Development of Highly Reliable Safety System with Enhanced Tolerance against Soft-Error

HIGUCHI Toshiyuki

We report on the development of a highly reliable safety system that ensures that the equipment continues to operate while maintaining safety functions without making emergency stops due to system errors caused by soft errors. As semiconductor devices become more highly integrated and miniaturized, transient bit errors (soft errors) in memory are a problem. If the data stored in semiconductor memory is temporarily modified due to a soft error, a short time breakdown or system downtime may occur. In equipment that operates 24 hours a day and handles expensive materials in a semiconductor manufacturing factory, temporary stoppage can cause excessive profit loss.

On the other hand, Safety PLCs (Programmable logic controller), which are often used in semiconductor manufacturing equipment, perform self-diagnosis on all semiconductor integrated circuits and memory circuits related to safety control, and immediately stop the equipment when an abnormal operation is detected. Even if data is garbled due to a soft error, the safety PLC will stop the equipment. Therefore, there is a need for measures to maintain productivity while maintaining safety functions and suppressing unnecessary outages. In response to this issue, we have realized the technique with not only the function of detecting random hardware failures that impair safety and maintaining the safety state but also with the function of detecting data corruption due to soft errors and recovering data. This paper describes the specific measures for the function of recovering data by detecting data corruption due to the soft error, and the verification results of the effect.

1. Introduction

Various types of safety PLCs (programmable logic controllers) have been adopted for establishing safety systems aimed at the protection of human bodies from the machinery and equipment in the factories. The safety PLC is the control unit for safety control that has acquired a certification in accordance with international safety standards as represented by IEC 61508[1]. Since safety PLCs are used to ensure the safety of workers, they must not operate in such a way that a dangerous condition of the machine or equipment cannot be detected because of a failure of the safety PLC itself and judging it as safe and allowing the machine to operate. For this reason, safety PLCs are equipped with redundancy and diversity in hardware and software, always self-diagnosing the safety control-related components and implemented with the ability to stop machine and equipment on the safe side if a random hardware failure that impairs safety is detected. Those have greatly enhanced the safety and reliability compared to the ordinary PLCs.

Contact : *HIGUCHI Toshiyuki* toshiyuki.higuchi@omron.com

Safety PLCs make it possible to realize more flexible safety systems in large and sophisticated applications through the adoption of software safety circuits and reduced wiring by safety networks. For this reason, semiconductor devices that can process large capacity programs at high speed are implemented in a safety PLC. The safety PLC performs a self-diagnosis on all safety control-related semiconductor integrated circuits and memory circuits and controls the equipment to perform an immediate stop if an abnormal operation is detected.

With the increasing high integration and miniaturization of semiconductor devices in recent years, transient bit errors (soft errors) in memory have been spotlighted. Soft errors occur, for example, by α particles, cosmic rays, or neutron collisions. In addition, data corruption induced by fine foreign objects and the corruption induced by noise that is sporadically generated by input/output control devices with noise from the outside have become problems as well. These soft errors can cause a temporary system malfunction or system downtime by temporarily rewriting the data stored in semiconductor memory. In facilities that operate 24 hours a day and handle expensive

materials, such as semiconductor manufacturing plants, even such a temporary system shutdown can cause an excessive loss of profit; consequently, a countermeasure is in demand.

It is generally known that the diagnostics for detecting bit flips caused by a soft error and the repair of data with bit flips can be realized by using Error Correction Codes (hereinafter referred to as "ECC"). However, in order to apply such a countermeasure using ECC functions to a safety PLC, it will lead to higher cost because it is necessary to change to memory or the MPU with a special built-in hardware to generate and check ECC. In addition, because the device is changed, the failure analysis and safety evaluation of the safety PLC must be conducted again, and separate hardware development and evaluation per model will be necessary.

Hence, we worked on the development of a safety PLC seeking enhanced soft-error tolerance through software countermeasures only. Implementing of a software countermeasure is advantageous with no increase in cost because they do not require additional hardware circuits, and ease of spreading for existing safety PLCs and other safety components.

2. Challenges in Software Countermeasures

Many existing safety components, including safety PLCs, have made the MPUs redundant. As a measure for detecting the data corruption, we adopt a method to check data between MPUs as shown in Fig. 1. But, the current situation is, while an abnormal state of the MPU, including data corruption, is detected through the comparison of data, the repair of data corruption is not performed yet.

