Dependability Evaluation of Complex Embedded Systems and Microsystems

Malassé O.⁽¹⁾, Buchheit G.⁽¹⁾, Hamidi K.⁽¹⁾, Pock M.^(1, 2), Belhadaoui H.⁽¹⁾, Walter M.⁽²⁾, Aubry JF.⁽³⁾

⁽¹⁾ A3SI, Arts et Métiers ParisTech, 4 rue Augustin Fresnel, F-57078 Metz cedex 3

⁽²⁾ LRR, TU-München, Institut für Informatik 10, Boltzmannstr. 3, D-85747 Garching bei München
olaf.malasse@ensam.eu, walterm@in.tum.de

The evaluation of the dependability performance (Reliability, Availability, Maintainability and Safety-RAMS) of complex embedded systems requires the development of new approaches. In software-intensive systems, the dependability structure of the functions depends on the software. The search of fault sequences must involve software and hardware. The proposed method contributes to the qualitative and quantitative safety analysis of systems and micro-systems.

Keywords: Complex systems, Finite State Automaton, Binary Decision Diagram, SW/HW analysis, RAMS analysis.

1. INTRODUCTION

The engineering of digital electronics and software play a central role in systems design and verification process. Levels of integration of electronic components and complexity and size of onboard code continually growing and is accompanied by a greater sensitivity to perturbations (the natural and technological environment are also becoming more aggressive) and attacks (strong requirement on the capacity of systems to communicate) miscellaneous. The functional structure of systems (hardware and software package) has de facto become complex and variable. Prevent and eliminate faults are part of the expected development and verification process. The faults from residual errors in the design or from the implementation of electronic components, or software, are traceable, identifiable and reproducible but with difficulty detectable! Potential causes of failure are many: hardware, software, development environments ... The non-consistency, the combinations of errors potentially latent and dormant depending on the system state, the complexity of applications makes the analysis difficult. The types of failure [1] can also be combined: failure to perform (non-progression, infinite loop, crash), functional (bad processing), operational (processing time not met, unavailability). A system is a complex set of functions, carriers and subjected to hazards (trigger systematic errors, Bit Flip, hardware failures), to provide a defined service, whatever its internal state, the state of its environment and the level of applied stress (Figure 1).

FIGURE 1: System

Processing (detection and recovery) errors provides, at the cost of increased levels of complication and complexity of the system, the guarantee of a better probability of behaviour of the system. The integration of mechanisms (hardware and software) for fault tolerance by design, by control or by diagnosis and reconfiguration, significantly modify the hardware and software architecture (firmware, application) and affect the dynamics and RAMS of systems. The functional and dysfunctional aspects must be considered for the performance assessment (in terms of reliability and quality of service) of a system. Many references frame the development of systems constrained by imperatives of safety and security, as the generic standard IEC 61508 [2] which describes the development process of E / E / PE systems (Electrical, Electronic or Programmable Electronic systems) dedicated to applications concerning the safety of industrial systems.

The paper is structured as follows: section 1 gives an overview on the problem of reliability evaluation of software / hardware set. The methodology (modelling and reduction technique model) is presented in section 2. Two cases of application (Safety Instrumented System, microprocessor) are treated as third party.

1.1 Evaluation place in the design phase

The complexity of systems and the integration of verification into the design process to meet the functional specifications require significant efforts that hinder the issues of dependability (RAMS). Introduced too late, their integration into the design process is poor, although they affect, as well as the verification of functional properties, on the design phase (Figure 2).

FIGURE 2: Positioning the reliability evaluation in the design cycle

The objective of the verification / validation is to reveal faults that have been introduced during any phase of the development lifecycle (specification, design, implementation, production, commissioning). It must accompany the development process to reduce the cost of their elimination [3]. Minimize the number of residual errors, elimination of systematic errors was a utopia, is one of the major objectives of the validation. The presence of random errors in operational phase requires validation of architectures, both hardware and software. The expected outcomes are qualitative (length of failing sequences, location of structurally weak points), and quantitative (assessment of the metric such as RAMS). Depending on the level of abstraction involved, on the level of detail of the models (granularity) and the type of traces searched, multiple approaches are available.

FIGURE 3: Positioning models / traces

The current level of abstraction leads to the use of a family of models and to obtaining a type of trace (Figure 3). Whether these traces are qualitative (formal) or quantitative (numerical), that the solution is analytical, or obtained by simulation, using a model is essential. Of its level of abstraction will depend its capacity to handle complex objects. In the field of predictive probabilistic assessment of dependability, the emergence of software in industrial control

systems has promoted the application of formal methods during software design and the use of verification (model checking). Some approaches consist in being interested only in the sequences of events leading to feared states for setting up defence barriers. Combinatorial approaches can be synthesized by fault trees, charts or reliability graphs... Their extensions (cf. temporal) gave birth to such dynamic fault trees [4] that incorporate some time aspects, but do not any more allow generating simply the structure function. Markovian approaches, despite the strong assumption on Aging, remain heavily used both in the case of hardware and software components [5]. Modelling in the form of automaton and associated languages, Petri nets, networks of queues, the stochastic process algebra... can easily handle large-scale models (adapted to the modelling of complex systems). These compact models can be combined with methods for reducing the size of the state space.

It is regrettable, however, in all these approaches the little concern to consider the interactions between software and hardware.

