Two Shortcomings in Software Design for Avionics: Timing Analysis and Soft Error Protection

Frank Mueller
Department of Computer Science, North Carolina State University, Raleigh, NC, *mueller@cs.ncsu.edu*

Abstract

This paper highlights two shortcomings in the current design process of embedded systems of avionics.

First, the current software design process does not adequately verify and validate worst-case timing scenarios that have to be guaranteed in order to meet deadlines. Consider the RTCA DO-178B standard requiring coverage testing. An additional requirement, namely predictable timing behavior, is essential real-time embedded systems. Airbus requires their suppliers to provide verifiable bounds on worst-case execution time of software for planes under development, Boeing is considering it (*e.g.*, for Airbus 380, Boeing 787 and military aircraft). The automotive industry, among others, is evaluating similar requirements. We provide an analysis of this problem that outlines directions for future research and tool development in this area.

Second, the correctness of embedded systems is currently jeopardized by soft errors that may render control systems inoperable. In general, transient faults are increasingly a problem due to (a) smaller fabrication sizes and (b) deployment in harsh environments. In commercial aviation, the next-generation planes (Airbus 380 and Boeing 787) will deploy off-the-shelf embedded processors without hardware protection against soft errors. Since these planes are designed to fly over the North Pole with an order of magnitude higher radiation (due to a thinner atmosphere), system developers have been asked to consider the effect of single-event upsets (SEUs), *i.e.*, infrequent single bit-flips, in their software design. Current developers do not know how to address this problem. We outline much needed research in this area.

1. Verification and Validation of Worst-Case Execution Times

Current software design for safety-critical embedded systems requires stringent compliance with coding standards to ensure safety and reliability. One example is avionics where the RTCA DO-178B standard requires coverage testing (for statements, branches and conditionals). A very important additional requirement for real-time embedded systems is predictable timing behavior of software components. In particular, so-called hard real-time embedded systems have **timing constraints that must be met or the system is may malfunction**. Airbus (and likely also Boeing in the near future), *e.g.*, requires their suppliers to provide verifiable bounds on worst-case execution time (WCET) for software to be deployed on planes currently under development (Airbus 380 and Boeing 787). The automotive industry is currently considering similar requirements, and others are likely to follow.

Determining bounds on the WCET of embedded software is a critically important problem for next-generation embedded real-time systems [1]. Currently, practitioners resort to testing methods to determine execution times of real-time tasks. However, testing alone cannot provide a verifiable (safe) upper bound on WCET. Exhaustive testing of inputs is generally infeasible, even for moderately complex input spaces due to its exponential complexity.

In contrast to dynamic testing, static timing analysis can provide *safe* upper bounds on the WCET of code sections, real-time tasks or entire applications. Hence, static timing analysis provides a safer and more efficient alternative to testing [2]. It yields verifiable bounds on the WCET of tasks regardless of program input by simulating execution along the control-flow paths within the program structure while considering architectural details, such as pipelining and caching [3].

These WCET bounds should also be *tight* to support high utilizations when determining if tasks can meet their deadlines *via* schedulability analysis. Tight bounds, however, can only be obtained if the behavior of hardware components is predicated accurately, yet conservatively with respect to its worst-case behavior. **Static timing analysis techniques are constantly trailing behind the innovation curve in hardware.** It is becoming increasingly difficult to provide tight *and* safe bounds in the presence of out-of-order execution, dynamic branch prediction and speculative execution. Simulation of hardware components is also prone to inaccuracy due to lack of information about subtle details of processors.

We advocate research on new approaches to bounding the WCET. Most importantly, a realistic hybrid approach is needed that combines formal static timing analysis with concrete micro-timings observations of actual architectures. First, a formal approach guarantees correctness. Second, dynamic timings on actual processors for small code sections will allow advanced embedded processor designs to be used in such time-critical systems, even in the presence of dynamic and unpredictable execution features. Third, any architectural modifications in support of such a paradigm have to be realistic in that they should reuse existing infrastructure both on the architecture side and the methodology for static timing analysis. There is an immediate need to develop software tools that can provide verifiable execution times to allow validation of task schedules within time-critical embedded systems.

