Proceedings of the Korean Nuclear Society Spring Meeting

Cheju, Korea, May 2001

Application of Discrete Function and Software Control Flow to Dependability Assessment of Embedded Digital System

Jong Gyun Choi and Poong Hyun Seong
Department of Nuclear Engineering
Korea Advanced Institute of Science and Technology
373-1 Kusong-dong, Yusong-gu, Taejon, Korea 305-701

Abstract

This article describes a combinatorial model for estimating the reliability of the embedded digital system by means of discrete function theory and software control flow. This model includes a coverage model for fault processing mechanisms implemented in digital system. Furthermore, the model considers the interaction between hardware and software. The fault processing mechanisms make it difficult for many types of components in digital system to be treated as binary state, good or bad. The discrete function theory provides a complete analysis of multi-state system as which the digital system can be regarded Through adaptation software control flow to discrete function theory, the HW/SW interaction is considered for estimation of the reliability of digital system.

Using this model, we predict the reliability of one board controller in a digital system, Interposing Logic System (ILS), which is installed in YGN nuclear power units 3 and 4.

Since the proposed model is general combinatorial model, the simplification of this model becomes a conservative model that treats the system as binary state. Moreover, if information for coverage factor of fault tolerance mechanisms implemented in system through fault injection experiment is obtained, this model can consider detailed interaction of system components.

1. Introduction

The use of digital systems in nuclear instrument and control system (I&C) prevails because of their increased capability and superior performance compared with the analog systems. However, it is very difficult to evaluate the reliability of digital systems because they include the complex fault processing mechanisms at various levels of the systems. Software is another obstacle in reliability assessment of the systems that requires ultra-high reliability. There are ongoing debates in industry, academia, and the international standards community on the problem whether software reliability can be quantified or not [1]. In addition, the reliability of digital systems has to be assessed considering software, hardware and SW/HW interactions because the software consideration cannot be fully understood apart from hardware considerations and vice versa [2-4]. In the hierarchical functional view of a digital system shown in Figure 1, the software system is designed to accomplish functions that the digital system is required to perform. The software system is composed of software modules. The software modules perform their allotted tasks through the combination of instruction sets provided by the microprocessor. The parts of hardware components such as microprocessors and memories are used for processing of one instruction. That is, in order that the digital system completes its required function, the software determines the correct sequence in which the hardware resources should be used. The failure of system, thus, occurs when the software cannot arrange the sequence of use of the hardware resources correctly or when the one or more of used hardware resources have the faults though the software has determined the correct sequences of use of hardware resources.

Many techniques have been discussed for analyzing the reliability of systems. There are three classical models for analyzing system reliability: Reliability Block Diagram (RBD) technique, Fault Tree Analysis (FTA) technique, and Markov modeling technique [5]. In order to assess reliability of system, RBD and FTA techniques use combinatorial models in which the causes of system failure can be expressed in terms of combinations of component failures. Combinatorial models include graph models used for the analysis of network reliability, fault trees and reliability block diagram.

Combinatorial models have been considered inappropriate for modeling coverage of fault tolerant systems. However, coverage model of fault tolerant system was developed to incorporate coverage modeling into combinatorial models [6-7]. The method for analyzing multi-state systems developed in [8] provides the concept for describing systems that are composed of a hierarchy of levels.

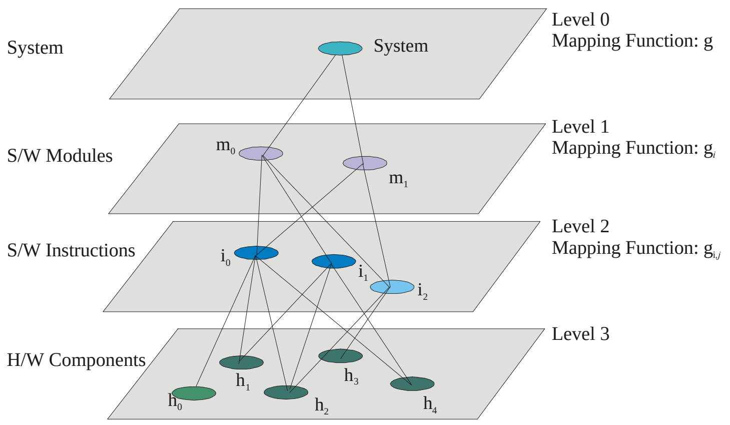

Figure 1. Hierarchical Functional Architecture of Digital System at Board Level

This work describes a combinatorial model for estimating the reliability of the nuclear digital instrumentation and control system by means of discrete function theory. This model includes not only coverage model of fault processing mechanisms implemented in digital system but also model that considers the interaction between hardware and software. The discrete function theory [9] provides a complete analysis of multi-state system as which the digital system can be regarded since the fault processing mechanisms make it difficult for many types of components in system to be treated as binary state, good or bad. In this work, the concept of coverage model provided in [6] is extended for modeling the fault tolerance mechanisms implemented hierarchically in digital system. Since, when the system reliability is estimated, the software should not be considered separately from the hardware, the effects of software control flow on system reliability is considered.

