AQTR 2004 (THETA 14)
2004 IEEE-TTTC - International Conference on Automation, Quality and Testing, Robotics
May 13 – 15, 2004, Cluj-Napoca, Romania

ENHANCED COMMITTED INSTRUCTIONS COUNTING (ECIC):

A Scheme for Error Detection Enhancement in COTS Processors

Amir Rajabzadeh[*], **Seyed Ghassem Miremadi**[**]

Sharif University of Technology
Azadi Ave., Tehran, Iran,
Tel:+98 21 616 4640, Fax: +98 21 601 9246
rajabzad@ce.sharif.edu[*]*, miremadi@sharif.edu*[**]

Abstract: Increasing use of commercial off-the-shelf (COTS) processors in industrial, embedded, and real-time systems necessitates the development of error detection mechanisms for such systems. This paper presents an error detection scheme called Enhanced Committed Instructions Counting (ECIC) to increase error detection in COTS processors without any external hardware. The scheme uses an internal Performance Monitoring feature which provides the ability to count the number of committed instructions in a program. The scheme is experimentally evaluated on a 32-bit Pentium® processor using software implemented fault injection (SWIFI). The results show that the error detection coverage varies between 90.52% and 98.18%, for different workloads.

Key words: error detection enhancement, COTS processors, control flow checking, software implemented fault injection.

1. INTRODUCTION

The increasing popularity of low-cost safety-critical computer-based applications, has caused the use of Commercial Off-The-Shelf (COTS) processors in these applications to become particularly attractive. COTS processors are widely used in industrial [1], embedded [2], real-time [2], [3] and space [4], [5] applications. Internal error detection mechanisms available in the COTS processors have limited error detection coverage and poor error containment provisions [3], [4], [6]. For example, Pentium® processors employ different levels of parity checking with 53% error detection coverage [7]. This is an unacceptable coverage, which necessitates the use of additional error detection techniques to enhance the error detection coverage. To enhance the error detection coverage, behavior-based error detection techniques, specially control flow checking (CFC) methods [8], are becoming an attractive solution when cost is a major concern. Some CFC methods are pure software (SW-CFC) without any extra hardware and some of them (HW-CFC) use an external watchdog processor (WDP). Traditional HW-CFC methods, such as TTA [9], TSM [10], may not be applied to modern processors with on-chip caches and instruction pipelines [11]. To eliminate these limitations, we have developed an error detection scheme, called Committed Instruction Counting (CIC) [11]. The CIC scheme can be applied to most modern COTS processors with internal cache and pipelines, such as Pentium processor. In this paper, we present an enhanced version of the CIC scheme, called Enhanced Committed Instructions Counting (ECIC). Low performance overhead and elimination of the WDP were major reasons for the design of the ECIC scheme. The ECIC scheme is a SW-CFC method. The SW-CFC methods are usually weak in detection of illegal jumps to out of the program areas and illegal infinite loops, such as CFCSS [12] and BSSC [13], because these methods check the correctness of the operation only at specific points of the program. However, the ECIC scheme has the ability to detect illegal jumps to out of the program areas and illegal infinite loops. The ECIC scheme uses the Performance Monitoring features of COTS processors. The scheme has been experimentally evaluated on a Intel Pentium® processor. The results show that the error detection coverage varies between 90.52% and 98.18%, for different workloads.

The next section discusses Performance Monitoring features. In section 3, the design of the ECIC scheme is presented. The experimental system and results are given in section 4. Finally, section 5 concludes the paper discussion.

2. PERFORMANCE MONITORING FEATURES

Many of the current superscalar processors such as Pentium [7], AMD x86-64 [14], Alpha [15], MIPS R10000 [16] and PowerPC-604 [17] include features called Performance Monitoring. The Performance Monitoring features use special internal counters, which can be configured to count the occurrences of several events in the processors. Examples of such events are cache hits, instructions committed and branches taken [7]. Each Performance Monitoring counter (PM-counter) can be set to an arbitrary value at any time. After that, the content of the PM-counter will be increased by one for each occurred event. Some processors also have special pins, called event-ticking pins [7], which can signal out the occurrences of internal events of the processors. In our study, we have used a Pentium® processor whose Performance Monitoring features have two PM-counters, CTR0 and CTR1, and two event-ticking pins, PM0 and PM1. The CTR0 and CTR1 overflow can be reported on PM0 and PM1 pin [7].

