行政院國家科學委員會專題研究計畫 成果報告

行為層高效能處理器的容錯設計及快速驗證與容錯能力分析(II)
研究成果報告(精簡版)

計 畫 類 別 ：個別型
計 畫 編 號 ：NSC 96-2221-E-216-006-
執 行 期 間 ：96 年 08 月 01 日至 97 年 07 月 31 日
執 行 單 位 ：中華大學資訊工程學系

計 畫 主 持 人：陳永源

計畫參與人員：碩士班研究生-兼任助理人員：汪碩彥
碩士班研究生-兼任助理人員：許書豪
碩士班研究生-兼任助理人員：彭建閔

報 告 附 件 ：出席國際會議研究心得報告及發表論文

處 理 方 式 ：本計畫可公開查詢

中 華 民 國 97 年 10 月 31 日

Summary

This report describes the results achieved in the second year of three-year research proposal. As mentioned in the proposal, an important issue in the design of high reliable system-on-chip (*SoC*) is how to verify the robustness of the system, the safety-critical components and the feasibility of the fault-robust design as early in the development phase to reduce the re-design cost. Therefore, a system-level fault-tolerant verification platform is required to assist the designers in assessing the dependability of a system with an efficient manner. The study is to propose a system-level fault injection framework in SystemC design platform to assist the dependability assessment. The proposed fault injection framework consists of two kinds of fault injection techniques: simulation-based and software-implemented fault injection schemes. In this year, we first enhance the simulation-based fault injection platform by devising a system bus fault injection methodology. As we know, the system bus, such as AMBA AHB, provides an integrated platform for IP-based *SoC*. Apparently, the robustness of system bus plays an important role in the *SoC* reliability. So, performing the system bus failure mode and effects analysis (FMEA) is imperative to validate the reliability of *SoC*. Secondly, we construct a fault injection tool under the environment of CoWare Platform Architect. The tool deals with fault injection at different modeling levels of abstraction and the fault trigger can be time-driven or event-driven approaches. The proposed fault injection tool can significantly reduce the effort and time for performing the fault injection campaigns. In addition to that, the tool dramatically increases the efficiency of carrying out the FMEA and system robustness validation. We demonstrate the feasibility of the proposed fault injection framework with an experimental ARM-based system that is modeled at different levels of abstraction. The details of this part can be found in the "SoC-Level Fault Injection Methodology and Tool Development in SystemC Design Platform" (page 1).

Simulation-based fault injection in IP-based *SoC* design platform has a difficulty in injecting the faults into the inside of IP components, especially for processors and memory modules. Such a limitation confines the injection capability for IP-based *SoC*. To cope with this problem, we also develop a software-implemented fault injection (SWIFI) mechanism under UNIX or Linux operating systems, which allows us to inject the faults into the processor registers and the memory. At current stage, we implemented the proposed SWIFI technique in the ARM926EJ-S development board embedded with Open Linux 2.6.19 to validate its feasibility. The experiments of FMEA were conducted to analyze the system behavior and the failure sensitivity for the errors occurring in the ARM processor registers. Since CoWare Platform Architect currently does not provide the UNIX or Linux operating systems in the design environment, we cannot port the proposed SWIFI scheme to the SystemC design platform. To solve it, we now cooperate with Chip Implementation Center (CIC) to try to port the Linux operating system to the CoWare Platform Architect. When this is done, we will integrate the presented SWIFI technique into our fault injection tool to expand its injection ability and diversity. The details of this part can be found in the appendix (page 12).

The other important issue is how to locate the safety-critical components in the system. For the complicated embedded systems or IP-based SoC, it is unpractical and not cost-effective to protect the entire system or SoC. Analyzing the vulnerability of a system can help designers not only to invest limited resources on the most crucial region but also to understand the gain derived from the investment. In this part of study, we propose a model to fast estimate the microprocessor's vulnerability with only slight simulation effort. From the assessment results, the rank of component vulnerability related to the probability of causing the microprocessor failure can be acquired. The ranking results can be used to achieve an effective fault-tolerant design. By choosing one of the mainstream microprocessors — VLIW (Very Long Instruction Word) processor — as an example, the practical usefulness of our estimation model is demonstrated. The details of this part can be found in the "An Estimation Model of Vulnerability for Embedded Microprocessors" (page 9).
Keywords: FMEA, fault-tolerant design, high-level abstraction modeling, high-level rapid verification, SystemC, system-level fault injection, system-on-chip (*SoC*), transient fault (soft error or SEU).

SoC-Level Fault Injection Methodology and Tool Development in SystemC Design Platform

Abstract — **Intelligent systems, such as intelligent car driving system or intelligent robot, require a stringent reliability while the systems are in operation. As system-on-chip (SoC) becomes prevalent in the intelligent system applications, the reliability issue of SoC is getting more attention in the design industry while the SoC fabrication enters the very deep submicron technology. In this study, we present a new approach of system bus fault injection in SystemC design platform, which can be used to assist us in performing the failure mode and effects analysis (FMEA) procedure during the SoC design phase. We demonstrate the feasibility of the proposed fault injection mechanism with an experimental ARM-based system.**

Index Terms: FMEA, reliability, system-on-chip, SystemC, system bus fault injection.

I. INTRODUCTION

As *SoC* becomes more and more complicated, the *SoC* could encounter the reliability problem due to the increased likelihood of faults or radiation-induced soft errors especially when the chip fabrication enters the very deep submicron technology [1]-[3]. Thus, it is essential to perform the FMEA procedure to locate the weaknesses of the system and provide the practical fault-tolerant strategies to improve the reliability [4]. However, due to the high complexity of the *SoC*, the incorporation of the FMEA procedure and fault-tolerant demand into the *SoC* will further raise the design complexity. Therefore, we need to adopt the behavioral level or higher level of abstraction to describe/model the *SoC*, such as using SystemC, to tackle the complexity of the *SoC* design and verification. An important issue in the design of *SoC* is how to validate the system reliability as early in the development phase to reduce the re-design cost. As a result, a system-level dependability verification platform is required to facilitate the designers in assessing the dependability of a system with an efficient manner. Normally, the fault injection approach is employed to verify the robustness of the systems.

Most of the previous fault injection studies focus on the VHDL design platform, whereas only a few works [5]-[9] address the fault injection issue in SystemC design platform. In our previous paper [7], we proposed a fault injection methodology for cycle-accurate register-transfer level (RTL) and compared the results of injection campaigns with the outcomes derived from the VHDL RTL. In [5], [6], the authors proposed a fault injection framework that is applicable to functional level and transaction layer 1 in SystemC. The paper [9] characterized the susceptibility of AMBA bus on errors in various signals over different transactions in SystemC cycle-accurate level.

As we know, the system bus, such as AMBA AHB, provides an integrated platform for IP-based *SoC*. Apparently, the robustness of system bus plays an important role in the *SoC* reliability. So, performing the system bus FMEA is imperative to validate the reliability of *SoC*. In previous related work, the issue of system bus fault injection in SystemC design platform is rarely addressed except the work proposed in paper [9]. However, the approach presented in [9] is dedicated to cycle-accurate level, which may still be time-consuming in fault injection and simulation runs. In addition, the previous fault injection methodologies are all based on time-driven approach to decide when to inject a fault. While the modeling levels of systems come to the untimed functional transaction-level modeling (TLM) and timed functional TLM, the time-driven fault injection approach is no longer applicable to these levels or becomes improper. Instead, the event-driven fault injection approach is effective in keeping the fault injection easier and efficient at untimed/timed functional TLM, especially in the performing of system bus FMEA.

The types of transaction in the bus normally consist of the single-read, single-write, burst-read and burst-write operations. Each type of bus transaction can represent a possible failure mode of the system bus. If we want to analyze the effect of a specific system bus failure mode, like burst-read failure, on the system behavior, the event of fault triggers in this case can be set as burst-read operation of bus transactions. In other words, the time instant of fault injection is related to the occurrence of the burst-read event in the bus transactions. It is clear that using the event-driven fault injection can easily produce the desired failure mode and effectively characterize its effect on the system functionality. Compared to event-driven fault trigger, the time-driven approach suffers from the poor injection effectiveness for a specific failure mode and its effect analysis, because the time-driven fault trigger cannot guarantee the injected faults that will cause the desired failure mode, such as burst-read failures.

The principal goal of this work is to propose an effective system bus event-driven fault injection framework in SystemC design platform at the abstraction levels of untimed/timed functional TLM to assist the reliability assessment. The advantages of our fault injection approach are two folds: one for simulation speed and the other for injection effectiveness. The remaining report is organized as follows. In Section II, the SystemC untimed/timed functional TLM and the concept of Transactor are presented. We propose a system bus fault injection methodology in Section III. A fault injection tool is demonstrated in the following section. We show some experimental results in Section V. The conclusions and future work appear in Section VI.

II. SYSTEMC UNTIMED/TIMED FUNCTIONAL TLM

SystemC, a system-level modeling language, provides a wide variety of modeling levels of abstraction and allows us to model a system utilizing one or a mixture of various abstraction levels. It is quite common that the modules within a *SoC* are modeled at different levels of abstraction using SystemC design language. The primary goal of TLM is to reduce the modeling complexity and increase the simulation speeds, while offering enough accuracy for the design task. The Open SystemC Initiative (OSCI) categorizes the TLM in SystemC into the following levels: Programmers View (PV), Programmers View with Timing (PV+T) and Cycle Callable (CC), where the modeling level of abstraction and simulation speed is from high to low among these three levels. The PV level is equivalent to untimed functional TLM and PV+T level is the level of timed functional TLM.

We adopt the CoWare Platform Architect [10] and AMBA bus [11] to demonstrate our system bus fault injection approach and its applications. The Platform

Architect provides the modeling levels of PV and PV+T and allows the mixture of these two levels in the IP-based *SoC* design. In the following, we address the issue of system bus fault injection in PV and PV+T levels, which can be used to assist us in performing the FMEA procedure during the *SoC* design phase. Fig. 1 shows the ARM-based systems modeled with the mixed abstraction levels of PV and PV+T, where the 'Transactor' likes bridge to connect the PV and PV+T levels and its function is to convert the bus protocols between PV and PV+T levels. In Fig. 1, the AHB and APB components are modeled at PV+T abstraction level with AMBA protocol; whereas the 'IP' slave modules are modeled at PV level with PV protocol. The PV bus can be utilized to connect the slave modules as shown in Fig. 1(a) and (c). Then, the 'Transactor' behaves like bridge between PV bus and AHB or APB. Fig. 1(b) and (d) do not use the PV bus for slave modules. Instead, each slave module connects to the AHB or APB through the 'Transactor'. The reason of employing the PV modeling level is to speed up both the modeling process itself as well as the simulation of the resulting specification.

