Impact of Soft Errors in a Jet Engine Controller

Olof Hannius[1] and Johan Karlsson[2]

[1] Volvo Aero Corporation,
S-46181 Trollhättan, Sweden
olof.hannius@volvo.com
[2] Department of Computer Science and Engineering,
Chalmers University of Technology,
S-412 96 Göteborg, Sweden
johan@chalmers.se

Abstract. We present an experimental study in which we investigate the impact of particle induced soft errors occurring in the microprocessor of an experimental FADEC system. The study focuses on the impact of single bit faults in the instruction set architecture (ISA) registers. For such faults, we investigate the effectiveness of the error detection mechanisms included in the FADEC system, and determine the consequences of errors that escape detection. To this end, we injected single bit faults in the ISA registers of a Freescale MC68340 microprocessor during execution of a prototype jet engine control program. Utilizing both random fault injection and partially exhaustive injections, we conducted six fault injection campaigns comprising in total more than 7000 injected faults. Twenty-three percent of the injected faults were effective, i.e., they affected the outputs of the control program. Of these, the system detected 91%. Of the 9 % that escaped detection, 7% caused a minor deviation in engine thrust that would be harmless to flight safety, while 2% caused severe or potentially catastrophic changes in engine thrust.

Keywords: jet-engine, controllers, FADEC, soft errors, cosmic neutrons, error detection, coverage, fault injection.

1 Introduction

Digital control systems for turbo-jet engines have been in operational use for almost 30 years. These systems are known as *Full Authority Digital Engine Control* systems, or *FADEC* systems. To ensure aircraft safety, FADEC systems must be highly reliable and fault-tolerant. A basic requirement is that a failure of a single hardware unit should never cause the engine to deliver inadequate thrust.

Most FADEC systems are provided with two redundant control channels configured as a primary/backup pair. Recently designed FADEC systems are typically equipped with two electronic channels, while older designs often use a single electronic channel with a hydro-mechanical backup. Regardless of whether the backup channel is electronic or hydro-mechanical, it is essential that the primary electronic channel is provided with highly efficient error detection mechanisms so that a fail-over to the backup channel is performed immediately if the primary channel should fail.

F. Ortmeier and P. Daniel (Eds.): SAFECOMP 2012, LNCS 7612, pp. 223–234, 2012.

One of the key challenges in designing a dual channel FADEC system is to provide the electronic channels with error detection mechanisms that can effectively detect hardware errors occurring in the microprocessor that executes the control software. These mechanisms must ensure that the FADEC does not exhibit critical failures. A critical failure occurs when the FADEC generates erroneous actuator commands that cause a significant change in the engine thrust.

There are two main design options available for detecting microprocessor faults in a FADEC control channel. One is to execute the control program on two lock-stepped microprocessors (or cores). This solution achieves very high detection coverage since the errors are detected by comparing the outputs of the two processors. The other option is to use a single microprocessor monitored by a watch-dog timer and various software implemented assertions and reasonable checks. The latter solution has been successfully used in several FADEC systems, including the one that controls the RM12 engine produced by Volvo Aero.

However, many existing FADEC systems were designed for microprocessors produced during the 1980's and 1990's. These microprocessors were manufactured in circuit technologies that are less sensitive to cosmic-ray induced soft errors and aging faults than current technologies are. It is expected that technology and voltage scaling will make future circuit technologies increasingly sensitive to these kinds of faults as well as process variations [1]. It is therefore an open question whether the classical design with a single microprocessor provides sufficient detection coverage for future FADEC systems.

This paper presents the results of a fault injection study aiming to provide insights into the error sensitivity of a single processor control channel with respect to microprocessor faults that manifest as transient bit errors in the instruction set architecture registers of the processor. Such errors can be caused by both transient and intermittent transistor level faults, including cosmic ray-induced soft errors [2] electromagnetic interference [3], intermittent faults caused by process variations [4], and aging effects such as NBTI [5], hot-carrier injection [6] and gate-oxide breakdown [7]. We conducted the fault injection experiments with an engineering prototype of a single processor control channel based on the Freescale MC68340 microprocessor. We injected single-bit faults in the instruction set architecture (ISA) registers of THE processor while it was executing a program controlling a software model of the Volvo Aero RM12 engine. To perform the experiments, we developed a fault injection tool called JETFI (JET Engine Fault Injection tool) [8].