Using this model, we predict the reliability of one board controller in a digital system, Interposing Logic System (ILS), which is installed in YGN nuclear power units 3 and 4. Since the proposed model is general combinatorial model, the simplification of this model becomes a conservative model that treats the system as binary state. Moreover, if information for coverage factor of fault tolerance mechanisms implemented in system through fault injection experiment is obtained, this model can consider detailed interaction of system components.

2. Model

2.1. Discrete Function Theory

2.1.1 Logic Network

Figure 2 shows the elementary logic gates, AND/OR. Each inputs of logic gates can have integer values, 0, 1, 2, .., n. The output of AND gate is the minimum of all the inputs and the output of OR gate is the maximum of all the inputs. A logic network is then defined as circuit composed of these gates.

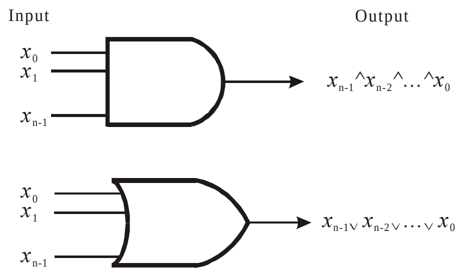
Input
Output
x_0
x_1
x_{n-1}
$x_{n-1} \wedge x_{n-2} \wedge \dots \wedge x_0$
x_0
x_1
x_{n-1}
$x_{n-1} \vee x_{n-2} \vee \dots \vee x_0$

Figure 2. Logic Gates

2.1.2 Discrete Function, Integer Function and Logic Function

A function $f : S \rightarrow L$ is a discrete function when the sets $\boldsymbol{S}$ and $\boldsymbol{L}$ are finite non-empty sets. If the domain $\boldsymbol{S}$ is the Cartesian product of n finite sets $\boldsymbol{S_i}$, the domain $\boldsymbol{S}$ is denoted by

$S = S_{n-1} \times S_{n-2} \times \Lambda \times S_1 \times S_0 \equiv \prod_{i=n-1}^{0} S_i$.

A discrete function is an integer function defined as $f : \prod_{i=n-1}^{0} S_i \to L$, where each of the sets ***L*** and ***S****ᵢ* are formed by non-negative integers. Therefore, an integer function is a mapping $f : \prod_{i=n-1}^{0} \{0,1,..., m_i - 1\} \to \{0,1,..., r - 1\}$ with *r* and m_i the cardinalities of the sets ***L*** and $\boldsymbol{S_i}$ respectively.

An integer function is a logic function when the sets ***L*** and $\boldsymbol{S_i}$ (i = 0, 1, .., *n*-1) have the same cardinality. Thus, a logic function is a mapping with *r* the cardinality of the sets ***L*** and $\boldsymbol{S_i}$ such that $f : \{0,1,..., r-1\}^n \to \{0,1,..., r-1\}$.

For example, when a system that is composed of two components can be in three states, 0 (Successful Operation), 1 (Undetected Failure) and 2 (Detected Failure) and both of two components also have three states respectively, the system is in *s* state when maximum state of states of the both components is *s*. Therefore, the system can be modeled mathematically by logic function (OR gate) with *r* = 3 and *n* = 2. When state variable of component *i* is x_i, this system has a graphical representation as shown in Figure 3 and the mapping function of this system has tabular form as shown in Table 1.

