SIGNATURE VERIFICATION: A NEW CONCEPT FOR BUILDING SIMPLE AND EFFECTIVE WATCHDOG PROCESSORS

Henrique Madeira, José Camões, and João Gabriel Silva
Deptartment of Electrical Engineering
University of Coimbra
3000 Coimbra - Portugal

ABSTRACT

This paper presents the architecture of the Checker, which is a watchdog processor able to detect transient, intermittent, and permanent errors in multiple processor systems using on-line signature analysis. The signatures are stored in the local memory of the Checker and verified according to a new approach called on-line signature verification. In this approach, the storage requirements for control flow information of the application programs are substancially reduced and the design of the watchdog processor is greatly simplified.

1. Introduction

Transient and intermittent faults are the major cause for computer failure. Even in optimum environments it is estimated that more than 90% of the failures in computers are caused by transient faults [1]. These kind of faults can lead to erroneous behaviour of digital systems. However, no permanent damage is caused to the hardware. Hence, recovery can be performed on the same hardware provided that enough state information is preserved in some way. Therefore, it is important to include in the computer design mechanisms that allow the recovery from malfunctions caused by transient faults. This is particularly true in applications such as industrial process control, robotics, railway control systems, and other applications where high reliable computing is required. The very first step of an overall recovery process is the error detection.

During the last years there have been many proposals for the use of a watchdog processor [2] to perform concurrent system-level error detection. A watchdog processor is a small co-processor meant to detect errors by monitoring the behaviour of the system. The watchdog processor is provided previously with information about the processor to be checked. Then, errors can be detected by comparing the information gathered concurrently from the monitored processor to the information previously provided.

The information provided to the watchdog processor is an abstraction of the correct behavior of the system. This abstraction can be formed by several aspects of the system behavior: memory access behavior [3], control flow [4-9], control signals [10], or reasonableness of results [11].

Studies [12], [13] comparing several behavior abstractions have shown that program control flow gives the best error detection coverage. On-line signature analysis is considered to be the most straightforward technique for the implementation of control flow error detection. The basic idea of these error detection techniques is as follows:

1) The application program is logically divided in blocks at the assembly time;
2) Each block has one valid and unchangeable sequence of instructions (e.g. a branch free block of instructions);
3) A signature is calculated as a function of the instructions within each block (normally a cyclic redundancy code algorithm is used);
4) At runtime the signature of each block is calculated by special hardware and errors can be detected by comparison between the runtime signature and the precomputed signature.

Most of all signature monitoring techniques store the signatures within the code of the application program. This strategy causes performance overhead because the main processor wastes time avoiding the execution of the signatures as instructions. This overhead can be removed by storing the signatures in the local memory of a watchdog processor. However, this requires storage of the information about the control flow of the monitored program. This information is the program control flow graph. Each node in the graph represents an instruction block and each arc represents a possible transition between blocks. Control flow errors can be detected by the watchdog processor checking if the sequences of instructions executed by the processor correspond to a valid path in the program graph. All existing approaches in which the signatures are stored in the local memory of the watchdog processor [5], [6], [14] require the explicit

CH2964-5/91/0000-1188$01.00

or implicit storage of the program control flow graph, which leads to large memory overhead. In the approach presented in this abstract the signatures are stored in the local memory of the watchdog processor, but the storage of the control flow graph is avoided and the memory overhead is greatly reduced.

2. General configuration of a system using the Checker

A typical configuration of a system using the Checker is presented in Figure 1. A small hardware called the Signature Generator (SG) is added to each application processor (AP). The task of this circuit is to regenerate the program signatures at runtime. To accomplish that the SG detects the beginning and the end of each instruction block executed by the application processor (AP), computes the signature of the block and sends it to the Checker.

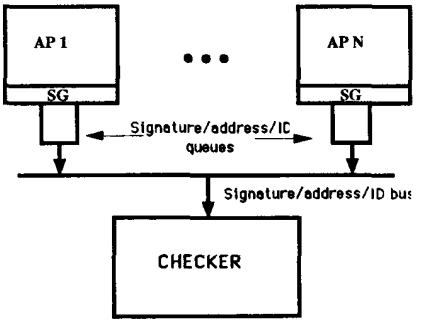

Figure 1 - General hardware configuration.

A generic implementation of a SG includes a Parallel Linear Feedback Shift Register (PLFSR), an opcode detector, an interrupt handling circuit, a signature stack, and some control logic. The complexity of the SG implementation is largely dependent on the information about the internal state of the application processor provided at the IC packaged pins. The more internal information is accessible from the outside, the easier the SG implementation is. As the recent microprocessors have internal caches and deep prefetch queues, actual implementations of the SG tend to be included into the processor chip, where all the information required for the signature generation is available.

The Checker checks the correctness of the signatures according to the signature verification principle (see next section). As the Checker is perfectly assynchronous in what regards the application programs it can easily be extended to a system containing several application processors (AP1 to AP N).

