Experimental Evaluation of Three Concurrent Error Detection Mechanisms

Alireza Vahdatpour, Mahdi Fazeli, Seyed Ghassem Miremadi

Dependable Systems Laboratory
Computer Engineering Department
Sharif University of Technology
Tehran, Iran
Email: {vahdatpour, m_fazeli}@ce.sharif.edu, miremadi@sharif.edu

***Abstract*—This paper presents an experimental evaluation of the effectiveness of three hardware-based control flow checking mechanisms, using software-implemented fault injection (SWIFI) method. The fault detection technique uses reconfigurable of the shelf FPGAs to concurrently check the execution flow of the target program. The technique assigns signatures to the target program in the compile time and verifies the signatures using a FPGA as a watchdog processor to detect possible violation caused by the transient faults. A total of 3000 faults were injected in the experimental embedded system, which is based on an 8051 microcontroller, to measure the error detection coverage. The experimental results show that these mechanisms detect about 90% of transient errors, injected by software implemented method.**

***Index Terms* — Control Flow Checking, Experimental Evaluation, FPGA, Fault Injection**

I. Introduction

Nowadays, the widely use of processor based embedded and ubiquitous systems in high critical and real-time applications requires more investigations on reliability and fault-tolerance as essential attributes for these systems. Fault detection and coverage is the first step to design a fault-tolerant computer system.

Transient faults are the major causes for computer system failures. It is reported in [1] and [2], that more than 70% of transient faults lead to control flow errors (CFE). Furthermore, faults in hardware components such as the program counter, the address circuitry, and the memory elements or the software bugs such as compiler and operating system bugs may result in control flow errors [19].

As the coverage of internal error detection techniques in COTS processors is relatively low [3, 4, 5], the use of additional error detection techniques seems to be necessary in order to make these systems reliable. To address this issue, several behavior-based error detection techniques, especially control flow checking (CFC) mechanisms [6, 7, 10, 11, 12, 13, 14, 15, 16, 17, and 19], have been proposed.

One important step in the design of fault-tolerant systems is their evaluation. Basically evaluation techniques are divided in two groups; Analytical methods, and experimental methods. Analytical techniques use mathematical models such as Markov chains [20] [21], fault trees [22], and Petri nets [23] to model the real system. Experimental evaluation techniques are usually done using fault injections. The injection of faults to the systems can be done in three ways:

- Simulation-based fault injection
- Physical fault injection
- Software-based fault injection

Simulation-based fault injection has been used in [24], [25], and [26]. The main advantages of this method are its low cost and the ability to evaluate the system before real implementation.

In the physical fault injection techniques [9, 27, 28], the target system is experimentally implemented and evaluated. The higher speed of the fault injection and precious results are the main advantage of this evaluation method in comparison to the simulation based injection methods.

In software implemented fault injection techniques (SWIFI), instead of using hardware fault injector devices, injecting faults into the experimental system is up to the software [19, 21]. In addition to physical fault injection advantages, this technique has usually low costs and simpler implementations.

In this paper, an experimental evaluation of three hardware-based CFC mechanisms is presented. The detection mechanisms use a reconfigurable hardware to design and develop a watchdog processor. In the compile time, the detection mechanisms insert some assertions in the main program as the signatures and produce a rather small synthesizable hardware description code. Then the hardware description is implemented on a FPGA chip to build the watchdog processor. Being Independent from processor architecture and using of the shelf reconfigurable components are the main advantages of the proposed technique in comparison to the previous ones.

A total of 3000 SWIFI faults were injected to the processor. The fault injection results are investigated and compared to determine the efficiency of the used fault detection mechanisms.

The structure of this paper is as follows: Following the introduction, the error model is presented in Section 2. The proposed control flow checking mechanisms will be presented in details in the third section. The experimental evaluation system is introduced in Section 4. In Section 5 the experimental evaluation result is provided. Finally Section seven concludes the paper and presents future works.

II. THE ERROR MODEL

A fault in a digital computer system can manifest itself in three different locations:

- System Memory
- System Buses
- CPU internals

The errors which may occur due to these faults can be modeled in three categories:

- CPU Crashes
- Data errors
- Control Flow Errors (CFE)

CPU crashes happen when the processor does not work as a Finite Sate Machine anymore, meaning that it goes off in an undetermined state. CPU crashes can be detected by a simple external watchdog timer; hence we do not consider these types of errors in the CFCSP mechanisms. Data errors can also be detected by some mechanisms like assertions, thus are not considered hereby. The CFCSP mechanisms are solely centered on detecting control flow errors.

