

# Interaction Mutation Testing

Ahyoung Sung, Byoungju Choi and Jangsoo Lee\*

Dept. of Computer Science & Engg, Ewha Woman's Univ., Seoul, Korea

\* Korea Atomic Energy Research Institute, Daejeon, Korea

{aysung, bjchoi}@ewha.ac.kr, \*jslee@kaeri.re.kr

**Abstract** – An embedded system is a combination of hardware and software subsystems. Interaction between these two subsystems may lead to unexpected behavior when faults are present in either. We propose the Interaction Mutation Testing to reveal interaction faults in an embedded system.

## I. INTRODUCTION

In these days, we tend to have high interests in an embedded system and embedded software. In consideration of the hardware technology development, the value of the embedded system tells us that the embedded system is a software-oriented technique-intensive industry, rather than a hardware-oriented. Even if there is software that independently operates properly, it may occur faults and leads to unexpected results in the process of software embed to the target board in the interaction. Hence we need a testing technique to assure software in an embedded system.

In this paper, we propose the Interaction Mutation Testing (IMT) using the mutation-based adequacy criteria [1,2] to reveal the interaction faults between hardware and software in an embedded system.

## II. INTERACTION MUTATION TESTING

Figure 1 indicates the procedure of the test data selection of the IMT through activity and the product as rounded box and rectangular box, respectively. As shown in Figure 1, the procedure of the test data selection of IMT is as follows: We generate the program S and  $Pf_H$ . And we select the input data that differentiates the result of the two programs. We will describe each activity in Figure 1 in detail.

### 1. Analyze a System Specification

We make the analysis diagram after we analyze the target embedded system. The analysis diagram describes the target system as hardware, software with the target system's behavior implemented, and interaction part. The embedded system interprets the hardware signal input into the software input, and then transforms the software result into hardware signal and exports to the target system. In other words, the interaction part of an embedded system is

the I/O part of the embedded system.

### 2. Compose Program S

The program S simulates behavior of the target system. And the program S is coded in C language since it can easily be mounted on boards.

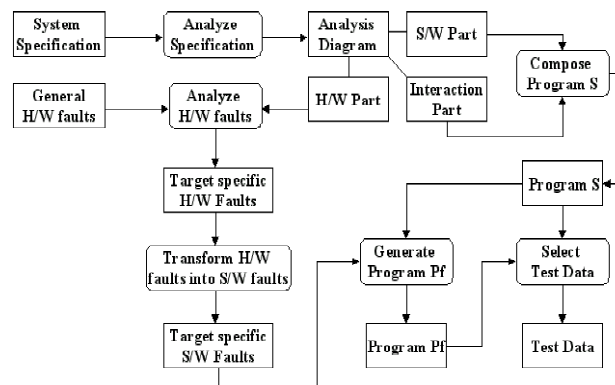


Figure 1. Interaction Mutation Testing

### 3. Analyze Hardware Faults

After hardware fault is transformed into software fault, we generate the program  $Pf_H$  by injecting the software fault into the program S, as shown in Figure 1. To do this, it is necessary to classify the hardware faults. As shown in Table 1, the hardware faults are classified as stuck-at 0[3,4,5], stuck-at 1[3,4,5], bridging fault[3,6], open fault[3], bit-flip fault[3,5], power surge[3,7,8], and spurious current fault[3,4,7,8,9]. In Table 1, hardware fault, description and impact of each hardware fault are listed.

Table 1. Hardware Faults

Fault Type	Description	Impact
stuck-at 0	The result value is fixed to 0.	The result value always comes out to be 0.
stuck-at 1	The result value is fixed to 1.	The result value always comes out to be 1.
bridging fault	When there are more than two crossing lines, the number of lines crossed varies.	The number of lines crossed is verified.

open fault	Resistance on either the line or the block occurs due to the bad connection.	The value associated to the line or the block is modified to different value.
bit-flip fault	The bit flips.	The variable based on the modified bit is verified.
power surge	Inconsistent power is supplied.	The problem not solely lies on the value of the specific location, but almost the entire system is affected.
spurious current	Exposures to heavy ion.	The problem not solely lies on the value of the specific location, but almost the entire system is affected.

