computing@tanet.edu.te.ua
www.tanet.edu.te.ua/computing

ISSN 1727-6209
International Scientific
Journal of Computing

SOFTWARE IMPLEMENTED HARDWARE-TRANSIENT FAULTS DETECTION

Goutam Kumar Saha

Centre for Development of Advanced Computing, Kolkata, India
Mail to: CA-2 / 4B, Baguiati, Deshbandhu Nagar, Kolkata 700059 India
gksaha@rediffmail.com

Abstract: *This paper examines a software implemented self-checking technique that is capable of detecting processor-registers' hardware-transient faults. The proposed approach is intended to detect run-time transient bit-errors in memory and processor status register. Error correction is not considered here. However, this low-cost approach is intended to be adopted in commodity systems that use ordinary off-the-shelf microprocessors, for the purpose of operational faults detection towards gaining fail-safe kind of fault tolerant system.*

Keywords: *Processor's hardware transient - bit-errors detection, low-cost software approach.*

1. INTRODUCTION

The objective of this proposed software based self-checking technique is to detect multiple transient bit errors (soft errors) in processor memory and processor status word (PSW). No error correction [7] is considered here. The issue of eliminating software design bugs is also not considered here. The proposed approach is intended to complement the intrinsic Error Detection Mechanisms (EDM) of a system (exceptions, memory protection, etc.) with software fix. There are various approaches that intend to complement the intrinsic EDM with a set of carefully chosen software- based error -detection techniques. Such techniques include Algorithm Based Fault Tolerance (ABFT) [2], Control Flow Checking (CFC) [3], and Assertions [4]. *ABFT* is suited for applications that use regular structures and therefore its applicability is valid for a limited set of problems. In *CFC*, the application program is partitioned in basic blocks and a deterministic signature is computed for each block and faults can be detected by comparing the run-time signature with a pre-computed one. The main problem with CFC is to tune the test granularity that needs to be used. The basic idea of Assertions is to insert logic statements at different points in the program that reflect invariant relationships between the variables of the program may lead to different problems , since assertions are not transparent to the programmer and their effectiveness depends on the nature of the application and on the ability of the programmer [8]. Readers may refer to the work [11] that makes available different fault-tolerant configurations and maintains run-time adaptation to changes in the availability requirements of an application. The work in [10] describes various fault tolerance approaches (Recovery Block and N-version Programming Schemes, Triple Modular Redundancy etc.,) that use design diversity.

The proposed work is intended to verify for immunity for electrical short-duration noises [5,9]. We detect the operational faults in a microprocessor or micro controller system by examining the Processor Status Word (PSW) bits - pattern. In this approach, we use a number (say, twenty) of successive No-Operation *(NOP)* instructions that are preceded and followed by *PUSH PSW* instructions. We need to save the PSWs into the stack area, one is before and another one is after the execution of the *NOP* codes. Then we pop both the PSWs from the stack and compare them to check for similarity at the corresponding flag bits of both the PSWs. We know that the execution of NOP codes does not alter the PSW flag bits - pattern. The effectiveness in detection of errors by this approach relies on either the unintended alterations (by transients) in the bits of the NOP codes (resulting in codes for some other instructions e.g., ADD, SUB) or in the processor status register. Again we have used a number of branch instructions (say, five) immediately after this proposed program code in order to bring the faulty program control (if a NOP code is changed to a branch or jump code) that takes the program flow out of the proposed program code, to the beginning of the detection code for ensuring the re-execution of the proposed code. However, if a NOP code is changed to a branch instruction that causes the program control to skip few NOP instructions, then such alteration remains fail-silent. In addition, in order to check for operational fault at a microprocessor environment or to harden a microprocessor system, programmer can insert (manually or by a macro) this detection- program code at the critical parts of an application program. In case, a disagreement between the PSWs is observed, programmer can bring the program control to a safe and known stable state (through error flags) in order to re-start the application for gaining fail-safe type of fault tolerance. This approach has the fault coverage limiting over the random and multiple bit errors in the memory area containing NOP codes, PUSH or POP PSW instruction codes, or in memory stack or in the processor status register containing the PSW bit-pattern and over a limited program control errors. The program code for Intel 8085 microprocessor is shown as an example. However, this method can also be easily implemented in modern microprocessors also. The work in [1] describes various automated fault injection tools. The effectiveness of the proposed transient errors detection scheme is verified on the microprocessor based system through debugging the manually modified (random single-bit flip) source program. This proposed approach is also useful for detecting the various kind of faults in a faulty processor that generates wrong and random answers (i.e., Byzantine faults). The proposed low-cost software-fix scheme is intended to detect run-time multiple transient errors at a part of the memory space and processor registers.

