

Assurance of some system reliability characteristics In formal design verification

S. Frenkel

Institute of Informatics Problems, Russian Academy of Sciences, Moscow, Russia

e-mail: fsergei51@gmail.com

ABSTRACT

In this paper we consider two ways of using both formal-logical models based on Model Checking and Markov probabilistic models for semi-automatic verification of robustness to Soft Errors of a target design, and assurance of these properties as well. One of this approach to verification is well-known Probabilistic Model Checking while another is suggested recently a combination of traditional Model Checking with a two-dimensional Markov chain.

A comparison of computational complexities of these approaches shows that combination of Model Checking formal verification with analysis of fault-tolerant property could reduce overall design cost, what can increase considerably the formal verification tools effectiveness.

Index Terms - Performance, Design, Reliability, Verification.

1. INTRODUCTION

Among various functional requirements to a system designed designer of digital programmable systems needs in reliability-related properties assurance [1,2,3], in particular, in fault-tolerant property assurance if the system has to work in presence of some transient faults [4].

A specification of software-hardware designs is based on different requirements to the system to be developed. As the requirements relate usually to different domains knowledge (functional properties, performance, reliability) a bridge between functional correctness specification, which is main goal of design formal verification and the numerical probabilistic properties such as reliability/fault-tolerance that are the targets for system level assurance is very important.

If we follow the traditional conceptual model of the requirements [1] based on distinguish between the “environment” and a machine specific model (e.g., a transition system, or, more specifically, Finite State Machine (FSM)), the modeling of some external impacts leading to some errors (incorrect computations) are performed in the “environmental” terms.

In general, there are two possibilities for the fault-tolerance/reliability assurance analysis - statistical and analytical model-based [2].

As for statistical models, some essential challenges can take place if a life-critical system has requirements to provide very small probability to be failed, say 10^{-9} and beyond, that is infeasible to measure these tiny rates statistically [2], as it would require very essential expenditures for data collection and simulation. That is the analytical models may be more practical, if they allow to implement a verification method with rather acceptable computational complexity. In fact, the computational complexity, along with ability to represent possible errors in terms of some “environmental” variables activations, e.g. inputs in FSM model is one of important factors of a model availability for the assurance analysis.

Another important issue of design verification is coordination of functional verification on a middleware level and top-level safety requirements stated probabilistically.

Our aim is to consider a possibility of concurrent exploiting of both models mentioned above both for verification and assurance.. The efforts which are required to coordinate their activity also lead to rise in the cost of the design process. In particular, as the functional correctness verification is performed by using Model Checking [4,6], which is very expensive both by own cost (caused by the computational complexity) and labor cost, a way of combination of this activity with providing of fault-tolerant property could be reduce overall design cost, that is to increase effectiveness of formal verification tools usage.

This paper extends the results reported in [7]. The outcome of the research was a model-based framework based on both Markov chain approach suggested in [7] and Model checking for quantitative verification, in comparison with Probabilistic Model Checking [8], which is an extension of model checking that allows to analyze the probabilistic information about possible errors.

2. FORMAL VERIFICATION OF MULTIPROCESSOR-BASED SYSTEMS ROBUSTNESS

Formal methods, Model Checking in particular, can tell a designer whether a system designed is correct functionally [3]. As for reliability issues, evaluation of digital systems robustness with respect to transient errors is an important part of the digital systems design process.

The checking of ability of a design to provide own correct functioning in presence of faults we will call as *robustness verification*.

Currently a number of formal methods based approaches to the robustness analysis of the transient faults have been suggested [4,5]. In general, the soft errors effects can be modeled by SEU (Single event Upset, that can be caused by so-called *single-event transients* (SET) [4] (e.g., registered by a latch) which provokes a variable flip (bit-flips, in particular) occurring. That is a flipping of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

VeriSure 2014, Vienna, Austria

Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...\$15.00..

a state variable of a system model (*transition system* based, as a rule) leads to the SEU. In any case, such errors resulting from SEUs are not permanent but rather transient (i.e., can be corrected in course of time), and are hence called *soft errors*.

