Analysis of System Bus Transaction Vulnerability Based on FMEA Methodology in SystemC TLM Design Platform

YUNG-YUAN CHEN, CHUNG-HSIEN HSU, AND KUEN-LONG LEU[+]

Department of Computer Science and Information Engineering
Chung-Hua University
No. 707, Sec. 2, Wu-Fu Rd., Hsin-Chu
TAIWAN
E-mail: chenyy@chu.edu.tw, m09402067@chu.edu.tw

[+]Department of Electrical Engineering
National Central University
No. 300, Jhongda Rd., JhongLi City, Taoyuan County
TAIWAN
945401025@cc.ncu.edu.tw

Abstract: - Intelligent safety-critical systems, such as intelligent automotive systems or intelligent robots, 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. 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. In this study, we propose a useful bus system vulnerability model and present a thorough analysis of system bus vulnerability in SystemC transaction-level modeling (TLM) design level by injecting faults into the bus signals, which can assist us in predicting the robustness of the system bus, in locating the weaknesses of the bus system, and in understanding the effect of bus faults on system behavior during the SoC design phase. The impact of benchmarks on system bus vulnerability is also addressed. 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 bus dependability in early design phase for safety-critical applications.

Keywords: Fault injection, reliability, SystemC, system bus dependability, system-on-chip (SoC).

1 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]. Such influences raise the urgent need to incorporate the fault tolerance into the high-performance microprocessors, *SoC* and embedded systems for safety-critical applications [4-6]. As a consequence, it is essential to perform the failure mode and effects analysis (FMEA) procedure to locate the weaknesses of the system and provide the practical fault-tolerant strategies to improve the reliability [7]. 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 dependability 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 [8]-[12] address the fault injection issue in SystemC design platform. In our previous paper [10], 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. This scheme can only apply to RTL,

which limits the scope of applications. In [8] and [9], the authors proposed a fault injection technique based on a centralized injection control approach that is applicable to functional level and transaction layer 1 in SystemC. We devised a distributed injection control method [11] to inject the faults into the bus-cycle accurate level and untimed functional TLM with primitive channel sc_fifo. The paper [12] 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 interconnected platform for IP-based *SoC*. Clearly, the robustness of system bus has a decisive influence on 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 dependability analysis in SystemC design platform is rarely addressed except the work proposed in paper [12]. The paper [12] characterized the susceptibility of AMBA bus on errors in various signals over different transactions in SystemC cycle-accurate level. However, the approach presented in [12] is dedicated to cycle-accurate level, which may still be time-consuming in fault injection and simulation campaigns. In addition, the previous fault injection methodologies are all based on time-triggering approach to decide when to inject a fault. While the modeling levels of systems come to the untimed functional TLM and timed functional TLM, the time-triggering fault injection approach is no longer applicable to these levels or becomes improper. Instead, the event-triggering 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. Besides the above points, the results given in [12] lack the analysis of the effect of errors occurring in burst-read and burst-write transaction modes on system reliability in various benchmarks.

The types of data transaction in the system bus normally consist of the single-read, single-write, burst-read and burst-write operations. Each type of bus transaction could be incorrect or failed during its operation because some faults occur in the bus signals. So, a question arises in our analysis as to how the susceptibility of a system to the faults occurring in different bus signals over various transaction types. Consequently, the analyses in this study focus on the following three issues: 1. the effect of faults occurring in different bus signals on system dependability; 2. the effect of errors occurring in various bus transaction types on system dependability; 3. the influence of benchmarks on bus and system dependability. The analyses of issues 1 and 2 can identify the vulnerability of bus signals and bring out the susceptibility of the system to a particular data transaction type error, respectively.

In summary, the goal of this research is to propose an effective system bus fault injection framework in SystemC design platform at the abstraction levels of untimed/timed functional TLM and to comprehensively investigate the susceptibility of a system to faults on bus signals as well as the impact of bus faults on system behavior and the vulnerability of the bus system at SystemC TLM abstraction level. The results of analyses can guide us to propose a proper protection scheme for bus system. Since the modeling of systems is raised to the level of TLM abstraction, the simulation speed and fault injection effectiveness are enhanced significantly.

The remaining paper is organized as follows. In Section 2, the SystemC untimed/timed functional TLM and the concept of Transactor are presented. We propose a system bus fault injection methodology and fault injection tool in Section 3. A vulnerability model of bus system is proposed in the Section 4. The experimental results and a thorough dependability analysis are given in Section 5. The conclusions appear in Section 6.

