MBSA Approaches Applied to Next Decade Digital System-On-a-Chip Components

Tiziano Fiorucci, Thomas Jacquet, Jean-Marc Daveau, Giorgio Di Natale, Emmanuel Arbaretier, Philippe Roche

MBSA Approaches Applied to Next Decade Digital System-On-a-Chip Components

Tiziano Fiorucci[1,2], Thomas Jacquet[3], Jean-Marc Daveau[1], Giorgio Di Natale[2], Emmanuel Arbaretier[3], Philippe Roche[1]

[1]STMicroelectronics, 850 Rue Jean Monnet 38926 Crolles Cedex, France

[2]Univ. Grenoble Alpes, CNRS, Grenoble INP*, TIMA, 38000 Grenoble, France

[3]Airbus Protect, 2 rue Marceau, 92130 Issy-Les-Moulineaux, France

***Abstract*—This paper proposes a first step towards the automation of the safety assurance level assessment using model-based methods applied to the domain of complex digital System-On-a-Chip (SoCs). Model-based safety and reliability engineering is widely deployed in critical systems area such as aeronautics but remain unused in the field of digital system design where Failure Mode Effect Defect Analysis (FMEDA) are performed manually or using fault injection. We describe here an automatic approach to build dysfunctional models of digital blocks suitable for use with existing safety engineering frameworks. The choice of SimfiaNeo (Airbus Protect) framework explores the possibility of modelling beyond classical electromechanical systems, exploring application of Model-Based Safety Analysis (MBSA) to digital systems. Experiments have been conducted on a simple serial $\leftrightarrow$ parallel adapter system, as a proof-of-concept laying the base for a more complete and extended application to complex digital systems. Such preliminary work is needed to ensure the viability and scalability of the approach when targeting heavily interconnected systems as such as complex SoCs.**

***Abstract*—Cet article propose une première approche vers l'automatisation de l'évaluation du niveau d'assurance de la sûreté fonctionnelle à l'aide de méthodes basées sur les modèles (Model-Based Safety Assessment ou MBSA) appliquées au domaine des systèmes digitaux complexes de type système sur puce (SoC). L'ingénierie de la sécurité et de la fiabilité basée sur les modèles est largement utilisée dans le domaine des systèmes critiques tels que l'aéronautique, mais elle reste marginale dans le domaine de la conception des systèmes numériques où l'analyse des modes de défaillance, des effets et des défauts (FMEDA) est effectuée manuellement ou par injection de fautes. Nous décrivons ici une approche automatique visant à construire des modèles dysfonctionnels de blocs digitaux permettant leur utilisation dans les flots d'ingénierie de sûreté fonctionnelle existants. Le choix de SimfiaNeo (Airbus Protect) permet d'explorer l'utilisation des méthodes MBSA au-delà des systèmes électromécaniques classiques, en appliquant l'ingénierie de la sûreté fonctionnelle basée sur les modèles (MBSA) aux systèmes digitaux complexes. Une application à un système simple d'adaptation série vers parallèle, est présentée comme preuve de concept en vue d'une application à des systèmes plus complexes. Ces travaux préliminaires sont nécessaires pour garantir la viabilité et l'évolutivité de l'approche qui vise une application à des systèmes fortement inter-communicants de type SoCs.**

I. Introduction

In the context of new applications for autonomous mobility, digital components a required to reach high levels of functional safety performances. This level of assurance is necessary to supply safely the computational power and advanced processing required by those applications. It is therefore necessary for digital SoCs safety engineers to be able to demonstrate thru advanced provable methods the achieved reliability of their system and counter measures.

Similarly to complex electomecanical systems, it is difficult to predict the failure modes of a complex SoC which exhibits an almost infinite state space (in the order of 2^n, n is the number of sequential elements, reaching ten's of thousands easily) and distributed-systems characteristics: numerous independents sub-systems operating and communicating asynchronously.

Digital systems are subject to two kind of errors, *permanent* which are created by destructive or aging effects, and *transient* [1] created by particle impacts such as thermal neutrons at ground level or solar wind in low and high earth orbits [2] [3]. Permanent effects shows in the form of a permanent stuck to an electrical value ('0' or '1' logic value) and can occurs on any digital element (combinational logic, i.e. *logic gate* or sequential element, i.e. *flip-flops*). So do transient faults, also called *soft errors*, but with different, non-permanent, effects on logic or sequential elements. Transient faults on logic gates are called *Single Event Transient* (SET) [4] and are particularly dangerous on clock trees and reset trees (which distributes the clock and reset signals through the chip using trees of *buffers*) of SoCs as their effect, that has the form of a glitch, is to reset or desynchronize the sequencing of a sub-part of the system. Transient faults on sequential elements (memories or flip-flops) only invert the value of the element which will retain the faulty value until overwritten by a new value. They are called *Single Event Upset* (SEU) and are the main cause of safety goals violations in digital SoCs [5].