The remainder of this report is organized as follows. Section 2 explains the basic operation of the RM12 jet engine including the main engine parameters, which we use to describe the failure modes of the engine. Section 2 also includes an overview of the main functions of the FADEC system. Section 3 describes the experimental system and the fault injection procedure. The results of our experiments are presented in Section 4. A summary is provided in Section 5 and Conclusions and Future Work are given in Section 6.

2 Jet Engine and Control System Description

The RM12 jet engine is a two-spool mixed flow turbofan engine shown in Fig. 1. Its principle operation is as follows. The intake delivers the air flow required by the engine. The fan and the low-pressure (LP) turbine are connected by the LP shaft and the compressor and the high-pressure (HP) turbine are connected by the HP shaft. (An engine with two shafts is a two-spool engine.) The compressor delivers compressed air to the burner. When the air-fuel mixture burns, it releases energy causing a high temperature gas flow that powers the high- and low-pressure turbines.

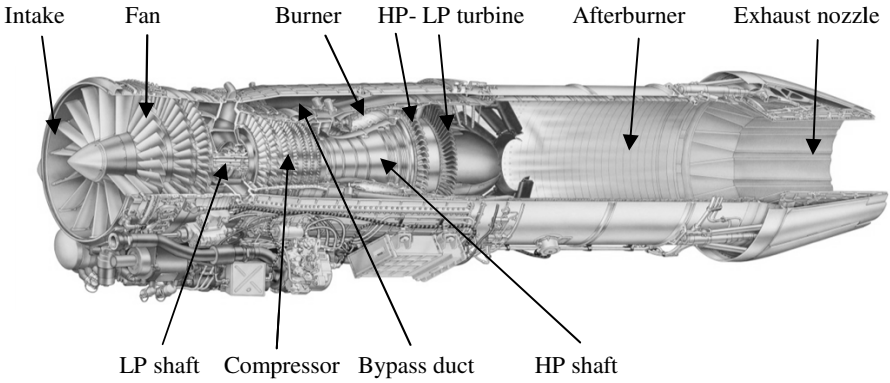

Fig. 1. Cross-sectional view of the two-spool mixed flow RM12 engine

The high-pressure turbine powers the compressor, while the low-pressure turbine powers the fan. When the hot gas flow has passed through the low-pressure turbine, it is further expanded and accelerated through the exhaust nozzle producing thrust. The FADEC system controls the thrust of the engine using five actuators. These control the guide vanes of the fan (FVG) and the compressor (CVG), the fuel mass flows to the burner (WFM) and the afterburner (WFR) and the area of the exhaust nozzle (A8). The pilot modulates thrust by changing the angle of a Power Lever (PLA). Besides from the demanded thrust, the control system needs six more inputs. These are the inlet temperature TT1, the LP shaft speed NL, the compressor inlet temperature TT25, the HP shaft speed NH, the compressor discharge pressure PS3 and the LP turbine exhaust gas temperature TT5. A comprehensive description of how to control the RM12 engine is found in [9].

3 Experimental System

This section describes the main elements of our experimental set-up. We provide an overview of the set-up in Section 3.1. The error detection mechanisms are described in Section 3.2, while the JETFI fault injection tool is described in Section 3.3.

3.1 System Overview

The experimental system consists of a host computer and two computer boards, called the FADEC board and the Engine board, as shown in Fig. 2. The computer boards are identical and use the Motorola 68340 processor. The FADEC board executes the control software while the Engine board executes a software model of the RM12 engine. The software for the two computer boards has been generated from models developed with MATRIXx v6.1 [10] and compiled with GNU ADA.

The host computer is used for controlling the fault injection experiments and for collection and analysis of experimental data. The RS232 serial link between the computer boards and the host computer are used for program download and control of the fault injection campaigns. Actuator commands and sensor data are exchanged via a RS232 link between the FADEC computer board and the Engine board. Due to the limited processing power of the 68340 processors, the set-up executes the control program approximately 1000 times slower than a real system.