2 SystemC 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 [13] and AMBA bus [14] 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 this paper, 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. 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.

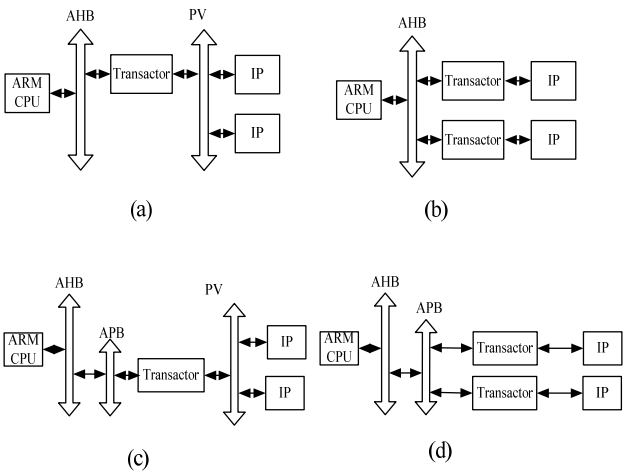

Fig. 1: ARM-based system modeled with mixed levels of PV and PV+T, where IP represents the slave module.

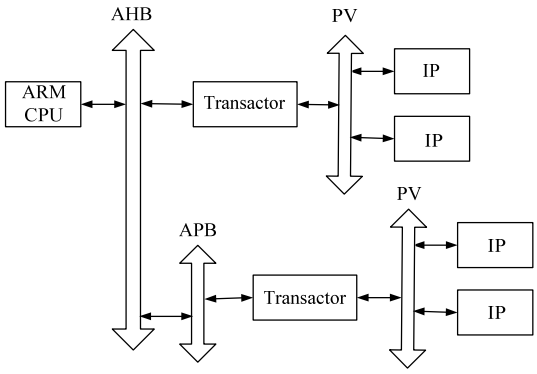

Fig. 2: An AMBA-based system modeled at PV and PV+T levels.

3 System Bus Fault Injection Scheme

An AMBA-based system as illustrated in Fig. 2 is exploited to demonstrate our system bus fault injection methodology. It is evident that the transaction error modes of the bus operation could be classified as single-read, single-write, burst-read and burst-write transaction errors. A specific transaction error mode, such as burst-read transaction error, means that a fault occurs in the bus signals during the burst-read data transaction. Therefore, if we want to perform the effect analysis of a specific transaction error mode, for example burst-read, on the system failure behavior, the event-driven fault trigger approach can be utilized to guide the fault injection to guarantee the faults injected in the bus signals during the simulation campaigns, which always lead to the burst-read transaction errors. In this case, the event of fault trigger can be set as burst-read operation of bus transactions.

A fault-triggering event is used to represent a particular type of bus transaction or a specific bus operation that decide the conditions of fault trigger/injection. In this work, we define four fault-triggering events for our injection campaigns, which are single-read, single-write, burst-read and burst-write events. It is important that using the event-driven fault injection can easily produce the desired transaction error mode and effectively characterize its effect on the system behavior. So, the event-driven fault injection approach is very suitable for the FMEA mission. Compared to event-driven fault trigger, the time-driven approach suffers from the poor injection effectiveness for a particular transaction error mode, because the time-driven fault trigger cannot guarantee the injected faults that will cause the desired transaction errors. As a result, it will degrade the

efficiency of carrying out the process of FMEA and system robustness validation.

The principal idea of our event-driven 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 to monitor the bus data transactions and check whether the declared event occurs; if yes, the fault is injected into the bus.

3.1 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 desired bus transaction information that include read/write, address, data and control signals into the operational profiles during the program execution. The operational profiles are used as a reference for generation of events embedded in the FIMs, which will be the conditions of fault trigger. The function of 'Transactor' module 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 [15] furnished by CoWare Platform Architect to implement the function of OPM. 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 each bus-read data 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 type of bus data 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 the address of master 1 to a profile;
    fstream profile_master_2("Data2.txt",ios::app);
    //gathering the address 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 create the desired fault-triggering events that decide the time instant of fault injection. Fig. 4 exhibits the event tree and its possible combinations for a bus system. Basically, the types of data transaction in the bus are read and write operations, which 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 a fault-triggering event can be a single event, like bus-read or bus-write event, or the combination of events, such as read combined with burst to form burst-read event, as illustrated in Fig. 4. The event combinations can be formed either from first-level events coupled with second-level events or from first, second then third-level events in sequence. To support the event-driven fault trigger, the OPM needs to create the desired operational profiles, which furnish the information for the generation of fault-triggering events.