2. Protection Against Soft Errors

Transient faults are becoming an increasing concern of system design for two reasons. First, smaller fabrication sizes have resulted in lower signal/noise ratio that more frequently leads to bit flips in CMOS circuits [4]. Second, embedded systems are increasingly deployed in harsh environments causing soft errors due to lack of protection on the hardware side [5]. The former reason affects computing at large while the latter is predominantly of concern for critical infrastructure. For example, the automotive industry has used temperature-hardened processors for control tasks around the engine block while space missions use radiation-hardened processors to avoid damage from solar radiation.

Current trends indicate an increasing rate of transient faults (*i.e.*, soft errors), not only due to smaller fabs but also because embedded systems are deployed in harsh environments they were not designed for. In commercial aviation, the next-generation planes (Airbus 380 and Boeing 787) will deploy off-the-shelf embedded processors without hardware protection against soft errors. Even though these planes are specifically designed to fly over the North Pole where radiation from space is more intensive due to a thinner atmosphere, target processors lack error detecting/correcting capabilities. Hence, system developers have been asked to consider the

effect of single-event upsets (SEUs), *i.e.*, infrequent single bit flips, in their software design.

In practice, future systems may have to sustain transient faults due to any of the above causes. There exists a significant amount of work on detection of and protection against transient faults. Hardware can protect and even correct transient faults at the cost of redundant circuits [6–14]. Software approaches can also protect/correct these faults, *e.g.*, by instruction duplication or algorithmic design [15–21]. Recent work focuses at a hybrid solution of both hardware and software support to counter transient faults [22–24]. Such hybrid solutions aim at a reduced cost of protection, *i.e.*, cost in terms of extra die size, performance penalty and increased code size.

We advocate novel research to address the problem of soft errors. Of interested are (1) software solutions and (2) hybrid hardware/software solutions. While a number of hardware solutions exist, commodity hardware is being deployed in systems subject to high rates of transient errors. In the complete absence of hardware support, **a software methodology to address soft errors needs to be developed that retains performance.** Current software schemes (*e.g.*, [16]) reduce the performance of systems considerably, if not prohibitively, and are not supported by tools. Further research is required to reduce this overhead to developing novel schemes to tolerate faults in software. **Hybrid solutions offer another promising avenue to address this problem. Minor architectural modifications that can be adopted within existing architectures should be accompanied by software solutions allowing soft errors to be detected at low overhead.** Early results [22, 23] outline the potential of such an approach but leave many facets for improvement open. Protection at the level of code and different data sections of programs can be specialized by tool support to significantly reduce overhead even further. **There is an immediate need to pursue innovative lines of research for soft error protection that have potentially high yields in performance while providing low error rates.**

3. Potential Impact

The proposed directions of research on verifiable execution times would benefit the embedded system community, specifically applications in avionics, automotive and safety-critical systems. Solutions to the soft error problem will benefit the increasing set of embedded applications in harsh environments, which comprise critical infrastructure in today's society. This is particularly true for to aircraft using commodity microprocessors for control systems. The results can further benefit the semi-conductor industry at large by complementing their efforts to counter problems, such as the decreasing signal/noise ratio in smaller fabrication feature sizes, with innovative, cost-effective methods.

References

[1] J. Wegener and F. Mueller. A comparison of static analysis and evolutionary testing for the verification of timing constraints. *Real-Time Systems*, 21(3):241–268, November 2001.

[2] R. Arnold, F. Mueller, D. B. Whalley, and M. Harmon. Bounding worst-case instruction cache performance. In *IEEE Real-Time Systems Symposium*, pages 172–181, December 1994.

[3] F. Mueller. Timing analysis for instruction caches. *Real-Time Systems*, 18(2/3):209–239, May 2000.

[4] C. Constantinescu. Trends and challenges in vlsi circuits reliability. *IEEE Micro*, pages 14–19, July-August, 1996.

[5] V.Narayanan and Yuan Xie. Reliability concerns in embedded system designs. *IEEE Computer magazine*, pages 106–108, January, 2006.