1.2 Joint assessment of software / hardware

Within the control, the design of the set software / hardware requires several levels of granularity and abstraction. In the case of Microsystems, from the TLM model (Transactional Level Modelling) are developed by descending granularity the PV for software development, then the PV (Programmer View) + T (Timing) directing architectural choices and the estimation of the performances. On the architectural side, the hardware synthesis will express itself through the RTL model (different shape / PV). Each model is capable of generating a record for its validation. Functional properties can be verified for each level of abstraction, by formal method, or by simulation (speed dependent on the level of granularity). The fully formal comparison of traces of different levels of abstraction, namely TLM (PV, PV + T) and RTL, becomes possible from traces RTL from synchronous deterministic automaton, whose performance product (via a translator) a trace in the form PV + T. However, the analysis of the robustness of functional and non-functional properties still resorts to the techniques of faults injection. Our proposal is to use the RTL model and the generated code to, in parallel with the phase of functional validation, validate the set software / hardware as qualitative and quantitative dysfunctional aspects (FIGURE 4).

FIGURE 4: Positioning the reliability evaluation in the design cycle of hardware / software

The verification of properties, qualitative and quantitative, dysfunctional for the set software / hardware will be based on the use of formal models and an analytical solution.

2. IFD METHODOLOGY

The modelling phase is done in 2 steps: a high-level model for building a finite state automaton. Dysfunctional sequences leading to systemic failures are derived from languages generated from the finite state automaton. These sequences are the basis of qualitative analysis: sequence length, location of weak points ... To overcome the inevitable combinatorial explosion of state space, a variant of BDD (z-BED) is gradually constructed from the final states of high-level model. From the system fail or minimal cuts sequences obtained, a multiphase Markovian approach allows to calculate the probabilities of errors of each hardware resource, allows quantifying the probabilities of occurrence of each of the corresponding lists for various operation modes of the system (productive, unproductive, dangerous ...). The process is described FIGURE 5.

FIGURE 5: Assessment process

2.1 The Information Flow Model

We are mainly interested in two different events of the system:

- Dangerous incidents which could lead to accidents
- Non-dangerous spurious trips

We want to extract all scenarios leading to one of these two undesired events of the whole system. Especially interesting are single point of failures, common cause failures and failure propagation. These kinds of failures are often not obvious, so they can be easily forgotten in a direct attempt to create a fault tree. This problem will be solved by a hierarchical approach. We use a directed block diagram representing the information flow through the system for high level and finite state automata and rules for low level modelling. This information flow diagram (IFD) and the automata are generalized versions of the diagrams and automata presented in [7].

2.1.1 Information Flow Diagram

For the IFD, we use different kinds of blocks which represent different functional entities. We distinguish:

- WD-blocks for watch dogs
- SRC-blocks as sources of information
- DEC-blocks for logical decisions
- ST-blocks for all other functions (storage of information, transformation of information, self-tests...)

Blocks of the type WD are used specially for control units with a watch dog. They have one input and one output and can detect the absence of sensible information in order to react accordingly afterwards by forwarding default or special error values. SRC-blocks create the information which flows through the diagram. They represent the sensors in the system and have only one output. ST-blocks (standard blocks) are the most versatile blocks. They have one in- and one output, and they are used for all functional entities which cannot be represented by the other blocks, e.g. the storage or the transformation of information. The last types of blocks are DEC-blocks. They represent logical decision entities. They have several inputs and one

output, and describe the behaviour of multiple interconnected sources of information. They do not describe any physical entities. The components which make this decision have to be included by adding a ST-block. One block in the diagram, normally a ST- or DEC-block, can be marked as final block. This block has no output and is used to generate the failure scenarios which will be described in Section 2.4. An example of an IFD for the example presented in [6] is shown in Figure 6. There are blocks for the different modules of the system, and some extra decision blocks. The information flows from the source blocks to the final block in one general time step t . In the source blocks, the sensors create the information which will flow through our diagram. This information proceeds to the successive blocks where it is processed and proceeded further. The exchange between the blocks always works faultless. While processing the data in the blocks, faults can occur or be detected. This means, that the state of signal can change within a block.

FIGURE 6: The IFD for an emergency stop system

We distinguish three different erroneous states for the signals:

- A non-existent failure has been detected. (Safe failure state S)
- An existent failure has not been detected. (Dangerous failure state D)
- The signal is lost. (Inhabitant failure state I).

2.1.2 Finite Automatons

The theory of finite state automata (with finite number of states) was mainly developed with the theory of the languages. These models mean specifying sets of states and transitions between these states [7]. Languages and automata allow to treat mathematically the problems related to Discrete Events Systems (SED), mainly from a logical point of view (qualitative analysis).

Each SED has a set of events associated with it. This set may be seen as an alphabet of a language and sequences of events are words (also called chains) of this language. An automaton is then a device that generates a language by manipulating the alphabet (the events).

