Transient Error Detection in Embedded Systems Using Reconfigurable Components

Alireza Vahdatpour
Dependable System Lab
Sharif University of Technology
Azadi Ave., Tehran, Iran
vahdatpour@ce.sharif.edu

Mahdi Fazeli
Dependable System Lab
Sharif University of Technology
Azadi Ave., Tehran, Iran
m_fazeli@ce.sharif.edu

Seyed Ghassem Miremadi
Dependable System Lab
Sharif University of Technology
Azadi Ave., Tehran, Iran
miremadi@sharif.edu

Abstract

In this paper, a hardware control flow checking technique is presented and evaluated. This technique uses reconfigurable of the shelf FPGA in order to concurrently check the execution flow of the target micro processor. The technique assigns signatures to the main 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. The main characteristic of this technique is its ability to be applied to any kind of processor architecture and platforms. The low imposed hardware and performance overhead by this technique makes it suitable for those applications in which cost is a major concern, such as industrial applications. The proposed technique is experimentally evaluated on an 8051 microcontroller using software implemented fault injection (SWIFI). The results show that this technique detects about 90% of the injected control flow errors. The watchdog processor occupied 26% of an Altera Max-7000 FPGA chip logic cells. The performance overhead varies between 42% and 82% depending on the workload used.

1. Introduction

Please follow the steps outlined below when submitting your final manuscript to the IES symposium

Nowadays, the use of processor and microcontroller based embedded systems in real-time and critical application, such as industrial and medical system has become a common trend. It is reported in [1] that more than 99% of produced processors have been used in embedded systems. The increasing use of embedded systems augments the importance of developing techniques which enhance reliability of such applications.

Transient and intermittent faults are the major causes for digital system failures [2][3]. As it is said in [3] and [4], more than 70% of transient faults lead to control flow errors (CFE). Beside, occurrence of faults in the 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 [5].

Control flow checking (CFC) techniques have always been the most effective techniques to detect this type of errors. Basically, there are two types of CFC; software based and hardware based techniques. In software based techniques, the entire system which consists of the main program and protection mechanism is implemented in software [6][7][8][9][17]. These techniques usually consume a lot of memory space because all of the detection mechanisms are embedded inside the main program area. In contrast, the additional cost for implementing these techniques is relatively low because they usually do not require additional hardware components.

Various hardware control flow checking techniques have been presented in [5], [10], [11] and [12]. The hardware based techniques usually employ an extra hardware component as a watchdog processor [13][14].

A watchdog processor is a small and simple coprocessor used to perform concurrent system level error detection by monitoring the behavior of a main processor [15]. To reach the aim, an abstract of program execution is selected and some signatures that represent the chosen abstract are extracted for the correct program execution before the system's execution (run-time).

Furthermore, one of the main obstacles of hardware based techniques is their cost. Adding one auxiliary processor to validate the runtime execution of program, at least duplicates the cost of final system.

In this paper, a hardware based technique, which is named, Control flow checking using shadow processing is introduced. This technique is based on using reconfigurable hardware to design and develop the watchdog processor. This technique inserts some assertion in the main program as the signatures and produces 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.

Since the overhead of the synthesized watchdog processor is low, the proposed system can be potentially used in the applications which have cost constraint such as industrial applications.

The structure of this paper is as follows: Following the introduction, the error model is presented in section 2. The proposed technique will be presented in details in the third section. Section 4 discusses some optimization steps. The experimental evaluation system is introduced in section 5. In section 6 the experimental evaluation results and a comparison with other previous hardware-based control flow checking techniques is provided. Finally section seven concludes the paper and presents future works.

2. CFE Models

The presented detection techniques in this paper, primarily detects control flow errors caused by transient faults. The following terminology and definitions are required to clarify the presentation.

Definition 1: A CFE is an illegal branch which can be caused by transient faults in hardware such as the program counter, address circuit or memory system.

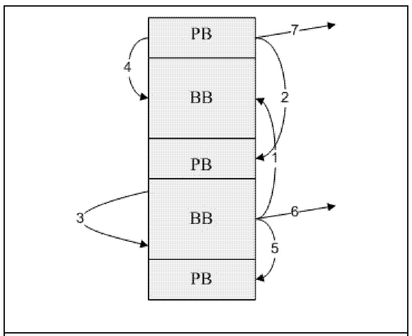
Definition 2: A Basic Block (BB) is a sequence of non-branching instructions (except in the last instruction or last consecutive instructions) or branch destinations (except in the first instruction) in which the execution always enters at the first instruction and leaves via the one of last branch instructions.

Definition 3: A Partition Block (PB) is a set of instructions between two physically consequent BBs.

Definition 4: A program crash occurs when either the execution illegally continues outside the program (unused memory space) or the processor generates an exception as the result of a CFE (e.g. invalid opcode or page fault).