However, digital system exhibit a natural resistance to soft errors and most of them have no functional effect while a small proportion of them ($\approx 10\%$ of them in a standard 5-stages processor [6], [7]) will lead to system execution failure. FMEDA analysis [8], targeting goals such as ISO26262 automotive safety norm [9] certification will consist in quantifying those failure modes, proving the effectiveness of counter measures and absence of safety goals violation. An effective solution consists in submitting the system to faults, by simulation or under radiation beam, therefore stressing it and provoking intentionally dysfunctional behaviors. Those 'out of trajectory' behaviors can then be recorded, analysed and used for FMEDA analysis in the certification process. However, both methods are costly both in term of engineering setup needed and cost: fault injection of a full SoC requires a complex setup, test suite and costly hardware emulator while radiation test requires an acquisition system, a test setup and access to costly and constrained radiation facilities, Both have the disadvantage to require, the full SoC gate netlist (fault injection [10]) or silicon (irradiation [11]). Also, both methods can be classified as experimental as it is a verification 'by observation' of the resilience of the system to faults. No proof, except statistical confidence is made on the extracted faults metrics.

In this work, we aim to assess the capability of Model-Based Safety Assessment methods to build the dysfunctional model of a digital SoC from its subsystems and perform the currently hand-made FMEDA of the full system automatically. We expect the methods to be able to quantify globally the system safety metrics more accurately than with hand-made spreadsheets which only basically multiply probabilities. Automatic failure analysis such as fault trees extraction, fault sequences leading to unwanted events are also expected to be of great help during the certification process. The problem to solve is then to extract and build the required dysfunctional models of the different subsystems of the SoC and to properly expose the failure modes in the constructed models to be able

*Institut National Polytechnique Grenoble Alpes

to use existing model composition frameworks.

The document is organized as follow: we first present the system used as example and how fault injection is used to expose dysfunctional behaviors and extract a model. The chosen approach is then detailed reminding generic principles before explaining specific mechanisms put in place to model digital system. Finally, the document details fault injection campaign post-processing methods and obtained results. We compare composition results with fault injection performed on the full system used as a reference.

II. State-of-the-Art

A. Probabilistic Methods in Digital Systems Safety

Probabilistic methods [12] [13] have been developed to estimate propagation and masking rates of errors in gate netlists. Such approaches, restricted to combinational logic provide an helper to estimate certain metrics (λ_{spf}, i.e. *Dangerous Undetected* by a safety mechanism faults [14]) required in ISO26262, but are far from being able to provide metrics even at the sequential block level. Likewise, industrial formal proof tools [15] [16] are able to compute such metrics by using formal methods.

Methods like FIDES [17] [18] targets Commercial Off-the-Shelf (COTS) based Electronic Control Unit (ECU), with components failure rates extracted from available reliability databases. It takes into account systematic or aging failures but not transient effects such as soft-errors.

B. Formal Methods in Digital Systems Safety

Formal methods [19], [20] are mostly used on unitary blocks or functionalities to prove assertions (i.e. properties) expressed in linear [21] or branching [22] timing logic. When applied to safety, it comes to proving absence of safety goals violations that are expressed as assertions on outputs in the presence of faults. Tools like [15] [16] are able to compute, given a nelist of logic gates and flip-flops and an initial state, the cone of influence of flip-flops or gates and whether a fault in such elements can propagate to a given output. Such structural analysis can perform *Out-of-Cone-of-Influence* (*COI*) fault analysis allowing to classify a fault as *safe* when it cannot reach a given output. Activation analysis determine whether a fault injected on a specific node can be activated. Propagatability analysis determine if an activated fault in a COI can propagate to a strobed output and detection analysis determines if a fault will (always) be propagated and detected at the checker output. Such analysis can reveal what logic is covered by a safety mechanism or not. However, no formal methods is able to address such safety properties at SoC level.

C. Altarica

Functional safety objective is to identify the most probable failure combinations leading to a feared event. Model-Based Safety Analysis performs safety analysis by building dysfunctional models for each block of the considered system and using formal methods to combine and extract failure modes at the system level [23]. MBSA introduces the use of high level modeling languages dedicated to functional safety analysis [24] [25] [26]. It allows extending classical methods such as FMEA or fault trees. These languages help capture system dynamics and how failures propagate inside it. Moreover, models support structural modeling allowing identifying and locating induced effects of a failure inside the architecture.

