Use of Natural Information Redundancy in On-Line Testing of Computer Systems and their Components

Oleksandr Drozd
Department of Computer Intelligent Systems and Networks
Odessa National Polytechnic University
Odessa, Ukraine
drozd@ukr.net

Anatoliy Sachenko[1,2]
[1]*Kazimierz Pulaski University of Technology and Humanities in Radom,*
Radom 26-600, Poland,
sachenkoa@yahoo.com
[2]
Ternopil National Economic University
Ternopil, 46027, Ukraine

Svetlana Antoshchuk
Department of Computer Intelligent Systems and Networks
Odessa National Polytechnic University
Odessa, Ukraine
asgonpu@gmail.com

Julia Drozd
Department of Computer Systems
Odessa National Polytechnic University
Odessa, Ukraine
dea_lucis@ukr.net

Mykola Kuznietsov
Department of Computer Systems
Odessa National Polytechnic University
Odessa, Ukraine
koliaodessa@mail.ru

I. INTRODUCTION

On-line testing occupies an important place in ensuring the functionality of computer systems and their digital components. Methods and means for on-line testing are aimed at checking the trustworthiness of the results calculated at the output of digital circuits. This goal was originally formulated as the detection of digital circuit faults in the operating mode, since the methods and means for on-line testing began to develop within the framework of the exact data model, which was reflected in the theory and practice of constructing totally self-checking schemes.

For exact data, i.e. integers in nature (the numbers of the elements of the sets), these goals of the on-line testing do not differ. Indeed, an error detected in a calculated result simultaneously indicates both the presence of a fault and an incorrect result, which for exact data consists entirely of most significant bits and is therefore unreliable.

Computer systems demonstrate the dominant development in the processing of approximate data, when it is impossible to further ignore the structure of the approximate number consisting of the most and least significant bits. Faults in them cause errors that are respectively essential and inessential errors for the trustworthiness of the result.

Approximate data are measurements and, as a rule, are presented and processed in floating-point formats. In the first personal computers, such processing was carried out in the optional Intel 287/387 coprocessors. The next Pentium family provides accelerated pipelined processing of approximate data, and a modern graphics processor contains thousands of floating-point pipelines used for performance of parallel calculations on CUDA technology.

From the position of the resource approach, this development of personal computers reflects the general tendency of structuring the resources, i.e. models, methods and means, under the peculiarities of the natural world, among which its parallelism and fuzziness turned out to be most apparent in computing.

The resource approach notes three levels of development of the resources in the process of their integration into the natural world: replication, diversification and self-sufficiency.

Replication, as the simplest form of production and parallelization, will always be selected with open resource niches – ecological, technological, market, etc. At the level of replication, integration into the natural world is carried out at the expense of productivity. In the natural world, it occurs under the slogan: “Give birth more than die”. With the closure of resource niches, clones can only survive by showing features, that is, becoming individuals, rising to a level of diversification, where survival is at the expense of authenticity, trustworthiness, i.e. adequacy to the natural world. At this level, we solve problems of analysis, including diagnostic problems that for the decision require the use of information redundancy, which reflects the fuzziness of the natural world.

We observe filling of the natural world with objects of the increased risk – powerful power plants, power grids, high-speed transport, various types of weapons. This process changes the benchmarks in the development of computer systems from productivity to trustworthiness, defining them as instrumentation and control safety-related systems, aimed at ensuring functional safety of both the system and the control object to prevent accidents and reduce their consequences.

In these systems, the operating mode is diversified, dividing into normal and emergency. The goal of on-line testing, which plays a key role in the operational assessment of the state of the system, is also diversified. In emergency mode, results are monitored with their check for trustworthiness. Normal mode is used as a test for clearing digital circuits from faults, where elements of testability and self-testing are important. The circuits have to show faults in the normal mode, i.e. to be checkable, and to mask faults in emergency mode for obtaining reliable results.

It should be noted that safety-related systems, like cyber-physical systems, and Internet of Things systems, receive initial data from sensors, i.e. get measurement results that relate to approximate data. Thus, the position of the on-line testing is enhanced in the processing of approximate data for new directions in development of computer systems and information technology.

Self-sufficiency determines the level of development goal. Resources strive in their development for self-sufficiency. In particular, our models, in which methods and means are being developed, rise to an understanding of the natural resources that are already embedded in the models, methods and means like the basis for the development of their self-sufficiency.

In confirmation to it, the results of arithmetic operations demonstrate self-sufficiency in the presence of natural information redundancy, sufficient for the implementation of on-line testing with the use of the forbidden values in the result code.

