A Novel Method for Generating Test Scenarios Based on IDF Model

In Hwa Choi[1], Jong Ho Paik[1]*, Jun Hwang[1] and Jaehyoun Kim[2]

[1]*Department of Multimedia, Seoul Women's University*
[2]*Department of Computer Education, Sungkyunkwaqn University*
{urangi, paikjh, hjun}@swu.ac.kr, jaekim@skku.edu

Abstract

This paper defines a new type of faults that can happen in the integration of embedded systems. When reusing a hardware component combining with a new software component, there can be discrepancy among the interfaces of the legacy hardware and the new software. IDF (interface discrepancy fault) means the faults that come from discrepancy between interfaces of reused hardware (legacy codes) and a new software to combine with. This paper defines the IDF model for embedded software testing. In this paper, the DFS (Depth First Search) algorithm is enlarged to insert IDF node and is used to automatically produce test scenarios considering the interface discrepancy. The resulted test scenarios raise the test coverage and help the testers to check the interface discrepancy faults between the HW and SW for embedded modules, which have usually been ignored and consumed much of the clueless efforts

Keywords: *Software Interface fault model, IDF, Test scenario, Interface fault test*

1. Introduction

Generally speaking, embedded systems had been designed and implemented focused on HW, except some critical functions to perform specific parts. However, recently the embedded SW has been used in the type of convergence in a wide area, such as in Smart-phones, Smart TV, medical instruments, telecommunication, aviation, automobiles. Thus the focus has been moved towards SW from HW to be the qualified products. These changes make the SW more crucial and complex, and so is the testing of SW to ensure the quality of the embedded systems.

To improve the quality of SW, various professional methods of testing has been developed, such as 'system static analysis, formal model proof, automated code generation, etc. [1, 2]. Also the structural test, random test, interface test has been used for embedded SW quality inspection [14, 15, 16]. However, the specific traits of embedded SW – high coupling for optimization, frequent changes in SW owing to the dependency on the target HW, big-bang type of integration and difficulties in V&V(Verification and Validation), participation of various companies and higher risk of interface discrepancy among the heterogeneous hierarchy of modules[4, 5, 6, 7] – makes the SW testing even more difficult.

According to recent trend of reusing HW modules, there are more and more interface discrepancy faults between SW component and HW, which makes the testing process quite expensive and time consumable.

Actually, the result of survey regarding on testing process of 3 DAB developing companies, there have been interface discrepancy faults and it took them 30% of testing schedule. The key reason was the fact that there has been no concept of this kind of

fault for the existing testing methodologies. This made it very difficult for the testers to expect and detect the fault.

Generally speaking, embedded SW testing methods assume that if there is a branch in HW interfaces, the SW would call and interface with the proper, exact one. They don't expect or consider the situation that the target function to interface with can be wrong – owing to some unexpected fault. In this case, the performing of the wrong function is meaningless, nevertheless whether the result is successful or not. Rather, if the result is the same with expected result by any chance, it is far more dangerous because the tester may consider it successful.

This paper defines these kinds of interface discrepancy fault (IDF) model, and proposes the algorithm that generates the test scenarios to test IDF. The number of the test scenarios generated by the algorithm adopting IDF model was a lot higher, compared to the existing methods that doesn't consider IDF model. It shows significant meaning of IDF model in SW testing in terms of raising test coverage and fault detecting probability.

This paper is composed as follows:

Chapter 2 introduced new fault model named 'IDF'. Chapter 3 explains an algorithm which automatically generates test scenarios based on IDF model. Lastly, chapter 4 states the conclusion of the study and the direction of research hereafter.

2. IDF (Interface Discrepancy Fault) Model

The IDF is the faults that can occur when the interfaces of heterogeneous layers to integrate are not matched. This phenomenon appears frequently while reusing HW modules to develop embedded system. For example, consider HW module has related interfaces A and A' in the embedded system but in the SW, only A is defined. Now, the SW module tries to call HW interface A. However, let's think about the situation that an unexpected fault occurs and A' is called instead of A. This is clearly the erroneous situation. However, HW has got A' defined, and not knowing that this call is mishap, it performs it out unfortunately. The result itself is fail regardless of the value, but the existing testing model hasn't had the concept of interface discrepancy fault, thus it has been impossible to identify the cause of the errors.