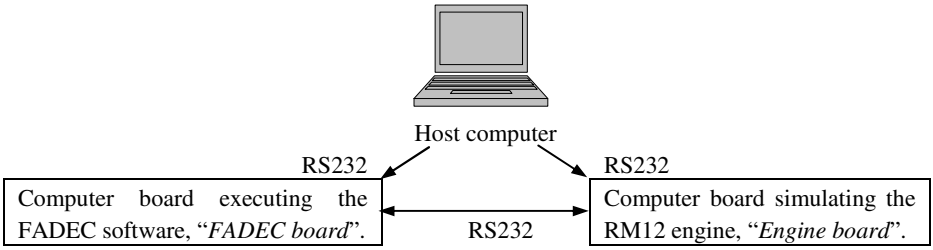

Fig. 2. Host computer and target system overview

The FADEC control software executes in a cyclic control loop with prescheduled control tasks. It consists of 29 subsystems. A subsystem is a set of control tasks with the same execution rate. The execution rate varies from 200 Hz for Subsystem 1 down to 1 Hz for Subsystem 29. Subsystem 1 performs demanding control activities such as positioning of the guide vanes, fuel flow metering and setting the exhaust nozzle area. The other subsystems perform a variety of other control tasks and trim functions.

The Engine board executes simulation models of the engine, sensors, actuators and the hydro-mechanical control system. We use a linearized model of the RM12 engine to minimize execution time. We believe the accuracy of this model is sufficient for the purpose of our experiments. The execution times would have become much longer if we would have used a more accurate non-linear engine model.

The Engine board emulates a use case where the Power Lever Angle (PLA) increases from 55º to 75º during one second of real-time execution. (Flight idle is at 28º and max dry thrust, i.e., without afterburner, is at 100º.)

3.2 Error Detection Mechanisms

The error detection mechanisms, EDMs, implemented in the experimental FADEC system include a watchdog monitor (WDM), hardware and software exceptions and software assertions shown in Table 1.

Table 1. Error Detection Mechanisms in the FADEC

EDM	Description
WDM	A timer which must be reset periodically to prevent it from tripping, i.e. signaling that an error has occurred
Hardware exceptions	Hardware EDMs supported by the Motorola 68340 processor.
Software exceptions	Software checks generated automatically by MATRIXx or by the programmer using the exception-clause in the ADA-language. They detect erroneous execution, erroneous calculations and other errors.
Software assertions	Range checks on engine parameters.

Watch Dog Monitor

The watchdog monitor is implemented in the host computer of the JETFI tool and detects if the FADEC computer board stops to produce output data for duration longer than 10 seconds. In our experimental setup, we consider a WDM-trip as a detected error.

Hardware Exceptions

The Motorola 68340 processor supports 256 hardware exception vectors numbered in the range 0 to 255. The exceptions that were triggered in the fault injection experiments are Bus error, Address error, Illegal instruction, Line 1111 Emulator and Format error, see Table 2.

Table 2. Hardware Exceptions

Hardware exception	No	Description
Bus error	2	Occurs when the processor attempts to use information from an aborted bus cycle (illegal memory access).
Address error	3	Occurs if a misaligned memory access is attempted. For instance a word transfer to an odd address.
Illegal instruction	4	Occurs if the processor attempts to execute an unimplemented instruction.
Line 1111 Emulator	11	A special case of illegal instruction. The name originates from the contents of the most significant bits for unimplemented instructions.
Format error	14	This check ensures that the program does not make erroneous assumptions about information in the stack frame.

Software Exceptions

A software exception is a general check concerning calculations and program execution. The FADEC software implemented software exceptions are shown in Table 3.

Software Assertions

The software assertions perform range checks on engine parameters and are based on physical limitations of the jet engine and its environment. The software assertions shown in Table 4 can detect engine failures, errors in data from sensors and wrap-around signals from actuators (torque motor currents).

