A Fault Injection Attitude based on Background Debug Mode in Embedded Systems

S.A. Asghari1, H.R. Zarandi1, H. Pedram1, M. Ansarinia2 and M. Khademi2

1Computer Engineering and Information Technology Department, Amirkabir University of Technology, Tehran, Iran
2Elecrical and Computer Engineering Department, Shahid Beheshti University, Tehran, Iran

Abstract The embedded systems usage in different applications are prevalent in recent years. These systems include a wide range of equipments from cell phones to medical instruments, which consist of hardware and software. In many examples of embedded systems, fault occurrence can lead to serious dangers in system behavior (for example in satellites). Therefore, we try to increase the fault tolerance feature in these systems. Therefore, we need some mechanisms that increase the robustness and reliability of such systems. These objects cause the on-line test to be a great concern. It is not important that these mechanisms work in which level (Hardware level, Software level or Firmware). The major concern is that how well these systems can provide debugging, test and verification features for the user regardless of their implementation levels. Background Debug Module is a real time tool for these features. In this paper we apply an innovative way to use the BDM tool for fault injection in an embedded system.

Keywords: Embedded systems, Fault injection, Fault tolerance, on-line test, Real time debugging and Background Debug Module.

Introduction

Fault injection has been introduced as a valuable way for systems fault tolerance verification. Until now, several fault injection methods have been proposed and implemented.

We can classify these techniques to the three categories [1-4]:
1) Simulation based fault injection [5-6]
2) Software implemented fault injection [7-10]
3) Hardware implemented fault injection [11]

Simulation based fault injection is used in primary design level or small applications, while the software and hardware implemented fault injection are used in systems which have easy prototype implementation. Also for the application whose simulations are very expensive, these two last techniques are much more suitable.

This paper represents the process of software implemented fault injection using an innovative usage of BDM [12] tool in Motorola processors and microcontrollers. BDM has been designed for the programming and debugging, indeed. However, it can be used as a fault injection tool.

Fault injection is a useful way for correctness evaluation, design accuracy and system fault tolerance. Also, this method is feasible for huge systems whose simulation cost is too high to implement.

The BDM based fault injection as a combination of software and simulation based fault injection techniques is economically acceptable to fault injection that placed in Motorola microprocessors. By using the BDM, we can test the different fault injection techniques and system fault tolerance.

In contrary to pure software fault injection, the trap and fault injection routines are not used in target system software.

The usage of BDM as a fault injection tool has been represented in previous works [13-15], but in all previous experiments default and wizard fault injection features have been used, while in this paper we represent the BDM as a fault injection tool by its details manipulation.

In the first section, the paper idea introduction has been explained. In second section of this paper, BDM tool has been introduced as a programming, test, verification and fault injection tool. In third section, the fault injection system architecture and system behavior has been analyzed. In forth section, the three workloads used in our fault injection experiments explained. In fifth section, the experimental results have been represented and finally in the last section, the paper conclusion will be represented.

Be BDM Introduction

The BDM is a tool which Motorola Corporation has been placed in their microprocessors and microcontrollers recently. By using of BDM, the user can program and
debug the microcontroller or microprocessor. BDM uses of few numbers of on-line logical circuits and a communication interface for connection to host processor. When the micro enters in debugging mode, the external host processor can control the microcontroller and access to registers and microcontroller or microprocessor memory.

![BDM physical connections](image1)

**Figure 1. BDM physical connections**

When the BDM is active, the normal operation is stopped and a series of micro codes are injected which the host processor controls. The operations which are run in this state with using debugger conclude of: 1) Using the Break Point, 2) Read (Write) from (to) registers and memory locations, 3) Single stepping command, 4) System reset and 5) Subroutine running.

We can see the general using of BDM in figure 1. BDM interface to host processor and microprocessor is a synchronous serial interface such as SPI (Serial Peripheral Interface) or USB (Universal Serial Bus).

When the BDM is active, microprocessor receives the host processor commands and runs them. Then, if needed, the microprocessor returns the value of registers and memory locations to host processor. We can see in figure 2, the BDM interface to processor core and its other interfaces.

![The BDM with its interfaces](image2)

**Figure 2. The BDM with its interfaces**

As it is shown in this figure, debug module has access to main bus. The BDM has several operational modes. These modes can inform users of microprocessor status such as registers status and memory status.

![BDM fault injection system](image3)

**Figure 3. BDM fault injection system**

In this fault injection method, we can use of any fault model. However, we should be concerned about that the faults are implemented in assembly level. Also with this model, we can inject faults in all three locations: 1) Data Memory 2) Instruction Memory and 3) Registers.

The fault model that has been used in our injection method is bit flip that means each fault causes to a failure in a memory location or a processor register [6-7].

Each fault is distinguished by these parameters:

1) **Fault location**: Instruction, register or memory address which faults have been injected in.

2) **Fault injection time**: This time is calculated based on the number of instruction which is run during fault injection.

As it is shown in figure 3, in first step, the fault list should be generated. This fault list can be generated by any language (for example by C or Java language).

Fault list generation consists of several steps: 1) Running the software without fault occurrence and distinguishing the assembly running procedure. 2) Random generation of fault list based on fault model. This list determines fault injection location and its time and 3) conversion of the fault list format to a format that the BDM can understand.

