The method of software reliability increasing for the microprocessors with multi-byte command system

Ju Akinina, T Sergeeva, S Tyurin, M Khudyakov and A Achkasov

Voronezh State Technical University, 14, Moscow ave., Voronezh, 394026, Russian Federation

E-mail: julakinn@mail.ru

Abstract. The article considers an original way of software reliability improvement of microprocessor systems with multi-byte command system in case of errors due to accidental failure while execution of the program. Program running errors (control flow errors) mean a discrepancy in the sequence of command codes executed by the microprocessor after the failure occurrence, with the working sequence of commands. The method is based on the preliminary marking of the program memory codes with the help of an additional parity/odd bit and the formation of the signal of the command code reading by microprocessor when accessing the program memory. The article proposes the analytical relations to predict the probability of detection of control flow errors for CISC-processors with any multibyte system of commands based on statistics of commands of different byteness which is contained in the code of the executable program. Along with the detection of control flow errors, all odd errors are also detected while reading the content of any program memory cell.

1. Introduction

Nowadays there are strict requirements of the reliability of operation of the functioning of the hardware and the software to the modern microprocessor systems. One of the most effective ways which allow to fend off the consequences of refusals and failures in the mode of processor functioning is the inclusion of embedded systems to improve the reliability, for example, based on watchdog mechanisms, a substantial review and classification of which are presented in [1]. Watchdog mechanisms, according to [1], mean the hardware and (or) the software components that allow to obtain information of the internal state of the controlled system. Based on this information, predictions of formalized model of the system, and the data from the inputs and outputs of the controlled system, the watchdog mechanism concludes the presence or absence of errors and malfunctions in it.

One of the classes of errors that significantly affect the reliability and safety of the functioning of the control processors are control flow errors (CFE). When such errors occur, unauthorized branches in the executable program and its various distortions occur. Such process is initialized by failures that spontaneously occur at random times and briefly distort the program counter codes, address codes or data codes in the control processor trunks. The consequences of such failures can be present for a long time in a passive state, without affecting on the functioning of the processor, but in case of certain conditions these consequences are activated and then lead to errors in the computing or control process. In this regard, tend to use such watchdog mechanisms, which maximize the probability of
detecting control flow errors with restrictions on the time of their detection, or minimize the detection time with restrictions on the probability of detection.

This article proposes various software methods for CFE detection. These methods are about dividing the executable program into blocks; program inserts are added in the beginning and in the end of each block to monitor the correctness of the transfer of control between these blocks [2, 3, 4, 5, 6]. However due to such methods applying it is impossible to predict time, needed for CFE detection. Moreover, there are restrictions for decreasing the program blocks size due to overhead increasing. Thus, time, needed for CFE detection, can be equal to time of concrete program block execution. Also, it is important to pay attention to the appearance of different articles, which contain doubts of the application efficiency of basically software methods of CFE detection. It is stated in [7] that “... performed quantitative analysis shows that the existing CFC methods are not only ineffective in providing the protection from mild faults, but also require additional operational and energy costs”.

Hybrid methods of CFE detection are developed to reduce the CFE detection time and overhead, for example [8, 9, 10, 11]. These methods assume the necessity of the executable program modification and the additional hardware using, which implement the watchdog mechanism.

Methods of CFE detection only with the help of hardware are least diverse. We can distinguish the method presented in [12, 13] among them. In these works, an algorithm for the operation of the watchdog mechanism based on predicting the address of a program memory cell from which the code for the next command should be read is proposed. The essence of this algorithm is as follows. The processor of the controlled system after start-up refers to a fixed address to the program memory cell, where it reads the code of the first executable instruction. The address of the read instruction (that is, the PC code) and the instruction code are also read by the watchdog, which remembers the address code and checks whether the instruction code belongs to the branch instructions. If the read code is not a branch instruction code, the watchdog predicts the address code of the cell containing the code of the next instruction by adding to the stored code of the program counter the value "instruction size", which determines the number of calls of the controlled system processor to the program memory necessary for the execution of the current instruction. If the read code is the code for the branch instructions, the watchdog monitor predicts code of the cell address containing the next instruction code, by the addition to the code of a program counter value in the "instruction size" – if the condition for branching is not performed, or adds the value of "Offset" and "instruction size" – if the condition for branching is performed. As stated in [12], the considered watchdog mechanism based on the prediction of the address of the program memory cell provides a good compromise between the coverage of fault detection and the occupied area on the chip. However, this watchdog mechanism has a certain drawback. It is clear, that to predict the address of the next memory cell containing the command code, the control processor and the watchdog use the same source data – the code of the executed command. Therefore, if the code of the executed command is distorted when reading from the program memory, both the control processor and the watchdog will determine with error of address of the next memory cell containing the command code. Moreover, for this watchdog mechanism there is not the formalized methodology of predicting its effectiveness for an arbitrary executable program. This circumstance requires the using to assess the effectiveness of multiple runs of a real executable program in the simulation of random failures [14, 15].