Even more dangerous case is that the result value matches with the expected successful value and deceives the tester and makes the tester ignorant of the fact that there WAS an error. Even when the error is perceived, the result is usually the one that are performed several steps afterwards that it is really difficult to detect where and how the fault occurred. Ideally, there must be tests that check whether the correct interfaces are called and performed between SW and HW. To solve this problem, this paper defines a new type of fault model that may occur in the integration of legacy HW and new SW.

Definition

Consider an embedded system P, and the test case t for P. Suppose that P is composed of SW unit F and G, and HW unit H.

- F makes hardware signals to H,
- H makes hardware signals to G

Consider $S_I(H)$ to be the n-tuple of values passed to H, and $S_I(G)$ the n-tuple of values passed to G, and $S_O(H)$ the n-tuple of values delivered from H.

The $S_I(H)$, $S_I(G)$, and $S_O(H)$ are defined as:

- $S_I(H)$: The n-tuple of input values used in a call to function H is determined by
  - The input parameters used in the function call and
  - The global variables used in G
- $S_I(G)$: The n-tuple of input values used in the call to function G is determined by
  - the input parameters used in the function call
- $S_O(H)$: The n-tuple of output values by function H is determined by
  - The n-tuple of result values that are resulted by the input values to H and stored in the hardware memory or register.

When performing P in the test case t, the interface faults that can be resulted from the signals from F to H are as follows:

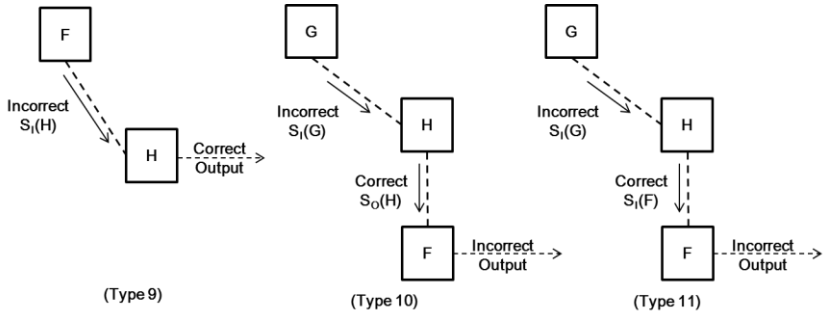

Figure 1. Interface Discrepancy Fault Model

***Type 9*:** At the introduction of H, $S_I(H)$ may have some unexpected values, which makes an unintended input values. Nevertheless, H accepts it as a normal input and tries to perform the next process.

Type 10: As a result of type 9, H performs the wrong function which is not intended by H, and the wrong values are stored in the memory or register by $S_O(H)$. Also the unit G use this wrong output and results in wrong results again.

Type 11: As a result of type 9, H performs the wrong function that is not intended by F, and then via $S_I(G)$, performing command is passed to unit G. Unit G also results in a wrong outcome.

The Figure 1 depicts the 9, 10, 11 types of IDF model defined ahead.

3. Test Scenario Generation Algorithm based on IDF Model

The automatic test scenario generation method proposed in this paper is composed in three steps. The first step is ‘activity diagram generation’ step where the target domain is analyzed and expressed as activity diagrams using UML. The second step is ‘graph drawing’ step where the activity diagram of the 1st step is converted into a graph by some determined rules. And lastly, the third step is ‘test scenario generation’ step where the graph is used to generate test scenarios by adopting an algorithm.

3.1. Activity Diagram Generation

Here, the target domain for test scenario is Digital Audio Broadcast (DAB) 'announcement' function. Figure 2 shows this announcement function as an activity diagram.

