

# A SOFTWARE FIX TOWARDS FAULT –TOLERANT COMPUTING

Goutam Kumar Saha

Member, ACM

CA –2 / 4B, Party Office Road, Baguitai, Deshbandhu Nagar, Kolkata  
700059 WB INDIA

## ABSTRACT

This article describes a low cost software technique for transient fault detection and fault tolerance in a processing system. The random errors caused by potential transients, Electrical Fast Transients (EFT) can be controlled by this proposed technique. Transient errors, if present, are detected and then necessary recovery action can be taken for attaining higher system reliability and tolerance thereof. It is a very cost effective tool for the application design engineers than the traditional expensive hardware fixes, or N-Version programming.

**KEYWORDS:** Burst Errors, Fault detection, Fault recovery, Fault Tolerance, Software Fix.

## INTRODUCTION

Transient error is associated with a high probability of random bit errors occurring in memory. As a result, the microprocessor-based application system becomes upset causing the total system mission failure. The EFT burst consists of multiple sparks. The burst duration is about 0.0666 ms. Each pulse within the burst has a 5 ns rise time and a 50 ns pulse width. The burst errors occur mainly due to sudden “ON” and “OFF” –ing (transient switching) of electrical heavy Electromagnetic Pulses (EMPs), EFT burst etc., [1,2].

Traditionally EFT burst potential is controlled by the use of clamping devices, shielding, grounding, isolation, balancing etc. Traditional method of error detection like checksum, CRC etc can not detect and repair any number of random errors. EFT errors are of very random in nature. However, properly designed firmware and software fixes have an important role in achieving the higher system EM Compatibility (EMC) by controlling potential EFT, EFT - burst without increasing size, weight and hardware cost. The design of this software fix is a very challenging task.

## THE NEW SOFTWARE TECHNIQUE

Application system design engineers can easily implement the proposed software fix (as shown in fig.1.) while designing a microprocessor based application system. If the application program starts from main memory location say,  $L_1^p$  through  $L_m^p$ ,

and the data memory space begins from say,  $L_1^d$  through  $L_m^d$ , then the unused memory space between the program space and data memory space is say,  $L_1^u$  through  $L_m^u$ . Now in this technique, "NOP" ( i.e., No Operation instruction code ) instruction is inserted say, n times ( n may vary from 2 to 10 ) in sequence in between the program memory space and data memory space i.e., starting from location  $L_1^u$ .

A subroutine namely "FAULTCHK" has been designed in order to detect any kind of fault in memory and program flow, caused by potential EFT, EFT Burst, EMP and then to take necessary recovery action. This proposed technique can detect errors or faults, if exist, very quickly before it has a chance to do any damage of the system. If the subroutine code "FAULTCHK" starts from say,  $L_1^s$  and one "NOP" instruction is inserted at location say,  $L_n^s$ . Let  $L_n^p$  be the memory location in program memory, and,  $L_n^d$  be the location in data memory where "NOP" instructions are inserted. This subroutine is called by the instruction "CALL FAULTCHK" statement inside the main application program with basic processing logic.

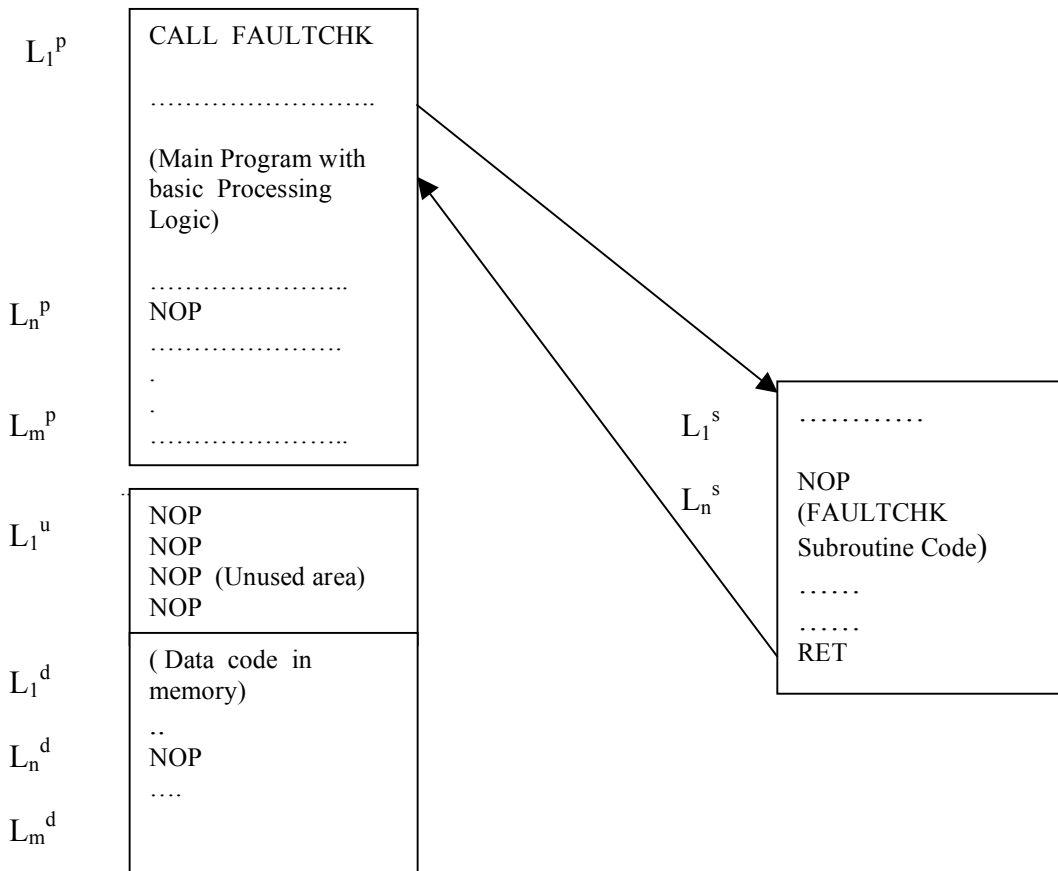


Fig. 1.