Altarica Dataflow (Fig. 1) is an event-driven asynchronous language that implements discrete variables with a finite number of values, leading to a finite number of combinations of state values and propagated flows, allowing theoretically to cover the entire system state space. AltaRica Dataflow is at the core of several Reliability, Availability, Maintainability and Safety (RAMS) environments: Cecilia OCAS (Dassault Aviation), SimfiaNeo (Airbus Protect), and Safety Designer (Dassault Systèmes)

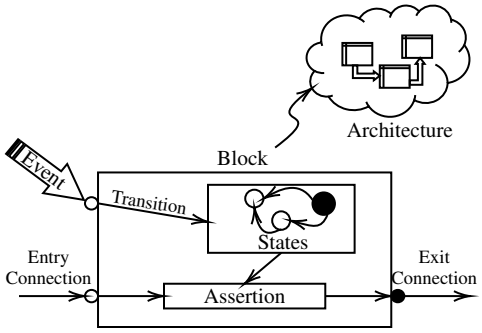

Fig. 1: Altarica Dataflow Model

- *Variables*: AltaRica variables are discrete and represents an enumerated finite set of values called its *domain*. Variable definition inside its domain is free. The variable can represent for example functional modes, dysfunctional status, message types ....
  Inside MBSA models, state variables are generally used to represent dysfunctional status with a default value as nominal behavior and a value for each degraded mode reached from any failure mode. Flow variables are generally to describe the type of data exchanged between components. This type can represent a functional value (e.g. instruction value) or a dysfunctional value (e.g. message status). It depends of the model level of detail. As flows are used to propagate failures, they can be described either by sending a status or a faulty value.
  Transitions: Transitions describe possible states changing values. Transitions are guarded by a condition allowing the transition to become fireable when true. A transition is associated with a triggering event and is fired when the event is triggered and the guard is true. In MBSA modeling, triggering events are used to represent failure modes. AltaRica allows to assign a probability law to an event, modeling the behavior of random failures or deterministic actions. The transition completion describes the effect of the failure mode on the component state. Guards can be enriched to restrict to describe conditional failures. For example, in a cold redundancy, some failures can't happen when the component is off.
- *Assertions*: Assertion is the mechanism used to set outputs values of a node. Output values are a function of input values and internal state values. Assertion can be interpreted as a logical function describing a truth table assigning outputs according to each combination of inputs and internal state values. Combinations are described through Boolean expressions and imperative programming constructs such as *if-then-else* or *case*.
  Assertions are used to propagate failures from a faulty component to other blocks. Fault injected on the internal state is propagated to its output and then to others blocks. Depending of the granularity level of the model, assertions are manipulating either functional values or states.

III. Modeling Digital Systems for MBSA

Digital systems, by essence, lend themselves well to finite state machines representation making the use of languages and formalism such as Altarica very suitable for their modelling. However, dysfunctional modeling requires extracting the faulty behaviour of the blocks composing the system. Such task is usually carried out by a Failure Mode and Effect Analysis (FMEA) to identify possible malfunction of the individual blocks. In digital system, such task can be performed automatically by simulation with fault injection [27] and possibly formal methods [28].

The main issue in modelling digital systems for MBSA is choosing the adequate level of abstraction avoiding a direct $1 \Leftrightarrow 1$ translation of *Hardware Description Languages* (HDL) modeling concepts into Altarica. When extracting a safety model from a digital block three points must be addressed:

- Structural hierarchy: Because Altarica support hierarchy [29], translating hierarchy with adequate granularity can be straightforward, especially as natural design hierarchy is usually a good candidate.
- Behavioral modelling: Faulty behavioral aspects requires extraction of failure modes which can be performed manually, based on design knowledge or automatically using fault injection or formal approaches. Fault injection is well suited to such analysis, especially in the world of digital design which rely heavily on HDL simulators and digital fault injection driven by ISO26262 requirements. In this work we will exclusively focus on fault injection.
- Faults propagation: Blocks in a SoCs are usually connected though buses with well defined protocols and their failures modes (*unaligned access* ...) are known. The issue comes in the granularity of the modelling that, if too low will lead to too numerous events (1 HDL signal $\rightarrow$ 1 flow variable) while a too high abstraction may prevent catching of some protocol failures.

Fault injection campaigns are used to characterize the behavior of the system from its output pins point of view which are the 'vectors' for faults propagation between blocks. Also, knowing the functionality of each of these pins, it is possible to attach some possible consequences to the failure to such (group of) output(s). Such semantic labelling is, however, still manual and based on safety engineers knowledge and experience.

IV. Methodology

On top of any explicit finite state machine or control code encoding the user specified behaviour, a more complete state exist that includes the totality of the signals belonging to the control path of a design, such as data states implicitly exposed in controls states, or implicitly coded control states. These signals compose a more complete and larger state machine exposing new states and transitions that are implicitly specified, for example resulting from Cartesian product of automatons. Those are, technically, the signals driving transitions conditions.

Combinations of these signals in those states can lead to a subtle set of fault states, difficult to identify from the HDL description as the encoding in this state machine is sparse due to correlations. Such argument comes from the fact that even for a small ($>\approx$50) number of flip-flops, the complete state space (2^{50}) cannot be traversed in a reasonable time. Therefore a non-negligible proportion of these states are what we call *illegal states* i.e. unreachable under normal behaviour, potentially leading to undesired and unspecified behavior when the block is exposed to those states though faults.

