CHEMICAL ENGINEERING TRANSACTIONS
VOL. 33, 2013
A publication of AIDIC
The Italian Association of Chemical Engineering
Online at: www.aidic.it/cet
Guest Editors: Enrico Zio, Piero Baraldi

ISBN 978-88-95608-24-2; **ISSN** 1974-9791
DOI: 10.3303/CET1333101

Research on the Simulation-based Fault Injection Design with Consideration of Board-level Built-In Test

Yi Li [a], Ping Xu [a], Han Wan *[,b]

[a]Key Lab of Science & Technology on Reliability & Environment Engineering, Beihang University, No.37 of Xueyuan Rd., Haidian District, Beijing 100191, China
[b]State Key Laboratory of Virtual Reality Technology and Systems, Beihang University, No.37 of Xueyuan Rd., Haidian District, Beijing 100191, China
wanhan@buaa.edu.cn

As embedded system is widely used in setting up avionic devices, it's important to verify and validate the system's reliability. With the help of Built-In Test (BIT) technology, faults can be reported and isolated. Especially the software BIT, which is a popular solution to test the dependability of embedded boards and the entire system. Furthermore, the demand for dependable BIT software becomes more urgent. This paper discusses the general requirements for BIT software testing including fault mode analysis, and then presents a simulation platform which simulates embedded hardware boards and the external environments, such as bus devices, Ethernet, and etc. At the same time, hardware faults and I/O data faults are modelled, simulated and injected in this simulation system. Those contributed a simulated BIT software operating system to perform board-level BIT software testing. Compared to traditional fault injection tools, this system takes advantage of flexibility based on the software simulation technology and does no harm or irruption to either the real hardware or the software. In addition, this system can be expanded with more other fault modes including both on-board and on-bus using the simulation method mentioned in this paper. In the future, it's expected to build a robust simulation-based fault injection system that satisfies all the general requirements in BIT software testing and this method can be applied to the further research on PHM system validation.

1. Introduction

Testability products are designed to determine their status (working, not working or performance degradation) timely and accurately, and have the characteristics to isolate the internal fault. With the development of the system, as well as equipment performance increased, testing attracts more and more attention. Built-In Test (BIT) can detect and isolate system or device's internal fault automatically. As airborne electronic equipment increased, and machine control becomes highly centralized, BIT achieves an unprecedented importance. Testability research hot spot lies in the study of BIT and the further development of its derivatives, such as prognostic and health management.

BIT software testing is regarded as a particular kind of embedded software testing, which associates with its operating environment (embedded hardware platform) closely. And fault injection method can be introduced for BIT software testing purpose. As far as known, injecting faults to the system that incorporates BIT manually is an effective way for BIT software validation. After observing whether BIT can detect and isolate the faults, we can find out that how the software BIT matched with the design requirements, and the suggestion of the improvements for the design can be made.

In this paper we suggest to develop and build a simulation environment with fault injection capability which is designed target for board-level software BIT operating environment, and this system can be used as a simulation and fault injection platform for BIT software testing.

The rest of this paper is organized as follows. Section II summarizes the general requirements for BIT software testing, as well as, the analysis of fault injection requirement. After that, the implementation of the simulation and emulation environment together with fault injection mechanism are described in Section III.

Please cite this article as: Li Y., Xu P., Wan H., 2013, Research on the simulation-based fault injection design with consideration of board-level built-in test, Chemical Engineering Transactions, 33, 601-606 DOI: 10.3303/CET1333101

And then, Section IV demonstrates fault injection cases designed to evaluate the software BIT with considering the test credibility factors. Finally, Section V concludes the paper and prospects future work.

2. Fault injection for board-level BIT software testing

Generally, the Board-level software BIT's workflow and test process that belongs to it are described in Figure 1. This section mainly discusses the requirements for board-level BIT software testing, in other words, analysis how to perform fault injection.

