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(III)
研究成果報告(精簡版)

計 畫 類 別 ：個別型
計 畫 編 號 ：NSC 97-2221-E-216-018-
執 行 期 間 ：97 年 08 月 01 日至 98 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：許仲賢
碩士班研究生-兼任助理人員：汪宜強
碩士班研究生-兼任助理人員：呂凱平

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 98 年 10 月 31 日

Summary

This report describes the results achieved in the third 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 identify the system failure behaviors, 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 and risk model 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 propose a risk model to facilitate the measure of the robustness and scales of failure-induced risks in a system, which can be used to identify the critical components and major failure modes for protection so as to effectively reduce the impact of failures to the system. Next, we complete the development of the simulation-based and software-implemented fault injection, robustness verification and risk assessment tool based on the devising SystemC and IP-based fault injection methodologies and failure mode and effects analysis (FMEA) process under the environment of CoWare Platform Architect. The proposed tool can significantly reduce the effort and time for validating the robustness and safety of *SoC*. Our tool can perform automatically the fault injection campaigns, and classify the failure modes of the system failure behaviors. 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 verification framework with an experimental ARM-based system that is modeled at different levels of abstraction.

Abstract: 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 due to the rapid increasing rate of radiation-induced soft errors while the *SoC* fabrication enters the very deep submicron technology. Therefore, the *SoC* dependability becomes a critical issue in safety-critical applications. Validating such systems is imperative to guarantee the dependability of the systems before they are being put to use. Moreover, it is beneficial to assess the *SoC* robustness in early design phase in order to significantly reduce the cost and time of re-design. To fulfill such needs, in this study, we propose a useful IP-based *SoC*-level risk model using failure mode and effects analysis (FMEA) method to assess the robustness of a *SoC* in SystemC transaction-level modeling (TLM) design level. The proposed risk model is able to facilitate the measure of the robustness and scales of failure-induced risks in a system, which can be used to identify the critical components and major failure modes for protection so as to effectively reduce the impact of failures to the system. A case study is used to demonstrate our risk model under CoWare Platform Architect environment. A system verification tool was created to assist us in measuring the robustness of the system, in locating the weaknesses of the system, and in understanding the effect of faults on system failure behavior during the *SoC* design phase. The contribution of this work is to promote the dependability verification to TLM abstraction level that can significantly enhance the simulation performance, and provide the comprehensive results to validate the system dependability in early design phase for safety-critical applications.

Keywords: FMEA, risk assessment, SystemC, safety-critical application, system-on-chip.

1. Introduction

As system-on-chip (*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]. Since *SoC* becomes prevalent in the intelligent system applications, such as intelligent automotive systems or intelligent robots, which require a stringent dependability while the systems are in operation. Thus, it is essential to perform the failure mode and effects analysis (FMEA) method to locate the vulnerability of the *SoC* and provide the practical fault-tolerant strategies to improve its reliability and safety [4]. However, due to the high complexity of the *SoC*, the incorporation of the FMEA 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 dependability as early in the development phase to reduce the re-design cost. As a result, a *SoC*-level dependability verification platform is required to facilitate the designers in assessing the robustness of a *SoC* with an efficient manner. Normally, the FMEA method and fault injection approach are employed to analyze the impact of failures to the system and measure the risks of the system.

Previously, the issue of *SoC*-level risk assessment is seldom addressed especially in SystemC transaction-level modeling (TLM) design level. In paper [4], the authors presented a FMEA method at *SoC*-level design in RTL description to design in compliance with IEC61508. A memory sub-system embedded in fault-robust microcontrollers for automotive applications was used to demonstrate the feasibility of their FMEA method. However, the scheme presented in [4] can only apply to RTL and gate level, which limits the scope of its application. Furthermore, the complexity of oncoming *SoC* increases rapidly, so it may still require considerable time and efforts to implement a *SoC* using RTL description. In paper [5], the authors proposed a dependability benchmark for automotive engine control applications. They showed the feasibility of the proposed dependability benchmark using a prototype of diesel electronic control unit (ECU) control engine system. The fault injection campaigns were conducted to measure the dependability of benchmark prototype. The domain

of application for dependability benchmark specification presented in paper [5] confines to the automotive engine control systems which are built by commercial off-the-shelf (COTS) components. While dependability evaluation is performed after physical systems have been built, the costs of re-designing systems due to inadequate dependability can be prohibitively expensive.

To cope with the above problems, we raise the modeling level of *SoC* design to SystemC TLM level. At TLM design level, we can more effectively deal with the issues of design complexity, simulation performance, development cost and dependability for safety-critical *SoC* applications. In this study, an IP-based *SoC*-level risk model combining FMEA with fault injection method is proposed to identify and assess the potential failure modes in a *SoC* modeled at SystemC TLM design level, and measure the risk scales of consequences resulting from various failure modes. Since the modeling of *SoCs* is raised to the level of TLM abstraction, the performance of fault injection and simulation is enhanced significantly. As a result, the risk assessment can be carried out efficiently in early design phase to validate the robustness of the *SoC* and identify the critical components and failure modes to be protected if necessary. Our risk model is valuable in that it provides the capability to quickly assess the *SoC* dependability, and if the measured dependability cannot meet the system requirement, the results of FMEA will be used to help us develop a feasible and cost-effective risk-reduction process.

The remaining report is organized as follows. In Section 2, the SystemC untimed/timed functional TLM and the concept of Transactor are presented. A *SoC*-level risk model is proposed in the following section. We briefly describe a system verification platform in Section 4. A case study with the experimental results and a thorough vulnerability and risk analysis are given in Section 5. The conclusions appear in Section 6.