After modelling the information flow it is necessary to specify the state changes of the information. For non-DEC-blocks finite, acyclic state automata are used to represent a mapping function. A deterministic finite automaton [7] is a quintuple $(S, \Sigma, \delta, x_0, F)$ with:

- A finite set of states S
- An input alphabet Σ
- A transition function $\delta : S \times \Sigma \rightarrow S$
- An initial state $x_0 \in S$
- A set of final states $F \subset S$

For our example, we have one predefined initial state x_0 and three predefined final states x_S , x_D , and x_I which represent the three faulty states of information. The used alphabet in these automata is symbols which represent different kind of failures. They are denoted as follows:

- input(i) with $i \in \{S, D, I\}$ (For ST- and WD-blocks)
- d(x, y) with x as a hardware resource and $y \in \{0, S, D, I\}$
- bf(e) with e as bit flip resource
- tf(f) with f as testing resource
- The empty word ϵ

The advantage of the automata for our purpose is that they are able to include different kinds of failures and several failure modes. Furthermore they are very intuitive and can be deduced quite easily for small subsystems.

An example, the automaton for the block Store1, is shown in FIGURE 7. While storing the results, different failures can occur. At first, the memory mem can be damaged physically, so that there are several bits which are locked to one (S), locked to zero (S) or the memory is not available at all (I). There is also the possibility of a bit flip in the memory, changing the value of the stored data. input(i) is used for the fault propagation of the predecessor block. It only occurs at the initial state x_0 . The possible values for i depend on the type of the current block. In ST-blocks, i can only be S or D as ST-blocks aren't capable of handling lost information. By contrast, WD-blocks only contain input(I). For hardware failures, the symbol $d(x, y)$ is used. y indicates the state of the hardware resource x: working correctly (0), non-dangerous failure (S), potential dangerous failure (D) or no output (I). $bf(e)$ represents environmental errors like bit flips of a resource e. $ft(f)$ indicates that a testing resource f has not detected an error. To simplify the calculation, we assume that the state of hardware, bit flips or fault test-resource is always the same in one time step. Note that all three final states are not always needed. Only final states which can be detected by the inits of the successor blocks have an influence on the result and are necessary. This will be explained in more detail in the next subsection.

FIGURE 7: The Automaton for the block Store1

The automata define a language which describes all scenarios leading to the different failures of the block. We can distinguish three different sub languages for non-dangerous failures, potential dangerous failures and failures with no output. These languages can be extracted and are saved in the three lists $L_S(B)$, $L_D(B)$, and $L_I(B)$ for the block B. The automaton in FIGURE 7 defines the following languages:

- $L_S(\text{Store1}) = \{d(\text{mem}, S); \text{input}(S)d(\text{mem}, 0); \text{input}(D)d(\text{mem}, 0)bf(m)\}$
- $L_D(\text{Store1}) = \{d(\text{mem}, D); \text{input}(D)d(\text{mem}, 0); \text{input}(S)d(\text{mem}, 0)bf(m)\}$
- $L_I(\text{Store1}) = \{d(\text{mem}, I)\}$

2.1.3 Rules of DEC-blocs

Decision blocks use another low level model to describe their behaviour. For this purpose, Boolean rules are introduced. These rules use the state of the signals (either S, I or D) of each input. There are three rules, they represent the lists L_S , L_D and L_I . If we take a look at the final block of the IFD shown in 7, the following rules are chosen:

- $S : V_1 = S \vee V_2 = S$
- $D : V_1 = D \wedge V_2 = D$
- $I : \text{false}$

So, the final block will create a spurious trip if at least one of the two valves will create one. A dangerous failure will only occur if both valves will fail dangerously. As we are searching a complete list, we have to define how to extract it. Disjunctions in the rules will be handled by unifying two lists, conjunctions will be handled by set products. For the given example we can conclude:

- $L_S(\text{Safe}) = L_S(V_1) \cup L_S(V_2)$
- $L_D(\text{Safe}) = L_D(V_1) \times L_D(V_2)$
- $L_I(\text{Safe}) = \{\}$

2.1.4 Generation of the global lists

The main interest is to generate all scenarios for dangerous failures and spurious trips of a final block. They will be stored in the lists L_D and L_S . To get these lists, the lists $L_D(B_f)$ and $L_S(B_f)$ of the final block B_f are created in order to connect them with the local lists of the other blocks. It's necessary to distinguish two cases: DEC-blocks and non-DEC-blocks. For DEC-blocks, the

method presented in the previous subsection to connect the blocks is used. For non-DEC-blocks, all $\text{init}(i)$ in the list are substituted recursively. The sequence after an $\text{input}(i)$ is combined with all sequences of a local list $L_i(B)$ of the previous block B by a set product. To illustrate this, the three following lists are used:

- $L_S(Bf) = \{\text{input}(S)d(x, S)d(y, D); \text{input}(D)d(y, 0)\}$
- $L_S(B) = \{\text{input}(S)d(v, 0); \text{input}(D)d(v, S)d(w, D)\}$
- $L_D(B) = \{\text{input}(D)d(v, D); \text{input}(S)d(w, D)\}$

If $\text{input}(S)$ and $\text{input}(D)$ are substituted with $L_S(B)$ and $L_D(B)$, we obtain:

$$L_S(Bf) = \{\text{input}(S)d(v, 0)d(x, S)d(y, D); \text{input}(D)d(v, S)d(w, D)d(x, S)d(y, D); \text{input}(D)d(v, D)d(y, 0); \text{input}(S)d(w, D)d(y, 0)\}$$

2.2 Decision Diagrams

It is quite obvious that using this method directly will lead to an exponential growth of the list. This is a severe problem as it will limit the usability of the proposed model. Therefore the size of the created list has to be reduced. In order to reach this aim the technique of Binary Decision Diagrams [8] (BDDs) are used to control the combinatorial explosion. Multiple variations of BDDs exist. In our case, two different approaches will be combined: Zero Suppressed BDDs (z-BDD) [9] and Binary Expression Diagrams (BED) [10].