After fault list generation, the Fault Injector function is run. We can see the Fault Injector function in pseudo code in figure 4.
while the application is running, the Injection and Observation routine are run concurrently. The task of this routine is fault injection and system's behavior observation after the fault occurs. All these steps are performed by suitable BDM commands. The system should be able to recover itself. In the following, we analyze the system behavior.

**Workloads in fault injection**

In this paper, we use three workloads for fault injection procedure:

a) **Bubble Sort program:** This program is a sorting program for numbers. In this workload that is programmed with C language, we sort 100 numbers from 1 to 100. These 100 numbers are placed in an array randomly.

b) **π Number computation program:** In this workload that is programmed with C language, the π number is computed based on tangent function. phyCORE®-MCF5485[16-17] has floating point computation capabilities. Therefore, floating point computation is performed and processed with this microprocessor easily.

c) **4×4 Matrix multiply program:** In this workload that is programmed with C language, two 4×4 matrices are multiplied.

**Experimental results**

The Phytech evaluation board has been used for implementation. We used from This board has a MCF5485 ColdFire microprocessor. We can use different toolchain or microprogramming. The proposed Motorola Corporation toolchain is CodeWarrior that has the IDE or Integration Development Environment. This IDE has many capabilities. The other toolchain that we can use is Eclipse with Sourcery G++ Lite for ColdFire (GCC).

We used BDM interface cable that proposed in P&D for programming and debugging. Also, we can use the debugger that is placed in GCC for fault injection. We can control the GNU-GDB by the C program that is run in PC. We utilized an innovative way for fault list injection. In this step, we generate a fault list then replace the software by the address injection for these commands. For it, we replace this text to default used command. Now, we can add our fault injection requirement into this text. The pseudo code that should be written in this text has been shown in figure 5.

```
1. Load
2. Set $PC=Starting Point
3. Break in a Random Point or Location
4. Continue
5. Disable Break
6. Fault Injection (For example
   Read a register and change its content)
7. Continue
8. Wait
```

![Figure 5. Fault injection list format](image-url)
As it is shown in this text file, the determined workload is loaded, and then the instruction with 19th line is selected as a stopping point. After that, the program is run and stopped in selected point. In this point, the PC and d0 registers have been changed (fault injection occurred).

After this injection, the program resumes. Now, we should place the break in last line number of program. This break has delay role that explained. After program is reloaded, these breakpoints are disabled.

After this fault injection, the exception is occurred (as it expected). By this text, we can inject our faults into main program but this text is run once. For many times of injection, we should program the application that generates 1000 times (for example) this text content (with uniform format).

So, in this phase, we programmed the application by Java language that generate random register name, data memory location and code address in constant format. (Register, data and memory place in arrays). For result obtaining, we should compare fault injected program by golden run of program (that does not have any fault). So, four states would be occurred. 1) The fault has no effect in result. 2) The fault changes the true result. 3) The exception is occurred and 4) The timeout is occurred.

In the implementation of this idea, three workload have been used (Bubble sort, \( \pi \) number computation, \( 4 \times 4 \) Matrix multiply).

First, these workloads are programmed in microprocessor with BDM (in normal mode for programming not debugging mode), then in debugging mode, we load fault injection text that consists of standard BDM commands [18].

We can inject faults with BDM in all location of microprocessors such as Data, Registers and Code memories. We can see the fault injection results in table I.

For each workload, 3000 faults are injected that each of data, Registers and code share of faults are 1000 faults.

Fault injection rate with detected, latent errors and failure rate have been shown in table I. The time requirements of fault injection experiment have been shown in table II that compares two state concluding golden run and faulty run.

<table>
<thead>
<tr>
<th>Workload</th>
<th>Total number of fault injection</th>
<th>The Number of Failures (Exceptions)</th>
<th>The results is true but processor status is false</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble Sort</td>
<td>3000</td>
<td>985</td>
<td>1271</td>
</tr>
<tr>
<td>( \pi ) number computation</td>
<td>3000</td>
<td>994</td>
<td>1328</td>
</tr>
<tr>
<td>( 4 \times 4 ) Matrix multiply</td>
<td>3000</td>
<td>991</td>
<td>1176</td>
</tr>
<tr>
<td>Average</td>
<td>3000</td>
<td>990</td>
<td>1258.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Workloads</th>
<th>Time required for workload running without fault injection (ms)</th>
<th>Time required for workload running with fault injection (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble Sort</td>
<td>1.7</td>
<td>541</td>
</tr>
<tr>
<td>( \pi ) number computation</td>
<td>2.7</td>
<td>652</td>
</tr>
<tr>
<td>( 4 \times 4 ) Matrix multiply</td>
<td>3.2</td>
<td>764</td>
</tr>
</tbody>
</table>
6 Conclusions

In this paper, the new attitude for fault injection in embedded systems is represented. In this attitude, we apply the innovative way for fault injection via BDM. In previous works, this tool is employed as a fault injection tool but in none of them, the BDM fault injection capability details were not employed. In this attitude, we reach the high degree of fault injection coverage and easy control with BDM tool manipulation.

References


[16] P.McConkey, Open source Coldfire IDE, Set up and configuration guide, Draft 1.0, Date 31 January 2007