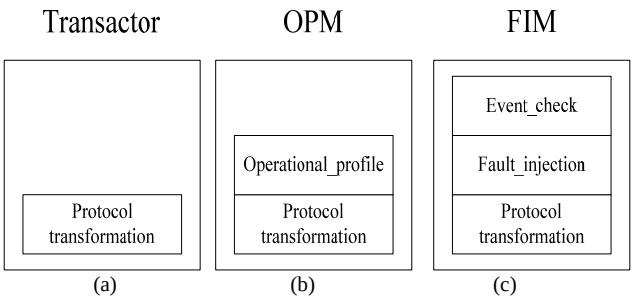

Fig. 3: Transactor, OPM and FIM functions.

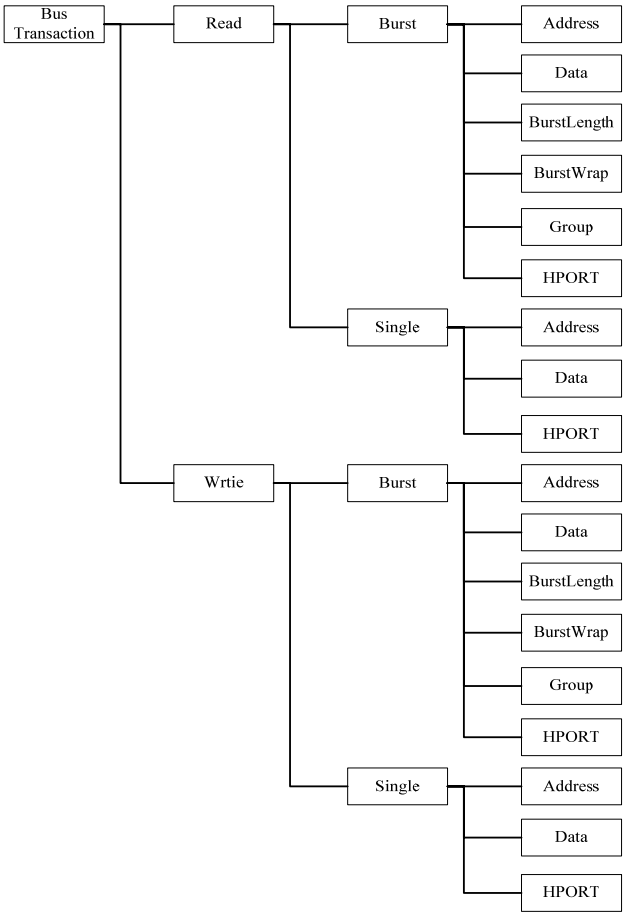

Fig. 4: Event tree and event combinations.

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 fault-triggering event, termed as burst-read event. An OPM is created to collect the information of each burst-read data transaction occurring in the bus operation.

According to the operational profile generated from the fault-free simulation campaign, assume that there are ten thousand times of burst-read data transactions occurring in the bus operation. To obtain the effect analysis of burst-read transaction errors on system operation, we need to conduct a huge amount of fault injection campaigns, saying five hundred campaigns, to obtain the solid results. The burst-read event is used here as a fault-triggering event in the fault injection campaigns. Each campaign injects a fault which is triggered when the number of burst-read transactions appearing in the bus has reached to a specific number x, where x is between 1 and 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 has reached 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, like the length data.