z-BDDs are a modification of the widely used BDDs and based on binary decision tree (BDT). A BDT is a tree with:

- a finite set of Boolean variables V with a given order
- a finite set of non terminal nodes N_{NT} , containing one variable $v \in V$ as attribute
- two terminal nodes N_1 (One-Node) and N_0 (Zero-Node)

BEDs [11], originally used for verifying circuit implementations, are an expansion of BDDs. In general, BEDs are a BDD enriched with different kinds of operator nodes for Boolean operations. For this paper just two of them will be used: OR-Nodes and AND-Nodes. It is possible to transform the BED into a BDD with the same complexity than creating the BDD directly. As BEDs are originally developed for standard BDDs an alternation of the original transformation rules is necessary, though, leading to zero-suppressed BEDs (ZBED).

2.2.1 Boolean interpretation of lists

Decision Diagrams are used for representing Boolean expressions. To use them, it is necessary to transform the local lists of the blocks into Boolean expressions. Bit flip and fault test-resources can have two states, so it is no problem to see them as simple Boolean variables. Hardware resources have four states, though. For these three different Boolean variables (For example x_S , x_D , and x_0) can be defined for every resource x . Note that three variables are enough, a variable x_i is not necessary. As x can only be in one state at one time, the value of x_i can be deduced from the values of the other three variables. Input-expression won't be changed at first. They will be replaced during the construction process of the z-BDD.

Sequences can be interpreted as conjunctions of their comprised resources. For hardware resources, all three variables are set according to the value. For example, $d(x, S)bf(e)$ is interpreted as $\bar{x}_0 \wedge x_S \wedge \bar{x}_D \wedge e$. A whole list is seen as a disjunction of all sequences. $\{d(x, S)bf(e); d(x, I)\}$ is interpreted as $(\bar{x}_0 \wedge x_S \wedge \bar{x}_D \wedge e) \vee (\bar{x}_0 \wedge \bar{x}_S \wedge \bar{x}_D)$.

2.2.2 DDs for simple serial Systems

In this subsection, two assumptions are made:

- The IFD does not contain any DEC-blocks
- Boolean variables do occur only in the lists of one block

With these assumptions, it is very easy to use the structuring of the IFD in order to create a z-BDD very efficiently. It is possible to create a z-BDD for the final block based on local(s) lists. This z-BDD can include three extra leafs (Input(S), Input(D) and Input(S) $\wedge$ Input(D)) in case of ST-blocks, one extra leaf for WD-blocks(Input(I)). These leafs can be substituted using the following method:

An Input(x)-leaf is replaced by the root of the z-BDD for the list L_x of the predecessor block. If there are multiple Input-leafs, the sub z-BDDs will also share equivalent nodes. After substituting the Input nodes of one block, new Input-nodes can occur if the predecessor block is not a SRC-block. So the substitution has to be repeated recursively until it arrives at the SRC-block. An example is shown in FIGURE 8, in which a z-BDD for only one block is shown on the

left side and the global z-BDD after replacing recursively all Input-nodes on the right side. The advantage of the method of modular constructing the z-BDD is that globally it will only grow linear in size compared to the number of blocks. After every block there are at most three non trivial leafs. Locally, it is still possible that a sub z-BDD grows exponentially, though. But this should not cause any problems as the local z-BDDs are normally quite small and they can be created independently from the z-BDDs of the other blocks. Overall this approach can reduce the complexity enormously.

2.2.3 DDs for general IFDs

In section 4.2 we assumed that no component will occur in more than one block. Unluckily, this assumption is quite unrealistic for many systems, so it is necessary to extend the presented algorithm in order to allow multiple occurrences of components. To be able to include such cases, the z-BDD is extended with several attributes:

- An array containing all local lists. Later, modified lists can be added.
- A hash table mapping components to blocks in which they appear.

The construction of the z-BDD begins like in the simple case. The decomposition of the local lists starts at the final block and ends with the source blocks. The only difference is that before every decomposition of a variable the hash table is checked to find other blocks which use it, too. If there are other blocks with the same variable, the decomposition is also applied to copies of the local lists of these blocks. All descendants of the current node will use these modified lists instead the original ones as soon as the blocks using these lists are reached. In order to achieve this, every node stores an array with pointers to the lists which have to be used in the future. Children inherit these lists from their parents and alter them only if their variable will also occur in other blocks of the IFD.

As DEC-blocks are defined by Boolean expressions, it would be possible to use a normal decomposition to create the z-BDD. The only problem is that the expressions use the lists of several predecessor blocks and that these lists can be combined with set products, too. This leads too much more possible permutations of *init*-values than for serial diagrams. The solution to this problem is to use ZBEDs as they can represent the rules directly. In the last case, the reduction of the ZBED to a ZBDD can be continued recursively by replacing all temporal leafs with the ZBED of the corresponding list and applying the reduction rules to the extended diagram until the source nodes are reached.