3. THE ECIC SCHEME

In this section, we present the ECIC scheme. The ECIC scheme like CIC scheme [11] assumes that the program is partitioned into Branch Free Blocks (BFBs); a sequence of non-branching instructions, and Partition Blocks (PBs); a set of instructions between two physically consequent BFBs [11]. In this model, we distinguish between seven types of control flow errors (CFEs) [9], [11], which are shown in Figure 1.

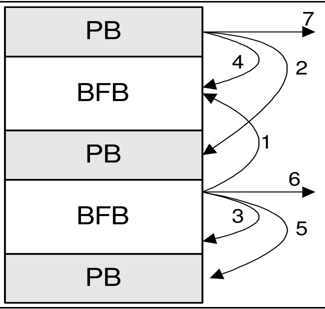

Figure 1 – Program partitions and types of CFEs

3.1. ECIC error detection mechanisms

The main concept of the ECIC scheme is based on the traditional signature monitoring, i.e., 1) program partitioning into BFBs and PBs, 2) offline signature generation for BFBs and PBs, and 3) online signature checking for BFBs and PBs. The ECIC scheme consists of four mechanisms to detect seven types of CFEs (Figure 1):

Enhanced BFB Instruction Counting (EBIC): The EBIC mechanism derives a signature (e.g. m*i*) for each BFB*i* at the compile time, which is the number of instructions executed in the BFB*i*. At the beginning of the BFB*i* the PM-counter will be set to 2^n-1- m*i*. *n* is the number of PM-counter bits and 2^n-1 is the largest number that PM-counter can hold. During the execution of the BFB*i,* for each instruction executed, the PM-counter will be increased by one. The PM-counter must exactly reach 2^n*-1* after the last instruction executed in the BFB*i*. Any inconsistency will be detected in one of the following two ways: 1) At the end of the BFB*i*, the PM-counter will be read. If the PM-counter value is less than the expected value (i.e. 2^n*-1*) a CFE is detected. 2) If the PM-counter overflows before reaching the end of BFB*i*, an event-ticking pin (i.e. PM0 or PM1) signals the occurrence a CFE. The EBIC mechanism detects all CFEs of types 3 and 6, as well as some CFEs of types 1 and 5.

Enhanced PB Instruction Counting (EPIC): The EPIC mechanism tries to detect CFEs within PBs. The mechanism checks the maximum number of instructions that are allowed to be executed continually outside the BFBs, say *p*. *p* is the pre-calculated signature, which is unique for all PBs and may be obtained from the source code. However, it may be easer to obtain *p* with experimental observation. At the beginning of a PB*i* (end of a BFB*i*), a PM-counter is set to 2^n*-1- p*. During the execution of the program outside the BFBs, the PM-counter is incremented for each instruction executed. In normal operation, the PM-counter is less than 2^n*-1* during the execution of PB instructions. If PM-counter overflows, an error will be signaled on an event-ticking pin. The EPIC mechanism detects all CFEs of type 7. CFEs of Types 2 and 4 can also be detected if the number of instructions executed violates the maximum value (i.e. *p*).

Index: At compile time, this mechanism assigns an arbitrary unique index to each BFB. At run time, the index of a BFB, called BFB-Index, is stored in a global variable at the beginning of the BFB. At the end of the BFB, the variable will be compared to the BFB-Index. If these two indices are different then a CFE will be reported. The Index mechanism detects all CFEs of type 1.