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

Phase 2: According to 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' by adding the following functions to it: fault-triggering event check and fault injection. The FIM replaces the 'Transactor' module 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 fault-triggering condition 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 [15] 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 bus owner & address is 0x40000000
  If (Master_ID == 2){
    If (port.getAddress()==0x80000000)
      {Fault_injection}}
  //master 2 is bus owner & address is 0x80000000
  Protocol transformation; send Transfer(); wait();}
```

3.2 Fault Injection Tool

We have created an effective fault injection tool based on the system bus fault injection methodology described in last subsection and the fault injection schemes presented in paper [11] under the environment of CoWare Platform architect [13] for dependability validation of system design with SystemC. 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: bus-cycle accurate level, untimed functional TLM with primitive channel sc_fifo, and timed functional TLM with hierarchical channel. Fig. 5 shows the flow of fault injection tool.

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 tool dramatically increases the efficiency of carrying out the system robustness validation. In the following, we briefly depict the main functions of the tool:

- 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* platforms for fault injection campaigns.

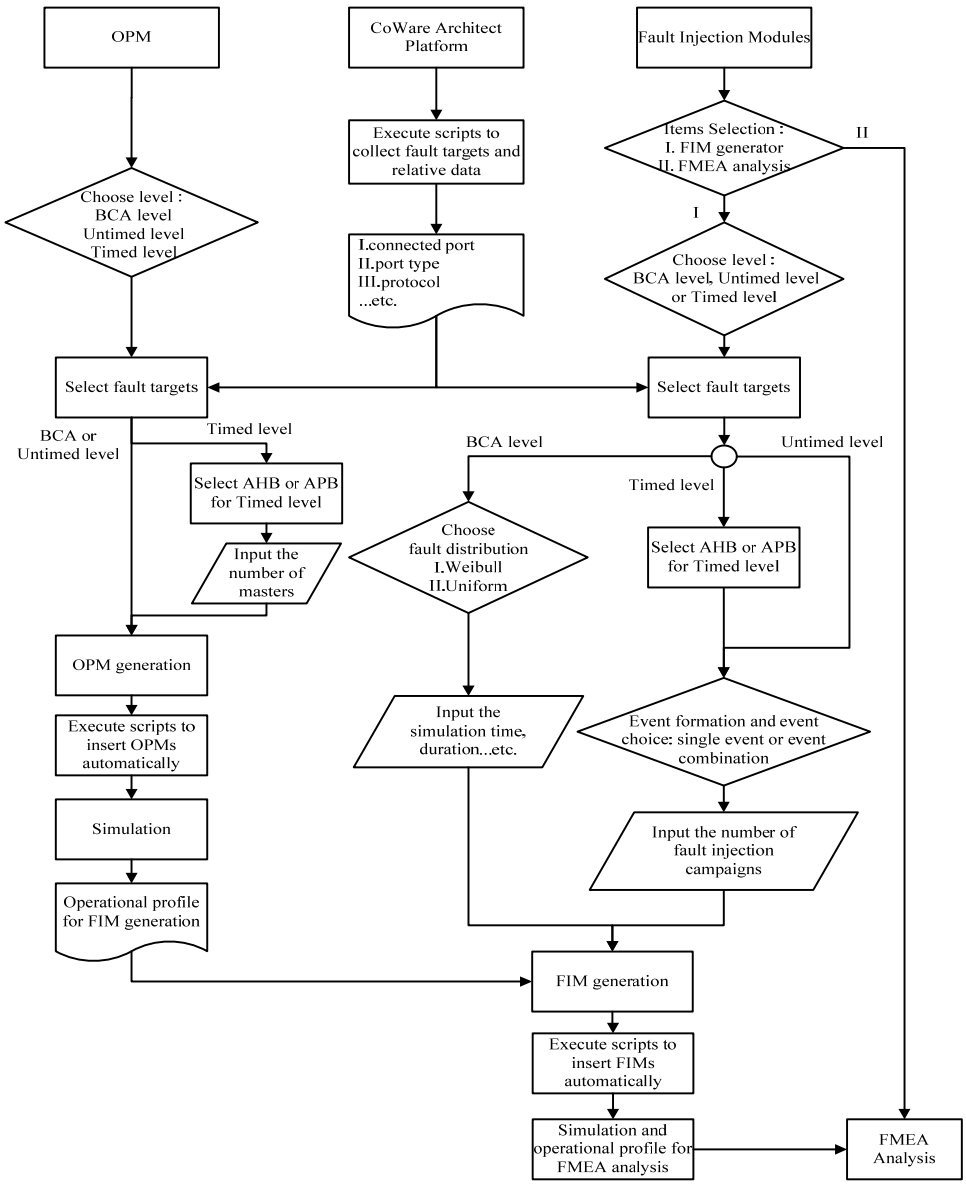

Fig. 5: The functional flow of fault injection tool.

4 Bus System Vulnerability Model

In general, there are four major data transaction modes: single-read, single-write, burst-read and burst-write offered in system bus operations. It is evident that the faults happening in the bus signals will lead to the data transaction errors and finally cause the system failures. So, in this study of system bus vulnerability, we devote our efforts to three aspects: 1. the susceptibility of a system to the faults occurring in various bus signals; 2. the susceptibility of a system to the errors occurring in various data transaction modes; 3. the influence of benchmarks on bus and system dependability.

The following notations are developed:

- x: number of categories in bus signals considered to be faulty;
- y: number of data transaction modes;
- z: number of possible failure modes of the system;
- $SC(i)$: i^{th} category of the bus signals, where $1 \le i \le x$;
- $W(SC(i))$: signal width of $SC(i)$;
- TW: total width of bus signals considered;
- $TM(j)$: j^{th} mode of data transaction, where $1 \le j \le y$;
- $N(TM(j))$: number of data transactions for $TM(j)$;
- TN: total number of data transactions in bus system;
- $FM(k)$: k^{th} failure mode of the system, where $1 \le k \le z$;
- NE: no effect which means that a fault/error occurring in the bus system has no impact on the system operation at all;
- $P(FM(K)) \mid SC(i), TM(j)$: probability of $FM(K)$ which is caused due to a fault happening in $SC(i)$ and leading to an error during the operation of $TM(j)$;
- $P(NE) \mid SC(i), TM(j)$: probability of no effect under the condition of a fault occurring in $SC(i)$ and leading to an error during the operation of $TM(j)$;