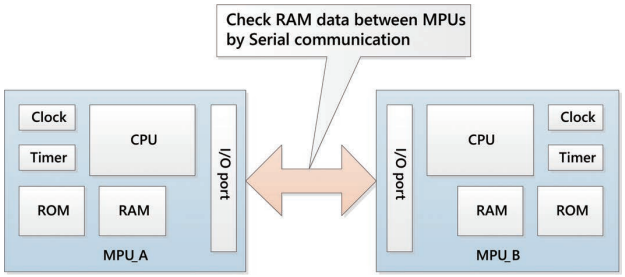


Fig. 1 Data Check between MPUs

Hence, a conventional technology is suggested as an example to have triple replicated variable data in each MPU and check them. Checking with triple replicated data is expected to be highly effective as a means of detecting data corruption and further repairing the data. However, checking the variable data by triple replication requires the variable data to be defined in the program in advance, and it implies the presence of risk where the only temporarily used data such as those in the stack area cannot be protected. In addition, since the checking with triple replication is performed when reading/writing the variable data, the interval of checking will be longer for the infrequently accessed variable data, as a result, there is a possibility that data corruption may occur on multiple variable datasets and are not repaired.

3. Countermeasure

For handling the aforementioned issues, the following three software countermeasures were investigated to realize a function to detect data corruption in each MPU and repair it.

(1) Triple replication of variable data
(2) Protection of stack area
(3) Prevention of error accumulation by cyclic testing

3.1 Triple Replication of Variable Data

In the program source code, variables are defined as the storage areas for the reading/writing of data handled in memory. The C language is known as one of the typical programming languages; we used the C++ language for this development. For Static and Auto variables that are declared as variables in the C++ language, we decided to declare and treat them as redundant variables in the program. Then, a majority decision was made before a relevant variable was used in the arithmetic process. This provided the detection of data corruption in variables and the repairing of data. The details of data triple replication process are shown in Fig. 2.

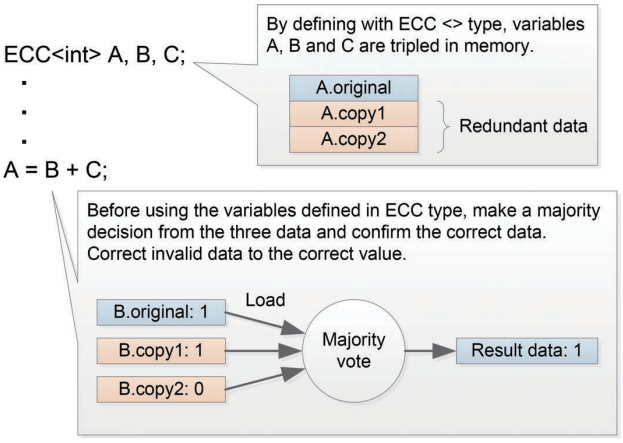


Fig. 2 Variable Data Replication Processing into Three Sets

The match of none of the three datasets is regarded as an error. Fig. 3 shows the details of the majority voting process.

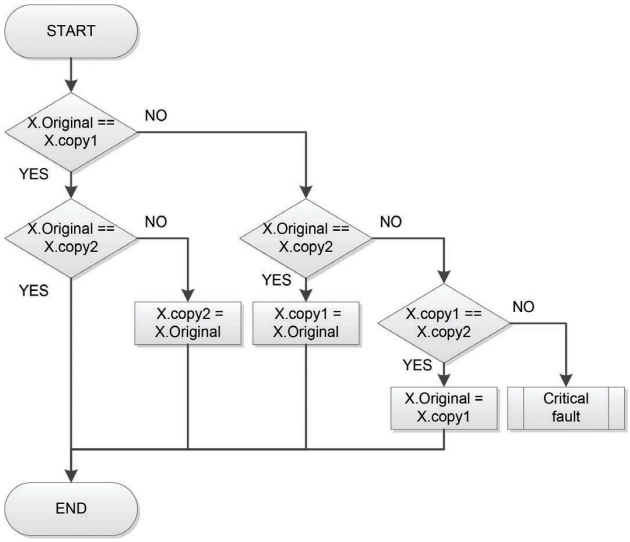


Fig. 3 Triple Replication Process for Variables

3.2 Protection of Stack Area

A processor that uses memory to execute arithmetic processing has built-in registers. Registers are used for calculations and for the use of pointing to a specific address in memory. The stack area is provided in memory to temporarily store the register data. When a function process (subroutine) occurs during the execution of the main routine of a program, a stack operation is performed to interrupt the main routine. There are two stack operation processes as follows:

(1) The operation to temporarily evacuate the data stored in the processor's built-in registers to the stack area of RAM (random access memory) at the start of subroutine processing (push operation).
(2) Operation to return the data evacuated in the stack area to the register at the end of subroutine processing (pop operation).