2 SystemC Functional TLM

SystemC [6], 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) [7] 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 [11] for system design platform. 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. 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.

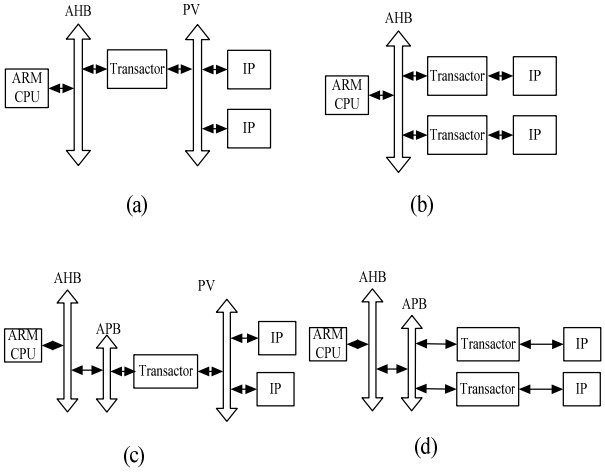

Fig. 1: ARM-based system modeled with mixed

levels of PV and PV+T, where IP represents the slave module.

3 SoC-Level Risk Assessment

When *SoCs* are applied to safety-critical applications, fault-robust designs with the dependability validation are required to guarantee that the developed *SoCs* are able to comply with the dependability or safety requirements defined by the international norms, such as IEC61508 [8, 9]. For the complicated IP-based *SoCs* or embedded systems, it is unpractical and not cost-effective to protect the entire *SoC* or system. Analyzing the vulnerability of *SoCs* or systems can help designers not only invest limited resources on the most crucial region but also understand the gain derived from the investment.

In this section, we propose a *SoC*-level risk model to quickly assess the *SoC's* vulnerability at SystemC TLM level. Conceptually, our risk model is based on the FMEA method with the simulation-based fault injection approach to measure the robustness of *SoCs*. From the assessment results, the rank of component vulnerability related to the risk scale of causing the system failure can be acquired. The notations used in the risk model are developed below.

- n: number of components to be investigated in the *SoC*;
- z: number of possible failure modes of the *SoC*;
- *C(i)*: the i^{th} component, where $1 \le i \le n$;
- *FR_C(i)*: failure rate of the i^{th} component;
- *SFR_C(i)*: the part of *SoC* failure rate contributed from the failure rate of the i^{th} component;
- *SFR*: *SoC* failure rate;
- *R(t)*: *SoC* reliability;
- *FM(k)*: the k^{th} failure mode of the *SoC*, where $1 \le k \le z$;
- *NE*: no effect which means that a fault/error happening in a component has no impact on the *SoC* operation at all;
- *P (i, FM(K))*: probability of *FM(K)* if a failure occurs in the i^{th} component;
- *P (i, NE)*: probability of no effect for a failure occurring in the i^{th} component;
- *P(i, SF)*: probability of *SoC* failure for a failure occurring in the i^{th} component;
- *S_FM(k)*: severity rate of the k^{th} failure mode, where $1 \le k \le z$;
- *RPN_C(i)*: risk priority number of the i^{th} component;
- *RPN_FM(k)*: risk priority number of the k^{th} failure mode.

The potential *SoC* failure modes can be identified from the fault injection campaigns. We can inject the faults into a specific component so as to result in the failures of that component, and then investigate the effect of component's failures on the *SoC* behaviors. Throughout the injection campaigns for each component, we can identify the failure modes of the *SoC*, which are caused by the failures of components in the *SoC*. The parameters of z and *P(i, FM(k))* can be derived from the fault injection campaigns. The derivation process by fault injection experiments is described as follows:
Several notations are developed first:

- *S_FM*: a set of *SoC* failure modes used to record the possible *SoC* failure modes happened in the fault injection campaigns.
- *counter(i, k)*: a counter array for the i^{th} component used to count the number of the k^{th} *SoC* failure mode occurred in the fault injection experiment of the i^{th} component, where $1 \le i \le n$, and k represents the k^{th} *SoC* failure mode in the set *S_FM*.
- *no_fi(i)*: the number of fault injection campaigns performed in the i^{th} component, where$1 \le i \le n$.

Fault injection process:

z = 0; *S_FM = Φ*;
for i = 1 to n //fault injection experiment for the i^{th} component;//
  for j = 1 to no_fi(i)
  {injecting a fault into the i^{th} component, and investigating the effect of component's failure on the *SoC* behavior; then, identifying which failure mode of the *SoC* encountered due to this fault injection.
  if (the *SoC* failure mode caused by this injection campaign is new, and therefore, it cannot be found in the set *S_FM*; in other words, this type of failure mode does not occur in the previous injection campaigns)
  then {*z = z + 1*; adding this *SoC* failure mode to the set *S_FM*; *k = z*; *counter(i, k) = counter(i, k) + 1*}
  else {find the value of *k* by locating the position of current *SoC* failure mode in the set *S_FM*;

then *counter(i, k) = counter(i, k) + 1*}
}

After carrying out the above injection experiments, the set *S_FM* is obtained. Next, the parameter of *P(i, FM(K))* can be computed by

$$P(i, FM(K)) = \frac{counter(i,k)}{no\_fi(i)}$$

Where $1 \le i \le n$ and $1 \le k \le z$. The following expressions are exploited to evaluate the terms of *P(i, SF)* and *P(i, NE)*.