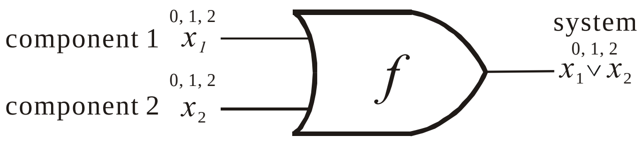

Figure 3. Modeling of digital system composed of two components by OR logic gate

	Component 2 (x_2)		
Component 1 (x_1)	**0**	**1**	**2**
0	0	1	2
1	1	1	2
2	2	2	2

Table 1. Function Table of example given by Figure 3

2.1.3 Computation Schemes and discrete functions of functions

A computation scheme *s* is defined as a finite sequence $(f_0, f_1,..., f_{N-1})$ of functions f_j (j = 0, 1, .., *N*-1) when there are exist a function h_i and *n* functions, $f_{j_0}, f_{j_1}, .., f_{j_{n-1}}$ ($f_{j_k} \in \sigma$; $j_k < j$) such that $f_j = h_i(f_{j_0}, f_{j_1},..., f_{j_{n-1}})$, where $f_j : S^n \to S$ and $h_i : S^n \to S$. That is to say, the computation scheme is the computation sequence for obtaining the output of logic network constructed by an interconnection of many logic gates. Figure 4 shows an example of a network composed of 4 logic gates with three inputs, x, y and z. Table 2 shows the computation scheme of the network given by Figure 4. First of all, for obtaining the output s of logic network, intermediate output o_1 is calculated using inputs x and y through function denominated as 1, Then, intermediate outputs o_2 and o_3 are calculated using input z through functions denominated as 2 and 3 respectively. Finally, the output s is computed using intermediate outputs o_2 and o_3 through function denominated as 4. Therefore, the computation scheme of logic network provide by Figure 4 is (1, 2, 3, 4).

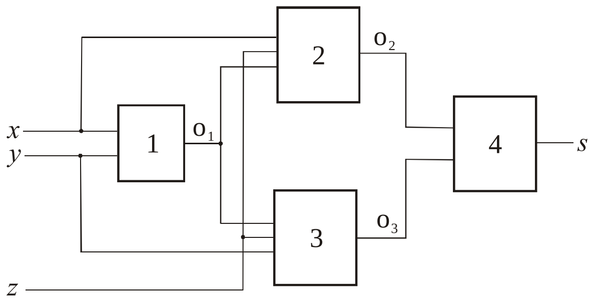

Figure 4. A network composed of 4 functions

Function denomination	**Operation**
x	x
y	y
z	z
1	$H_1(x, y)$
2	$H_2(x, z, 1)$
3	$H_3(y, z, 1)$
4	$H_4(2, 3)$

Table 2. Computation scheme of the network given by Figure 4

2.2. Coverage Modeling of System components

Generally digital systems are composed of a hierarchy of levels [10]. Faults and errors may be generated at any of the levels in the hierarchy. Figure 6 is an example of a hypothetical system composed of three hierarchical levels. If an error is not detected at the level in which it originated, the detection of the error is left to higher levels. Similarly, if the current level lacks the capacity to recover from a particular detected error, appropriate information about the detected error must be passed onto a higher level.

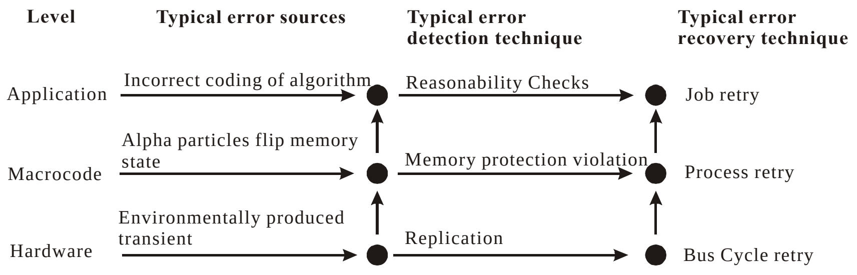
Level
Typical error sources
Typical error detection technique
Typical error recovery technique
Application
Incorrect coding of algorithm
Reasonability Checks
Job retry
Macrocode
Alpha particles flip memory state
Memory protection violation
Process retry
Hardware
Environmentally produced transient
Replication
Bus Cycle retry

Figure 5. Typical error and recovery techniques at multiple system levels due to imperfect coverage

Figure 6 shows the general structure of a coverage model representing a recovery process initiated when a fault occurs. The entry point to the model signifies the occurrence of the fault and the 3 exits signify 3 possible outcomes. The transient recovery exit (labeled TR) represents the correct recognition of, and recovery from, a transient fault. Successful recovery from a transient fault restores the system to a consistent state without discarding any components. The detected fault exit (DF) denotes the determination of the permanent nature of the fault. Although the fault is detected, recovery from the detected fault is passed onto a higher level because recovery techniques of the current level lacks the capacity to recover from the detected fault, The undetected fault exit (labeled UF) is reached when a fault is not even detected. This type of fault can cause the higher-level components to operate erroneously. Each exit in the coverage model has an exit probability associated with it which is determined by solving the appropriate coverage model. Thus we define [TR_{ij}, UF_{ij}, DF_{ij}] to be the probability of taking the [Transient Recovery, Undetected Fault, Detected Fault] exit, given that a fault occurs. The three exits are mutually exclusive and their probabilities sum to 1.