Table 3. Software exceptions

Software exception	Description
EXEC_ERROR	Raised by execution checks generated by the MATRIXx-tool.
MATH_ERROR	Raised when the predefined ADA exception NUMERIC_ERROR or CONSTRAINT_ERROR is raised. This happens if a numeric operation is undefined or when a variable is erroneously assigned.
TIME_ OVERFLOW	Two types of scheduler errors can cause time overflow. 1) If the scheduler is interrupted while executing the non-interruptible critical section and 2) If a subsystem is ready to run but has still not finished running. Both are due to inconsistency in the scheduler.
STOP_BLOCK	This refers to a Stop Simulation Block.
UCB_ERROR	Error in a User Code Block.
UNKNOWN_ ERROR	An error that is not recognized by the code generated by MATRIXx. A possible cause is an incorrect user-written error condition.
OTHERS	Raised if an unexpected exception occurs, i.e. it is not identified as any of the other defined exceptions.

Table 4. Software assertions in the FADEC

S/W assertion	Failure condition	Possible cause	Effect when not detected
TT1 out of range	The reading from the TT1 sensor is not within range.	Sensor or input data failure.	Low engine thrust and even fan surge[1].
NH over-speed (HP shaft)	The measured speed of the compressor and high-pressure turbine is too high.	Overspeed of the HP shaft, failure in the input data.	There is a risk for engine disintegration.
NL sensor loss	Missing pulses in the pulse train from the NL sensor.	NL sensor failure detected by h/w.	Fan overspeed. Possible engine damage.
A8 or WFM LVDT/TM failure (actuators)	The relationship between demanded current and the position change of the actuator does not match.	Sensor, actuator or mechanical failure of the actuation hardware.	A missed detection will result in a low or high engine thrust.
PS3 fails high	Out of range failure of the comp. discharge pressure.	Sensor failure.	Incorrect fuel flow and erroneous thrust.
Flame out	Engine speed and turbine exhaust temp. decrease below allowed limits.	Erroneous fuel metering.	The engine may flame out, if this occurs.

3.3 Fault Injection Tool

The JETFI tool [8] can inject single and multiple bit-flip faults in the ISA registers of the CPU running the FADEC control program. In the experiments presented in this paper we injected one single bit-flip in each experiment. A fault is defined by an injection time and a bit in an ISA register. The injection time is defined by the execution of a target machine instruction and an invocation counter. The injection time and the targeted bit can be selected randomly by the tool or deterministically by the tool user.

[1] Fan surge causes an abrupt reversal of the airflow through the engine.

A fault injection experiment begins by replacing the target machine instruction in the program memory of the FADEC board with a trap instruction. This is done by a piece of code executed on the FADEC board. The host computer then orders the Engine board to start the RM12 simulator and the FADEC board to start the FADEC control program.

Each time the FADEC control program executes the trap instruction, the corresponding trap handling routine notes the number of times it has been called. If this number is lower than the value of the invocation counter, the trap handler executes the original machine instruction without any modifications and then directs the execution back to the control program. When the trap instruction has been executed the same number of times as stated by the invocation counter value, the trap handling routine injects the fault by inverting the value of the target bit, replaces the trap instruction with original machine instructions, and finally directs the execution back to that instruction. The JETFI tool then monitors the behavior of the continued simulation and automatically starts a new experiment as soon as the outcome from the previous experiment has been recorded.

4 Results

This section presents the results of our fault injection experiments. Section 6.1 describes how we classify the outcomes of the experiments. Section 6.2 describes the results from five fault injection campaigns denoted A to F.

4.1 Classification of Experiment Outcomes

The outcome from an experiment is divided in five categories, Detected error, No effect, Non-critical failure, Critical failure and Failed experiment. An explanation of the categories is found in Table 6.

Table 5. Outcome classification

Category	Description
Detected error	An error detected by the watchdog monitor (WDM), a hardware or software exception or a software assertion.
No effect	The outcome No effect occurs when nothing can be observed that is different from a fault free experiment. The injected error is either overwritten or remains in the system but does not have any impact on the outputs of the system (dormant error).
Non-critical failure	A negligible deviation in the control system outputs caused by an undetected error.
Critical failure	A significant change in engine thrust caused by an undetected error.
Failed experiment	A Failed experiment occurs when the fault injection routine uses a non-valid fault time. It can happen if the address for the injected fault is never executed by the software or if it is executed a fewer number of times than specified as condition for the fault injection routine.

In our experiments, undetected errors are identified by an automatic check in the JETFI-tool. The automatic check compares the control system outputs with reference data from an error-free (golden) experiment. We have defined an output signal that deviates more than 5% from the correct value as a critical failure.