3. On-line signature verification

Signature monitoring techniques in which the signatures are stored in the local memory of the watchdog processor have three major advantages comparing to techniques using embedded signatures: 1) they do not cause performance overhead; 2) the watchdog processor and related circuits are really independent from the monitored system; and 3) one watchdog processor can be shared between several application processors, allowing error detection in multiple processor systems.

However, all existing techniques in which the signatures are stored in the watchdog processor have a very large storage overhead. The technique presented in this paper (on-line signature verification) has the advantage of having low storage overhead while keeping good error detection coverage.

The storage overhead is formed by two components: the signature overhead and the control flow graph overhead. In all existing approaches [5], [6], [14] the overhead due to control flow graph storage is much larger than the signature overhead.

In on-line signature verification the storage of the program control flow graph is avoided, thus reducing the storage overhead. In addition to low storage overhad, the presented technique also simplifies the design of the watchdog processor.

The signature verification is based on the following principles (see also Figures 2 and 3):

- The monitored program is logically divided in small enough sections, in such a way that there is only a small number N of signatures (i.e. blocks) for every program section. The number N of signatures per section should be small compared to the total number of possible signatures for a given signature size. For instance, for a 16 bit signatures (which means 65,536 different signatures) some possible choices for N are 64, 128, or 256;

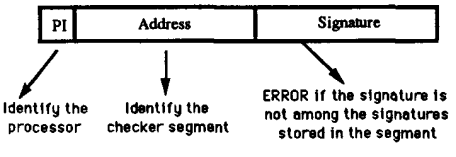

Figure 2 - Format of the information received by the Checker.

- Each program section includes as many contiguous program instructions as required to have N blocks/signatures;

- For each monitored program section there is a correspondent segment in the Checker memory where all signatures of that program section are stored;
- Every time the Signature Generator sends a signature to the Checker it also sends the address of the last instruction of corresponding block; in this way the watchdog processor can identify the segment that signature belongs to (see figure 2);
- A signature is considered correct by the Checker if it is present among the signatures stored in its segment. As the number N (signatures per section) is small this verification can be done quite fast using a binary search. For instance, for N=128 the verification takes 6 memory cycles average. Within each segment the signatures must be stored in a sorted way because of the binary search.

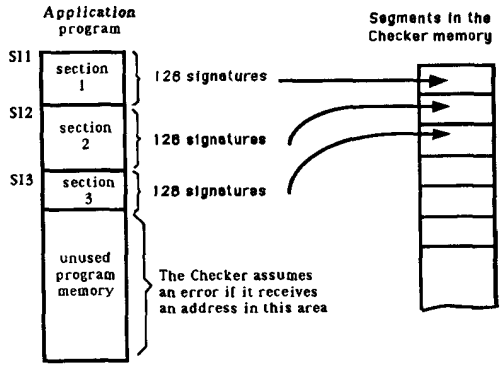

Figure 3 - The Checker principle.

The basic idea of the signature verification lies in the fact that the probability of a wrong signature being equal to any of the other signatures in the same section is very low, if a random distribution for the signatures is assumed. For instance, for sections with 64 signatures each and for 16 bit signatures, the probability of a wrong signature becoming one of the others in the same section is less than 0.0009, representing a potential error coverage of 99.9 percent.

As we can see, the Checker requires only extra storage for the segment identifiers (SI1, SI2, etc in figure 3) in order to establish a link between the address received with each signature and the Checker memory segment where that signature must be checked. For instance, for a Checker with a signature memory of 64K words and for N=128 signatures per segment the segment identifiers overhead is only 0.98% of the signature overhead. Thus, the segment identifiers overhead is irrelevant and the memory overhead required by the Checker is in fact reduced to the signature overhead. Several studies [2], [7], have suggested typical block sizes from 4 to 10 words which represent a signature memory overhead of 10 to 25 percent.

4. The experimental Checker

An experimental version of the Checker was designed both for an accurate evaluation of the hardware overhead required by the Checker and for evaluation of its real-time performance in order to estimate the Checker contribution to the error detection latency.

The Checker is a two stage pipeline machine (see Figure 4). The first stage, the Address-Segment Converter, is formed by a small memory (for the segment identifiers), a successive approximation register, and some control logic, which implements a binary search. Every time the Checker receives a set PI+Address+Signature (see figure 2) the first stage uses the information in the PI+Address fields to find the segment identifier of the signature. The segment identifier tells the second stage in which memory segment the signature should be checked. If the signature is among the signatures stored in the segment the signature is correct; otherwise it is incorrect, which means that an error has ocurred in the correspondent application processor.

The second stage (the Signature Verification stage) uses the segment identifier as part of the signature memory address. The implementation of this stage is very similar to the first. It has a signature memory, a successive approximation register, and some control logic as well.