Soft errors can occur in the stack area where data is temporarily evacuated by the stack operation as well. The codes for stack operations associated with the subroutine processing are automatically generated by the compiler. For this reason, when a general-purpose compiler is used, the variable data triple replication process described in paragraph 3.1 is not applicable to the stack operation.

For this reason, it was decided to make the data redundant by creating a replication of data saved in the stack area in RAM in the beginning of the subroutine process and to check the redundant data in the end of the subroutine process.

The details of the triple replication process of the stack area are shown in Figs. 4 and 5. The majority voting process to check the triple replicated data is the same as the method shown in Fig. 4.

– In the functionEntry process of funcY, the stack pointer before the call out of funcY (=prevSP) and the stack pointer after the call out of funcY (=nowSP) are compared, and a copy of the increased data is created in RAM.
– Immediately before terminating the execution of funcY, a majority decision is carried out using the data increased by the functionExit process and the copied data.

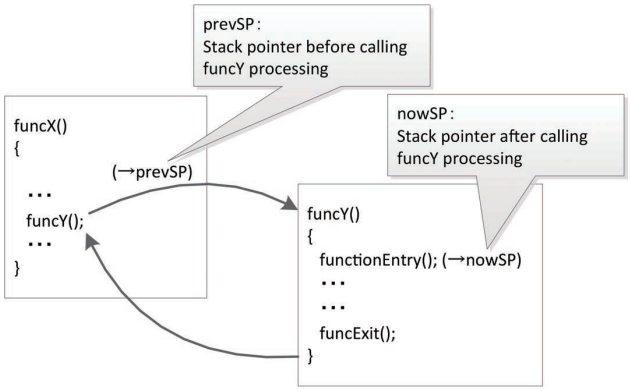


Fig. 4 Reading Out the Stack Pointer

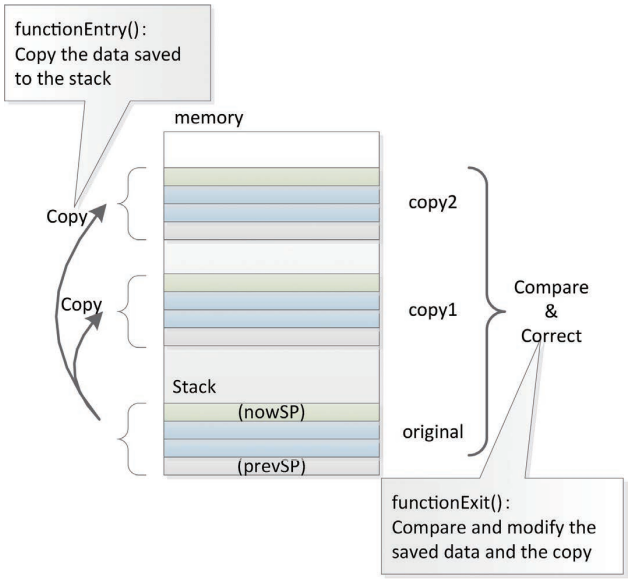


Fig. 5 Triple Replication of Stack

3.3 Prevention of Error Accumulation by Cyclic Testing

In the triple replication process of variable data described in paragraph 3.1, the data stored at the address corresponding to the readout are repaired when a variable is read out by the program execution. However, the frequency of the readout of a variable depends on the program. For variables that are read out more frequently, there are more opportunities for the data to be

repaired. However, for variables that are infrequently read out, there are fewer opportunities for the data to be repaired, and the accumulation of errors can cause a soft error on multiple addresses among the addresses of the triple replicated variables in RAM.

Hence, apart from the execution of tasks, such as the arithmetic processes carried out during the processing of the main routine or subroutine, we decided to establish a dummy processing section and execute a dummy process to read out the variable at every predetermined cycle.

Fig. 6 shows the task execution section, dummy processing section, and read/write processing section. The first through third addresses of the read/write processing section indicate the main memory addresses; the first address indicates the original variable data, and the second and third addresses indicate the copied variable data.