$$P(i, SF) = \sum_{k=1}^{z} P(i, FM(k))$$

$$P(i, NE) = 1 - P(i, SF)$$

The derivation of the component's failure rate is out of the scope of this study, so we here assume the data of *FR_C(i)*, for $1 \le i \le n$, are given. The part of *SoC* failure rate contributed from failure rate of the i^{th} component can be calculated by

$$SFR\_C(i) = FR\_C(i) \times P(i, SF)$$

It is evident that the *SoC* failure rate and *SoC* reliability can be written as

$$SFR = \sum_{i=1}^{n} SFR\_C(i)$$

$$R(t) = e^{-SFR \times t}$$

The meaning of the parameter *S_FM(k)* and the role it playing can be explained from the aspect of FMEA process [10]. The method of FMEA is to identify all possible failure modes of a *SoC* and analyze the effects or consequences of the identified failure modes. In general, an FMEA records each potential failure mode, its effect in the next level, and the cause of failure. We note that the faults occurring in different components could cause the same *SoC* failure mode, whereas the severity degree of the consequences resulting from various *SoC* failure modes could not be identical. The parameter *S_FM(k)* is exploited to express the severity rate of the consequence resulting from the k^{th} failure mode, where $1 \le k \le z$.

We illustrate the risk evaluation with FMEA idea using the following example. An ECU running engine control software is employed for automotive engine control. Its outputs are used to control the engine operation. The ECU could encounter several types of output failures due to hardware or software faults in ECU. The various types of failure mode of ECU outputs would result in different levels of risk/criticality on the controlled engine. A risk assessment is performed to identify the potential failure modes of ECU outputs as well as the likelihood of failure occurrence, and estimate the resulting risks of the ECU-controlled engine.

In the following, we propose an effective *SoC*-level FMEA method to assess the risk-priority number (*RPN*) for the components inside the *SoC* and for the potential *SoC* failure modes. A component's *RPN* aims to rate the risk of the consequences caused by component's failures. In other words, a component's *RPN* represents how serious is the impact of component's failures on the system safety. A risk assessment should be carried out to identify the critical components within a *SoC* and try to mitigate the risks caused by those critical components. Once the critical components and their risk scales have been identified, the risk-reduction process, for example fault-tolerant design, should be activated to improve the system dependability. *RPN* can also give the protection priority among the analyzed components. As a result, a feasible risk-reduction approach can be developed to effectively protect the critical components and enhance the system robustness and safety.

The parameter *RPN_C(i)*, i.e. risk scale of failures occurring in the i^{th} component, can be computed by

$$RPN\_C(i) = FR\_C(i) \times \sum_{k=1}^{z} P(i, FM(k)) \times S\_FM(k)$$

where $1 \le i \le n$. The expression of *RPN_C(i)* contains three terms which are, from left to right, failure rate of the i^{th} component, probability of *FM(K)* if a failure occurs in the i^{th} component, and severity rate of the k^{th} failure mode. As stated previously, a component's failure could result in several different failure modes, and each identified failure mode has its potential impact on the system safety. So, *RPN_C(i)* is the summation of the following expression *FR_C(i)* × *P (i, FM(K))* × *S_FM(k)*, for *k* from one to *z*. The term of *FR_C(i)* × *P (i, FM(K))* represents the occurrence rate of the k^{th}

failure mode, which is caused by the i^{th} component failing to perform its intended function.

The *RPN_FM(k)* represents the risk scale of the k^{th} failure mode, which can be calculated by

$$RPN\_FM(k) = S\_FM(k) \times \sum_{i=1}^{n} FR\_C(i) \times P(i, FM(k))$$

where $1 \le k \le z$. $\sum_{i=1}^{n} FR\_C(i) \times P(i, FM(k))$ expresses the occurrence rate of the k^{th} failure mode in a *SoC*. This sort of assessment can reveal the risk levels of the failure modes to its system and identify the major failure modes for protection so as to reduce the impact of failures to the system.

4 System Verification Platform

We have created an effective robustness verification tool under the environment of CoWare Platform Architect [11] for dependability validation of system design with SystemC. Figure 2 shows the operational flow of the verification tool. The tool platform provides the capability to quickly handle the operation of fault injection campaigns and dependability analysis for the systems modeled by one or a mixture of the following levels of abstraction [12, 13]: bus-cycle accurate level, untimed functional TLM with primitive channel sc_fifo, and timed functional TLM with hierarchical channel. So, the tool is able to deal with the fault injection at different modeling levels of abstraction and offers the time-triggered or event-triggered methodologies to decide when to inject a fault. This injection tool can significantly reduce the effort and time for performing the fault injection campaigns. Besides that, the verification platform dramatically increases the efficiency of carrying out the system robustness validation and risk assessment.

5 Case Study

An ARM-based *SoC* platform provided by CoWare Platform Architect [11] was used to demonstrate the feasibility of our risk model. The illustrated *SoC* platform was modeled at the timed functional TLM abstraction level. This case study is to investigate two components, AMBA AHB and the memory sub-system, to assess their risk scales to the *SoC*-controlled system. We exploited the system verification platform to perform the fault injection process associated with the risk model presented in Section 3 to identify the potential failure modes and obtain the values of *z* and *P(i, FM(k))* for the components of AMBA AHB and memory sub-system. The possible *SoC* failure modes classified from the fault injection process could be fatal failure (FF), such as system crash or process hang, silent data corruption (SDC), correct data/incorrect time (CD/IT), and deadlock (DL) (note that we declare the failure mode as DL if the execution of benchmark exceeds the 1.5 times of normal execution time). In the following, we summarize the data used in this case study.