The AMBA library of Platform Architect provides three kinds of 'Transactor' module, which are named as AHBLiteTarget_PV, APBTarget_PV and ScmlPost_AHBInitiator. The former two types of 'Transactor' offer the bridge between slave modules modeled at PV level and AHB/APB modeled at PV+T level; ScmlPost_AHBInitiator connects the master modules modeled at PV level to AHB modeled at PV+T level. In general, an AMBA-based *SoC* contains multiple masters and slaves. So, without loss of generality, we exploit the system platform as illustrated in Fig. 2, which combines the Fig. 1(a) and (c) to demonstrate our system bus fault injection methodology. The injection mechanism for systems as shown in Fig. 1(b) and (d) is similar to the one applicable to Fig. 2 system. Therefore, we omit its details.

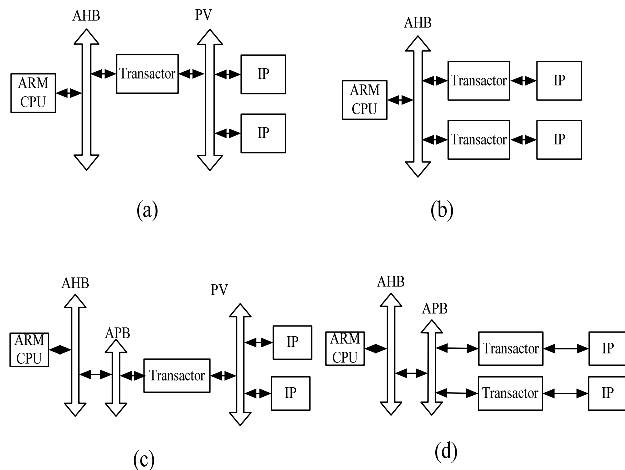

Fig. 1. ARM-based system modeled with mixed levels of PV and PV+T, where IP represents the slave module.

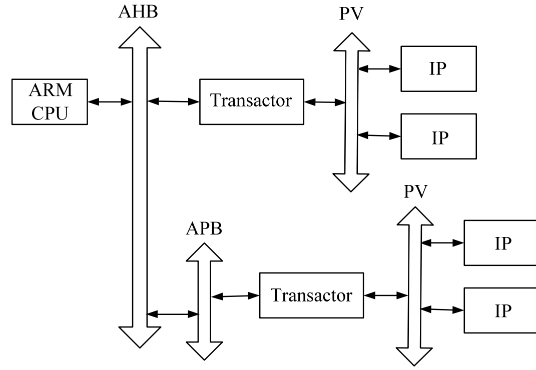

Fig. 2. An AMBA-based system modeled at PV and PV+T levels.

III. SYSTEM BUS FAULT INJECTION SCHEME

An AMBA-based system as illustrated in Fig. 2 is exploited to demonstrate our system bus fault injection methodology. The event-driven fault trigger is utilized to inject the faults into the system bus during the data transactions. An event is used to represent a particular condition that decides the time of fault occurrence. The principal idea of our approach is based on the insertion of a fault injection module (FIM) into the bus interconnection, where the FIM is to control the fault injection activity. The function of FIM is first to monitor the bus and collect the bus transaction information including address, data and control signals; then, check whether the declared event occurs; if yes, the fault is injected into the bus.

A. *FIM generation flow*

The flow of FIM generation consists of two phases and is described as follows:

Phase 1: Since FIM employs the event-driven fault trigger approach, we need to collect the bus transaction information including address, data and control signals into the operational profiles during the program execution. The operational profiles are used as a reference for generation of FIM events that will be the conditions of fault trigger. The function of 'Transactor' as shown in Fig. 2 and 3(a) can be enhanced by adding the ability of bus transaction information collection to 'Transactor'. This modified version of 'Transactor' is called operational profile module (OPM) as displayed in Fig. 3(b). We utilize the AMBA bus API [12] furnished by CoWare Platform Architect to implement the function of operational profile generation. What kinds of bus transaction information should be collected all depends on the designer need. The following pseudo code is used to exhibit the address information collection for bus-read transaction. We note that AHB bus allows multiple masters, and therefore we offer the OPM to have the capability to collect the bus transaction information for each master. The other classes of bus transaction

information, such as data and protocol signals, can be achieved in the similar way. The information for each class of bus transaction is gathered to a profile for each master during the fault-free simulation campaign.

```
While (1) {
   int Master_ID;          //multiple masters, each given an
                           unique number;
   fstream profile_master_1("Data1.txt",ios::app);
   //gathering address information of master 1 to a
  profile;
  fstream profile_master_2("Data2.txt",ios::app);
  // gathering address information of master 2 to a
  profile;
  port.getReadDataTrf()  //check whether the bus is
                          performing read transaction or
                          not
     Master_ID =port.getMasterId()      //which master is
                                        using bus;
     If (Master_ID == 1){
        profile_master_1<< port.getAddress()<<endl;
        //acquire the address and write it to the operational
        profile of master 1 ;}
     If (Master_ID == 2){
       profile_master_2<< port.getAddress()<<endl;
       //acquire the address and write it to the operational
        profile of master 2 ;}
      Protocol transformation;       //original function of
                                     Transactor
      send Transfer();    //call function of set/get read
data;
      profile_master_1.close();
      profile_master_2.close();
     wait();
}
```

Next, we discuss the relationship between event and operational profile. As mentioned before, the operational profiles are utilized to help us creating the desired events that decide the time instant of fault injection. Fig. 4 exhibits the event tree and its possible combinations. Basically, the types of transaction in the bus are 'read' and 'write' that are the first level of events as shown in Fig. 4. The second level of events includes 'burst' and 'single' data transactions. The third level of events consists of 'data', 'address' and 'burst length' etc. Our event-driven fault trigger methodology provides diverse sorts of events. The types of event can be a single event or the combination of events as illustrated in Fig. 4. We use Fig. 4 to explain the event concept. For bus transaction, single event could be either 'read' bus transaction or 'write' bus transaction, which pertain to the first level of events. The event combinations can be formed either from first-level events with second-level events or from first, second then third-level events in sequence. To support the event-driven fault trigger, the function of OPM needs to create the desired operational profiles, which furnish the information for event formation.

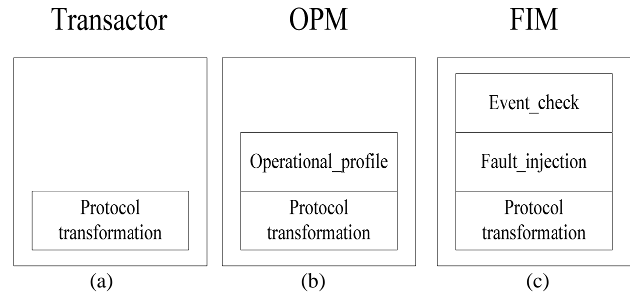

Fig. 3. 'Transactor', OPM and FIM functions.

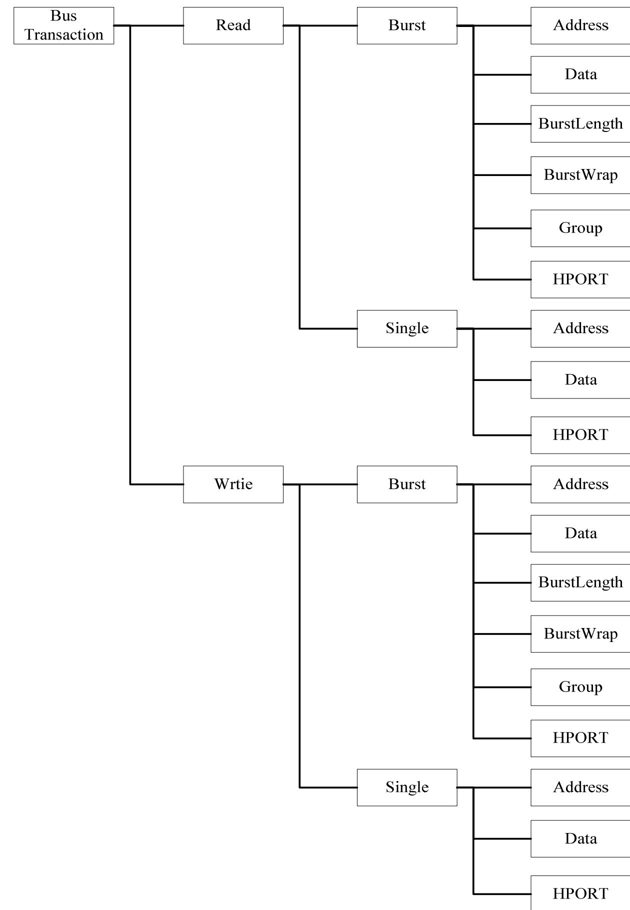

Fig. 4. Event tree and combinations.

We give an example of event combination and its application below:

Example 1: From Fig. 4, we can construct an event combination using first-level events coupled with second-level events, like 'read' associated with 'burst' to compose a new sort of event, termed as burst-read event. If we treat the bus as a component, the failure mode of the bus could be 'single-read', 'single-write', 'burst-read' and 'burst-write' failures. The burst-read failure means that the bus failure occurs during the burst-read transactions.

Therefore, if we want to perform the effect analysis of burst-read failure mode on the system behavior, the burst-read event can be utilized to guide the fault injection to guarantee the faults injected, which always lead to the burst-read failures. We note that the event-driven fault trigger approach offers a highly effective injection operation, which is very suitable for the FMEA needs. Assume that there are ten thousand times of burst-read transactions occurring in the bus of an experimental *SoC* platform. To obtain the effect analysis of burst-read failures on system operation, we need to conduct a huge amount of fault injection campaigns, saying five hundred campaigns. Each campaign injects a fault that is triggered when the number of burst-read transactions appearing in the bus is equal to x, where x is from 1 to 10000. The number of x for each injection campaign is decided by randomly choosing a number between 1 and 10000. For instance, the number of x for an injection campaign is 100. In this case, the fault is triggered while the number of burst-read transactions appearing in the bus is equal to 100. The OPM in this case needs to produce an operational profile that collects the information of burst-read bus transactions including the total number of burst-read transactions and the details of each burst read transaction, such as the length of burst read.

We should point out that the OPM is developed to gather the bus transaction information for a particular event. So, we need to decide the adopted event for fault triggering condition, and develop the corresponding OPM for operational profile production of the adopted event. As discussed before, the event formation can be either a single event or event combination. The more number of events are combined, the more control of fault trigger will be.

