International Journal of Multimedia and Ubiquitous Engineering
Vol.8, No.5 (2013), pp.145-158
http://dx.doi.org/10.14257/ijmue.2013.8.5.14

Comparative Evaluation of an Error Checking Approach for MPSoC

Tang Liu[1,2], Huang-Zhang Qin[1], Hou-Yi Bin[1], Fang-Feng Cai[2] and Zhang-Hui Bing[3]

[1]Embedded Software and Systems Institutes Beijing University of Technology
2Department of Physics and Electronics Engineering, Guangxi Teachers Education University, Nanning, 530024, China
[3]Department of computer science and Electronics Engineering, GuiLin University of Electronic Technology, GuiLin, 541004, China

tangliugx@163.com, cfengcaifang@163.com zhanghuibing@guet.edu.cn

Abstract

This paper proposes a hardware error checking approach（CCRC）by using redundancy core for multiprocessor system-on-chip (MPSoC) and describes several main error detection methods based on Software-Implemented Hardware Fault Tolerance (SHIF) idea proposed in literatures. The CCRC approach insert some error detection code in high level code, detect the existing of redundancy core in MPSoC, then complete the calculation of detection code in redundancy core. The author compares the CCRC approach with several main error detection methods on error detection capabilities, area, memory and performance overheads in an experiment platform. The result of comparative evaluation shows that the CCRC approach is effective for MPSoC, taking some advantages in versatility and lower cost.

Keywords: *MPSoC , hardware error, redundancy core, Comparative evaluation*

1. Introduction

The increasing computational power requirements recommend development of multi-processor system-on-a-chip (MPSoC). Unfortunately, the high integration level of circuits may be particularly sensitive to transient faults, which may possibly cause failures [1]. In this scenario, new techniques that are able to detect, locate, and correct the effects of transient faults in MPSoC are fundamental [2].

At present, several error detection techniques have been proposed in the literature [3-14], where each option has different tradeoff options in terms of energy, performance, area, coverage, complexity, and programmer effort. Software-based techniques introduce some sort of redundancy by adding code executed in microprocessor. Among the most important solutions proposed in the literature, there is the Enhanced Control Flow Checking Using Assertions (ECCA) and the Control Flow Checking by Software Signatures (CFCSS) [9, 10]. -- Other techniques harden SoCs against faults affecting the data they elaborate. An example of such a group of techniques is given by the method proposed in [11], which is based on instruction-level duplication and is able to achieve detection of the data faults. Another approach, named Error Detection by Data Diversity and Duplicated Instruction (ED4I), has been proposed in [12]. Some solutions combining hardware and software techniques have been proposed. These solutions usually referred to as hybrid fault detection solutions [13, 14] can achieve a better trade-off between fault coverage and area as well as performance overheads. A hybrid approach typically introduces some reduced redundant information in the

ISSN: 1975-0080 IJMUE

application code and data, while resorting to an I-IP to perform on-the-run consistency checks [14].

For adapting MPSoC System this paper proposes the CCRC (the calculation task of detecting code on redundancy core) approach by using redundancy core. It inserts some error detection code in application software, detect the existing of redundant core in MPSoC, then move the calculation and comparison task of detection code to it. This technique suit to the MPSoC design flow and get a better trade-off between performance and overhead with relatively high error checking rate. This paper induces the main idea of CCRC approach and several representative error detecting techniques mentioned above, comparatively evaluate the CCRC approach in an FPGA-Based MPSoC system, and give the best option for a given operating scenario and application.

The rest of the paper is organized as follows: Section 2 gives a brief related works of error checking in SoC; Section 3 explains the CCRC and several main error detection mechanisms; Section 4 explains the verification and comparison experiment platform and plan; Section 5 shows the experiment result analysis; Section 6 gives a conclusion.

2. Related works

There are two main hardening techniques for SoC fault tolerance: hardware-based and software-based technique according to its placement in the system hierarchy. The hardware-based technique main based on a design the dependability system architecture, *i.e.*, design custom circuit or adding new core to achieving dependability enhancement in different system level [3, 4, 5]. This approach has less effect in computing performance but it needs additional hardware it needs more area overhead, lack scalability and difficult to transplant . The software-based techniques introducing some sort of redundancy by adding code executed in microprocessor [9-10, 12] to check the software control flow and data consistency. Those approaches are easy to use but need high overhead in code size and decreased performance. Except that, another class of error detection technique has been developed that relies on software level symptoms to detect errors [6-7]. The main idea of symptom-based detection technique: Fault injection experiments show that the underlying hardware error will spread to the upper software system, so we only need to deal with the error can ultimately have an impact on the system. This technique can detect transient errors and permanent faults. In addition, those hybrid methods which combine the hardware-based and software-based method have been proposed [13-14].