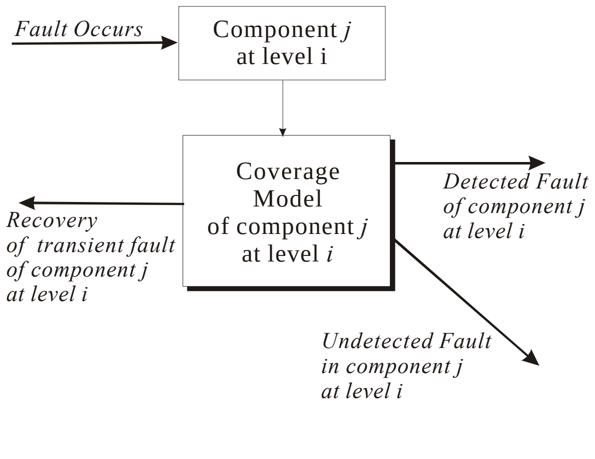
Fault Occurs
Component j at level i
Coverage Model of component j at level i
Recovery of transient fault of component j at level i
Detected Fault of component j at level i
Undetected Fault in component j at level i

Figure 6. Coverage Model of Fault Tolerant System

2.3 Modeling of Embedded System

2.3.1 Modeling of Hardware Components

To determine state probability, it is assumed that a component has the three states, 0, 1 and 2. The states of a component are given by

- 0: the component operates correctly
- 1: the component operates erroneously, but the coverage model does not know this.
- 2: the component operates erroneously, and the coverage model knows this.

If the failure of hardware component *i* is distributed exponentially and the failure rate of the hardware component is λ_i, then cumulative failure probability is $F(t) = 1 - e^{-l_i t}$. When the state variable of a hardware component *i* is x_i, state probability is given by

$Pr\{x_i = 0\} = Pr\{$no faults occurs in component $i\} + TR_{3i} \cdot Pr\{$fault occurs in component $i\}$

$$= e^{-lt} + TR_{3i} \cdot (1 - e^{-l_i t}) = TR_{3i} + (1 - TR_{3i})e^{-l_i t},$$

$Pr\{x_i = 1\} = UF_{3i} \cdot Pr\{$fault occurs in component $i\} = UF_{3i} \cdot (1 - e^{-l_i t}) = UF_{3i} - UF_{3i}e^{-l_i t}$,

$Pr\{x_i = 2\} = DC_{3i} \cdot \{$fault occurs in component $i\} = DC_{3i} \cdot (1 - e^{-l_i t}) = DC_{3i} - DC_{3i}e^{-l_i t}$,

where $Pr\{x_i = 0\} + Pr\{x_i = 1\} + Pr\{x_i = 2\} = 1$ and $TR_{3i} + UF_{3i} + DF_{3i} = 1$.

2.3.2 Modeling of Software Instructions

The state of an instruction execution depends on not only the state of all hardware resources required for the instruction execution and instruction itself. That is to say, in order for one instruction to be executed successfully, all of hardware resources required for the instruction execution must operates correctly and the instruction itself must have no fault that implemented into instruction by coding errors. The state of the instruction is defined by

- 0: the instruction operates correctly
- 1: the instruction operates erroneously, but the coverage model at instruction level does not know this.
- 2: the instruction operates erroneously, and the coverage model at instruction level knows this.

When the state variable of the instruction is *y*, state probability is given by

$Pr\{y = 0\} = Pr\{$no faults occurs in the instruction $\} + TR_{2i} \cdot Pr\{$fault occurs in the instruction $\} =$

$$p_i + TR_{2i} \cdot (1 - p_i) = TR_{2i} + (1 - TR_{2i})p_i,$$

$Pr\{y = 1\} = UF_{2i} \cdot Pr\{$fault occurs in the instruction$\} = UF_{2i} \cdot (1 - p_i) = UF_{2i} - UF_{2i}p_i$,

$Pr\{y = 2\} = DC_{2i} \cdot \{$fault occurs in the instruction $\} = DC_{2i} \cdot (1 - p_i) = DC_{2i} - DC_{2i}p_i$,

where $Pr\{y = 0\} + Pr\{y = 1\} + Pr\{y = 2\} = 1$ and $TR_{2i} + UF_{2i} + DF_{2i} = 1$.

For example, an Assembly code, **ANI 01(Assembly 8085)**, uses hardware resources;

Microprocessor and ROM. This instruction is thus modeled by a logic OR gate as shown in Figure 7.

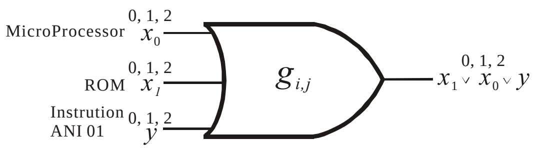

Figure 7. Model of Software Instruction

2.3.3 Modeling of Software Modules

Each of the software modules, m_0, m_1, ..,. m_i is composed of instruction sets. The state of software module is dependent on states of its instruction sets. That is to say, in order for software module to execute its intended function successfully, all of instructions executed by the software module must operate successfully. Therefore, each of the software modules, g_1, g_2, ..,. g_i is a logic function of functions. The state of the module is defined by

- 0: the module operates correctly
- 1: the module operates erroneously, but the coverage model at module level does not know this.
- 2: the module operates erroneously, and the coverage model at module level knows this.