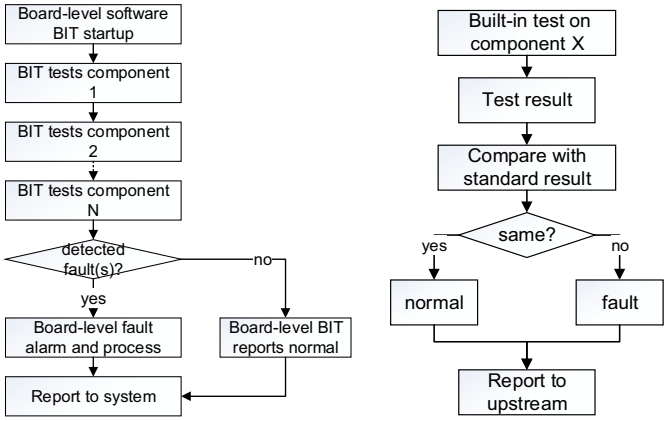

Figure 1: Board-level software BIT process and test flow

2.1 Board-level BIT software testing requirements

Board-level software BIT is designed for fault detection and fault isolation. So when faults occur on hardware system, it's critical for BIT software to detect all of them and report their fault modes correctly. Regarding to the function requirements and design patterns of software BIT, a complete BIT test should contain two aspects: whether software BIT could detect all the possible hardware fault modes of system correctly, and whether software BIT false alarm the hardware faults that do not exist in fact.

Table 1: BIT software testing requirements

Function overview	Functional testing requirements
state monitor	real-time monitoring the key parameters of the system record state information correctly
fault detection	detect fault in proper way reasonable fault detection design complete record fault information
fault isolation	recognize the fault mode recognize the fault location point the fault components need be replaced
fault alarm	alarm the faults whether false alarm

2.2 Fault mode analysis for software BIT detection

After comprehensive analysis of the fault modes of embedded hardware board and its external environment, software BIT mainly detects the digital circuits and components on the hardware board. The function fault modes and corresponding BIT detection method are expressed in Table 2 and Table 3.

Table 2: Functional fault modes and corresponding BIT detection method for CPU

Target	Functional fault mode	Fault location	Fault type	BIT detection method
CPU	cannot execute instructions correctly cannot process software data	Data register Address register Program counter State register Stack pointer Float point register	one or more bit(s) flip one word flip stuck at 0 stuck at 1	Function testing Data value compare

Table 3: Functional fault modes and corresponding BIT detection method for Memory

Target	Functional fault mode	Fault location	Fault type	BIT detection method
RAM	wrong data write to RAM	Stack segment	one or more bit(s)	Function testing
	wrong data read from RAM	Ccde segment	flip	Data value compare
		Data segment	one word flip	
		Global or local variables	stuck at 0	
		User defined variables	stuck at 1	
ROM	Storage corruption	Stack segment	one or more bit(s)	Checksum method
		Ccde segment	flip	
		Data segment	one word flip	
		Global or local variables	stuck at 0	
		User defined variables	stuck at 1	
NVM	read or write wrong data	Stack segment	one or more bit(s)	Function testing
		Ccde segment	flip	Data value compare
		Data segment	one word flip	
		Global or local variable	stuck at 0	
		User defined variable	stuck at 1	

Accordingly, the key work of BIT software testing is to inject faults on its hardware platform, and observing that how the system under test (SUT) detect and report them. Software simulation method is highly appropriate for this purpose to build the operating environment with fault injection capable.

3. Simulation & emulation platform with fault injection

3.1 Architecture design

As we known, the real operating environment for software BIT is composed of multiple embedded boards, which communicate with each other through the Ethernet, USB, bus and etc.. The Built-In Test application is running on the operating system that above each board. We design the simulation platform for the fault injection system as shown in Figure 2: each embedded board is abstracted as the simulation program running on the host, which supports the guest OS with the BIT application on it. In the meanwhile, we decoupled the whole system by using emulation program to produce the excitation signal instead of the information from other connected embedded boards.