Phase 2: Based on the operational profiles produced in Phase 1, the FIM as illustrated in Fig. 3(c) is generated for each injection campaign. A FIM can be constructed from the 'Transactor' and its function contains four parts: bus monitoring, event check, fault injection and protocol transformation. The FIM replaces the 'Transactor' as shown in Fig. 2, and is responsible for event check to determine when the fault should be injected. If the event check finds the particular event happens, the fault is injected. The following pseudo code is employed to demonstrate how to implement the event check and fault injection operations in the FIM. The function of FIM in this demonstration is to inject a fault when the bus is in read transaction and a specific address occurs. Again, we utilize the AMBA bus API [11] furnished by CoWare Platform Architect to implement the function of FIM.

```
While (1) {
  int Master_ID;
  port.getReadDataTrf();   // check whether the bus is
                              performing read transaction
                              or not;
  Master_ID =port.getMasterId();   // which master is
                                      using bus
  If (Master_ID == 1){
    If (port.getAddress()==0x40000000)
      {Fault_injection}}
  //master 1 is the bus owner and address is 0x40000000.
  If (Master_ID == 2){
    If (port.getAddress()==0x80000000)
      {Fault_injection}}
  //master 2 is the bus owner and address is 0x80000000
  Protocol transformation;
  send Transfer();
  wait();
  }
```

IV. FAULT INJECTION TOOL

In this section, we present a fault injection tool based on the system bus fault injection methodology described in last section and the fault injection scheme for communication channels at the following abstraction levels: sc_signal at bus-cycle-accurate (BCA) level and the primitive channel sc_fifo at untimed functional TLM [8]. Fig. 5 exhibits the operational flow of fault injection tool. We construct this tool under the environment of CoWare Platform Architect. The tool deals with fault injection at different modeling levels of abstraction and the fault trigger can be time-driven or event-driven approaches. The proposed fault injection tool can significantly reduce the effort and time for performing the fault injection campaigns. In addition to that, the tool dramatically increases the efficiency of carrying out the FMEA and system robustness validation. In the following, we briefly depict the tool main functions:

- Automatically generate the OPM that replaces the 'Transactor' as shown in Fig. 2 to establish the *SoC* platform, which is used to produce the desired operational profiles for each master.
- According to the operational profiles, choose the event for fault triggering condition; automatically generate the FIMs that replace the 'Transactors' to establish the targeted *SoC* platform for fault injection campaign usage.

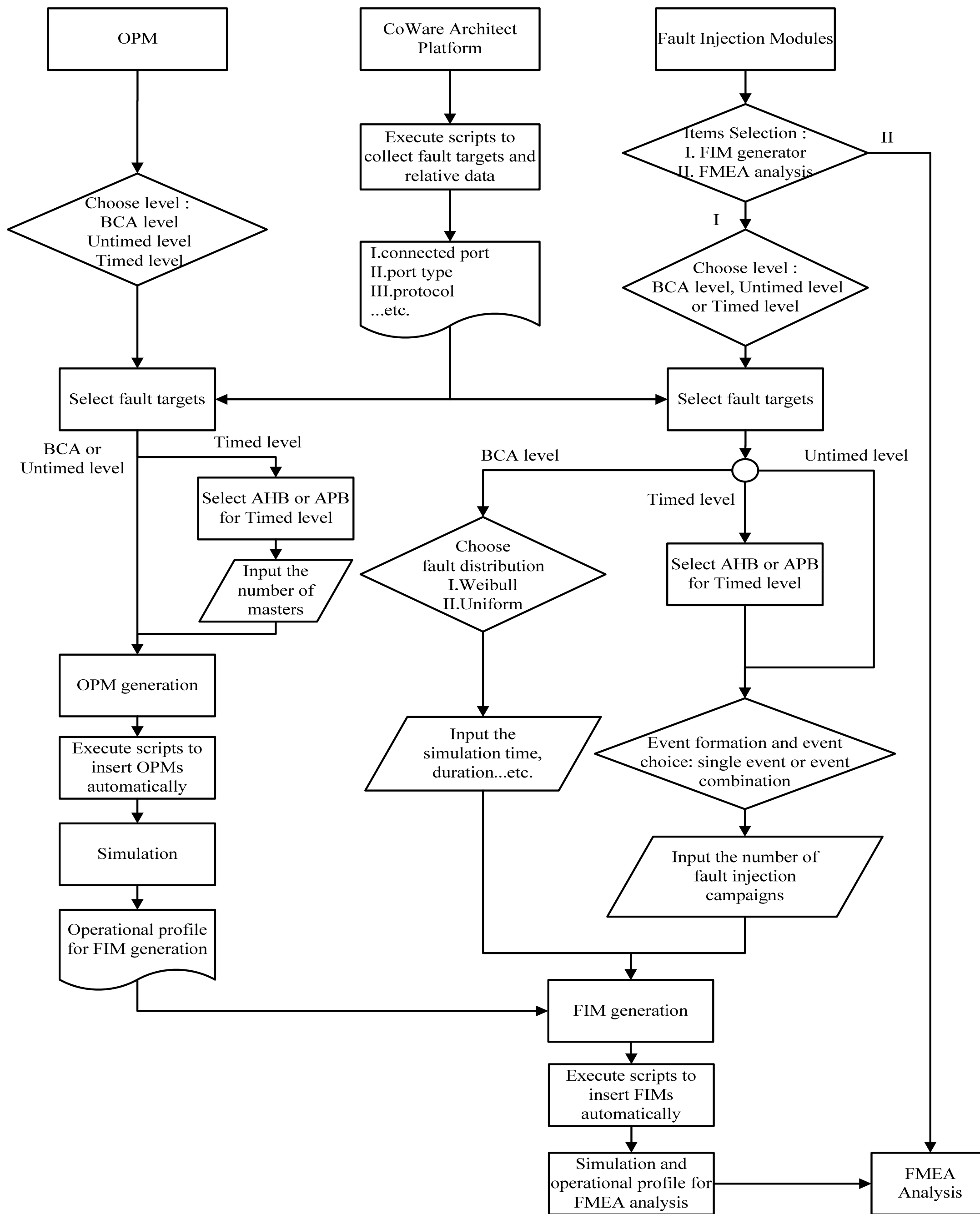

Fig. 5. The operational flow of fault injection tool.

V. EXPERIMENTAL RESULTS

The following experimental studies were performed to validate the feasibility of our fault injection framework and tool proposed in Sections 3 and 4. Fig. 6 shows an ARM-based system used in fault injection experiments. We use the AMBA bus library [11] provided by CoWare Platform Architect to implement the system as illustrated in Fig. 6. The hexadecimal numbers followed the IP modules are memory mapping addresses, which could be the targets of fault injection. The purposes of peripherals: disp_1, disp_2 and disp_3 are for displaying the final results to easily verify our experimental results.

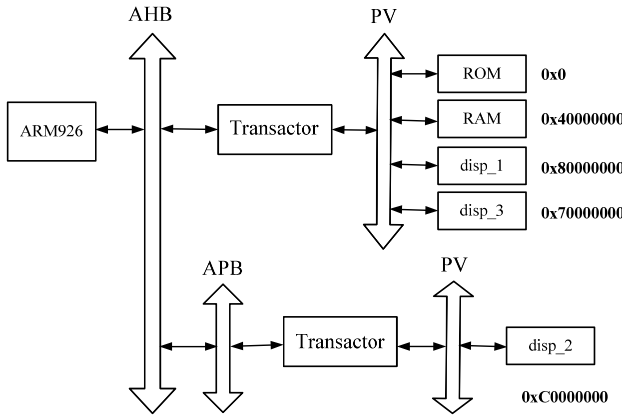

Fig. 6. A system platform for experimental demonstration.

A. Tool Validation Experiment

The goal of this experiment is to verify the functional correctness of the operational flow of fault injection tool. We conducted two event-driven injection campaigns: one for single-event-driven fault trigger and the other for combined-event-driven fault trigger.

Single-event-driven fault trigger: In this experiment, we choose 'bus address' as the fault-triggering event. In other words, the event check is to examine whether the address for the current bus transaction reaches a particular address that is set as the fault-triggering condition or not. We first employ the fault injection tool to generate the OPM and perform the simulation to obtain the desired operational profile, which collects the bus address information. According to operational profile derived from the fault-free simulation, we choose a specific address '0x80000000' to be the fault-triggering condition. It means that when the address '0x80000000' happens, the event check of FIM is positive and a wrong address '0x70000000' is injected into the address bus. Fig. 7 illustrates the experimental results, where the left and right parts show correct and wrong results, respectively. The results indicate that originally the ARM CPU writes a data to disp_1 (address: 0x80000000) to display. Due to fault interference, the data is written to the wrong address '0x70000000' (disp_3).

Combined-event-driven fault trigger: The event formation comprises the events of 'write', 'burst', 'burst length' and 'group' of bus transaction. The burst length represents the number of data transfer for each burst transaction and group means the type of data transfer. The fault-triggering condition is set as 'burst write, burst length = 8 and group = 1'. The FIM checks the combined event during the bus transactions to decide when to inject a fault. Fig. 8 shows the experimental results, where a fault is injected into the data bus while the particular event occurs.

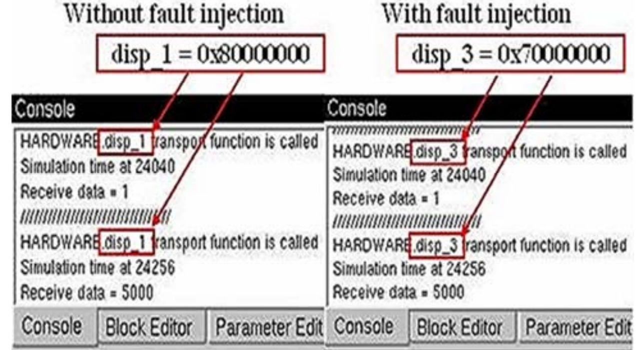

Fig. 7. Single-event-driven fault injection.

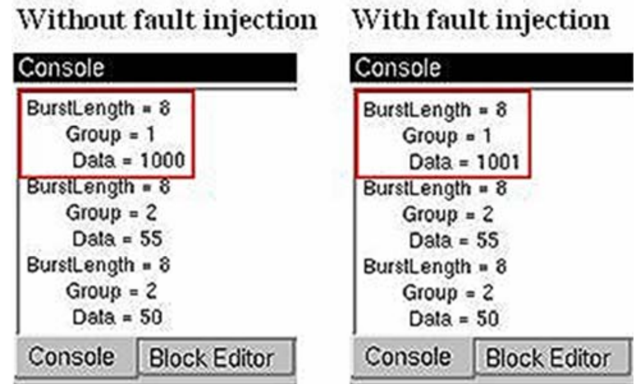

Fig. 8. Combined-event-driven fault injection.