When the state variable of the module is z, state probability is given by

$Pr\{z = 0\} = Pr\{\text{no faults occurs in the module }\} + TR_{1i}\cdot Pr\{\text{fault occurs in the module }\}$

$$= q_i + TR_{1i} \cdot (1 - q_i) = TR_{1i} + (1 - TR_{1i})q_i \,,$$

$Pr\{z = 1\} = UF_{1i}\cdot Pr\{\text{fault occurs in the module}\} = UF_{1i} \cdot (1 - q_i) = UF_{1i} - UF_{1i}q_i \,,$

$Pr\{z = 2\} = DC_{1i}\cdot\{\text{fault occurs in the module }\} = DC_{1i} \cdot (1 - q_i) = DC_{1i} - DC_{1i}q_i \,,$

where $Pr\{z = 0\} + Pr\{z = 1\} + Pr\{z = 2\} = 1$ and $TR_{1i} + UF_{1i} + DF_{1i} = 1$.

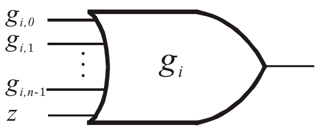

Figure 8. Model of Software Module

2.3.4 Application of Software Control Flow to Logic Network

The operational profile of the embedded system determines the control flow of the software. If ***I*** is the input domain set of the software, then it can be partitioned into an indexed family $\{I_i\}$ with the following properties:

(a) $\mathbf{I} = \mathrm{Y}_{i=0}^{n-1}\mathbf{I}_i$,

(b) $i \neq j \Rightarrow \ \mathbf{I}_i \cap \mathbf{I}_j = f$.

To consider control flow of software by input domain, we define a set ***C*** called control set and a set ***SW****ᵢ* as $C = \{0,1,\ldots,n-1\}$ and $SW_i = \{r_{0,i},\ r_{1,i},\ \ldots,\ r_{n-1,i}\}$ respectively. *n* is the number of input domains and element $r_{k,i}$ of $\boldsymbol{SW_i}$ is a binary random number defined by

$$r_{k,i} = \begin{cases} r-1 \text{ if module } \mathrm{g_i} \text{ is in sequence of software execution by input domain } \mathbf{I}_\mathrm{k} \\ 0, \text{ otherwise} \end{cases}$$

The element *k* of set ***C*** means that the input domain $\boldsymbol{I_k}$ is selected for software execution. Additionally, the switching function is defined as $sf : C \rightarrow \prod_{i=0}^{p-1} SW_i$, where *p* is the number of software module. Therefore, the state of digital system is obtained by computation scheme (g_0, g_1, .., g_{p-1}, *sf*, *f*).

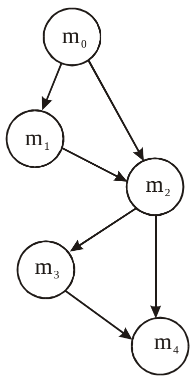

Figure 9. Control flow of example software

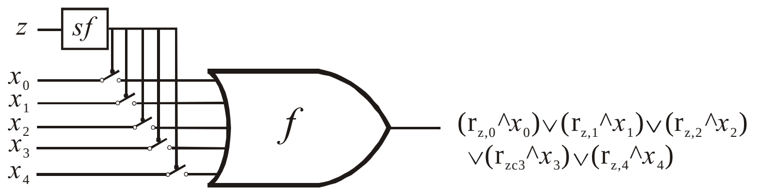
z
sf
x_0
x_1
x_2
x_3
x_4
f
$(r_{z,0}\wedge x_0)\vee(r_{z,1}\wedge x_1)\vee(r_{z,2}\wedge x_2)$
$\vee(r_{zc3}\wedge x_3)\vee(r_{z,4}\wedge x_4)$

Figure 10. Logic gate of Example Software of Figure 6

For example, Figure 9 shows the control flow of software. It is assumed that all of software modules have the same state se, {0, 1, 2}. In this figure, the path number of software is 4. Thus, the input domain can be partitioned into {I_0, I_1, I_2, I_3} and software control set ***C*** is {0, 1, 2, 3}. It is assumed that the software executes the sequence of software modules, g_1, g_2, g_3, g_4 and g_5 as follows:

- If $i \in I_0$, then, $m_0 \rightarrow m_1 \rightarrow m_2 \rightarrow m_3 \rightarrow m_4$
- If $i \in I_1$, then, $m_0 \rightarrow m_1 \rightarrow m_2 \rightarrow m_4$
- If $i \in I_2$, then, $m_0 \rightarrow m_2 \rightarrow m_3 \rightarrow m_4$

- If $i \in I_3$, then, $m_0 \rightarrow m_2 \rightarrow m_4$

When x_i is the state variable of module g_i and z is the state variable of software control set **C**, the logic network of Figure 9 is shown in Figure 10.

