An Implementation of Simulation-Based Environment for Bus Fault Injection Techniques

Biaobiao Shi, Xiaopeng Gao
School of Computer Science and Engineering
Beihang University
Beijing, China
{shibiaobiao, gaoxiaopeng}@les.buaa.edu.cn

***Abstract*—Reliability is the most important feature in this more and more complex computer system era. Fault injection is dependability validation technique to evaluating the system. Hardware and Software implementations of fault injection have a long history and are much more mature than simulated fault injection. In this paper, we compare the differences between these three types of fault injections at first. Then, we identify and understand the types of fault. We design a low-cost, simulation-based fault injection system and design experiments to verify the correctness.**

Keywords-simulation fault injection; reliability;

I. Introduction

Reliability, the ability of a system or component to perform its required functions under stated conditions for a specified period of time[1], is the key feature for a series of success on system engineering. Built-In Test(BIT) technology is one of the most important mechanisms that permit a machine to test itself and Fault injection is one of the core technologies in BIT.

Fault injection technology generates the appropriate fault test cases by actual scene, selects the suitable fault injection tests to proceed BITs and finishes the entire integrity tests. To do prototype-based fault injection, hardware level and software level are traditional methods[2]. software fault injection mainly based on modification of the program being run by the system under study, so it can emulate software defects as well; while hardware fault injection methods generally use direct contact with circuit pins, intercept and alter the electrical signals at the pins, or inject faults by creating heavy-ion radiation, which is known as injection without contact. However, the disadvantages on both of them list below:

Firstly, the real target physical platforms are needed and they are not convenient enough to have tests. Secondly, the computer systems with highly-integrated, multichip hybrid, tightly encapsulated circuits limit the accessible of physical fault injection. Thirdly, the physical fault injection can not be used, judged by availability and Mean Time Between Failures(MTBF) in design process.

Luckily, simulated method on fault injection is proposed by [3]. Simulation method has the advantages of relatively uninhibited access to a modeled system's internal nodes. The ability to precisely control and monitor injected faults, coupled with low-cost computer automation, and the potential for earlier application make simulated injection an attractive alternative to physical injection.

Table 1 compares these methods mentioned above. It's obviously that simulation fault inject is a remarkable method for fault injection.

TABLE I. MERIT AND DEMERIT OF DIFFERENT INJECTION METHOD

	Simulation Injection	**Software Injection**	**Hardware Injection**
Available Time	During the process of System Design	After System Design finished	After System Design finished
Inject level	From High level to Low level	High level available	Low level available
Implementation Cost	High	Low	High
Integrity	Yes	Yes	Probably No
Real time	Low	Low	High
Fault Coverage Rate	Low	A bit low	High

The researches on simulation fault injection based on Bus device are not popular for the limitation on the details implementation of target platform, inaccurate sequential control and slow runtime environment.

In this paper, we design a method on simulated fault injection to VME bus. VME bus[4] is a universal asynchronous bus, it defines a system based which computing components can transfer data access, store data, and connect peripheral devices, in a closely coupled hardware architecture. With a feature of high speed and high reliability, VME bus widely used in military field and aerospace field.

We analyze the three fault type and give expression on fault model. A simulated fault inject system on VME is designed and the experiments on verifying the correctness is exhibited at the last of the paper.

The structure of this article is as follows. After introducing the related work in section 2, we give fault expression in section 3. Then, a global design and details design of our simulated fault injection system are introduced in section 4. We have experiments on verifying correctness in section 5.

II. Related Work

There are some studies on simulation fault injection. FAUmachine[5] [6] which is capable of injecting faults into emulated PC hardware. FAUmachine enhances the concept of a virtual machine to simulate complex systems with both capabilities of fault-injection and a framework to conduct large experiments in an automatic manner.

Saboteur [7] is a specific module of Simics. Not only fault injection can inject in several locations, but also faults can target a specific processor register, the data bus during a memory or I/O (Input/Output) operation, or the address bus during a memory operation.