- $P_f(i, j) \mid SC(i), TM(j)$: probability of system failure which is caused due to a fault happening in $SC(i)$ and leading to an error during the operation of $TM(j)$;
- $P_f(i) \mid SC(i)$: probability of system failure which is caused due to a fault occurring in $SC(i)$;
- $P_f(j) \mid TM(j)$: probability of system failure resulting from an error happening in the operation of $TM(j)$;
- $P(FM(K)) \mid SC(i)$: probability of $FM(K)$ due to a fault taking place in $SC(i)$;
- $P(NE) \mid SC(i)$: probability of no effect for a fault taking place in $SC(i)$;
- $P(FM(K))$: probability of $FM(K)$;
- $P(NE)$: probability of no effect;
- $P(SC(i))$: probability of a fault hitting the ith category of bus signals;
- $P(TM(j))$: probability of the bus system performing the jth mode of data transaction.

The bus system vulnerability can be computed by the following equations:

$$TW = \sum_{i=1}^{x} W(SC(i)); \quad TN = \sum_{j=1}^{y} N(TM(j)) \tag{1}$$

$$P(SC(i)) = \frac{W(SC(i))}{TW}; \; P(TM(j)) = \frac{N(TM(j))}{TN} \tag{2}$$

$$P_f(i, j) \mid SC(i), TM(j) = \sum_{k=1}^{z} P(FM(K)) \mid SC(i), TM(j) \tag{3}$$

$$P_f(i) \mid SC(i) = \sum_{j=1}^{y} P(TM(j)) \times P_f(i, j) \mid SC(i), TM(j) \tag{4}$$

$$P_f(j) \mid TM(j) = \sum_{i=1}^{x} P(SC(i)) \times P_f(i, j) \mid SC(i), TM(j) \tag{5}$$

$$P(FM(K)) \mid SC(i) = \sum_{j=1}^{y} P(TM(j)) \times P(FM(K)) \mid SC(i), TM(j) \tag{6}$$

$$P(FM(K)) = \sum_{i=1}^{x} P(SC(i)) \times P(FM(k)) \mid SC(i) \tag{7}$$

$$P(NE) \mid SC(i), TM(j) = 1 - \sum_{k=1}^{z} P(FM(K)) \mid SC(i), TM(j) \tag{8}$$

$$P(NE) = 1 - \sum_{k=1}^{z} P(FM(k)) \tag{9}$$

Equations (4) and (5) represent the susceptibility of a system to the faults occurring in various bus signals and the susceptibility of a system to the errors occurring in various data transaction modes, respectively. Also from the above notations and equations, we can infer that the terms of $N(TM(j))$, TN, $P(TM(j))$, $P_f(i) \mid SC(i)$, $P(FM(K)) \mid SC(i)$ as well as $P(FM(K))$ are benchmark-variant. As a result, the bus and system dependability is benchmark-variant too.

5 Experimental Results

An ARM-based system platform provided by CoWare Platform Architect [13] was used to investigate the data transaction vulnerability of AMBA AHB. The system platform is modeled at the timed functional TLM abstraction level. We

exploit the SoC-level fault injection platform described in Section 3 to analyze the AMBA AHB vulnerability based on the equations (1) to (7). The bus signals considered in the vulnerability analysis are 'HADDR[31:0]', 'HSIZE[2:0]' as well as 'HDATA[31:0]'. The failure modes of the system identified from the fault injection campaigns are fatal failure (FF), 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 the bus vulnerability analysis.

- $x = 3$, $\{SC(1), SC(2), SC(3)\}$ = {HADDR[31:0], HSIZE[2:0], HDATA[31:0]}; $\{W(SC(1)), W(SC(2)), W(SC(3))\} = \{32, 3, 32\}$
- $y = 4$, $\{TM(1), TM(2), TM(3), TM(4)\}$ = {Burst-Read (BR), Burst-Write (BW), Single-Read (SR), Single-Write (SW)}; $N(TM(j))$, $j = 1$ to 4, can be found in Table 1;
- $z = 4$, $\{FM(1), FM(2), FM(3), FM(4)\}$ = {FF, SDC, CD/IT, DL}.

The benchmarks employed in the fault injection campaigns are: JPEG (pixels: 255 × 154), matrix multiplication (M-M: 50 × 50), quicksort (QS: 3000 elements) and FFT (256 points). Table 1 gives the statistics of the data transactions of the aforementioned benchmarks and the average statistics of the four benchmarks.

