Considerations for Deciding Data Verifying Methods of the Nuclear Power Safety-Related Controller

Taehee Kim [a], Sung Jae Hwang [a] and Donghwa Yun[a]
[a]*Research and Development Center, POSCO Nuclear Technology, Seoul, Korea*

1. Abstract

Unintentional changes of data in memory should be treated as one of the most significant errors in nuclear power safety-related controller. So far, several data verifying methods are proposed, but they should be carefully applied according to data and hardware characteristics. In this paper, existing data verifying methods are shown and then we describe how they can be applied in nuclear power safety-related controller.

2. Introduction

Nuclear Power Safety-Related Controller (NPSC) is one of the most important components of instrumentation and control (I&C) systems in nuclear power plant. The safety guarantee activities such as protection, control, supervision and monitoring of nuclear power plant are performed by NPSC. Therefore, NPSC is designed to withstand or tolerate the errors caused by various events.

However, some errors should be detected and managed carefully as serious errors. One of those significant errors is unintentional changes of data, namely damaged date in memory. If system data such as instructions and variables referenced by system data are changed unintentionally, NPSC may become unpredictable. In the worst case, damaged data can shut NPSC down if the shut down bit is damaged and not detected before it is referenced.

Although data verification is essential task of NPSC, it needs resources such as CPU execution time shared by I&C task that is main purpose of NPSC. The more time is assigned on data verification, the less time is assigned on I&C. Therefore, data verifying process should be designed to minimize the effects on I&C and detect damaged data quickly and accurately. To achieve these design goals, CPU execution time, hardware complexity and reliability should be considered.

In this paper, we investigate considerations for deciding data verifying methods and then show the example board reflecting those considerations.

3. Related Works

3.1 Existing Data Verifying Methods

Mirroring [1] is a typical data verifying method that employs multiple memories for data redundancy. In mirroring, multiple memories store identical datum at same address as shown in Fig. 1. When CPU read certain memory address, a hardware called comparator reads that address from multiple memories and compares the each read data. If the data are identical, they will be forwarded to CPU, while the data will be considered as damaged data if they are different. Because data are compared at a hardware level, mirroring is real-time fast and does not consume CPU execution time. On the other hand, addition of hardware such as comparator and multiple memories may increase a failure rate of NPSC.

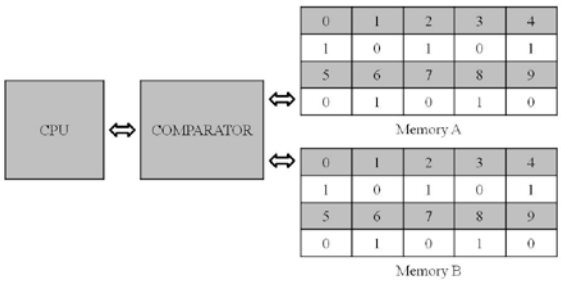

Fig. 1. Mirroring example

Checksum [2][3] and CRC [4] can be considered as data verifying methods. They are proposed to be used in network communication but they are also effective as data verifying methods. All data value is added at every address of memories and the result namely sum is stored at specific location, typically the last address as shown in Fig. 2. When NPSC want to verify data, the addition is re-performed and the sum is compared to the previously stored sum. If they are identical, NPSC can trust the memories do not contain any damaged datum while they are different, there exists one or more damaged datum. CRC is basically similar with checksum, although they employ different methodologies. Checksum and CRC investigate whole memory addresses and then derive an outcome. Whenever CPU wants to verify data, the outcome is

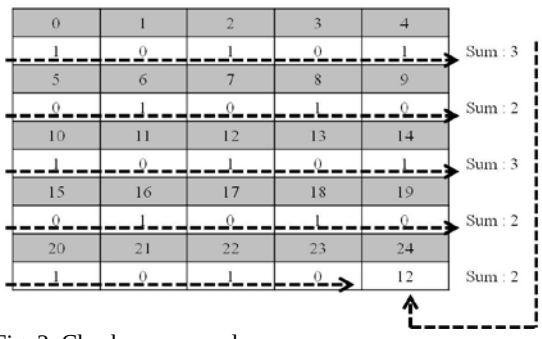

Fig. 2. Checksum example

derived again and compared with previous outcome. Whole memory investigation may be significant overhead but it can be achieved by software without hardware support. Simplicity of hardware may cause not only low cost but also high reliability.

3.2 Data Characteristics

As a kind of embedded system, NPSC stores compiled data in memory. Compiled data are classified as several sections. For example, when we compile data by well-known Code Composer Studio (CCS) [5], we may get ".text", ".switch", ".const", ".cinit", ".far", ".cio", ".stack", ".bss", etc. These sections can be classified as initialized and uninitialized sections by their data consistency.

In initialized sections, every address of the sections is set to certain initial values whenever NPSC is booted up. The representative data of initialized sections are constant variables. The constant variables may be declared in "const" keyword in most compilers. They are set to designated values by programmer at initialization and cannot be changed. Instructions such as "add", "div" and "mul" are also representative data located in initialized sections. If instructions are compiled and located in memory, they are not changed by any events except damage on memory.