FIGURE 8: A local z-BDD for one block (left) and the global z-BDD for three blocks (right)

2.2.4 Quantitative evaluation

Estimating metrics such as PFD and PFS requires 3 phases:

- Modelling of the evolution of environmental stresses imposed on the architecture (temperature, pressure, radiation ...) and maintenance of components. The operational impact of environmental parameters on components are available in many fields such as aeronautics [FIDES], automotive, glass and petroleum industries ... The consideration of human constraints (maintenance operations ...) is not considered in the cases presented here.
- Stochastic modelling of the evolution of probabilities of hardware errors, environmental faults and failure tests.
- Generation of characteristics lists of sequences (combinations of errors) leading to the states sought. For example, a list leading in a necessary and sufficient manner to a lack of reaction of the safety function under the hypothesis of a presence of demand at the time of acquisition, another leading to a spurious trip of the reaction actions of the dedicated safety function (reaction in the absence of demand).

After the evaluation of environmental stresses, the conditional probabilities of Markov processes associated with the evolution of the state of hardware resources, and internal and external events are updated. The probabilities of errors for each hardware resource are updated by iteration of Markovian process. Finally we use these probabilities of error evaluated previously to calculate the probabilities of occurrences for each list at time $t=kT$. The probability of occurrence of each list is given by the sum of probabilities of each combination of simultaneous errors it contains. By piecewise integration up to $t=T_{life}$, we will obtain the probability sought.

3. APPLICATIONS

The formalism adopted is hierarchical and applicable to many problems. We illustrate here the applications in the case of systems (SIS) and Microsystems (microprocessor).

The design of SIS must refer to the IEC 61508 standard offering an approach to the quantitative evaluation of reliability performance indicators in the case of electronic and programmable safety functions. This standard has undergone many variations in the fields of industry (for Machine IEC 62061, IEC 61511 for the process), nuclear (IEC 61513), railway (IEC 62278) ... These referentials for functional safety, assimilate the quality of service of a dedicated safety function with a level of integrity, known as SIL (Safety Integrity Level) based on 4 conditions:

- Quality of design and verification of architecture, by using structured methods for the demonstration of basic sub functional properties and their conformity with the specifications.
- Implementation of maintenance procedures adapted to the operational maintains.
- PFDavg (Probability of Failure on Demand) complying the specifications.
- Respect of the architectural specifications, allowing the guaranty of fault tolerance properties (diagnosis with requirements on coverage, redundancy ...).

If in the first sample application, the study will involve the assessment of quantitative metrics (PFD, PFS), the second shows an evaluation of the impact of a fault tolerance mechanism integrated within a stack processor.

3.1 Evaluation of a Safety Instrumented Function on an industrial system

The introduction of a safety dedicated function in an industrial installation is intended to make possible the detection of a situation considered potentially dangerous and to avoid the occurrence of an accident. The principle of avoidance is based initially on the detection of an abnormal situation, precursor of the accident, and on the activation, in this case, of a set of actions. These actions have to force the return of the installation in a safe mode, defined as a state where the installation must be able to continue its core missions without causing catastrophic consequences for its users and its environment. The spectrum of use of such functions is extremely broad: energy and nuclear, industries, infrastructures and transport systems (road, rail, air). These include, for example, the emergency stop function of nuclear reactors, the airbag or emergency brake functions in transport applications (automotive, rail) ...

The demonstration of the quality of service of a safety dedicated function, ie of all its features justify its capacity, during its period of use, to satisfy both expressed needs (ability to avoid accidents) and implicit needs (non-disruption of mission system in the absence of dangerous situation) ISO84102 [12] is therefore essential. It must enable the operator to ensure that the constraints that may induce the introduction of a dedicated function on the installation, in terms of loss of production or service, are acceptable and that a satisfactory reduction in the risk level of the facility is reached. The introduction of programmable electronic components (PE), for the implementation of safety dedicated functions, highlights the inadequacy of conventional approaches for the assessment and identification of functional performance indicators, to judge the consequences in terms of risk level and economic cost caused by such an introduction into

an existing installation. The verification of compliance of a software architecture is needed to avoid systematic design errors, which can lead to dangerous behaviour upon system start, but not sufficient to demonstrate its reliability or the maintenance of service quality of the studied system towards alterations of its hardware and information resources. This verification is generally not sufficient to demonstrate avoidance of systemic failure modes during its operational life. These failures are defined by the IEC61508 [2] as the cessation of ability to deliver a correct service linked to a certain cause, which can be eliminated only by changing the design, the manufacturing process, the manual, the documentation or other appropriate factors. The constraints of use of hardware resources can indeed generate this type of failure, despite the absence of design errors. Similarly it is not sufficient to demonstrate that a deviation in the inputs laws or error in hardware resources ensuring the fulfilment of a programmed function can cause the endangerment of the plant. We therefore assume, for the rest of our study, a sufficient level of knowledge of our functional and hardware architecture, and a verification of these properties. These preliminary hypotheses should allow us to be able to describe the hardware and software architecture, in a sequential and reconfigurable process of information processing for each of which a finite and clearly identifiable set (but not necessarily disjoint) of hardware resources are used.

Two performance indicators, for judging the quality of service of a system at sampling times, will be assessed:

- The probability of failure on demand at time t , $PFD(t)$, corresponds to the probability of non reaction of the safety function at time $t + T_r$ if demand is present at time t .
- The probability of spurious trip at time t , $PFS(t)$, corresponds to the probability of a spurious activation of the reaction system at time $t + T_r$ in the absence of demand at time t .

