Reusable Modeling of Diagnosis Functions for Embedded Systems

Shingo Nakano, Tatsuya Shibuta, Masatoshi Arai, Noriko Matsumoto, Norihiko Yoshida
Graduate School of Science and Engineering
Saitama University
Saitama, Japan
Emails: {nakano, shibuta, arai, noriko, yoshida}@ss.ics.saitama-u.ac.jp

***Abstract*—This paper presents a technique to embed diagnosis functions in model-based design of embedded systems, allowing designers to do this at early design stages, before separation of hardware and software (HW/SW) implementations and derivation of several variations. First, we develop some simple monitoring functions suitable for both HW/SW based on JTAG (Joint Test Action Group). Then, in order to identify faulty components in a complex embedded systems where a fault in a component can affect others, we employ a method proposed by Kutsuna et al., which is based on "model-based diagnosis" studied in the field of Artificial Intelligence. This paper uses MATLAB/Simulink as a modeling framework, and employs aspect-oriented approach for diagnosis description to promote reuse of diagnosis function models. This method enables to locate a fault source even if the fault propagates multiple modules.**

***Keywords*—Embedded System Diagnoses; Aspect-Oriented Systems**

I. INTRODUCTION

Recently, embedded systems are getting large, complex, distributed, and are composed of many components both in HW/SW. When a system fails, specifying the source is difficult because it often involves a number of components. Therefore, by implementing a function of detecting failure and specifying the source in the system, one can reduce cost and increase running rate of the system by shortening the average time to repair.

There exist various techniques based on studies of diagnosis of embedded systems, such as abstract model-based diagnosis [1], modelling and verification [2], and health assessment [3]. Additionally, we can effectively develop embedded systems by adding diagnosis functions at design modeling stages in accordance with model-based design, since we can verify the model of the system, which includes these functions, and generate source code from these models [4].

However, these techniques [1][2][3][4] have been studied individually, so it is still not clear how they will be applied in actual development of embedded systems.

This paper proposes a technique to implement diagnosis functions that detect failed components, and embed them at modeling stage. We aim to include diagnosis functionality into embedded systems in a way that allows implementing HW easily and with less resources. In this research, we use MATLAB/Simulink [5] for modeling.

First, we describe a function getting Input and Output (I/O) of component in the modeling stage of implementing the HW/SW (referring to a method proposed by Irizuki et al. [6] for monitoring the I/O) in order to get I/O needed for the diagnosis. Then, we model and implement a diagnosis method proposed by Kutsuna et al. [1] to locate the fault source.

Embedding of the diagnosis functions occurs at the modeling stage using aspect-oriented programming. By using aspects, we can easily add modularized functions, and it is possible to handle variations as well as make design more effective.

The structure of the paper is as follows. In Section II, we propose how to obtain the I/O of components and create a model using MATLAB/Simulink. In Section III, we summarize the diagnosis methods proposed in [1] and explain our method. In Section IV, we describe how to apply diagnosis functions to the target using aspects, since it enables function parts to be reused. In Section V, we simulate proposed diagnosis functions using a simple model to verify feasibility of this research. Finally, in Section VI, we describe the conclusion and outline future work of this study.

II. OBTAINING INPUT AND OUTPUT OF A COMPONENT

A. Idea

Our idea is that diagnosis requires Input and Output of a component in an embedded system. In this research, we use JTAG (Joint Test Action Group) [7], which is the standard test access port and boundary scan architecture of IC chips to obtain Input and Output.

B. JTAG

Irizuki et al. propose a method which monitors Inputs and Outputs of an embedded system using JTAG to prevent malfunction [6]. In JTAG, it is possible to I/O cells corresponding to the respective I/O pins of the IC chip from outside. The controller for JTAG test is standardized as TAP (Test Access Port) Controller and has at least four serial interfaces: TCK (Test Clock), TMS (Test Mode Select), TDI (Test Data In) and TDO (Test Data Out).

Fig. 1 shows the structure of the IC chip according to JTAG. Cells are placed between I/O pins of the IC and the internal logic, and store the values of the I/O. Since cells are implemented as cascading shift registers, it is possible to input from TDI and output to TDO.