3. Model Application to ILS system

The target ILS software is a part of AFS-1000 system developed by Forney International Cooperation and installed in YGN nuclear power units 3 and 4. It is written in the Intel 8085 assembly language using top-down modular design techniques. Reliability of ILS software is predicted only considering the software failure by memory faults.

Hardware		**Application**	
CPU	8085	SIZE	72 Byte
MEMORY	64k EPROM	EXECUTION TIME	379 Clock time
CLOCK FREQ.	1 Mhz	CHECKING PERIOD	5 Minutes

Table 2. Information of application software

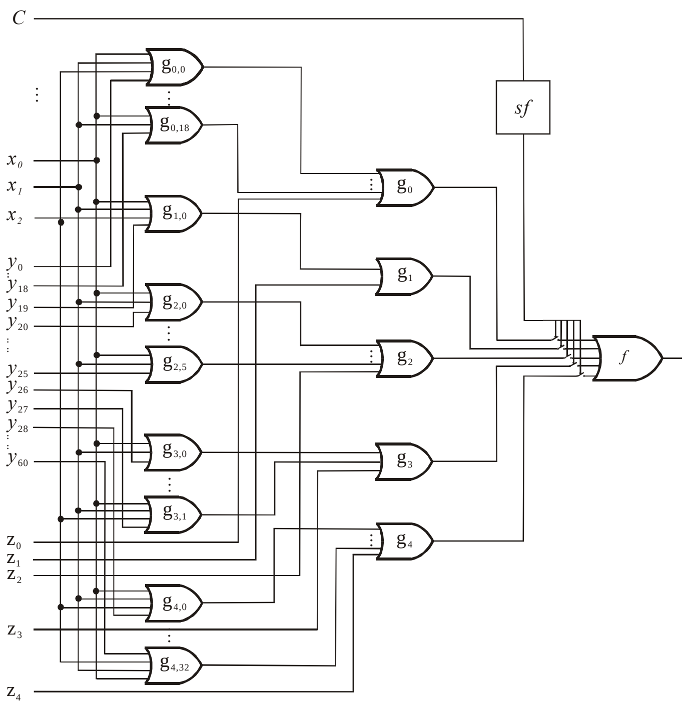

Figure 11. Logic Network of application software

As shown in Table 2, the memory in the system is the Erasable Programmable Read Only

Memory (EPROM) and the capacity of the memory is 64k bites. The clock frequency is 1Mhz. The sample program that is shown in Appendix is a part of the executive program that consists of the various subroutines that generate the logic to perform miscellaneous functions (ANDs, ORs, Counters, etc.).

Appendix shows the Assembly code of application software. The first column denominates the module function. The second column denominates the mapping functions of instructions. The third column represents the hardware components. Final column describes assembly codes of application software. Figure 11 shows the logic network of application software.

4. Application Results

For model application to ILS system, the failure rates of all hardware components are assumed to be10^{-7}/hr. Figure 12 shows the state probability of system when any fault tolerance mechanisms are not considered and Input domain of software is always $\boldsymbol{I}_0$. That is to say, (TR_{ij}, UF_{ij}, Df_{ij}) = (0, 1, 0) and $\boldsymbol{I} = \boldsymbol{I}_0$. This result is equal to the result calculated by part count method used in MIL-HNBK-217. Therefore, it indicates that the simplification of model proposed in this work is combinatorial model in which the system and components are regarded binary state, GOOD/BAD.

Figure 13 shows the state probability when the fault tolerance mechanisms at only hardware component level is considered and Input domain of software is always $\boldsymbol{I}_0$. That is to say, (TR_{3j}, UF_{3j}, Df_{3j}) = (0.8, 0.1, 0.1), j = 0, 1, 2 and $\boldsymbol{I} = \boldsymbol{I}_0$. The result indicates that state probability of system goes to steady state value. The steady state value depends on the values, TR_{3j}, UF_{3j} and Df_{3j}

Figure 14 shows the state probability when any fault tolerance mechanisms are not considered and Input domain of software is always $\boldsymbol{I}_3$. That is to say, (TR_{3j}, UF_{3j}, Df_{3j}) = (0, 1, 0), j = 0, 1, 2 and $\boldsymbol{I} = \boldsymbol{I}_3$. Although the shape of state probability is similar to that of Figure 12, the values of state probability shows the different result with that of Figure 12.

