Experimental Dependability Evaluation of a Fail-Bounded Jet Engine Control System for Unmanned Aerial Vehicles

Jonny Vinter[1], Olof Hannius[2], Torbjörn Norlander[2], Peter Folkesson[1], Johan Karlsson[1]

[1]*Department of Computer Engineering*
Chalmers University of Technology
S-412 96 Göteborg, Sweden
+46 31 7721667, +46 31 7723663 fax
{vinter, peterf, johan}@ce.chalmers.se

[2]*Volvo Aero Corporation*
Performance & Control Systems
S-461 81 Trollhättan, Sweden
+46 520 93846, +46 520 98573 fax
{Olof.Hannius, torbjorn.norlander}@volvo.com

Abstract

This paper presents an experimental evaluation of a prototype jet engine controller intended for Unmanned Aerial Vehicles (UAVs). The controller is implemented with commercial off-the-shelf (COTS) hardware based on the Motorola MPC565 microcontroller. We investigate the impact of single event upsets (SEUs) by injecting single bit-flip faults into main memory and CPU registers via the Nexus on-chip debug interface of the MPC565. To avoid the injection of non-effective faults, automated pre-injection analysis of the assembly code was utilized. Due to the inherent robustness of the software, most injected faults were still non-effective (69.4%) or caused bounded failures having only minor effect on the jet engine (7.0%), while 20.1% of the errors were detected by hardware exceptions and 1.9% were detected by executable assertions in the software. The remaining 1.6% is classified as critical failures. A majority of the critical failures were caused by erroneous booleans or type conversions involving booleans.

1. Introduction

Important development issues for future aircraft are to combine high safety requirements with low maintenance, development and production costs. This is especially true for applications such as Unmanned Aerial Vehicles (UAVs). The market for military UAVs is growing and civil and commercial UAV applications are also emerging [1]. Revolutionary systems and new technologies are needed to meet the demands of future aircraft, requiring increasingly advanced electronic equipment and software. One of the major challenges is to build cost-effective computer systems for execution of safety-critical functions. This challenge provides the impetus for two important development trends. One is the development of generic distributed safety-critical systems that can be used for a wide range of air vehicles and engines. The other is the use of *commercial off-the-shelf* (COTS) components.

In this paper, we evaluate the fault-tolerance properties of a prototype FADEC (Full Authority Digital Engine Control) controller based on a COTS microcontroller, the Motorola PowerPC MPC565. The controller is developed for the Volvo Aero RM12 turbofan engine. This engine is suitable for large UAVs comparable to the Boeing X45 variants B and C [2], which use a similar engine (F404-GE-102D).

The controller is implemented on a single computer node intended to be part of a distributed control system. The controller is designed to exhibit fail-bounded or fail-stop failures in the presence of internal errors. Executable assertions in the FADEC software and the hardware error detection mechanisms included in the microcontroller are used to enforce the failure mode assumptions.

Our aim is to investigate the validity of the failure mode assumptions with respect to single event upsets. We do so by injecting single bit-flip faults into CPU-registers and the main memory, while carefully monitoring the behavior of a very accurate simulation model of the jet-engine.

Particle radiation induced single event upsets have become an increasingly important source of failure in electronic systems as the feature sizes of VLSI circuits have decreased. Previously, such upsets mainly occurred in electronic equipment in space because of heavy-ions. The physical properties of new circuit technologies make circuits sensitive also to neutrons caused by cosmic radiation which are frequent at flight altitudes and also appear at ground level [3, 4]. Thus, SEUs are no longer negligible for critical equipment and must be carefully considered in flight applications.

The next section briefly describes the jet engine and the failure model of the FADEC controller. The experimental setup used for the dependability evaluation is described in Section 3, and the results of the evaluation are presented and discussed in Section 4. Finally, the conclusions are given in Section 5.

2. Jet engine control and failure model

2.1. Jet engine description

The RM12 engine is a turbofan engine with afterburner designed for vehicles traveling at supersonic speed.