We have considered that the program is partitioned into BBs. In this model, we distinguish between seven types of CFEs which are shown in Figure 1.

In fact, each PB can be defined as a BB. But because of reasons discussed in section 4, it is preferred to neglect this short program sections and not to consider them as BB.



Type 1: from a BB to other BB
Type 2: from a PB to itself or another PB
Type 3: from a BB to itself
Type 4: from a PB to a BB
Type 5: from a BB to a PB
Type 6: from a BB to the unused memory space.
Type 7: from a PB to the unused memory space.

Figure 1. Program partitions and the seven types of CFEs.

3. The Proposed Technique

In our CFC technique, which is named CFCSP (**C**ontrol **F**low **C**hecking using **S**hadow **P**rocessing), 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. (Figure 2)

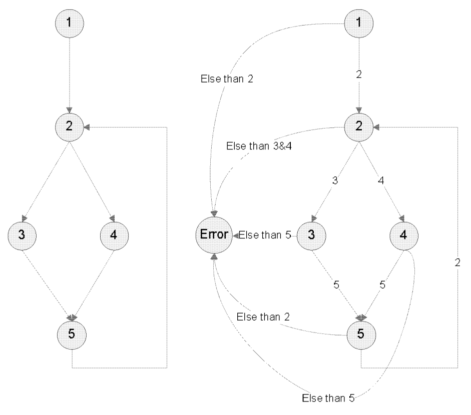


Figure 2. Program CFG and its related FSM

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 check the execution flow of the target processor. Beside the main technique which is based on mentioned FSM, three auxiliary techniques were used.

3 of 4 used techniques need signature checking, therefore it is required for target processor to send some signatures to watchdog in the execution time. Hence,

some instructions must be inserted to each basic block. As it is shown in figure 3, these instructions are inserted to the start and end of each BB.

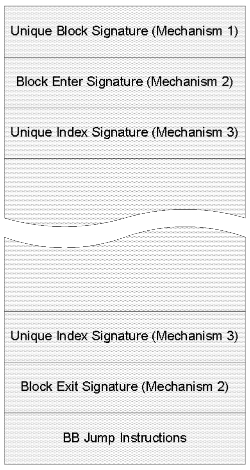

Unique Block Signature (Mechanism 1)
Block Enter Signature (Mechanism 2)
Unique Index Signature (Mechanism 3)
Unique Index Signature (Mechanism 3)
Block Exit Signature (Mechanism 2)
BB Jump Instructions

Figure 3. General basic block with inserted signatures

In the following, each technique is discussed completely.

3.1. Execution flow checking

In order to check the correctness of execution flow, each basic block has assigned a unique block ID. Upon entering the block, microprocessor sends this ID to the watchdog. Using FSM, that has prepared from the CFC, watchdog component follow the signatures that microcontroller sends out. Whenever, FPGA get wrong signatures, it will go to fault detection state and will send error detection signals.

If this block exists in the expected execution path of FSM, watchdog will change its current state to the next state related to the input block ID and again wait for the next signatures.

As it is showed in the results section, this technique is the main technique for detecting control flow errors in the system. The idea of shadow processing comes from the fact that the watchdog FPGA only pursues the flow graph of microcontrollers program and does not require computation power of the microcontroller to follow its instructions.

3.2. Enter-Exit checking

There is the possibility of jumping from the end of one block to the middle of another one. In this situation it is possible that the first mechanism fails to detect errors because the signature announcement is in the top of each block (Figure 3). In order to cover such errors an enter exit checking mechanism has been used. After entering each block, microprocessor sends an entrance message to the watchdog. Also, upon completing the execution of each block the processor sends exit signature to the FPGA. Every time watchdog checks the order of signatures and if observes any irregular sequences, It will send an error detection signal. The hardware for this algorithm is a simple three states FSM (Figure 4).

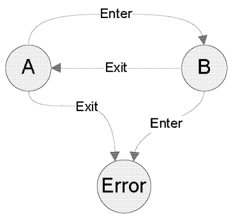


Figure 4. FSM for enter-exit checking mechanism

3.3. Block complete execution checking

In order to detect those jumps from middle of a block to middle of another one, watchdog FPGA controls block IDs in the first and end of each basic block (Figure 3).

The implementation of the circuit for this technique is a simple memory and a comparison circuit. After the first signature released, this memory is set to the value of the signature. Upon releasing next signature, FPGA compares new value with saved one. And on the conditions of inequality, an error detection signal will be announced.

In fact, this technique has overlap with the enter-exit turn checking, but the advantage of this technique is using unique IDs for each block, that leads to better overall coverage rate of the system. Furthermore, each mechanism has its own latency. Therefore, by adding redundant detection mechanisms, it is possible to decrease detection latency.