3. Introduction of Error Detection Approach

The approach described in this section based on the idea of SHIFT. It is a low cost alternative to hardware fault because it does not require any hardware modification and can be easily adapted to most hardware platforms. In this section, we will begin with definitions of terminologies and then describe the main idea of these methods respectively.

A basic block is a sequence of consecutive statements in which the flow of control enters at the beginning and leaves at the end without branching except at the end. By defining $V=\{v_1,v_2,\ldots,v_n\}$ as the set of vertices representing basic blocks and $E=\{(i,j)|(i,j)$ as a branch from v_i to $v_j\}$ and the set of edges denoting possible flow of control between the basic blocks, a program can be represented by a program graph, $PG=\{V,E\}$.

As an example we consider the code fragment shown in Figure 1, where the basic blocks are also numbered in (a) and the corresponding program graph is shown in (b).

(a)

B1	i=1;
B2	While(i<10)
B3	{if（a[i]<b[i]）
B4	z[i]=a[i];
B5	i=i+1;}
B6	

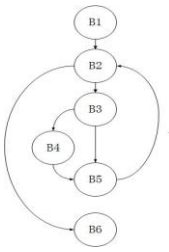

Figure 1. Example of source code and its corresponding program graph

3.1 Software

In software-based approaches, the reliability is generally achieved by combining data-flow and control-flow checking techniques. Control-Flow checking is usually performed by comparing a run-time signature with a pre-computed one. On the other hand, data-flow checking techniques rely on redundant computation by replicating instructions. This section describes the technique based on the introduction of additional executable assertions to check the correct execution of the program control flow [11] and the technique based on a set of rules for introducing redundancy in the high-level code for detecting transient error affected data [8].

In literature [11], the proposed approach checks the control flow of programs by using a dedicated global integer variable (call code), which contains the run-time signature associated with the current node in the program flow graph. Every basic block is identified by a unique signature defined at the compile time. The following assertions are introduced into each basic block v_i.

A test assertion controls the signature of the previous basic block and checks if it is permissible, according to the Program Graph, *i.e.*, if the entering node v_j belongs to the pried(v_i)

A set assignment updates the signature, setting it to the corresponding value B_i.

During the program execution, for each basic block, the global integer variable checks if the block is reached from a legal block (according to the Program Graph), otherwise a control flow error is detected.

The main of this approach is in each basic block the test and set statements are introduced at the beginning and at the end of the block. It allows the approach to cover all the single control flow faults, including the ones not crossing the block boundaries.

In literature [8], the author proposed a method which performs control flow and data detection by redundancy. Its basic ideas behind a set of transformation rules to get data and code redundancy is as following:

Every variable x must be duplicated: Let x_0 and x_1 be the names of the two copies. Two sets of variables are thus obtained, the former (set (v_0)) holding all the variables with footer 0 and the latter (set (v_1)) holding all the variables with footer 1.

Every write operation performed on a variable x must be performed on its replica x_0 in v_0 (using only variables belonging to v_0 and its replica x_1 in v_1 (using only variables belonging to v_1); after each read operation on a variable x, the two replicas x_0 and x_1 must be checked, and if an inconsistency is detected, an error procedure is activated.

In this paper, we combine above two methods to achieve the detection of control flow and data consistency and compare evaluate it by experiment.

3.2 Ed^{4}I [12]

Ed^{4}I is a technique that detects both permanent and temporary errors by executing two "different" programs (with the same functionality) and comparing their outputs [12]. ED^{4}I map each number x, in the original program into a new number x', and then transforms the program in order to operate on the new numbers for the results can be mapped backwards for comparison with the results of the original program.

It is the key method of Ed^{4}I that the transformation algorithm transforms a program (integer or floating point numbers) to a new program with diverse data. First of all, we show the definitions of terminologies and then describe the transformation algorithm.

If x is k times greater than y, x is k-multiple of y. Program transformation transforms a program P to a new Program P' with diverse data in which all variables and constants are k-multiples of the original values when the program P is executed. It consists of two transformations: expression transformation and branching condition transformation. The expression transformation changes the expressions in P to new expressions in P' so that the value of every variable or constant in the expression of P' is always the k-multiple of the corresponding value in P. The branching condition transformation adjusts the inequality relationship in the conditional statement in P' so that the control flow in P and P' is identical.