Table 1: Statistics of data transaction modes for several benchmarks and the average statistics.

	Cycles	BR	BW	SR	SW	Total
QS	3887348	470416	5	251677	30007	752105
M-M	4689502	403083	2	265196	10102	678383
FFT	8060732	1059811	29365	441430	118058	1648664
JPEG	21115374	18584	19632	1457996	325063	1821275
Avg.	9438239	487974	12251	604075	120808	1225108

Table 2 shows the results of $P(FM(K)) \mid SC(i), TM(j)$ for JPEG benchmark, where $i = 1$ to 3, $j = 1$ to 4 and $k = 1$ to 4. We omit the $P(FM(K)) \mid SC(i), TM(j)$ results for other three benchmarks for space consideration. The results of a row in Table 2 were derived from the five hundred fault injection campaigns, where each injection campaign injected 1-bit flip fault to bus signal $SC(i)$ while bus system is performing the $TM(j)$ operation. The fault duration lasts for the length of one time data transaction. The statistics derived from five hundred times of fault injection campaigns have been verified to guarantee the validity of the analysis.

The $P_f(i, j) \mid SC(i), TM(j)$, $P_f(i) \mid SC(i)$ as well as $P_f(j) \mid TM(j)$ can be calculated by expressions (3), (4) and (5). Table 3 illustrates the results of $P_f(i, j) \mid SC(i), TM(j)$, $P_f(i) \mid SC(i)$ and $P_f(j) \mid TM(j)$ for the used benchmarks. The results of Table 3 reveal that the rank of vulnerability of bus signals is 'HADDR' > 'HSIZE' > 'HDATA' for all benchmarks, and the susceptibility of the system to errors over different data transaction modes is benchmark-variant. We note that an error occurring in the SR transaction operation during the running of JPEG and FFT benchmarks has the highest probability to cause the system failure compared to errors in BR, BW and SW, whereas the susceptibility of the system to SW data transaction error is the highest for M-M and QS benchmarks.

Table 4 furnishes the data of $P(FM(K)) \mid SC(i)$ and $P(NE) \mid SC(i)$ for JPEG benchmark. It is clear that the data of Table 4 can be easily derived from the data of Table 2 and expression (6). Again, we omit the data for other three benchmarks for space consideration. The data shown in Table 4 indicate the probability distribution of failure modes for each of the signal category considered to be possibly faulty in bus system. For system running JPEG, we can observe that SDC is the failure mode that has the highest probability to occur while a fault arises in the bus signals. The other interesting phenomenon worth to mention is the 'HSIZE' and 'HDATA' faults rarely cause the FF, whereas 'HADDR' faults show a significant possibility to lead to the FF.

Based on the data of $P(FM(K)) \mid SC(i)$, $P(FM(K))$ can be evaluated by expression (7) and the results are given in Table 5. From Table 5, it is evident that the susceptibility of the system to faults is benchmark-dependent and the rank of system 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.

$P(burst \mid \text{error})$ and $P(single \mid \text{error})$ illustrated in Table 6 represent the data of the susceptibility of the system to errors happening in the burst transaction mode and single transaction mode, respectively; similarly, $P(read \mid \text{error})$ and $P(write \mid \text{error})$ shown in Table 6 represent the data of the susceptibility of the system to errors happening in

the read operation and write operation in the bus system, respectively. *P (burst | error)* can be computed by the following expression:

$$P\ (burst \mid \text{error}) = \frac{N(TM(1))}{N(TM(1)) + N(TM(2))} \times (P_f(1) \mid TM(1)) + \frac{N(TM(2))}{N(TM(1)) + N(TM(2))} \times (P_f(2) \mid TM(2)) \quad (10)$$

Other three probability expressions can be obtained similarly as expression (10). The salient points observed from Table 6 are: first is the system is more sensitive to errors arising in single data transaction than errors arising in burst data transaction; second is the system is more sensitive to errors arising in write data operation than errors arising in read data operation except the JPEG benchmark.

Table 7 offers the average dependability statistics over four benchmarks employed in the validation process. The data of $P_f\ (i, j) \mid SC(i), TM(j)$ in Table 7 were obtained by mathematical average of $P_f\ (i, j) \mid SC(i), TM(j)$ for all benchmarks. Then, the rest of the data in Table 7 can be computed using the equations presented in Section 4. Since the severity of system vulnerability is benchmark-dependent, the results of Table 7 give us the expected probabilities for the vulnerability of the developing system, which are very valuable for us to gain the robustness of the bus system and the critical bus signals to be protected. With reference to Table 7, for bus signals, we see that the 'HADDR' is the top priority to protect; for data transaction modes, single-read as well as single-write are the ones to be protected. The robustness measure of the bus system is only 0.2678, which means that a fault occurring in the bus system, the application system 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 application system responding to the bus faults or errors.

Table 2: *P (FM(K)) | SC(i), TM(j)* results for JPEG benchmark.

	FF(%)	SDC(%)	CD/IT(%)	DL(%)	NE(%)
HADDR, BR	42.4	30.4	0.2	7.6	19.4
HADDR, BW	42.0	23.2	0.2	2.2	32.4
HADDR, SR	38.4	42.0	0.0	14.0	5.6
HADDR, SW	40.6	49.0	0.4	1.0	9.0
HSIZE, BR	0.0	57.4	1.4	10.6	30.6
HSIZE, BW	0.0	30.0	0.2	2.0	67.8
HSIZE, SR	0.2	69.8	0.2	14.2	15.6
HSIZE, SW	0.0	64.0	0.4	0.8	34.8
HDATA, BR	0.0	46.8	0.4	20.6	32.2
HDATA, BW	0.0	60.2	0.0	12.0	27.8
HDATA, SR	0.0	48.2	0.2	25.4	26.2
HDATA, SW	0.0	39.6	0.4	0.0	60.0

Table 3: $P_f\ (i, j)|SC(i),TM(j)$, $P_f\ (i)|SC(i)$ and $P_f\ (j)|TM(j)$ for the used benchmarks.

JPEG	BR (%)	BW (%)	SR (%)	SW (%)	$P_f\ (i)\|SC(i)$
HADDR	80.6	67.6	94.4	91.0	93.36%
HSIZE	69.4	32.2	84.4	65.2	80.26%
HDATA	67.8	72.2	73.8	40.0	67.69%
$P_f\ (j)\|TM(j)$	73.99	68.21	84.11	65.49	
M-M	BR (%)	BW (%)	SR (%)	SW (%)	$P_f\ (i)\|SC(i)$
HADDR	79.4	45.8	98.6	99.4	87.20%
HSIZE	74.4	48.8	87.0	94.4	79.62%
HDATA	56.8	0.0	94.8	98.2	72.27%
$P_f\ (j)\|TM(j)$	68.38	24.06	96.27	98.60	
QS	BR (%)	BW (%)	SR (%)	SW (%)	$P_f\ (i)\|SC(i)$
HADDR	65.8	64.6	76.4	98.2	70.64%
HSIZE	64.6	50.6	76.4	66.2	68.61%
HDATA	26.2	25.0	56.4	97.0	39.13%
$P_f\ (j)\|TM(j)$	46.83	45.06	66.85	96.19	
FFT	BR (%)	BW (%)	SR (%)	SW (%)	$P_f\ (i)\|SC(i)$
HADDR	73.8	78.6	88.4	80.4	78.27%
HSIZE	65.4	59.4	84.2	60.6	69.98%
HDATA	37.2	77.6	66.4	67.2	47.89%
$P_f\ (j)\|TM(j)$	55.94	77.26	77.70	73.21	

Table 4: *P (FM(K))|SC(i)* and *P (NE)|SC(i)* for JPEG benchmark.

	FF	SDC	CD/IT	DL	NE
HADDR	38.87%	42.93%	0.08%	11.49%	6.64%
HSIZE	0.16%	68.21%	0.25%	11.64%	19.74%
HDATA	0.00%	46.78%	0.24%	20.67%	32.31%

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

	FF (%)	SDC (%)	CD/IT (%)	DL (%)	NE (%)
JPEG	18.57	45.90	0.16	15.88	19.49