- n = 2, {*C(1), C(2)*} = {AMBA AHB, memory sub-system}.
- z = 4, {*FM(1), FM(2), FM(3), FM(4)*} = {FF, SDC, CD/IT, DL}.
- The benchmarks employed in the fault injection process are: JPEG (pixels: 255 × 154), matrix multiplication (M-M: 50 × 50), quicksort (QS: 3000 elements) and FFT (256 points).

5.1 AMBA AHB Vulnerability Assessment

The system bus, such as AMBA AHB, provides an interconnected platform for IP-based *SoC*. Apparently, the robustness of system bus plays an important role in the *SoC* reliability.

The results of fault injection process for AHB system bus under various benchmarks are shown in Table 1, which has been published in our previous paper [14]. The results of a particular benchmark in Table 1 were derived from the six thousand fault injection campaigns, where each injection campaign injected 1-bit flip fault to bus signals. The fault duration lasts for the length of one-time data transaction. The statistics derived from six thousand times of fault injection campaigns have been verified to guarantee the validity of the analysis. We also found that the rank of vulnerability of bus signals is ‘HADDR’ > ‘HSIZE’ > ‘HDATA’ for all benchmarks, if the fault targets are restricted to those three categories of bus signals.

From Table 1, it is evident that the susceptibility of the *SoC* to bus faults is benchmark-dependent and the rank of system bus vulnerability over different benchmarks is JPEG > M-M > FFT > QS. However, all benchmarks exhibit the same trend in that the probabilities of FF show no substantial difference, and while a fault arises in the bus signals, the occurring probabilities of SDC and FF occupy the top two ranks. The results of the last row offer the

average statistics over four benchmarks employed in the validation process. Since the probabilities of *SoC* failure modes are benchmark-variant, the average results illustrated in Table 1 give us the expected probabilities for the system bus vulnerability of the developing *SoC*, which are very valuable for us to gain the robustness of the bus system and the critical bus signals to be protected. From the experimental results, we see that the 'HADDR' is the top priority to protect. The robustness measure of the bus system is only 0.2678, which means that a fault occurring in the bus system, the *SoC* has the probability of 0.2678 to be survived for that fault. Last but not least, we note that the SDC is the most popular failure mode for the demonstrated *SoC* responding to the bus faults or errors.

Table 1: *P (1, FM(K)), P (1, SF)* and *P (1, NE)* for the used benchmarks.

	FF (%)	SDC (%)	CD/IT (%)	DL (%)	SF (%)	NE (%)
JPEG	18.57	45.90	0.16	15.88	80.51	19.49
M-M	18.95	55.06	2.15	3.57	79.73	20.27
FFT	20.18	21.09	15.74	6.38	63.39	36.61
QS	20.06	17.52	12.24	5.67	55.50	44.50
Avg.	19.41	38.16	7.59	8.06	73.22	26.78

5.2 Memory Sub-System Vulnerability Assessment

The memory sub-system could be affected by the radiation articles, which may cause the bit-flipped soft errors. However, the bit errors won't cause damage to the system if one of the following situations occurs:

- Situation 1: The benchmark never reads the affected words after the bit errors happen.
- Situation 2: The first access to the affected words after the occurrence of bit errors is the 'write' action.

Otherwise, the bit errors could cause damage to the system. Clearly, if the first access to the affected words after the occurrence of bit errors is the 'read' action, the bit errors will be propagated and could finally lead to the failures of *SoC* operation. So, whether the bit errors will become fatal or not, it all depends on the occurring time of bit errors, the locations of affected words, and the benchmark's memory access patterns after the occurrence of bit errors.

According to the above discussion, two interesting issues arise; one is the propagation probability of bit errors and another is the failure probability of propagated bit errors. We define the propagation probability of bit errors as the probability of bit errors which will be read out and propagated to influence the execution of the benchmarks. The failure probability of propagated bit errors represents the probability of propagated bit errors which will finally result in the failures of *SoC* operation. We then performed two types of experiments to assess the propagation probability and failure probability of bit errors.

Type 1 experiment: we develop the experimental process as described below to measure the propagation probability of bit errors. The following notations are used in the experimental process.

- N_{bench}: the number of benchmarks used in the experiments.
- $N_{inj}(j)$: the number of fault injection campaigns performed in the j^{th} benchmark's experiment.
- $C_{p\text{-}b\text{-}err}$: counter of propagated bit errors.
- $N_{p\text{-}b\text{-}err}$: the number of propagated bit errors.
- S_m: address space of memory sub-system.
- $N_{d\text{-}t}$: the number of read/write data transactions occurring in the bus system during the benchmark execution.
- T_{error}: the occurring time of bit error.
- A_{error}: the address of affected memory word.
- $S_{p\text{-}b\text{-}err}(j)$: set of propagated bit errors conducted in the j^{th} benchmark's experiment.
- $P_{p\text{-}b\text{-}err}$: propagation probability of bit errors.

