Testing of Timer Function Blocks in FBD

Eunkyoung Jee, Seungjae Jeon, Hojung Bang, Sungdeok Cha
Div. of Computer Science
Korea Advanced Institute of Science and Technology
Daejeon, Republic of Korea
{ekjee,sjjeon,hjbang,cha}@dependable.kaist.ac.kr

Junbeom Yoo
Next Generation Mobile Equipment Team
Samsung Electronics Co., Ltd.
Suwon-si, Gyeonggi-do, Republic of Korea
junbeom.yoo@samsung.com

Geeyong Park, Keechoon Kwon
I&C/Human Factors Division
Korea Atomic Energy Research Institute
Daejeon, Republic of Korea
{gypark,kckwon}@kaeri.re.kr

Abstract

Testing for time-related behaviors of PLC software is important and should be performed carefully. We propose a structural testing technique on Function Block Diagram(FBD) networks including timer function blocks. In order to test FBD networks including timer function blocks, we generate templates for timer function blocks and transform a unit FBD into a flowgraph using the proposed templates. We apply existing testing techniques to the generated flowgraph and describe how the characteristics of timer function blocks are reflected in the testing process. By the proposed method, FBD networks including timer function blocks can be tested thoroughly without the intermediate code which was essential in the previous FBD testing. To demonstrate the effectiveness of the proposed method, we use a trip logic of bistable processor of digital plant protection systems which is being developed in Korea.

1. Introduction

Testing of safety critical software is an indispensable step to assure software quality because the failures of safety critical software can cause serious damage to human life or property.

This work focuses on the programable logic controller(PLC) programs implemented by Function Block Diagram(FBD), one of the most widely used standard PLC programming languages. As existing analog systems have been replaced by digital systems controlled by software, testing of digital control systems has become more important in nuclear power plant control systems.

An FBD program is automatically compiled to PLC machine code and executed on PLC. In the previous case[1], functional testing on FBD has been done on the intermediate C source code transformed from an FBD network. We propose a structural testing method to test FBD networks including timer function blocks without having to generate the intermediate code. In [4], for the FBD testing, a unit FBD is transformed into a flowgraph and existing structural testing techniques are applied to the flowgraph. However, it did not address how timer function blocks could be tested. Because many PLC programs use timer function blocks and misused timer function blocks can cause serious errors, testing of timer function blocks is essential.

In this paper, we extend work reported in [4] by defining flowgraph segment templates corresponding to the timer function blocks. The proposed method makes systematic structural testing for FBD networks including timer function blocks possible. This method also has an advantage that it can be applied to any FBD program, whatever its intermediate format is. We demonstrate our approach by applying it to a trip logic of Bistable Processor(BP) of Reactor Protection Systems(RPS) which is being developed in Korea Nuclear Instrumentation and Control System R&D Center(KNICS)[2]. We confirm that various errors including timer function block errors of a unit FBD can be found effectively by applying the proposed method.

The remainder of the paper is organized as follows:

XIII ASIA PACIFIC SOFTWARE ENGINEERING CONFERENCE (APSEC'06)
0-7695-2685-3/06 $20.00 © 2006 IEEE

section 2 introduces FBD, and section 3 describes the template generation for the timer function blocks to transform a unit FBD including timer function blocks into a flowgraph. In section 4, we apply control flow and data flow testing strategies to the flowgraph transformed from a unit FBD. Finally, the conclusion and future works are described in section 5.

2. Background

2.1. Function Block Diagram

A PLC[6] is an industrial computer widely used in control systems. The standard PLC programming languages identified in IEC 61131-3[3] are Structured Text(ST), Function Block Diagram(FBD), Ladder Diagram(LD), Instruction List(IL), and Sequential Function Chart(SFC). FBD, one of the most widely used PLC languages, is easy to understand and good for representing data flow between control blocks.

FBD is based on viewing a system in terms of the flow of signals between processing elements[5]. Figure 1 shows five function block groups and representative examples of each group. RPS, which is being developed at KNICS, is programmed with function blocks out of five groups in figure 1.