A k-factor diverse program is a program with a new program graph p'G= {v', E' } which is isomorphic to PG, but all the variable and constants in P' are ok-multiples of the ones in P.S and S' are the sets of variables in P and P' respectively, and n are the number of vertices (basic blocks) executed ;

S (n): the set of values of the variable in S after n basic blocks are executed ,

S' (n): the set of values of the variable in S' after n basic blocks are executed

The program transformation should satisfy:

- PG and P'G are isomorphic;
- K*S (n) =S' (n) , for ∨n>0. (Where k*S (n) is obtained by multiplying all elements in S (n) by k).

The example of transforming a program to a diverse program is shown in figure 2.

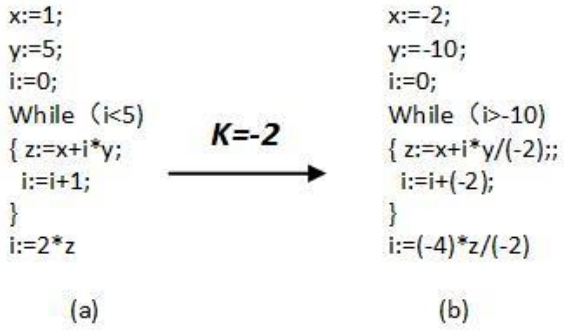

Figure 2. The original program transform to a new program with k=-2[12]

The factor k determines how diverse the transformed program is. So it is important that how to choose an optimal value of K that maximizes the diversity of the transformation

program. The literature [4] show that, for integer programs, the transformation with k=-2 was the most desirable choice in six out of seven benchmark programs.

This technique plays an important role when hardware design is fixed (such as COTS components or IP blocks in SoC design and cannot be modified in embedded systems that operate in safety and mission-critical applications.

3.3 Hybrid [14]

The contribution of the hybrid method is described in the literature [14] in the development of a new Infrastructure IP suitable to be adopted to improve the reliability of processor-based SoC: The goal of this I-IP is to guarantee high detection capability with respect to transient faults affecting data and code memory, as well as the memory elements within the processor. The hybrid integrates the solutions presented in [8] and [11], moves the computer effort to external Infrastructure IP (I-IP) to reduce its cost and enhance its performance in terms of fault detection capabilities. It is not only addresses faults affecting both data integrity and control flow check, but demonstrates that cleverly combining together the previous approaches result in higher detection capabilities with reduced overhead.

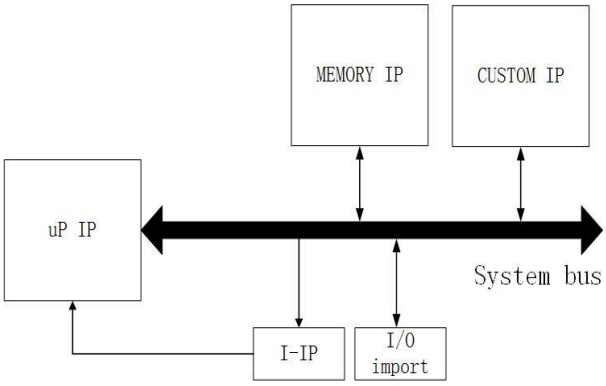

Figure 3. The architecture of the generic SoC system including the fault detection-oriented I-IP [14]

The architecture of the system including the I-IP is reported in Figure 3. According to the hybrid method, the code is in charge of signaling the I-IP when a new basic block is entered. Since the I-IP is not intended to record any information about the application code, the hardened program must send to the I-IP all the information required to check whether the new block can be legally entered according to the list of previous blocks. The I-IP records the current signature in an internal register. Once it is informed that a new block is entered and it has received the list of blocks that can legally reach the new block, it checks whether the stored signature is included in this list. If not, the ERROR signal is raised. Otherwise, the current signature is updated with the signature of the new block.

The hybrid method can be easily adopted in the typical SoC design flow, *i.e.*, it requires minimal changes in the hardware (apart from the insertion of the I-IP), while the software is simplified with respect to the purely software fault detection approach above.

3.4 The CCRC approach

The CCRC approach, inherited the idea of SHIF, insert some redundant codes in high-level codes to check hardware error. At the same time, it takes advantage of the natural redundancy

characteristics of a multi - processor system, *i.e.*, migrate the main calculation and comparison task of redundant code to the redundancy core. The CCRC approach suits to the MPSoC design flow without any changes in the hardware. It Combines the benefits of pure software approach, *i.e.*, none additional hardware cost, and the advantage of multi-core resources to reduce the impact of the performance and get better balance between the detection capacity and overhead.

The CCRC approach transforms the original program using the rules in literature [8] and literature [11] and introduces some functions, *i.e.*, RCdetect() ,flag(), RCtest() and RCset() to complete the fault detection of MPSoC system.