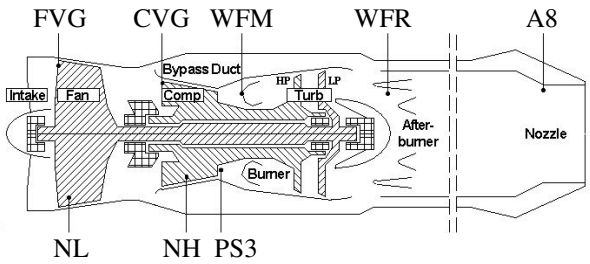

Figure 1. Schematic view of a turbofan engine with afterburner (RM12).

The engine is controlled by five actuators denoted, FVG, CVG, WFM, WFR and A8, see Figure 1. By positioning the variable stator guide vanes FVG (Fan Variable Geometry) and CVG (Compressor Variable Geometry), the RM12 fan and compressor are controlled to an optimal working point to achieve good engine performance. The fuel flow to the core engine is denoted WFM, and the afterburner fuel flow WFR. The rotational speed for the fan is denoted NL and for the compressor NH, where N stands for rotational speed while L and H identify the Low- and High-pressure parts respectively. The compressor outlet pressure is denoted PS3. The exhaust nozzle area, denoted A8, is controlled so that the pressure ratio over the complete engine is optimized to obtain maximum thrust.

2.2. Failure model

A fail-bounded failure model [5, 6] is assumed for the FADEC controller. This means that the system (controller and jet engine) is allowed to produce wrong outputs as long as the system stays within defined bounds. When an error is detected, e.g. when jet engine parameters exceed predefined or run-time calculated bounds, the FADEC controller decides if the error is manageable or if a backup system has to be switched in to prevent an accident. If the FADEC decides that a backup system must be activated (or if a hardware exception is triggered) it has to stop producing outputs. If not (or if the error is undetected) the system may deliver erroneous outputs as long as the system stays bounded. Thus, the engine control system is fail-bounded with a fail-safe mode. The bounds used in this study are presented in Section 3.3.

3. Experimental setup

3.1. The experimental platform

A dynamic model of the RM12 engine controlled by a model of a FADEC prototype developed for an UAV application study are used for the evaluation. Both the engine model and FADEC prototype have been developed in MATRIXx which is a graphical simulation and analysis tool that has the capability to auto generate code from dynamic simulation models. The generated code can then be compiled, linked and downloaded to a target system. The advantages of using a dynamic simulation model is that we can study how transient errors in the FADEC affect the RM12 engine operation, we can feed the result from experiments back to the MATRIXx environment, make improvements to error detection and error handling and verify that the improvements are effective.

A FADEC evaluation platform relying on COTS hardware has been developed (see Figure 2).

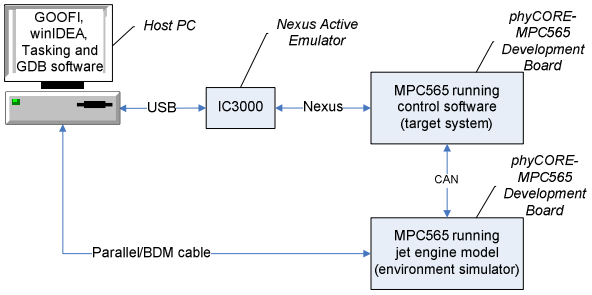

Figure 2. The experimental platform.

The hardware consists of two development boards featuring the Motorola PowerPC MPC565 microcontroller executing at a clock frequency of 40 MHz, equipped with 2 MB external SRAM, 1 MB external Flash memory and 1 MB on chip Flash memory. MPC565 is one of the first microcontrollers to implement the Nexus standard [7] enabling on-chip debugging with advanced features such as real-time trace of program and data flow. The platform relies on a commercial Nexus debug environment from iSYSTEM which takes advantage of the features defined by the Nexus standard (another Nexus-based fault injection environment targeting the MPC565 can be found in [8]). The debug environment consists of an iC3000 active emulator and the winIDEA debug software.