Experimental Process: We injected a bit-flipped error into a randomly chosen memory address at random read/write transaction time for each injection campaign. As stated earlier, this bit error could either be propagated to the system outside the memory sub-system or not. If yes, then we add one to the parameter $C_{p\text{-}b\text{-}err}$. The parameter $N_{p\text{-}b\text{-}err}$ is set by users and employed as the terminated condition for the current benchmark's experiment. When the value of $C_{p\text{-}b\text{-}err}$ reaches to $N_{p\text{-}b\text{-}err}$, the process of current benchmark's experiment is terminated. The $P_{p\text{-}b\text{-}err}$ can then be derived from $N_{p\text{-}b\text{-}err}$ divided by N_{inj}. The values of N_{bench}, S_m and $N_{p\text{-}b\text{-}err}$ are given before performing the experimental process.

for j = 1 to N_{bench}
{
Step 1: Run the j^{th} benchmark in the experimental *SoC* platform under CoWare Platform Architect to collect the desired bus read/write transaction information that include address, data and control signals of each data transaction into an operational profile during the program execution. The value of $N_{d\text{-}t}$ can be obtained from this step.
Step 2: $C_{p\text{-}b\text{-}err}$ = 0; $N_{inj}(j)$ = 0;
While $C_{p\text{-}b\text{-}err} < N_{p\text{-}b\text{-}err}$ do
{T_{error} can be decided by randomly choosing a number x between one and $N_{d\text{-}t}$. It means that T_{error} is equivalent to the time of the x^{th} data transaction occurring in the bus system. Similarly, A_{error} is determined by randomly choosing an address between one and S_m. A bit is randomly picked up from the word pointed by A_{error}, and the bit selected is flipped.
If ((Situation 1 occurs) or (Situation 2 occurs))
then {the injected bit error won't cause damage to the system;}
else {$C_{p\text{-}b\text{-}err} = C_{p\text{-}b\text{-}err} + 1$;
record this propagated bit error to $S_{p\text{-}b\text{-}err}(j)$ including T_{error}, A_{error} and bit location.}
//Situation 1 and 2 are described in the beginning of Section 5.2. The operational profile generated in Step 1 is exploited to help us investigate the resulting situation caused by the current bit error. From the operational profile, we check the memory access patterns beginning from the time of occurrence of bit error to identify which situation the injected bit error will lead to. //
$N_{inj}(j) = N_{inj}(j) + 1$;}
}

The Type 1 experimental process was carried out to estimate $P_{p\text{-}b\text{-}err}$, where N_{bench}, S_m and $N_{p\text{-}b\text{-}err}$ were set as the values of 4, 524288, and 500 respectively. Table 2 shows the propagation probability of bit errors for four benchmarks. It is evident that the propagation probability is benchmark-variant and a bit error in memory would have the probability between 0.866% and 3.551% to propagate the bit error from memory to system. The results imply that most of the bit errors won't cause damage to the system.

Table 2: Propagation probability of bit errors.

Benchmark	N_{inj}	$N_{p\text{-}b\text{-}err}$	$P_{p\text{-}b\text{-}err}$
M-M	14079	500	3.551%
QS	23309	500	2.145%
JPEG	27410	500	1.824%
FFT	57716	500	0.866%

Type 2 experiment: From Type 1 experimental process, we collect $N_{p\text{-}b\text{-}err}$ bit errors for each benchmark to the set $S_{p\text{-}b\text{-}err}(j)$. Those propagated bit errors are used to assess the failure probability of propagated bit errors. Therefore, $N_{p\text{-}b\text{-}err}$ simulation-based fault injection campaigns are conducted under CoWare Platform Architect, and each injection campaign injects a bit error into the memory according to the error scenarios recorded in the set $S_{p\text{-}b\text{-}err}(j)$. Therefore, we can examine the *SoC* behavior for each injected bit error.

We should point out that the function of Type 1 experiment can be accomplished by Type 2 experiment. However, Type 2 experiment is based on the simulation-based fault injection approach, which requires higher simulation time than Type 1 experiment. As can be seen from Table 2 and 4, if we use only Type 2 experiment to assess the propagation probability and failure probability of bit errors as illustrated in Table 2, 4, and 5, a huge number of simulation-based fault injection campaigns should be conducted. As a result, an enormous amount of simulation time is required to complete the injection campaigns. Instead, we developed a software tool used in Type 1 experiment to quickly identify which situation the injected bit error will lead to. Using this approach, the number of simulation-based fault injection campaigns performed in Type 2 experiment decreases dramatically. Since the performance of software tool adopted in Type 1 experiment is higher than that of simulation-based fault injection campaign employed in Type 2 experiment. Therefore, we can save a considerable simulation time. The data of Table 2 indicate that without the help of Type 1 experiment, we need to carry out a few ten thousand simulation-based fault injection campaigns in Type 2 experiment. As opposite to that, with the assistance

of Type 1 experiment, only five hundred injection campaigns are required in Type 2 experiment. Table 3 gives the experimental time of our approach and pure simulation-based fault injection approach, where the data in the column of ratio are calculated by the experimental time of our approach divided by the experimental time of pure simulation-based approach. It is evident that the performance of our experimental approach is quite effective compared to the pure simulation-based approach.

Given $N_{p\text{-}b\text{-}err}$ and $S_{p\text{-}b\text{-}err}(j)$, the Type 2 experimental results are illustrated in Table 4. From Table 4, we can identify the potential failure modes and the distribution of failure modes for each benchmark. It is clear that the susceptibility of a system to the memory bit errors is benchmark-variant, and the M-M is the most critical benchmark among the four adopted benchmarks, according to the results of Table 4.