The paper is devoted to the analysis of natural information redundancy as an inherent property of the results of arithmetic operations with approximate data. There are some examples of the use of natural information redundancy for the on-line testing of digital circuits. This paper reveals the universal nature of this resource, its availability in all applications for on-line testing in the processing of approximate data. Section 2 proves the existence of natural information redundancy in the results of all arithmetic operations with approximate data. Here information redundancy is considered in the traditional form of the presence of forbidden values in the result code. Section 3 draws attention to the natural information redundancy generated by fixed and floating-point data formats. Section 4 shows the natural information redundancy in version form for program code of FPGA projects in critical applications.

II. Universal nature of natural information redundancy in the approximate results

Traditionally, information redundancy of a result code is interpreted as the presence of forbidden values in it, which cannot be calculated in a properly operating digital circuit.

To obtain information redundancy, the binary number, as a rule, is supplemented with a check code, which lengthens this number. For example, in residue checking, which is widely used for on-line testing of arithmetic operations, the check code by modulo three complements the original n-bit number with two bits to the $(n + 2)$-bit number.

The values of the original number with the corresponding check code form a set of allowed words that can be calculated with the correct functioning of the scheme. This set contains 2^n words.

However, an $(n + 2)$-bit number can take 2^{n+2} different values, of which only 2^n words are allowed. The remaining words make up a set of forbidden words. For $n = 8$, the checking by modulo three generates 256 allowed and $1024 – 256 = 768$ forbidden words.

On-line testing is based on the distinction between allowed and forbidden words. Identification of a forbidden word is unambiguously associated with an error caused by a circuit fault, and determines the result to be unreliable.

Natural information redundancy is the presence of forbidden words in the result code, considered without the check code.

Specific examples of results, the codes of which contain forbidden words, indicate the existence of natural information redundancy. However, the presence of forbidden words in the results of processing approximate data is not a special case, but a general rule.

The universal nature of the natural information redundancy in the results of arithmetic operations on approximate data follows from the following two statements:

1) The code of a product has natural information redundancy.

2) Multiplication is a key operation of approximate calculations.

The first statement is based on the same length of the input and output words in the multiplication operation and on the commutative law for it. The input word consists of codes of factors. For n-bit binary factors, the input word has a size of $2n$ bits. The output word, i.e. the complete product code also has a length of $2n$ bits. Therefore, the input word and the output word take the same number of 2^{2n} values. However, the commutative law states that the product does not change from changing the places of the factors. Thus, the same product corresponds to different input words with interchanged factors. Therefore, any output words have no match in the set of input words and therefore belong to the forbidden.

For example, for $n = 2$, the factors take the values 0, 1, 2, 3, and their products, 0, 1, 2, 3, 4, 6, 9, form a set of seven allowed values. The product code has 4 bits in size and takes values from 0 to 15, of which the numbers 5, 7, 8, 10, 11, 12, 13, 14 and 15 form a set of forbidden values.

The second statement follows from the use of the normal form in the representation of numbers when they are written in floating-point formats. This representation contains the mantissa, the exponent, by default implies the base of the number system and combines these elements using the multiplication operation.

The presence of a multiplication operation in the floating-point number itself determines the use of multiplication, in one form or another, in all operations with mantissas. The results of these operations inherit the properties of the product, i.e. possess natural information redundancy.

The residue checking does not distinguish between allowed and forbidden values of the product code and the code of any other result of the processing of approximate data. He relates all the values of a product to allowed words and spends

resources on creating information redundancy, ignoring its natural form. Diversification of words in the code of an approximate result by dividing them into allowed and forbidden allows you to perform on-line testing at the expense of internal resources of arithmetic operations with approximate data. Successful examples of such use are known for multiplication of binary codes and their squaring.

In case of multiplication of two mantissas, the binary code of the product contains the forbidden values of a type $P = k\,C$, where $k = 2^{n-1}, \ldots, 2^n - 1$, $C = 2^n + 1$ – a prime number, for example, 17, 257 and 65537 for $n = 4$, 8 and 16, respectively. Really, number C and multiple to it numbers cannot be the product of two n-bit binary numbers as they have no decomposition on two factors with the size smaller than $n + 1$. These forbidden values easily are being identified as they form the code with repetition, when a younger half of the product coincides with the senior half. The Z error is being detected in case of performing a condition of $A\,B + Z = P$, where A and B are binary codes of mantissas of the normalized factors: $A = 2^{n-1}, \ldots, 2^n - 1$, $B = 2^{n-1}, \ldots, 2^n - 1$.