3.4. Time out checking

In case of entire system failures or jumps to the out of program area space, the system will not send any signatures and will not respond any inquiries. Therefore, using a watchdog timer, with implementation outside of target processor, is inevitable. The processor is required to send alive messages to timer before some determined time limits.

Already have inserted some signatures to the program, we also used them for live acknowledgement messages. This means that each time processor send a signature in order to check its control flow, the signature resets the timer. By using this idea, the time limit for the timer could be the longest basic block execution time. Because this is the largest time in the program that processor does not send any signature to watchdog FPGA.

4. Optimization Steps

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. (Figure 5)

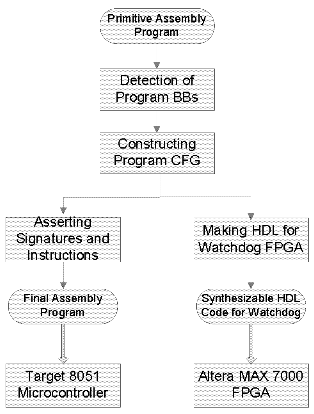

Primitive Assembly Program
Detection of Program BBs
Constructing Program CFG
Asserting Signatures and Instructions
Making HDL for Watchdog FPGA
Final Assembly Program
Synthesizable HDL Code for Watchdog
Target 8051 Microcontroller
Altera MAX 7000 FPGA

Figure 5. Work flow diagram

As mentioned, our proposed algorithm is based on basic blocks. Each basic block means a state in a FSM, which will be synthesized in FPGA. Furthermore, it is required to insert some instructions and signatures in the primitive program for each basic block. Therefore, reducing the number of considered BBs will lead to lower memory overhead and FPGA space occupation. Usually, there are some basic blocks with short length. Because of the shortness of these basic blocks, the runtime error occurrence is low in them. The best way to optimize the FSM is to remove the short length basic blocks from the CFG. The length threshold for removing BBs is highly related to the program contents and structure. Figure 3 shows an example CFG which is reconstructed by removing its shorts basic block.

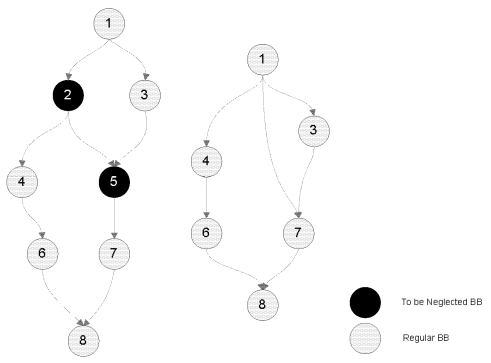


Figure 6. Sample CFG optimization

Although, removing these basic blocks from CFG effects the total coverage of the detection mechanisms, but the effect can be negligible comparing to the decreased memory overhead of the program.

5. Experimental System

The system which is used to experimentally evaluate the feasibility of the CFCSP technique and its overhead and coverage figures for the assumed error model consists of a watchdog processor, an 8051 microcontroller evaluation board, fault injection and result logging interface and a host system. Figure 7 shows the whole experimental system with its components.

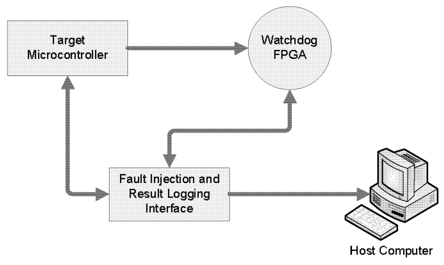


Figure 7. Experimental system diagram

5.1. Target Microcontroller

All of 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.

5.2. Watchdog System

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.

5.3. Fault Injection and Result Logging Interface

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

5.3.1. Fault Injection

Fault injection into the microcontroller must meet the conditions of considered fault model, Single Event Upset (SEU). This is done using software implemented fault injection technique (SWIFI). The Interface microcontroller interrupts target processor by setting hardware interrupt pin of target microcontroller. The interrupted target processor leaves program execution, saves its current program counter (PC) and starts a predefined interrupt routine. In this routine, a bit of saved program counter will be inverted. 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).

The injector is programmed to normally distribute injection periods on the execution time, in order to have fair testing results.

5.3.2. 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 occurred CFE. After getting error signal, interface gets detected error type and detection latency send them to the host computer.

In the case of getting program completion signatures, the results of the program will be checked to see if the unexpected jump has caused program execution error. This is because there is the rare possibility that the changed PC does not lead to CFE and hence no error will be detected. 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.

5.4. Host Computer

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

6. Experimental Results

Three benchmarks were used in order to experimentally evaluate the presented techniques; Bubble sort, link list allocation and matrix multiplication. Table1 individually show the error detection coverage of each detection mechanisms used by the proposed technique. As shown in table 1, the average error detection coverage is 90%. The error detection coverage varies between 84% and 97% depending in the benchmark used due to the different structure of the used benchmarks.