This paper focuses on the transient effects called SEUs (Single Event Upsets). Several reports have mentioned that the SEU is important not only for the circuits operating in the space, but also for the digital equipments operating at the ground level [17]. It is reported in [18] that the majority (>60%) of control flow errors differ from the correct ones in only a single bit (i.e. SEU) of an address. SEUs can also occur in memory cells' contents (registers, internal memory, etc). However, memories are usually protected against SEUs by means of error detecting/correcting codes (Hamming code, CRC code, Reed-Solomon code, etc) [17]. Therefore, internal registers are of much importance. For example several reports have mentioned that SEUs in the Program Counter (PC) register are a major source of CFEs in comparison to other internal registers [18]. Thus in this paper we focus on a class of errors dubbed Program Sequence Change. These types of CFEs can only occur in CPU internals and in Address buses. In this class, basically the next instruction address changes in a way that the next instruction is not the right one in the correct program order. For instance a modification of the Program Counter (PC) register can result in such an error.

III. THE PROPOSED CONTROL FLOW CHECKING MECHANISMS

In order to more clarify the presentation, the following definition is made:

Definition 1: A Basic Block (BB) is a sequence of non-branching instructions (except in the last instruction or last consecutive instructions) or branch destinations (except in the first instruction) in which the execution always enters at the first instruction and leaves via the one of last branch instructions.

In our CFC mechanisms (CFCSP), we have used control flow graph (CFG) as the main criteria. CFG is a simple directed graph, which its vertices represent basic blocks and its edges show the relation (jumps) from one BB to another BB. (Fig. 1)

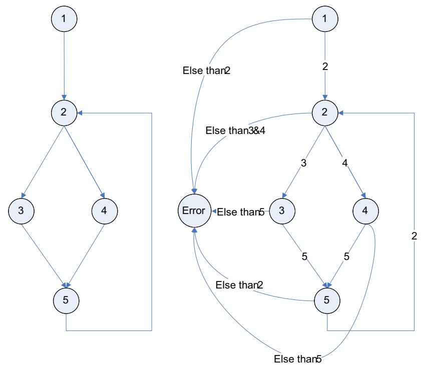

Figure 1. Program CFG and its related FSM

The main idea of shadow processing is in making a FSM from the program's CFG. The watchdog system will use this FSM as correctness criterion and concurrently checks the execution flow of the target processor. This requires the insertion of some signatures to each BB in the compile time. Beside the main mechanism which is based on mentioned FSM, three auxiliary mechanisms were used. Two block checking algorithms and a work load timer. The block checking algorithms use signatures in the main program to check if each BB is executed completely. The Work Load Timer is a simple hardware timer which is used to detect CPU crash errors. At the beginning of program execution the work load timer is initiated with a maximum allowed time in which the processor must send an alive signal.

As it was said in the previous section, it is required to make some assertions into the microcontroller program and also, build a HDL code for the reconfigurable watchdog system. In order to accomplish these tasks, tool chain software is developed. This program processes the primitive microcontroller's assembly program and gives two output files; final microcontroller's assembly program and watchdog FPGA HDL code. (Fig. 2).

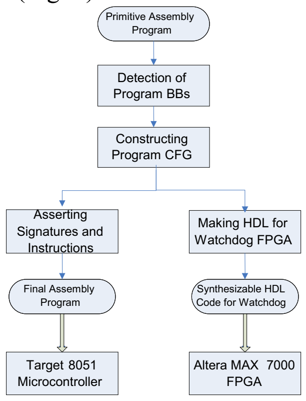
Primitive Assembly Program
Detection of Program BBs
Constructing Program CFG
Asserting Signatures and Instructions
Making HDL for Watchdog FPGA
Final Assembly Program
Synthesizable HDL Code for Watchdog
Target 8051 Microcontroller
Altera MAX 7000 FPGA

Figure 2. Preprocessing Steps

Detailed information on the error detection mechanisms can be found in [8].