Phase: This mechanism checks the correct order of entering and exiting the BFBs [9]. During error-free operation, an entry to a BFB should always be followed by an exit from that BFB. Hence, a fault, which causes the execution to erroneously pass through two subsequent entry points or two subsequent exit points, can easily be detected. The Phase mechanism is implemented in the same manner as the Index mechanism. The Phase mechanism detects all CFEs of types 4 and 5.

3.2. ECIC implementation

The assembly codes of workload programs can be used to add the redundant instructions needed to implement the ECIC scheme. To do this, a program, called a *postprocessor* was developed which accepts an assembly program and generates a version of that assembly program protected with the ECIC scheme, see Figure 2. The structure of a program after inserting the redundant instructions is shown in Figure 3.

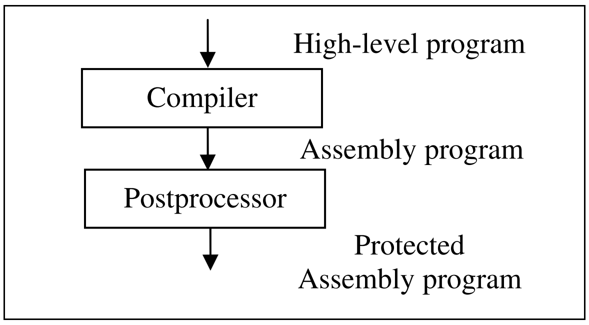

Figure 2 – Generation of a program protected with the ECIC scheme

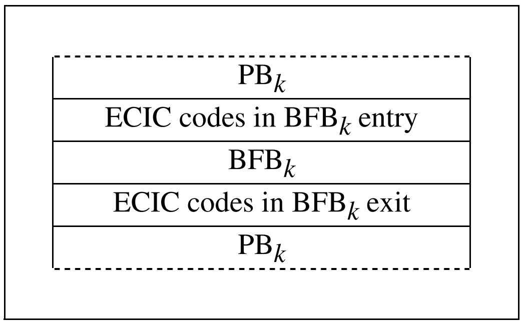

Figure 3 – Protected program with the ECIC scheme

To reduce overheads, it is recommended that BFBs with few lines of instructions (e.g. less than four) merge to the adjacent PBs and will be considered as one larger PB. Two programs, a linked list (List) and an inverse matrix (Matrix), have been used as workloads. Using the postprocessor, the extra codes needed to implement the ECIC scheme were added to the workloads. The extra instructions inserted in the workload programs incur program size and performance overheads as are summarized in Table 1. As shown in Table 1, the overhead of the program size and number of instruction executed in the ECIC and CIC schemes are equal, because the number of extra instructions in the both schemes are equal. The percentages of performance reduction in the ECIC scheme for Matrix and List programs are 42% and 67%, respectively. However, these values were 210% and 245% in the CIC scheme because of I/O operation. The ECIC scheme eliminates WDP and its I/O instructions and therefore improves performance.

Table 1. Overheads

	Matrix	List
Number of instructions executed in the original program	11,551,140	4,170,314
Number of instructions executed in the protected program (ECIC and CIC)	17,610,375	7,646,631
Overhead of the number of instructions executed (ECIC and CIC)	52%	86%
Overhead of the program size (ECIC and CIC)	10%	5%
Overhead of the execution time (ECIC)	42%	67%
Overhead of the execution time (CIC)	210%	245%

4. Experimental evaluation

In this section, we present the organization of the experiment system and experimental results.

4.2. Experimental system

The organization of the experimental system is shown in Figure 4. The system consists of three parts: a Pentium board, an FPGA board and a host computer.

The Pentium board: The board has been equipped with a 100 MHz Intel Pentium® processor. Two important programs are executed on the Pentium® processor under Linux OS; the workload program (i.e. Matrix and List) and fault injector routine. The routine generates CFEs as follows: 1) the fault activator logic activates the NMI pin of the Pentium® processor, 2) the NMI service routine reads the return address from the stack, changes a bit in the least significant bits (bits 0~7) of the return address and then writes it back to the stack.