A *Model Checking problem* is a problem of checking whether a given model, describing a design verified, satisfies a given property: $M \models \phi$ which means that a property is satisfied for the model M , where the model M is a *finite state transition system*, and its properties which must be true for an *implementation* of the system at the lower level of the system design (e. g., by a *structural* FSM, that is, with binary coding of its inputs, states, outputs [6,14]) are expressed in a *propositional temporal logic* like *Linear temporal logic* (LTL) and *Computational Tree Logic* (CTL) [3].

Let us consider a Model Checking-based method for the robustness analysis of transient faults/SEU [4,6]. One of modeling device is *assertions* which are represented in Linear Temporal Logic (LTL) or a branching logic (CTL) format [4]. An assertion is said to trigger when its antecedent evaluates to true. The fulfillment of these conditions guarantees to find all system's statements such as that erroneous execution of these statements causes the assertion to fail during formal verification. Correspondingly, the state variable can be protected, e.g., by replication (at the architectural level of modeling), or by self-correcting coding (at a micro-architectural one). This is possible to do as this model comprises a state variable that records whether an SEU has occurred.

However, in fact, such solution considers only a *potential* erroneous functioning with respect to a particular fault SEU hit, while an error a local node may be disappeared before it might violate the correct behavior. It can lead the designer to unwarranted overhead for this node hardening without any reliable assurances. Nevertheless, it is possible to modify this Model Checking approach to the robustness analysis in order to get more explicit quasi-quantitative characteristics. For example, in accordance with the model [6], faults can occur in any state of FSM modeling the circuit designed. The faulty states is derived such that a state is defined over a set $S = 2^R$ of sequential elements R . Within these R elements, some threatened elements $P \subseteq R$ will be found to be protected. The set of potential faulty states, which must be protected, represent the states in which the system could be after a time during which faults occur.

The *metrics* which characterize the states threatened by the SEU, are based on the three particular state sets, that reflect the design D robustness and can be specified from the CTL expressions: (i) *safe(D)* - the set of repairing states, defined by so-called *repairing sequence* of states of D such that it begins from a state in the reachable state space of the faulty model, and it ends in a repairing state, (ii) *required(D)*, that is at least one state of this set must be visited along the repairing sequence before reaching the repairing state, (iii) *forbidden(D)*, that is none of the state of this set must be encountered along the repairing sequence. On the base of this model, designers can estimate the set of protected variables P (with corresponding register set, *if they have been assigned to this design stage*), the cardinality of the reachable state space of D in absence of fault, the cardinality of the reachable state space of the faulty model, and so-called "potentially repairing" states, and "systematically repairing" states. From these measures one can evaluate the robustness of the circuit in terms of *ratio of repairing states* over total states. The strategy of repairing capability analysis

with these metrics is to analyze the ratios, such that a ratio near 100% represents a high robustness degree while a ratio below 50% characterizes a poorly robust circuit.

However, transient faults induced by the SEU may or may not be latched by a storage cell, but in case of the fault occurrence a correct operation of the corrupted module can be restored, that is this system can have a *self-repairing* (*self-healing*) ability [9]. If the probability of such "self-healing" is very high, it may be not reasonable to use this protection.

Therefore, it would be helpful to have a possibility to estimate the self-healing time for different architectures at early design stages.

3. PROBABILISTIC MODEL OF SEUs IMPACT ON STATE VARIABLES

Self-healing ability can be achieved, for example, either due to the monotonic property of Boolean functions describing the transitions of FSM representing the given system [9] or by a reconfiguration [11].

As long as we consider FSM formalism as a model of target system at the high level of the design (network of components FSM, in general) we may use a conceptual model based on comparison of the FSM correct behavior with erroneous one (in the present of some transient faults) [7]. Thus, since the FSM, which is used for functional Model Checking-based are accessible us, we may arrange the estimation of the probability distribution function of various characteristics of the self-healing/repairing time. For this aim we should compute transition probability matrix of the Markov chain (MC) Z_t defined on the product of two FSMs, both with transition table corresponding to the target design and with tables corrupted by a transient fault (SEU, in particular).

Below, for sake of the paper reading convenience, some of results of [5] will be repeated briefly.