; else, there is no error in program flow and go  
; ahead with the application program.

{ End of Algorithm }

## DISCUSSION

The above steps of the subroutine “FAULTCHK” show that if some errors are present and then they will be detected in the program and data memory space, and program control is transferred to error-handling routines namely, “Error\_Program\_Memory” and “Error\_Data\_Memory” for necessary recovery or repairing the corresponding code.

If adjacent bytes in the unused memory space (i.e., from  $L_1^u$  and so on) are affected by the potential EFT burst then the program control is automatically transferred to the error routine namely, “Error\_Burst”. Similarly, the code of the “FAULTCHK” subroutine is also verified while the routine itself busy to verify the main application program with basic processing logic and data memory space for its integrity or immunity. If errors or faults are detected in the “FAULTCHK” routine then also the program control is transferred to error handling routine namely, “Error\_Subroutine”. The correctness of the program flow is also verified by checking the internal registers (if C is a register variable) say, C. If the Program Counter (PC) register is also affected at that point of run time, it is very effective way to detect it by this thoughtful checkpoint. The inserted NOP instruction codes behave as a test bed in order to catch the presence of burst errors due to potential EFTs. Thus this software fix technique provides a high reliable processing logic for any microprocessor based application. It eliminates possibilities of system ambiguity or erroneous results during run time of an application system.

$$\text{Fault Detectibility} \propto \frac{\text{Total No. of Machine Cycles with Software Fix}}{\text{Total No. of Machine Cycles in application without Fault Tolerance}}$$

..... 1(a)

$$\text{Time Redundancy} = \frac{\text{Time Required with Fault Tolerance}}{\text{Time Required without Fault Tolerance}}$$

.... 1(b)

$$\text{Space Redundancy} = \frac{\text{Space required with Fault Tolerance}}{\text{Space required with Fault Tolerance}}$$

.... 1(c)

From the above equation 1(a), it is obvious that Fault detection is higher if we insert more number of NOP instruction codes at various points inside the application. Execution of NOP instructions do not change any processor status word (psw) but provides a delay of one machine cycle only. Thus it also bring the processor into sleep mode while transients are present.

The equation 1(b) shows that Time redundancy of this proposed technique is also very less (less than 2) and easily affordable in comparison to the traditional N-Version Programming, because the total number of machine cycles involved in this proposed software fix is less than that of a typical industrial application. It is negligible than an N-version software fix.

Similarly, the equation 1(c) shows that the space redundancy is very less than 2 but little above the value of 1. Whereas a three version program has the space redundancy of more than 3.

Thus, this software fix is a very cost effective and economical tool in order to detect faults, if present, periodically and then to bring the system to a known stable state through various error-handling routines, with as little damage as possible. Interested readers may refer other related works on fault recovery [3-8].

## CONCLUSION

This proposed software technique is very useful and economical tool for transient fault detection, program code repair or for software maintenance, because it can also help to locate the error positions, by means of extra NOP instruction codes where the locations of such NOP insertion is known. NOP instruction codes can be inserted at program's sensitive parts, e.g., at decision making points. However, this technique demands a thorough study of the application system in order to find out appropriate and more critical locations for inserting the NOP codes, for avoiding system degradation. This article describes in detail how and why this proposed software technique is so useful and cost effective for gaining higher transient fault tolerance with little more but easily affordable redundancy in time and space, without using any extra hardware fix or without conventional N-version programming.

## References:

1. J.J. Donovan, "System Programming", McGraw Hill Book Company.
2. Roger L. Tokhchim, "Theory and Problems of Microprocessor", McGraw Hill Book Company.
3. G.K. Saha, (1997) "EMP-Fault Tolerant Computing- A New Approach", Journal of Microelectronic Systems Integration, 5(3), Plenum Press, USA.
4. G.K. Saha, (2000) "Transient Fault Tolerant Processing in a RF Application", SAMS Journal, Vol 38, pp 81-93, Gordon Breach.
5. Boris Beizer, Software Testing Techniques, Van Nostrand Reinhold, NY, 2<sup>nd</sup> Ed.
6. O.Serlin, "Fault Tolerant Systems in Commercial Applications," IEEE Computer, pp 19-30, Aug. 1984.
7. K.H.Kim, et.al. "Strategies for structured and fault-tolerant design of recovery programs," in Proc. IEEE Computer Soc. Int. Computer Software Application Conference (COMPSAC), Nov. 1978, pp. 651-656.

8. K.N. Chandy and C.V. Ramamoorthy, "Rollback and recovery strategies for Computer Programs," IEEE Trans.Computer,pp.59-65, June 1972.

AUTHOR:

Goutam Kumar Saha is with Centre for Development of Advanced Computing, Kolkata, India. Previously, he worked in LRDE, Defence R & D Organisation, Bangalore, India, as a scientist. His field of interest is transient software fault tolerance.

Source: Ubiquity, Volume 6, Issue 16 (May 10 - May 17, 2005)

<http://www.acm.org/ubiquity/http://www.acm.org/>