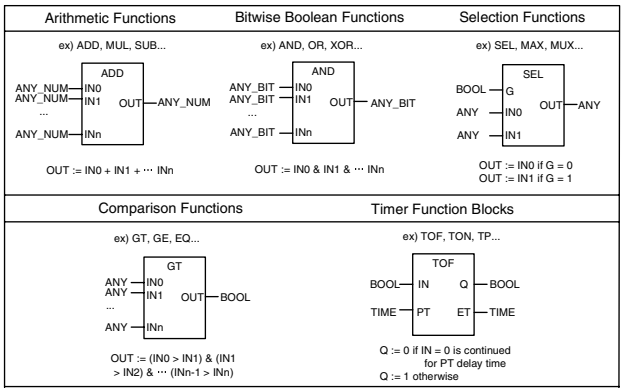


Figure 1. Representative examples of function and function blocks

Figure 2 shows an example FBD network to calculate *th_Prev_X_Trip*. This is a part of fixed set-point falling trip logic. *th_Prev_X_Trip* is set to *true* if the processing value has been beyond the trip set-point for the specified delay time or the value validity error, module error or channel error occurred.

We will use the FBD in figure 2 as an example to represent our approach in following sections. To show the effectiveness of the proposed testing method, we intentionally inserted 4 kinds of errors into this FBD. We switched inputs of 1.47(7) LE_INT and changed input variable name of 1.23(8) SEL from *th_Prev_X_Trip* into *th_Prev_Trip*. We used TON function block instead of FBI_1.10(9) TOF function block and omitted an inverter attached to *f_X_Valid* input of 1.13(12) AND_BOOL.

Execution of FBD blocks is deterministic in that they are executed sequentially in each scan cycle by the predetermined order. The number inside parentheses on the top of a function block is its execution order. For example, in figure 2, the 1.44(5) ADD_INT function with the execution number (5) is executed first and the 1.56(14) MOVE function is executed last.

An FBD network includes functions and function blocks. A function block has defined set of variables for internal storage and temporary data as well as input and output variables, while a function has no internal variables[5]. In figure 1, arithmetic, bitwise Boolean, selection and comparison groups are function groups and timer group is a function block group.

2.2. FBD Testing

In this paper, we focus on unit testing of FBD. A unit of FBD is defined as a meaningful set of function blocks used to compute a primary output[8]. The primary output is stored in the memory of the PLC for external output or internal use in other units. Figure 2 is considered as a unit FBD because they perform the computation of an external primary output *th_Prev_X_Trip*.

In [4], FBD testing was performed by transforming a unit FBD into a flowgraph. Once a flowgraph is generated from a unit FBD, existing control and data flow testing techniques can be applied. All functions of arithmetic, bitwise Boolean, selection and comparison groups are transformed into one out of 3 types of flowgraph segments. Figure 3 shows flowgraph segment templates proposed in [4]. Figure 3(a) is a representative example of functions in arithmetic, bitwise Boolean or comparison groups. They are transformed into single nodes in the flowgraph. Selection group has SEL, MAX, MIN, LIMIT and MUX functions. Figure 3(b) and 3(c) show the templates for SEL and MUX functions. Other functions in selection group - MAX, MIN and LIMIT - are transformed into single nodes.

Transformation from a unit FBD to a flowgraph is the most fundamental and important process in FBD testing. The following process describes the transformation from a unit FBD to a flowgraph briefly.