In order to build a failure model from a nominal behavioral *Register Transfer Model* (RTL) in Verilog or VHDL, behaviour of the system under faults must be analyzed and faulty behavior as well as failure modes must be extracted. We proceed using the following steps:

1) *Identification and Extraction of State Signals*: Starting from the functional description, the set of flip-flops, belonging to both the *control* and possibly *data*) paths composing what we name as the *state*, has to be identified and extracted. This set, composed by all the flip-flops composing the control path and possibly the datapath which maintain the control state of the block, correspond to possible fault injection sites.
2) *Testbench Setup* : A standalone testbench is set up with care given to coverage and testbench representability as the states traversed during this golden execution will serve as non-faulty reference behavior. Tools like *Incisive Metrics Coverage* (IMC) [30] or *Certitude* [31] can be used to assess testbench coverage. A first reference run is performed to allow extraction of golden functional states that will be used later in the process to be differentiated from non-functional ones under fault injection.
3) *Fault Injection Campaign*: Fault injection is the mean by which the misbehavior and faulty execution is exposed on purposes. Probes (i.e., observation points) are defined during the setup of the fault injection campaign. They are set on the outputs of all blocks in order to identify failures that propagates to other blocks. Probes monitor and compare the probed signal value at each clock cycle with the golden reference and report any difference. They have been set to stop simulation when a fault reaches an output of the design. This step is the core of our analysis aimed at extracting faulty behaviour, modes and effects though exploration of the faulty states by fault injection.
4) *Extraction of Faulty Behavior*: Once the faulty runs have completed, non-functional (i.e. *faulty*) states and behavior are extracted by subtracting functional (*golden*) states taken from the golden run state dictionary to the faulty run states, leaving only newly discovered faulty states and transitions.
5) *Construction of the Faulty Model*: The newly discovered states and transitions are used to augment the functional models with faulty behavior. Transitions from a functional to a non-functional state are labeled with the responsible faults so are states responsible for an incorrect output. This model serves as a base for the translation into the Altarica language.

Currently, the method is limited in the *effect* analysis of the FMEA. Effect such as *loss of power* cannot be attached automatically to a faulty state as it would requires an inference and abstraction process out the reach of the tool currently. Thus, such labelling is performed manually by attaching effects to outputs and then back-propagating them into the states and faults responsible for the given outputs corruptions.

A. Faulty Behavior Model Construction

Once the faulty behavior has been extracted from faulty runs, the faulty model can be constructed using graph analysis algorithms. The first step in the model construction is collapsing states that are not meaningful for the dysfunctional model. We proceed currently with the following rules:

- Any component (connected subgraph) comprising only legal states and legal transitions are collapsed into one single *functional* state.
- Legal states with illegal transitions or incorrect outputs (outputs values do differs from reference in these states) are kept and illegal transition probabilities are attached.
- Any component comprising only nodes not propagating any faults to outputs are collapsed into one single *faulty* state. Probabilities to enter this state can be extracted from transitions leading to the collapsed states.
- Faulty nodes propagating faults to outputs are kept and transitions probabilities are attached to allow computing incorrect outputs probabilities.
- Effect attached to output pins are back propagated in the state graph faulty states where output corruptions occurs.

However additional rules may be added like to remove faulty nodes and transitions from masked faults for example, especially those not leading to any latent faults (execution is correct with no faults propagated to outputs and internal state doesn't differs from reference one at some point, i.e. fault has *vanished*). We ultimately target discrete-time Markov chains [32] for our dysfunctional behaviour modelling (Fig. 2).

B. Completeness of Extraction

The main risk in state identification is to under or overestimate the state which would lead to uncovered faulty states (fault not injected in a flip-flop misidentified as not *control*)

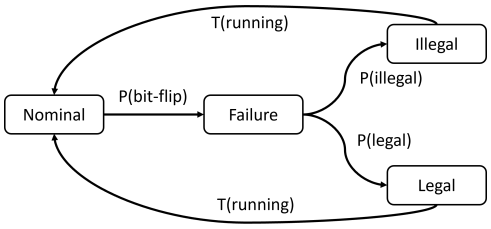

Fig. 2: Altarica Dysfunctional Model

or over estimate the state leading to classification of what are, in fact, data state as control states. The latter can be easily identified as randomizing data in the golden or faulty state machine extraction step leads to an increasing number of states with the number of runs. On Fig. 3, a correct identification leads to a saturating number of states (green curves) while an incorrect one leads to a diverging number of states as the number of tests grows (red curve).