B. Time-Driven and Event-Driven Injection Experiment

In this experiment, the effectiveness of the event-driven fault injection methodology is compared with time-driven one. The mission of this experiment is to analyze the effect of system bus failure mode, such as 'burst-read' failures, on the system behavior. To meet the mission requirement, we need to conduct a huge number of fault injection campaigns to guarantee the validity of the FMEA results. We note that the time-driven fault injection is inefficient for this kind of mission because the time instants of injected faults derived from the time-driven approach are often not the time of the bus performing the burst-read transactions. Therefore, we will waste enormous time and effort if the time-driven method is adopted. In contrast to time-driven method, the event-driven approach can accomplish the mission discussed very effectively because the event check in FIM is 'burst-read' event, and therefore the faults injected always guarantee to cause the 'burst-read' failures. Two benchmark programs *50×50 matrix multiplication* and quicksort (sorting 99 elements) were developed and used in the fault injection campaigns. Table I gives the comparison results for the time-driven and event-driven fault injection campaigns. The term of 'Burst-Read Count' in

Table I means the total number of 'burst-read' bus transactions occurs during the benchmark execution. The 'hitting rate of fault injection' represents the probability of faults injected that cause the 'burst-read' failures. The results are derived from two thousand times of fault injections. It is evident that the hitting rate of fault injection for event-driven method is 100%, whereas time-driven method only has 5.25% and 6.75% for matrix multiplication and quicksort, respectively. The experimental results indicate that the event-driven fault injection approach is very suitable for the FMEA applications.

TABLE I
THE COMPARISON RESULTS FOR TIME-DRIVEN AND EVENT-DRIVEN FAULT INJECTION APPROACHES

Benchmark	Matrix	Quicksort
Clock cycle	4370925	42703
Burst-Read Count	421990	5443
Hitting rate of fault injection (Event-driven)	100%	100%
Hitting rate of fault injection (Time-driven)	5.25%	6.75%

C. FMEA Experiment

This experiment is a demonstration of employing the developed tool to perform the AMBA bus FMEA. The goal of this demonstration is to show the capability of the proposed fault injection methodology and tool. We use '*50 ×50 matrix multiplication*' and '*quicksort*' as our benchmark programs and choose 'burst-read' as the targeted failure mode. Therefore, the fault-triggering event is set as 'burst-read' bus transaction. The fault targets include 'address', 'data' and 'access size' signals. For each injection campaign, a single-bit-flip fault was injected into the fault targets of AMBA bus on the happening of a specific burst-read transaction. The fault duration sustains the length of one bus transaction. This experiment is to investigate the effect of 'burst-read' failure mode of AMBA bus on the system behaviors. There are five classes of system outcomes [9]: fatal error (FE), silent data corruption (SDC), correct data/incorrect time (CD/IT), deadlock (DL) and no effect (NE) appeared in the injection campaigns. The system crash or process hang are classified into fatal error; silent data corruption is caused by the errors that remain unnoticed until the end of the simulation but provide incorrect results; one kind of errors which won't affect the correctness of final results but changes the program execution time is classified into correct data/incorrect time; deadlock is the errors that lead the system to get into no progress states; no effect is the errors which have no impact on the system operation at all.

The results of each row listed in Table II were derived from one hundred injection campaigns respectively. From the results of Table II, we can see that for example an AHB address fault occurring during 'burst-read' bus transaction results in 21% FE, 44% SDC, 6% CD/IT, 1% DL and 28% NE for matrix multiplication benchmark. The meaning of data for other rows in Table II is similar to the row of HADDR fault. Table III generated from the data of Table II illustrates the effect of 'burst-read' failures on the system operation, i.e. the occurring probability of FE, SDC, CD/IT, DL and no effect when the system encounters the 'burst-read' failures. We note that the FMEA results are program-variant as evidenced in Table III. It is clear that the system is more sensitive to errors while system executes the matrix multiplication program. The preliminary results obtained indicate that the attributes of benchmarks have a significant impact on the FMEA results.

From the demonstration of this experiment, our proposed fault injection methodology and tool for assisting the performing of FMEA can be validated. In the near future, we will further provide more complete FMEA results with more benchmarks and more fault-targeted signals in AMBA bus. The effect of attributes of benchmarks on FMEA results will be discussed in depth. We will also investigate how to enhance the efficiency of performing FMEA procedure by conducting the fault injection (component's failure mode production) with different combination of events.

TABLE II
FMEA DATA FOR 'BURST-READ' FAILURE MODE. (a) MATRIX MULTIPLICATION (b) QUICKSORT

System failure	FE	SDC	CD/IT	DL	NE
HADDR fault	21%	44%	6%	1%	28%
HRDATA fault	3%	45%	0%	1%	51%
HSIZE fault	3%	43%	28%	1%	25%

(a)

System failure	FE	SDC	CD/IT	DL	NE
HADDR fault	19%	12%	12%	1%	56%
HRDATA fault	7%	17%	13%	2%	61%
HSIZE fault	7%	21%	69%	0%	3%

(b)

TABLE III
THE PROBABILITY OF FE, SDC, CD/IT, DL AND NE FOR VARIOUS BENCHMARKS

	FE	SDC	CD/IT	DL	NE
Matrix	9%	44%	11.3%	1%	34.7%
Quicksort	11%	16.7%	31.3%	1%	40%
Matrix + Quicksort	10%	30.4%	21.3%	1%	37.3%

VI. CONCLUSIONS AND FUTURE WORKS

In this work, a system-bus fault injection methodology in SystemC design platform is presented, and a fault injection tool is developed for performing the FMEA of *SoC* platform. The main contributions of this study are to raise the level of fault injection to the untimed/timed functional TLM and devise an effective

event-driven fault-triggering method. The proposed event-driven method offers diverse event formation and efficient injection capability for FMEA applications. Our fault injection tool can dramatically increase the efficiency of carrying out the FMEA and system robustness validation. Several experiments based on CoWare Architect Platform were conducted to validate the feasibility of our fault injection approach and tool ability. We will conduct more fault injection campaigns for FMEA experiments with more benchmarks and carry out a thorough analysis of the effect of various failure modes, like 'burst-write' failure, on the system behavior.

REFERENCES

[1] C. Constantinescu, "Impact of deep submicron technology on dependability of VLSI circuits," in *2002 Proc. IEEE Int. Conf. On Dependable Systems and Networks (DSN)*, pp. 205-209.

[2] R. Baumann, "Soft errors in advanced computer systems," *IEEE Design & Test of Computers*, vol. 22, issue 3, pp. 258 – 266, May-June 2005.

[3] Y. Zorian, V. A. Vardanian, K. Aleksanyan, and K. Amirkhanyan, "Impact of soft error challenge on *SoC* design," in *2005 Proc. 11th IEEE Int. On-Line Testing Symposium,* pp. 63 – 68.

[4] R. Mariani, G. Boschi, and F. Colucci, "Using an innovative *SoC*-level FMEA methodology to design in compliance with IEC61508," in *2007 Proc. Design, Automation & Test in Europe Conf. & Exhibition*, pp. 492-497.

[5] K. Rothbart, U. Neffe, Ch. Steger, R. Weiss, E. Rieger, and A. Muehlberger, "High level fault injection for attack simulation in smart cards," in *2004 Proc. 13th Asian Test Symposium*, pp. 118-121.

[6] K. Rothbart, U. Neffe, Ch. Steger, R. Weiss, E. Rieger, and A. Mühlberger, "A smart card test environment using multi-level fault injection in SystemC", in *2005 Proc. 6th IEEE Latin-American Test Workshop*, pp. 103-108.

[7] K. L. Leu, Y. Y. Chen, and J. E. Chen, "A comparison of fault injection experiments under different verification environments," in *2007 Proc. IEEE 4th Int. Conf. on Information Technology and Applications*, pp. 582-587.

[8] K. J. Chang, and Y. Y. Chen, "System-level fault injection in SystemC design platform," in *2007 Proc. 8th Int. Symposium on Advanced Intelligent Systems*, pp. 354-359.

[9] I. C. Lin, S. Srinivasan, and N. Vijaykrishnan, "Transaction level error susceptibility model for bus based *SoC* architectures," in *2006 Proc. 7th Int. Symposium on Quality Electronic Design,* pp. 775-780.

[10] CoWare Model Library, "Platform Creator User's Guide," Product Version V2006.1.2.

[11] CoWare Model Library, "AMBA Bus Library," Product Version V2006.1.2.

[12] CoWare Model Library, "SystemC Modeling Library Manual," Product Version V2006.1.2.

An Estimation Model of Vulnerability for Embedded Microprocessors

Abstract — *Embedded systems, and also the embedded microprocessors, have encountered the reliability challenge because the occurring probability of soft errors has a rising trend. When they are applied to safety-critical applications, designs with the fault tolerant consideration are required. For the complicated embedded systems or IP-based system-on-chip (SoC), it is unpractical and not cost-effective to protect the entire system or SoC. Analyzing the vulnerability of systems can help designers not only invest limited resource on the most crucial region but also understand the gain derived from the investment. In this study we propose a model to fast estimate the microprocessor's vulnerability with only slight simulation effort. From our assessment results, the rank of component vulnerability related to the probability of causing the microprocessor failure can be acquired. By choosing one of the mainstream microprocessors — VLIW (Very Long Instruction Word) processor — as an example, the practical usefulness of our estimation model is demonstrated.*

1. Introduction

Soft errors caused by SEU (Single Event Upset) have more and more apparent influence upon electronic products including the microprocessors [1]. Some techniques, like the structural duplication or ECC (Error Correcting Code), can be utilized to improve the microprocessor's vulnerability to soft errors. These techniques are effective but also cost-consumed. Performing the FMEA procedure is imperative to validate the reliability of systems and to acquire the vulnerability of the systems. Based on the results of FMEA, we can achieve a more effective investment of the precious resources to the system to enhance the reliability in an efficient manner. Therefore, the effect of soft errors on the system behaviors/failures needs to be analyzed and modeled.

Many literatures such as [2-4] have proposed their own estimating methodologies to derive the microprocessor's vulnerability. However, these modeling procedures are complicated and maybe difficult to be adopted by designers. In this work, we want to propose a simple and effective model which can provide designers knowledge about: (i) how vulnerable

a microprocessor is and (ii) rank of components by their contribution to the microprocessor's vulnerability. Thus this model must be able to sincerely reflect the influence of each component's malfunction to whole microprocessor. In the following section, this model is proposed, and then in Section 3 some experimental results for a VLIW processor are provided and discussed.

2. Estimation Model for Microprocessor's Vulnerability

For simplifying the estimation model, two hypotheses are given below:

(a). The circumstances of unexpected transmissions are regarded as equivalent to components' malfunction and therefore our estimation model only takes the components into account. Thus components which are assumed to be critical must be collected into a set called *component set* $\{C_1, C_2, ..., C_n\}$; other non-critical components are viewed as error-free. If designers do better at this step, then our model have more precise results.

(b). The single-fault assumption is adopted.

