Analysis on the Effect of Injected Faults to the Functioning of a Digital System

Man Cheol Kim [a*]
[a] School of Energy Systems Engineering, Chung-Ang University, 84 Heukseok-ro, Dongjak-gu, Seoul, Korea, 06974
[*] *Corresponding author: charleskim@cau.ac.kr*

1. Introduction

As more and more digital instrumentation and control (I&C) systems are introduced to nuclear power plants (NPPs), the need for effective risk assessment of digital I&C systems is also growing. Kang et al. [1] provided an overview of the risk quantification issues related to the digitalized safety systems in NPPs, and the fault coverage was one of them. Kim and Lee [2] identified important factors affecting the fault coverage of digital I&C systems, and found that system-related factors such as hardware, software, input/system state, and fault detection algorithm, and fault-related factors such as fault type, fault location, fault occurrence time, and fault duration affects fault coverage.

One of the most widely used methods for estimating fault coverage is the fault injection experiment. After intentionally injecting a fault into a system, it is observed whether the system functions properly or not. Such experiments are repeatedly performed to produce statistically meaningful measures on fault coverage. Because of the large number of experiments, often the attention is given to the final result of each experiment, rather than how the final result of each experiment was achieved.

This paper is intended to provide a somewhat closer look at some degree of details on how a specific fault injection experiment arrived at the specific result that will be used as a single data point out of thousands or sometimes millions that will be used to quantify the fault coverage of a digital I&C system.

2. Methods and Results

2.1 Fault injection experiment

The example digital system equips with a 32-bit central processing unit (CPU), which is the type of CPUs that is widely used in modern digital I&C systems, and executing quick sort software, which is widely used in many previous fault injection experiment studies such as Nicolescu et al.[3]. The initial input set to the digital system is set as {9,8,7,6,5,4,3,2,1}. Without injecting a fault into the digital system, the digital system provide the calculation result of {1,2,3,4,5,6,7,8,9} after executing 863 steps.

Fig. 1 shows the result of example fault injection experiments on the example digital system. Stuck-at-0 faults were injected into 16 different registers in the CPU, and therefore faults were injected into 512 different locations. About 79% of the injected faults did not affect the final calculation result and the number of execution steps. About 13% of the injected faults resulted in the software hang, i.e. the software did not produce its calculation result in a predefined time interval. About 8% of the injected faults resulted in the wrong output, i.e. the software successfully terminated its execution for the input but produced incorrect calculation results. From the safety viewpoint, the wrong output is considered to be most threatening.

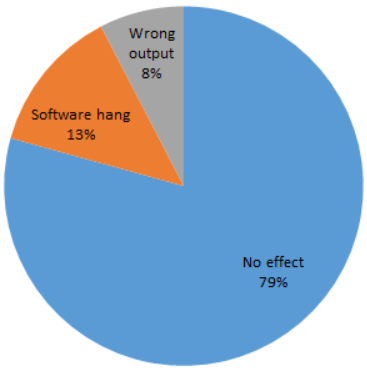

Fig. 1. Results of example fault injection experiments on the resistors of a digital system

When a stuck-at-0 fault is injected into the 0^{th} bit of the R0 register of the CPU, the calculated finishes after the execution of 1075 steps and the final calculation result was {0,2,2,4,4,6,6,8,0x3000011C}. Therefore, the injected fault resulted in one case of about 8% of wrong output cases. In an ordinary fault injection experiments for the purpose of estimating the fault coverage of a digital system, further details would seldom reviewed.

2.2 Effect of an injected fault

For the purpose of understanding how the stuck-at-0 fault injected into the 0^{th} bit of the R0 register of the CPU affect the execution of the software, the process of how the calculation is performed is analyzed in more detail. It is found that up to 108 steps, what are written in the R0 register are all even numbers such as 0, 0x300000B8, and 0x30000230. The first discrepancy between the fault free execution and the execution with the stuck-at-0 fault at the 0^{th} bit of R0 register occurs when the software tries to write the number 9 to the R0 register, which results in 8 being written in the R0 register due to the injected fault. The discrepancy

remains until the software tries to write an even number (0x30000230) to the R0 register.

