A Multi-Level Memory Integrity Verification Method for Safety-Grade PLC CPU Modules(POSAFE-Q)

Jeong-Chan Lee [a*], Myeong-Jun Jeon[a] , Jae-won Choi[a], Tae-Kang Yeo[a] ,
Jun-Won Lim[a], Soo-Hyun Lim[a], Ji-Hyun Hwang[a]
[a]*SOOSAN ENS Co., Hyundai Venture Ville 3F 315, 10, Bamgogae-ro 1-gil, Gangnam-gu, Seoul, Korea, 06349*
[*]*Corresponding author: jungchan0308@soosan.co.kr*

Keywords : Safety Grade PLC, POSAFE-Q, Memory Integrity Verification

1. Introduction

Safety-grade Programmable Logic Controllers (PLCs) used in nuclear power plants are core control devices that perform protective logic operations and safety-related control functions. The memory inside the CPU module stores control programs and operational data, and is continuously referenced during system operation. Therefore, if a single-bit error or a specific bit fault occurs in memory, it may affect computational results.

Verification of memory integrity is a fundamental procedure for ensuring system reliability. In particular, it is important to detect memory cell stuck-at faults or abnormal data modifications. In this study, a multi-level memory integrity verification method combining static full-memory testing and runtime monitoring is proposed for a safety-grade PLC CPU module. The proposed approach provides a structured verification framework to detect both static memory faults and runtime data modifications in safety-critical PLC systems.

2. Target System and Memory Architecture

2.1 CPU Module Overview

The target of this study is the CPU module of a safety-grade PLC. The CPU module performs control logic operations and processes operational data, managing programs and runtime variables through its internal memory. The memory is repeatedly accessed during system operation, and computational results and status information are continuously updated.

The memory inside the CPU module is classified according to its function into a memory area for computation and a shared data storage area required for redundant operation. In this study, SRAM and DPRAM are selected as the verification targets.

2.2 SRAM Structure and Role

SRAM (Static Random Access Memory) is a volatile memory directly used for CPU computations. It stores the execution code of the operating software and application programs, as well as operational data and internal state variables required during program execution. Continuous read and write operations are performed while the control program is running, and the accuracy of system computation is directly related to the integrity of this memory.

If a stuck-at fault or abnormal data modification occurs in a specific bit within the SRAM area, it may distort execution behavior or computational results. Therefore, SRAM requires static integrity verification over its entire memory space.

2.3 DPRAM Structure and Role

DPRAM (Dual Port Random Access Memory) is a memory area used for storing shared data required for redundant operation within the CPU module. This area stores operational variables, status information, and shared data for maintaining consistency between redundant systems.

Since the values stored in DPRAM are referenced during redundant operation, data accuracy and retention characteristics are important. If a specific variable value is abnormally changed, data inconsistency between systems may occur. Therefore, DPRAM requires not only static full-memory testing but also runtime monitoring of data changes.

2.4 Definition of Memory Areas for Verification

In this study, the entire SRAM area used directly for CPU computation and the entire DPRAM area storing redundant shared data are defined as verification targets. In addition, runtime data change monitoring is performed mainly on the application area of DPRAM.

3. Memory Integrity Verification Architecture

The memory integrity verification structure proposed in this study consists of three levels. Each level has a different application environment and objective, and they perform complementary roles.

Level 1 is an intensive test targeting the entire memory area. Level 2 and Level 3 are monitoring structures designed to detect data changes during system operation.

Table I summarizes the configuration of the proposed verification architecture.

Table I: Memory integrity verification levels

Level	Verification Method	Objective
Level 1	Pattern-based full test	Detection of static memory cell faults
Level 2	OS-level retention test	Verification of DPRAM data retention
Level 3	Application-level monitoring	Detection of data changes during operation

Level 1 and Level 2 are performed under a dedicated test operating environment, while Level 3 operates under the normal application execution environment.

4. Pattern-Based Full Memory Test

4.1 Test Operating System Configuration

In a real-time operating system environment, intensive full-memory testing may be limited due to the influence of periodic tasks and interrupts. Therefore, a dedicated test operating system was separately configured for the purpose of memory verification only. The test operating system was implemented with a simplified structure that performs only memory access and comparison functions.

4.2 Data Pattern-Based Verification

To detect memory cell stuck-at faults and abnormal data behavior, four data patterns were applied:

1. Incremental data
2. Decremental data
3. 0x5A5A5A5A
4. 0xA5A5A5A5

The incremental and decremental patterns are used to verify data transition characteristics during sequential address access. The 0x5A5A5A5A and 0xA5A5A5A5 patterns have alternating bit structures, which include adjacent bit combination conditions.

As shown in Fig. 1, each test pattern is generated by the DSP and written to the entire memory area. After the write operation, the same address is read and compared with the expected value. In case of a mismatch, the corresponding address and bit position can be identified.