The MC Z_t is computed by the known probabilities of input vectors and transition tables of the FSM, considering the signals with the same input probabilities distributions $\Pr\{u_i = 1\}$, $i = 1, \dots, m$, where m is the number of Boolean variables. Let $\{X_t, t \geq 0\}$ and $\{Y_t, t \geq 0\}$ are the MC describing the target behavior of a fault-free system, that is, functioning without effect of any transient faults (X_t), and in the presence of some faults (Y_t), S is the set of states of the MCs. Let $Z_t = \{(X_t, Y_t), t \geq 0\}$ be an MC corresponding to behavior of the MCs pairs that is an MC with space $S_2 = S \times S$ of pairs (a_i, a_j) , $a_i, a_j \in S$. Fixing the states numeration in some way, for example, as $s_1 = (a_1, a_1)$, where the "ones" are the numbers of states of X_t, Y_t , $s_2 = (a_1, a_2), \dots, s_n = (a_1, a_n)$, $s_{n+1} = (a_2, a_1)$, $s_{n \times n} = (a_n, a_n)$, the vector of the state transition probabilities of the Markov chain MC after t steps may be defined as $\mathbf{P}(t) = (p_1(t), \dots, p_{|S \times S|}(t))$, $t \geq 0$.

Then, the event of coincidence of the trajectories of X_t and Y_t after the soft error effect stopped (self-healing) may be denoted as hit of the MC Z_t in a subset $D \in S_2$ of all states (a_i, a_j) , $i = 1, \dots, n$, with the same right and left elements.

Using these notions, the self-healing time can be expressed as:

$$T = \min\{t : X_t = Y_t\} = \min\{t : Z_t \in D\}$$

Let us define the set of Z_t states as $S^* = \{(i, j), i, j = 1, \dots, n, j \neq i\} \cup A0 \cup A1$, where (i, j) means that fault-free automaton is in the state i , but the automaton under a transient fault is in the state j , $j \neq i$; $A0, A1$ are absorbing state of Z_t with probabilities $p_{(0)}(t)$ is the probability that a faulty behavior of the automaton can appear right up to the step t and the output both automata will coincide by this moment;

- $p_{(1)}(t)$ is the probability that a faulty output of the automaton can appear right up to the step t before the automaton behavior will be restored.

Also we consider the probability of $p_{ij}(t)$ that the MC Z_t will transit to a state (i, j) , that is fault-free automaton will be in the state I while the faulty one in the state $j \neq i$, but the outputs of both automata are the same.

The number of the Z_t states is $n(n-1)+2$.

If the fault-free automaton is at the initial moment 0 in the state $i_0 = X_0$, where the faulty automaton will be in state $j_0 = Y_0$, $j_0 \neq i_0$, then component (i_0, j_0) of the vector of initial states $\mathbf{P}(0)$ is equal 1 and other components of the vectors are zero. Indeed, this condition means that there exist a clock t when the SEU corrupts the system behavior (modeled by a FSM) such that it will get in a state j_0 instead i_0 . Initial state $\vec{p}(0)$ is determined by the distribution of initial states of both automata. If the fault-free automaton is at the initial moment 0 in the state i_0 , whereas the faulty automaton will be in state $j_0 \neq i_0$, then $p_{(i_0, j_0)}(0) = 1$, while other components of the vector $\vec{p}(0)$ are zero.

$$\vec{p}(t) = \vec{p}(t-1)P^* = \vec{p}(0)(P^*)^t.$$

Correspondingly, the probability that there exists an time instance that moment the output of the automaton will be fault-free after the SEU effect topping is

$$\pi_0 = \lim_{t \rightarrow \infty} p_{(0)}(t),$$

the probability of the output corruption (in a remote time instance) is

$$\pi_1 = \lim_{t \rightarrow \infty} p_{(1)}(t),$$

Then, the probability that (after the SEU effect terminated its action) the automaton will not return on its right transition trajectory but its output remains correct on each transitions is

$$\pi = 1 - \pi_0 - \pi_1.$$

Thus, we can characterize the robustness of a system relatively to the SEUs as: (i) stationary probabilities that finally the system, represented by given automaton either will recover (π) after the SEU termination or it does not recover after such event (π_1), and (ii) by the number of clocks after which the system may be self-repaired with a suitable probability.

The choice of one or another of these characteristics depends on the reliability model which the system designer uses.

State transition probabilities matrix P is computed by the functions $X(a_m, a_s)$, which describe transitions from the current state a_m to the next state a_s , takes the Boolean value. This computation is performed under assumption that input variables of the FSM are independent random vectors from one time step to another and that their probability distribution is fixed so that, at any given time step, an input (of total m) takes place with probability $\Pr\{x_i = 1\} = p_i$, $i = 1, \dots, m$.