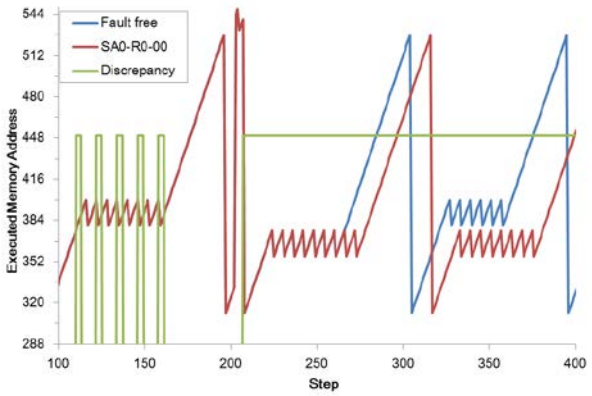

Fig. 2. Change of executed memory addresses as a function of execution steps for the fault free execution and the execution with stuck-at-0 fault at the 0th bit of the R0 register (SA-R0-00).

Fig. 2 shows the change of executed memory addresses as a function of execution steps for the fault free execution and the execution with stuck-at-0 fault at the 0th bit of the R0 register. Such discrepancies described above occur five times between 110th step and 160th step while the executed memory addresses of the fault free execution and the execution with the injected fault remain same. The five time discrepancies correspond to the number of odd numbers in the input set to the digital system.

Another discrepancy occurs after 200th step when the software tries to write an odd number but eventually an even number is written in the R0 register due to the injected fault. It is worth to note that the executed memory addresses between the two execution cases remain same until around 260th step. This means that even though some data in the registers or memory may differ from the fault-free execution, the two execution cases execute the same binary codes. After around the 260th step, the two execution cases start to execute different memory addresses, i.e. different binary codes, and therefore the final calculation results of the two execution cases differ from each other.

Fig. 3 shows the change of executed memory addresses as a function of execution steps for the fault free execution and the execution with stuck-at-0 fault at the 1st bit of the R0 register. Discrepancy occurs when the software tries to write the number 2 to R0 register but eventually 0 is written in the R0 register due to the injected fault. From that point on, it can be seen in Fig. 3 that the two execution cases execute different memory addresses, i.e. different binary codes. The injection of stuck-at-0 fault to the 1st bit of the R0 register resulted in a software hang, i.e. the software could not provide the final calculation result in a predefined time period.

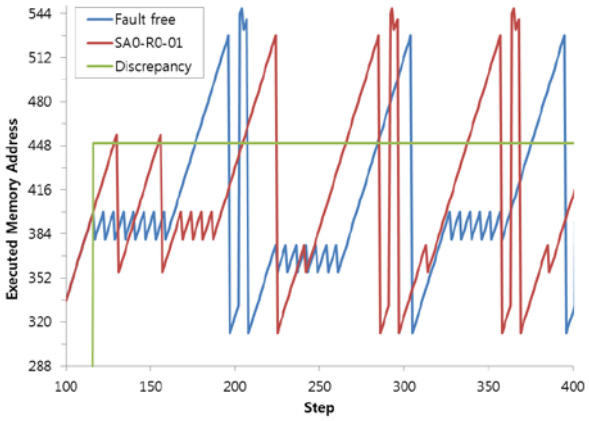

Fig. 3. Change of executed memory addresses as a function of execution steps for the fault free execution and the execution with stuck-at-0 fault at the 1st bit of the R0 register (SA-R0-01).

3. Conclusions

The importance of fault coverage in risk assessment of digital I&C systems has led our attention to fault injection experiments. While we have mainly focused on the quantitative measure of fault coverage by statistically treating the results of fault injection experiments, it seems that little attention is given on understanding how each fault occurring in a digital system affect the functioning of the digital system.

By reviewing somewhat detailed information on the process of executing binary codes under the existence of injected faults, it is found that it is important to clearly identify when, where, and how the injected fault affects the execution of the binary codes and produces discrepancies compared to the case of fault-free execution. Even though this analysis is conducted on one of the simplest form of fault injection experiments, it is expected that the findings can be applicable to fault injection experiments on more complicated real-world digital I&C systems for nuclear power plants.

REFERENCES

[1] H. G. Kang, M. C. Kim, S. J. Lee, H. J, Lee, H. S. Eom, J. G. Choi, S-C, Jang, An overview of risk quantification issues for digitalized nuclear power plants using a static fault tree, Nuclear Engineering Technology, Vol.41, p. 849, 2009.
[2] M. C. Kim, S. J. Lee, Important factors affecting fault detection coverage in probabilistic safety assessment of digital instrumentation and control systems, Journal of Nuclear Science and Technology, Vol.51, p.809, 2014.
[3] B. Nicolescu, Y. Savaria, R. Velazco, Software detection mechanisms providing full coverage against single bit-flip fault, IEEE Transactions on Nuclear Science, vol.51, p.3510, 2004.