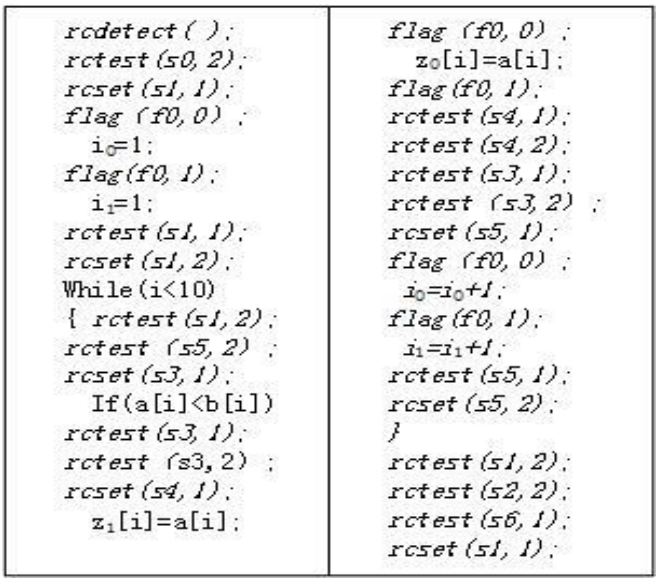

Figure 4. The transformation program for MPSoC

The harden code combining data and control flow check for MPSoC is shown in Figure 4. (Example program in Figure 1)

The roles of three high-level functions, RCdetect(), RCtest( ) and RCset( ) are as the following:

RCdetect（）: determine whether there is a redundant core in the system

RCset（B_i）: informs the spare core that the program has just entered into basic block Bi

RCtest（B_j）: informs the spare core that block Bj belongs to the set of the predecessors of the newly entered block.

The redundancy core contains two registers, A and B. The parameter of the function is written in the register, thus becoming available to the redundancy core for processing. A sequence of calls to the two functions should be inserted in the code at the beginning and at the end of each block B_k. First, a call to RCtest(B_i) is inserted for any block Bi prev(B_k). Then, a call to RCset(B_k) is inserted. When noticing a write operation on register A, the redundancy core set or reset an internal flag, depending on the result of the comparison between the function parameter and the internally stored signature. When noticing a write operation on register B, the redundancy core verifies the value of the flag and possibly activates the ERROR signal. Otherwise, the signature of the current block is updated using the value written in register B.

For Support data checking the redundancy core need to know the addresses of the two replicas of the same original variable belonging to the core we monitor and to understand whether a given address corresponds to the first or second replica. Moreover, it is important to note that the two bus cycles accessing the two replicas of the same variable are not necessarily consecutive. In fact, the compiler often reorganizes the assembly code so that instructions are recorded in such a way that the two instructions are interleaved with others. In literature [14], the author assumed that the compiler never modifies the code in such a way that the second replica of a variable is accessed before the first replica. But this solution limits the generality of their methods. Besides that, for the technique we proposed must be identified the access cycle belong to which core in MPSoC. For resolving the above problem, we induced a function flag($f_{i,j}$), which send information to redundancy core before read variable for identifying which core and replica of the variable is accessed. There are two registers C and D in redundancy core which place the parameter of flag().When redundancy core receives the information, it informs the other core do not to use the bus. The values of register C and D separately represent the sign of the processor core and the collection of variable belongs to, *i.e.*, the first replica or the second replica collection. The redundancy core provides an ADM memory, which is used to store the address-data copy corresponding to each variable accessed in memory, whose replica has not been accessed yet. The redundancy core has two registers to store the core flag of MPSoC and in charge of informing other cores not to use the bus.

The redundancy core implements the data checking algorithm as following:

1）If receiving the flag information from the checked core, the redundancy core updates the core register C and replica flag register D, informs the other core do not use buses.

2） If a memory read is detected on the bus, the address and data values are captured.

3）If the read operation relates to the first replica of a variable, a new entry is inserted in the ADM, containing the just captured address and data values.

4） If the read operation relates to the second replica of a variable, an access is made to the ADM:

- If an entry with the same address is not found, the ERROR signal is raised.

- Otherwise, the data is compared with the one stored in the ADM entry and the ERROR signal is raised in the case of a mismatch.

- The entry is removed from the ADM.

When the end of a basic block is reached, the ADM should be empty since the two replicas of all the variables should have been accessed. If this is not the case, an error (most likely, a control flow error) has happened: The ERROR signal is raised.

In harden code program, after the first replica variable is complier, the parameter of the flag() is set 0, when a second replica of the variable is complied, the parameter of the flag() be set 1. As soon as a memory access cycle is detected on the bus, the address and the value of the accessed variable will be extracted respectively. If value of register D is 0, the first replica of the variable is currently being accessed; otherwise, the second replica is being accessed.