C. Abstract JTAG

JTAG is implemented in HW and does not involve complex computations, therefore it is also suitable for embedded systems with limited resources and functionality. However, because of that, it is only possible to monitor Inputs and

Outputs in HW. In this research, we propose getting Inputs and Outputs needed for diagnosis at the modeling stage before separation of HW/SW, since diagnosis is HW/SW-independent. For this purpose, we make JTAG more abstract in the following way.

- Inputs and Outputs go directly into an IC without using cells.
- There is no need for instructions and instruction register.
- The value of TMS is used for control, instead of TAP controller.
- We obtain the whole value of the Input and Output rather than obtaining it bit by bit.
- The value of TDI is shifted only after the shift in values of the Input and Output.

D. Getting Inputs and Outputs with abstract JTAG

Abstract JTAG obtains Inputs and Outputs switching the following two states of the value of TMS.

TMS=0: Inputs and Outputs of diagnosis target are stored in cells.

TMS=1: Stored values are shifted.

Then, we get Inputs and Outputs using above states as follows.

Step 1	Input and Output values are stored in cells (TMS=0).
Step 2	Saved values are shifted to TDO (TMS=1).
Step 3	Repeat Step 2 times the number of Inputs and Outputs.

E. Modeling in MATLAB/Simulink

In this research, we make JTAG more abstract and model it in MATLAB/Simulink, which is widely used in embedded system design.

If HW implementation is done according to this model, it becomes identical or similar to JTAG. On the other hand, if SW implementation is done according to this model, there is no need for dynamic memory assignment, and control is performed with fixed amount of memory (determined by the number of Inputs and Outputs) using assignment operations and control statements.

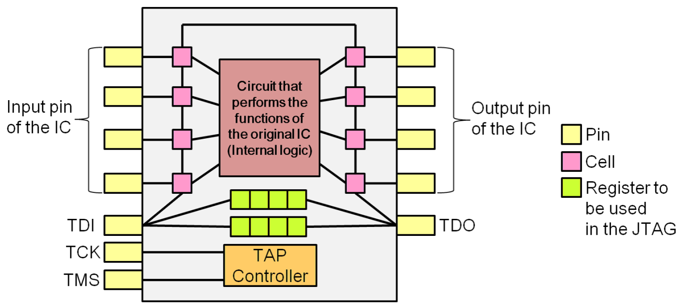

Input pin of the IC
Circuit that performs the functions of the original IC (Internal logic)
Output pin of the IC
TDI
TCK
TMS
TAP Controller
TDO
Pin
Cell
Register to be used in the JTAG

Fig. 1. IC chip construction based on JTAG.

III. Locating faults by model-based diagnosis

A. Idea

If one component fails and outputs abnormal value in embedded systems, from the outside it may look like multiple components fail, since components that receive the output also output abnormal values. This problem is known as fault propagation problem, which makes it difficult to identify faulty component in large-scale systems.

In this research, we use abstract model-based diagnosis [1] to identify the faulty component.

B. Abstract model-based diagnosis

Model-based diagnosis is a framework for system diagnosis that defines the behavior of each component and determines whether components are normal or not using logical relations derived from the structure of the system, and observations of data flows through the system [8][9]. For this purpose, it uses statements SD (System Description), OBS (Observations) and DIAG (Diagnosis).

SD	Statement indicating the logical relations derived from the structure of the system.
OBS	Statement which represents observations of data flows through the system.
DIAG	Statement which represents whether each component is normal or not.

Model-based diagnosis is usually necessary to describe the behavior of components. However, writing down exact behavior of the software is difficult. To solve this problem, abstract model-based diagnosis has been proposed [1]. There, the logical relations derived from the configuration of the system are acquired by using the formula ”outputs are normal if components and inputs are normal”, so it becomes unnecessary to write down the behavior of the components.

As an example, let us show the steps of the abstract model-based diagnosis using an abstract model shown in Fig. 2 which is used as example in [1]. First, SD is defined as follows.

$$\begin{aligned}\mathbf{SD} \equiv \{ok(C_1) \wedge ok(a) \wedge ok(b) \rightarrow ok(c) \wedge ok(d)\}\\ \wedge\{ok(C_2) \wedge ok(c) \rightarrow ok(e)\}\\ \wedge\{ok(C_3) \wedge ok(d) \rightarrow ok(f)\}\end{aligned} \qquad (1)$$

The first line of the equation (1) shows that outputs c, d are normal if C_1 is normal and inputs a, b are normal.