4.2 Description of Experiments and Presentation of Results

Campaign A is used as a reference for comparison with the other campaigns. Each of the campaigns B to F has separate focus to investigate different aspects of fault injection and error detection.

Campaign A – Random Fault Selection
In Campaign A, we used random fault selection among instructions in all subsystems. The result from 991 experiments is shown in Table 6. The last row of Table 6 shows the relative frequency of each outcome with a 95% confidence interval bound. Of all experiments, 715 were non-effective and 276 were effective.

Table 6. Results of Campaign A (faults selected randomly).

						Undetected	
	No effect	**Watch-dog**	**Hardware Exception**	**Software Exception**	**Software Assertion**	**Non-crit. failure**	**Critical failure**
No. of faults	715	32	200	12	13	15	4
Rel. freq. (%)	72.2±2.8	3.2±1.1	20.2±2.5	1.2±0.7	1.3±0.7	1.5±0.8	0.4±0.4

The number of non-effective faults relative to the total number of injected faults is quite normal compared to other studies [11-13]. The distribution of experimental outcomes for the effective faults is also typical with hardware exception as the primary error detection mechanism.

Campaign B – Scheduler Fault Injection
In Campaign B, we injected faults in the control task scheduler. The purpose of this campaign was to investigate the sensitivity to faults in this part of the code. We selected nine instructions in the initial part of the scheduler. For each of these, we exhaustively injected faults in the bits 0- 15 in the D0- D7 and A0- A7 CPU registers. The scheduler reads input data and calls the subsystems and output routine periodically. The fault injection was directed to the input reading part of the scheduler. The result of Campaign B is shown in Table 8. Most faults have no effect at all. Non-effective faults were 86.1%. The corresponding number for Campaign A is 72.2%. This part of the code showed to be less sensitive to faults than other parts.

Table 7. Results of Campaign B (target instructions in the initial part of scheduler)

					Undetected	
No effect	**Watch-dog**	**Hardware Excep.**	**Software Excep.**	**Software Assertion**	**Non-crit. failure**	**Critical failure**
1983 (86.1%)	2 (0.9%)	267 (11.6%)	0 (0%)	20 (0.9%)	10 (0.4%)	2 (0.1%)

Campaign C – Control Subsystem Fault Injection
In this campaign we injected faults in Subsystem 1, containing fuel metering control software and executing with the highest frequency. We selected seven target instructions and exhaustively injected faults in bits 0-31 in the D0-D7 and A0-A7 registers. A total number of 3584 experiments were performed. The result is shown in Table 8. Compared to Campaign A, the number of non effective faults is lower and the number of undetected errors is higher.

Table 8. Results of Campaign C (target instructions in Subsystem 1)

					Undetected	
No effect	**Watch-dog**	**Hardware Excep.**	**Software Excep.**	**Software Assertion**	**Non-crit. failure**	**Critical failure**
2465 (68.8%)	97 (2.7%)	739 (20.6%)	29 (0.8%)	103 (2.9%)	86 (2.4%)	65 (1.8%)

Campaign D –Partially Exhaustive Fault Injection
The objective of Campaign D was to investigate a fault selection technique that covers a selected fault space with a minimum of experiments. We apply a manual pre-injection analysis to avoid injecting faults that have known effect. Only those registers that may change the behavior of the system compared to previous experiments are selected. At the first address in the fault space, all data and address registers are injected with faults so that the outcome is known for any register bit flip. From that instruction and forward, only the registers that are used are injected with faults, since they are the only that can change the outcome from what is already known.

The method was applied to a sequential piece of code in subsystem 1 consisting of 28 addresses. With pre-injection analysis, the number of experiments was reduced to 1664, instead of 14336 without pre-injection analysis. Table 9 shows the outcome.

Table 9. Result from Campaign D

					Undetected	
No effect	**Watch-dog**	**Hardware Excep.**	**Software Excep.**	**Software Assertion**	**Non-crit. failure**	**Critical failure**
11278 (78.7%)	308 (2.2%)	1640 (11.4%)	0 (0.0%)	780 (5.4%)	328 (2.3%)	2 (0.01%)

Campaign E – Faults in the Program Counter and Status Register
Campaign E was performed to investigate the effect of faults in the Program Counter (PC) and Status Register (SR). These registers have not been selected for fault injection in the campaigns B to D. For fault injection in the Program Counter, we used a subset of the address space used in Campaign B. The faults were injected in the 16 lowest bits with a total of 64 experiments. The outcome in Table 10 shows that the distribution of detected and undetected errors differs a lot from the previous fault injection campaigns. The number of non-effective errors is considerably lower. The number of errors detected by Watchdog monitor and Hardware Exceptions are high.