The GOOFI [9] fault injection tool has been extended to control the Nexus debug environment in order to simplify and automate the injection of faults into MPC565 [10]. The GOOFI and winIDEA software runs on a PC connected to the emulator via a USB connection. The iC3000 emulator communicates with the Nexus port on the MPC565 target

system. The target system executes the FADEC control software (272 kB) and communicates with the jet engine software (308 kB) on the second board via a CAN bus which closes the control loop. The software is compiled with the Tasking PowerPC C/C++ compiler tool suite, release 2.1 while the open source GDB software is used to debug and download the jet engine software via a BDM (Background Debug Mode) on-chip debug interface of the MPC565. For each experiment, the FADEC software is first downloaded to the external SRAM of the target system which then is restarted.

3.2. Experiment and software setup

The single bit-flip fault model is widely accepted as a reasonably accurate representation of SEUs [11]. Single bit-flip faults were injected in two separate campaigns targeting the MPC565 CPU registers included in the PowerPC User Instruction Set Architecture (UISA) [12] and the data segments (stack, data and read-only data) of the memory respectively. The code segments of the memory were not targeted as they are assumed to be located in ROM in the actual FADEC implementation. Both campaigns were carried out using a pre-injection analysis of the assembly code which enabled only effective faults to be injected. The technique optimizes the fault-space by utilizing assembly-level knowledge of the target system in order to place single bit-flips in registers and memory locations only immediately before these are read by the executed instructions. This way, we avoid injecting faults that are overwritten before they affect the program execution. Experimental results obtained by random sampling of the optimized fault-space and the complete (non-optimized) fault-space were compared for two different workloads running on the MPC565 microcontroller in [13]. The study showed that the pre-injection analysis yields an increase of one order of magnitude in the effectiveness of faults, a reduction of the fault-space of two orders of magnitude in the case of CPU-registers and four to five orders of magnitude in the case of memory locations, while preserving a similar estimation of the error detection coverage.

The FADEC software is divided into procedures which execute at different frequencies, determined by a static cyclic scheduler. The procedures with the highest frequency (200Hz) execute at each control loop and the procedures with the lowest frequency (10Hz) execute once in an interval of 20 control loops. Thus, 20 control loops (loop 21 to 40) are targeted as the temporal trigger for the experiments to ensure that all procedures are executed at least once. During this time interval (0.1 to 0.2 seconds), the thrust demand to the jet engine increases from 35% to 51% of the maximal thrust, see Figure 3.

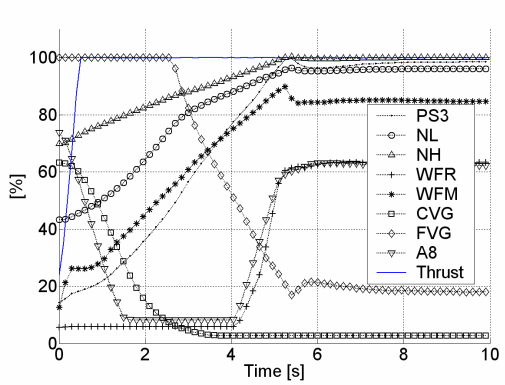

Figure 3. Thrust demand vs fault-free response.

The thrust demand and the observed engine parameters for a reference run of the nominal (fault-free) system are shown in Figure 3. The values of the engine parameters are used as reference and are compared to the engine parameters logged during fault injection experiments to identify any violations of the system failure bounds defined in Section 3.3. The thrust command in Figure 3 makes the core engine fuel flow WFM increase. When fuel flow increases, the high pressure and low pressure rotor speeds NH and NL and the compressor outlet pressure PS3 also increase. The variable guide vanes CVG and FVG opens (decrease) to maintain an adequate pressure ratio across the fan and compressor. The exhaust nozzle area A8 is initially open for minimum thrust and closes when the engine accelerates. Afterburner fuel flow is zero (minimum measured WFR is 6%) until it is engaged at 4 s. After 5 s, when both WFR and WFM are constant, the engine thrust is at maximum. Note that the exhaust nozzle area, A8, opens with the same rate as WFR increases to obtain the correct engine pressures.

