'TimingOut' states were calculated according to this formula and plotted in Table III.

TABLE III
VALUES OF SVF, F AND CVF FOR DIFFERENT STATES

| States | SVF | F | CVF |
|---|---|---|---|
| FoeDetected | 0.39 | 0.27 | 0.66 |
| Evading | 0.15 | 0.17 | 0.32 |
| CMsDeployed | 0.10 | 0.08 | 0.18 |
| FiringWeapons | 0.23 | 0.40 | 0.63 |
| TimingOut | 0.10 | 0.08 | 0.18 |

After measuring CVF, user may define the criticality threshold to find the critical states. If for example the criticality threshold is 0.30 then it can be observed from Table III that there are three critical states and they are 'FoeDetected', 'FiringWeapons', and 'Evading' states. 'FoeDetected' is the most critical state and the criticality of 'FiringWeapons' is close to 'FoeDetected' state followed by 'Evading' state.

## V. CONCLUSION

This paper proposes the use of metrics to assess the system design to flag where transient faults may have significant impact. These tools then allow the design to be changed to minimize that impact, and they also flag where particular design techniques need to be applied in later stages of design. Hence whatever prevention is nominated will flow through to the remaining stages. Eventually this will end as some form of hardware or software, or both and thus will prevent transient faults in real-time systems design via a small shift in the design methodology.

## REFERENCES

[1] 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," Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, vol. 14, pp. 1368-1378, 2006.

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

[3] Z. Xinping and Q. Wei, "Prototyping a fault-tolerant multiprocessor SoC with run-time fault recovery," presented at 43rd ACM/IEEE Design Automation Conference , pp. 53 - 56, 2006.

[4] V. Narayanan and Y. Xie, "Reliability concerns in embedded system designs," Computer, vol. 39, pp. 118-120, 2006.

[5] M. W. Rashid, E. J. Tan, M. C. Huang, and D. H. Albonesi, "Power-efficient error tolerance in chip multiprocessors," Micro, IEEE, vol. 25, pp. 60-70, 2005.

[6] Meaney, S. B. Swaney, P. N. Sanda, and L. Spainhower, "IBM z990 soft error detection and recovery," Device and Materials Reliability, IEEE Transactions on, vol. 5, pp. 419-427, 2005.

[7] S. Krishnamohan, "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.

[8] R. K. Iyer, N. M. Nakka, Z. T. Kalbarczyk, and S. Mitra, "Recent advances and new avenues in hardware-level reliability support," Micro, IEEE, vol. 25, pp. 18-29, 2005.

[9] B. T. Gold, J. Kim, J. C. Smolens, E. S. Chung, V. Liaskovitis, E. Nurvitadhi, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk, "TRUSS: a reliable, scalable server architecture," Micro, IEEE, vol. 25, pp. 51-59, 2005.

[10] J. M. Cazeaux, D. Rossi, M. Omana, C. Metra, and A. Chatterjee, "On transistor level gate sizing for increased robustness to transient faults," presented at 11th IEEE International On-Line Testing Symposium, pp. 23 - 28, 2005.

[11] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," Micro, IEEE, vol. 25, pp. 10-16, 2005.

[12] Y. Xie, L. Li, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Reliability-aware co-synthesis for embedded systems," presented at 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors, pp. 41 - 50, 2004.

[13] M. Hiller, A. Jhumka, and S. Neeraj, "EPIC: profiling the propagation and effect of data errors in software," Transactions on Computers, vol. 53, pp. 512-530, 2004.

[14] A. G. Mohamed, S. Chad, T. N. Vijaykumar, and P. Irith, "Transient-fault recovery for chip multiprocessors," IEEE Micro, vol. 23, pp. 76, 2003.
[15] T. N. Vijaykumar, I. Pomeranz, and K. Cheng, "Transient-fault recovery using simultaneous multithreading," presented at 29th Annual International Symposium on Computer Architecture, pp. 87-98, 2002.
[16] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Error detection by duplicated instructions in super-scalar processors," Reliability, IEEE Transactions on, vol. 51, pp. 63-75, 2002.
[17] J. Ray, J. C. Hoe, and B. Falsafi, "Dual use of superscalar datapath for transient-fault detection and recovery," presented at 34th ACM/IEEE International Symposium on Microarchitecture, pp. 214 - 224, 2001.
[18] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," presented at 27th International Symposium on Computer Architecture, pp. 25- 36, 2000.
[19] T. M. Austin, "DIVA: a reliable substrate for deep submicron microarchitecture design," presented at 32nd Annual International Symposium on Microarchitecture, pp. 196 - 207, 1999.