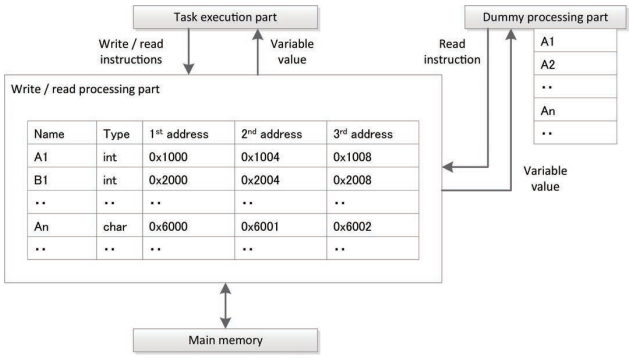


Fig. 6 Task Execution Section, Dummy Processing Section, Read/Write Processing Section

Fig. 7 shows an example of repaired data corruption by the dummy processing section.

In spite of the low occurrence frequency of tasks that use variable A1, the dummy processes are executed at every regular cycle. Consequently, even if data corruption occurs before the next task that uses variable A1, the data will be repaired, and the accumulation of errors is prevented.

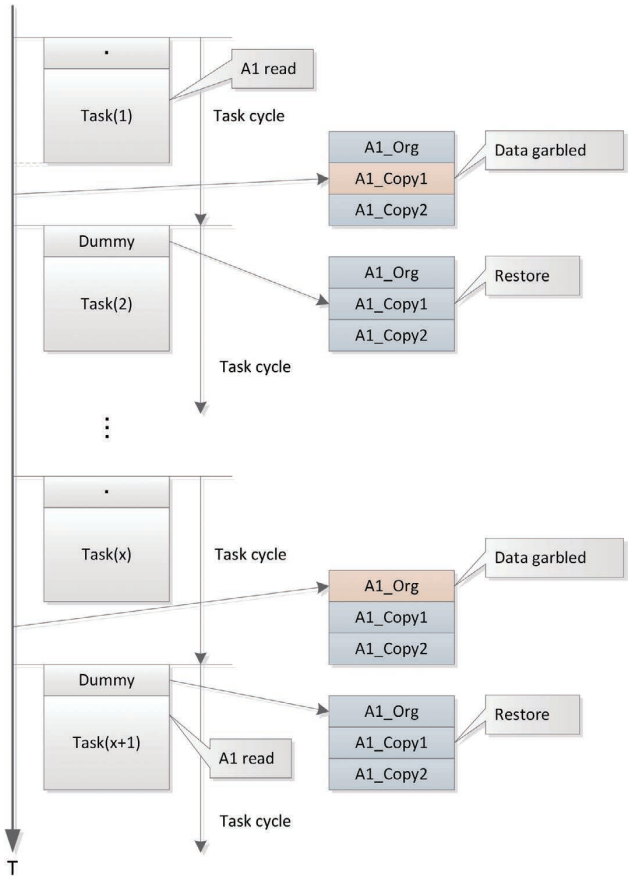


Fig. 7 Repaired data corruption by the dummy processing section

4. Validation

In order to validate the effectiveness of the aforementioned three measures to prevent data corruption-triple replication of variable data, protection of stack area, and prevention of error accumulation by cyclic testing, it is necessary to generate soft errors and measure the degree of repair of memory in which the data corruption occurred and the occurrence and frequency of system abnormalities. Hence, the following two methods were adopted for validation. These allowed us to accelerate the soft error evaluation in a short period of time:

- Evaluation by pseudo soft error processing
- Evaluation by α-ray irradiation test

4.1 Evaluation Through Pseudo Soft Error Processing

A pseudo-soft error processing evaluation is performed by implementing a software batch process that generates pseudo-bit corruption in a program. The bit corruptions were exhaustively generated over the entire RAM area, and the evaluation method and configuration were studied in such a way that the location where the bit corruption that occurred could be identified when a system error occurred.

Fig. 8 shows the evaluation configuration using the pseudo soft error processing. Using the following procedure, the location information of the bit corruption and the abnormality information are mutually indexed and stored in the database:

(1) Using a personal computer, specify the position of a single bit where the bit corruption is generated in main memory of the safety PLC.
(2) Invert a single data bit at a specified position in main memory by software batch processing in a safety PLC. Collect abnormality information regarding an abnormality that occurred within a specified time after reversing a single data bit.
(3) The personal computer acquires the abnormality information from the safety PLC.
(4) The personal computer registers the location information on the location specified in (1) and the abnormality information acquired in (3) and mutually indexed in the database.