The main work of redundancy core as follows:

- Communicate with the other processor core in MPSoC
- Decodes the bus cycles being executed and, in case of read or write cycles to the memory, it samples the address (adx) and the value (data) on the bus.

- Calculates the address of the corresponding replica, accesses the ADM memory, and verifies whether the searched entry exists for verifying whether any data stored in the memory or the processor has been modified.

The proposed solution of using redundancy core to check computing can be easily adapted to different MPSoCs and can be reworked for adapting to the bus protocol implemented because it is one part of the MPSoC architecture. In addition, when we design a MPSoC system considered dependability, we only need to consider how to program the function of RCdetect(), RCtest(), RCset() and flag() respectively.

4. Experiment Environments for Verification

4.1 Architecture of experiment platform

The experiment MPsoc system platform includes fault inject function. The architecture of the experiment MPSoC system is presented in Figure 5. We developed a prototypical implementation of MPSoC and mapped it on a Xilinx Virtex-6 FPGA. The system consists of multiple processing elements (PowerPC or microblaze), peripherals and PLB bus. The core of MPSoC we proposed is connected to the PLB bus. Every processing element includes a internal memory to store the redundancy code for checking the error. When the redundancy core detects an error, it activates an ERROR signal, which can be sent either to the processor or to the outside, depending on the preferred recovery scheme.

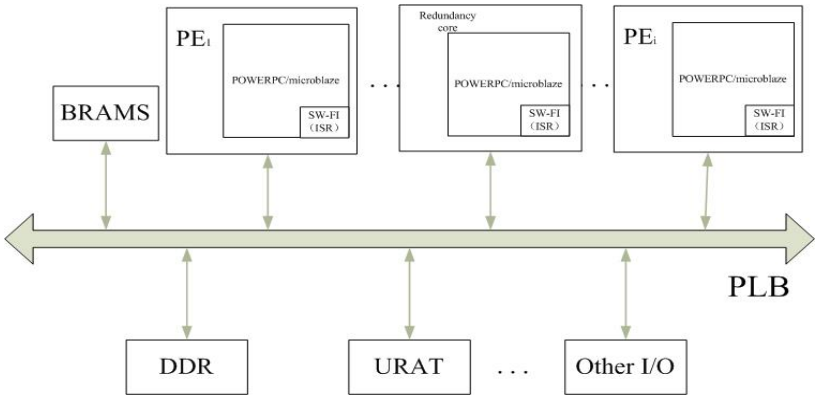

Figure 5. The architecture of the MPSoC with fault inject system

To emulate an SEU by flipping any writable bit within the processor including the general purpose register, special purpose register and the instruction we run an interrupts service routine (ISR) in the processor (PowerPC or Microblaze). The SW-FI responsible for performing the necessary system to introduce an SEU(bit-flip) into the PowerPC. It can modify any software writable register [15]. The type of injection is chosen at random by a pseudo-random number generator.

4.2 Experiment Plan

We performed several fault-injection experiments while the system is running a program. We repeated the injection experiments for three benchmark programs that are inspired by those in the EEMBC automotive/industrial suite:

- Matrix multiplication of two 3×3 matrices: it computes the product of two 3×3 integer matrices.
- Fifth Order Elliptical Wave Filter: It implements an elliptic filter over a set of 6 samples.
- Lempel-Ziv-Welch Data Compression Algorithm: It compresses a stream of 8 characters.

For each such benchmark, up to four different implementations have been compared:

- Plain: The plain version of the considered benchmark; no hardware or software fault detection techniques is exploited.
- Software: the basic version of using the rule in [8] and [11]
- ED^{4}I: The hardened version of the benchmark, obtained using the purely software hardening approach described in [12].
- Hybrid: The hardened version of the benchmark, obtained using the hybrid approach proposed in paper [14]
- CCRC: The hardened version of the benchmark, obtained using the new approach we proposed CCRC in this paper.

To model the effects of SEUs, we exploited the transient single bit flip fault model, which consists of the modification of the content of a single storage cell during program execution.

When performing the fault injection experiments, we classified fault effects as follows:

- Time-out: The fault modified the program execution in such a way that the processor entered an endless loop, thus not providing final results.
- Detected: The fault modified the program execution, but some of the detection features detected it.
- Wrong Answer: The fault modified the program execution and its output results differed from the expected ones. These faults are the most critical ones and their number should be minimized by the adopted fault detection mechanism.

We implemented MPsoC system by microblaze soft core on a Virtex-6 FPGA. The core connects to other core and the local memory through the Processor Local Bus (PLB). The processor elements (core) are working at 100Mhz. In our experiments, we injected 50 000 randomly selected SEUs in the processor's internal and external memory elements.