In this work, the vulnerability of a microprocessor, V_{MP}, is defined as the probability that an inner error ultimately results in a failure of this microprocessor. We can use V_{MP} to model the entire process from a component's error to whole microprocessor's failure. (A similar estimation model can be found in [6] but it is for the AMBA architecture.) For this end, we defined three parameters: AR_{Ci}, U_{Ci} and SES_{Ci} for each component C_i in *component set* — where i is from 1 to n. AR_{Ci} is the *area ratio* of C_i, U_{Ci} is the frequency of activating C_i, also named C_i's *utilization ratio* and SES_{Ci} — termed C_i's *soft error sensitivity* — defined as the probability of system failure caused by an error emerged form C_i and the error has effectively propagated to the system. Their values can be derived by equations (1) ~ (3):

$$AR_{C_i} = \frac{area\ of\ component\ C_i}{\sum_{i=1}^{n} area\ of\ component\ C_i} \quad (1)$$

$$U_{C_i} = \frac{execution\ cycles\ using\ component\ C_i}{execution\ cycles\ of\ benchmark\ programs} \quad (2)$$

$$SES_{C_i} = \frac{errors\ cause\ the\ microprocessor\ failed}{errors\ emerged\ from\ component\ C_i\ and\ effectively\ propagated} \quad (3)$$

Firstly, the area of a component will dominate the probability that particles strike it, so AR_{Ci} can be used to represent the error rate of C_i. Then, assumed the striking triggers an error emerged on C_i, this error will still be ineffective if C_i is not used presently, thus U_{Ci} is required to model the probability of error propagation. Moreover, even an error has been propagated, it may not cause the microprocessor failed. For example, an error which propagates to the *branch predicting unit* won't have influence on consequences of benchmark programs. Therefore how this error propagates is the last factor which needs to be considered and modeled by SES_{Ci}. Both of U_{Ci} and SES_{Ci} are very relevant to the characteristic of benchmark programs as observed from Table 1. To identify the component which is most vulnerable, we define another parameter V_{Ci} which is the product of three factors above. Finally, we can compute the microprocessor's vulnerability by the summation of V_{Ci} as shown in the equation (4):

$$V_{MP} = \sum_{i=1}^{n} V_{C_i}, \text{ where } V_{C_i} = AR_{C_i} \times U_{C_i} \times SES_{C_i} \quad (4)$$

3. Experimental Results

To derive SES_{Ci}, we have conducted a set of error injection simulations on the RTL model of the VLIW processor proposed in [5] (but without involving the fault tolerant components in the execution stage). For each component in the *component set* — assigned as {ALU_A, ALU_B, ALU_C, control unit, forwarding unit, branch unit, instruction dispatch unit} — 2000 simulations are performed and the processor is monitored to gather the statistic data. Besides, an extra simulation without error injection is performed to derive the U_{Ci}. It is worthy to note that the storages such as the instruction cache and data cache are not put into this set. The reason is that their vulnerability has been finely investigated in many literatures (e.g. [2-4]), so we place our interests on the other components. However, our estimation model is still applicable to storage components. Experimental results shown in Table 1 are preliminary with two benchmark programs — matrix multiplication and IDCT.

According to Table 1, the V_{MP} of VLIW is 0.4658. From Table 1, not only the most vulnerable component can be identified by the rank, but also the effect of the three factors on the vulnerability can be analyzed. It is very interesting that ALU_B, ALU_C and branch unit are first three components with highest soft error sensitivity but their rank of V_{Ci} is totally inversed. On the contrary, the forwarding unit has smallest SES_{Ci} but largest V_{Ci}. This dramatic reverse proves that none of AR_{Ci}, U_{Ci} and SES_{Ci} can be neglected. This is a first confirmation of our estimation model for helping us precisely identify the vulnerable components.

Acknowledgements. The authors acknowledge the support of the National Science Council, Republic of China, under Contract No. NSC 96-2221-E-216-006.

References

[1] F. Irom et al, "Single-Event Upset in Evolving Commercial Silicon-on-Insulator Microprocessor Technologies," *IEEE Trans. on Nuclear Science*, Vol. 50, No. 6, pp. 2107-2112, December 2003.

[2] S. Kim and A. K. Somani, "Soft error sensitivity characterization for microprocessor dependability

enhancement strategy," *DSN'02*, pp. 416-425, June 2002.

[3] S. S. Mukherjee et al, "A Systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor", *36th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 29-40, 2003.

[4] S. Wang, J. Hu, and S. G. Ziavras, "On the Characterization of Data Cache Vulnerability in High-Performance Embedded Microprocessors", *IC-SAMOS,* pp. 14-20, 2006.

[5] Y. Y. Chen, K. L. Leu and C. S. Yeh, "Fault-Tolerant VLIW Processor Design and Error Coverage Analysis," *The 2006 IFIP International Conference on Embedded and Ubiquitous Computing*, pp. 754-765. (Springer: LNCS 4096).

[6] I.C. Lin et al, "Transaction level error susceptibility model for bus based SoC architectures", *7th International Symposium on Quality Electronic Design*, pp. 6, 2006.

Table 1. Experimental results for two benchmark programs

Component \ parameter	AR_{Ci}	U_{Ci}	SES_{Ci}	V_{Ci}	Rank by V_{Ci}
Alu_A	0.092	0.849	0.697	0.0550	4
Alu_B	0.092	0.271	0.861	0.0217	5
Alu_C	0.092	0.178	0.881	0.0146	6
Control unit	0.149	0.895	0.579	0.0772	3
Forwarding unit	0.368	0.876	0.483	0.1557	1
Branch unit	0.017	0.019	0.952	0.0003	7

Self-Evaluation of Research Results:

- The above report summarizes the second-year results accomplished from this three-year research project. The extended versions of the results will be submitted to be considered for journal publication. In the first-year report, we stated: "We are going to develop a system-level fault-injection tool, which exploits the simulation-based fault injection scheme proposed in this research and can be installed in the CoWare Architect Platform. The tool takes the fault scenario description from the user and then automatically generates the system platform supplemented with the fault injection capability. This kind of fault injection tool can not only facilitate the failure mode and effects analysis (FMEA) and the fault-tolerant validation process, but raise the validation efficiency. The embedded fault-tolerant systems have found fertile ground in intelligent system applications, such as intelligent driver assistance system or intelligent robot system, which require a stringent dependability while the systems are in operation. Since more works depend on the intelligent machines, the reliability issue becomes more important than ever. The fault-tolerant verification platform developed from this research can be applied to the design and analysis of the fault-tolerant systems modeled at high level of abstraction to enhance the overall system dependability. The previous study for the fault injection approach mainly focuses on the VHDL modeling level and rarely discusses the fault injection in SystemC system-level design. We want to fulfill this lack." The above statements describe the main research goal of this three-year project. From this report, it is evident that we definitely achieve the second-year goal with one extra supplement, i.e. software-implemented fault injection (SWIFI) technique. The SWIFI approach can enhance the injection capability and diversity for dependable IP-based *SoC* design platform.
- However, the subjects expressed in our proposal are big and deserve to be further explored. The ongoing works are depicted as follows: First, we will port the Linux O.S. to CoWare Architect Platform and then the SWIFI technique will be added to the fault injection tool as well. As a result, our fault injection tool will provide more comprehensive injection functions. Second, we are going to develop a useful analysis tool with some analysis functions for validation of system robustness and dependability. Thirdly, we are building a more complex system using CoWare Architect Platform to completely study the effects of faults on system behaviors, and to detect the weaknesses in the reliability of the system. Finally, a fault-robust IP approach will be proposed to improve the system dependability by using the results of FMEA and safety-critical component's analysis. The contribution of this research is to construct a complete and comprehensive dependability validation framework that consists of the system-level fault injection to study the system's failure behavior, and to perform the FMEA procedure to locate the weaknesses in the reliability of the system, and to exploit the fault-tolerant design to effectively enhance the system dependability.

Publications associated with the second-year research:

- Kun-Chun Chang, Yi-Chinag Wang, Chung-Hsien Hsu, Kuen-Long Leu and Yung-Yuan Chen, "System-Bus Fault Injection in SystemC Design Platform," *2nd IEEE International Conference on Secure System Integration and Reliability Improvement*, pp. 211-212, July 2008. (EI)
- Yung-Yuan Chen, Shu-Hao Hsu, and Kuen-Long Leu, "An Estimation Model of Vulnerability for Embedded Microprocessors," *2nd IEEE International Conference on Secure System Integration and Reliability Improvement*, pp. 224-225, July 2008. (EI)
- Yung-Yuan Chen, Yi-Chiang Wang, and Jian-Min Peng, "**SoC-Level Fault Injection Methodology in SystemC Design Platform,**" *Asia Simulation Conference 2008/ the 7th International Conference on System Simulation and Scientific Computing*, pp. 1787-1794, October 2008. (EI)
- 汪碩彥、陳永源，"利用軟體實踐錯誤注入進行嵌入式系統的強韌度驗證", *2008 資訊系統應用學術研討會*, October 2008.

Appendix

利用軟體實踐錯誤注入進行嵌入式系統的強韌度驗證

摘要

當製程技術進入深次微米(deep submicron)之後，系統因為雜訊或輻射線干擾而產生軟性錯誤(soft error)的機率也會明顯增加，因此必需在系統設計的過程中加入容錯設計來提高系統的可靠度。透過軟體實踐錯誤注入結果提供有關失敗模式與效應分析(Failure Mode and Effect Analysis，FMEA)參考數據給系統設計者，讓設計者了解哪些元件對錯誤較敏感，可以根據不同需求選擇最需要保護的元件來加入容錯設計。在本研究中，我們以不用更改系統內部資源為前提下，提出一個適用於一般嵌入式系統的軟體實踐錯誤注入(Software-Implemented Fault Injection)方法，並利用 ARM 為基礎的系統晶片中的暫存器單元進行實驗，利用實驗的結果來進行失敗模式與效應分析及失敗類型的分類。最後我們可以歸納出 ARM 處理器對於錯誤較敏感的暫存器種類，以及錯誤一旦發生在暫存器單元時系統可能產生的八種失敗類型。

關鍵詞：軟性錯誤(soft error)，軟體實踐錯誤注入(Software-Implemented Fault Injection)，系統晶片(System-on-Chip)，失敗模式與效應分析(Failure Mode and Effect Analysis)。

1. 前言

隨著嵌入式應用產品的普及化，例如智慧型的手持裝置到汽車、機器人及航太運用上，皆內建嵌入式系統或系統晶片(System-on-Chip, SoC)於其中。由於應用的範圍相當廣泛，當其應用於需要高安全性及可靠性的系統時，相對就有較高的風險會造成人身生命上的威脅，因此系統的可靠度就會是一個很重要的課題。

當製程進入深次微米時代之後，系統晶片的設計越來越複雜，包含的電晶體也越來越多，晶片面臨輻射性干擾或是雜訊所產生軟性錯誤的機率也會大大的提升，特別是在暫存器及記憶體元件中[1][2]。因此就必須在系統晶片設計的過程中加入容錯設計來提高系統的可靠度。