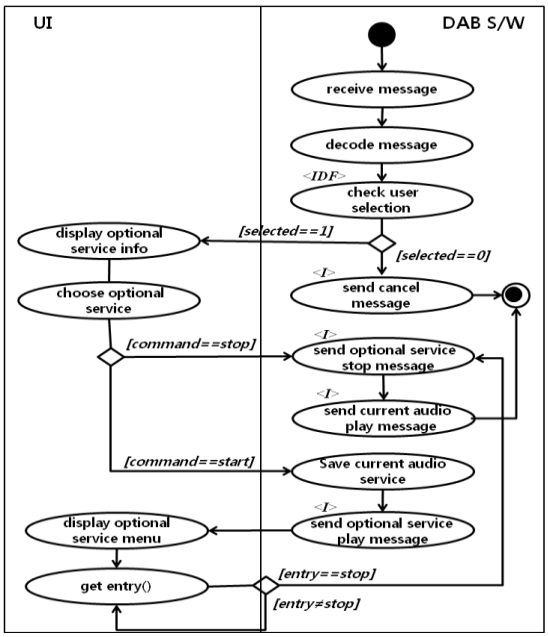

Figure 2. Activity Diagram of DAB Announcement

The central line distinguish the DAB SW module in the right area, and User Interface (UI). DAB SW module receives the broadcasting signal and decodes it, and on request from the user, it passes signal to specific hardware interface. UI offers the information received from DAB S/W module to the user, and delivers the request from the user to the module. The process to generate the activity diagram is omitted, for it is not the proposed content in this paper.

3.2. Graph Drawing by using Activity Diagram

There has been various studies based on UML regarding with methods to draw graphs for automatic test-cases generation [10, 11, 12, 13]. However, existing methods cannot generate test scenarios that consider interface discrepancy faults. This paper extends the existing graph drawing method by adding a new attribute to the nodes, so that the graph become capable of adopting the IDF model. The graph used here is directed graph and defined as:

$$G = \{N, E\} \quad (1)$$
$$N = \{n_1, n_2, \cdots, n_n\} \quad (2)$$
$$E = \{e_1, e_2, \cdots, e_n\} \quad (3)$$
$$n_i = \{id, a\} \quad (4)$$
$$e_i = \{id, \text{tail node}, \text{head node}, c\} \quad (5)$$
$$a = \{i \text{ or } IDF \text{ or } NIDF\} \quad (6)$$

- The graph G is compose of N (a set of more than one node *n*) and E (a set of more than one edge *e* which links two *n*) (1), (2) and (3).

- *n* is compose of "unique identifier id and attribute *a*" (4).
- *e* is composed of "unique identifier *id* and a pair of tail node and head node, and necessary condition *c* that forms a pair of *n*" (5).
- *a* is composed of *i* (interface node) and *IDF*(IDF node) and *NIDF*(general node) (6).

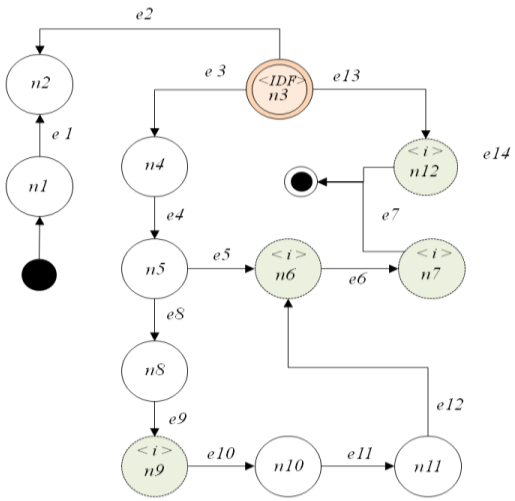

Figure 3. Graph converted from Diagram

Figure 3 depicts the result of graph conversion from the activity diagram of Figure 2 by using definition formula.
The graph is generated by the following rules.