IV. THE EXPERIMENTAL EVALUATION SYSTEM

The experimental system which is used to evaluate the mentioned mechanisms is shown in figure 3. It consists of

Top

four main components:

1. A target 8051 microcontroller
2. A Watchdog system
3. A fault injector and result logging unit
4. A Host computer

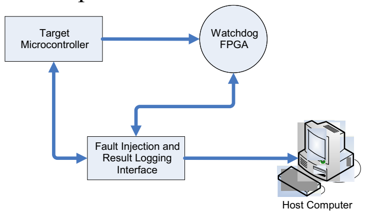

Figure 3. Experimental system diagram

A. *Target Microcontroller*

All proposed mechanisms are independent from processors architectures. Therefore, it is preferable to choose a widely used commercial processor, which can simply be implemented and tested. To meet these characteristics, an 8051 is used. 8051 has been used widely in industrial and commercial systems.

B. *Watchdog System*

The Watchdog system consists of a FPGA, which has connected pins to microcontroller output ports. The choice of FPGA should meet the minimum requirements of proposed mechanisms. In our evaluation sample system, we used an Altera MAX 7000S which has approximately 5000 gates, that is beyond the needs of our system.

C. *Fault Injection and Result Logging Interface*

In order to automatically inject faults and log the results, a fault injector and result logger is designed. The task of this component can be divided to two sections.

I. *Fault Injection*

Fault injection into the microcontroller must meet the conditions of considered fault model, SEU. Hardware faults that occur in CPUs, buses, and registers usually lead to software errors. Therefore, it is possible to mimic these software errors by the use of software implemented fault injection techniques (SWIFI). The Interface microcontroller interrupts target processor by setting hardware interrupt pin of the target microcontroller. The interrupted target processor leaves program execution, which is a standard workload program, saves its current program counter (PC) and starts a predefined interrupt routine. In this routine, a bit of saved program counter will be inverted. In each turn of fault injection the location of victim bit is changed in order to normally distribute the SEU on the PC. After that interrupt execution completed, the processor will load its PC to resume its main program, but the changed PC will lead to an unexpected jump in the runtime execution (CFE).

In order to make the injections time distributed and random, a series of random numbers has been generated by a PC and used as injection times.

At the time of fault injection, also watchdog FPGA and the timer should be restarted. Furthermore, the interface microcontroller should keep the start time of injection in order to log the detection latency of the proposed mechanisms.

II. *Result Logging*

After the fault was injected, the interface microcontroller waits to get either error detection signal or program completion signature. The first condition happens when the watchdog FPGA has been able to detect the CFE. After getting error signal, interface will get detected error type from the FPGA. Detection latency and the type of detected error also would be sent to the host computer for more offline analysis.

In the case of getting program completion signatures, a result checking program starts to see if the results of program execution (memory saved results) are as they were expected. If the results of the executed program are unexpected values, it means that a CFE has occurred and the detection system could not detect it. But there is also the rare possibility that the changed PC in interrupt routine does not lead to CFE and hence no error will be detected.

D. *Host Computer*

The task of this computer is to collect data from its RS232 serial port, which is connected to the interface system, and keep them in the database.

V. Experimental Results

In this section the CFCSP evaluation results which are extracted by physical fault injection are presented. Three workload programs, all written in Assembly language, were used in the experiment: a linked list, a matrix multiplication and a bubble sort. A reasonableness checking function is added to all of the workload programs to verify the correctness of the results. A total of 3000 faults in SWFI method were injected into the evaluation system while running each workload. The error detection coverage of CFCSP mechanisms is then extracted for this fault injection method.

As mentioned in previous section, since in SWIFI method the content of program counter register is directly changed and a CFE error is produced, it is actually an error injection method, not a fault injection method. The experimental results for each work load are shown in table 1. As it is shown in this table the average of 10% of injected errors are not detected by none of the error detection mechanisms.