1. Create the first node with the content that read

0-7695-2685-3/06 $20.00

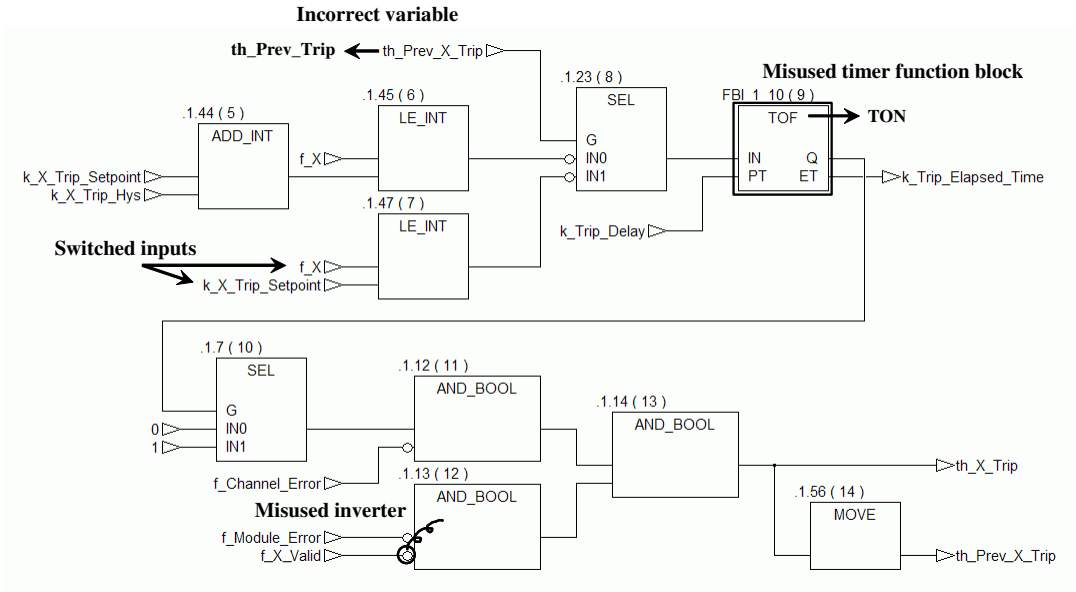
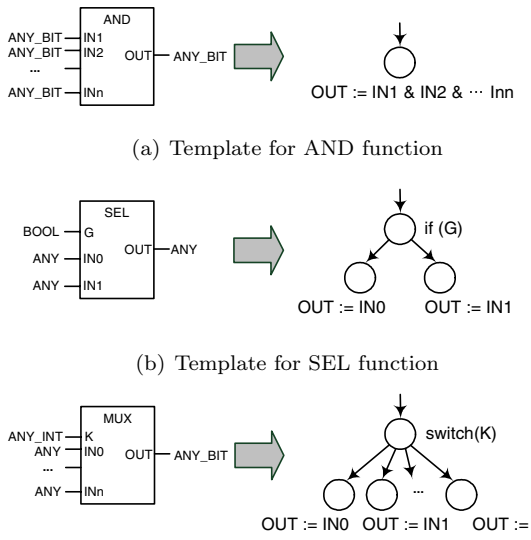


Figure 2. FBD for *th_Prev_X_Trip*



(a) Template for AND function

(b) Template for SEL function

(c) Template for MUX function

Figure 3. Templates for functions

all variables of the unit FBD.

2. Transform a function or function block into a flowgraph segment based on the corresponding template in the execution order.

   - If the output of a function or function block is not specified, create temporary output variable named as 'v[execution number]' such as *v2* and *v5*.
   - Whenever a node is added into the flowgraph segment, specify contents for the node.

3. Attach the generated flowgraph segment into the whole flowgraph.

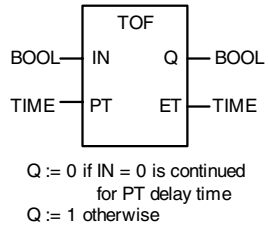
4. If the untransformed function block exists, repeat 2 for the function or function block with the next execution number, else finish.

However, in [4], timer function blocks were not considered. Testing of timer function blocks is not as simple as testing of functions. Correctness of the FBD networks containing timer function blocks cannot be tested in one cycle. In order to test timer function blocks, test cases must specify inputs covering multiple scan cycles and expected intermediate outputs at each scan cycle. Considering these issues, we will propose the flowgraph generation templates for the timer function blocks in the following sections.