There are some possible ways of incorporating of such characteristics in general design verification process. For example, the probability π corresponds to an event

(described in Section 2, (i)- (iii)) which can be interpreted as a *liveness* property [3] usually checked in formal verification, that one of those properties whose violation never has a finite witness ("no matter what happens along a finite run, something good could still happen later"). As this property can be fulfilled with some probability only, it is necessary to consider a decision making model. Below this aspect for the *Combined Logical-Numerical* approach mentioned above (depicted in details in the Figure1) and Probabilistic Model Checking will be considered and compared.

4. PROBABILISTIC MODELS OF ROBUSTNESS VERIFICATION

In order to combine the possibility of formal verification of a system design with account of some aspects of their stochastic behavior mentioned in the Introduction, the verification model has to contain some theoretical-probabilistic devices. Below two possible approaches will be compared (Figure 1): (i) the *Combined Logical-Numerical* method mentioned above which is based on formal verification (Model Checking) of elements which potentially can be corrupted by some external noise (provoking the SEU effects) with computation of the probabilities of these errors manifestation at the system primary output using the 2-dimensional Markov chain Z_t mentioned above (Section 3), or (ii) popular in the last decade Probabilistic Model Checking [12] which is an algorithmic procedure for recognizing whether a given system with some stochastic behavior satisfies probabilistic specifications, such as *the probability of logical correctness at the output of a logic network must be at least 0.9*. This approach to verification are employed to quantify properties such as a component failure, what can be used in formal analysis of robustness. Correspondingly, the system models are probabilistic (typically variants of Markov chains), in the sense that they describe the property in terms of the probability of making a transition between states.

4.1 Probabilistic Model Checking

The Probabilistic Model Checking uses a state transition system with probability values attached to the transitions as the system model. Examples of such transition systems are discrete time Markov chains (DTMCs), continuous time Markov chains (CTMCs) and Markov decision processes (MDPs) [8]. CTMC allows to express the system transition rates (defined as transition rate matrix), that is a performance measure. However, in this paper we will consider DTMCs only, which enable define conditions of error manifestation.

Mathematically, underlying model is a Markov chain (DTMC mentioned above) defined along with some way of various classes of transitions labeling, namely [14]:

A discrete-time Markov chain (DTMC) is a tuple $\mathbf{M} = (S, P, L)$ where S is a (countable) set of states, $s \in S$ is an initial state, $P : S \times S \rightarrow [0; 1]$ is a transition probability matrix such that $\sum_{s' \in S} P(s', s) = 1$ for all $s \in S$, and $L : S \subseteq 2^{AP}$ is a labelling function mapping each state to a set of *atomic propositions* (AP).

Thus, the robustness verification in the framework of such model can be performed as a result of comparison of the Markov chain states-and-transitions with conditions for corresponding probabilities, defined by the labeling, expressed by PCTL.

Formally, PCTL formulas are interpreted over the states of a DTMC. PCTL replaces the CTL existential and universal quantification over paths (see Section 2) with the probabilistic operator $P_{\geq p}[\psi]$, where $p \in [0; 1]$ is a probability bound or

threshold. Path formulas can occur only within the scope of the probabilistic operator.