3.3. Undetected bounded failures and mission or flight critical failures

We define a divergence from the reference run of at least 20% for one or more of the parameters shown in Figure 3 as a *mission-* or *flight critical failure*. Otherwise, the failure is considered as an *undetected bounded failure*. A mission critical failure may interrupt the mission since the UAV should return to the base for engine diagnosis. A flight critical failure may lead to a lost engine and a crash. The assumptions for our classification are based on previous experience of the engine.

4. Results

Table 1 summarizes the results of the fault injection experiments. Despite the use of the pre-injection analysis technique described in Section 3.2, a significant percentage of non-effective errors are produced. The main causes for

this are i) *booleans* which are *True* if the numerical value is non-zero will not change state due to most single bit-flips, ii) many software statements may mask errors (e.g., errors in the variable *a* of the statement [*if a* > 10 *then* …] will be masked if *a* is larger than 10 before the bit-flip and assumes an even higher value after the bit-flip), and iii) some variables in the FADEC prototype are periodically initialized to their default values and that errors may therefore be overwritten.

Table 1. Error and failure classification.

Target (# exp)	Non-effective errors	Detected by MPC565 hardware exceptions	Detected by FADEC executable assertions	Undetected bounded failures	Mission critical failures	Flight critical failures
Registers (2873)	61.9% (1778)	31.5% (904)	1.5% (44)	4.8% (137)	0.3% (9)	<0.1% (1)
Memory (2402)	78.6% (1888)	6.4% (154)	2.2% (54)	9.8% (235)	2.0% (47)	1.0% (24)
TOTAL (5275)	69.4% (3666)	20.1% (1058)	1.9% (98)	7.0% (372)	1.1% (56)	0.5% (25)

4.1. Errors detected by MPC565 exceptions

As shown in Table 1, 31.5% and 6.4% of all faults injected in registers and memory respectively are detected by MPC565 hardware exceptions [12]. When an exception is triggered, the experiment is stopped and a new experiment is started. In an actual system implementation, the FADEC node should stop producing results and leave the control to a backup when an exception is triggered. The processor was configured to enter the Checkstop State (CHSTP) instead of taking the Machine Check Exception (MCE) itself when MCE occurs. CHSTP occurred for 52.4% and 26.6% of the hardware exceptions observed for faults injected in registers and memory respectively while the corresponding figures for Floating-Point Assist Exceptions (FPASE) are 19.0% and 50.6%. Software Emulation Exceptions (SEE) occurred for 15.2% and 7.8% of the exceptions observed for faults injected in registers and memory respectively while External Breakpoint Exceptions (EBRK) occurred for 3.3% and 13.0%. For Alignment Exceptions (ALE), the corresponding figures are 9.5% and 1.9%. The remaining 13 exceptions of the MPC565 were seldom or never triggered.

4.2. Errors detected by executable assertions

When an error is detected by the FADEC control software, e.g. when engine parameters exceed fixed or dynamic bounds, the controller decides if the error is negligible or if it should resign control to a backup system to prevent a critical event. The FADEC prototype software provides several mechanisms for detecting errors and collecting status information. Detected errors considered as severe will trigger at least one of the 17 final executable assertions visualized in Figure 4, denoted EA1 to EA17 (see also Table 2). The FADEC node gives up control if one or several of them are executed (evaluated true). The assertion that has been activated most frequently is EA14 which means that an error affecting the Compressor Variable Geometry (CVG) functionality has been detected (28.9% for register faults and 26.5% for memory faults).