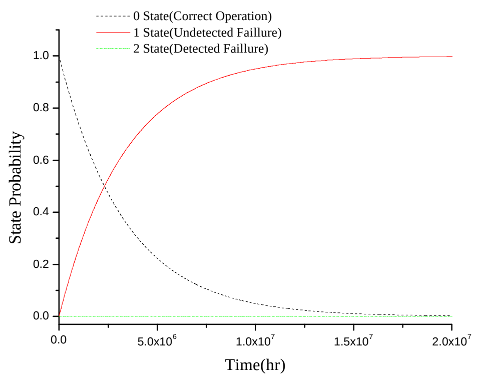

Figure 12. State probability of system without fault tolerance and software consideration

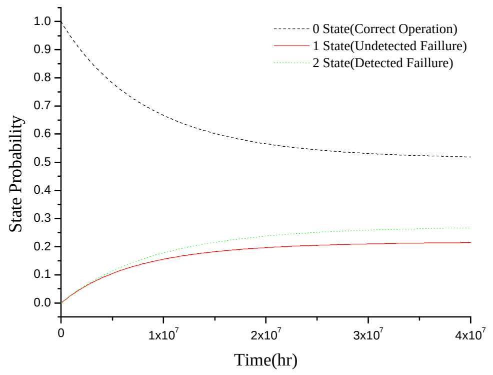

Figure 13. State probability of system with fault tolerance of hardware components

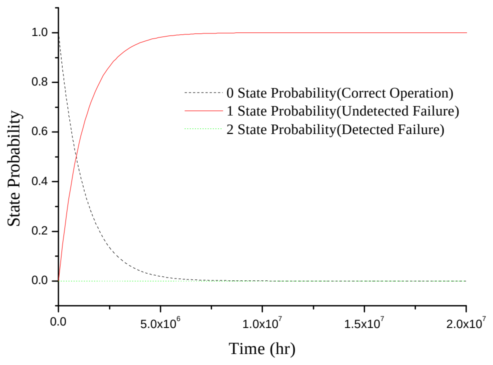

Figure 14. State Probability of system with consideration of Software Control Flow but without consideration of fault tolerance mechanisms

5. Conclusions

In this work a combinatorial model was described for estimating the reliability of the nuclear digital instrumentation and control system by means of discrete function theory. This model includes not only coverage model of fault processing mechanisms implemented in digital system but also model that considers the interaction between hardware and software.

Since the proposed model is general combinatorial model, the simplification of this model becomes a conservative model that treats the system as binary state. Moreover, if information for coverage factor of fault tolerance mechanisms implemented in system through fault injection experiment is obtained, this model can consider detailed interaction of system components.

This modeling method is particularly attractive for embedded system in which medium size software is implemented such as digital protection systems of nuclear power plants since it requires very laborious work to be applied to systems in which large software is implemented.

Reference

[1] R. W. Butler and G. B. Finelli, “The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software”, IEEE Transactions on Software Engineering, Vol. 19, No. 1, pp. 3-12, January 1993.

[2] J. G. Choi and P. H. Seong, “Software Dependability Models under Memory Faults with Application to a Digital system in Nuclear Power Plants”, Reliability Engineering and System Safety, No. 59, pp. 321-329, 1998.

[3] K. K. Goswami and R. K. Iyer, “Simulation of Software Behivior Under Hardware Faults,” Proc. On Fault-Tolerant Computing Systems, pp. 218-227, 1993.

[4] J. C. Laprie and K. Kanoun, “X-ware reliability and availability modeling,” IEEE Trans. Software Eng., Vol. 18, No. 2, pp. 130-147, Feb. 1992.

[5] N. Viswanadham and M.G. Singh, “Reliability of computer and control systems”, North-Holland ed., 1987.

[6] S. A. Doyle, J. B. Dugan and F. A. Patterson-Hine, “A Combinatorial Approach to Modeling Imperfect Coverage”, IEEE Trans. Reliability, Vol. 44, No. 1, pp. 87-94, March, 1995.

[7] M. Veeraraghavan and Kishor S. Trivedi, “A Combinatorial Algorithm for Performance and Reliability Analysis Using Multistate Models”, IEEE Trans. Computers, Vol. 43, No. 2, pp. 229-234, Feb. 1994.

[8] X. Janan, “On multistate system analysis,” IEEE Trans. Reliability, Vol. R-34, pp. 329-337. Oct. 1985.

[9] M. Davio, J. P. Deschamps and A. Thayse, “Discrete and Switching Functions”, McGraw-Hill ed, 1978.

[10] D. P. Siewiorek and R. S. Swarz, “Reliable Computer Systems: Design and Evaluation”, A K Peters ed, 1998.