5. Experiment Result

5.1 Fault Detection Capabilities Analysis

The result of fault injection is shown in Figure 6:

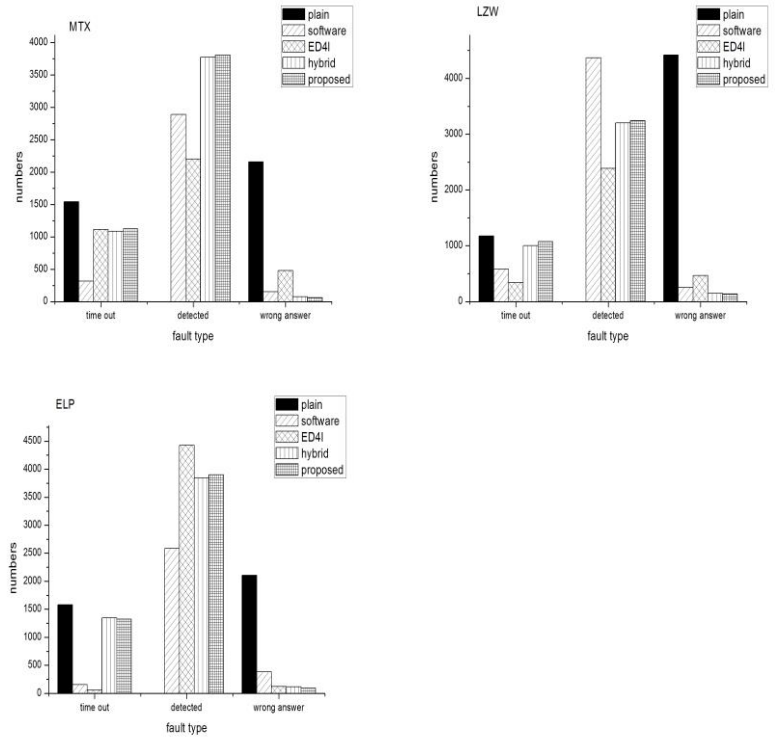

Figure 6. Fault Injection Results

As we can see, the time-out and the wrong answer fault of the Plain version is the highest, as well as zero detected in all versions, because there is none an error checking measure with it. For other versions, the numbers of leading effect are following:

1) Time-out: On average, the faults number of the ED^{4}I method leading to time out is the lowest. Software approach may introduce additional branches to the Program Graph to continuously check the value of the ERROR flag. Thus, the possibility of time out of Software detection techniques increases. Conversely, for the Hybrid and CCRC approach, no additional branches are introduced, resulting in a lower number of faults leading to time out situations.

2) Wrong answer: ED^{4}I methods significantly reduced; when exploring the Hybrid method and CCRC method, once the unexpected content appears in the memory which store variable replica during data checking, evoking an error signal .Then program generally come back normal execution flow. So, these two approaches can be further reduced the wrong answer than Software approach.

3) Detected: Since the CCRC and Hybrid can check some fault by control and data check mechanism make the program come back to normal execution flow, the detection capabilities are higher than Software method. There is little difference between Hybrid and CCRC, CCRC approach shows a little higher detection than hybrid approach.

5.2 Overhead analyses

The approach we proposed includes three types of overheads with respect to the unhardened version:

- Area overhead, related to the adoption of a redundancy core;
- Memory overhead, due to the insertion in the code of the RCdetect(), RCtest(), RCset() and flag() functions and to the duplication of variables;
- Performance overhead, as additional instructions are executed.

To quantify the memory and performance overheads, we measured the memory occupation of the programs that were hardened according to these techniques. We also measured the area occupation and program execution time of the original programs. Memory occupation was measured in terms of number of bytes in the code segments, while duration was measured in terms of the number of clock cycles for program execution. The result is shown in Figure 7.

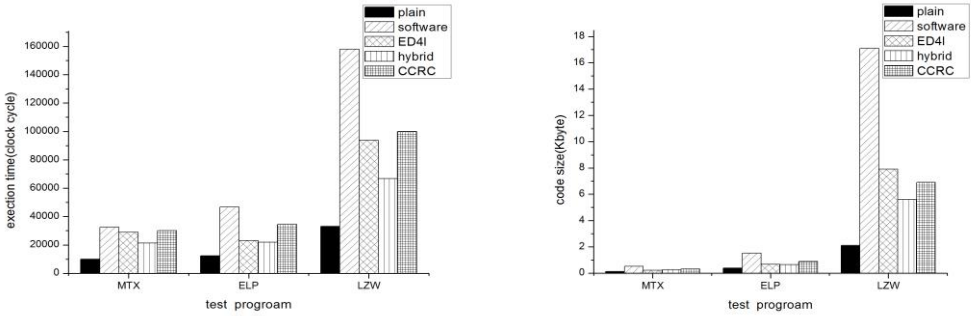

Figure 7. Performance and Memory Overheads of considered program

Results reported in Figure 7 show that the performance overhead of the Hybrid version is, about 50% on average of that of the one with Software version. The average block size of the LZW programs is smaller than in the other programs LZW show a higher code overhead. The ratio between branch and functional instructions is higher in LZW. The CCRC version's memory and performance overhead are little higher than Hybrid version.