- One activity is expressed as a node.
- To grant each node a unique id and an attribute.
- Attribute rule: If an activity is linked to hardware interface, grant *i* attribute, if it is a linking branch with hardware interface, grant *IDF* attribute, and grant *NIDF* attribute for the other cases.
- When expressing linkage information *e* from activity A to activity B, A is tail node and B is head node.
- In an activity, there can be more than two divergences to different activities by some condition. Here, the condition is expressed as the condition for each edge of those pairs of activities.

3.3. Test Scenario Generation Algorithm using the Graph

This chapter explains test scenario generation algorithm using the graph in two parts. The first part is about the algorithm to generate general test scenarios, which is an extended algorithm of DFS. The second part explains the algorithm to generate test scenarios for IDF that this paper defines.

3.3.1. General Test Scenario Generation: The first algorithm is the one adapted from DFS (Depth First Search). The adapted graph algorithm store the paths using a stack structure, and then draw the paths needed for the testing.

By adapting the Main Algorithm in Figure 4 to the graph in Figure 3, three graph paths can be obtained as follow.

- path 1: *n1 → n2 → n3 → n4 → n5 → n6 → n7*
- path 2: *n1 → n2 → n3 → n4 → n5 → n8 → n9 → n10 → n11 → n6 → n7*
- path 3: *n1 → n2 → n3 → n12*

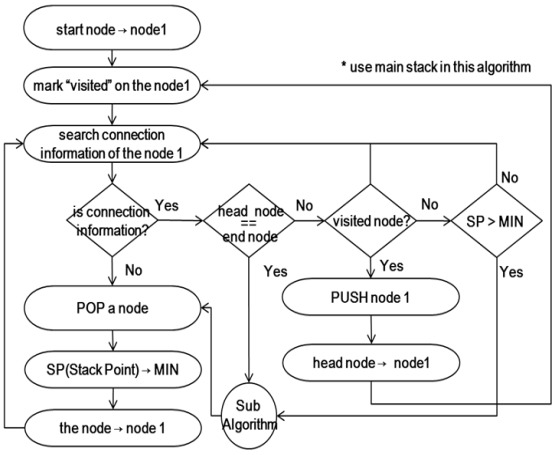

Figure. 4. Main Algorithm

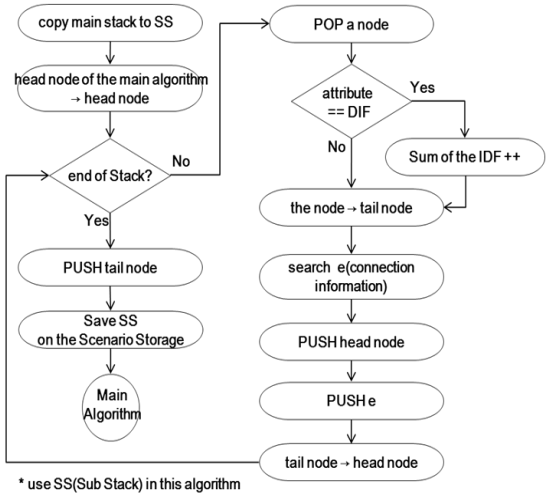

Figure. 5. Sub Algorithm

Figure 5 shows the extended Sub Algorithm which adds edge information on the paths above and get the number of IDF nodes in each path, so that we can generate test scenarios. By these processes, test scenarios are generated as follows:

- TS1: By these processes, test scenarios are generated as follows:
- TS2: *n1 e1 → n2 → e2 → n3 → e3 → n4 → e4 → n5 → e7 → n8 → e8 → n9 → e9 → n10 → e10 → n11*
- TS3: *n1 → e → n2 → e2 → n3 → e12 → n12*

3.3.2. IDF Test Scenario Generation: IDF test scenario can be generated by predicting fault occurrence paths in the graph of Figure 3. There are 6 paths and they are labeled from ① to ⑥. Path ⑥ is branched from TS3. This is the case where some error occurs in the way to HW interface n12 and goes to wrong interface n9.