These two studies can only inject in logic level of simulator and are they all limited by the encapsulation of their simulator's infrastructure, since they did not amend the functional models in the simulator. This problem is particularly serious for the bus injection, for the bus architecture is always simplified a lot in simulators for ease of use, hence reduce the feasibility of fault injection to multiple levels of detail model.

III. Fault Expression

In this section, we analyze types of the bus fault model based on simulation.

After analyzing the reason of fault, we find out that the descriptions of Bus-Level Fault can be represented below:

- Fault Component: Bus;
- Fault Locations: Data-Bus, Address-Bus and Control Bus;
- Fault Types: Data Substitution, Bit Error and Control Fault
- Fault Arguments: Different Fault types induce different arguments. Such as fault address, fault occurrences times and so on.
- Trigger Events: Under the Trigger Injection situation, the events which cause the injection activity.
- Fault Timeliness: Permanent Fault, Transient Fault and Intermittent Fault.

The trigger events can be listed:

- At the beginning of working loads start.
- Read/Write period of Bus.
- Specified Signal, such as interruption requires.
- Access of special address.
- Specified events, such as the specified address appears *n* times at Read period.

These fault features can be represents into a tuple. Let M represents a fault type which show as:

M = {component, target, model, params, trigger, time}

IV. The Framework of Simulation Fault Injection

We introduce the framework of Simulation-based fault injection in this section, including global design and details.

A. Global Design

Fig. 1 is the global design of our simulation-based fault injection. The bus Fault Injection framework, it consists of VME_HOST failure module (Saboteur), Fault management module (Fault Manage), and a single Fault Injection Console (Injection Console). Injection Console use a piece of Shared memory to store the fault instruction; Fault Manage, monitor and read the data of the same region, and load the Fault injection instruction to the relevant the module on which the fault happens. The VME_HOST bus transmission,

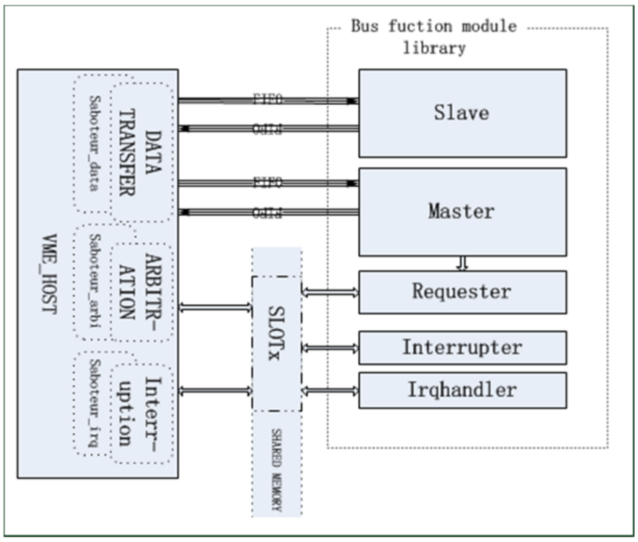


Figure 1. Framework of our Simulation-based fault injection system

bus interrupt and bus arbitration module, are all equipped with the corresponding failure module, in accordance with the circumstances, it active the related fault according to the fault injection instruction to complete fault injection process.

B. Details of Design

1) Data Transfer: first of all, the master activates the attached request unit to issue a Bus request. When it gets Bus cycle successfully, it starts a DTB cycle (Data Transfer Bus cycle) and sends the corresponding Data request. VME Bus support 64 address, 32 bits of Data Bus Data transmission. As shown in figure 4.1 shows, when master request unit get the access of the bus successfully, it will write data-read/write instructions to output pipe.

2) Data Arbitration: VME bus system complete bus arbitration process by the interaction between the circuit boards request unit and the system board bus arbitration device. VME bus support three types of request unit, it is respectively: Release-When-Done (RWD), Release-On-Request (ROR), FAIR. RWD type request unit, is the most basic bus request unit, it will release the control of the bus when Master complete a bus transmission (directly write: unless some other requester on the bus drives one of the bus request lines low); ROR requesters reduce the number of arbitrations initiated by a master that is generating a large percentage of the bus traffic. In systems with more than four masters or interrupt handlers, fairness can be provided by FAIR requesters. After it has been granted the bus, the FAIR requester refrains from requesting the bus again as long as there are any active bus requests pending on its own request level. And three types of arbitration algorithm are supported: Prioritized(PRI), Round-robin-select(RRS) and Single-level(SGL).