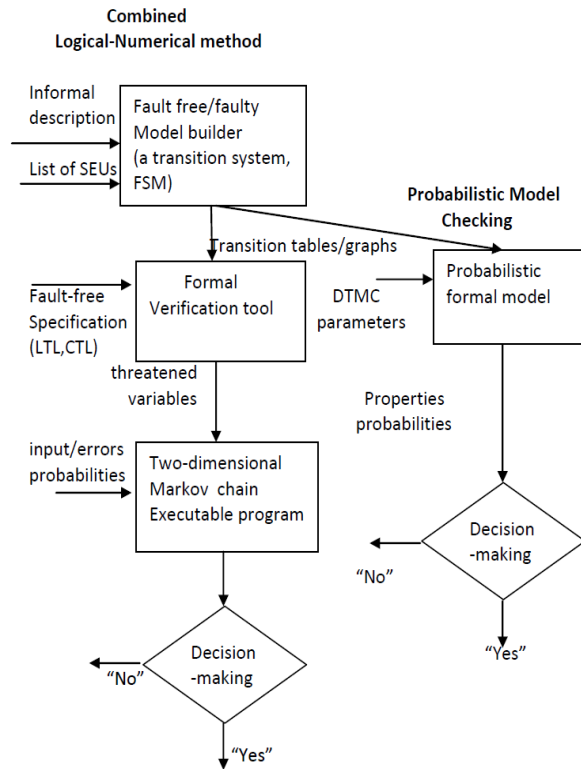


Figure 1. Probabilistic models of Verification

A state satisfies $P_{\triangleright \triangleleft} [\psi]$ if the probability of taking a path from s that s satisfies ψ is in the interval specified by $\triangleright \triangleleft p$.

The PCTL specification for Model Checking allows to described the failure as a set of states, identified by some labels, that corresponds to the faulty states of the system (i.e., when a state is different from a specified in system design). The PCTL formulae of the form $P_{\triangleright \triangleleft} p(aUb)$ with $a, b \in AP$, meaning that the probability to go along a path from the initial state to a state which satisfies b , passing only states in which a holds, is less or equal p . Then, a question concerning assurances can be formulated as “what is the probability of reaching a b -labelled state passing only through states that do not satisfy a ?”, what corresponds, say, to one conditions determined the threatened elements (Section 2) *forbidden(D)*. Technically, such verification can be implemented by different well-known program tools for Probabilistic Model Checking, say PRISM [12].

The most commonly studied property for reliability analysis is the probability of reaching a certain state, which typically represents the success of the system or some failure condition. Both success and failure are modeled by absorbing states, representing (Section 3)) either the correct accomplishment of the task (“healing”) or the task’s failure, respectively. The reachability formula in this case has the following form: $P_{\triangleright \triangleleft} p(\psi_l)$, where l is the label of the target absorbing state.

The operator P expresses computing the probability mass of all paths satisfying its operand mentioned above. For a given DTMC P computes the probability mass of the set of all paths through M which lead to any target state within the time

period T . The procedure for the computation of P consists of two steps. First, the probabilistic model checker renders all target states absorbing, i.e. for each target state s all outgoing transitions are ignored and a self loop with transition probability 1 is added to s . Second, the model checker computes the transient probability of all target states at the time point T [12].

Let us compare computational complexity of these two probabilistic methods.

5. COMPLEXITY ANALYSIS

From the implementation point of view, the execution time which is affected computational complexity of underlying algorithm is very important challenge.

5.1 Complexity of Model Checking Based Robustness Analysis

The computational complexity of traditional (non-probabilistic) Model Checking depends on the type of logic used. In linear-time logic, all instants are linearly ordered from past to future there is only one possible future: future is determined. In programs. For nondeterministic programs, the linear-time viewpoint applies to the runs of the system: any given run determines a single future. The nondeterminism of the system is taken into account at a different level, by considering all the runs. In branching-time logics, the future is not determined and any given instant may have several distinct immediate successors the tree-like structure of the frame.

Let TS be a transition system with N states and M transitions.

Then [13]:

- Model-checking LTL-formula Φ has time-complexity $O((N+M)2^{|\Phi|})$
 - linear in the state space of the system model,
 - exponential in the length of the formula
- Model-checking CTL-formula Φ has time-complexity $O((N+M)|\Phi|)$, and
 - linear in the state space of the system model and the formula.

However, Model-checking LTL versus CTL can be *exponentially shorter* than their equivalent in CTL.

That is linear-time may be preferred for its simplicity, both from notational and conceptual point of view.

But some of very important properties, especially regarding the concurrency is better to express by CTL that is why is used in many industrial-oriented tools. Correspondingly, the mentioned above approach to using of deterministic model checking to assurance proof in the presence of different soft error has the same complexity. Indeed, by defining specifications to reason about the correct and faulty behaviors of system designed a designer enables to assess the extent to which faulty behavior affects properties of the system and the system’s ability to recover from faulty states.

Let some conditions concerning correctness behavior is expressed by a CTL formula $AG\phi$.

In the system model corrupted by some faults, this ϕ formula holds if the property of the system is unaffected by the faulty behavior. In this case the property would hold along all paths. That is if the formula holds the system would be completely tolerant to the fault.