2. SOFTWARE IMPLEMENTATION

The proposed transient errors detection program module consists of the following basic steps. The coding involves primarily a number of NO-Operation codes that are preceded and followed by *PUSH PSW* and (*PUSH PSW, POP PSW)* respectively.

```
Step 1. Save the processor status word
        (PSW) onto a stack.
Step 2. Execute a few number of extra NO-
        Operation codes.
Step 3. Save the current processor status
        word (PSW) onto a stack.
Step 4. Pop both the PSWs from the stack.
Step 5. If both the PSWs are same, Then:
            "No operational transient fault"
                ;Go forward with application.
        Else:
            "Error Detected"
                ;Jump to an Error-Handling-
                ;Routine to reset because
                ;transient errors are found at the
                ;operational environment.
        Endif
[End of Fault Detection Steps]
```

The proposed approach can easily be implemented using any modern microprocessor. Though the software implementation is shown, as an example, for Intel 8085 assembler for better understanding, this approach is easily scalable to any 16/32/64-bit microprocessors. In fact, the code size is less and code-complexity is reduced for modern microprocessors.

Modern microprocessors have limited hardware error detection, especially for transient faults, which are vast majority of hardware failures. About *30%* failures are reported because of transient faults. At present, the frequency of the transient faults is low. However, small device size, increasing transistor counts, high clock frequency and low power supply that are accompanied with deep sub-micron technology, do not only reduce noise margin and reliability but also increase the impact of defects [12].

The *probability to detect errors* in an application, in this proposed approach, is proportional to $(m*n / (m*n + N))$. Here, n is the number of extra NOP codes, m is the number of macro calls to this detection code and N is the application size in bytes without any software fix toward fault tolerance. The advantage of this proposed approach over simple write, read and check for consistency is its faster detection and wider coverage for errors at both the memory as well as at the processor registers including PSW.

Intel 8085 is an 8 – bit microprocessor [6]. This software fix can also be applied to any microcomputer-based application. Intel 8085 microprocessor handles 8 – bit data at a time. It has the following registers:

(i) One 8- bit accumulator (ACC) i.e., register A.
(ii) Six 8 – bit general-purpose registers. These are B, C, D, E, H, and L.

(iii) One 16 –bit stack pointer, SP.
(iv) One 16 – bit Program Counter, PC.
(v) Instruction Register.
(vi) Status register.
(vii) Temporary register.

In order to handle 16 – bit data two 8 – bit registers can be combined. The combination of two 8 – bit registers is called a register pair. The valid register pairs in INTEL 8085 are B-C, D-E, and H-L. The H-L pair is used to address memories. The processor status word (PSW) consists of five status flags and three undefined bits. PSW and the ACC are treated as a 16- bit unit for stack operations.

CF	→ Carry Status Flag
PF	→ Parity Status Flag
X	→ Undefined Bit
ACF	→ Auxiliary Carry Flag
ZF	→ Zero Status Flag
SF	→ Sign Status Flag

NOP (NO Operation): States: 4, Flags: None, Machine Cycle: 1.

Nothing happens when this instruction is executed. The registers and flags remain unaffected.

Micro-operation PUSH PSW:

$[[SP] - 1] \leftarrow [A]$

$[[SP] - 2] \leftarrow PSW$

$[SP] \leftarrow [SP] - 2$

The content of the accumulator is pushed into the stack. The contents of status flags are also pushed into the stack. The content of the register SP is decreased by 2 to indicate new stack top.

Micro-operation POP PSW:

$PSW \leftarrow [SP]$

$[A] \leftarrow [[SP] + 1]$

$[SP] \leftarrow [[SP] + 2]$

The processor status word that was saved earlier during the execution of the program is moved from stack to PSW. The content of the accumulator that was also saved is moved from the stack to the accumulator.