3.1.1 Presentation of the system

To illustrate the presented model, the emergency stop system of a chemical reactor (Figure 1) will be used which is described in this section. This system should stop the reaction if the temperature in the reactor is getting too high by stopping the inflow of the chemicals. The sensors S1 and S2 measure the current temperature of the chemicals in the tank and transport their results to the controller. The controller reads this result via his inputs In1 and In2 and will store these values for synchronization in its memory. (St1, St2). To avoid a loss of information, the watchdogs Wd1 and Wd2 supervise the inputs. If an input is lost or arrives too late, the watchdog will pass a default value to the Voter. Afterwards, the Voter decides, if there is a dangerous situation. If after the voting process the control unit CU decides to shut down the system, this information is proceeded to the output modules Out1 and Out2 and passed to the motors M1 and M2 which get the order to close the valves V1 and V2. If at least one of these valves is closed, the shutdown was successful. For this system, there are two possible kinds of failures in general. Either the emergency system is not available (dangerous failure), or it shuts down the system in a safe state leading to an unnecessary unavailability of the whole reactor (spurious shutdown). A failure of the whole emergency system can be caused by several different failures of its components.

FIGURE 9: General HW-Architecture of studied example

3.1.2 Design of the model

The model is built by successive refinements [7]. The first step considers the basic functional entities along the flow of F-information from the source block SRC1, to the terminal blocks of class TF. The second step allows integrating the test procedures and control of the update of

the data. Sources blocks SRC2 symbolizing the existence of combinations of hardware faults detected by means of an online test, are then added and connected to an entity class CT, and control procedures. The model is then supplemented by description of the flow of D-information from the entities class ST and CT: added class blocks TF (signal processing), SB (storage) and eventually IP of type 3 (merger of diagnosis). Finally, IP blocks, representing the properties of recovery and degraded modes, are inserted on the flow of F-information. Thus the model of high-level FIGURE 10:

FIGURE 10: High-Level model of the SIF

Legend :

- 01 - Acquisition of temperature measurement from the sensor through the distributed input module
- 02 - Transmission of the temperature measurement to the input module of the microprocessor
- 03 - Storing the value in the input register of the microprocessor
- 04 - Check update of the register 1 (way 1)
- 05 - Check update of the register 2 (way 2)
- 06 - Assigning a backup value in absence of input register update
- 07 - Vote function 1oo2
- 08 - Request to close the valve if close test negative
- 09 - Transmission of the control order from microprocessor to actuator
- 10 - Valve command
- 11 - Valve closing solicitation (Periodic test)
- 12 - Acquisition of the tested valve's position sensor
- 13 - Transmission of the measurement to the microprocessor

3.1.3 Qualitative and quantitative assessment

From the constructed finite-state automaton, if the system is of reasonable size, the lists (ordered sequences of events leading to system states) are constructed and simplified. In the case of large-sized models, z-BDD are built gradually from the final states (states of the overall system), and allow the obtaining of minimal cuts (non-ordered sequences of events). This second approach, certainly less accurate, allows keeping an analytical approach as to the quantitative results. The process of generation of lists is however maintained for the qualitative analysis of systemic failures.

The 2nd level of representation describes the functional and dysfunctional behaviour of each sub-functional entity. The finite state automata support languages expressing sequences of events leading to the modes of operation: reaction is activated in absence of dangerous situation (PFS) and non-activation or deactivation of the reaction in presence of a dangerous situation (PFD).

	RBD/Markov (conventional method)	Stochastic Petri Net Monte Carlo Simulation	IFD Method
--	-------------------------------------	--	------------

PFD _{avg}	4,8679.10 ⁻² /h	2,3462.10 ⁻⁴ /h	3,586.10 ⁻⁴ /h
PFS _{avg}	2,734.10 ⁻³ /h	4,2535.10 ⁻² /h	4,8679.10 ⁻² /h

TABLE 1: Quantitative results

The Table 1 tends to show an underestimation of the PFS, and an overestimation of the PFD, by conventional approaches. In the case of the PFD, this leads to the achievement of a solution whose cost of ownership is more important. As for PFS underestimated, this will result in exploitation in a stop rate of the installation, and therefore non-productivity, more important. The results obtained analytically by the IFD approach are very close to those obtained by stochastic simulation.

The differences are changing depending on the level of complexity of the structure.

Conventional methods are less suitable.

The performance indicators of a dedicated security (PFD_{avg} and PFS_{avg}) are related to the concept of residual risk by [13]:

$$R_{res} = \frac{1}{T_{life}} \sum_{k=0..N} f T PFD(kT) \cdot I_{dem} + \frac{1}{T_{life}} \sum_{k=0..N} (1 - f \cdot T) PFS(kT) \cdot I_{abs} =$$

$$f \cdot PFD_{avg} \cdot I_{dem} + \left(\frac{T_{life}}{T} - f \right) \cdot PFS_{avg} \cdot I_{abs}$$

with f , the average probability of occurrence of demand

I_{dem} : the impact of the feared accident

I_{abs} : the impact of the spurious tripping

If the dangerous situation detected by the safety function (presence of demand) leads to certain accident: f will be equal to the average frequency of occurrence of an accident before the addition of a dedicated security function.