The errors typical for the iterative array multiplier distort the product on the weight of any one its bit $U = n, \ldots, 2n - 1$ and they can be described as $Z = \pm 2^U$. We experimentally showed that all such errors can be detected for $n = 4$, 8 and 16. In the course of the experiments, the specially developed program determined A, B and k values for which the condition of error detection is satisfied for each U. For example, for $n = 4$, the error $Z = 2^5$ or $Z = -2^5$ in bit $U = 5$ is detected on the condition of $8\times13+32=8\times17$ or $12\times14-32=8\times17$ in case of $A = 8$, $B = 13$, $k = 8$ or $A = 12$, $B = 14$, $k = 8$, respectively.

The S result of squaring shows natural information redundancy in values of the residue by the modulo. For example, $S \bmod 3 \neq 2$. The error is detected when performing a condition $(S + 1) \bmod 3 = 0$.

We experimentally showed that all Z errors typical for the matrix scheme of a squarer with word size of $n = 8$, 16 and 32 can be detected. The error detection scheme defines the residue $1 = 01_2$, or $2 = 10_2$ during the correct work and $+0 = 00_2$ or $-0 = 11_2$ in case of error detection. The scheme belongs to totally self-checking.

III. Natural information redundancy of data formats

Statements about the only form of information redundancy based on the existence of forbidden values are known.

At the same time, data formats demonstrate a different form of information redundancy, which is characterized as natural and has properties important for on-line testing of the digital circuits in the processing of approximate data.

Modern fixed and floating-point formats can be opposed to each other by the positions at which codes of numbers are placed. In the case of a fixed point, the low bits of the numbers occupy the lower positions of the format. On the contrary, the floating-point format is characterized by the placement of the most significant bits of the number in higher positions.

The size of the fixed-point format is selected based on the representation of the longest number. When writing numbers of shorter length, the highest format positions are not used. Such natural information redundancy is the basis of logarithmic checking, which could not compete with a method of the residue checking within the framework of the exact data model. At the same time, this method demonstrates an important property of more probable error detection in higher positions than in younger ones. Such a distinction between essential and inessential errors based on the natural information redundancy of data formats is particularly important for the on-line testing of approximate calculations.

In floating-point formats, the unit value of the most significant bit of the mantissa is fixed. The following zero values form the natural information redundancy of floating-point formats.

We developed the program in Delphi 10 Seattle demo-version for a pilot study of logarithmic checking on the example of the fixed-point multiplier. The Ac check code of number A is defined by the number of bits of its significant part and $Ac = 0$ in case of $A = 0$. Then for A, $B > 0$, the product $P = A \times B$ can be checked as $Pc = Pc^*$ or $Pc = Pc^* - 1$, where $Pc^* = Ac + Bc$. In case of A, $B \geq 0$, $Pc^* = Ac\,Bz + Bc\,Az$, where $Az = 0$ for $A = 0$ and $Az = 1$ for $A > 0$; $Bz = 0$ for $B = 0$ and $Bz = 1$ for $B > 0$. It follows from the following inequalities: $2^{Ac-1} \leq A < 2^{Ac}$, $2^{Bc-1} \leq B < 2^{Bc}$, $2^{Pc-1} \leq A < 2^{Pc}$ and definitions of the lower (top) bound of the product as a product of the lower (top) bounds of factors.

The iterative array multiplier contains $n \times (n - 1)$ matrix of operational elements. Multiplication is carried out for one clock cycle on the factors generated in a random way. In each clock cycle, fault of short circuit between two any points of the scheme in accidentally chosen operational element is injected. In an experiment, the word size $n = 8, \ldots, 15$ and the number $n_M \leq n$ of most significant bits is determined.

The multiplier has calculated both incorrect result $Pe = 6948$, $Pe = 0001011111111110000_2$ and its check code $Pc = 13$ which is compared to the Pc^* code. The error E = 4096 was caused by fault of short circuit between the s sum and the level of logical unit in the full adder of operational element 76 (line 7 and column 6 of multiplier matrix). This error is detected when checking $Pc > Pc^*$. The experiment also shows probabilities of detection and the skipping of essential and inessential, positive (0 - 1) and negative (1 - 0) errors. Besides, the trustworthiness of logarithmic (LC) and residue (MC) checking is estimated.

The trustworthiness is calculated by the formula $T = P_{ED} + P_{ID}$, where $P_{ED} = P_E\,P_D$ and $P_{ID} = (1 - P_E)\,(1 - P_D)$ are probability of essential error detection and the probability of inessential error skipping; P_E and P_D – the probability of an essential error and probability of error detection, respectively. Residue checking showed detection of all errors, i.e. $P_D = 1$, and its trustworthiness is defined as $T_{MC} = P_E$.