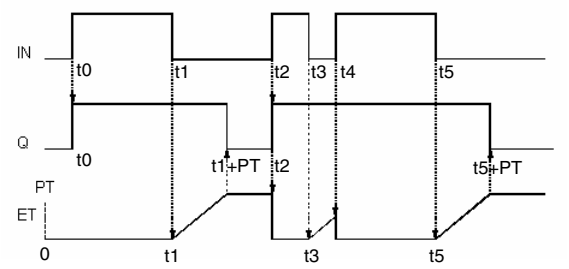
3. Flowgraph Generation for Timer Function Block

3.1. Timer Function Blocks

According to the international standard IEC 61131-3[3], timer group includes function blocks such as TOF,



(a) TOF function block



(b) TOF timing diagram

Figure 4. TOF function block and its behavioral definition

TON and TP. Figure 4(a) represents TOF(Off Delay) function block. TOF has two input variables, *IN* and *PT*, and two output variables, *Q* and *ET*. *IN* is an input Boolean variable and *PT* is a variable specifying delay time. *Q* is an output Boolean variable and *ET* represents elapsed time of the internal timer. TOF function block outputs *Q* as 0 when input *IN* is kept as 0 during the delay time specified by variable *PT* since input *IN* changed from 1 to 0. Otherwise, the output *Q* is 1.

The timers TOF, TON and TP are specified using timing diagrams. Figure 4(b) is a timing diagram of TOF describing its behavioral definition. The diagrams show the behavior of outputs *Q* and *ET* depending on input *IN*. The time axis runs from left to right and is labelled 't'. The Boolean variables *IN* and *Q* change between 0 and 1 and the time value *ET* increases as shown.

TON and TP are specified with timing diagrams similarly. Because template generation processes for all timer function blocks are similar, we therefore only describe template generation process of TOF function block. Templates for other timer function blocks can be generated in the similar way.

3.2. Flowgraph Generation Template for TOF

To transform TOF into a flowgraph segment, we represent the behavior of TOF as the condition and action table. Because TOF is a function block, it has internal state which is described by combination of values of internal variables. The condition is composed of input and internal variables, and the action is an assignment to output and internal variables. We can identify the internal variables of TOF as *preIN* and *inT* from the timing diagram definition of TOF. The *preIN* represents the value of *IN* in the previous scan cycle and the *inT* represents internal timer. To describe the behavior of TOF completely as conditions and actions, we need to specify all possible cases of related variables.

Variables affecting the condition are *preIN*, *IN*, and *inT*. The *preIN* and *IN* are Boolean. The *inT* variable can have infinite number of values. However, we don't need to deal infinite number of values because the domain of *inT* can be divided into 3 equivalent classes - [0,0], (0,*PT*), [*PT*,∞] - in the aspect of evaluation result of the condition. As a result, there are 12 different evaluations for combination of three variables. Table 1 represents actions of TOF for 12 possible conditions. Each row represents that when the evaluation condition for the *preIN*, *IN* and *inT* is satisfied, the corresponding action for *Q* and *inT* occurs. The rightmost column of table 1 specifies the corresponding cases in figure 4(b).

In table 1, we can see four nonexistent cases - b8, b9, b11, and b12. These cases never occur because *inT* is always 0 when *preIN* is 1. Table 2 is a reduced version of table 1: conditions for the same action are logically combined. We define that two actions of a TOF are identical if the value of *Q* and the action of *inT* are identical. We classified the actions of *inT*, internal timer, into 5 different cases: 'remains stopped', 'continues increasing', 'stops and remains', 'stops and is reset' and 'is reset and starts'. Because the actions of b4, b5 and b6 cases in table 1 are identical, these three cases are logically combined into '*preIN* = 0 and *IN* = 1', r4 case in table 2. By combining conditions for the same action, the behavior of TOF can be described by 7 distinct cases represented in table 2.

With this result, we make a template to transform the TOF function block into a flowgraph segment. Figure 5 is a resulting template for TOF. The rightmost column in table 2 specifies the corresponding nodes of figure 5 for each case. The first and second number in the rightmost column of table 2 are node numbers representing the condition and action, respectively.