[6] Hisashige Ando, Yuuji Yoshida, Aiichiro Inoue, Itsumi Sugiyama, Takeo Asakawa, Kuniki Morita, Toshiyuki Muta, Tsuyoshi Motokurumada, Seishi Okada, Hideo Yamashita, Yoshihiko Satsukawa, Akihiko Konmoto, Ryouichi Yamashita, and Hiroyuki Sugiyama. A 1.3ghz fifth generation sparc64 microprocessor. In *Design Automation Conference*, pages 702–705, New York, NY, USA, 2003. ACM Press.

[7] Y.C. Yeh. Triple-triple redundant 777 primary flight computer. In *1996 IEEE Aerospace Applications Conference. Proceedings*, volume 1, pages 293–307, 1996.

[8] Y. C. (Bob) Yeh. Design considerations in boeing 777 fly-by-wire computers. In *IEEE International High-Assurance Systems Engineering Symposium*, page 64, 1998.

[9] Joel R. Sklaroff. Redundancy management technique for space shuttle computers. *IBM Journal of Research and Development*, 20(1):20–28, 1976.

[10] Todd M. Austin. DIVA: A reliable substrate for deep submicron microarchitecture design. In *International Symposium on Microarchitecture*, pages 196–207, 1999.

[11] Mohamed Gomaa, Chad Scarbrough, T. N. Vijaykumar, and Irith Pomeranz. Transient-fault recovery for chip multiprocessors. In *International Symposium on Computer Architecture*, pages 98–109, San Diego, CA, May 2003. ACM SIGARCH / IEEE CS. Published as Proc. 30th Ann. Intl Symp. on Computer Architecture (30th ISCA 2003), FCRC'03 ACM Computer Architecture News, volume 31, number 2.

[12] A. Mahmood and E. J. McKluskey. Concurrent error detection using watchdog processors - A survey. *IEEE Transactions on Computers*, 37(2):160–174, 1988.

[13] Joydeep Ray, James C. Hoe, and Babak Falsafi. Dual use of superscalar datapath for transient-fault detection and recovery. In *International Symposium on Microarchitecture*, pages 214–224, 2001.

[14] Steven K. Reinhardt and Shubhendu S. Mukherjee. Transient fault detection via simultaneous multithreading. In *International Symposium on Computer Architecture*, pages 25–36, 2000.

[15] P. Shirvani, N. Saxena, and E. McCluskey. Software-implemented edac protection against seus. *IEEE Transactions on Reliability*, 49(1):273–284, 2000.

[16] N. Oh, P. Shirvani, and E. McCluskey. Error detection by duplicated instructions in super-scalar processors. *IEEE Transactions on Reliability*, 51(1):63–75, 2002.

[17] Rajesh Venkatasubramanian, John P. Hayes, and Brian T. Murray. Low-cost on-line fault detection using control flow assertions. In *International On-Line Testing Symposium*, pages 137–143, 2003.

[18] Joakim Ohlsson and Marcus Rimén. Implicit signature checking. In *International Symposium on Fault Tolerant Computing*, pages 218–227, 1995.

[19] Guilin Chen, Mahmut T. Kandemir, and Mustafa Karaköy. Memory space conscious loop iteration duplication for reliable execution. In *Static Analysis Symposium*, pages 52–69, 2005.

[20] Sri Hari Krishna Narayanan, Seung Woo Son, Mahmut Kandemir, and Feihui Li. Using loop invariants to fight soft errors in data caches. In *Asia and South Pacific Design Automation Conference*, pages 1317–1320, Shanghai, China, January 18–21, 2005.

[21] Jie S. Hu, Feihui Li, Vijay Degalahal, Mahmut T. Kandemir, Narayanan Vijaykrishnan, and Mary Jane Irwin. Compiler-directed instruction duplication for soft error detection. In *Design, Automation and Test in Europe*, pages 1056–1057, 2005.

[22] George A. Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, and David I. August. SWIFT: Software implemented fault tolerance. In *International Symposium on Code Generation and Optimization*, pages 243–254, 2005.

[23] George A. Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, David I. August, and Shubhendu S. Mukherjee. Design and evaluation of hybrid fault-detection systems. In *International Symposium on Computer Architecture*, pages 148–159, 2005.

[24] Jun Yan and Wei Zhang. Compiler-guided register reliability improvement against soft errors. In *International Conference on Embedded Software*, pages 203–209, 2005.