Once the relationship between residual risk and the metric PFD_{avg} and PFS_{avg} defined, it becomes possible to judge the quality of service of a safety function by defining a method for estimating the economic loss caused by his tripping. We denote this metric:

$$C_{ind} = \sum_{k=1..E\left(\frac{T_{life}}{T}\right)} \{ [1 - p(\text{demand}, kT)] PFS(kT) \cdot c^*(kT) \} = \left(\frac{T_{life}}{T} - f \right) \cdot PFS_{avg} \cdot c^*$$

c^* : induced average cost (in terms of economic loss per unit of time T) associated to an unavailability

Given a level of acceptable residual risk and a level of maximum economic loss accepted, it is possible to establish unequivocally the threshold requirements of both performance indicators PFD_{avg} and PFS_{avg} to ensure compliance with the stated objectives under the mentioned assumptions and assuming that the trips did not result in an accident.

3.2 Evaluation of a microprocessor

The method, hierarchical and modular, can thereby be used for the evaluation of processor architecture (HW and SW) which may include fault tolerant mechanisms. Fault tolerant and reconfiguration mechanisms can also be implanted in the application and system level, we can thus imagine to estimate globally the detection / recovery mechanisms. The aspects of propagation, inhibition, common cause failures... becoming essential on the scale of a system of system (including communication networks). Mechatronic applications, combining analog and digital multi-technology components, can also be accepted by this method.

It is thus possible to assess the fault tolerance mechanisms impact on the reliability of a microprocessor [15]. Each basic instruction is evaluated individually according the described method. Then, either the instruction set is purely sequential (direct calculation of probabilities), or it is not the case, or detection / fault tolerance mechanisms are also present at this level (reuse of the described method in a higher hierarchical level). Below is an example of modelling of a basic instruction on a stack microprocessor.

3.2.1 Instruction processing

A preliminary RTL modelling is done for DUP instructions in order to make easy their information flow modelling [14]. Before explaining these instructions, we should mention that the processor has two stacks. One stack is used for the data treatment called data stack (DS). The top-of-stack (TOS) and the next-of-stack (NOS) correspond respectively to the first and the second element of this stack. The second stack is used for the subroutine return addresses, interruption addresses and temporary data copies, and is called return stack (RS). The top-of-return-stack (TORS) corresponds to the first element of this stack. The stack buffers are managed in an external memory of the processor in order to have no restriction in the stack depth. They are addressed by internal pointers (data stack pointer DSP and return stack pointer RSP). Since some architecture features are explained, we can detail clearly the DUP instruction. The DUP instruction allows the duplication of the top-of-stack (TOS).

- $DSP \leftarrow DSP + 1$ 'incrementation of Data Stack Pointer to push a new element'
- 3rd DS Element (Memory value addressed by the new DSP) $\leftarrow$ NOS
'Next Of Stack becomes the 3rd Data Stack element'
- $NOS \leftarrow TOS$ 'duplication of the Top Of Stack value'

FIGURE 12: Architecture of the DUP function from a RTL model

Concerning the control logic for the execution of this instruction (bold lines in FIGURE 12):

- The data stack memory is in write enable mode;
- The control signals of the Mux_DSP are in position so that the (+1) input is selected;
- The control signals of the Mux_NOS are in position so that the connection from TOS is selected.

At the clock's rising edge, all the operations are done and DUP instruction is executed.

3.2.2 Modeling

The DUP instruction copies the top of stack (TOS) into the next of stack register (NOS) and pushes the NOS to the third element in the data stack memory (Mem_DS). This instruction needs to increments the data stack pointer (DSP) to allocate a new cell in Mem_DS. The low-level model consists of modelling the functional and dysfunctional behaviour of each sub-functional entity in the high level model by associating a finite state automaton [15]. The transition events in this automaton are classified and labelled between different states. The result of the application of the high-level model on the DUP instruction is illustrated in FIGURE 13.

FIGURE 13: High-level model of the DUP function

3.2.3 Assessment of fault-recovery mechanism

The case study is the bubble sorting of variables. From a table containing these variables, the program consists of reading the data, swapping in the case of disorder and keeping the table with new ordered values. First of all, the program initializes the variables in memory. We make a reset of the permutation flag. Every two successive data is compared. If a permutation is done, the flag is putted to 1. We make the same comparison until the end of variables and until the flag is null.

The protection technique consists of recovering the last dependable state on each error detected and not corrected by hardware. The dependable state is periodically obtained thanks to storage of stack pointers, program counter and top-of-stacks. We consider the example of the

bubble sorting program above, the recovery strategy is translated by the addition of the functions of backup and recovery that we model next. This strategy is based on recovering the last dependable state on each error detected and not corrected by hardware. The dependable state is periodically obtained thanks to storage of stack pointers, program counter and top-of-stacks before starting of each comparison cycle. In case of any failure event during the program execution, a restore of saved data is done. After this restoration, the program continues to run with reliable data. The faults tolerance method proposed in this work uses a set of features to save and restore periodically the sensitive elements in stack processor architecture (DSP, RSP, PC, TOS, NOS and TORS). As example, we can illustrate the backup and the restore of the Data Stack Pointer (DSP), given by the following features:

- <i>Save_DSP: Push_DSP</i>	- <i>Restore_DSP: DLIT</i>
@_Sauv_DSP	
DLIT @_Sauv_DSP	FETCH
STORE	Pop_DSP

Even for this very simple example, it is interesting to note in this type of study, a percentage increase of the probability of the system to remain in a productive mode of operation (contribution of mechanisms for fault tolerance), in compliance with the expectations of designers. However, the probability to remain in a functional state without faults is lower, due to the increased number of treatments to achieve. We can also quantify the probability to lead, for each treatment cycle, to the possible different states of the system: unproductive and even dangerous (failure(s) not detected). In the example above, the probability of dangerous failures were multiplied by 4 due to the introduction of a fault tolerance mechanism.