Micro-operation PCHL (Jump to address specified by H-L pair):

$[PC] \leftarrow [H\text{-}L]$

$[PCH] \leftarrow [H]$

$[PCL] \leftarrow [L]$

The content of the H-L pair is transferred to program counter. The content of register H is moved to high-order 8 bits of the register PC. The content of register L is transferred to low – order 8 bits of register PC.

The program code is stated below.

```
/* Operational faults at the processing system is
verified by this built-in self-checking code. No-
Operation (NOP) instructions are used as an on-line
test bed. */
                PUSH  PSW
; push the current processor status word into ;stack
                NOP
; execute extra  No Operation instructions
                NOP
                NOP
                NOP
                NOP        ;  No operation
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP        ;  execute extra
;No Operation instructions
                NOP
                NOP
                NOP
                NOP        ;  No operation
                NOP
                NOP
                NOP
                NOP
                NOP
                PUSH  PSW    ; push PSW
                             ; into stack
                POP  H      ;  move the
                ;latest PSW (i.e., the one that
                ;was saved after executing -
                ; the extra  NOP
                ;instructions ) to H-L pair.
                ;POP  D
;  move the earlier  PSW (that was saved
;before executing the NOP-
; - instructions) to  D-E  pair.
; compare both the  PSWs'  bit patterns.
                MOV   A, L
;  move  L  to  register  A
                CMP   E
;  compare with the content of register E
                JNZ     ERR
;  if not zero, jump to an error handler ERR
                MOV   A,H
                CMP    D
;  compare contents of  H  and  D
                JNZ     ERR
;  if not zero (i.e., mismatch of  PSWs) , then
;jumps to  ERR  for  re-execution or to
;restart.
                MVI     A, 80H
;  move data 80H (active value) to register  A
                STA     2020H
; active data 80H at the location 2020H,
;designated as Flag_OK
;  Flag_OK  is active.
                XRA    A
;  register A is Exclusive ORed with A
```

```
                STA     2021H
;  2021H is meant for  Flag_Fault, data 00H
;is to mean  inactive,  data 80H  at  2021H
;indicates that Flag_Fault  is active.
;  Flag_Fault is inactive
                RET
;  return to  the calling  program  or  jump to
;the beginning for
;  re-executing this code by a JMP code in
;place of RET
           ERR:  XRA    A
;  Accumulator is cleared to zero
                STA     2020H
;  Set  Flag_OK  to  00H  i.e.,  Flag_OK  is
;inactive
                MVI     A, 80H
;  Accumulator is set to 80H (active data)
                STA     2021H
; set  Flag_Fault  byte to 80H to  mean the
;Flag_Fault as active.
                RET
;  return to  the calling  program or,   jump to
;the beginning for
;  re-executing this code by a JMP code in
;place of RET
```

3. DESCRIPTION

We save the current PSW into the stack area prior to execution of extra NOP instructions. A set of twenty NOP instructions is executed and then we save the PSW. We compare both the PSWs for an agreement. If both the PSWs' flag-bits are similar then it indicates that there has been no bit errors at the NOP codes and at PSWs. Any disagreement between the PSWs indicate bit errors at NOP codes or at the PSWs. Because the execution of a NOP code does not alter the flag bits of a PSW. Again, the number of NOPs can be varied from ten to forty (depending on the allowed time and space redundancy of an application and its size). It is intended to keep the size of the software fix close to a typical application size. This approach is not intended to detect transient bits- errors at the application program code. This is useful towards detection of operational faults at a microprocessor system with an overhead of (x 2) in both the memory space and time redundancy for a typical system with same size. This scheme is also intended to be useful to take preventive measures because the errors in NOPs indicate the possibility of errors in an application code also and vice versa.

4. EXPERIMENTAL RESULTS

Intel SDK85 kit, an 8085 based microcomputer (after assembling the parts) has been used for experimenting the effectiveness of this proposed technique. Machine language program has been entered (in hexadecimal format) and debugged sitting at the small keyboard. Specific keystroke sequences have been used to examine the contents of 8085 register and memory locations. In order to practically evaluate the feasibility and effectiveness of the proposed approach, the fault model of single-bit flip (at random bit number) into memory locations (at the extra NOP instructions) has been adopted by manually modifying the machine language program (while entering the source program in hexadecimal code) and then debugged. It has been observed that about 21.4% of errors have been detected by the intrinsic Error Detection Mechanisms (EDM) of the system (microprocessor exceptions, memory protections etc.). The *proposed software code could detect about 26.7% errors.* While the *Fail Silent errors* (i.e., they did not produce any difference in the program behavior) were of 38.3%. While the rest errors (13.6%) were found to be of Fail *Silent Violations* (i.e., they have not been detected by any EDM and have produced a different behavior). Transient bit errors at the NOP codes that might have occurred prior to executing this code are detected by this low cost software fix.