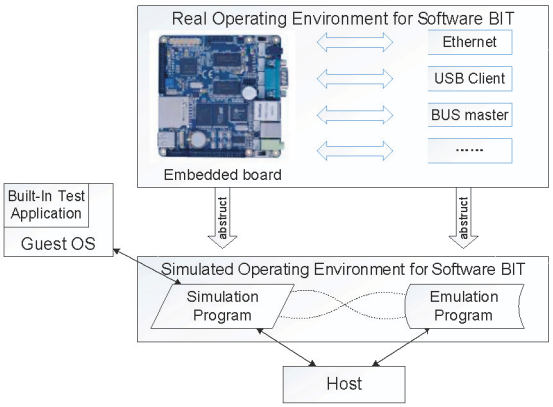

Figure 2: Architecture design for the simulation system

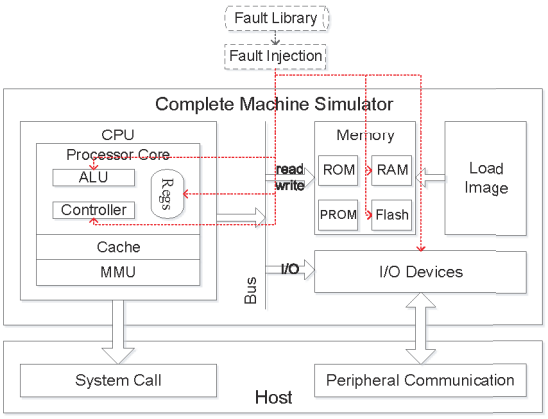

Figure 3: Complete machine simulator with fault injection capable

3.2 Embedded hardware board simulation

Simulator is a software program running on host machine that simulates the function of the embedded hardware board, so it should provide an operating environment the same as the real hardware platform.

Here we modified a complete system simulator QEMU to simulate the target embedded hardware board, which is regarded as the BIT software operating environment.

The goal of the simulator is to simulate the behavior of the target hardware board, so that it simulates the execution of instructions, exceptions, interrupts and virtual to physical memory mapping as shown in Figure 3. It uses Tiny Code Generator with basic block translation, which is a type of dynamic binary translation technique. This technique is used to accelerate the guest to host binary code generation and function simulation process.

Furthermore, we developed fault injection functions in the simulator to support BIT software testing requirements. The faults injection in QEMU is implemented by simulate their fault behavior according to the hardware related fault modes.

3.3 External environment emulation

The external environment emulation program is used to emulate external I/O request and retrieve simulator's response. Since the emulator is a discrete event driven program, we need a robust, flexible way to configure a set of triggers (e.g. BIT data query requests) and responses. This part of work is implemented by a workflow engine -- Ruote, in which the process definition and input/output data format can be described with Domain-Specific Language (DSL). Take the advantage of DSL, it's easy to define input/output data content, as well as, create or change the emulation process.

Data definition

```
<request-data>
   <bits>
      <bit-1>0xC<bit-1>
      <bit-2>0x6<bit-2>
      … …
   </bits>
</request-data>
```

```
<response-data>
   <bits>
      <bit-1>0xA<bit-1>
      <bit-2>0x3<bit-2>
      … …
   </bits>
</response-data>
```

Process definition

```
Ruote.process_definition :name => 'BIT query case' do
   sequence do
      participant 'initial task', :task => 'environment initialization'
      concurrence do
         participant 'request 1', :task => 'request for gather info'
         participant 'request 2', :task => 'request for gather info'
         participant 'request n', :task => 'request for gather info'
      end
      participant 'response', :task => 'handle case'
   end
end
```

3.4 Fault simulation and injection mechanisms

As shown in Figure 4, the normal simulation process is a loop of fetching an instruction, decoding and executing the instruction. When the simulation arrives at a fault checkpoint such as read/write register, ALU computation, memory access, and etc.. Then, searches in the customized fault sequence to find out whether there contains associated faults that need to be injected. After that, pattern recognition is used to judge whether there's a fault need to be injected at this fault checkpoint. Comparing the fault trigger condition (e.g. trigger type, fault inject time) with associated parameters from simulator's runtime environment (e.g. program counter, execution time), the fault is injected when satisfies the condition.

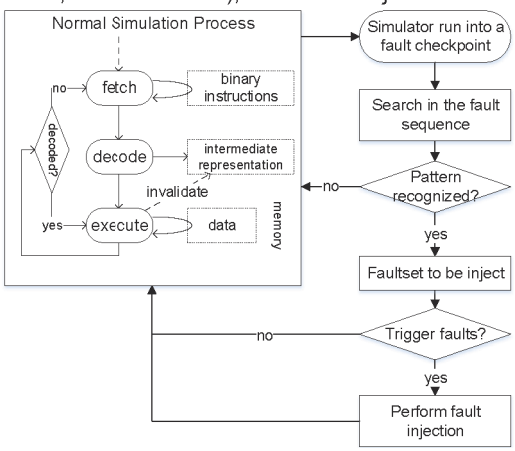

Figure 4: Fault Injection Workflow

Finally, the fault is injected by dispatching fault simulation procedure when the above steps have been passed. The steps include fault pattern recognition and fault trigger condition judgement. Otherwise, nothing would happen. Table 4 lists primary fault checkpoints setting in the simulation platform.