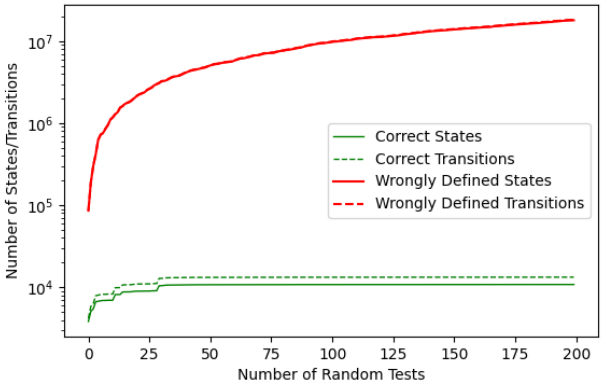

Fig. 3: Completeness of the Extraction

C. Altarica Modeling

Such an automaton representation is adequate for Altarica modelling as described below. When performing translation to Altarica, two elements shall be extracted:

1) The internal state machine corresponding to failures.
2) The assertion part corresponding to the propagation failure probability from input to one or more output of element.

Base modelling must include at least four states (Fig. 6): a *nominal* state where no failure occurs, a *failure* state corresponding to a bit-flip error injection and an *illegal* state corresponding to propagation of the failure to one or more outputs. The *legal* state correspond to failures leading to legals transitions, without failure propagation to outputs.

Assertions on outputs are conditioned by the internal state machine and inputs of the block. Every time internal state machine is in the illegal state, outputs values are updated. In same way, if one input of the block is set in the failed state, outputs are updated. Probabilities to generate a faulty output or to propagate failures from inputs to outputs are extracted from fault injection campaigns (Table II). Currently, all illegal states are collapsed into a single one, but different non-functional states corresponding to different failure modes can be extracted as well, such as represented on Fig. 6 where two illegal states are identified whether or not a simulation timeout (10% of golden execution time) occurs. Criteria for refined dysfunctional automaton extraction are not yet addressed as well as construction and reduction rules from fault injection data for such an automaton.

TABLE I: IMC Coverage Figures (%)

	I2C			AHB		
	cov.	tot.	overall	cov.	tot.	overall
Overall	352	410	93.8%	333	1057	61.3%
block	164	180	95.4%	51	68	85.55%
Expression	44	44	100%	7	17	41.18%
Toggle	112	148	75.68%	266	963	30.17%
FSM	32	38	83.36%	9	9	100%

V. Application: *I2C* to *AHB* Bridge

In order to exercise the methodology presented in Section IV, we use a test case composed of 2 blocks: an *I2C* slave [33] connected to an *AHB* [34] bus master interface (Fig. 4). Commands (*read* or *write*) along with parameters (*address* and *data*) are received on the serial line and transformed into a series of AHB read and write transactions. Such a system, composed of two interconnected blocks, is humanely understandable so are its dysfunctional modes, while being complex enough to detail thoroughly the methodology.

The I2C slave, taken from [35], receives *read* or *write* commands followed by an address byte and an optional data byte. On an I2C read, the byte returned from the AHB read transaction is returned. chronogramm for the read and write sequences are represented on Fig. 5. The system is represented on Fig. 4. At both end of the system (I2C input and AHB output buses), I2C master and AHB slave Verification IPs (VIP) are attached to generate and verify correctness of I2C and AHB transactions.

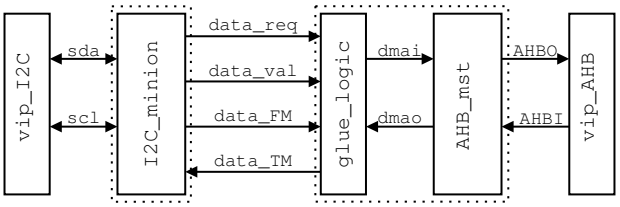

Fig. 4: I2C to AHB System Block Diagram

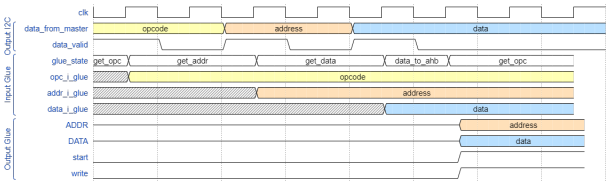

Fig. 5: I2C/AHB System Model

A. I2C Block Modeling

The testbench is composed of a series of read and write random transactions. The coverage evaluation of the design has been carried out, results are presented in Table I. Having considered the results of the coverage evaluations sufficient for the demonstration, application of the method presented in section IV have been performed. The list of all injection sites, reported by Cadence *Xcelium Fault Simulator* (FSV) [30] fault injection tool are considered for state including ones containing data as the serial nature of the I2C protocol, which mixes control and data frames on the same signals thought time-multiplexing, doesn't allow differentiation. However, the small size of data considered (8-bit) only induce a low (256) superset of the real control states. All outputs are probed so that any mismatch with the reference run will stop the simulation and report the fault as *Detected*. State (flip-flop value, i.e. '0' or '1') is simply extracted at each clock cycle and printed in the simulation logs to be post-processed.