In this situation, HW unit does not recognize the fault and thus proceed to perform processes related with n9. If SW unit already performed processed regarding n4, n5, n8 which located before n9, there would be no collision with the result of HW processing. However, in fact the SW unit has processed by n3 and there shall be unexpected collision between SW unit and HW unit. In the same way like path ⑥, path ⑤ performs properly by *n1 → e1 → n2 → e2 → n3* but there shall be error in e12 and interface n6 is performed instead of n12.

The rest of the paths are also the cases that from IDF node to the first interface node, there occurs some errors on the edge that links to the target interface and thus makes a new edge.

Figure 7 depicts the algorithm that generates IDF test scenarios. The general TSs(4.3.1) are only calling the target interfaces. However, the extended IDF model TSs are inducing node information in legacy TS. When it encounters with IDF node, it adds paths to the other interfaces (excepting the target interface) and generate test scenarios. The followings are the paths generated by the extended algorithm

- IDF TS1: $n1 \rightarrow e1 \rightarrow n2 \rightarrow e2 \rightarrow n3 \rightarrow e3 \rightarrow n4 \rightarrow e4 \rightarrow n5 \rightarrow e15 \rightarrow n9$
- IDF TS2: $n1 \rightarrow e1 \rightarrow n2 \rightarrow e2 \rightarrow n3 \rightarrow e3 \rightarrow n4 \rightarrow e4 \rightarrow n5 \rightarrow e16 \rightarrow n12$
- IDF TS3: $n1 \rightarrow e1 \rightarrow n2 \rightarrow e2 \rightarrow n3 \rightarrow e3 \rightarrow n4 \rightarrow e4 \rightarrow n5 \rightarrow e7 \rightarrow n8 \rightarrow e17 \rightarrow n6$
- IDF TS4: $n1 \rightarrow e1 \rightarrow n2 \rightarrow e2 \rightarrow n3 \rightarrow e3 \rightarrow n4 \rightarrow e4 \rightarrow n5 \rightarrow e7 \rightarrow n8 \rightarrow e18 \rightarrow n12$
- IDF TS5: $n1 \rightarrow e1 \rightarrow n2 \rightarrow e2 \rightarrow n3 \rightarrow e19 \rightarrow n6$
- IDF TS6: $n1 \rightarrow e1 \rightarrow n2 \rightarrow e2 \rightarrow n3 \rightarrow e20 \rightarrow n9$

These are definitely the error scenarios and if the SW is written properly, it shall detect the error and perform a proper exception dealing routines. In this way, by adding TS based on IDF model, the tester can check the interfacing errors between the legacy HW and the SW. These IDF TSs can check the erroneous part as vague as a black box. By IDF model, SW become more robust against interface discrepancy faults which has made the embedded SW testing very difficult.

4. Conclusions

This paper proposes two things regarding with embedded system test. The first is to define a new interface fault model which has not been dealt before. This model can be categorized in three types which can occur when a legacy HW is reused and integrated with a new SW. Secondly, this paper introduces test generating algorithm that considers the new fault model. The original algorithm based on DFS graph is extended and via main and sub algorithms, test scenarios can be generated to check not only the normal cases but also the IDF cases

Compared with previous TSs, IDF model results in more test scenarios which deals with crucial error cases in embedded system. The proposed model has significant meaning in that it raises the test coverage and the probability of fault occurrence.

This paper covers the proposal of the fault model and method to generate related test scenarios. Hereafter, it is planned to implement the model and scenario and apply actually to embedded systems. Through this, the result will be compared with the existing test methods and analyzed to prove the significance of the proposed model.

References

[1] B. E. Notenboom, “Testing Embedded Software”, Addison-Wesley, **(2003)**.

[2] J. Regehr, “Random testing of interrupt-driven software”, In Proc. Of the Int’l Conf. on Embedded Software **(2005)** September, pp.290-298.

[3] J. Yo. Seo, “Imbedded Software Test reflecting System state”, **(2009)**.

[4] A. A. Jerraya and W. Wolf, “hardware/software interface codesign for embedded systems”, IEEE Computer, **(2005),** pp. 75-84.