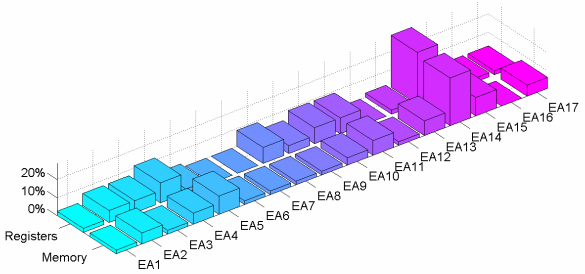

Figure 4. FADEC executable assertions triggered.

Table 2. Executable assertions.

Acronym	Description
EA1, EA2	Engine operating over limits
EA3 - EA5	Erroneous temperature input
EA6, EA7	Erroneous speed input
EA8	Erroneous pressure input
EA9, EA10	Erroneous position measurement
EA11 - EA15	Erroneous servo system
EA16, EA17	Erroneous discrete output

4.3. Critical failures due to faults in memory

Errors in the stack area, the data area and the read-only data (rodata) area of the SRAM memory were either i) detected by hardware exceptions or executable assertions (208 errors), ii) undetected bounded failures (235 errors) or iii) mission or flight critical failures (71 errors). 65.3% of the critical failures are due to faults injected into the data area and the remaining 34.7% of critical failures are due to faults injected into the read-only data area. No critical failures were observed for faults injected in the stack area.

A majority of the 71 critical failures observed for faults injected in memory could be sorted into three groups. A representative plot of the engine parameters observed for each group and their causes are presented in the following paragraphs.

Group 1 - Errors resulting in a lost afterburner. Over 59% (42 experiments) of the observed critical failures showed a behavior similar to that shown in Figure 5. For at least 35 of those, a unique boolean used for initialization of the software changed state and a re-initialization was performed. The errors in this group resulted in lost afterburner control, but the control of the core engine was normal. Afterburner failures are considered mission critical and not flight critical. As long as the exhaust nozzle area (A8) is correctly controlled (closed position for operation at maximum speed) without afterburner, thrust level is sufficiently high.

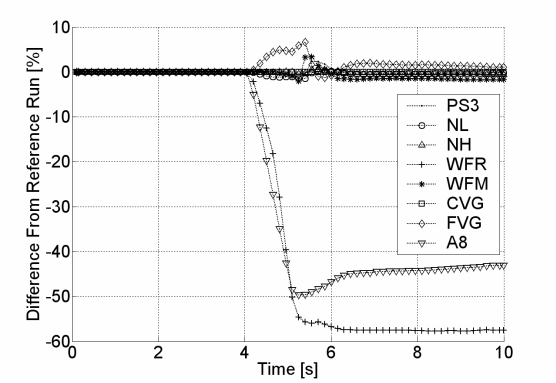

Figure 5. Error in a boolean resulting in a disabled afterburner.

Group 2 - Error in read-only data for type conversion. The source code for the FADEC model prototype is mainly generated from MATRIXx but an additional software module is also required. For each control loop, data are exchanged between the two software modules. Different data types are currently used in the FADEC model compared to the additional software module and during exchange of data between the modules, type conversions are performed. Type conversions involve using a read-only converter mask. When it assumes a faulty value due to a bit-flip, the mask will be permanently corrupted. Thus, this fault will affect all subsequent boolean-to-float conversions in the software. In 12 experiments (16.9% of all critical failures) the converter mask was the target for fault injection and since all subsequent conversions were affected, the state of the system experienced erroneous behavior almost identical to that shown in Figure 6.

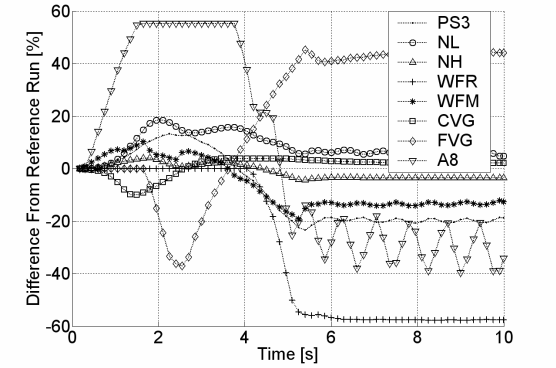

Figure 6. Error in constant used for type conversion resulting in faulty boolean states.