Table 4: Fault checkpoints setting

Fault checkpoint	Corresponding fault modes	Location for setting
Accessing registers	register bit flip / stuck-at (0 or 1) / miss load / extraneous load	Simulator -> register
CPU execution	illegal interruption / exception incorrect / extraneous instruction	Simulator -> CPU
Accessing memory	memory bit flip / stuck-at (0 or 1) memory cells coupled	Simulator -> CPU Simulator -> memory
I/O device initialize	device initial error	Simulator -> device
I/O operations	read / write error	Simulator -> device Emulator -> I/O process

4. Experiments and evaluation

4.1 Work form design

For each fault injection experiment on BIT software testing, the detection and isolation capability should be measured, as while as, less false alarm is expected. So we design the work form in our experiment as Table 5, which can roughly evaluates software BIT system.

Fault Detection Rate (FDR) analysis: whether the faults injected in the simulation environment can be detected by the software BIT under testing. Moreover, how many of them can be detected correctly.

Fault Isolation Rate (FIR) analysis: whether the faults injected in the simulation environment can be isolated by the software BIT under testing. Moreover, how many of them can be isolated correctly.

False Alarm Rate (FAR) analysis: whether the under tested software BIT alarm of those not happened faults, moreover, how many of them are false alarmed.

Table 5: Work form designed for BIT software testing

Item	Designed fault rate	FDR	FIR	FAR
1 CPU				
2 Memory				
3 I/O Devices				
4 Input data				
Total				

4.2 Fault injection capability

Taking advantage of the /proc file system in Linux, we can examine the hardware information to detect the fault. The ID Code Register (c0_cpuid) of CP15 Coprocessor is used to define a 32-bit device ID code, which returns part number by [15:4] and layout revision by [3:0] in ARM architecture. Here we inject stuck-at-1 fault to c0_cpuid's 1st bit and bit-flip fault to its 5th and 6th bit. As shown in Figure *5*, the fault can be detected by the wrong display in /proc/cpuinfo, changed from normal value part.926 rev.5 (0x41069265) to abnormal value part.920 rev.7 (0x41069207). It illustrates that these types of fault can be perceived.

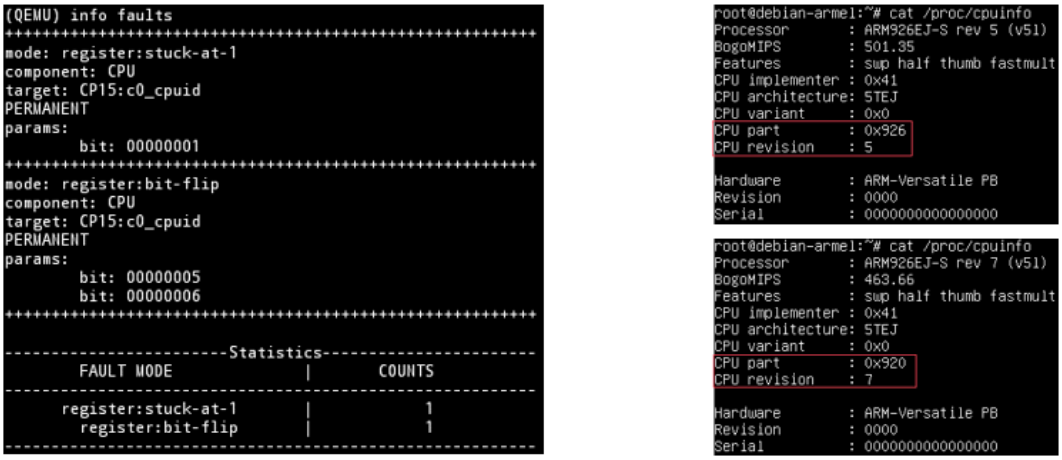

Figure 5: Fault Injection Experiment on Processor Version Register

The General Purpose Registers (GPR) is used to transfer or register data which may affect in arithmetic and logical operations. Sometimes, fault may influent the applications' running or even lead to unrecoverable failure to the OS. A permanent stuck-at-0 fault to GPR14 makes the system go into error and automatically shut down after dump the exception messages.
Finally, we have tested all the faults simulated in the fault injection environment which can be configured and injected correctly when the simulator and emulator is running. Furthermore, the faults to be injected in an experiment can be configured and queried by monitor as well. Under the help of fault monitor and guest operating system, we can detect the injected faults and observe their effect to the system.