在考量如何有效的在系統晶片設計時加入容錯技術，有兩個很重要的問題需要探討。一是如何更早的在系統設計初期去驗證其強韌度，二是如何更快並且有效的提供系統晶片設計時，有關失效模式及效應分析的資料，讓設計者了解哪些元件對錯誤較敏感，可以根據不同需求選擇最需要保護的元件來加入容錯設計，進而降低重新設計的成本。而這兩項問題的解決方式都仰賴錯誤注入實驗能否提供有效數據給系統設計者。

錯誤注入是用來驗證系統可靠度的方法，主要是利用硬體或軟體的錯誤注入器將錯誤注入到硬體或是軟體的目標中，再來觀察系統產生的行為。一般來說錯誤注入的方式有(i)實體錯誤注入(ii)模擬基礎錯誤注入(iii)軟體實踐錯誤注入。第一種方法主要是利用特殊目的的硬體來做錯誤注入及觀察系統產生的行為，優點是有較快的速度及較高的準確度，缺點是要花費較高的開發成本以及有較高的風險會造成系統的損害。第二種方法是利用硬體描述語言模擬器(HDL simulator)來對模擬電路進行錯誤注入。由於是在模擬的基礎下，所以其準確度可能較差，但可節省開發的成本及時間。而軟體實踐錯誤注入則比較多樣化，目標可以是實際硬體或是軟體，通常比較容易實踐而且也可以因為不同的需求去做改變，更不需要有額外的硬體需求。雖然進行實驗所需的時間較長，但可以節省較多的成本，在可攜性上也優於其它兩種錯誤注入方式。

在先前關於軟體實踐錯誤注入的相關研究中，FERRARI(Fault and ERRor Automatic Real-time Injector) [3]利用 UNIX ptrace 系統函式對於運行中的行程(process)插入軟體中斷。當程式執行到軟體中斷時就觸發(trigger)錯誤注入，去破壞行程使用的暫存器或記憶體空間。這個工具提供了注入永久性及暫時性錯誤的方法。在這篇文章中作者只針對程式計數器(Program Counter)進行實驗，主要是在觀察當程式流程出錯時系統可能的反應，以及偵測錯誤的覆蓋率及錯誤在被偵測到之前的潛伏時間。由於測試的目標點只有一個，所以如果當錯誤發生在不同特性的暫存器時，系統會產生什麼樣的結果，我們並沒有辦法從中得知。在[4]中，作者提出一個商業化的錯誤注入工具 Xception，利用其除錯(debugging)和監控(monitoring)的特性來進行錯誤注入。這個方法不需要去修改目標程式的原始碼和插入任何的軟體中斷，由於它使用的是內建的硬體例外事件(exception)來觸發錯誤注入，所以必需更改中斷處理器(interrupt handler)。而當嵌入式系統資源有限的情況下，任意的更改系統資源可能會增加開發上的複雜度。在[5]中，作者詳細的介紹了如何利用 ptrace (process trace)函式，來開發錯誤注入工具，並透過經驗告訴我們，利用 ptrace 開發錯誤注入工具時有哪些重點是需要注意的:

- 由於大部份的 Unix 系統時間解析度(clock resolution)不盡相同，可能是 1/60 秒或是 1/100 秒，所以在時間的測量上會有些許的不同。
- 作者也提出 ptrace 函式，在執行上會很慢，會有許多次的內文切換(context switch)介於錯誤注入行程及目標行程之間。因此每一次的內文切換就會使得記憶體管理單元(MMU)及快取記憶體(cache)的內容需要重新更新，如果當目標程式使用了更多的記憶體空間，這個情況會更明顯。

而作者在文章中也提出利用 ptrace 實踐錯誤注入可能會有的優缺點。

優點：
- 以 ptrace 函式實踐的錯誤注入方式，只要稍做修改即可移植到以 Unix 為主的系統上。
- 由於使用了許多系統函式，來實踐錯誤注入程式，因此產生的執行檔很小。
- 在進行的實驗過程中，不需要更改目標程式的原始檔案，也不需要對原始檔案重新編譯。

缺點：
- 錯誤注入只能針對目前作業系統在執行的行程。
- 在同一時間只有一個行程能被注入錯誤。

在[5]中，作者不但提出了 ptrace 函式可能會有的優缺點，也詳細的介紹如何利用 ptrace 來進行錯誤注入。

在[6]中，作者利用硬體除錯介面(On Chip Debug)，利用非侵入式的技術，利用軟體實踐錯誤注入，透過內建除錯電路去存取系統內部的資訊，在不用修改或中止目標程式的情況下來進行容錯能力的驗證。但是如果我們的系統晶片在不具有內建除錯電路的情況下，就無法執行同樣的錯誤注入。

過去的研究都是在高效能的工作站或是具有內建除錯電路的環境下進行實驗。由於應用的層面不同，在嵌入式開發環境下，成本及系統資源有限(暫存器、記憶體)的情況下，並沒有辦法完全以過去的軟體錯誤注入方式來進行實驗。所以在本研究中我們以不用更改系統內部資源為前提下，提出一個適用於一般嵌入式系統的錯誤注入方法，而且利用此錯誤注入方法對暫存器進行錯誤注入實驗。我們可以觀察暫存器單元在軟性錯誤發生的情況下，那些暫存器具有較高的錯誤敏感度，並且提出相關的失敗模式與效應分析數據，告訴設計者不同的暫存器會有那些不同的失敗類型。使得設計者可以在系統設計初期，對於較敏感的元件加入適當的容錯機制，以提高系統的可靠度。

2. 錯誤注入架構

2.1 錯誤注入流程

為了達成上述目標，我們利用具有超級用戶權限(root)的 ptrace 系統函式，透過所設計的錯誤注入行程(fault injection process)對目標行程(target application process)中所執行的測試程式進行監控及更改暫存器或記憶體的內容，以此改變目標行程中程式運行的狀態，達到系統晶片內部元件受到干擾時產生錯誤的目的。在錯誤注入的過程中，錯誤注入行程利用 fork()產生目標行程，而且設定為可被追蹤，使我們設計的錯誤注入行程可以針對它進行錯誤注入，然後利用 ptrace 函式中的引數 GETREGS / SETREGS 及 PEEKTEXT / POKETEXT 去對目標行程的暫存器及記憶體進行錯誤注入。當錯誤注入完成後再讓目標程式繼續執行，最後比較執行的結果，看錯誤是否被激發，再根據不同的暫存器特性做失敗類型的分類。

2.2 錯誤注入環境

我們利用上述的軟體實踐錯誤注入方法，在一個嵌入式系統開發平台上進行錯誤注入實驗。為了收集模擬時間及暫存器使用頻率等錯誤注入參數，首先要執行一次無錯誤注入(fault free)模擬實驗。接下來即可利用收集的結果決定錯誤注入的時間點以及為之後的失敗類型分析時所用。在錯誤注入的過程中，我們利用時間觸發(timing trigger)的方式，透過 setitimer()及alarm()製作計時器。當計時器倒數為 0 時，錯誤注入行程就會透過 kill()送出 ‘SIGINT’ 訊號暫停目標行程，以進行錯誤注入。錯誤注入的值以亂數選擇 32 位元的其中 1 個位元，進行位元翻轉(bit-flip)錯誤注入。實驗流程如圖 1 所示，先執行(1)錯誤注入，利用 ptrace 系統函式呼叫核心(kernel)， (2)錯誤注入器(fault injector)透過核心對目標行程進行錯誤注入，(3)目標行程回傳錯誤注入結束狀態給錯誤注入器，透過回傳的狀態我們再分析比較系統錯誤行為，找出對於錯誤較敏感的元件。

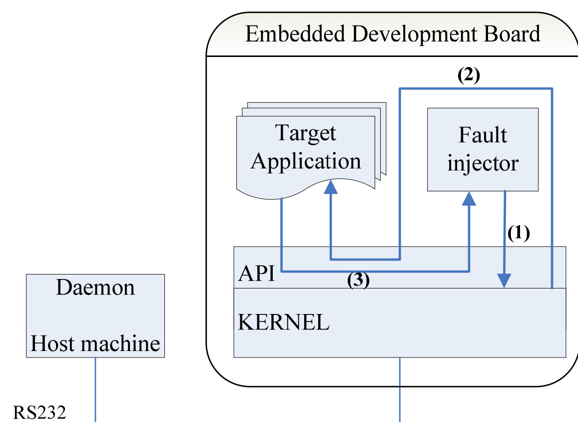

圖 1. Experimental Environment

2.3 實驗設定

由於軟性錯誤發生在暫存器的機率相當的高[2]，所以我們針對系統晶片中的暫存器單元進行軟體實踐錯誤注入，來驗證它的敏感度及可能會導致的失敗類型。在本研究中我們利用嵌入式系統開發平台(CDK)[1]作為錯誤注入的實驗環境，而整個開發平台包含ARM926EJ-S的處理器及Open Linux 2.6.19 為核心的作業系統[7]，並且選擇ARM 處理器使用者模式(user mode) 下 R0-R12 及 SP(stack pointer)、LR(linker register)、PC(program counter)、CPSR(Current Program Status Register)等暫存器，設計A、B兩個實驗來執行灌入錯誤實驗。

A. 我們利用 30x30 的矩陣相乘及 500 個數字的快速排序兩個測試程式，測試系統晶片在執行一般程式時發生錯誤可能產生的失敗類型，以及暫存器出錯可能造成系統失敗的機率。最後再利用統計的結果，分析出不同的失敗類型可能是哪些暫存器所造成。

B. 分析當特殊用途(special purpose)暫存器(PC、LR、SP)存取到錯誤的程式區段(code segment)、資料區段(data segment)、周邊裝置(I/O)及作業系統核心(Kernel)等非法空間時，系統可能產生的失敗類型。由

[1] Socle Technology – SoC Platform Solution and Service Company

於這些暫存器控制了系統晶片執行程式的流程，所以我們要對這些比較重要的元件進行詳細的測試與分析。

根據這兩項實驗，我們希望可以看到不同特性的暫存器在執行程式時可能會產生哪些失敗類型，以及每一個暫存器對錯誤的敏感度，進而再針對特殊用途的暫存器探討它存取到非法空間時系統可能產生的行為。最後根據實驗的結果可以觀察不同的暫存器錯誤時所導致系統產生不同的失敗類型。在下一章節我們會針對這些不同的失敗類型做深入的探討。