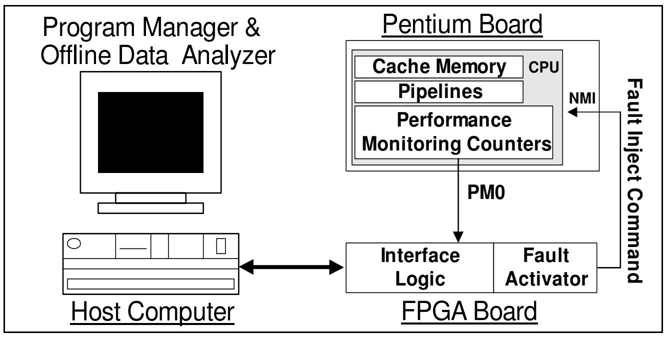

Figure 4 – The experimental system

After returning from the NMI service routine, the execution continues at an unexpected address due to the change of the value of the return address.

The FPGA board: The fault activator logic and the interface logic have been integrated on this board. The FPGA board has been equipped with an Altera Flex10k30 FPGA.

The Host computer: The Host computer contains a Manager Program to manage and control the whole experiment and an offline data analyzer which analyzes the raw data and extracts the results.

4.2. Experimental results

This section presents the experimental results of the error detection coverage. The results are based on about 6000 faults. The coverage results are presented in Table 2.

Table 2. Fault detection coverage

		Coverage (%)				
Workloads	Error classes	EBIC	EPIC	Index	Phase	**Total**
Matrix	CFEs resulted in program-crashes (67.50%)	52.32 (35.32)	47.34 (31.95)	0 (0)	0 (0)	99.66 (67.27)
	Other CFEs (32.50%)	18.57 (6.04)	10.42 (3.39)	0.7 (0.23)	75.82 (24.64)	95.11 (30.91)
	All CFEs (100%)	**41.36**	**35.34**	**0.23**	**24.64**	**98.18**
List	CFEs resulted in program-crashes (70.50%)	47.08 (33.19)	52.36 (36.91)	0 (0)	0 (0)	99.44 (70.10)
	Other CFEs (29.50%)	27.59 (8.14)	1.28 (0.38)	4.85 (1.43)	38.01 (11.21)	69.22 (20.42)
	All CFEs (100%)	**41.33**	**37.29**	**1.43**	**11.21**	**90.52**
ECIC Total	CFEs resulted in program-crashes (69%)	49.70 (34.26)	49.85 (34.43)	0 (0)	0 (0)	99.55 (68.69)
	Other CFEs (31%)	23.08 (7.09)	5.85 (3.77)	2.78 (0.83)	56.92 (17.93)	82.79 (25.67)
	All CFEs (100%)	**41.35**	**38.20**	**0.83**	**17.93**	**94.36**
CIC Total	CFEs resulted in program-crashes (69%)					100 (69.00)
	Other CFEs (31%)					86.79 (25.67)
	All CFEs (100%)					**94.67**

Note that an error can be detected with more than one mechanism. To determine the effectiveness of the ECIC mechanisms, the CFEs are divided into two classes: 1) CFEs resulting in program-crashes, and 2) other CFEs. As shown in Table 2, the percentages of all errors resulting in program crashes were 67.50% and 70.50% for Matrix and List programs, respectively. These values depend on several parameters; the average instruction length, the location of fault injection in the program, etc. The Index and Phase mechanisms are unable to detect any program crash at all, because they need to reach the end of a BFB in the program to detect an error. Although, the ECIC scheme was capable of detecting most program crashes (99.55%), however, about 0.45% of the program crashes were left undetected. In these cases the PM-counter has stopped. This may have been caused by misconfiguration of MSR's configuration register caused by an illegal jump to an incorrect place in the sequence of instructions configuring this register. The CIC scheme is capable of detecting all program crashes (100%), because the CIC scheme uses an external workload timer (WL-Timer [11]) which can detect all crashes. The WL-Timer causes that error detection coverage of the CIC scheme (94.67%) to become 0.31% more than the coverage of the ECIC scheme (94.36%).