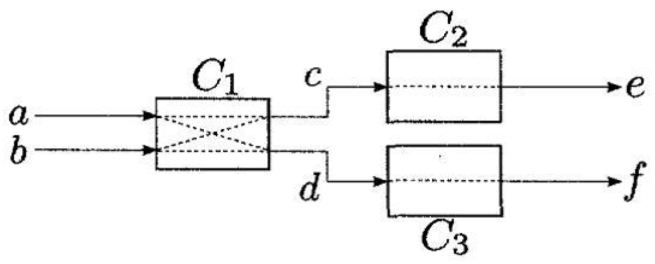


Fig. 2. Example of an abstract model.

In abstract model-based diagnosis, OBS is defined as a result of checking whether each data is normal or not according to some criteria. For example, if c, e are abnormal and other data are normal in Fig. 2, OBS is defined as follows.

$$\begin{aligned}\mathbf{OBS} \equiv\, & ok(a) \wedge ok(b) \wedge \neg ok(c) \\ & \wedge ok(d) \wedge \neg ok(e) \wedge ok(f)\end{aligned} \tag{2}$$

As for the method for determining whether data is normal or not, the following two are considered.

- Methods based on designers knowledge, such as specifying range or period within which the data must be generated.
- Methods that are not based on knowledge, such as using statistical learning or data mining on accumulated data, or building separate model for the interior of the component and using formal methods like model checking.

DIAG shows whether each component is normal or not, and in case component C_1, C_3 are abnormal and component C_2 is normal, DIAG is written as follows.

$$\mathbf{DIAG} \equiv \neg ok(C_1) \wedge ok(C_2) \wedge \neg ok(C_3) \tag{3}$$

$\neg ok(C)$ means that the component C is abnormal. Therefore we will write DIAG as a list of abnormal components in parentheses. For example, DIAG of the equation (3) is represented as $\{C_1, C_3\}$.

Model-based diagnosis seeks appropriate DIAG from SD and OBS. That is, it seeks DIAG based on the condition ”given what DIAG, conflict does not occur between SD and OBS”, which is represented by the following equation.

$$\mathbf{SD} \wedge \mathbf{OBS} \wedge \mathbf{DIAG} = \mathbf{True} \tag{4}$$

When SD is given by the equation (1) and OBS is given by the equation (2), possible values of DIAG are $\{C_1\}$,$\{C_1, C_2\}$,$\{C_1, C_3\}$, $\{C_1, C_2, C_3\}$. Among all the possible DIAGs, minimal diagnosis is defined as the one which contains minimal number of components that does not make (4) contradicting. In the example above, $\{C_1\}$ is the minimal diagnosis. Minimal diagnosis may also contain two or more faulty components, which means that it is possible to identify multiple faults.

As shown in the example above, in abstract model-based diagnosis, sometimes multiple DIAGs are obtained for given OBS and SD. In this case, we consider the probability of multiple abnormal components at the same time to be small, and abstract model-based diagnosis will output the DIAG which is minimal number of abnormal components as the diagnosis result. Thus, in example above, abstract model-based diagnosis outputs a result showing that C_1 is abnormal.

C. *Modeling in MATLAB/Simulink*

We propose the way to model diagnosis systems using abstract model-based diagnosis (in particular, dealing with OBS) similar to modeling the acquisition of Inputs and Outputs in Section II, so that diagnosis is implemented independent from HW or SW.

1) Placement of the diagnosis system: For example, we place the diagnosis system for target model in Fig. 2 as shown in Fig. 3. Here, Diagnosis Component (DC) stands for the component that receives Inputs and Outputs of each component from abstract JTAG (introduced in Section II) to determine normality, and DCC (Diagnosis Component Center) stands for the component that outputs OBS based on results from DCs.

2) Criteria of data normality: As mentioned in Section III-B, in the abstract model-based diagnosis, there are two types of methods for determining data normality. In this study, we use the method based on designers knowledge with the following two criteria.

1. Combination of Inputs and Outputs is determined uniquely.
2. Inputs and Outputs must fall within a certain range.

There is no established format describing information for deciding whether Inputs and Outputs of each component are normal or not in [1], so in this research, we use a matrix called normal data matrix.