Templates for TON and TP of timer group can be

0-7695-2685-3/06 $20.00 © 2006 IEEE

Table 1. Condition and action table describing the behavior of TOF

Cases	Condition			Action		Cases in fig.4(b)
	preIN	*IN*	*inT*	*Q*	*inT*	
b1	0	0	0	0	remains stopped	[0,t0)
b2	0	0	$0<inT<PT$	1	continues increasing	(t1,t1+PT, (t3,t4), (t5,t5+PT)
b3	0	0	$inT>= PT$	0	stops and remains	[t1+PT,t2), [t5+PT,-)
b4	0	1	0	1	stops and is reset	t0
b5	0	1	$0< inT<PT$	1	stops and is reset	t4
b6	0	1	$inT>=PT$	1	stops and is reset	t2
b7	1	0	0	1	is reset and starts	t1,t3, t5
b8	1	0	$0< inT<PT$	-	nonexistent case	-
b9	1	0	$inT>=PT$	-	nonexistent case	-
b10	1	1	0	1	remains stopped	(t0,t1),(t2,t3), (t4,t5)
b11	1	1	$0<inT<PT$	-	nonexistent case	-
b12	1	1	$inT>=PT$	-	nonexistent case	-

Table 2. Reduced condition and action table describing the behavior of TOF

Cases	Condition			Action		Cases in table 1	Nodes in fig.5
	preIN	*IN*	*inT*	*Q*	*inT*		
r1	0	0	0	0	remains stopped	b1	10,12
r2	0	0	$0<inT<PT$	1	continues increasing	b2	1,3
r3	0	0	$inT>=PT$	0	stops and remains	b3	2,5
r4	0	1	-	1	stops and is reset	b4,b5, b6	4,7
r5	1	0	0	1	is reset and starts	b7	6,9
r6	1	1	0	1	remains stopped	b10	8,11
r7	1	-	$0<inT$	-	nonexistent case	b8,b9, b11,b12	-

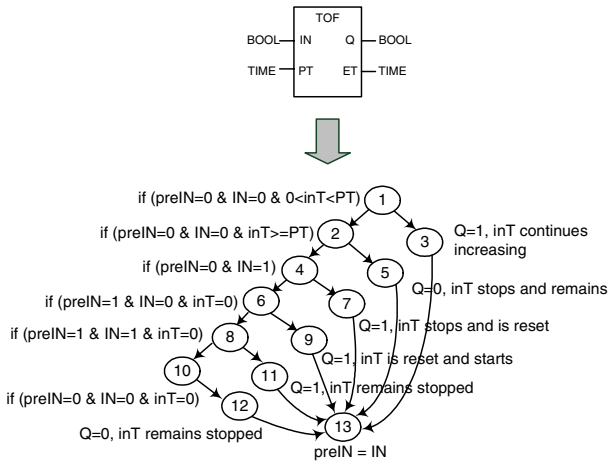


Figure 5. Template for the TOF function block

generated similarly. Moreover, this template generation process also can be applied to other kinds of function blocks besides timer group.

Figure 6 is a flowgraph transformed from the unit FBD with seeded errors in figure 2. After creating the first node 0 which reads all variables used in the FBD except temporary output variables, each function or function block was transformed into the corresponding flowgraph segment according to its execution order. The generated flowgraph segment was attached to the whole flowgraph.