5.3 Comprehensive comparison

We compare these techniques with Plain version in adapting system, error detection capability, hardware overhead, computing performance cost and code overhead. The result is shown in Table 1

Table 1. The comprehensive comparison of error detection methods

Fault methods	Adapting system	Error detection rate （%）	Hardware overhead （%）	Performance overhead （%）	Code size (%)
Software	Processor-based	84.5	0	3.76	3.79
Ed4I	Processor-based	76.2	0	2.87	1.73
Hybrid	SoC	74.3	5	1.77	1.68
CCRC	MPSoC	74.2	3	2.79	2.3

As observed, The CCRC method can be used to MPSoC system, and other approaches adapt SoC or processor-based computing system. The detection rates of Hybrid and CCRC

dropped 10 percent, with their impact on performance reduced by half. The Code size of Hybrid and Ed^{4}I are also reduced by half. The code size of CCRC is little more than hybrid method, but the hardware overhead of CCRC method is lower than the hybrid method.

6. Conclusions

In this paper we proposed an error checking approach （CCRC） combined the software-based techniques and redundant character of MPSoC. The fault detection capabilities of this approach have been experimentally measured by performing extensive fault-injection campaigns; the comparative evaluation result shows that the main advantages of the CCRC approach are the following:

- It is suitable to MPSoC design since it need no changes in the system design description
- It is both versatile and scalable. It does not require a special IP core or other hardware module design for different system because that the redundancy core is completely independent of the code executed by the processor.
- It insures fault detection capabilities and takes advance in lower cost of hardware and code size.

Acknowledgements

The authors would like to thank Guangxi Natural Science Foundation (2012GXNSFBA053171, 2013GXNSFAA019324), Guangxi Experiment Center of Information Science（20130321）, the Dean's fund of the Guangxi Key Laboratory of Trusted Software (kx201214, kx201203)

References

[1] J. O. R. Henkel, L. Bauer, J. Becker, O. Bringmann, U. Brinkschulte, S. Chakraborty, M. Engel, R. Ernst, H. H A Rtig, L. Hedrich, A. Herkersdorf, R. U. D. Kapitza, D. Lohmann, P. Marwedel, M. Platzner, W. Rosenstiel, U. Schlichtmann, O. Spinczyk, M. Tahoori, J. U. R. Teich, N. Wehn and H. Wunderlich, “Design and architectures for dependable embedded systems”, in CODES+ISSS '11, **(2011)** New York, NY, USA.

[2] P. Bernardi, L. M. V. B. Poehls, M. Grosso and M. S. Reorda, “A Hybrid Approach for Detection and Correction of Transient Faults in SoCs”, IEEE Trans. Dependable Sec. Comput., vol. 7, **(2010)**, pp. 439-445.

[3] N. Hebert, G. M. Almeida, P. Benoit, G. Sassatelli and L. Torres, “A Cost-Effective Solution to Increase System Reliability and Maintain Global Performance under Unreliable Silicon in MPSoC”, IEEE Computer Society, **(2010)**, pp. 346-351.

[4] H. Pham, L. Devaux and S. Pillement.Re2DA: Reliable and reconfigurable dynamic architecture. Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC). 6th International Workshop on **(2011)**, pp.1-6.

[5] R. Obermaisser and O. Hoeftberger, “Fault Containment in a Reconfigurable Multi-Processor System-on-a-Chip”, In Proceedings of 21st IEEE International Symposium on Industrial Electronics, **(2011)**.

[6] M. Li, P. Ramachandran, S. K. Sahoo, S. V. Adve, V. S. Adve and Y. Zhou, “Understanding the propagation of hard errors to software and implications for resilient system design”, In ASPLOS XIII, **(2008)**, New York, NY, USA.

[7] M. Li, P. Ramachandran, S. K. Sahoo, S. V. Adve, V. S. Adve and Y. Zhou, “SWAT: An Error Resilient System”, **(2008)**.