2. The original watchdog monitor based on the code layout of the executable program

In [16], there is a method of software reliability improvement based on the code markup of the executable program. It is focused on the control processors with multi-byte command format. A feature of such processors is that during random branching in the program, not only a part of the program code is skipped, but subsequent distortion of the program is possible due to the interpretation of instruction codes as data codes, and data codes as instruction codes.

To implement this proposed method, you should use the program memory with an additional bit of parity / odd control and the control processor which generates an external signal in case of access to the program memory to notify how the read code will be interpreted from the data bus. If this code is
interpreted as a command code, the alert signal will take the value of the logical unit, otherwise it will be equal to the logical zero.

Based on the preliminary analysis of the processor working program presented in the form of binary codes, the addresses of those cells of the program memory, which will contain the command codes, are determined. For each cell, the amount of single values contained in the command code is determined, and if the amount of single values in the command code is even, then the unit must be written to the additional bit of program memory (i.e. the command code is supplemented to odd). For all other code words of the working program, including unused program memory cells, code words are supplemented to parity. The working program, represented as binary codes, is loaded into the program memory along with the parity / odd bits. During the operation of the microprocessor system, a special watchdog mechanism [16], in case of access to the program memory, sums modulo the two bit values of the readable code: the parity / odd bit value and the value of the alert signal generated by the microprocessor. If "0" is formed at the output of the adder modulo two, it indicates that the code read from the program memory has not got an odd number of distortions and is correctly interpreted by the processor. If the output of the adder the unit - "1" is formed, it indicates that the code read from the program memory, has distortion, or this code is interpreted by the processor incorrectly. In this case, the watchdog mechanism will generate a signal that causes the processor interruption. The maintenance routine of this interruption is selected by the user and focuses on a targeted response to possible random failures. The following situations are possible.

The control flow errors (regardless of the specific reason) are manifested in the fact that at some point in time \( t_c \) the microprocessor begins to interpret command codes as data codes, and data codes as command codes. Let's identify the moment of time \( t_c \) with the moment of command code reading from the program memory cell with the address \( A \). We believe that the control flow error is equivalent to an unauthorized transition of the microprocessor to a memory cell with a random address \( A_o \). Then, for the considered method of increasing the fault tolerance and the selected type of microprocessor, we have the following cases.

Case 1. From the cell with the address \( A \) the code of a single-byte command is read. The processor goes to an unauthorized \( A \) address. If there is a command code (one-, two-, or three-byte) in the \( A \) cell, the failure is not detected, and the control flow error continues to propagate. If the \( A \) address doesn't contain the command code, then the failure is detected, and the control flow error can be localized.

Case 2. From the cell with the address \( A \) the code of the two-byte command is read. The processor goes to an unauthorized \( A \) address. If there is a command code in the \( A \) cell, the failure is detected. If the \( A \) cell doesn't contain the command code, the processor goes on unauthorized address \( A_o \). If there is a command code in the cell of \( A_o \), the failure is not detected. If the data code is retrieved at \( A_o \), the failure is detected.

Case 3. From the cell with the address \( A \) the code of the three-byte command is read. The processor goes to an unauthorized \( A \) address. If there is a command code in the \( A \) cell, the failure is detected. If the cell \( A \) doesn't contain the command code, the illegally processor goes over to the address \( A_o \). If there is a command code in the cell of \( A_o \), the failure is detected. If there is a data code in the cell with the address \( A_o \), the processor illegally goes to the address \( A_o \). If at the address \( A_o \) the command code is read, the failure is not detected, and if the data code is read, the failure is detected.

3. Analytical method for evaluating the effectiveness of the original watchdog monitor
Let's consider the possibility of analytical determination of the effectiveness of the original watchdog monitor on the base of the executable program codes layout.

As follows from the analysis, any program executed by the processor is represented by a mixture of one-, two- and three-byte commands. At the same time, we believe that the codes in the program memory are not changed (not distorted) under the influence of failures – failure only distorts the address code when accessing memory cells. Under such accepted conditions, the method of analytical evaluation of the efficiency of the considered method of increasing the failure resistance of
microprocessors is reduced to the solution of the classical problem of probability theory—"extraction of balls with return". As can be seen from the above, the procedure for analyzing various cases of failures that lead to control flow errors consists in successive checks of code belonging to a set of commands and a set of data. This can be interpreted as finding the probability of co-occurrence of events that may be dependent or independent.