3) Interrupt Handling Process: Interrupt request is proposed by the IRQ1* - IRQ7* ,when the interrupt handler monitor the interrupt request, it first get system bus, and then send interrupt response signal IACK *, start interrupt response signal of the Daisy chain, and send the interrupt number to the bus address line (A0 to A3), interrupt module get the IACKIN * (effective), check A01 - A03 D0, D1 to find whether the number is the same with its own interrupt number . If it is the same, interrupt module will send its interrupt vector/STATUS code (STATUS/ID) on the data line, and at the same time, drive DTACK *. When interrupt handler detect DTACK * is effective, it will read interrupt vector/STATUS code (STATUS/ID), and then execute interrupt handler. In the realization of this paper, VME_HOST and request unit, interrupt handler.

V. EXPERIMENTS

In this section, we verify the simulation correction of data transfer in sub-bus, bus arbitration and bus interruption.

A. Experiments system

As Fig. X. shows, SLOT0 is the VME_HOST system board which employs Prioritized(PRI) as arbitral method. In order to exhibit arbitral algorithm, we set two seconds pause between arbitral activities which enlarges the bus occupation period. One second pause between the bus transactions in SLOT1, SLOT2 and SLOT3 three simulated boards with the master module is set to simulate the activity of bus intermittent transfer. The priorities irqhandler in SLOT2 monitors are 4, 3, 2, 1. It monitors the interrupt module with PRI 3 in SLOT1 and module with PRI 4 in SLOT2. The priorities irqhandler in SLOT3 monitors are 7, 6 5. It monitor the interrupt module with PRI 7 in SLOT3.

TABLE II. TABLE TYPE STYLES

	master	**requester**	**interrupter**	**Irqhandler**	**slave**
SLOT 1	✓	ROR，PRI:1	PRI:3	×	A64(0x0000,0001,0000,0000~ 0x0000,0001,0000,FFFF)
SLOT 2	✓	FAIR, PRI:2	PRI:4	PRI of Irq:4,3,2,1	A32(0x9002,0000~0x9002,ffff)
SLOT 3	✓	RWD，PRI:2	PRI:7	PRI of Irq:7,6,5	A32(0x9003,0000~0x9003,ffff)

The details bus activity of simulation boards is:

- Firstly, the programs on VME_HOST run. VME bus system initializes. After one second, the simulation board SLOT1, SLOT2 and SLOT3 are online.
- SLOT1: writes 32 bits data 0x1111,1111 at the VME bus address 0x9002,0000~0x9002,0003; Then, pauses one second; After that, reads the data from the address 0x9002,0000~ 0x9002,0003; At the time of 14 seconds, sends Bus irq request with PRI 3.
- SLOT2: begin the BLT write transaction with start address 0x9003,0000 on VME bus address; Then, pauses one second; After that, begin the BLT read transaction with start address 0x9003,0000 on VME bus address; At the time of 14 seconds, sends Bus irq request with PRI 4.
- SLOT3: writes 32 bits data 0x3333,3333 at the VME bus address 0x0000,0001,0000,0000~0x0000,0001,0000,0003; Then, pauses one second; After that, reads the data from the address 0x0000,0001,0000,0000~0x0000,0001,0000,0003; At the time of 14 seconds, sends Bus irq request with PRI 7.