4. FBD Unit Testing

4.1. Timer Function Block Testing

After transforming of a unit FBD into a flowgraph, we select proper test coverage criteria and generate the satisfying set of test cases. When a unit FBD includes only functions, one scan cycle testing is sufficient. On the other hand, if a unit FBD includes timer function blocks, one cycle testing is insufficient because timer

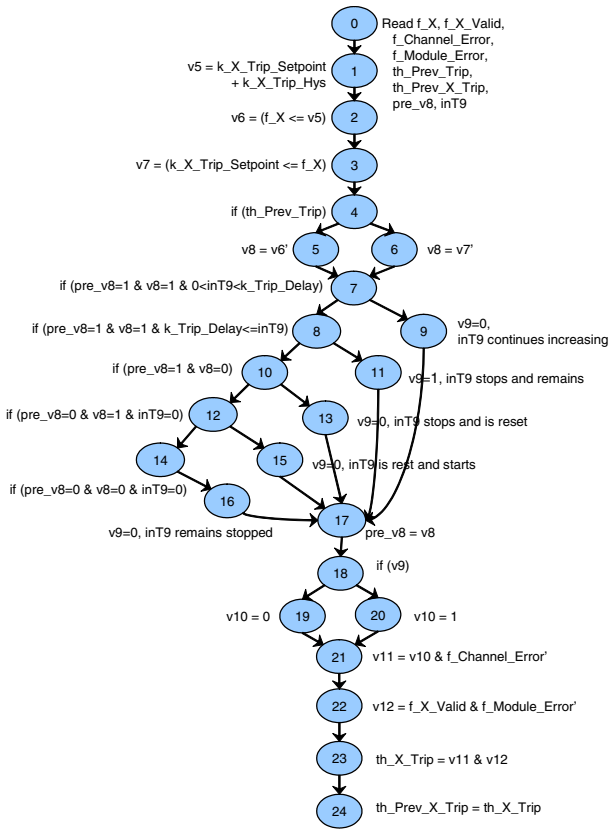


Figure 6. Flowgraph generated from the FBD unit for *th_Prev_X_Trip*

function blocks have internal state, i.e., generate different outputs for the same inputs according to its internal state. In order to test FBD networks including timer function blocks sufficiently, it is desired to generate test cases to cover combination of input variables and internal states of timer function blocks as much as possible.

We can generate a set of test cases which covers every internal states of timer function blocks at least once by applying All-Edges coverage criteria to the proposed flowgraph. With stronger coverage criteria than All-Edges, we can generate more sophisticated set of test cases.

The most distinct feature of FBD testing with timer function blocks is that we should specify a precondition for each test case. Precondition is combination of evaluations of internal variables of timer function blocks. For the test case with a precondition, test stubs that lead to the state in which the precondition is satisfied are required.

4.2. Control Flow Testing

Control flow testing coverage criteria include All-Nodes, All-Edges and All-Paths. Table 3 shows a set of test cases that satisfies the All-Edges test coverage criteria in the flowgraph of figure 6. All-Edges coverage criterion requires that all edges in flowgraph should be executed at least once. *Prev_Trip*, *Prev_X_Trip*, *X*, *Ch_Err*, *Md_Err* and *Valid* in table 3 represent *th_Prev_Trip*, *th_Prev_X_Trip*, *f_X*, *f_Channel_Error*, *f_Module_Error* and *f_X_Valid* variables respectively. The last two columns represent the actual output and the expected output for the variable *th_Prev_X_Trip*. Test case CT1 means that if inputs are 1, 0, 100, 0, 0 and 1 for *th_Prev_Trip*, *th_Prev_X_Trip*, *f_X*, *f_Channel_Error*, *f_Module_Error* and *f_X_Valid* variables, respectively when *pre_v8* and *inT9* timer is 0, the expected output *th_Prev_X_Trip* is 1 and the actual output is 0. Because actual output, 0, is different from expected output, 1, we can find an error from this test case. We can see that preconditions are specified for each test case because the unit FBD has a timer function block.

4.3. Data Flow Testing

In order to apply data flow testing strategy to the flowgraph, we identify du-paths for each variable after identifying the definition and usage nodes for all the variables. Then, we apply data flow testing coverage criteria such as All-Defs or All-Uses to the flowgraph. We should consider the following points to reflect characteristics of FBD in FBD data flow testing.