[5] T. Wei-Tek, T. Lian, Z. Feng and P. Ray, “Rapid embedded system testing using verification patterns”, IEEE Software, **(2005)**, pp. 68-75.

[6] S. Yoo and A. A. Jerraya, “Introduction to hardware abstraction layers for Soc”, In Proc. Of the Int’l Conf. On Design, Automation and Test in Europe and Exhibition(DATE), **(2003)**, pp. 336-347.

[7] E. A. Lee, “What’s ahead for embedded software?”, IEEE Computer, **(2000)**, pp. 18-26.

[8] M. E. Delamaro, J. C. Maldonado and A. P. Mathur, "Interface mutation: an approach for integration testing", IEEE Transactions on Software Engineering, vol. 27, no. 3, **(2001)** March.

[9] Haley and S. Zweden, "Development and application of a white box approach to integration testing", The Journal of Systems and Software, vol. 4, **(1984)**, pp. 309-315.

[10] H. Kim, "GeGenerating Test cases from UML Activity Diagrams", Information and Communications University.

[11] C. Minsong, Q. Xiaokang and L. Xuandong, "Automatic test case generation for UML activity diagram", Proceedings of the 2006 international workshop on Automation of software test AST' 06, **(2006)**.

[12] D. Kundu and D. Samanta, "A Novel Approach to Generate Test Cases from UML Activity Diagrams", Journal of Object Technology, vol. 8, no. 3, **(2009)** May.

[13] H. Li and C. P. Lam, "Using Anti-Ant-like Agents to Generate Test Threads from the UML Diagrams", TestCom 2005, LNCS 3502, **(2005)**, pp.69-80.

[14] S. Kandl, R. Kirner and P. Puschner, "Development of a framework for automated systematic testing of safety-critical embedded systems".

[15] J. Regehr, "Random testing of interrupt-driven software", In Proc. of the Int'l Conf. on Embedded Software(EMSOFT), **(2005)**, September, pp. 290-298.

[16] J. T. Alander, T. Mantere and G. Moghadampour, "Testing software response times using a genetic algorithm", In Proc. of the 3rd Nordic Workshop on Genetic Algorithms and their Applications (3NWGA) **(1997)**, pp. 293-298.

Authors

In Hwa Choi received the B.S. and M.S. degrees in Computer Science from Seoul Women's University, Korea, in 2005 and 2007 respectively. She is currently pursuing her doctoral degree at Seoul Women's University. Her present research interests are in the areas of digital broadcasting and embedded software testing.

Jong Ho Paik received the B.S., M.S., and Ph.D. degrees in the school of Electrical and Electronic Engineering from Chung-Ang University, Seoul, Korea, in 1994, 1997, and 2007, respectively. He was a Director with Advanced Mobile Research Center at Korea Electronics Technology Institute (KETI) by 2011. He is currently an assistant professor in the department of multimedia, Seoul Women's University, Seoul, since 2011. His research interests are in the areas of web-based communication, software testing, wireless/wired communications system design, video communications system design and system architecture for realizing advanced digital communications system and for advanced mobile broadcasting networks as well.

Jun Hwang received the B.S and M.S. and Ph.D. degrees in Computer Science from Chung-Ang University, Korea, in 1985, 1987, and 1991 respectively. Since 1992, he has been a professor at the College of Information & Media of Seoul Women's University. His current interests are IPTV, convergence computing, and digital broadcasting.

Jaehyoun Kim received his B.S. degree in mathematics from Sungkyunkwan University, Seoul, Korea, M.S. degree in computer science from Western Illinois University and Ph.D. degrees in computer science from Illinois Institute of Technology in U.S.A. He was a Chief Technology Officer at Kookmin Bank in Korea before he joined the Department of Computer Education at Sungkyunkwan University in March 2002. Currently he is a chairman of the department of Computer Education at Sungkyunkwan University. His research interests include object-oriented modeling and design, software architecture, software process issues, and computer education.