5. CONCLUSION

The proposed low-cost software-fix scheme is intended to detect run-time operational multiple transients caused bit-errors at a part of the memory space and processor registers. On complementing the intrinsic EDM, this scheme is particularly suited for safety-critical applications implemented by low-cost embedded systems where memory availability and execution speed are not a concern. The work can also be extended for a faster microprocessor with proper modification thereof towards gaining a reliable commodity system. System engineers can find the scheme as a useful one to harden a microprocessor system also. This approach can easily be implemented in a modern microprocessor system also. It is intended that while designing an application based on this approach, the overhead in execution time (caused by extra NOPs and examining the PSWs) will be compensated by using an affordable and faster processor.

6. REFERENCES

[1] M-C Hsueh, T.K. Tsai, R.K. Iyer, "Fault Injection Techniques and Tools," IEEE Computer, pp 75-82, April 1997.

[2] K.H. Huang, J.A. Abraham, "Algorithm - Based Fault Tolerance for Matrix Operations," IEEE Transactions on Computers, vol 33, Dec 1984, pp 518-528.

[3] S. Yau, F. Chen, "An Approach to Concurrent Control Flow Checking," IEEE Transactions on Software Engineering, vol. SE-6, No. 2, March 1980, pp. 126-137.

[4] M. Zenha Rela, H. Madeira, J.G. Silva, "Experimental Evaluation of the Fail-Silent Behaviour in Programs with Consistency Checks," Proc. FTCS - 26, 1996, pp. 394-403.

[5] T.L. Criswell, et.al, "Single Event Upset Testing with Relativistic Heavy Ions," IEEE Trans Nucl. Sci., vol. 31, no. 6, 1984, pp. 1559-1562.

[6] Roger L. Tokhcim, "Theory and Problems of Microprocessor," McGraw Hill Book Company, 1997.

[7] Stephen B. Wicker "Error Control Systems for Digital Communication and Storage," Prentice Hall, NJ, USA, pp.72- 127, 1995.

[8] A. Benso, P.L. Civera, M. Rebaudengo, M. Sonza Reorda, "An Integrated HW and SW Fault Injection Environment for Real -Time Systems," Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 117-122, 1998.

[9] T.P. Ma, P. Dussendorfer, "Ionizing Radiation Effects in MOS Devices and Circuits," Wiley, N.Y., 1989.

[10] Jean-Claude Laprie, Jean Arlat, Christian Beounes, Karama Kanoun, "Hardware- and Software - Fault Tolerance: Definition and Analysis of Architectural Solutions," Proc. 17th International Sympsium Fault-Tolerant Computing, Computer Society Press, Los Alamitos, Calif., pp. 116-121, 1987.

[11] Goutam Kumar Saha, "Transient Software Fault Tolerance Using Single-Version Algorithm," ACM Ubiquity, vol.6(28), ACM Press, USA, August, 2005.

[12] T. Sato and I. Arita, "Tolerating Transient Faults in Microprocessors," 13th Joint Symposium on Parallel Processing, 2001, Japan.

Goutam Kumar Saha
In his last seventeen years' research and development experience, he has worked as a scientist in LRDE, Defence Research & Development Organization, Bangalore, and at Electronics Research & Development Centre of India, Calcutta. At present, he is with the Centre for Development of Advanced Computing, Kolkata, India, as a Scientist-F. He has authored around one hundred research papers at various national and international journals, magazines and proceedings that include SAMS Journal, ACM, C&EE J., IEEE, CSI, Elsevier Science and International Conference Proceedings etc. He is a senior member in IEEE, Computer Society of India, ACM and Fellow in IETE etc. He has received various awards, scholarships and grants from national and international organizations. He is a reviewer of the CSI Journal, AMSE Journal, IJCPOL, IJZUS and IEEE Magazine. His field of interest is on software based fault tolerance, dependable computing and NLP. He can also be reached by <sahagk@gmail.com>.