We suppose that we have a processor whose command system consists of one-byte, two-byte and three-byte commands (for example, i80 microprocessor). Then any program for this microprocessor will consist of \( n_1 \) single-byte commands, \( n_2 \) double-byte commands and \( n_3 \) three-byte commands. The size (in bytes) of the entire program is defined as (1):

\[
n = n_1 + 2n_2 + 3n_3
\]

Find the probability \( P \) of missing control flow errors as (2):

\[
P = c_1P_1 + c_2P_2 + c_3P_3
\]

where \( P \) is the probability of absence of errors while the \( i \)-byte command execution; \( c_i \) - the share of \( i \)-byte commands in the working program.

We introduce the following values: \( P_k \) - the probability of command code reading from the program memory cell and \( P_d \) – the probability of data code reading from the program memory cell. Then, due to (1), we have:

\[
P_1 = P_k = \frac{n_1 + n_2 + n_3}{n}
\]

\[
P_2 = \frac{n_2 + 2n_3}{n} \times \frac{n_1 + n_2 + n_3}{n}
\]

\[
P_3 = \frac{n_2 + 2n_3}{n} \times \frac{n_2 + 2n_3}{n} \times \frac{n_1 + n_2 + n_3}{n}
\]

In the analytical method of performance evaluation, based on the expressions (3) – (5), it was assumed that the addresses of the \( A_i \), \( A_d \), \( A_z \) program memory cells after the failure are formed by a random law. Such address formation would be adequate only in case of very intense failures. In reality, the intensity of such failures is very low, and therefore the addresses of program memory should be formed as \( A = A_i \), \( A_d = A_{i+1} \), \( A_z = A_{i+2} \). Due to this algorithm of accessing to program memory after a failure, it follows that skipping a single-byte command failure is not detected when a command code is found in a cell with the \( A \) address. A two-byte command failure is not detected when a data code is found in a memory location with the \( A \) address and a command code is found in a memory location with the \( A_i = A_{i+1} \) address. A three-byte command failure is not detected when a data code is found in a memory location with the \( A \) address, and a data code is found in a memory location with the address of \( A_i = A_{i+1} \), and a command code is found in a location with the address of \( A_z = A_{i+2} \). In other words, the probability of the absence of failure in a single-byte command is proportional to the number of commands in the program; the probability of the absence of failure of two-byte commands is proportional to the number of consecutive "data-command" fragments in the program; the probability of the absence of failure of three-byte command is proportional to the number of consecutive “data-data-command” fragments. Then:

\[
P_1 = \frac{(c_1 + c_2 + c_3)n}{c_1n + 2c_2n + 3c_3n} = \frac{c_1 + c_2 + c_3}{c_1 + 2c_2 + 3c_3}
\]

\[
P_2 = \frac{(c_2 + c_3)n}{c_1n + 2c_2n + 3c_3n} = \frac{c_2 + c_3}{c_1 + 2c_2 + 3c_3}
\]
Expressions (3), (4), (5) and (6), (7), (8) can be represented in a general form for a system of arbitrary byte commands.

The verification of the adequacy of the proposed analytical method is based on a specially developed simulation model. The initial data for modeling are the volume (in bytes) of the hypothetical program and the percentage of one-, two- and three-byte commands in it. The volume of a hypothetical program can vary from $10^3$ to $10^6$ bytes. The percentage of commands with different number of bytes can be specified both deterministically and randomly.

The estimated program is represented as an array of memory cells, where each one has its own unique address $A_i$. In the memory cells, which contain the command codes are recorded the binary number 1,2,3 – pointing at the same time, at the number of bytes in this command; and at the binary number 0, which is written in the memory cells, indicates an additional byte (operand) of each command. After entering the initial data and starting the simulation model, a hypothetical program is generated as a random sequence of one-, two- and three-byte commands in accordance to their specified percentage.

The table 1 presents some results of the effectiveness evaluations of the original watchdog monitor based on the simulation model ($P_m$ – the probability of failure) and the calculated expressions $P_{(3)-(5)}$ and $P_{(6)-(8)}$.