TABLE 1. Detection coverage according to each mechanism

Detection Type	Execution Flow	Enter -Exit	Block Complete Execution	Time Out	Undetected	Total
Bubble	57%	10%	5%	11%	16%	84%
Matrix Multiply	69%	10%	5%	13%	3%	97%
Link List	58%	5%	15%	11%	11%	89%
Total Average	61.3%	8.3%	8.3%	11.6%	10%	90%

TABLE 2. The CFCSP control flow error detection coverage beside performance and memory overhead

Workload	Memory Overhead	Performance Overhead	Error detection Latency
Bubble Sort	104%	82%	2.9 ms
Link List	60%	77%	1.7 ms
Matrix Multiply	56%	42%	1.1 ms
Total Average	73%	67%	1.9 ms

Top

A brief comparison between the proposed mechanisms and the previous hardware based methods is shown in table 3. Since most of the previous mechanisms are based on the monitoring the processor buses or the exact execution of the program, they are not useable in COTS processors. On the other hand, mechanisms which can be used in COTS processor are mainly based on the debugging features of a specific processor family and are not applicable to any kinds of processors or platforms. A key advantage of The CFCSP technique is that it is a processor independent technique and also can be used in COTS processor. In comparison with the other techniques, it provides high error detection coverage as well as having acceptable overheads.

TABLE 3. COMPARISON OF THE CFCSP MECHANISMS WITH SOME OF THE PREVIOUS, HARDWARE-BASED CFC MECHANISMS

CF Mechanism	Coverage	Memory Overhead	Hardware Complexity	Performance Penalty	Detection Latency	Usable in COTS	Processor Independent
SIS	82%	6%~15%	Low	<10%	3.8ms	No	Yes
PSA	99.5%~99%	18%~27%	Low	-	7%~17%	No	Yes
TSM	93%	10%~16%	Low	<10%	3~6 Inst.	No	Yes
TTA	98%	24%~27%	Medium	17%~18%	11~18 Inst.	No	Yes
CIC	90%~98%	5%~28%	Medium	210%~245%	>80 cyc.	Yes	No
CFCSP	84%~97%	56%~104%	Low	42%~82%	1.1~2.9 ms	Yes	Yes

VI. CONCLUSION AND FUTURE WORK

Control flow checking mechanisms provide a viable solution to the modern embedded systems reliability requirements. The evaluation of three hardware based CFC mechanism was represented in this paper. Total of 3000 software implemented fault injection were made in order to evaluate the performance of the proposed mechanisms. The experimental results showed that these mechanisms detect about 90% of transient errors in the average case. Also, in order to more evaluate these mechanisms, power supply disturbance (PSD) technique can be used as fault injection method.

REFERENCES

[1] Pataricza A., I. Majzik, W. Hohl, J. Hoenig, “Watchdog Processors in Parallel Systems”, Proc. of the 19th Symposium on Microprocessing and Microprogramming (EUROMICRO'93), Spain, 1993, p.p. 69-74.

[2] Venkatasubramanian et al., “Low-Cost On-Line Fault Detection Using Control Flow Assertions”, Proc. of the 9th IEEE International On-Line Testing Symposium (IOLTS’03), 2003

[3] Chevochot P. and I. Puaut, “Experimental Evaluation of the Fail-Silent Behavior of a Distributed Teal-Time Run-Time Support Built from COTS Components”, Proc. of the IEEE/IFIP Int’l Conference on Dependable Systems and Networks (DSN’01), July 2001, p.p. 304 -313.

[4] Avizienis A., “A fault tolerance infrastructure for high-performance COTS-based computing in dependable space systems”, Proc. of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC’04), March 2004, p.p. 336.

[5] Madeira H., R. R. Some, F. Moreira, D. Costa and D. Rennels, “Experimental Evaluation of a COTS System for Space Applications”, Proc. of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’02), June 2002. p.p. 325–330.

[6] Mahmood A. and E.J. McCluskey, “Concurrent Error Detection Using Watchdog Processors – A Survey”, IEEE Trans. on Computers, Feb. 1988, p.p. 160 -174.

[7] Miremadi G. and J. Torin, “Evaluation Processor-Behavior Three Error-Detection Mechanisms Using Physical Fault-Injection”, IEEE Trans. on Reliability, Vol. 44, No. 3, Sept. 1995, p.p. 441-453.

[8] Vahdatpour et al., “Transient Error Detection in Embedded Systems Using reconfigurable Components”, Technical report, Sharif University of Technology, May 2006.

[9] Karlsson J., “Reliability Evaluation of a Fault-Tolerant Computer for a Multi-phased Mission and a Use of Heavy-ion Radiation for Fault Injection Experiments”, PhD Thesis, School of Electrical and Computer Engineering, Chalmers University of Technology, 1990.

[10] Oh N., P. P. Shirvani, and E. J. McCluskey, “Control-Flow Checking by Software Signatures”, IEEE Trans.On Reliability, Vol. 51, No. 2, March 2002.