We then manipulated the data of Table 2 and 4 to acquire the results of Table 5. Table 5 shows the probability distribution of failure modes if a bit error occurs in the memory sub-system. Each datum in the row of 'Avg.' was obtained by mathematical average of the benchmarks' data in the corresponding column. This table offers the following valuable information: the robustness of memory sub-system, the probability distribution of failure modes and the impact of benchmark on the *SoC* dependability. Probability of *SoC* failure for a bit error occurring in the memory is between 0.738% and 3.438%. We also found that the *SoC* has the highest probability to encounter the SDC failure mode for a memory bit error. In addition, the vulnerability rank of benchmarks for memory bit errors is M-M > QS > JPEG > FFT.

Table 6 illustrates the statistics of memory read/write for the adopted benchmarks. The results of Table 6 confirm the vulnerability rank of benchmarks as observed in Table 5. Situation 2 as mentioned in the beginning of Section 5.2 indicates that the occurring probability of Situation 2 increases as the probability of performing the memory write operation increases. Consequently, the robustness of a benchmark rises with an increase in the probability of Situation 2.

Table 3: Comparison of experimental time between ours & pure simulation-based approach.

Bench	Type 1 + 2 (min.)	Type 2 (min.)	Ratio
M-M	3123	15252	20.476%
QS	8353	27194	30.716%
JPEG	75968	157608	48.201%
FFT	32577	96193	33.866%
Total	120021	296247	40.514%

Table 4: Type 2 experimental results.

Benchmark	FF	SDC	CD/IT	DL	NE
M-M	0	484	0	0	16
QS	0	138	103	99	160
JPEG	0	241	1	126	132
FFT	0	177	93	156	74

Table 5: *P (2, FM(K))*, *P (2, SF)* and *P (2, NE)* for the used benchmarks.

	FF (%)	SDC (%)	CD/IT (%)	DL (%)	SF (%)	NE (%)
M-M	0.0	3.438	0.0	0.0	3.438	96.562
QS	0.0	0.592	0.442	0.425	1.459	98.541
JPEG	0.0	0.879	0.004	0.460	1.343	98.657
FFT	0.0	0.307	0.161	0.270	0.738	99.262
Avg.	0.0	1.304	0.152	0.289	1.745	98.255

Table 6: The statistics of memory read/write for the used benchmarks.

	#R/W	#R	R(%)	#W	W(%)
M-M	265135	255026	96.187%	10110	3.813%
QS	226580	196554	86.748%	30027	13.252%
JPEG	1862291	1436535	77.138%	425758	22.862%
FFT	467582	240752	50.495%	236030	49.505%

5.3 *SoC*-Level Risk Assessment

For simplicity of presentation, two components, AMBA AHB system bus and memory, are utilized to demonstrate the proposed risk model to assess the scales of failure-induced risks in a system. The

following data were used to show the risk assessment for the selected components: {*FR_C(1), FR_C(2)*} = {0.001/hour, 0.005/hour} {*S_FM(1), S_FM(2), S_FM(3), S_FM(4)*} = {10, 8, 4, 6}. According to the expressions presented in Section 3, the *SoC* failure rate, reliability and *RPN* are obtained below:

$$SFR\_C(1) = 0.001/\text{h} \times 0.7322 = 7.322 \times 10^{-4}/\text{h}$$

$$SFR\_C(2) = 0.005/\text{h} \times 0.01745 = 8.725 \times 10^{-5}/\text{h}$$

$$SFR = SFR\_C(1) + SFR\_C(2) = 8.1945 \times 10^{-4}/\text{h}$$

$$R(t) = e^{-SFR \times t}$$

$$RPN\_C(1) = 0.001/\text{h} \times ((19.41 \times 10 + 38.16 \times 8 + 7.59 \times 4 + 8.06 \times 6) \times 10^{-2}) = 5.781 \times 10^{-3}/\text{h}$$

$$RPN\_C(2) = 0.005/\text{h} \times ((0.0 \times 10 + 1.304 \times 8 + 0.152 \times 4 + 0.289 \times 6) \times 10^{-2}) = 6.387 \times 10^{-4}/\text{h}$$

$$RPN\_FM(1) = 10 \times ((0.001/\text{h} \times 19.41 + 0.005/\text{h} \times 0.0) \times 10^{-2}) = 1.941 \times 10^{-3}/\text{h}$$

$$RPN\_FM(2) = 8 \times ((0.001/\text{h} \times 38.16 + 0.005/\text{h} \times 1.304) \times 10^{-2}) = 3.5744 \times 10^{-3}/\text{h}$$

$$RPN\_FM(3) = 4 \times ((0.001/\text{h} \times 7.59 + 0.005/\text{h} \times 0.152) \times 10^{-2}) = 3.34 \times 10^{-4}/\text{h}$$

$$RPN\_FM(4) = 6 \times ((0.001/\text{h} \times 8.06 + 0.005/\text{h} \times 0.289) \times 10^{-2}) = 5.703 \times 10^{-4}/\text{h}$$

Compared *RPN_C(1)* with *RPN_C(2)*, it is evident that the failure of AMBA AHB is more critical than the failure of memory sub-system. So, the result suggests that the AHB system bus is more urgent to be protected than the memory. Moreover, the data of *RPN_FM(k)*, *k* from one to four, infer that SDC is the most crucial failure mode in this illustrated example. Throughout the above vulnerability and risk analyses, we can identify the critical components and failure modes, which are the major targets for design enhancement. In this case study, the top priority of the design enhancement is to raise the robustness of the AHB 'HADDR' bus signals to significantly reduce the rate of SDC occurrence and the scales of system risks if the system reliability/safety is not adequate.