Fault injection traces are then processed following rules described in section IV-A extracting transitions probabilities between the connected subgraphs:

- `Nominal 1` - Subgraph made of legal states only, part of the nominal execution.
- `Nominal 2` - Subgraph made of legal states only, part of the nominal execution.
- `Faulty 1` - Illegal state Subgraph, leading to a propagation of the fault to the output.
- `Faulty 2` - Illegal state Subgraph, leading to a simulation timeout.

The resulting model is represented on Fig. 6.

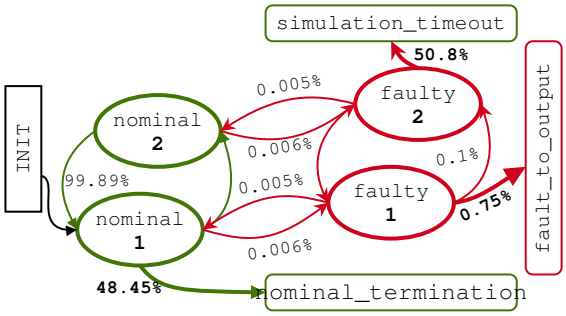

Fig. 6: Extracted FSM for the I2C Block

B. AHB Block Modeling

The AHB bus interface is taken from the GRLIB [36] library with added custom logic to connect it to the master parallel interface of the I2C. The added logic comprise an interpreter for the command received by the I2C and the glue logic interface to the AHB master side. A verification IP is connected to the AHB slave interface side to respond to transactions and check protocol. Fig. 5 represents the translation of I2C signals into an AHB transaction by the system. Coverage for AHB block is low and can be explained as only a limited use of the AHB protocol is made:

1) only byte accesses are performed.
2) only single (SINGLE) non-sequential (NONSEQ) transfers are performed.
3) the VIP has not been programmed to insert HREADY wait states in the transaction.
4) the VIP has not be programmed to generate HRESP transaction response error.

The low coverage obtained here doesn't restrict the generality of the methodology but may prevents some failure modes to be identified in this specific case.

C. Complete System Test Case

The complete system is composed of both the I2C slave and AHB master along with VIPs at both ends. As previously mentioned, probes are placed on all outputs of the complete system, leaving this time, faults freely propagating internally between the I2C and the AHB without being reported by FSV nor the simulation to be stopped. The main difference of this testbench regarding the two standalone previous ones is that faults injected in one block will be able to propagate to the other one (I2C $\rightarrow$ AHB, for example) and back-propagate to the first block (AHB $\rightarrow$ I2C) as simulation will not be stopped when the fault will output from the first (i.e. I2C), and later second (i.e. AHB), block. Such "fault loop" (I2C $\circlearrowleft$ AHB or AHB $\circlearrowleft$ I2C) are expected to be the main possible source of faulty states differences between the standalone and full system faulty states extraction. However, as faults are injected on the inputs in both approach (standalone and full system), we expect to capture, at least a part of theses "fault loops" induced faulty states in the standalone extractions, if such case exist.

The AltaRica structural model architecture follows the natural hierarchy of the system. As shown on Fig. 7, the AltaRica models includes the exact same blocks with the same interconnections between blocks as the functional model. The main I2C and AHB modules are composed of two sub-elements, shown respectively in Fig. 8 and Fig. 9.

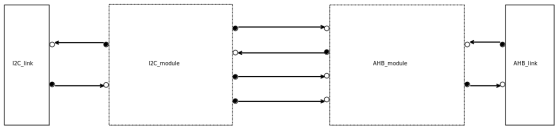

Fig. 7: I2C to AHB System Model

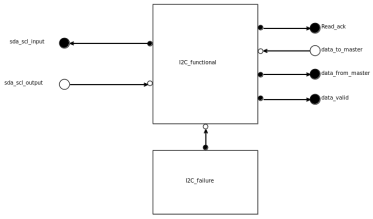

Fig. 8: I2C Block Model

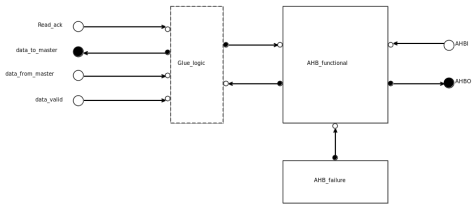

Fig. 9: AHB Block Model

The first element is the *functional* state machine of the module. In case of internal or external fault, this state machine will dispatch the fault to the impacted outputs. This state machine only model the internal faults propagation and do not generate any random failure on its own. The second element is the *internal failure* state machine. This state machine generate internal random failures and provide to the functional state machine the outputs impacted by it. In addition, the AHB module include an additional glue-logic block that converts I2C output signals to AHB bridge input signals. No internal failure are generated by this element. At both end of the I2C and AHB blocks, links module have been added to model the faults coming from outside of the system. To model the behavior of the I2C and AHB system, only standalone block fault injection test results have been used.