[11] Kanawati G. A., V. S. S. Nair, N. Krishnamurthy, and J. A. Abraham, “Evaluation of Integrated System-Level Checks for On-Line Error Detection”, Proc. of IEEE Intemational Computer Performance and Dependability Symposium, 1996, p.p. 292-301.

[12] Venkatasubramanian R., J. P. Hayes and B. T. Murray, “Low-Cost On-Line Fault Detection Using Control Flow Assertions”, Proc. of the 9th IEEE International On-Line Testing Symposium (IOLTS’03), July 2003, p.p. 137–143.

[13] Goloubeva O., M. Rebaudengo, M. Sonza Reorda and M. Violante, “Soft-Error Detection Using Control Flow Assertions”, 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'03), Boston, Massachusetts Nov. 2003, p.p. 57-62.

[14] Mahmood A. and E.J. McCluskey, “Concurrent Error Detection Using Watchdog Processors – A Survey”, IEEE Trans. on Computers, Feb. 1988, pp. 160 -174.

[15] Raul Barbosa, Jonny Vinter, Peter Folkesson, and Johan Karlsson, “ Assembly-Level Pre-injection Analysis for Improving Fault Injection Efficiency”

[16] J. Arlat, Y.Crouzet, J. Carlson, P. Folkeson, E.Futchs and H. Lenbers, “Comparison of Physical and Software-Implemented Fault Injection Techniques”, IEEE Trans. on Computers, VOL. 52, NO. 9, Sep. 2003.

[17] Nicolescu B., R. Velazco1, M. Sonza-Reorda 2, M. Rebaudengo2, M. Violante,“A Software Fault Tolerance Method for Safety-Critical Systems:Effectiveness and Drawbacks”, Proceedings of the 15th Symposium on Integrated Circuits and Systems Design (SBCCI’02).2002, p.p. 101-106.

[18] Rimén M., J. Ohlsson and J. Karlsson, “Experimental Evaluation of Control Flow Errors”, Proc. 1995 Pacific Rim International Symposium on Fault Tolerant Systems (PRFTS), IEEE Computer Society Press, CA, USA, December 1995.p.p. 238 – 243.

[19] M. Fazeli, R. Farivar, S. G. Miremadi, "A Software-Based Concurrent Error Detection Technique for PowerPC Processor-based Embedded systems", 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT), California, 2005.

[20] Johnson B. W., “Design and Analysis of Fault-Tolerant Digital Systems”, Addison-Wesley, 1989.

[21] Pradhan D. K., “Fault-Tolerant Computer System Design”, Prentice-Hall, ISBN:0-13-057887-8, 1996.

[22] Ejlali A. and S. G. Miremadi, “Time-to-Failure Tree”, Proc. of the 49th Annual Reliability and Maintainability Symposium (RAMS’03), Florida, USA, pp. 148 -152, January 2003.

[23] Yin L., M. A. J. Smith, K. S. Trivedi, “Uncertainty Analysis in Reliability Modeling”, Proc. of the 2001 Annual Reliability and Maintainability Symposium, IEEE Press, pp. 229-234.

[24] Zarandi H. R., G. Miremadi and A. R. Ejlali, “SILVER: A Simulation-Based Fault Injection Tool at Switch Level Using Verilog”, Proc. of the SCIS & ISIS 2002 Conference, Tsukuba, Japan, 2002.

[25] Jenn E., J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson, “Fault Injection into VHDL Models: The MEFISTO Tool”, Proc. of the 24th International Symposium on Fault-Tolerant Computing (FTCS-24), 1994, pp. 336-344.

[26] Kanawati G. A., N. A. Kanawati, and J. Abraham, “EMAX: An Automatic Etractor of High-Level Error Models”, Proc. of the American Institute of Aeronautics and Astronautics, 1993, pp. 1297-1306.

[27] Folkesson P., “Assessment and Comparison of Physical Fault Injection Techniques”, Thesis for the degree of PhD, Department of Computer Engineering, CHALMERS University of technology, Sweden 1999.

[28] Steininger A., C. Scherrer, “On Finding an Optimal Combination of Error Detection Mechanisms Based on Results of Fault Injection Experiments”, Proc. of the 27th International Symposium on Fault-Tolerant Computing (FTCS-27), USA, pp. 238-247, June 1997