### Table 1. The results of the effectiveness evaluations of the original watchdog monitor.

| $n_1$, % | $n_2$, % | $n_3$, % | $P_m$ | $P_{(3)-(5)}$ | $P_{(6)-(8)}$ |
|----------|----------|----------|-------|---------------|---------------|
| 33       | 33       | 34       | 0.330 | 0.289         | 0.332         |
| 20       | 30       | 50       | 0.303 | 0.230         | 0.300         |
| 20       | 50       | 30       | 0.336 | 0.259         | 0.329         |
| 30       | 50       | 20       | 0.357 | 0.306         | 0.363         |
| 30       | 20       | 50       | 0.316 | 0.254         | 0.314         |
| 50       | 20       | 30       | 0.377 | 0.360         | 0.383         |
| 50       | 30       | 20       | 0.404 | 0.387         | 0.406         |
| 40       | 30       | 30       | 0.351 | 0.321         | 0.353         |
| 30       | 35       | 35       | 0.320 | 0.279         | 0.326         |
| 10       | 30       | 60       | 0.293 | 0.198         | 0.292         |
| 10       | 60       | 30       | 0.323 | 0.235         | 0.332         |

4. Conclusion
The effectiveness of the proposed method for improving reliability based on the markup of the codes of the executable program decreases as the single-byte instructions in the executable program increase. Thus, the probability of missing control flow errors does not exceed 0.4, if the amount of single-byte commands in the executable program does not exceed 50%. In this case, the efficiency can be improved only by modifying the executable program as follows. The fragments of the executable program consisting of long sequences of one-byte commands are determined. These sequences are divided into shorter ones and between which are placed have additional blocks containing of several bytes. The number of bytes in additional blocks is should be in 1.5...2 times greater than the number of bytes of the longest command. Inserted bytes are marked with an additional bit as data bytes. With an equal ratio of one-, two- and three-byte commands in the program, the probability of detecting control flow errors is 0.6...0.7, and all odd errors are detected while reading the contents of any program memory cell.

References
[1] Platunov A E and Sterkhov A S 2017 Watchdog mechanisms in embedded computing systems
Journal scientific and technical of information technologies, mechanics and optics 17(2)2 301

[2] Vemu R and Abraham J A 2011 CEDA: Control-Flow Error Detection Using Assertions IEEE Transactions on Computers 60(9)

[3] Boroomandnezhad T and Azgomi M A 2013 An efficient control-flow checking technique for the detection of soft-errors in embedded software Computers & Electrical Engineering 39(4) 1320–32

[4] Li A and Hong B 2010 Online control flow error detection using relationship signatures among basic blocks Computers and Electrical Engineering 36(1) 132-41

[5] Khoshavi N, Zarandi H R and Maghsoudloo M 2011 Control-Flow Error Detection Using Combining Basic and Program-Level Checking in Commodity Multi-core Architectures 6th IEEE Symposium on Industrial Embedded Systems (SIES’11) 190-1

[6] Oh N, Shirvani P P and McCluskey E J 2002 Control-Flow Checking by Software Signatures IEEE Transactions on Reliability 51(2) 111-22

[7] Rhisheekesan A, Jeyapaul R and Shrivastava A 2019 Control Flow Checking or Not? (for Soft Errors) ACM Transactions on Embedded Computing Systems 18(1)

[8] Azambuja J R, Lapolli A, Rosa L and Kastensmidt F L 2011 Detecting SEEs in microprocessors through a non-intrusive hybrid technique IEEE Transactions on Nuclear Science 58(3) 993-1000

[9] Popov G 2007 A Hardware Dual Watchdog Timer for Increasing the Reliability of Embedded Systems Computer engineering 1(2)

[10] Trephena Patricia A, Abinaya E, Harika S, Jasmine Florence Hebciba P and Sumathi C 2019 FPGA Implementation of an Improved Watchdog Timer for Safety-Critical Applications International Research Journal of Engineering and Technology (IRJET) 6(3) 2218-21

[11] El-Attar Ashraf M and Fahmy G 2007 An improved watchdog timer to enhance imaging system reliability in the presence of soft errors IEEE International Symposium Processing and Information Technology 1036-40

[12] Parra L, Lindoso A, Portela M, Entrena L, Restrepo-Calle F, Cuenca-Asensi S and Martinez-Álvarez A 2013 Efficient Mitigation of Data and Control Flow Errors in Microprocessors 14th European Conference on Radiation and its Effects on Components and Systems (RADECS) 402-5

[13] Parra L, Lindoso A, Portela M, Entrena L, Grosso M and Sonza Reorda M 2011 Control Flow Checking through Embedded Debug Interface 26th Conference on Design of Circuits and Integrated Systems pp 339-43

[14] Romansky R and Kirilov K 2018 Architectural Design and Modelling of a Web Based Application for GDPR Clarification AIP Conference Proceedings (American Institute of Physics) Proceedings of the 44th International Conference on Applications of Mathematics in Engineering and Economics (AMEE’18) vol 2048 pp 060006-1-8

[15] Romansky R 2019 Formalization and Discrete Modelling of Communication in the Digital Age by Using Graph Theory ed Madhumangal Pal, Sovan Samanta & Anita Pal (IGI Global, USA)

[16] Rozhkov M V and Tyurin S V 2014 The method of improving the microcomputer reliability Patent N 2530325 RU (Moscow: Rospatent)