During the first 4 - 5 s, the exhaust nozzle area (A8) is too wide, the low pressure turbine speed (NL) too high and the FVG too closed. In addition to low thrust, the fan is outside the operating range with risk for stall or resonance. After 4 s, the afterburner fails to light up and the exhaust nozzle area (A8) starts to oscillate. All these factors may have serious impact on flight safety. Especially during critical flight phases such as take-off and landing. This failure was not observed for faults in registers since faults in the converter mask stored in a register will eventually be overwritten.

Group 3 - FADEC produced NaN (Not a Number) double precision floats. In eight experiments, the FADEC node produced erroneous control outputs which were (or resulted in) NaN floats. Since arithmetic with NaN floats will produce more NaN floats (e.g., $f1 = f2 + \text{NaN} \Rightarrow f1 = \text{NaN}$) the error may propagate quickly. Five out of eight NaN errors were detected by EAs while the remaining three resulted in flight critical failures. Figure 7 shows the engine behavior for one of those. Only the Compressor Variable Geometry (CVG) value stays within reasonable bounds for normal engine operation. The engine would probably flame out and important engine parameters such as fuel flow and engine speed quickly decrease to levels below ground idle, which is critical in all flight phases.

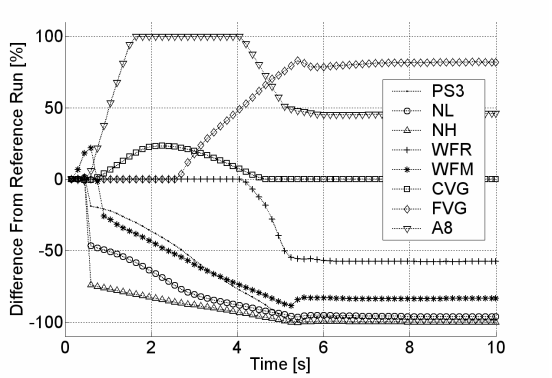

Figure 7. Behavior due to "Not a Number" floats.

Group 4 – Others. The system behavior for the remaining 14 (19.7%) critical failures can not be visualized by a representative plot. Seven experiments converged to nominal behavior within the observed time interval while seven did not. Eight of these failures are considered flight critical.

4.4. Critical failures due to faults in registers

Only 0.3% of the faults injected in registers resulted in critical failures (compared to 3.0% of the faults injected in memory). Nevertheless, nine mission critical and one flight critical failure were observed for the register faults. Due to a design flaw in the commercial debug environment used for fault injection, only the 32 least significant bits of each 64-bit floating-point register could be reached. This corresponds to the 32 least significant bits of the mantissa part of the float. Thus, faults in floating-point data only cause minor errors which very likely affected the number of critical failures observed. Different system behavior

could be observed for each failure and Figure 8 shows the most severe behavior, a transient engine failure.

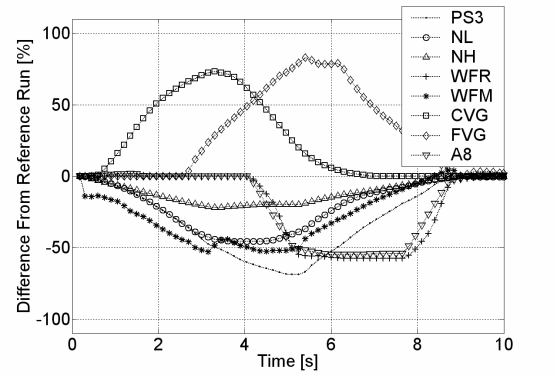

Figure 8. Register fault directly affecting the core engine fuel flow (WFM).