For example, if we consider a counter (e.g., four-bits) than the formula $AG ((bit[0] = 0) \& (bit[1] = 0) \& (rst = 0)) \rightarrow AX ((bit[0] = 1) \& (bit[1] = 0))$

which means that designed correctly counter with the states encoded as above must transit from the state $\{bit[0] = 0, bit[1] = 0\}$ to $\{bit[0] = 1, bit[1] = 0\}$, if there is no reset signal rst .

Denoting a proposition concerning faulty model transformation as *fault*, the following formula can be used to verify that ϕ always holds when the system behaves correctly:

$$AG(\neg \text{fault} \rightarrow \phi)$$

This states that ϕ always holds when a fault is never occur into the model.

As it can easily be shown the following formula can be used to reason about the recoverability of the system when a fault occurs, that may be used for degree of assurance estimation.

$$AG(\text{fault} \rightarrow AF \phi);$$

This formula captures the fact it is always true that whenever there is a fault presents along all paths at some point ϕ holds. Thus, if this specification holds, the system will recover from the fault in terms of property ϕ .

5.2 Probabilistic Model checking Complexity

Formally, the Probabilistic Model checking verify a formula (linear time or CTL) for DTMC, corresponding to probabilistic model of the system designed.

Let $|\phi|$ be the size of the formula ϕ defined as the number of Boolean connectives plus the number of temporal operators in the formula.

As it follow from short description of the procedure mentioned above (Section 4) the complexity is in general polynomial in the size of the model and polynomial or exponential in the size of the formula [8].

Main runtime bottleneck of the probabilistic model checking algorithms is the linear algebraic operations (linear equation solving for the Markov Chain). Their share of the total runtime of the algorithms increases with the states number. So, for real world problems, that tend to have large state spaces, this dependence is even more critical.

More exact estimation, for finite DTMC M and PCTL state-formula, as it follows from [13], the PCTL model-checking problem can be solved in time $O(\text{poly}(\text{size}(M))) \cdot n_{\max} \cdot |\phi|$, where $n_{\max} = \max\{n \mid \phi 1 U^{\leq n} \phi 2 \text{ occurs in } \phi\}$ with and $n_{\max} = 1$ if ϕ does not contain a bounded until-operator.

Note, that most expensive are the operators for unbounded U ("until" [3]) and also so-called *rewards operators* which represent (measure) a reward is a numerical quantity that is assigned to particular states or transitions of the system, for example, to define the expected time to the first failure, or the average fraction of time during which the system is fault free. However, in this paper the computational complexity is considered without the reward calculation.

5.3 Comparison

In order to compare these probabilistic approaches mentioned above we should consider all computation needed to a design verification which include both functional correctness verification and robustness (reliability), which could mean the assurance of the design ability.

Let us compare from this point of view the Combined Logical - Numerical method (Figure 1, left part) and PMC.

In order to exclude all ambivalence, we will consider CTL-based formal specification only.

Thus, the complexity of the CTL based (that is specified using CTL) robustness model checking is

$$O(\sum_{i=1, K} |\phi_i| + O(L(S(S+1)+2)^3))$$

where S is the number of FSM states modeling the system verified, K is the number of formulas used for design specification under given faults, first summand is the complexity of threatened elements of the design model (given design level), second summands is the computational (time) complexity of product of two Markov chains with S states, corresponding to fault-free and corrupted systems (Section 3), L is the number of faults considered for the given design, represented by vector $P(0)$ of initial distribution in accordance with Section 3 (as for analysis each possible transient fault defined by $P(0)$ it is necessary to solve the linear equations).

The time complexity of PMC expression depends on robustness verification model that is if the model checking is implemented for DTM like Z_t Section 3, or the recovery conditions formulas are used as input for a PMC tool (e.g., PRISM).

In first case, the time complexity is $O((2S(S+1)+2)^3) \cdot |\phi|$.

However, the designer which is interested to make a decision about protection one of another elements of the design is forced to interpret these probability in terms of events, corresponding to system behavior in a real-life condition.

From this point of view, one benefit of model checking using in the Combined Logical-Numerical scheme is that when a property is violated, a counterexample is produced describing a trajectory that led to the property's failure. The counterexample can then be used as an aid in determining the design elements threatened by SEU or other transient faults. Besides, as the state of each module of design is represented by a set variables, and the global state of the model at any point in time is determined by the values of all such module variables, it is possible use different modelling level-Algorithmic State Machine [14], FSM, Register-Transfer Level (RTL).