Table 10. Results from Fault injections in the PC register (Campaign E)

					Undetected	
No effect	**Watch-dog**	**Hardware Excep.**	**Software Excep.**	**Software Assertion**	**Non-crit. failure**	**Critical failure**
7 (11.0%)	17 (26.6%)	32 (50.0%)	0 (0%)	2 (3.1%)	5 (7.8%)	1 (1.5%)

It showed to be hard finding an address where fault injection in the Status Register was effective. Most instructions change the contents of the Status Register but only a few use the contents, for example branch instructions. The probability is therefore high that a fault in the SR is overwritten. One example of an outcome from one effective experiment was a software exception. No further experiments were performed.

Campaign F – Fault Injection Time
In fault injection campaigns A- E we injected faults at, or close to, the 90th loop count. In this campaign we set the fault injection time to 0.450, 1.015, 1.505 and 2.035 seconds corresponding to the 90th, 203rd, 301st and 407th loop count. The total simulation time was the same as the other experiments (3.0 seconds/600 loop counts). The goal for this setup was to find a fault location for which the outcome changed due to injection time. We were especially interested to find out if the same fault location could produce undetected and detected errors depending on injection time. A change of outcome was observed in one experiment. At 0.450 seconds, the outcome was an undetected error, but at the other time instances, the outcome was a software assertion detection. The campaign was terminated when it was confirmed that the change of fault injection time can change the outcome.

5 Summary

The effects from soft errors in a prototype FADEC controller have been evaluated by injecting single bit-flip faults in the controller's microprocessor while simulating a jet-engine during an acceleration sequence. Of all experiments, 67% were non-effective. The distribution of the remaining 23% of effective errors is shown in Table 11.

Table 11. Distribution of effective errors

						Undetected	
Campaign	**No. of eff. exp.**	**Watch-dog**	**Hardware Excep.**	**Software Excep.**	**Software Assertion**	**Non-crit. failure**	**Critical failure**
A (Random)	276	11.6%	72.5%	4.3%	4.7%	5.4%	1.4%
B (Scheduler)	321	6.9%	83.2%	0%	6.2%	3.1%	0.6%
C (Subsys 1)	1119	8.7%	66.0%	2.6%	9.2%	7.7%	5.8%
D (Subsys 1)	3058	10.1%	53.6%	0%	25.5%	10.7%	0.1%
E (PC reg.)	57	29.8%	56.1%	0%	3.5%	8.8%	1.8%
Average	**-**	**13.4%**	**66.3%**	**1.4%**	**9.8%**	**7.2%**	**1.9%**

The efficiency of the error detection mechanisms are (in descending order):

1) Hardware Exception
2) Watchdog Monitor and Software Assertion
3) Software Exception

Hardware Exception is the most efficient mechanism (66.3%). Watchdog Monitor (13.4%) and Software Assertions (9.1%) have roughly the same efficiency. Of the undetected events (9.1%), Non-critical failures are dominating. It is worth to note that the ratio of errors and failures differ much between campaigns.

6 Conclusions and Future Work

The results of our fault injection experiments provide valuable insights into the relative effectiveness of the error detection mechanisms included in our FADEC prototype. They show that the hardware exceptions included in the MC68340 processor obtained the highest error coverage, in average 66.3%. This result is consistent with results obtained in several other fault injection studies. The results also show that the watchdog timer and the software assertions were quite effective obtaining average coverage values of 13.4% and 9.8%, while the software exceptions detected merely 1.4% of the effective errors in average. Another important observation is that most of the undetected failures were non-critical. However, the percentage of critical failures, which varied between 0 and 6%, was higher than desirable. In particular, the high percentage of critical failure observed for errors injected into Subsystem 1 in Campaign C, suggest that the code for that subsystem needs to be provided with additional error detection mechanisms.