First, extra variables should be handled in the same way as original input and output variables. There are two types of variables in the flowgraph generated from a unit FBD. Original variables are input and output variables which appear in the source FBD and extra variables are newly generated ones during the transformation process such as *v5* and *inT9*. We need to extract du-path information for extra variables as well as original variables. For example, in figure 6, we identified du-path information for temporary output variables such as *v5* – *v12* and internal variables of the timer function block such as *pre_v8* and *inT9* as well as original input variables such as *f_X* and *f_Channel_Error*.

Second, the first node of the flowgraph should be a definition node for all variables except temporary output variables. Since a unit FBD is a part of whole FBD program, definitions of some variables of current testing unit may be in another unit. In this case, data anomaly can occur in the form of undefined use. This

0-7695-2685-3/06 $20.00

Table 3. Test cases satisfying All-Edges test coverage criterion

Test	Precondition		Inputs						*th_Prev_X_Trip*	
Cases	*pre _v8*	*inT9*	*Prev_Trip*	*Prev_X_Trip*	*X*	*Ch_Err*	*Md_Err*	*Valid*	Actual	Expected
CT1	0	0	1	0	100	0	0	1	0	1
CT2	0	0	0	0	100	0	0	1	0	1
CT3	1	50	0	0	100	0	0	1	0	1
CT4	1	100	0	0	100	0	0	0	0	1
CT5	1	100	1	1	80	0	0	1	0	1

data anomaly leads to fictitious errors. To prevent this, we create the first node of the flowgraph with the content of reading all variables of the unit FBD. Only temporary output variables are exempts because they always have their definition nodes in the generated flowgraph. We can see that all variables of the unit FBD except temporary output variables are read in the first node 0 of the flowgraph in figure 6.

Third, sequences of test cases are important. Table 4 represents du-paths for variables in figure 6 and table 5 shows test cases satisfying All-Uses test coverage criteria. A set of test cases in table 5 was generated based on the identification of du-paths in table 4. We can observe the characteristics of FBD data flow testing in the table 4. The du-path no.3 represents that the variable *f_X* is defined at node 0 and used at node 3. This du-path is covered by a single test case executed within a single scan cycle. On the other side, the du-path no.25 represents that a definition node of the variable *inT9* is node 9 and its use node is node 7 of the next cycle. In this case, the du-path is not covered by a single test case. In order to cover this kind of du-path, a sequence of two test cases is required. For example, du-path no.20 is covered by the sequence of test cases DT1 and DT2 and du-path no.25 is covered by the sequence of DT6 and DT7.

4.4. Case Study

We applied the proposed approach to the BP trip logic of RPS in digital plant protection system, which is being developed at KNICS. This section explains how we could find various errors in a unit FBD using the proposed FBD testing method. We seeded four different errors into the unit FBD in figure 2. All seeded errors are frequently occurred ones in FBD programming. More errors occurring in FBD programming are explained and classified in [7]. The seeded errors were all found by a set of test cases satisfying All-Edges coverage criteria in table 3. A set of test cases satisfying All-Uses coverage criteria in table 5 could also find all seeded errors.

Table 4. Du-paths for variables of the flowgraph in figure 6

Du-Path No.	Variable	Def node	Use node
...	...	...	...
3	*f_X*	0	3
...	...	...	...
19	*v8*	6	14
20	*pre_v8*	17	next 7
...	...	...	...
25	*inT9*	9	next 7
...	...	...	...

- *Case 1 (Misused timer function block)*: FBD programmers often make a mistake in use of timer function blocks because TOF and TON have similar functionality. We inserted TON instead of (9)TOF. This error was found by the CT2 and CT3 test cases of the table 3 in control flow testing and by the DT3, DT4, DT6 and DT7 test cases of the table 5 in data flow testing.
- *Case 2 (Switched inputs)*: One of the frequently occurring errors in FBD programming is switched inputs. While the change of the input order in AND_BOOL function is not a problem, switched inputs in SEL, MUX, or LE functions can cause errors. We reversed inputs of the (7)LE_INT function. This error was found by the CT5 test case in control flow testing and by the DT10 test case in data flow testing.
- *Case 3 (Misused inverter)*: The inverter, drawn by small circle, is often added in unnecessary position or omitted in necessary position. We omitted a necessary inverter of IN1 input in the (12)AND function. In the control flow testing, this error was found by the CT4 test case and it was found by the DT8 test case in data flow testing.