The normal data matrix is defined, as shown in Fig. 4. We write the numbers of corresponding components to column C, each input condition to columns of inputs (1,· · ·, m), each output condition to columns of outputs (1,· · ·, n), and last two columns contain position of the boundary between the Input and Output data and the number of criteria.

DC determines whether Inputs and Outputs are normal or not by corresponding criteria as following.

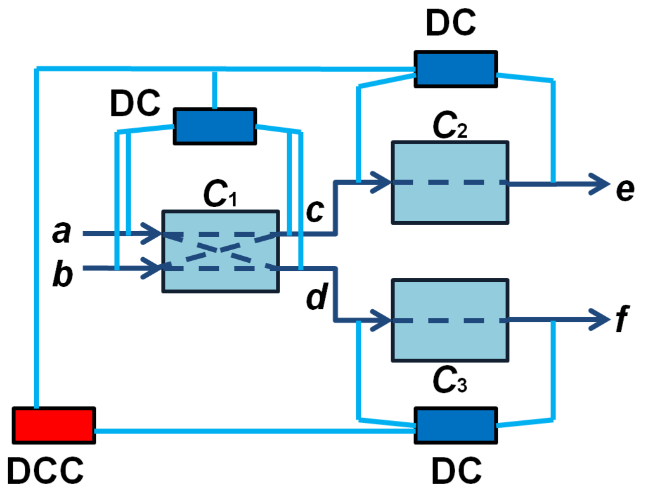


Fig. 3. Placement of diagnosis system.

$$\begin{array}{cccccccc} C & input\ 1 & \cdots & input\ m & output\ 1 & \cdots & output\ n & boundary\ between\ input\ and\ output & criteria \\ \end{array}$$

$$\begin{pmatrix} x & a_1 & \cdots & a_m & b_1 & \cdots & b_n & m+1 & y \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ x & c_1 & \cdots & c_m & d_1 & \cdots & d_n & m+1 & y \end{pmatrix}$$

Fig. 4. Normal data matrix notation.

Criteria 1:

Step 1 Compose normal data matrix that contains combination of the normal I/O of the diagnosis target in each line.
Step 2 Given the observation values of I/O and values of I/O values in normal data matrix, we choose the ones that contain more identical values.
Step 3 Given the observation values and selected combination of I/O, we consider the same I/O as normal and different ones as abnormal.

Criteria 2:

Step 1 Compose normal data matrix that contains the smallest values of I/O of diagnosis target in a first line, and the largest values in a second line.
Step 2 Define I/O that are within the range as normal and others as abnormal.

3) Example of normal data matrix: As an example, here we describe normal data matrix using the model from Fig. 3.

First, we assume that normality for Inputs and Outputs of each component is determined as follows using two criteria mentioned above.

C_1: a	is normal if $-1 \leq a \leq 1$. (Criteria 2)
C_1: b	is normal if $-2 \leq b \leq 2$. (Criteria 2)
C_1: c	is normal if $0 \leq c \leq 1$. (Criteria 2)
C_1: d	is normal if $0 \leq d \leq 1$. (Criteria 2)
C_2: c,e	are normal if value (c,e) is any of the following: (0,0), (0,1), (1,0), (1,1). (Criteria 1)
C_3: d,f	are normal if value (d,f) is any of the following: (0,$-$1), (0,1), (1,1), (1,$-$1). (Criteria 1)

Normal data matrices of C_1, C_2, C_3 are described as follows.

Normal data matrix of C_1:

$$\begin{pmatrix} 1 & -1 & -2 & 0 & 0 & 3 & 2 \\ 1 & 1 & 2 & 1 & 1 & 3 & 2 \end{pmatrix}$$

Normal data matrix of C_2:

$$\begin{pmatrix} 2 & 0 & 0 & 2 & 1 \\ 2 & 0 & 1 & 2 & 1 \\ 2 & 1 & 0 & 2 & 1 \\ 2 & 1 & 1 & 2 & 1 \end{pmatrix}$$

Normal data matrix of C_3:

$$\begin{pmatrix} 3 & 0 & -1 & 2 & 1 \\ 3 & 0 & 1 & 2 & 1 \\ 3 & 1 & -1 & 2 & 1 \\ 3 & 1 & 1 & 2 & 1 \end{pmatrix}$$

4) Rapid increase in the size of normal data matrix: There is a problem that normal data matrix size increases rapidly with the number of I/O of a component and the number of possible value combinations according to criteria 1. To address this problem, we propose either creating a program describing the combination of the normal I/O or using criteria 2. In order to deal with situations where such approach is not possible, it would be necessary to think of other criteria of data normality.