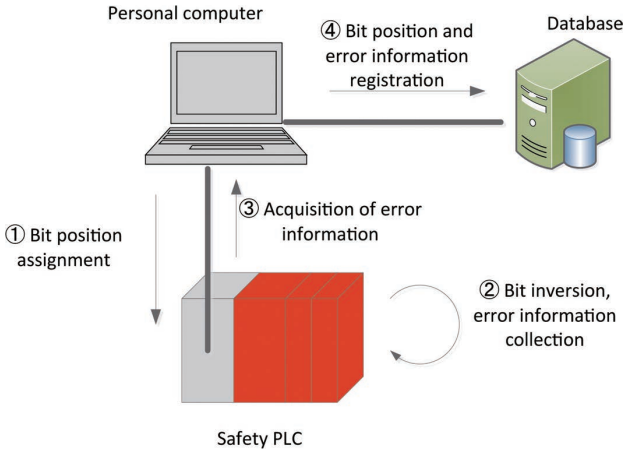


Fig. 8 Evaluation Configuration for Pseudo Soft Error Processing

Table 1 shows an example list of location information and abnormality information registered in the database. By referring to the database, it is easy to analyze the cause when a system abnormality occurs and to check the impact of the occurrence of a software error.

Table 1 Location Information and Abnormality Information Database

No.	Address	Bit Position	Variable Name	Abnormality ID	Source Code (File Name, Line number)
1	0x200010AC	1	val_A	30	file_A.c, 867
2	0x20006000	7	val_B	57	file_B.c, 620
3	0x20007032	3	val_C	42	file_C.c, 500
4	0x20009A00	4	stuck	30	file_B.c, 827
⋮	⋮	⋮	⋮	⋮	
N	0x20002000	5	val_X	77	file_A.c, 712

Table 2 shows the evaluation results of the pseudo soft error process. Using identical hardware, a running test was conducted by implementing a software batch process that generates pseudo bit corruption in both the software before and after implementing the software error countermeasure. As a result, the system abnormality occurrence rate ratio of the unit implemented with the soft error countermeasure to the unit before the countermeasure was 0.00082.

Table 2 System Abnormality Frequency

	Unit Before Countermeasure	Unit After Soft Error Countermeasure
a. Bit corruption frequency	77,465 times	289,636 times
b. System abnormality frequency	1,630 times	5 times
c. System abnormality ratio (b/a)	0.021041	0.000017
d. System abnormality rate before/ after countermeasure	1	0.00082

4.2 Evaluation by α-Ray Irradiation Test

JEDEC JESD89[2] specifies the following three soft error evaluation tests.

JESD89-1: Field Test (Test Method for Real-Time Soft Error Rate)
JESD89-2: Alpha (α) Ray Irradiation Test Using Radioactive Substances (Test Method for Alpha Source Accelerated Soft Error Rate)
JESD89-3: Neutron Irradiation Test Using Accelerator (Test Method for Beam Accelerated Soft Error Rate)

Field tests are costly and time consuming because they require a large number of evaluation samples over a long period of time. Neutron irradiation tests using accelerators are not easy to perform because of the limited number of facilities that can facilitate neutron irradiation. On the other hand, α-ray irradiation tests using radioactive substances are conducted in a short time if a source is available. This time, in order to confirm the effect of software countermeasures at an early stage by actually generating software errors, we referred to the α-ray irradiation test, which provided a simple acceleration test.

Figs. 9 and 10 show the test configuration. An MPU implemented with memory was directly irradiated by an α-ray source, and the behavior of the product was observed.

– With an MPU mounted in a product, open the MPU package surface and expose the IC chip inside.
– Place the ^{241}Am (americium), an α-ray source, on the top surface of an IC chip.
– Operate the product and observe the behavior. Measure the

frequency and time of occurrence of any system abnormality.