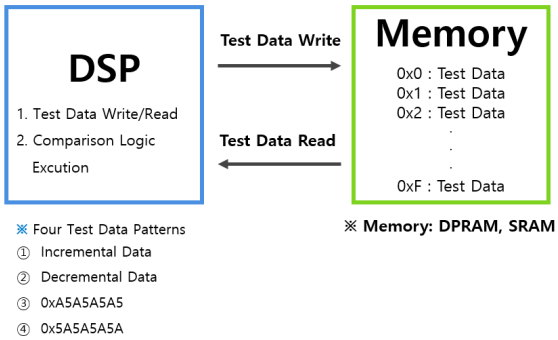

Fig. 1. Pattern-based full memory test structure using DSP and memory write/read comparison.

5. Runtime DPRAM Data Monitoring

5.1 OS-Level DPRAM Monitoring

To verify the data retention characteristics of DPRAM, a dedicated test operating software was separately implemented for verification purposes. This software is designed to execute independently from the normal application operating environment.

Test data (0xFFFFFFFF) is written once to the entire DPRAM area, and the same memory region is repeatedly read to detect any unintended data modification.

As illustrated in Fig. 2, the DSP writes the test data to all DPRAM addresses one time and continuously reads the same memory region. The read values are compared with the expected value (0xFFFFFFFF), and if a mismatch is detected, the corresponding memory address and bit position can be recorded.

This method enables focused verification of DPRAM data retention characteristics within a dedicated test execution environment.

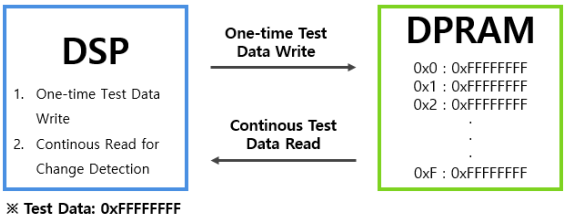

Fig. 2. OS-level DPRAM monitoring structure with one-time write and continuous read verification.

5.2 Application-Level DPRAM Monitoring

To verify whether data stored in DPRAM is abnormally modified during CPU application execution, a test program was developed to monitor the DPRAM application area under normal application operating conditions.

After writing the test data once to the DPRAM application area during the first scan cycle, the program

sequentially reads the entire target memory region at every application scan period (100 ms). The read value is compared with the expected value to detect any unintended data modification. This test is performed specifically on the application area of the DPRAM memory region.

The detailed test procedure is as follows:

1. During the first application scan, test data is written once to the DPRAM application area.
2. At every subsequent scan cycle (100 ms), the entire DPRAM application area is sequentially read.
3. The read value is compared with the expected value to verify whether any data modification has occurred.
4. If no abnormality is detected, the same procedure is continuously repeated for the entire region.
5. If a modification is detected, the corresponding DPRAM address and read data are stored in predefined variables.

The function block configuration of the monitoring program is shown in Fig. 3, and the definitions of the main variables are summarized in Table II.

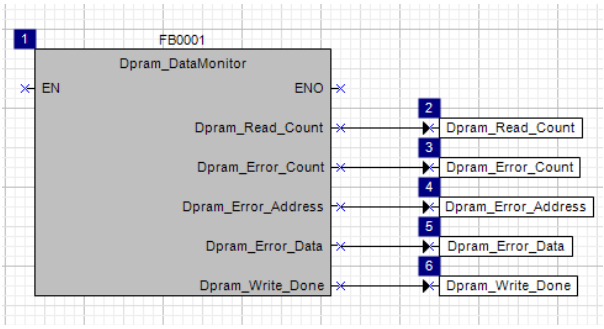

Fig. 3. Application-level DPRAM data change monitoring function block.

Table II: Variable definitions for application-level DPRAM monitoring

No	Variable Name	Description
1	Dpram_Read_Count	Number of read operations performed on the DPRAM region
2	Dpram_Error_Count	Number of detected data mismatches
3	Dpram_Error_Address	Address where abnormal DPRAM data was detected
4	Dpram_Error_Data	Read data value at the detected abnormal address
5	Dpram_Write_Done	Status flag indicating completion of test data write

6. Conclusion

This paper proposed a multi-level memory integrity verification method for a safety-grade PLC CPU module used in nuclear power plants. The proposed architecture integrates a pattern-based full-memory test with runtime DPRAM monitoring mechanisms.

The pattern-based verification enables detection of static memory faults across the entire SRAM and DPRAM regions, while the runtime monitoring mechanisms detect unintended data modifications during system operation. The proposed structure allows independent verification of static faults and runtime data modifications without interfering with normal system operation.

Future work includes long-term operational verification and optimization of test execution time to enhance applicability in safety-critical environments.

REFERENCES

[1] SOOSAN ENS, Processor Module User Manual