The Checker is much simpler than other proposed watchdog processors. For example, the RMP described in [14] uses 90 SSI/MSI packages (excluding memory) while the Checker uses only 14 SSI/MSI packages.

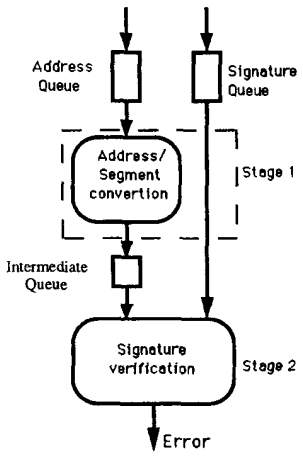

Figure 4 - Checker organization.

The signature and the address identifier memories are dual ported memories. They are accessed by the Checker during normal signature checking, and by an external computer for initial downloading of the signatures and address identifiers. The implementation of these dual-ported memories is very easy because there is no concurrent access.

5. Support software required by the Checker

As with other signature monitoring techniques the Checker requires some previous steps before it is ready for the error detection: 1) the application programs are logically divided in blocks; 2) a signature is generated for each block as a function of the instructions of the block; 3) the signatures and the segment identifiers are organized in the proper format and stored in the Checker memory.

These steps are performed by software tools working on an assembly file(s) generated by the compiler from the application program(s). The application program can be divided in basic blocks (a sequence of instructions without control flow altering instructions) or dynamic blocks [14] (a basic block with unconditional branches).

6. Summary and future work

A new watchdog processor architecture has been presented. In this approach, the storage of detailed control flow information (of the application program) is avoided and the design of the watchdog processor is very simple. A watchdog processor based on the presented approach has been built and is being used in a new signature monitoring technique, which is being investigated at the University of Coimbra.

Acknowledgement

This work has been supported in part by the Instituto Nacional de Investigação Científica (INIC) and the Ministério da Indústria e Energia under contract no. CDI-U 064/86.

References

[1] Siewiorek, D and Swarz, R 'The Theory and Practice of Reliable Design', Digital Press Digital Equipment Corporation, Massachusetts, USA (1982) pp 18; pp 113-122

[2] Mahmood, A 'Concurrent Error Detection Using Watchdog Processors - A Survey' IEEE Trans. on Computers, Vol. 37, No 2 (Feb. 1988) pp 160-174

[3] Namjoo, M 'Techniques for Concurrent Testing of VLSI Processor' Proc. Int. Test Conf., Philadelphia, USA (Nov. 1982) pp 461-468

[4] Shridhar, T and Thatte, S M 'Concurrent Checking of program flow in VLSI Processors' Proc. of 12th Int. Test Conf. (1982) pp 461-468

[5] Eifert, J B and Shen, J P 'Asynchronous Signature Instructions Streams' Proc. 14th Int. Conf. on Fault-Tolerant Computing Kissimee FL. (June 1984) pp 394-399

[6] Namjoo, M 'CERBERUS-16: An Architecture for a General Purpose Watchdog Processor', Proc. of 13th Conf. on Fault-Tolerant Computing Milano, Italy (June 1983) pp 216-219.

[7] Schuette, M and Shen, J 'Processor Control Flow Monitoring Using Signatured Instruction Streams' IEEE Trans. on Computers Vol 36 No 3 (March 1987) pp 264-275

[8] Wilken, K D and Shen, J P 'Concurrent Error Detection Using Signature Monitoring and Encryption' 1st Inte. Working Conference on Dependable Computers in Critical Application, Santa Barbara, CA, USA (August 1989).

[9] Sosnowski, J 'Detection of Control Flow Errors Using Signature and Checking Instructions' 18th Int. Test Conf. (September 1988) pp 81-88

[10] Daniels, S F 'A concurrent test technique for standard microprocessors' Proc. Compcon Spring 83, San Francisco, CA, USA (28 Feb. - 3 Mar. 1983) pp 389-394

[11] Mahmood, A, Ersoz, A and MaCluskey, E J 'Concurrent system level error detection using a watchdog processor' Proc. Int. Test Conference, Philadelphia, PA, USA (November 1985) pp 145-152

[12] Schmid, M E, Trapp, R L, Davidoff, A E and Masson, G M 'Upset Exposure by Means of Abstraction Verification' Proc. 12th Int. Symp. on Fault-Tolerant Computing, Santa Monica, CA, USA (June 1982) pp 237-244

[13] Gunneflw, U., Karlsson, J., and Torin J, "Evaluation of error detection schemes using fault injection by heavy-ion radiation", Fault Tolerant Computing Symposium, Chicago, 21-23, June 1989, pp. 340-347

[14] Shen, J P and Tomas, S P 'A Roving Monitoring Processor for Detection of Control Flow Errors in Multiple Processor Systems' Microprocessing and Microprogramming Vol 20 (1987) pp 249-269