#### 4. Transform Hardware Faults into Software Faults

When the system is affected by any hardware fault, there is a software fault somewhere lying on the program S. In order to change the hardware fault into software fault, we need two stages. In the first stage, we need to determine the specific location of the program S that the hardware fault has affected, and then make decision about the Fault Injection Target (FIT). The next stage deals about “How to change the hardware fault into software fault?” at the selected specific location.

##### 4.1. Determine Fault Injection Target

The FIT is defined as a variable at the very last location of the program S since it is more likely for an unexpected result to happen at the last variable when there is a hardware fault. It is possible to expect how the hardware fault affects the program S, because the variables used in the program S involve in the operation of the embedded system. For example, as shown in Table 1, the Open fault, Bridging fault, Bit-flip fault, Stuck-at 0 fault, and Stuck-at 1 fault could be mapped to the variables in the program S respectively, because impacts of these faults are very specific. And it means that the mapped variable in the program S becomes FIT. Since the Spurious current fault, and Power surge fault affect the entire system, the last output variable becomes FIT.

##### 4.2. Transform Hardware Faults into Software Faults

In order to transform the hardware faults into software faults, the hardware faults identified in the Table 1 are transformed into the software faults through the code patch. The code patch means to add the program code to modify the FIT. The code patch method to transform the hardware faults into the software faults are as following:

- ① If the variable of FIT has the digital signal value, negate the variable value to change.
- ② If the variable of FIT has the analog signal value, change the variable value with random value.

#### 5. Generate Program $Pf_H$

We generate the program  $Pf_H$  by inserting the software fault transformed from the hardware fault.

#### 6. Select Test Data

The test data T is defined as an input data, which makes the program S output different from the program  $Pf_H$  output. Supposing the input data as t, the S(t) as the result of the program S and  $Pf_H(t)$  as the result of  $Pf_H$ . We select test data T according to the following criteria.

$$T = \{t | S(t) \neq Pf_H(t)\}$$

### III. CONCLUSION

As the feature of an embedded system become complicated, the importance of the embedded system testing is recognized. In the case of an embedded system, it is not easy to correct the faults. Even if software independently works well, the interaction fault may occur when the hardware embeds the software. And the interaction fault is hard to reveal. Therefore we proposed the test data selection technique, called Interaction Mutation Testing.

Recently, we executed empirical studies by mounting embedded software into a board with 80C196 processor. From the study, we concluded that IMT is highly effective in the aspect of fault detecting capability. In the future, we will enhance IMT so that it can be applied to a general embedded system testing.

### REFERENCES

- [1] R.A.DeMillo, R.J.Lipton, F.G.Sayward, *Hints on test data selection: Help for the practicing programmer*. IEEE Computer, 11(4):34-41, April 1978.
- [2] M.E.Delamaro, J.C.Maldonado, A.P.Mathur, “Integration Testing Using Interface Mutation”, In Proceedings of International Symposium on Software Reliability Engineering, pp. 112–121, April, 1996.
- [3] Mei-Chen Hsueh, T.K. Tsai and R.K. Iyer, *Fault Injection Techniques and Tools*, Computer, Apr, pp75-82.1997.
- [4] J.V.Carreira, D. Costa and J.G. Silva, *Fault Injection spot-checks computer system dependability*, IEEE Spectrum, Aug, 1999.
- [5] J.V. Carreira, H. Madeira and J.G. Silva, *Xeption : A technique for the Experimental Evaluation of Dependability in modern computers*, IEEE Transaction on Software Engineering, VOL 23, No.2, Feb, 1998.
- [6] M.Dalpasso, M. Favalli, P. Olivo and B.Ricco, *Fault Simulation of parametric Bridging Faults in CMOS ICs*, IEEE Transaction on computer aided design of integrated circuits and systems, VOL:12. no. 9, Sept, 1993.
- [7] J.A. Clark, and D.K. Pradhan, *Fault injection: a method for validating computer-system dependability*, IEEE Computer, VOL: 28 Issue: 6, pp47-56, June 1995.
- [8] G. Miremadi and J. Torin, *Evaluating processor-behavior and three error-detection mechanisms using physical fault-injection*, IEEE Transaction on Reliability, VOL:44 Issue 3, pp441-454, Sept, 1995.
- [9] J.Karsson, P. Liden, P. Dahlgren, R. Johansson and U. Gunneflo, *Using Heavy-ion Radiation to validate fault-handling mechanisms*, IEEE micro, pp8-23, 1994.