2.4 失敗類型

根據不同的程式特性，執行完錯誤注入之後可能對系統產生不同的影響，不同特性的暫存器也會有不同的失敗類型。透過分析被注入且激發的錯誤，我們可將其可能造成的失敗類型加以歸納如圖 2 所示。首先第一層將程式執行結果分成未完成(Incomplete)及完成(Complete)兩類，第二層再根據未完成的部份分出異常結束(Abnormal Ending)及沒有結束(No Ending)；在完成的部份又可根據執行時間分為正確時間(Correct Time)及錯誤的時間(Incorrect Time)結束，最後總共可分出圖 2 所列的八種失敗類型。

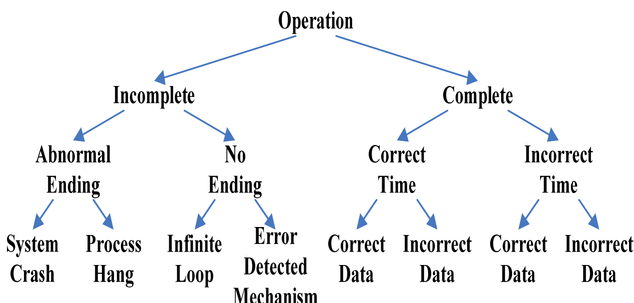

圖 2. System Failure Mode

- System Crash (SC)：因為系統晶片內部執行出錯，導致整個作業系統崩潰，而無法繼續執行。
- Process Hang (PH)：因為測試程式所使用的記憶體空間或暫存器內容出錯，導致程式執行到錯誤的代碼使測試程式中止，但此時作業系統還是可以正常運作。
- Infinite Loop (IL)：因為更改到條件判斷式的變數，導致程式無法結束，進入無窮迴圈。
- Error Detected Mechanism (EDM)：此類型會被作業系統 EDM 偵測出來(例如分頁錯誤)。
- Correct Time Correct Data (CTCD)：注入的錯誤沒有造成任何失敗，有可能是因為錯誤被注入到下一次是執行寫入的元件時，錯誤會被正確的值覆蓋，使得程式依然可以正常執行。
- Correct Time Incorrect Data (CTID)：此類型可能因為暫存的變數出錯，導致最後運算產生錯誤的結果，但是在合理的時間內結束。
- Incorrect Time Incorrect Data (ITID)：此類型可能因為控制流程錯誤(control flow error)[10]，會使得程式最後的執行時間及結果都是不正確。
- Incorrect Time Correct Data(ITCD)：此類型因為控制流程錯誤，使程式執行時間延長，但運算的結果是正確的。

3. 實驗結果與分析

由於不同特性的暫存器，在錯誤發生時可能會產生的反應行為也會有所不同，所以我們為了分析出在系統晶片中對於錯誤敏感度較高的元件，進行了章節 2.3 所描述的實驗 A 和 B。針對每一個測試程式，在每一個暫存器注入了 1000 個錯，總共注入了 17000 個錯誤在不同的暫存器當中，再根據系統晶片產生的行為做出下面的分析。

A-1 暫存器在受到干擾時，不同失敗類型的分佈。

由圖 3 我們可以看出，系統晶片在相同的環境下，執行不同程式時，暫存器單元發生錯誤的情況下，有 27.14%錯誤會被激發造成系統失敗。其中有高達 22.17%的機率會使測試程式無法執行完成，只有 0.07%的比例會被系統所偵測出來。以一個系統的角度來看，如果錯誤被系統偵測出來的機率如此的低，此嵌入式系統的可靠度可能會受到質疑。換句話說，當暫存器單元出錯時，有 27.14%的可能性會使得測試程式出錯，而導致嵌入式系統無法正常執行，這是一個相當高的比例。如果當這個系統是應用在需要高安全度及可靠度的環境下，所造成的影響就會是不可預期的。透過上述的分析，我們再利用圖 4 和 5 可以看出在不同的測試程式中錯誤所造成的失敗類型及其機率分佈。

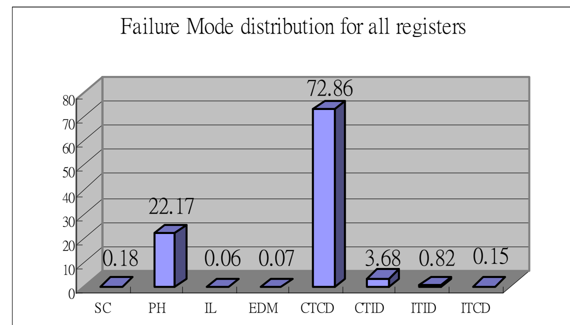

圖 3. Failure Mode Distribution for all Registers

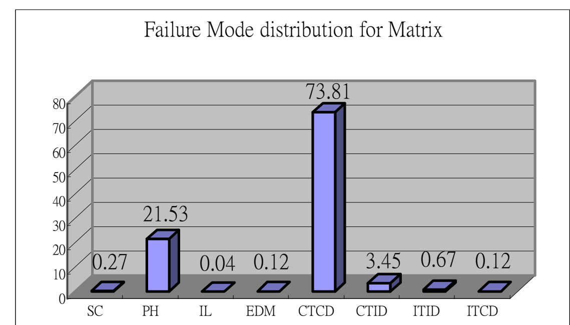

圖 4. Failure Mode distribution for Matrix

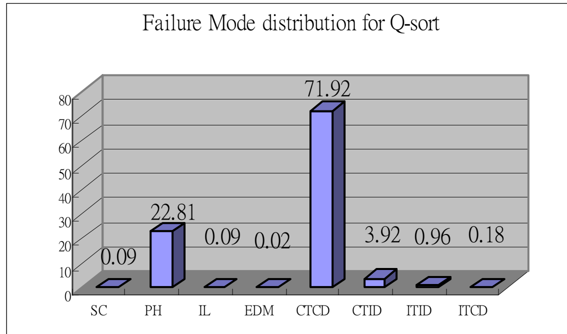

圖 5. Failure Mode distribution for Q-sort

A-2 不同暫存器的系統失敗機率。

從表 1 和 2 可以看出，每一個不同的暫存器，在不同的測試程式下，發生錯誤所造成系統失敗的機率。失敗機率(failure rate)主要是根據每一個暫存器發生的失敗總數除以所注入錯誤總數所統計出來。在這裡以 R11、SP、PC 錯誤被激發的機率最高。根據[8][9]，R11 暫存器可以是儲存變數或者是指向堆疊框架(stack frame) 的框架指標(frame pointer)，當系統要存取區域變數時需透過 R11 找出資料位置。所以當它出錯時會造成失敗的機率也會相對較高。而 SP 主要是用來記錄所儲存資料的位置，當 SP 出錯時抓取到錯誤的值也會造成失敗的情形，而 PC 出錯則會使程式執行到錯誤位址中的的指令或資料，使得系統出錯的比例升高。而值得一提的是 CPSR，失敗的機率只有 3%-5%。這個暫存器主要是儲存 ARM 處理器目前的狀態，在錯誤注入的情況下失敗的類型都是落在 PH 中，並沒有因為錯誤注入導致其改變處理器的模式位元(mode bit)中的內容，而使程式進到錯誤的處理器模式下執行，產生未知的後果。

表 1. Register Failure Rate in Matrix

Reg.	failure rate	Reg.	failure rate	Reg.	failure rate
R0	18.60%	R6	41.60%	R12	10.80%
R1	11.80%	R7	10%	SP	63.40%
R2	20.80%	R8	1.20%	LR	8.40%
R3	23%	R9	7.20%	PC	80%
R4	10%	R10	25.20%	CPSR	3%
R5	14.80%	R11	95.40%		

表 2. Register Failure Rate in Q-sort

Reg.	failure rate	Reg.	failure rate	Reg.	failure rate
R0	19.8%	R6	55.2%	R12	1.8%
R1	9.8%	R7	12%	SP	70.2%
R2	13%	R8	2.4%	LR	12.4%
R3	15.2%	R9	6.8%	PC	84.2%
R4	36.4%	R10	19.2%	CPSR	5.4%
R5	18%	R11	95.6%		

再從表 3 我們可以看出 R11、SP、PC 這三類暫存器，在不同的測試程式下失敗的類型大部份都是落在 PH 的比例較高。這也意味著當這三類暫存器出錯時，大部份都會導致應用程式無法執行結束。因此系統廠商可以在系統設計初期將這些數據列入考量。

	Reg. &failure rate	SC	PH	IL	EDM	CTCD	CTID	ITID	ITCD
Matrix	R11	0.2%	86%	0%	1.4%	4.6%	7.8%	0%	0%
	SP	2%	61%	0%	0%	36.6%	0.4%	0%	0%
	PC	0.2%	72.8%	0%	0.2%	20%	4.8%	0%	2%
Q-sort	R11	0.2%	93.4%	0%	0.4%	4.4%	1.6%	0%	0%
	SP	0%	69.2%	0%	0%	29.8%	1%	0%	0%
	PC	0%	79.8%	0%	0%	15.8%	3.2%	0.2%	1%

表 3. Failure Mode Distribution of R11,SP,PC

最後我們利用實驗 B，將錯誤注入在特殊用途的暫存器 SP、LR 及 PC 中，分析比較當這些暫存器存取到錯誤的位址時(此錯誤的位址可能是落在 Code、Data、I/O 或 Kernel 記憶體區段)，系統會產生怎樣的反應。由表 4 我們可以看出，由於 LR 只有在程式執行過程中，有呼叫副程式時才會使用到。而它可能因為跳回錯誤的位置，而導致大部份都是落在 PH 的失敗類型。由於整個程式中 LR 的使用頻率較低，所以造成系統失敗的機率也會偏低；而 SP 及 PC 存取到錯誤的位址時，有很高的機率會造成系統失敗。SP 因為存取到錯誤區段，而導致程式變數或暫存器的值出錯，使得它失敗的機率也會偏高；PC 所存的是下一道指令的位址，所以 PC 執行到錯誤的程式區段時，會造成程式控制流程的錯誤。由上述的統計分析發現，使用頻率越高的暫存器，出錯的機率也會變高。透過失敗模式與效應分析我們可以看出，在特殊用途的暫存器中，SP 及 PC 這兩類暫存器對於錯誤較為敏感。

表 4. Failure Mode Distribution of Special Purpose Registers

PC	SC	PH	EDM	IL	CTCD	CTID	ITID	ITCD
Code	0%	54.38%	1.25%	0%	16.46%	11.46%	6.04%	10.42%
Data	0%	100%	0%	0%	0%	0%	0%	0%
I/O	0%	100%	0%	0%	0%	0%	0%	0%
Kernel	0%	100%	0%	0%	0%	0%	0%	0%
LR	**SC**	**PH**	**EDM**	**IL**	**CTCD**	**CTID**	**ITID**	**ITCD**
Code	0%	7.5%	0%	0%	90.5%	1%	0%	1%
Data	0%	12%	0%	0%	88%	0%	0%	0%
I/O	0%	9%	0%	0%	91%	0%	0%	0%
Kernel	0%	9.5%	0%	0%	90.5%	0%	0%	0%
SP	**SC**	**PH**	**EDM**	**IL**	**CTCD**	**CTID**	**ITID**	**ITCD**
Code	0%	98.5%	0%	0%	1.5%	0%	0%	0%
Data	0%	77%	0%	5%	18%	0%	0%	0%
I/O	0%	97%	0%	0%	3%	0%	0%	0%
Kernel	0%	100%	0%	0%	0%	0%	0%	0%