Uninitialized sections, however, do not have initial value and can be changed at run time. Stacks and most variables are located in uninitialized sections.

3.3 Memory Hardware

NPSC commonly adopts hybrid memory structure consisted of synchronous static ram (SRAM), flash memory, etc. SRAM and flash memory can be categorized as several types such as dual port ram (DPRAM), NOR and NAND flash memory is well-known types.

Generic SRAM has a few megabytes capacity and a few nanoseconds access speed. It is one of the fastest memory types, thus it is suitable for data needing frequent access such as instructions. However, SRAM cannot maintain data without continuous power supply. In other words, it is volatile.

Flash memory can maintain hundreds of megabytes or a few gigabytes data after power down. But flash memory has relatively slow access speed of decades of nanoseconds. NOR flash memory can be read by random addresses but write by an unit called sector while NAND flash memory read and write by and unit called block. NAND is disadvantageous in terms of memory access, they has bigger capacity than NOR.

4. Deciding Memory Verifying Method

In this section, we investigate deciding actual data verifying methods on the basis of related works by proposing a board in shown in figure. The proposed board is consists of 4 memory modules.

4.1 The Proposed Board

First module is NOR flash memory. It is used for boot loader, that is system data those are need for boot and initialization. In general, CPUs for embedded system require NOR flash memory for boot loader because it is non-volatile and can be read by random access. As we mentioned above, NOR flash memory has smaller capacity than NAND flash memory, but it is enough for boot loader of typical a few kilobytes.

Second module is NAND flash memory. It is used for the rest system data excluding boot loader. In other words, NOR and NAND flash memory are correspond with above mentioned initialized sections. Data belongs to initialized section are not changed, so slow access speed of flash memory cannot be a significant drawback.

Third module is generic SRAM. It is used for the execution space of initialized sections stored in flash memory. Flash memory is generally slow and need sequential access of memory addresses, therefore, it is not suitable for frequent memory access such as actual execution. The data of first and second modules are copied to this SRAM module when NPSC is booted up, and then actual access for execution is occurred in the module.

Fourth module is also generic SRAM memory This module is correspond with uninitialized sections.

4.2 Deciding Data Verifying Methods

For the first and second module, namely NOR and NAND flash memory respectively, we hire checksum or CRC. Basically, the outcome of checksum and CRC is re-derived whenever any datum in the memory is changed. Furthermore, the outcome also should be re-derived for the every data verifications for comparing. The outcome re-derivation needs whole memory investigation and thus consumes significant CPU execution time. However, data on flash memories are copied to third module, that is generic SRAM, after boot and initialization. It means that data in first and second module will not be accessed except boot and initialization. As a result, whole memory investigation is occurred just one time and this is affordable to most

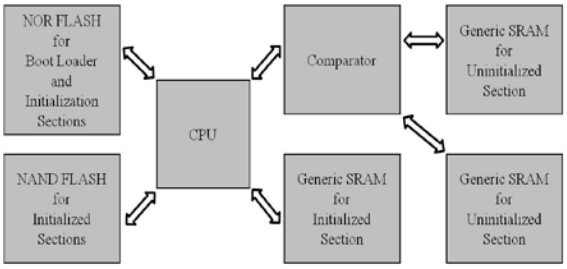

Fig. 3. The Proposed Board

NPSC.

Mirroring, checksum and CRC can be applied to third generic SRAM module. All data of the module is from flash memories, and thus they will not be changed. However, data of initialized sections should be accessed fast for actual execution because they are system data. As a result we adopt mirroring for third generic SRAM module. One more reason for choosing mirroring is that some part of verifying software itself is located in the module if we apply checksum or CRC. We cannot trust data verifying result, if the result is derived by software on the verifying target.

Forth module, that is generic SRAM, is applied with mirroring in the proposed board. Data of the module is changeable. They belong to uninitialized sections, and thus checksum and CRC should be avoided. The reason for we split third and forth module although they are both SRAM and share mirroring as data verifying methods is expandability. We assume that the proposed board could be expanded to dualization. In the dualization, two boards should be connected to each other and share their calculation result. For such sharing purpose, generic SRAM that stores variables may be replaced to DPRAM. To be prepared for that replacement, we split third and forth module.

5. Conclusions

In this paper, we investigated memory verifying methods according to data and hardware characteristics. A board that reflects the characteristics and adopts various data verifying methods was proposed. We will investigate numerical aspects such as actual access and verifying speed of the methods in the future.

REFERENCES

[1] R. W. Hamming, Error Detecting and Error Correcting Codes, Bell Sys. Tech. Journal, Vol. 29, pp. 147-160, 1950.
[2] J. G. Fletcher, An Arithmetic Checksum for Serial Transmissions, IEEE Transactions on Communications, vol. COM-30, no. 1, pp. 247-252, 1982.
[3] P. Deutsch and J.-L. Gailly, ZLIB Compressed data format specification version 3.3, Network Working Group Request for Comments (RFC) 1950, 1996.
[4] A. B. Marton and T. K. Frambs, A Cyclic Redundancy Checking (CRC) Algorithm, The Honeywell Computer Journal, Vol. 5, No. 3, 1971.
[5] Code Composer Studio v5, Texas Instrument, http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v5, 2014