4. CONCLUSION

The proposed method presented here allows the combined software/hardware analysis of E/E/PE functions. It allows a better approach to modeling of common cause failures, shared resources... It is hierarchical and allows the evaluation of applications whose dependable structure varies (skipping functions, reconfiguration of architectures). This approach has many advantages toward current methods for the study of repairable and reconfigurable architectures:

- It clearly distinguishes the influence of architectural constraints, given by the lists of failed sequences, and the environmental constraints, taken into account dynamically by the interaction of levels of representation.
- It recommends the implicit and dynamic inclusion of environmental coupling problems, assessing for each period T, the stochastic processes modeling the evolution of probabilities of errors for each hardware resource.
- It considers due to the construction of lists of sequences, the problems of sharing resources (hardware and informational) in the architecture, and of test and reconfiguration constraints.
- It allows a strong distinction between the existence and the activation of error, to better apprehend the influence of simultaneous failure modes (and hence the common cause failure modes, common modes and latent modes).
- It takes into account the properties of fault tolerance, reconfiguration and repair intrinsic to the system of study.
- It addresses the problem of dependency in a different way of usual combinatorial approaches (beta, alpha, MLG), which the lack of representativeness of the phenomena of dependency tends to make certain current estimations strongly biased for architectures of increasing complexity.

The method has been applied for the assessment of safety instrumented systems (as defined in IEC 61508) and microsystems (stack processor). The metric considered (PFD and PFS) would not be restricted to SIS. They are used to quantify the ability of systems to produce (and thus influence the design in case of non-compliance with operational requirements), which should be characteristic of many systems, or industrial plants. In the microsystems case, the proposed method allows to evaluate the impact of features whose implementation is spread over the software and hardware, such as mechanisms for fault tolerance. Their contributions are not always obvious and the search for a compromise is needed based on rigorous methods (without dreaming exhaustive).

Consideration of scheduling problems (modeled by automata) will be our future work.

ACKNOWLEDGMENT

We want to thank Siemens, the Fondation CETIM, the French-Bavarian Center for cooperation of universities (BFHZ-CCUFB) and the Région Lorraine for their support of this work.

REFERENCES

- [1] Goble, WM., Brombacher A, Bukowski JV., "Using Stress-Strength Simulations to Characterize Common Cause", Probability Safety Assessment and Management, PSAM 04, Mosleh. A., Bari R. (ed), Springer, 1998/09, pp.399-448, ISBN:3540762620..
- [2] IEC 61508, 1999 Functional Safety of Electrical/ Electronic/Programmable Electronic Safety-Related Systems, Part 1-7. International Electrotechnical Committee.
- [3] Laprie JC., Sûreté de fonctionnement informatique : concepts, défis, directions, ACI Sécurité et Informatique - Toulouse, 2004.
- [4] J. Bechta Dugan, S.J. Bavuso, and M.A. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. IEEE Transactions on Reliability, 41:363–377, 1992.
- [5] Arlat J., Kanoun K., Laprie JC., 'Dependability modelling and evaluation of software fault-tolerant systems', IEEE Tr. On computers, vol.39, pp. 513-540, 1990.
- [6] K. Hamidi, O.Malassé, J.F. Aubry "Coupling of information-flow aggregation method and dynamical model for a more accurate evaluation of reliability", European Safety and Reliability Conference, ESREL, June 2005.
- [7] Arnold A., "Systèmes de transitions et sémantique des processus communicants", Masson, 1992.
- [8] Bryan RE., Chen YA., Verification of arithmetic functions with binary moment diagram, IEEE of Transaction on Computer, pp 677-691, 1986.
- [9] Minato, S. (1993). Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In 30th ACM/IEEE Design Automation Conference, pp. 272–277ACM/IEEE.
- [10] Andersen, H. and H. Hulgaard (1997). Boolean Expression Diagrams. In 12th Annual IEEE Symposium on Logic in Computer Science, pp. 88–111. IEEE.
- [11] Pock M., Belhadaoui H., Malassé O., Walter M., 'Efficient generation and representation of failure lists out of an information flux model for modeling safety critical systems', ESREL 2008 and 17th SRA-Europe Conference, Valencia (Espagne), 2008.
- [12] ISO 8402, "Management de la qualité et assurance de la qualité – Vocabulaire", International Standard Organisation, août 1995.
- [13] Hamidi K., Malassé O., Aubry J-F., 'Contribution to an amelioration of quantitative evaluation's model of reliability for safety-related functions', 2004 IEEE International Symposium on Industrial Electronics, IEEE-ISIE, Ajaccio (France), Mai 2004, Vol. 1, pp 115- 120, ISBN: 0-7803-8304-4.
- [14] A. Shaout and T. Eldos, "On the classification of computer architecture", International Journal of Science and Technology, vol. 14, Summer 2003.
- [15] Jallouli M., Belhadaoui H., Malassé O., Buchheit G., Aubry J-F., Medromi H., 'Evaluation of important reliability parameters using VHDL-RTL modelling and information flow approach', ESREL 2008 and 17th SRA-Europe Conference, Valencia (Espagne), September 2008.