We should notice that the case study presented here uses only two components for easy demonstration of our idea. In the future, for completeness, the work will include more components, such as ARM CPU, in the vulnerability and risk analyses.

6 Conclusions

In this work, we have presented a valuable *SoC*-level risk model, and exploited an ARM-based *SoC* platform to demonstrate its feasibility and usefulness. The main contributions of this study are first to raise the level of dependability validation to the untimed/timed functional TLM, and therefore, the efficiency of the validation process is dramatically increased; second to develop a useful risk model to assess the scales of failure-induced risks in a system; third to conduct a thorough vulnerability analysis of the AMBA bus and memory sub-systems based on a real ARM-based system platform modeled in SystemC TLM abstraction level. The analyses help us measure the robustness of the bus and memory sub-systems and locate the critical bus signals to be guarded.

Acknowledgments: The authors acknowledge the support of the National Science Council, R.O.C., under Contract No. NSC 97-2221-E-216-018. Thanks are also due to the National Chip Implementation Center, R.O.C., for their support of SystemC design tool – CoWare Platform Architect.

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*, 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 et al., "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] J.-C. Ruiz et al., "On Benchmarking the Dependability of Automotive Engine Control Applications," in *2004 Proc. IEEE Int. Conf. on*

Dependable Systems and Networks, pp. 857 – 866.

[6] G. Thorsten et al., "System Design with SystemC," Kluwer Academic Publishers, 2002.

[7] Open SystemC Initiative (OSCI), "SystemC 2.0 Language Reference Manual," Revision 1.0, www.systemc.org, 2003.

[8] CEI International Standard IEC 61508, 1998-2000.

[9] S. Brown, "Overview of IEC 61508 design of electrical/electronic/programmable electronic safety-related systems," *Computing & Control Engineering Journal*, pp. 6-12, February 2000.

[10] A. H. Mollah, "Application of Failure Mode and Effect Analysis (FMEA) for Process Risk Assessment," *BioProcess International*, pp. 12–20, November 2005.

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

[12] Y. Y. Chen, Y. C. Wang & J. M. Peng, "SoC-Level Fault Injection Methodology in SystemC Design Platform," *in 2008 7th Int. Conf. on System Simulation & Scientific Computing*, pp. 680-687.

[13] 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.

[14] Y. Y. Chen, C. H. Hsu, and K. L. Leu, "Analysis of System Bus Transaction Vulnerability in SystemC TLM Design Platform," *3rd WSEAS International Conference on Computer Engineering and Applications*, pp. 284-289, January 2009.