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%

In Table 2 the performance, memory overhead and the error detection latency of the CFCSP technique is reported.

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

As mentioned before, the watchdog processor is implemented in an Altera Max-7000 FPGA chip which has the total number of 5000 logic cells. Statistic in table 3 shows the low hardware overhead is imposed to the system for implementing the watchdog processor.

Table 3. Used space for implementing watchdog FPGA

	Bubble Sort	Link List	Matrix Multiplier	Average
Total occupied logic cells	26%	25%	28%	26.3

A brief comparison between the proposed technique and the previous hardware based methods is shown in table 4. Since most of the previous techniques 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 techniques 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 4. Comparison of the CFCSP technique with some of the previous, hardware-based CFC techniques [16, 10].

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

7. Conclusion and Future work

A hardware based technique control flow checking has been proposed in this article. In this technique, a simple watchdog processor which is implemented on a FPGA chip is used to control the flow of the program. Since it can be simply implemented on of the shelf reconfigurable components it is applicable to various embedded systems which are based on microcontrollers or microprocessors. Furthermore, designed software tool chain automatically performs all of the preprocessing steps on the target program. Like many other detection techniques, memory overhead and coverage rate is a trade off in the proposed technique. Optimizing this trade off by further study on the structure of industrial and commercial used programs can dramatically enhance the results of this technique. In contrast with the other technique, a key advantage of The CFCSP technique is that it is a processor independent technique and also can be used in COTS processor.

The proposed scheme was evaluated experimentally and the results show that this technique can cover about 90% of control flow errors in average. Its memory overhead is about 73% in average and the performance overhead varies between 42% and 82%. In fact, industrial applications have some evident differences with common academic benchmarks. Therefore, evaluation of these CFC techniques on real industrial applications, can give better ideas to improve the trade off.

Furthermore, the use of proposed techniques in order to overcome other fault models, such as power supply disturbance (PSD) can be evaluated.

8. References

[1] Kopetz, IFIP WG 10.4, Jan. 2002

[2] Majzik I., “Software Monitoring and Debugging Using Compressed Signature Sequences”, Proc. of the 22nd EUROMICRO Conference (UROMICRO-22), Sept. 1996, p.p. 311–318.

[3] 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.

[4] 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.

[5] Lu. D.J , “Watchdog processor and Structural Integrity Checking”, IEEE Trans. on Computer, Vol. C-31 , July 1982, pp. 681-685.

[6] Miremadi G., J. Karlsson, U. Gunneflo, and J. Torin, “Two Software Techniques for On-Line Error Detection”, Proc. of the 22th International Symposium on Fault-Tolerant Computing (FTCS-22), July 1992, pp. 328-335.

[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, pp. 441-453.

[8] Rabejac C., J.-P. Blanquart, J.-P. Queille, Lab. for Dependability Eng., CNRS, Toulouse, France, “Executable assertions and timed traces for on-line software error detection,” Proc. of the 26th International Symposium on Fault-Tolerant Computing (FTCS-26), 1996.

[9] Alkhalifa Z., V. S. S. Nair, N. Krishnamurthy and J. A. Abraham, “Design and Evaluation of System-level Checks for On-line Control Flow Error Detection”, IEEE Trans. on Parallel and Distributed Systems, Vol. 10, No. 6, Jun. 1999, pp. 627-641.

[10] Namjoo M.,“Techniques for Concurrent Testing of VLSI Processor”, Proc. of the International Test Conference (ITC), pp. 416-468, PA, Nov. 1982.

[11] Eifert J. B. and J.P. Shen. “Processor Monitoring Using Asynchronous Signatured Instruction Streams”, Proc. of the 14th International Symposium on Fault-Tolerant Computing (FTCS-14), 1984, pp. 394-399.

[12] Wilken K. and J. P. Shen, “Continuous Signature Monitoring: Low-Cost Concurrent Detection of Processor Control Errors”, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 9, June 1990, pp. 629-641.

[13] D. J. Lu, “Watchdog processors and VLSI,” in Proc. Nat. Electron. Conf., vol. 34, Chicago, IL, Oct. 27-28, 1980, pp. 240-245.

[14] A. Mahmood and E. J McCluskey, “Concurrent error detection using watchdog processors- A survey” CRCTech. Rep. 85-7, CSL TR. 85-266, Center Reliable Computing, Computer Systems Lab., Stanford Univ., Stanford, June 1985.

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

[16] A. Rajabzadeh, S. G. Miremadi, "CFCET: A Hardware-Based Control Flow Checking Technique in COTS Processors Using Execution Tracing", Elsevier Journal of Microelectronics and Reliability, Volume 46, Issues 5-6, May-June 2006, Pages 959-972.

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