Our future work will focus on development and evaluation of software-implemented error detection techniques that can complement the ones we have evaluated in this paper. Techniques that we plan to investigate include selective time-redundant execution of sensitive code portions, software implemented control flow checking and new types of software assertions. Our aim is to reduce the likelihood of critical failure to below 0.01%. To this end, we plan to extend the JETFI tool to support test port-based fault injection and pre-injection analysis. We also plan to port our experimental setup to a new hardware platform with faster CPUs, so that we can run larger fault injection campaigns and thereby increase the confidence in our experimental results. In addition, we also intend to evaluate our FADEC prototype with respect to multiple bit errors.

Acknowledgements. This work has partially been supported by the research project Reliable Jet Engines funded by the NFFP programme.

References

1. Chandra, V., Aitken, R.: Impact of Technology and Voltage Scaling on the Soft Error Susceptibility in Nanoscale CMOS. In: IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems (DFTVS 2008), pp. 114–122 (October 2008)
2. Ibe, E., Taniguchi, H., Yahagi, Y., Shimbo, K.S., Toba, T.: Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule. IEEE Transactions on Electron Devices 57(7), 1527–1538 (2010)

3. Benso, A., Di Carlo, S., Di Natale, G., Prinetto, P.: A watchdog processor to detect data and control flow errors. In: 9th IEEE International On-Line Testing Symposium, pp. 144–148 (July 2003)
4. Jahinuzzaman, S.M., Sharifkhani, M., Sachdev, M.: Investigation of process impact on soft error susceptibility of nanometric SRAMs using a compact critical charge model. In: 9th International Symposium of Quality Electronic Design (2008)
5. Islam, A.E., Kufluoglu, H., Varghese, D., Mahapatra, S., Alam, M.A.: Recent issues in negative-bias temperature instability: Initial degradation, field dependence of interface trap generation, hole trapping effects and relaxation. IEEE Trans. Electron Devices 54(9), 2143–2154 (2007)
6. Kufluoglu, H., Alam, M.A.: A Computational Model of NBTI and Hot Carrier Injection Time-Exponents for MOSFET Reliability. Journal of Computational Electronics 3(3-4), 165–169 (2004)
7. Cannon, E.H., KleinOsowski, A.J., Kanj, R., Reinhardt, D.D., Joshi, R.V.: The impact of aging effects and manufacturing variation on SRAM soft-error rate. IEEE Transactions on Device and Materials Reliability 8(1), 145–152 (2008)
8. Hannius, O., Karlsson, J.: JETFI – A Fault Injection Tool for Assessment of Error Handling Mechanisms in Jet-engine Control Systems.Technical Report 2012:06, Chalmers University of Technology (2012) ISSN 1652-926X
9. Härefors, M.: A study in jet engine control - control structure selection and multivariable design. Ph.D. Thesis, Chalmers University of Technology, Sweden (1999)
10. Ward, D.K., Andrews, S.F., McComas, D.C., O'Donnell, J.R.: Use of the MATRIXx integrated toolkit on the Microwave Anisotropy Probe Attitude Control System. NASA's Goddard Space Flight Center, `http://lambda.gsfc.nasa.gov/product/map/team_pubs/aas99.pdf`
11. Autran, J.L., Roche, P., Sauze, S., Gasiot, G., Munteanu, D., Loaiza, P., Zampaolo, M., Borel, J.: Real-Time Neutron and Alpha Soft-Error Rate Testing of CMOS 130nm SRAM: Altitude versus Underground Measurements. In: Proc. International Conference On IC Design and Technology (ICICDT), Grenoble, pp. 233–236 (2008)
12. Autran, J.L., Roche, P., Sauze, S., Gasiot, G., Munteanu, D., Loaiza, P., Zampaolo, M., Borel, J.: Altitude and Underground Real-Time SER Characterization of CMOS 65 nm SRAM. IEEE Transactions on Nuclear Science 56(4) (August 2009)
13. Normand, E.: Single Event Upset at Ground Level. IEEE Transactions on Nuclear Science 43(6) (December 1996)
14. Normand, E.: Single Event Effects in Avionics. IEEE Transactions on Nuclear Science 43(2) (April 1996)