Depending on the methodology, two types of metrics can be extracted. The first one is the probability to propagate internal failure to one or more outputs of the system. From the test results, this probability has been extracted by considering all faults injected in the studied system. The probability to propagate internal faults to an output of the system is then equal to the ratio between the faults detected by the output probe and the total number of faults injected.

The second metrics is the probability to propagate a failure from an input of the system to one or more output of the system. For this metric, only fault injected on inputs have been taken into account. This probability is the ratio of the input faults leading to an erroneous output over the total number of input faults injected.

SimfiaNeo allows to perform Monte-Carlo simulation. In this type of simulation, a large number of failure scenarios are generated to assess the mean behavior of the system under random failure scenarios. The first possible assessment randomly injected one failure by failure scenario inside the system

TABLE II: Block and Full System Fault Injection Results

		I2C	AHB	I2C + AHB	
Golden states (static)		44	11	44	11
Golden transitions (static)		90	20	89	19
Faulty states (static)		223	126	198	193
Faulty transitions (static)		1093	470	899	428
Injected faults (400 / FF)		11670	41200	52878	
Detected faults	golden	6840 (58.61%)	19285 (46.80%)	23840 (45.08%)	22089 (41.77%)
	faulty	19 (0.16%)	466 (1.31%)	219 (0.41%)	1970 (3.72%)
Detecting states	golden	44 (100%)	11 (100%)	44 (100%)	11 (100%)
	faulty	5 (1.87%)	13 (9.48%)	5 (2.06%)	70 (34.31%)

TABLE III: RTL Fault Injection Vs Altarica Model Failures

AHB Failure	I2C+AHB RTL	Altarica Model	I2C Failure	I2C+AHB RTL	Altarica Model
haddr	0.00551	0.00551	SDA	0.00338	0.00339
hwdata	0.00382	0.00382	Read req	0.00254	0.00255
hsize	0.00068	0.00068	Data	0.00580	0.00581
hbusreq	0.00026	0.00026	Data valid	0.00322	0.00323
hwrite	0.00022	0.00022			

while the outputs are monitored. If at least one output triggers a faulty state, the error is accounted to have been propagated outside of the system. With this methodology, it's possible to estimate the probability to have a failure propagation from the I2C+AHB system to the I2C or AHB external signals. The second possible assessment randomly injected one failure by failure scenario in a link module and monitored the other link module. If the opposite link module triggers a faulty state, the error is accounted to have been propagated from one end to another. With this methodology, it is possible to estimate the probability to have a failure propagation from AHB or I2C back to the other link.

VI. Results

Result of composition obtained by SimfiaNeo are compared to fault injection performed on the full system with probes set only on external outputs of the system on table III for the I2C and AHB side signal. Because the system is simple and faults propagate only forward, it came to simple probability multiplication explaining exact matching of model and system fault injection. No back-propagating faults were observed.

VII. Discussion and Future Work

In this work, we have proposed and experimented the use of Model-Based Safety Assessment on digital system for safety analysis. We have addressed the construction of dysfunctional model for digital system using simulation and have been able to build a simple, but functional dysfunctional model in Altarica. Ongoing work include automatic dysfunctional models reduction to more than one state and the application to a small RISCV SoC and software reliability [37].

References

[1] R. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, pp. 305 – 316, 10 2005.

[2] J. F. Ziegler and G. R. Srinivasan, "Ibm journal of research and development," *IBM J. Res. Dev.*, vol. 40, no. 1, p. 2, 1996.

[3] M. Ottavi, D. Asciolla, and T. Fiorucci, "Setup and experimental results analysis of cots camera and srams at the isis neutron facility," in *2018 13th International Conference on Design Technology of Integrated Systems In Nanoscale Era (DTIS)*, 2018, pp. 1–4.

[4] T. Karnik, P. Hazucha, and J. Patel, "Characterization of soft errors caused by single event upsets in cmos processes," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, 2004.

[5] S. Mukherjee, J. Emer, and S. Reinhardt, "The soft error problem: An architectural perspective," 03 2005.

[6] E. Touloupis, J. A. Flint, V. A. Chouliaras, and D. D. Ward, "Study of the effects of seu-induced faults on a pipeline protected microprocessor," *IEEE Transactions on Computers*, vol. 56, no. 12, 2007.

[7] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin, "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor," 2003.

[8] R. Mariani, G. Boschi, and F. Colucci, "Using an innovative soc-level fmea methodology to design in compliance with iec61508," in *Design, Automation Test in Europe Conference Exhibition*, 2007.

[9] ISO26262, "Road vehicles – functional safety," 2018. [Online]. Available: https://www.iso.org

[10] N. Wang, A. Mahesri, and S. Patel, "Examining ace analysis reliability estimates using fault-injection," vol. 35, 06 2007.