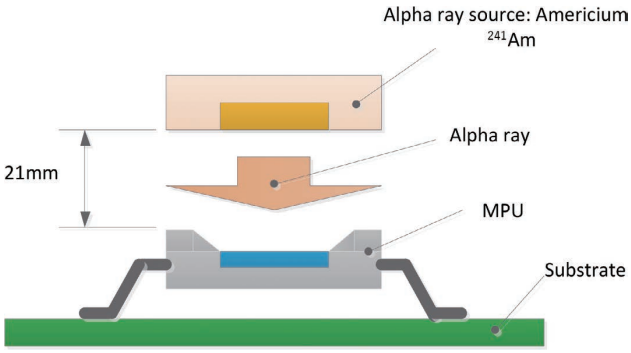
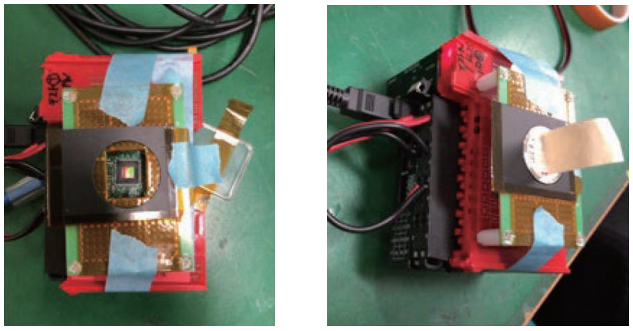


Fig. 9 α-Ray Irradiation Test Configuration



Before installation of the α-ray After installation of the α-ray

Fig. 10 Configuration of Pseudo Soft Error Processing Evaluation

Table 3 shows the α-ray irradiation test results for an existing unit and a unit implemented with the soft error countermeasure. As a result, the system abnormality occurrence rate ratio of the unit implemented with the soft error countermeasure with regard to the existing unit was 0.0012.

Table 3 System Abnormality Frequency (Distance from α-ray source: 21 mm)

	Existing Unit	Unit After Software Countermeasure
a. Cumulative irradiation time	76 min	13,000 min
b. System abnormality frequency	5 times	1 time
c. System abnormality interval (b/a)	15.2 min	13,000 min
d. System abnormality ratio before/after countermeasure	1	0.0012

For more information, the system error rate ratios are different between the evaluation by pseudo soft error processing and the evaluation by the α-ray irradiation test. The possible reason for this is the inability to carry out the comparison tests for the units used for a test on an identical hardware because the surface seal of the MPU package of the unit implemented with the software is opened for testing. In addition, since the flow rate (number of α-particles) of α-rays emitted from the americium varies depending on the distance between the americium and the MPU, the α-ray flow rate depends on the precision of the jig that secures the α-ray source. Since the test was conducted with a simple attachment method this time, it is possible that the comparison was not conducted with an equal α-ray flow rate. These are the future issues that need to be addressed in the evaluation of soft errors.

5. Conclusions

This time, we worked on the development of a safety PLC that enhances the resistance to soft errors by simply changing the software without any addition of hardware.

We confirmed that the three software measures-triple replication of variable data, protection of stack area, and prevention of error accumulation by cyclic testing-could, through a pseudo soft error evaluation that exhaustively generates soft errors in memory, reduce the system error occurrence rate to 0.0009 or less compared to that before the countermeasures. In addition, through α-ray irradiation tests, we could confirm the continuous operation of the system while repairing bit corruption even in an environment where actual radiation is generated. Through the use of this technology, we are expecting an improvement in the frequency of system downtime due to soft error to approximately 1/1,000.

This time, the safety PLC was targeted, but since software measures are used, it will be easy to spread and apply to other safety components. In addition to realizing the functions as a safety component, such as detecting random hardware failures that impair the safety required by the international safety standards and maintaining a safe state, by utilizing this technology, the function to detect data corruption caused by soft errors and to further repair the data will be realized with the software. And, it will be possible to build up a highly reliable safety system that will operate continuously without system abnormalities caused by bit corruption.

The future challenges are the reduction in processing time for monitoring and repairing data corruption and the high speed. This technology is expected to have a similar effect on applications dealing with the network devices that handle large capacity memory, where the impact of soft errors is a concern. We will continue the consideration simultaneously with the issues of speeding up the response time of the entire safety system, including the network. Furthermore, we would like to contribute to improved productivity through the creation of highly reliable safety systems that keep the equipment continuously operating while maintaining the safety functions.

References

1) Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, IEC 61508, 2010.
2) Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices, JEDEC JESD89, 2006.

About the Authors

HIGUCHI Toshiyuki
Development Dept., Safety Div.
Product Business Division H.Q.
Industrial Automation Company
Specialty:Safety Engineering, Software Engineering