透過實驗 A 的結果，我們也可以反過來估算，不同的失敗類型可能是由哪些暫存器所引起的。從圖 6 中可以看出，主要導致系統產生 SC 的暫存器以 R0 及 SP 最高。由於 R0 主要是儲存程式所使用的參數、計算時暫存的值及運算後的結果，因此當 R0 及 SP 出錯時，有很高的機率會造成 SC。

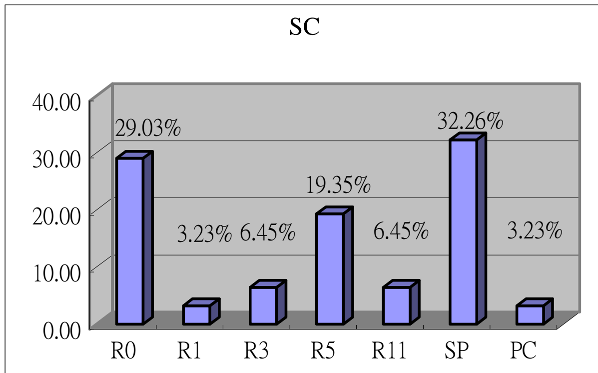

圖 6. Register Failure Mode Distribution of SC

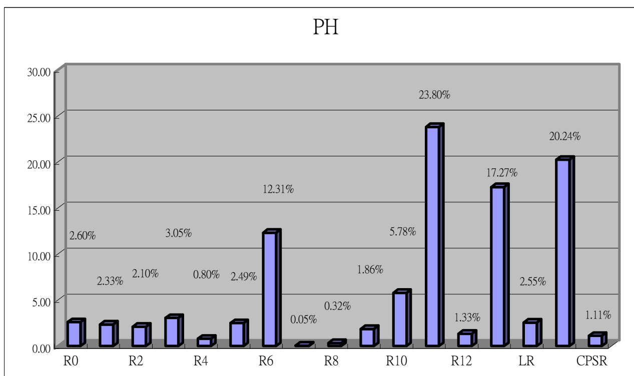

圖 7. Register Failure Mode Distribution of PH

從圖 8 我們可以看出，R5 造成無窮迴圈的機率最高。而導致這種失敗類型的原因，可能因為 R5 中儲存了條件判斷式的變數，一旦 R5 發生錯誤，造成可能永遠都無法符合離開迴圈的條件，導致系統進入無窮迴圈。

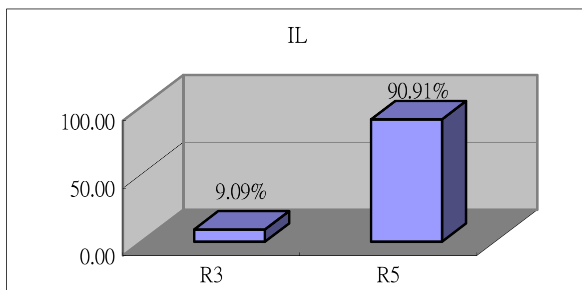

圖 8. Register Failure Mode Distribution of IL

從圖 9 中我們可以看出，R11 的錯誤會被錯誤偵測機制所偵測出來的機率最高。從圖 9 中我們可以判斷，實驗所使用的嵌入式系統的錯誤偵測機制並不完善，系統設計者需要提出更完善的偵測方法，來提升系統可靠度。

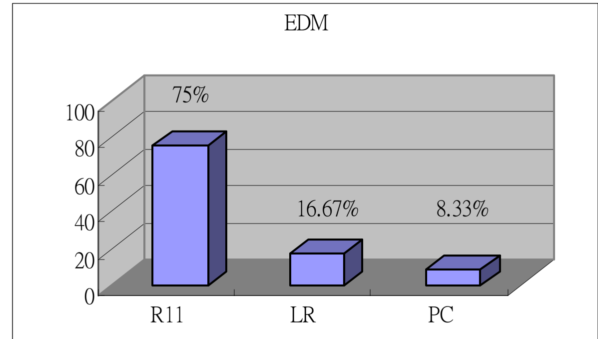

圖 9.Register Failure Mode Distribution of EDM

4. 結論

在本研究中，我們以不用更改系統內部資源為前提下，提出一個適用於一般嵌入式系統的錯誤注入方法。由於軟性錯誤發生在暫存器的機率偏高，我們利用所提出的錯誤注入方法對暫存器進行錯誤注入實驗，來針對系統晶片的暫存器單元進行失敗模式與效應分析及失敗類型的分類。根據實驗結果及分析歸納出，以 ARM 為主的處理器，在使用者模式下，R11、SP、PC 這三類暫存器，在遭遇到較惡劣的環境時，對於錯誤是較敏感的。而且我們根據錯誤導致系統產生的行為分出八種失敗類型，最後我們再根據不同失敗類型，找出其可能導致的暫存器單元。透過這樣的分析，提供有效的數據給系統設計者。讓設計者根據其需求，選擇最需要保護的暫存器。因此對於嵌入式系統的設計者來說，可以利用此錯誤注入方式來驗證其原始系統，對於錯誤的敏感度。進而了解其中一旦發生錯誤時，最容易造成系統失敗的元件。根據此分析結果，設計者即可決定容錯機制的設計方針。這也是本研究所提供的軟體實踐錯誤注入方法最重要的貢獻。

5. 致謝

本研究要特別感謝國科會計畫編號 NSC 96 – 2221 – E – 216 – 006 的贊助。

6. 參考文獻

[1]. C. Constantinescu, “Impact of Deep Submicron Technology on Dependability of VLSI Circuits,” *IEEE Intl. Conf. On Dependable Systems and Networks (DSN)*, pp. 205-209, 2002.

[2]. Alfredo B ,Paolo P. “Fault injection techniques and tools for embedded systems reliability evaluation”, Kluwer Academic Publishers, 2003

[3]. G.A. Kanawati, N.A. Kanawati, and J.A. Abraham, “FERRARI: A Flexible Software-Based Fault and Error Injection System,” IEEE Trans. on Computers, vol. 44, no. 2, pp. 248-260, Feb. 1995.

[4]. J. Carreira, H. Madeira, and J.G. Silva, “Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers,” IEEE Trans. on Software Eng., vol. 24, no. 2, pp. 125-136, Feb.1998.

[5]. Sieh, "Fault-Injector using UNIX ptrace Interface", Internal Report 11/93, IMMD3, Universität Erlangen-Nürnberg, 1993.
[6]. Fidalgo, A.V.; Alves, G.R.; Ferreira, J.M." Real time fault injection using a modified debugging infrastructure" On-Line Testing Symposium, 2006. IOLTS 2006. 12th IEEE International, Page(s):6,10-12 July 2006.
[7]. http://www.socle-tech.com.tw/
[8]. AAPCS, "Procedure Call Standard for the ARM Architecture", April 2008
[9]. ATPCS, " The ARM-THUMB Procedure Call Standard", 24 October, 2000
[10]. Yasser Sedaghat, Seyed Ghassem Miremadi, Mahdi Fazeli. "A Software-Based Error Detection Technique Using Encoded Signatures". Symposium on Defect and Fault-Tolerance in VLSI Systems, pp. 389-400, 2006

行政院國家科學委員會補助國內專家學者出席國際學術會議報告

97 年 08 月 04 日

報告人姓名	陳永源	服務機構及職稱	中華大學資訊工程學系 教授
會議 時間 地點	07 月 23-25, 2008 希臘克里特島伊拉克利翁市	本會核定補助文號	NSC 96－2221－E－216－006－
會議名稱	（中文）第 12 屆 WSEAS 國際電腦會議 （英文）12th WSEAS International Conference on COMPUTERS		
發表論文題目	（中文） （英文）Datapath Error Detection Using Hybrid Detection Approach for High-Performance Microprocessors		

報告內容應包括下列各項：

一、 參加會議經過

此會議是在希臘克里特島伊拉克利翁市舉行，作者是搭乘泰航班機到雅典，再轉搭奧林匹亞航空到克里特島伊拉克利翁市。此第12屆WSEAS CSCC Multiconference 會議共有超過1300篇論文投稿，錄取641篇論文，錄取率不到五成。在錄取641篇論文中，再精選43篇論文，直接推薦WSEAS Transactions 期刊的發表。作者所發表的論文，獲得最佳論文獎，並直接被錄取期刊的發表。此會議的主題範圍包括了High Performance Languages, Operating Systems, Hardware Engineering, Supercomputing, Parallel Computing Systems Architectures, Software Evaluation Standards, E-commerce, Interconnection Networks, Mobile Networks, Distributed Real Time Systems, Distributed Data Base, and other relevant topics and applications。參加的學者來自美國，台灣，日本，馬來西亞以及歐洲的國家。第一天的下午有一場特別的專題演講，講題是"Embedded Systems - Scientific Challenges and Work Directions" by Prof. Joseph Sifakis, Turing Award 2007, CNRS researcher and the Founder of Verimag laboratory, in Grenoble, France. 同時主辦單位特別邀請希臘國防部副部長，來頒發WSEAS Fellow給Prof. Joseph Sifakis，來表彰其在 'Computer' 領域上的貢獻。作者的論文被安排在第一天晚上報告，講題是"Datapath Error Detection Using Hybrid Detection Approach for High-Performance Microprocessors"。

二、 與會心得

此會議是一年一度 WSEAS 組織所舉辦在 Circuits, Systems, Communication 和 Computers 方面的會議，今年是第 12 屆。可以透過此會議與其他國家的學者討論交流，並且掌握最新的研究題材與研究結果。可以用來檢視作者目前及未來的研究方向與課題的價值與重要性，對於以後的研究有相當的幫助。另外也有機會請教一些國際級的學者，傾聽他們對一些議題的意見及看法，可以幫助作者對一些困惑的地方及觀念做一澄清，對於往後的研究也是有相當的幫助。研究心得是未來輻射線粒子，對於深次微米製程的晶片影響力越來越大，造成暫時性錯誤的機率也越來越高。此問題將會影響處理器及系統晶片的可靠度。所以有幾個問題值得進一步的探討(針對深次微米製程的系統晶片): FMEA 分析流程建立，容錯技術的有效性，灌錯及錯誤模擬分析工具環境的建立，系統驗證分析平台建立等等。

三、 攜回資料名稱及內容

一本會議的論文集及光碟片