[8] P. Cheynet, B. Nicolescu, R. Velazco, M. Rebaudengo, M. S. Reorda and M. Violante, “Experimentally Evaluating an Automatic Approach for Generating Safety-Critical Software with Respect to Transient Errors”, IEEE Transactions on Nuclear Science, vol. 47, **(2000)**, pp. 2231-2236.

[9] Z. Alkhalifa, 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 Transactions on Parallel and Distributed Systems, vol. 10, **(1999)**, pp. 627-641.

[10] N. Oh, P.P. Shirvani and E.J. McCluskey, "Control-Flow Checking by Software Signatures", IEEE Trans. Reliability, vol. 51, no. 2, **(2002)**.
[11] Goloubeva, O. Rebaudengo, M. Reorda, M. Sonza and M. Violante, "Soft-Error Detection Using Control Flow Assertions", Proceedings of IEEE Symp Defect and Fault Tolerance in VLSI Systems, **(2003)**, pp. 581-588.
[12] N. Oh, S. Mitra and E. J. McCluskey, "ED4I: Error Detection by Diverse Data and Duplicated Instructions", IEEE Trans. Computers, vol. 51, no. 2, **(2002)**, pp. 180-199.
[13] R. Gong, W. Chen, F. Liu and K. Dai, "Control Flow Checking and Recovering by Compiler Signatures and Hardware Checking", Journal of Computer Research and Development, vol. 46, no. 2, **(2009)**, pp. 345-351.
[14] P. B. Bolzani, L. M. V. Rebaudengo, M. Reorda, M. S. Vargas, F. L. Violante and Massimo, "A New Hybrid Fault Detection Technique for System-on-a-Chip", IEEE Trans. Computers, vol. 55, no. 2, **(2006)**, pp. 185-198.
[15] M. Bucciero, J. P. Walters and M. French, "Software fault tolerance methodology and testing for the embedded PowerPC", Washington, DC, USA: IEEE Computer Society, **(2011)**, pp. 1-9.

Authors

Tang Liu was born in Guangxi, China, on August 8, 1977.She received the B.S. Degree in applied electrical technology from South-Central University for nationalities, Wuhan, China, in 1999, the M.S. Degree in control theory and control engineering from Guangxi University, Nanning, China, in 2005.

She is currently pursuing her PHD at Beijing industrial university and is an instructor at Guangxi teachers Education University. Her research interests include embedded software and systems, hardware and architecture, and multiprocessor system-on-chip.

Huang Zhang-Qin (M'95) was born in Zhejiang, China, on December 28, 1965. He received the B.S., M.S., and Ph.D. degrees in computer science from Xi'an Jiaotong University, Xi'an, China, in 1986, 1989, and 2000, respectively, and the Postdoctoral degree from the Technische Universiteit Eindhoven (TU/e), Eindhoven, The Netherlands, in 2003.

He is currently the Deputy Director of the Embedded Software and Systems Institute (ESSI), Beijing University of Technology (BJUT), Beijing, His current research interests include co-design for embedded software and hardware, human-computer interaction based on Internet, mass data storage, and network information security. He has authored or co-authored more than 50 papers.

Hou Yi-bin (M'85) was born in ShanXi, China, on April 12, 1952. He received the B.S. Degree in electrical engineering and computer science and the M.S. Degree fromXi'an Jiaotong University of Xi'an, Xi'an, China, in 1975 and 1981, respectively, and the Ph.D. degree from the Technische Universiteit Eindhoven (TU/e), Eindhoven, The Netherlands, in 1986.

He is currently the Deputy Principal of Beijing University of Technology (BJUT), Beijing, China, where he is also the Dean of the School of Software Engineering as well as a Professor. He has authored or co-authored more than 100 journal papers and 30 conference papers. His current research interests include human–computer interacting

system, embedded software and systems, and ambient intelligence. He is also a reviewer of the Journal of Computer and System Sciences of USA. Dr. Hou is a member of the New York Academy of Science.

Fang Feng-Cai was born in Guangxi, China, on December 8, 1973.He received the B.S. Degree in applied electrical technology from Guangxi teachers Education University, Nanning, China, in 2003.

He is currently an instructor at the Guangxi Education University. Her research interests include Single-chip microcomputer system application, circuit design.

Zhang Huibing was born in Henan, China, on December 14, 1976. He received the B.S. and M.S. Degrees in computer science FromGuilin University of Electronic Technology (GUET), Guilin China, in 2000 and 2005, respectively, and the Ph.D.degree in computer science from Beijing University of Technology (BJUT), Beijing, in 2012.

He is currently a part of the Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin. His research interests include embedded systems, Ambient Intelligence, Service-Oriented Computing and Trusted Computing.