Appendix

Level 1	Level 2	Level 3	Code			
g_0	$g_{0,0}$	MicroP, ROM, RAM	0000	MAN_AUTO:	LXI	H, FAOF
	$g_{0,1}$	MicroP, ROM, RAM	0003		LXI	D, FGPF
	$g_{0,2}$	MicroP, ROM	0006		MOV	B, M
	$g_{0,3}$	MicroP, ROM	0007		INX	H
	$g_{0,4}$	MicroP, ROM	0008		MOV	A, M
	$g_{0,5}$	MicroP, ROM	0009		INX	H
	$g_{0,6}$	MicroP, ROM	000A		MOV	C, A
	$g_{0,7}$	MicroP, ROM	000B		CMA	
	$g_{0,8}$	MicroP, ROM	000C		ANA	B
	$g_{0,9}$	MicroP, ROM	000D		MOV	B, A
	$g_{0,10}$	MicroP, ROM, RAM	000E		LDA	FAOF+13D
	$g_{0,11}$	MicroP, ROM	0011		ORA	C
	$g_{0,12}$	MicroP, ROM	0012		ANA	M
	$g_{0,13}$	MicroP, ROM	0013		INX	H
	$g_{0,14}$	MicroP, ROM	0014		ORA	B
	$g_{0,15}$	MicroP, ROM	0015		ORA	M
	$g_{0,16}$	MicroP, ROM	0016		INX	H
	$g_{0,17}$	MicroP, ROM	0017		ANI	01
	$g_{0,18}$	MicroP, ROM	0019		JZ	AUTO_MAN1
g_1	$g_{1,0}$	MicroP, ROM, RAM	001C		STAX	D
g_2	$g_{2,0}$	MicroP, ROM	001D	AUTO_MAN1	MOV	B, M
	$g_{2,1}$	MicroP, ROM	001E		INX	H
	$g_{2,2}$	MicroP, ROM	001F		MOV	A, M
	$g_{2,3}$	MicroP, ROM	0020		INX	H
	$g_{2,4}$	MicroP, ROM	0021		ANI	01
	$g_{2,5}$	MicroP, ROM	0023		JZ	AUTO_MAN2
g_3	$g_{3,0}$	MicroP, ROM	0026		XRA	A
	$g_{3,1}$	MicroP, ROM, RAM	0027		STAX	D
g_4	$g_{4,0}$	MicroP, ROM, RAM	0028	AUTO_MAN2	LDAX	D
	$g_{4,1}$	MicroP, ROM	0029		INX	D
	$g_{4,2}$	MicroP, ROM	002A		MOV	C, A
	$g_{4,3}$	MicroP, ROM	002B		MOV	A, B
	$g_{4,4}$	MicroP, ROM	002C		CMA	
	$g_{4,5}$	MicroP, ROM, RAM	002D		STAX	D
	$g_{4,6}$	MicroP, ROM	002E		INX	D
	$g_{4,7}$	MicroP, ROM	002F		MOV	A, M
	$g_{4,8}$	MicroP, ROM	0030		INX	H
	$g_{4,9}$	MicroP, ROM	0031		ANA	C
	$g_{4,10}$	MicroP, ROM	0032		ANA	M
	$g_{4,11}$	MicroP, ROM	0033		INX	H
	$g_{4,12}$	MicroP, ROM, RAM	0034		STAX	D
	$g_{4,13}$	MicroP, ROM	0035		INX	D
	$g_{4,14}$	MicroP, ROM	0036		MOV	A, M
	$g_{4,15}$	MicroP, ROM	0037		INX	H
	$g_{4,16}$	MicroP, ROM	0038		ANA	C
	$g_{4,17}$	MicroP, ROM	0039		ANA	M
	$g_{4,18}$	MicroP, ROM	003A		INX	H
	$g_{4,19}$	MicroP, ROM, RAM	003B		STAX	D
	$g_{4,20}$	MicroP, ROM	003C		INX	D
	$g_{4,21}$	MicroP, ROM	003D		MOV	A, M
	$g_{4,22}$	MicroP, ROM	003E		INX	H
	$g_{4,23}$	MicroP, ROM	003F		CMA	
	$g_{4,24}$	MicroP, ROM, RAM	0040		STAX	D
	$g_{4,25}$	MicroP, ROM	0041		INX	D
	$g_{4,26}$	MicroP, ROM	0042		MOV	A, M
	$g_{4,27}$	MicroP, ROM	0043		ORA	C
	$g_{4,28}$	MicroP, ROM, RAM	0044		STAX	D
	$g_{4,29}$	MicroP, ROM	0045		INX	D
	$g_{4,30}$	MicroP, ROM	0046		MOV	A, C
	$g_{4,31}$	MicroP, ROM	0047		CMA	
	$g_{4,32}$	MicroP, ROM, RAM	0048		STAX	D

Assembly code of application software