5. Summery and Conclusions

This paper presents a scheme called Enhanced Committed Instructions Counting (ECIC) to enhance error detection coverage in COTS processors without any external hardware. The scheme uses the Performance Monitoring features of the processors. The scheme is a cost-effective to enhance error detection in COTS processors with internal Performance Monitoring features. The scheme has the ability to detect illegal jumps to out of the program areas and illegal infinite loops. The results show that the error detection coverage varies between 90.52% and 98.18%, for different workloads.

6. REFERENCES

1. Croll P. and P. Nixon, (1991), “Developing Safety-Critical Software within a CASE Environment”, *IEE Colloquium on Computer Aided Software Engineering Tools for Real-Time Control*, p.p. 8.
2. Gill C. D., R. K. Cytron and D. C. Schmidt, (2003), “Multiparadigm Scheduling for Distributed Real-Time Embedded Computing”, *Proceedings of the IEEE* , Vol. 91, Issue 1, p.p. 183 -197.
3. Chevochot P. and I. Puaut, (2001), “Experimental Evaluation of the Fail-Silent Behavior of a Distributed Real-Time Run-Time Support Built from COTS Components”, *IEEE/IFIP International Conference on Dependable Systems and Network*, p.p. 304 -313.
4. Madeira H., R. R. Some, F. Moreira, D. Costa and D. Rennels, (2002), “Experimental Evaluation of a COTS System for Space Applications”, *IEEE/IFIP International Conference on Dependable Systems and Networks*.
5. Some R. R. and D. C. Ngo, (1999), “REE: A COTS-Based Fault Tolerant Parallel Processing Supercomputer for Spacecraft Onboard Scientific Data Analysis”, *Proc. of the Digital Avionics System Conference*, Vol. 2, p.p. B3-1-7 -B3-1-12,
6. Avizienis A., (2000), “A Fault Tolerance Infrastructure for Dependable Computing with High-Performance COTS Components”, *Proceedings International Conference on Dependable Systems and Networks*, p.p. 492 -500.
7. Intel Corp., (1997), *Pentium ® Processor Family Developer’s Manual*.
8. Mahmood A. and E.J. McCluskey, (1988), “Concurrent Error Detection Using Watchdog Processors - A Survey”, *IEEE Transactions on Computers*, p.p. 160 -174.
9. Miremadi G., J. Ohlsson, M. Rimen, and J. Karlsson, (1998), “Use of Time, Location and Instruction Signatures for Control Flow Checking”, *Proc. of the DCCA-6 International Conference, IEEE Computer Society Press*.
10. Madeira H., M. Rela, P. Furtado and J. G. Silva, (1992), “Time Behaviour Monitoring as an Error Detection Mechanism”, *3rd IFIP Working Conference on Dependable Computing for Critical Applications (DCCA-3)*, p.p. 121-132.
11. Rajabzadeh A.; Mohandespour M.; Miremadi Gh., (2004), ”Error Detection Enhancement in COTS Superscalar Processors with Event Monitoring Features”, *10th International Pacific Rim Dependable Computing Symposium (PRDC10)*, p.p. 49-54.
12. Oh N., P. P. Shirvani, and E. J. McCluskey, (2002),”Control-Flow Checking by Software Signatures”, *IEEE Trans. In Reliability*, Vol. 51, No 2. p.p. 111-122.
13. Miremadi G., J. Karlsson, U. Gunneflo, and J. Torin, (1992), “Two Software Techniques for On-Line Error Detection”, *22nd Annual International Symposium on Fault-Tolerant Computing (FTCS-22)*, p.p. 328-335.
14. Advanced Micro Devices, Inc**.**, (2002), *AMD x86-64 Architecture Programmer’s Manual,* Volume 2: System Programming.
15. Compaq Computer Corp., (1998), *Alpha Architecture Handbook*.
16. MIPS Technologies Inc., (1996), *MIPS R10000 Microprocessor User’s Manual*.
17. Motorola Inc., (1994), *PowerPC ™ 604 RISC Microprocessor Technical Summary*.