5) Procedures of DC and DCC: Procedures of DC and DCC are described using normal data matrix as follows.

Procedure of DC:

Step 1 Obtain observed value of the I/O of the diagnosis target from abstract JTAG.
Step 2 Get normal data matrix and choose normality criteria.
Step 3 According to normality criteria, determine normality.
Step 4 Report each Input and Output as normal or not to DCC.

Procedure of DCC:

Step 1 Get the information reported from the all DCs.
Step 2 Output OBS from reported information.

D. Modeling in MATLAB/Simulink

User-defined function block enables modeling diagnosis functions in MATLAB/Simulink. Since users can write the process of the block as a program, diagnosis system is modeled by describing the process of DC and DCC like that.

IV. Modularize and reuse functions using aspects

A. Idea

In this study, we propose embedding the diagnosis functions by using aspects and model transformation, so that we are able to reuse them. Because aspect can be applied after the completion of the target model, changing the model and adding functions can be done easily. Also, it allows variations, such as implementation with just diagnosis functions removed.

In this study, we use model transformation with aspects not as specific means for embedding the diagnosis functions, but in a way that enables to use it in the general model.

B. *Model transformation with aspects*

Aspect is a technique that is designed to extract and modularize process that is common in many models or programs. Such process is called Crosscutting Concern [10], and while it is difficult to modularize it only with modularization, aspects perform well in this case.

Join point model, which is a representative model of the aspect, consists of join point, pointcut and advice. Join point is a ”point in code” to which aspect can be applied, and a set of more than one join points is called pointcut. Advice is a set of instructions that indicate what should be handled by aspect code. Aspect can be applied to the target program at compile time or at a run time. Action that adds the functionality defined by advice to a join point specified by a pointcut is called weave, and the utility which performs weave is called weaver.

Models created with MATLAB/Simulink are stored as a code on a layer-structured document. Therefore, in this study, we perform model transformation by changing the contents of the code with aspects. That is, we indicate certain location in code as a pointcut, and change internal structure of model by changing the contents of the code with advice (Fig. 5). For embedding the diagnosis functions modeled by MATLAB/Simulink, we apply aspect that adds blocks, lines and branches of line needed for diagnosis. However, since weaver that can be used in aspect applying is currently incomplete, we apply aspects by hand.

Since the amount of components and the amount and names of each component’s I/O are different depending on diagnosis targets, it is difficult to design common embedding diagnosis functionality. Therefore, we divide diagnosis functionality according to target and common parts, and modularize common parts together. This way we can design efficiently only by changing the point of embedding common part of diagnosis functionality.

In this study we use MATLAB/Simulink for describing model in XML, since we describe aspect using XML format also. The outline of description is shown in Fig. 6. Here, the entire aspect is contained in ⟨aspect⟩ tag, description of pointcut is contained in ⟨pointcut⟩ tag, and advice is contained in ⟨advice⟩ tag. When adding block and line, join point is represented by target system, and for the branches of line, join point is represented by a target line. The action in advice is selected according to type. In this study, the type can be block-add, line-add and branch-add.

V. EXPERIMENTS

A. *Details of the experiment*

We verify feasibility of proposed method using simple test model. As test model, we use Wave.mdl [11] shown in Fig. 7.

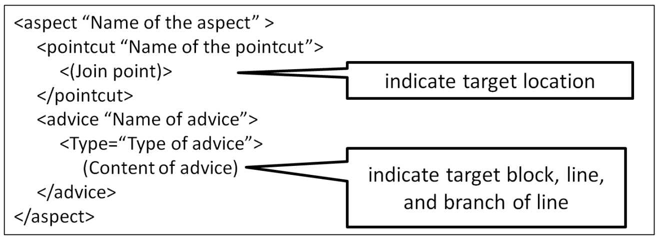
Here, Twice_Component has input of sine wave **a** with amplitude of 1. We regard this component as C_1. C_1 outputs sine wave **b** with amplitude twice as large as **a**. Add_Component has inputs of sine wave **c** with amplitude of 1 and sine wave **b** which is the output of C_1. We regard this component as C_2. C_2 outputs **d** which is the sum of sine waves **b** and **c**. We set time step for performing Input and Output to 0.1, and perform simulation from 0.0 till 6.0.