After the error has occured, the core engine fuel flow (WFM) is lower than normal, resulting in too low compressor outlet pressure (PS3), fan (NL) and compressor speed (NH). Accordingly, the guide vanes for the fan (FVG) and compressor (CVG) are more closed than in the reference run. After nine seconds, the engine has recovered and operates correctly. This engine failure is flight critical for a few seconds during take-off and landing since the engine response is abnormal.

5. Conclusions

We have experimentally evaluated a prototype FADEC jet engine controller executing on the COTS microcontroller Motorola MPC565 intended for UAV applications. Transient faults were injected into the FADEC prototype to investigate the efficiency of the nominal error detection mechanisms. The fault injection experiments were carried out using a pre-injection analysis of the assembly code to avoid the injection of non-effective faults. However, most of the faults were still non-effective (69.4%) or caused bounded failures having only minor effect on the UAV (7.0%) in our experiments which suggests that the FADEC node is to some extent inherently robust. The hardware exceptions of MPC565 detected 20.1% of the errors while the executable assertions in the FADEC software detected 1.9%.

Critical failures, which could potentially lead to the loss of the UAV, were observed for 1.6% of the experiments. A majority of the critical failures were caused by errors affecting boolean states in the software, either directly or indirectly through erroneous type conversions. Consequently, special care should be taken how to declare and use booleans in software.

Acknowledgements

This work was financed by NFFP (Swedish National Flight Research Program). We want to specially thank ALTIUM LIMITED for sponsoring us with the Tasking compiler used in this study.

References

[1] http://www.spacedaily.com/news/uav-04a.html, March 18th, 2005.

[2] http://www.air-attack.com/page.php?pid=10, March 18th, 2005.

[3] E. Normand, *Single event upset at ground level.* IEEE Transactions on Nuclear Science, 1996. **43**(6, pt.1): p. 2742-50.

[4] P.E. Dodd, M.R. Shaneyfelt, J.R. Schwank, and G.L. Hash. "Neutron-induced latchup in SRAMs at ground level", in *International Reliability Physics Symposium*. March 30 - April 4, 2003. Dallas, TX, USA.

[5] J.G. Silva, P. Prata, M. Rela, and H. Madeira. "Practical issues in the use of ABFT and a new failure model", in *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*. 1998.

[6] J.C. Cunha, R. Maia, M.Z. Rela, and J.G. Silva. "A study of failure models in feedback control systems", in *Proceedings International Conference on Dependable Systems and Networks.* Göteborg, Sweden. 2001.

[7] IEEE-ISTO, *The Nexus 5001 Forum™ Standard for a Global Embedded Processor Debug Interface.* 1999: p. 9-10.

[8] J.-C. Ruiz, P. Yuste, P. Gil, and L. Lemus. "On benchmarking the dependability of automotive engine control applications", in *Proceedings International Conference on Dependable Systems and Networks*. June 28 - July 1, 2004. Florence, Italy.

[9] J. Aidemark, J. Vinter, P. Folkesson, and J. Karlsson. "GOOFI: generic object-oriented fault injection tool", in *Proceedings International Conference on Dependable Systems and Networks*. Göteborg, Sweden, 2001.

[10] D. Skarin, J. Vinter, P. Folkesson, and J. Karlsson, *Implementation and usage of the GOOFI MPC565 Nexus fault injection plug-in*. Tech. Report No. 04-08, Dept. of Comp. Eng., Chalmers University of Technology, Göteborg, Sweden, 2004.

[11] G.C. Messenger, *Collection of charge on junction nodes from ion tracks.* IEEE Transactions on Nuclear Science, 1982. **ns-29**(6): p. 2024-31.

[12] Motorola, *MPC565/MPC566 User's Manual*. 2003.

[13] R. Barbosa, J. Vinter, P. Folkesson, and J. Karlsson. "Assembly-level pre-injection analysis for improving fault injection efficiency", in *Proc. Fifth European Dependable Computing Conference (EDCC-5)*. April 2005. Budapest, Hungary.