M-M	18.95	55.06	2.15	3.57	20.27
QS	20.06	17.52	12.24	5.67	44.50
FFT	20.18	21.09	15.74	6.38	36.61

Table 6: *P (burst* | error*)* vs. *P (single* | error*)* and *P (read* | error*)* vs. *P (write* | error*)*

	P (burst | error*)*	*P (single* | error*)*	*P (read* | error*)*	*P (write* | error*)*
JPEG	71.02%	80.72%	83.99%	65.64%
M-M	68.38%	96.35%	79.45%	98.59%
QS	46.83%	69.97%	53.81%	96.19%
FFT	56.52%	76.76%	62.34%	74.02%

Table 7: Average dependability statistics over four benchmarks.

	BR (%)	BW (%)	SR (%)	SW (%)	$P_f(i)\|SC(i)$
HADDR	74.90	64.15	89.45	92.25	83.68%
HSIZE	68.45	47.75	83.00	71.60	75.73%
HDATA	47.00	43.70	72.85	75.60	62.53%
$P_f(j)\|TM(j)$	61.29	53.65	81.23	83.37	
	FF (%)	SDC (%)	CD/IT (%)	DL (%)	NE (%)
	19.41	38.16	7.59	8.06	26.78
	P (burst | error*)*	*P (single* | Error*)*	*P (read* | error*)*	*P (write* | error*)*	
	61.10%	81.59%	72.32%	80.64%	

6 Conclusions

In this work, we have presented a valuable dependability model for *SoC* bus system, and exploited an ARM-based application system 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 devise an effective system verification platform under the CoWare Platform Architect to assist us in performing the fault injection and simulation campaigns. Therefore, the efficiency of the validation process is dramatically increased; second to develop a useful dependability model to analyze the robustness of the bus system; third to conduct a thorough vulnerability analysis of the AMBA bus system based on a real ARM-based system platform modeled in SystemC TLM abstraction level. The analyses help us measure the robustness of the bus system 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 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 et al., “Impact of soft error challenge on *SoC* design,” in *2005 Proc. 11th IEEE Int. On-Line Testing Symposium*, pp. 63-68.

[4] Short, M., Schwarz M. and Boercsoek J.: ‘Efficient Implementation of Fault-Tolerant Data Structures in Embedded Control Software’, *WSEAS TRANSACTIONS on ELECTRONICS*, 5, (1), January 2008, pp. 12-24.

[5] Hahanov, V., Hahanova, A., Chumachenko, S. and Galagan, S.: ‘Diagnosis and Repair Method of SoC Memory’, *WSEAS TRANSACTIONS on CIRCUITS AND SYSTEMS*, 7, (7), July 2008, pp. 698-707.

[6] Hahanov, V., Obrizan, V., Litvinova, E. and Man, K. L.: ‘Algebra-Logical Diagnosis Model for SoC F-IP’, *WSEAS TRANSACTIONS on CIRCUITS AND SYSTEMS*, 7, (7), July 2008, pp. 708-717.

[7] 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.

[8] K. Rothbart et al., “High level fault injection for attack simulation in smart cards,” in *2004 Proc. 13th Asian Test Symposium*, pp. 118-121.

[9] K. Rothbart et al., “A smart card test environment using multi-level fault injection in SystemC”, in *2005 Proc. 6th IEEE Latin-American Test Workshop*, pp. 103-108.

[10] 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.
[11] 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.
[12] I. C. Lin, S. Srinivasan, & N. Vijaykrishnan, "Transaction level error susceptibility model for bus based *SoC* architectures," in *2006 7th Int. Symposium on Quality Electronic Design*, pp. 775-780.
[13] CoWare Model Library, "Platform Creator User's Guide," Product Version V2006.1.2.
[14] CoWare Model Library, "AMBA Bus Library," Product Version V2006.1.2.
[15] CoWare Model Library, "SystemC Modeling Library Manual," Product Version V2006.1.2.