[11] C. Bottoni, M. Glorieux, J.-M. Daveau, G. Gasiot, F. Abouzeid, S. Clerc, L. Naviner, and P. Roche, "Heavy ions test result on a 65nm sparc-v8 radiation-hard microprocessor," in *2014 IEEE International Reliability Physics Symposium*, 2014.

[12] J. Torras Flaquer, J.-M. Daveau, L. Naviner, and P. Roche, "Handling reconvergent paths using conditional probabilities in combinatorial logic netlist reliability estimation," in *2010 17th IEEE International Conference on Electronics, Circuits and Systems*, 2010.

[13] J. Torras-Flaquer, "Méthodes probabilistes pour l'estimation de la fiabilité dans la logique combinatoire," *Thèse de Doctorat de l'école doctorale TELECOM Paristech, 2011*.

[14] ISO26262, "Acronyms and abbreviations." [Online]. Available: https://www.arteris.com/blog/top-35-iso-26262-acronyms-and-abbreviations/

[15] K. Devarajegowda, J. Vliegen, G. Petrovity, and K. Fotouhi, "A mutually-exclusive deployment of formal and simulation techniques using proof-core analysis," 10 2017.

[16] P. Yeung, D. Smith, and A. Ayari, "Whose fault is it formally? formal techniques for optimizing iso 26262 fault analysis," 02 2018.

[17] J. Marin and R. Pollard, "Experience report on the fides reliability prediction method," in *Annual Reliability and Maintainability Symposium, 2005. Proceedings.*, 2005.

[18] N. Yakymets and M. Adedjouma, "Model-based quantitative fault tree analysis based on fides reliability prediction," in *IEEE International Symposium on Software Reliability Engineering Workshops*, 2020.

[19] R. Brayton, G. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-t. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, T. Shiple, and G. Swamy, "Vis : A system for verification and synthesis," 08 2000.

[20] R. K. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *CAV*, 2010.

[21] S. Venkataramanan, A. Kumari, and L. Piper, *SystemVerilog Assertions Handbook*. CreateSpace Independent Publishing Platform, 2015.

[22] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science*, vol. B: Formal Models and Sematics, 1990.

[23] H. Mortada, T. Prosvirnova, and A. Rauzy, "Safety Assessment of an Electrical System with AltaRica 3.0," in *4th International Symposium on Model-Based Safety Assessment*, Munich, Germany, Oct. 2014.

[24] T. Prosvirnova, "Altarica 3.0: a model-based approach for safety analyses," *Thèse de doctorat de l'ecole Polytechnique, 2014*.

[25] A. Arnold, G. Point, A. Griffault, and A. Rauzy, "The altarica formalism for describing concurrent systems," *Fundam. Inf.*, vol. 40, no. 2–3, 1999.

[26] M. Bozzano, A. Cimatti, O. Lisagor, C. Mattarei, S. Mover, M. Roveri, and S. Tonetta, "Statistical fault injection: Quantified error and confidence," in *Proceedings of the 10th International Workshop on Automated Verification of Critical Systems*, 2010.

[27] M. Kooli and G. Di Natale, "A survey on simulation-based fault injection tools for complex systems," in *2014 9th IEEE International Conference on Design Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2014.

[28] S. Ramírez and C. Rocha, "Formal verification of safety properties for a cache coherence protocol," in *2015 10th Computing Colombian Conference (10CCC)*, 2015.

[29] T. Prosvirnova and A. Rauzy, "The structural constructions of AltaRica 3.0," in *Actes du congrès LambdaMu'19*. IMdR, 2014.

[30] "Cadence." [Online]. Available: https://www.cadence.com/en_US/home.html

[31] "Certitude." [Online]. Available: https://www.synopsys.com/verification/simulation/certitude.html

[32] Y.-C. Chen, "Discrete-time markov chain," 2018. [Online]. Available: http://faculty.washington.edu/yenchic/18A_stat516/Lec3_DTMC_p1.pdf

[33] J. Mankar, C. Darode, K. Trivedi, M. Kanoje, and P. Shahare, "Review of i2c protocol," *International Journal of Research in Advent Technology*, vol. 2, no. 1, 2014.

[34] P. Giridhar and P. Choudhury, "Design and verification of amba ahb," in *1st International Conference on Advanced Technologies in Intelligent Control, Environment, Computing Communication Engineering*, 2019.

[35] "I2c minion." [Online]. Available: https://github.com/oetr/FPGA-I2C-Minion

[36] "Grlib." [Online]. Available: https://www.gaisler.com/

[37] T. Fiorucci, G. Di Natale, J.-M. Daveau, and P. Roche, "Software product reliability based on basic-block metrics recomposition," *IEEE International Symposium on On-Line Testing*, 2023, To appear in.