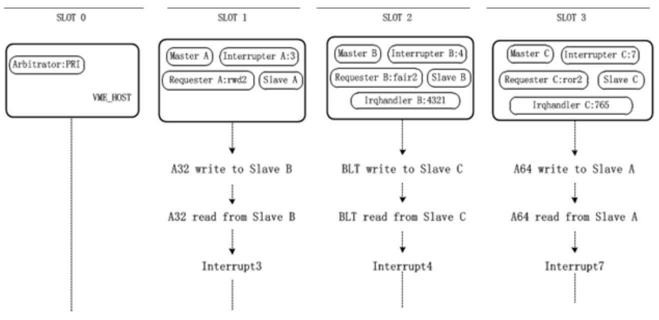
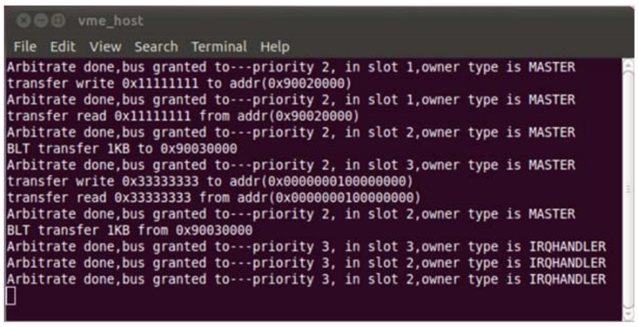


Figure 2. The Deployment for Bus Function Verification

B. Experiments results

1) VME Bus Function Test: These log messages in Figure 3 shows the whole process when the VME_HOST deals with all the transactions in the VME bus. During the arbitration procedure，all the three SLOTs call request, for they are at same level of request priority, so the daisy chain arbitration algorithm in the VME_HOST responds the request in the closest SLOT, i.e., SLOT1. Bus arbitrated twice and transferred twice; then SLOT2 got the bus, for requester in SLOT2 is a FAIR type, after one transaction, the bus is granted to SLOT3, and SLOT2 have not get the bus until SLOT3 transfer all its transaction; the feature of ROR requester result in the continuous twice transactions without give up and reapply bus, for there is no other requester apply for bus. During the interruption procedure, the irqhandler in SLOT3 respond the priority 7 interrupter request, then the irqhandler in SLOT2 respond priority 4 and 3 in SLOT2 and SLOT1, this sequence coincide their priority order. The whole process verify the data transfer function , arbitration, interruption handling algorithm all work properly in VME bus standard.



```
Arbitrate done,bus granted to---priority 2, in slot 1,owner type is MASTER
transfer write 0x11111111 to addr(0x90020000)
Arbitrate done,bus granted to---priority 2, in slot 1,owner type is MASTER
transfer read 0x11111111 from addr(0x90020000)
Arbitrate done,bus granted to---priority 2, in slot 2,owner type is MASTER
BLT transfer 1KB to 0x90030000
Arbitrate done,bus granted to---priority 2, in slot 3,owner type is MASTER
transfer write 0x33333333 to addr(0x0000000100000000)
transfer read 0x33333333 from addr(0x0000000100000000)
Arbitrate done,bus granted to---priority 2, in slot 2,owner type is MASTER
BLT transfer 1KB from 0x90030000
Arbitrate done,bus granted to---priority 3, in slot 3,owner type is IRQHANDLER
Arbitrate done,bus granted to---priority 3, in slot 2,owner type is IRQHANDLER
Arbitrate done,bus granted to---priority 3, in slot 2,owner type is IRQHANDLER
```

Figure 3. Log messages in VME_HOST

2) *Fault Injection Test:* We design this test to check the fault injection function in our system. We start the VME_HOST first as usual, then after one second,we set up a board, which deployed with a RWD type request in PRI 2, a interrupter in PRI 7, a irqhandler monitoring PRI 7,6,5. It firstly write data(0x11111111) to address (0x90001100),then it reads from address(0x90002200),after that, it calls for a interrupt.

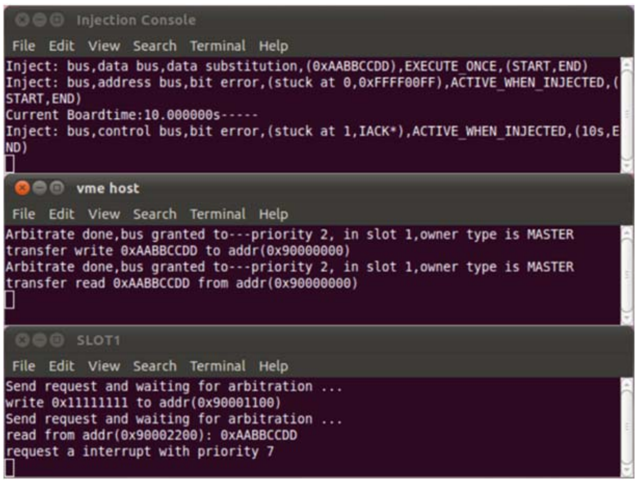
Figure 4 shows the log messages which are run the injection console, inject three typical faults, load from configure file. The configure file shows below:

{bus, data bus, data substitution, (0xAABBCCDD) , execute once,(start-end)}

{bus, address bus, bit error, (stuck at 0,0xFFFF00FF), active when injected, (start-end)}

{bus, control bus, bit error, (stuck at 1, IACK*) , active when injected, (10s-end)}

In Figure 4, we can find that the injection console injects all the fault instructions by sequence. Then during the first write transaction, the write instruction that SLOT1 write 0x11111111 to (0x90001100) was injected with two fault, data was substituted to 0xAABBCCDD, and the address turned into (0x90000000) by stuck-at 0 error, as we see in the VME_HOST log message. During the second transaction when SLOT1 read from(0x90002200), the address was changed to (0x90000000), so it got 0xAABBCCDD, as what showed in SLOT1 log message. The first two fault injection turn out to be successful. Finally when SLOT1 call for a interrupt request, it was not responded for the stuck at 1 on IACK* shut down the interrupt handling process, this consequence verified the success of the third fault injection. Hence we got the verification results of fault injection function of our system.


Injection Console
File Edit View Search Terminal Help

```
Inject: bus,data bus,data substitution,(0xAABBCCDD),EXECUTE_ONCE,(START,END)
Inject: bus,address bus,bit error,(stuck at 0,0xFFFF00FF),ACTIVE_WHEN_INJECTED,(
START,END)
Current Boardtime:10.000000s-----
Inject: bus,control bus,bit error,(stuck at 1,IACK*),ACTIVE_WHEN_INJECTED,(10s,E
ND)
```

vme host
File Edit View Search Terminal Help

```
Arbitrate done,bus granted to---priority 2, in slot 1,owner type is MASTER
transfer write 0xAABBCCDD to addr(0x90000000)
Arbitrate done,bus granted to---priority 2, in slot 1,owner type is MASTER
transfer read 0xAABBCCDD from addr(0x90000000)
```

```
Send request and waiting for arbitration ...
write 0x11111111 to addr(0x90001100)
Send request and waiting for arbitration ...
read from addr(0x90002200): 0xAABBCCDD
request a interrupt with priority 7
```

Figure 4. Log messages in fault injection test

VI. CONCLUSION

Fault injection has become a valuable asset for evaluating computer system dependability. It has been wildly-used by hardware and software injection. Simulation method has the advantages of relatively uninhibited access to a modeled system's internal nodes. The ability to precisely control and monitor injected faults, coupled with low-cost computer automation, and the potential for earlier application make simulated injection an attractive alternative to physical injection.

In this paper, we compare the differences between these three types of fault injections first. Then, identify and understand the types of fault. We design a low-cost, simulation-based fault injection system and verify the correctness of this VME simulated fault injection.

REFERENCES

[1] Institute of Electrical and Electronics Engineers (1990) IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY ISBN 1-55937-079-3

[2] Fault Injection techniques and tools

[3] Mei-chen Hsueh, Timothy K. Tsai, Ravishankar K. Iyer. Fault Injection Techniques and Tools.IEEE Computer - COMPUTER , vol. 30, no. 4, pp. 75-82, 1997

[4] American National Standard for VME64

[5] S. Potyra, V. Sieh, and M. Dal Cin. 2007. Evaluating fault-tolerant system designs using FAUmachine. In Proceedings of the 2007 workshop on Engineering fault tolerant systems (EFTS '07). ACM, New York, NY, USA, , Article 9 .

[6] FAUmachine Team. FAUmachine. URL: http://www.FAUmachine.org/, 2003–2007.

[7] Yangyang Yu, B. Bastien, B. W. Johnson.2005.A state of research review on fault injection techniques and a case study.Reliability and Maintainability Annual Symposium - RAMS , pp. 386-392, 2005