Thus, in our approach based on combination of traditional Model Checking and product Markov Chains, the models of different level are accessible (and visible) during all design cycle.

In contrast, the Probabilistic Model Checking approach to robustness verification, where the design input (Figure 1) is, in fact a DTMC (representing its probability transition matrix and vector of initial state probabilities) does not provide any diagnostic information but only the pure probabilities that a given property holds. In contrast to traditional model-checking techniques, the feedback diagnostic support for in case when a property is violated is rather limited.

The counterexample generation is, in fact, a combinatorial search problem which problem which, in general, growth exponentially with the DTMC space size.

Besides, using this approach we may verify both functional correctness and robustness, whereas PMC tool has to be added with a Model checking one, that means the total design cost

reducing due to excluding part of expenses regarding both the tools purchase and verification labor cost.

Thus, taking into account the computational complexity together with the problem of errors analysis (diagnostic) accompanied the functional verification, the benefits of combined logical-numerical approach is obvious.

6. CONCLUSION

This paper considers assurance of design quality using both formal and probabilistic verification of some properties concerning fault-tolerance, provided, for example, by a hardening of some system's elements. One of important factor of implementation of such combination is computational complexity. It has been shown, that adding the standard set of tools with a program, computed transition probability matrix of the FSMs product mentioned above can combine the formal verification with some aspects of fault-tolerant analysis with complexity less than Probabilistic Model Checking. This approach provides a bridge between functional correctness verification and the system robustness assurance in presence of possible soft errors.

7. ACKNOWLEDGMENTS

This work has partially been supported by the Russian Foundation for Basic Research under grant RFBR 12-07-00109.

8. REFERENCES

- [1] A. Filieri, C. Ghezzi, G. Tamburrelli: "A Formal Approach to Adaptive Software: Continuous Assurance of Non-Functional Requirements". Special Issue of Formal Aspects of Computing 2011.
- [2] J. Rushby, Software Verification and System Assurance, 2009, Seventh IEEE International Conference on Software Engineering and Formal Methods.
- [3] E.M. Clarke O. Grumberg, and D. A. Peled. Model Checking, MIT Press, 2000.
- [4] S. A. Seshia, W.Li, S. Mitra, "Verification-guided soft error resilience", Proceedings of the Conference on Design, Automation and Test DATE07, 2007, p.1442-1447.
- [5] S. Frenkel, A. Pechinkin, Estimation of self-healing time for digital system under transient faults, *Informatics and its Application Journal*, vol.4, issue 3, 2010, pp.2- 8.
- [6] S. Baarir et al, Complementary Formal Approaches for Dependability Analysis, 2009 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems.
- [7] S. Frenkel, "Some probabilistic and formal techniques for fault tolerant Design", Proceedings of 5th IEEE International workshop on Impact of Low Power Designs on Test and Reliability, Annecy, France, Jun. 2012, p. 16-17.
- [8] H. Hansson, B. Jonsson, A logic for reasoning about time and reliability, *Formal Aspects of Computing* 6 (1994) 512–535.
- [9] Levin, A. Matrosova, S. Ostanin, "Survivable self-checking sequential circuits", Proceedings of DFTT01, 2001, p. 395.
- [10] U. Krautz, et al, "Evaluating coverage of error detection logic for soft errors using formal methods", In Proceeding DATE 2006, p.176-181.
- [11] M.G. Gericota, G.R. Alves, M.L. Silva, J.M. Ferreira, Run-time management of logic resources on reconfigurable systems, *Design, Automation and Test*
- [12] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in Proc. 23rd

International Conference on Computer Aided Verification (CAV'11), ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[13]. C. Courcoubetis et M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 24(4) :857-907, 1995.

[14]. Baranov, S. Frenkel, V. Zakharov, "Semi-formal verification for pipelined digital design based on Algorithmic State Machines", *Informatics and its Application Journal*, Vol.4, issues 4, 2010, p.49- 60.

[15] M. Kwiatkowska. Quantitative verification: Models, techniques and tools. In ESEC/FSE'07, pages 449–458. ACM Press, 2007.