5. Conclusion and future work

According to the requirements for BIT software testing, this paper proposed a QEMU-based fault simulation and injection system, together with a workflow engine implemented I/O data emulation process. It combines software simulation technique with fault injection to executes tests, and automatically inject faults in the simulation environment. In addition, unmodified operating systems and applications, especially the software BIT system can run on the prototype system without intrusion. Benefit from software simulation technology, the simulation-based fault injection system also has the whole control to the entire environment, which contributes to the fault injection process's monitor and results' efficient feedback.
In the future, we intend to consider the reproducibility of the system test and to design a hardware board simulation library for easier extension, as well as, a fault injector library for more easily injecting and testing. It's also expected to evolve this research in PHM system development and validation.

Acknowledgement

This research was supported by the Technological Foundation Project of China Industrial Bureau for National Defence Science and Technology (No.Z132012A004) and the fundamental research funds for the Central Universities (Project No. YWF-12-LJJC-001).

References

Bellard F., 2005, QEMU, a Fast and Portable Dynamic Translator, Proceedings of the 2005 USENIX Annual Technical Conference, Anaheim, USA, pp. 41-46.

DeBardeleben N., Blanchard S., Guan Q., Zhang Z. M., Fu S., 2012, Experimental Framework for Injecting Logic Errors in a Virtual Machine to Profile Applications for Soft Error Resilience, Euro-Par 2011: Parallel Processing Workshops, Lecture Notes in Computer Science, 2012, Vol. 7156, 282-291, DOI: 10.1007/978-3-642-29740-3_32.

Hanawa T., Koizumi H., Banzai T., Sato M., Miura S., Ishii T. Takamizawa H., 2010, Customizing Virtual Machine with Fault Injector by Integrating with SpecC Device Model for a Software Testing Environment D-Cloud, 2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing, Tokyo, Japan, pp. 47-54, DOI: 10.1109/PRDC.2010.37.

Huang F.Q., Xu P., Liu B., Li Y., 2009, The Research on Fault Equivalent Analysis Method in Testability Experiment Validation, 8th International Conference on Reliability, Maintainability and Safety, Chengdu, China, pp. 902-906, DOI: 10.1109/ICRMS.2009.5269965.

Kosar T., López P.M., Barrientos P.A., Mernik M., 2008, A Preliminary Study on Various Implementation Approaches of Domain-Specific Language, Information and Software Technology, Volume 50, Issue 5, April 2008, 390-405.

Sun J.Z., Wang J.Y., Yang X.Z., 2001, The Present Situation for Research of Fault Injection Methodology and Tools. Journal of Astronautics, Vol. 22, No. 1 (in China).

Valderas M.G., Garcia M.P., Cardenal R.F., Ongil C.L., Entrena L., 2007, Advanced Simulation and Emulation Techniques for Fault Injection, 2007 IEEE International Symposium on Industrial Electronics, Vigo, Spain, 3339-3344.

Xu P., Kang R., 2004, The Research of Fault Injection System's Framework in the Testability Experiment Validation, Control Technology, Vol. 23, No. 8, 12-14 (in Chinese).

Wang Y.C., Zhou Z.Z., 2009, Software BIT Design and Testing for Embedded Software, 8th International Conference on Reliability, Maintainability and Safety, Chengdu, China, pp. 703-707, DOI: 10.1109/ICRMS.2009.5270099.

Wang Y.C., Xu P., 2009, Build-In-Test Design and Test for Embedded Software, Computer Engineering, Vol. 35, No. 17, 34-39 (in Chinese)

Ziade H., Ayoubi R., Velazco R., 2004, A Survey on Fault Injection Techniques, The International Arab Journal of Information Technology, Vol. 1, No. 2, 171-186.