The experiment showed the following values of probabilities: $P_{ED} = 1.3 / 20.2 = 6.4\%$, $P_{ID} = 10.4 / 20.2 = 51.5\%$, $P_E = 9.8 / 20.2 = 48.5\%$. Trustworthiness of logarithmic and residue checking is estimated as 57.8% and 48.5%, respectively.

IV. VERSION FORM OF THE NATURAL INFORMATIOM REDUNDANCY

In systems of critical application, functional safety is ensured through the use of fault-tolerant solutions, among which multi-version technologies are of particular importance. They allow to resist to common cause failures and accumulation of the hidden faults, to raise integrity and also at the same time a checkability of FPGA projects and trustworthiness of results in normal and emergency mode, respectively.

FPGA designing with the LUT-oriented architecture widely is used by in critical applications, creating conditions for development of natural information redundancy in a version form when for the ready FPGA project, the set of versions of the program code with various properties is generated.

The LUT unit generates function from n arguments. For $n = 4$, the LUT unit has 4 address inputs of A, B, C, D on which a, b, c, d arguments forming the $dcba_2$ address code, 16 bits of memory and one output. The sequence of LUT codes forms a program code of the FPGA project. The version redundancy of this code is based on an opportunity to create its versions for the same hardware implementation of the project. Versions are generated at the level of ordered pair LUT1 - LUT2 of the LUT units connected among themselves: LUT1 output bit transfers to the address input of the LUT2 unit.

This bit can be transferred by a direct or inverse value, forming two versions of a program code. The inverse value of bit forms by inversion of bits of the LUT1 memory. This inversion on an address input of LUT2 unit is compensated by change of places of its bits of memory.

The contrast of the described versions can be effectively used for opposition to faults of short circuit of the next address inputs of LUT. Really, inversion of one of them interchanges the position of a set of input data on which this fault is shown in the form of an error or is masked. It allows to find versions which promote fault detection on some input data and mask on others, i.e. to raise a checkability of circuits and trustworthiness of results in the normal and emergency modes of safety-related systems.

Each version shows LUT2 unit with the correct circuit and in case of short circuit between address inputs C and D. The fault copies the line $dc = 00_2$ into lines $dc = 01_2$ and $dc = 10_2$. Inversion on the D address input is compensated by change of places of the lines $dc = 0X_2$ and $dc = 1X_2$ where X = 0 or X = 1. This example is reviewed for a case when bits with addresses $XX0X_2$ and $XX1X_2$ are addressed in normal and emergency mode, respectively.

The checkability of the circuit and trustworthiness of result in mode U is estimated by the formulas $C_U = Z_U / Y_U$ and $T_U = 1 – C_U$, where $U = N$ and $U = E$ for normal and emergency mode, respectively, Z_U and Y_U – the number of the distorted bits and their total in mode U.

An example shows the following results: $Z_N = 3$, $Y_N = 8$, $C_N = 37.5\%$, $T_N = 62.5\%$, $Z_E = 3$, $Y_E = 8$, $C_E = 37.5\%$, $T_E = 62.5\%$ and $Z_N = 4$, $Y_N = 8$, $C_N = 50\%$, $T_N = 50\%$, $Z_E = 1$, $Y_E = 8$, $C_E = 12.5\%$, $T_E = 87.5\%$ for versions 1 and 2, respectively. Thus, transition from version 1 to version 2 raises at the same time both a checkability of the circuit and trustworthiness of result respectively in normal and emergency mode. The checkability improves for 12.5%, and trustworthiness – for 25%.

V. CONCLUSIONS

Information redundancy is the basis for solving the analysis issues, including the issues of on-line testing the digital circuits in computer systems and their components. Information redundancy is created in the form of forbidden values by forming the check codes of numbers, which requires the certain hardware and time resources. Existing examples of using the natural information redundancy are not regular. The orientation of modern computer systems on critical applications and processing the approximate data increases the importance of on-line testing the digital circuits in relation to checking the approximate calculations. In these circumstances, a multiplication becomes the key operation, and the natural information redundancy inherent in the product gets the universal character, extending to the results of all arithmetic operations on mantissas. Thus, the on-line testing obtains a natural resource for the dominant processing of approximate data.

Natural information redundancy also takes a place in fixed and floating-point formats in the form of unused positions. Logarithmic checking, using this form of redundancy, demonstrates an important feature of more likely detection of essential errors compared to inessential ones.

The version form of natural information redundancy allows to reach at the same time a high checkability of FPGA projects and trustworthiness of results respectively in normal and emergency mode of safety-related systems.