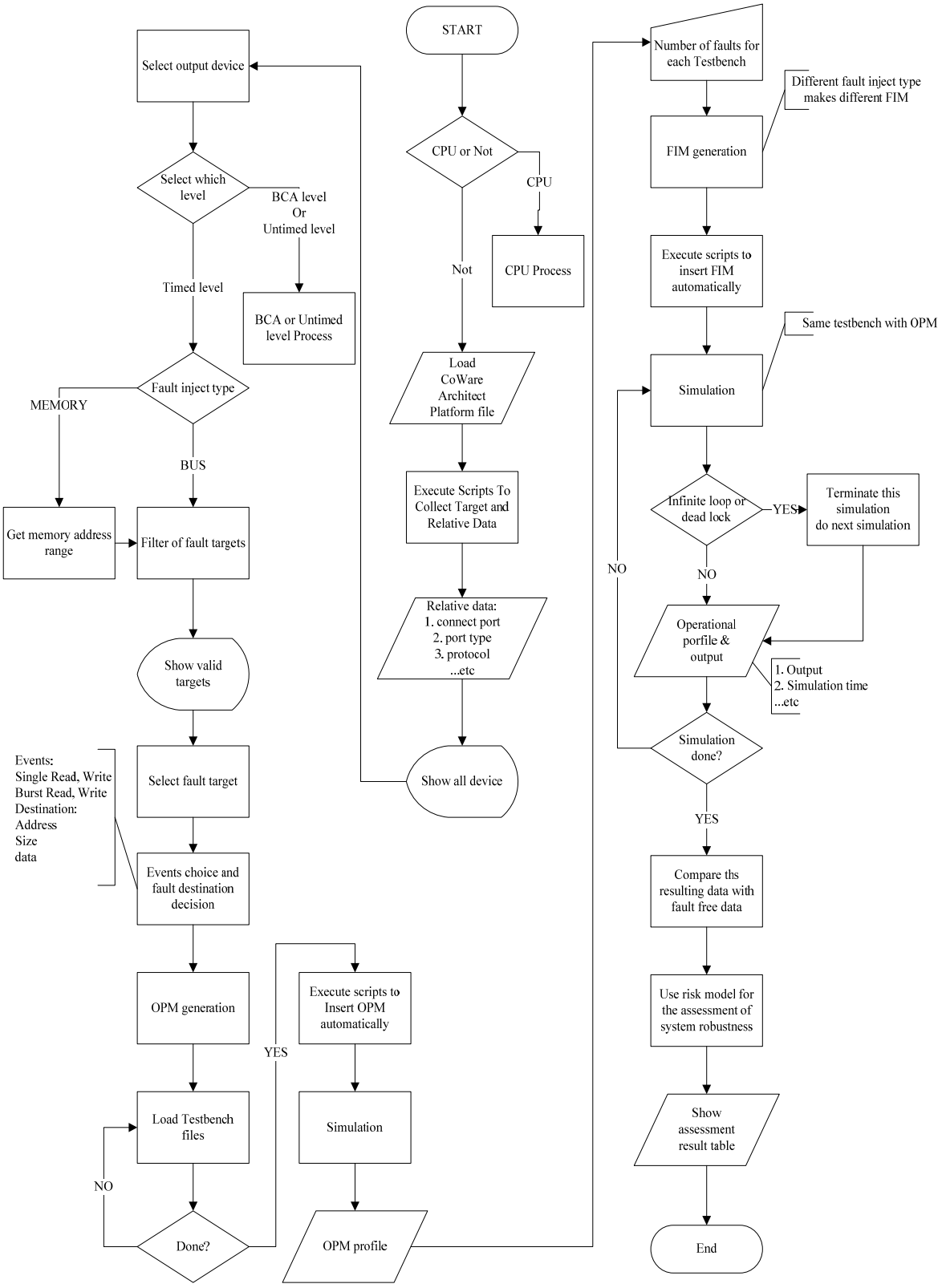

Fig. 2. The operational flow of robustness verification tool.

Self-Evaluation of Research Results:

- The above report summarizes the third-year results accomplished from this three-year research project. The research results have been published in one journal and three conferences. The extended version of the results will be submitted to be considered for journal publication. We definitely achieve the main goals set in the proposal.
- We propose a risk model at SoC level to facilitate the measure of the robustness and scales of failure-induced risks in a system, which can be used to identify the critical components and major failure modes for protection so as to effectively reduce the impact of failures to the system. We develop system-level robustness verification and risk assessment tool under the environment of CoWare Platform Architect. The tool platform takes the fault scenario description from the user and then automatically generates the system platform supplemented with the fault injection capability. Our tool can not only facilitate the failure mode and effect 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 vehicle 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 robustness and safety 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 robustness verification approach mainly focuses on the VHDL modeling level and rarely discusses the verification in SystemC-level design. Our study fulfills this lack.

Publications associated with this research:

- Yung-Yuan Chen, Chung-Hsien Hsu, and Kuen-Long Leu, "Analysis of System Bus Transaction Vulnerability in SystemC TLM Design Platform," *3rd WSEAS International Conference on Computer Engineering and Applications*, pp. 284-289, January 2009, China. (EI) NSC 97-2221-E-216-018
- Yung-Yuan Chen, Chung-Hsien Hsu, and Kuen-Long Leu, "Analysis of System Bus Transaction Vulnerability based on FMEA Methodology in SystemC TLM Design Platform," *WSEAS Transactions on Computers*, Issue 2, Vol. 8, pp. 406-416, Feb. 2009. (EI) (NSC 97-2221-E-216-018)
- Yung-Yuan Chen, Chung-Hsien Hsu, and Kuen-Long Leu, "SoC-Level Risk Assessment Using FMEA Approach in System Design with SystemC," *IEEE Symposium on Industrial Embedded Systems*, pp. 82-89, July 2009, Switzerland. (EI) (NSC 97-2221-E-216-018)

  This paper is prepared to submit the journal with an extended version.
- 陳信宇、陳永源，"系統晶片強韌度驗證分析工具平台開發" 2009 全國計算機會議(即將發表)，頁數: 12 頁，Nov. 27-28, 2009。

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

98 年 7 月 21 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
時間 會議 地點	7 月 8-10, 2009 瑞士、洛桑	本會核定 補助文號	NSC 97-2221-E-216-018
會議 名稱	（中文） （英文）IEEE Symposium on Industrial Embedded Systems (SIES 2009)		
發表 論文 題目	（中文） （英文）1. SoC-Level Risk Assessment Using FMEA Approach in System Design with SystemC 2. Robustness Investigation of the FlexRay System		

報告內容應包括下列各項：

一、　參加會議經過

此會議是在瑞士洛桑市舉行，作者是搭乘長榮班機到奧地利、維也納，再轉搭歐洲鐵路到洛桑市參加第4屆IEEE Industrial Embedded Systems會議。此次會議共有23篇長篇論文，分成6個梯次作口頭發表，每一篇論文有25分鐘的報告時間。另外有三個梯次的work-in-progress發表，每一篇論文有10分鐘的報告時間，報告結束後還有論文海報，與會者可以直接和作者討論其研究的議題。此會議的主題範圍包括了Network Analysis and Modeling, SoC Platforms, Design Methods and Real Time, Wireless Health, Networking and communications, Embedded Software, Network Design and Optimization, Reconfigurable Platforms and Industrial Control, Control and Designs。參加的學者來自美國，台灣，韓國，日本以及歐洲的國家。作者有一篇長篇論文和一篇work-in-progress論文發表，長篇論文被安排在第一天下午報告，講題是 “SoC-Level Risk Assessment Using FMEA Approach in System Design with SystemC”。在這篇論文報告後，與會學者提出了兩個問題及一些意見，這些意見具有參考的價值。另一篇work-in-progress論文被安排在第二天下午報告，講題是 “Robustness Investigation of the FlexRay System” 。在這篇論文報告後，同時有論文海報張貼，與會者直接和作者討論這個研究的議題。作者也利用此次會議的機會，和來自韓國，日本以及歐洲國家的學者交換一些研究的心得和經驗。第二天晚上會議主辦單位安排了一場晚宴，地點是在奧林匹克博物館。所以我們先參觀奧林匹克博物館，之後再到博物館二樓用餐。

二、　與會心得

此會議是一年一度 IEEE Industrial Electronics Society 組織所舉辦在 Industrial Embedded Systems 方面的會議，今年是第 4 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

會議論文集光碟片

表 Y04