We embed diagnosis functions to Wave.mdl, and we identify the component that is causing error by obtaining I/O of C_1 and C_2 and detecting error for each component.

We made Wave_DCC_JTAG.mdl shown in Fig. 8, which models the embedding diagnosis functionality. Diagnosis modules may look complicated in the figure, but actually they have a simple structure and does not require much resources to implement.

Criteria for deciding whether Inputs and Outputs of each component are normal or not are as follows.

C_1: a	is normal if $-1 \leq \mathbf{a} \leq 1$. (Criteria 2)
C_1: b	is normal if $-2 \leq \mathbf{b} \leq 2$. (Criteria 2)
C_2: b	is normal if $-2 \leq \mathbf{b} \leq 2$. (Criteria 2)
C_2: c	is normal if $-1 \leq \mathbf{c} \leq 1$. (Criteria 2)
C_2: d	is normal if $-3 \leq \mathbf{d} \leq 3$. (Criteria 2)



```
<aspect “Name of the aspect” >
    <pointcut “Name of the pointcut”>
        <(Join point)>
    </pointcut>
    <advice “Name of advice”>
        <Type=“Type of advice”>
            (Content of advice)
    </advice>
</aspect>
```

Fig. 6. Outline of aspect description.

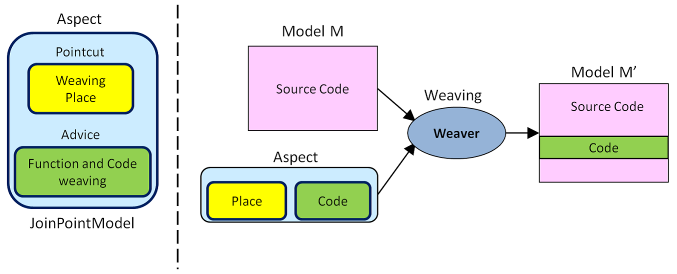


Fig. 5. Model transformation with aspects.

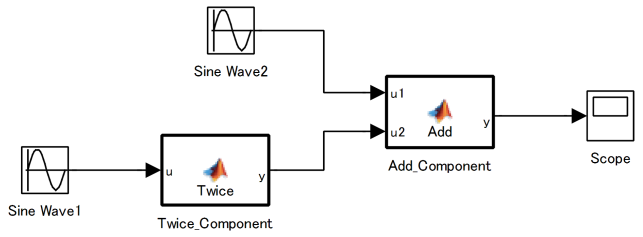


Fig. 7. Test model: Wave.mdl.

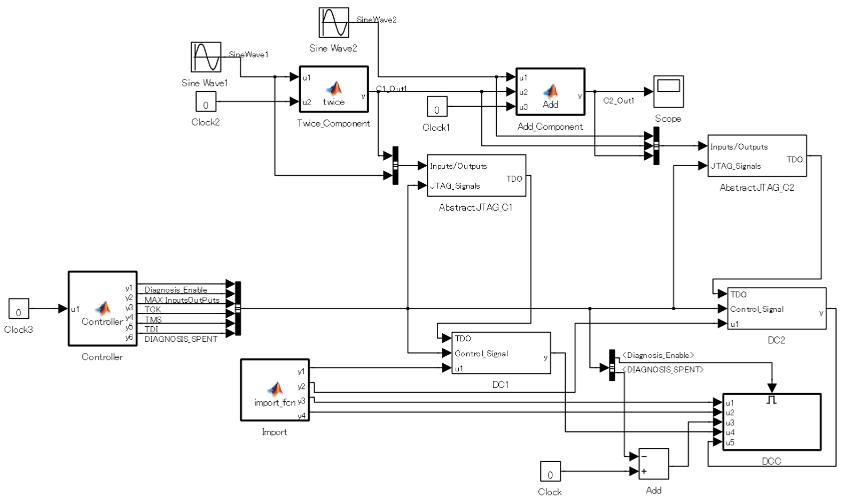


Fig. 8. Model: Wave_DCC_JTAG.mdl.