Table 5. Test cases satisfying All-Uses test coverage criteria

Test	Precondition		Inputs						*th_Prev_X_Trip*	
Cases	*pre _v8*	*inT9*	*Prev_Trip*	*Prev_X_Trip*	*X*	*Ch_Err*	*Md_Err*	*Valid*	Actual	Expected
DT1	0	0	0	0	91	0	0	1	0	0
DT2	0	0	1	0	100	0	0	1	0	1
DT3	0	0	0	0	100	0	0	1	0	1
DT4	1	50	0	0	80	0	0	1	0	1
DT5	0	0	0	0	80	1	1	1	0	0
DT6	1	50	0	0	100	0	0	1	0	1
DT7	1	100	0	0	80	0	0	1	0	1
DT8	1	100	0	0	100	0	0	0	0	1
DT9	1	100	0	0	100	0	0	1	1	1
DT10	0	100	1	1	100	0	0	1	0	1

- *Case 4 (Incorrect variable)*: Variable names are often written incorrectly. Incorrect variable names result in wrong value assignments or computations. We wrote an input of (8)SEL as *th_Prev_Trip* instead of *th_Prev_X_Trip*. This error was found by the CT1 test case in control flow testing and by the DT2 test case in data flow testing.

5. Conclusion

We proposed a structural testing technique on the FBD networks including timer function blocks. We transformed a unit FBD including timer function blocks into a flowgraph based on several templates and applied existing structural testing techniques for the generated flowgraph. We presented how to generate transformation templates for the timer function blocks and described how the characteristics of timer function blocks are reflected in control and data flow testing.

To demonstrate the effectiveness of the proposed method, we use a trip logic of BP in RPS which is being developed at KNICS in Korea as a case study. We seeded frequently occurring errors into the unit FBD and could find all the seeded errors by executing the test cases generated by the proposed approach.

By the proposed method, systematic structural testing for the FBD including timer function blocks became possible while there was no structural testing method for them before. This approach also has an advantage that it can be applied to any FBD, whatever its intermediate format is.

We have a plan to support FBD testing automation. Integration testing of FBD which focuses on interfaces and interactions between tested units should also be considered.

Acknowledgments

This work was partially supported by the Korea Science and Engineering Foundation(KOSEF) through the Advanced Information Technology Research Center(AITrc) and also partially supported by the Information Technology Research Center(ITRC), Software Process Improvement Center(SPIC) and Internet Intrusion Response Technology Research Center(IIRTRC).

References

[1] http://www.framatome-anp.com.

[2] KNICS(Korea Nuclear Instrumentation and Control System research and development center), http://www.knics.re.kr/english/eindex.html.

[3] IEC. *International Standard for Programmable Controllers: Programming Languages (Part 3)*, 1993.

[4] E. Jee, J. Yoo, and S. Cha. Control and data flow testing on function block diagrams. In *proceedings of the 24th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2005), LNCS 3688*, pages 67–80, September 2005.

[5] R. Lewis. *Programming industrial control systems using IEC 1131-3 Revised Edition(IEE Control Engineering Series)*. The Institute of Electrical Engineers, 1998.

[6] A. Mader. A classification of plc models and applications. In *proceedings of WODES 2000: 5th Workshop on Discrete Event Systems*, August 2000.

[7] Y. Oh, J. Yoo, S. Cha, and H. Son. Software safety analysis of function block diagrams using fault trees. *Reliability Engineering and System Safety*, 88(3):215–228, 2005.

[8] J. Yoo, S. Park, H. Bang, T. Kim, and S. Cha. Direct control flow testing on function block diagrams. In *proceedings of the 6th International Topical Meeting on Nuclear Reactor Thermal Hydraulics, Operations and Safety(NUTHOS-6)*, October 2004.