TABLE I. THE DETAILS OF CAUSING ERRORS

Time	Position	Error details
1.0	C_1	Multiplies input **a** by 5.
3.0	C_2	Multiplies input **b** by -1 before addition.
$4.0 \sim 5.0$	C_2	Multiplies input **c** by 3.
5.0	C_1	Multiplies input **a** by -4.

TABLE II. DIAGNOSIS RESULT

Time	Input and Output	Component
0.0	All normal	All normal
1.0	Output **b** in C_1, input **b** and output **d** in C_2 are abnormal	C_1 is abnormal
2.0	All normal	All normal
3.0	All normal	All normal
4.0	Output **d** in C_2 is abnormal	C_2 is abnormal
5.0	Output **b** in C_1, input **b** and output **d** in C_2 are abnormal	C_1 is abnormal

The diagnosis is performed at time steps 0.0, 1.0, 2.0, 3.0, 4.0 and 5.0. In addition, we raise errors shown in Table I and verify whether diagnosis function can detect them.

B. Experiment results

Experiment results are shown in Table II. The results at 0.0, 1.0, 2.0 and 4.0 are all successful. However, the results at 3.0 and 5.0 show abnormalities. First, at 3.0, output **d** is supposed to become abnormal because we introduced an error to C_2, but the error was not detected, since it did not exceed the range of normal values that were used in the criteria. When using criteria that check if Input and Output fall within a certain range, we can see that detection is not possible if the error does not make Input and Output exceed the range. In addition, at 5.0, we caused an error in both the C_1 and C_2, but only C_1 was identified as abnormal component. The reason for this is that model-based diagnosis outputs minimal diagnosis. That is, if several components become abnormal at the same time, only the component that is the starting point would be identified as abnormal.

VI. CONCLUSION AND FUTURE WORK

In this paper, we described abstract JTAG which monitors I/O of component, and abstract model-based diagnosis which identifies faulty component at modeling stage. We also proposed model transformation using aspects for modularizing and reuse of those functions. This method enables to locate a fault source even if the fault propagates multiple modules.

In the future, we will confirm the utility of proposed method by actually generating HW and SW from MATLAB/Simulink model and evaluating overhead and increase in code size that is caused by embedding diagnosis functions. In addition, since we currently calculate DIAG manually, we need to automate it as well.

REFERENCES

[1] T. Kutsuna, S. Sato, and N. Chyujo, "Fault Location in Collaborative Systems Using Abstract Model Based Diagnosis", Workshop on embedded technology and network (ETNET2009), 2009, pp. 43–48.

[2] Z. Simeu-Abazi, M. Di Mascolo, and M. Knotek, "Fault diagnosis for discrete event systems: Modelling and verification", Reliability Engineering & System Safety, vol. 95, no. 4, 2010, pp. 369-378.

[3] M. Dievart, P. Charbonnaud, and X. Desforges, "An embedded distributed tool for transportation systems health assessment", Embedded Real Time Software and Systems (ERTS2 2010), 2010, pp. 1–10.

[4] P. F. Smith, S. M. Prabhu, and J. Friedman, "Best Practices for Establishing a Model-Based Design Culture", SAE 2007 World Congress, 2007, pp. 1–7.

[5] MATLAB/Simulink, http://www.mathworks.com/products/simulink/ [accessed: 2014-07-08].

[6] Y. Irizuki, M. Ohara, and K. Sakamaki, "An Online Self-Monitoring Approach for Embedded Systems Using JTAG Interface", The journal of Reliability Engineering Association of Japan, vol. 32, no. 3, 2010, pp. 185–190.

[7] IEEE Standard Test Access Port and Boundary-Scan Architecture-Description, 1990.

[8] J. De Kleer and J. Kurien, "Fundamentals of model-based diagnosis", Proceedings of IFAC Safeprocess 3, 2004, pp. 25–36.

[9] J. De Kleer and B. C. Williams, "Diagnosing multiple faults", Artificial Intelligence, 32 (1), 1987, pp. 97–130.

[10] R. Laddad, "AspectJ in Action", Manning, 2003.

[11] mdl, http://www.mathworks.com/help/simulink/ug/saving-a-model.html [accessed: 2014-07-08].