Cross-Layer Dependability Modeling and Abstraction in System on Chip

Andreas Herkersdorf*, Michael Engel†, Michael Glaß‡, Jörg Henkel§, Veit B. Kleeberger*,
Michael A. Kochte††, Johannes M. Kühn‖, Sani R. Nassif¶, Holm Rauchfuss*, Wolfgang Rosenstiel‖,
Ulf Schlichtmann*, Muhammad Shafique§, Mehdi B. Tahoori§, Jürgen Teich‡, Nobert Wehn**,
Christian Weis**, and Hans-Joachim Wunderlich††

*Technische Universität München †Technische Universität Dortmund ‡Universität Erlangen-Nürnberg
§Karlsruher Institut für Technologie ¶IBM, Austin Research Laboratory ‖Universität Tübingen
**Technische Universität Kaiserslautern ††Universität Stuttgart

***Abstract*—The Resilience Articulation Point (RAP) model aims at provisioning researchers and developers with a probabilistic fault abstraction and error propagation framework covering all hardware/software layers of a System on Chip. RAP assumes that physically induced faults at the technology or CMOS device layer will eventually manifest themselves as a single or multiple bit flip(s). When probabilistic error functions for specific fault origins are known at the bit or signal level, knowledge about the unit of design and its environment allow the transformation of the bit-related error functions into characteristic higher layer representations, such as error functions for data words, Finite State Machine (FSM) state, macro interfaces or software variables. Thus, design concerns at higher abstraction layers can be investigated without the necessity to further consider the full details of lower levels of design. This paper introduces the ideas of RAP based on examples of radiation induced soft errors in SRAM cells and sequential CMOS logic. It shows by example how probabilistic bit flips are systematically abstracted and propagated towards higher abstraction levels up to the application software layer, and how RAP can be used to parameterize architecture level resilience methods.**

I. Introduction / Motivation

Nanometer feature size CMOS technologies are susceptible to a variety of dependability threats affecting all abstraction layers of System on Chip (SoC). A non-exhaustive list of examples for possible errors and their corresponding root causes are: Intermittent or permanent bit flips (SEU, SET) in memories as well as combinatorial and sequential logic due to radiation induced charge separation in the CMOS substrate; Transient signal integrity degradations and register timing violations due to capacitive coupled cross-talk or NBTI aging; Irreversible electromigration damages on interconnect wires due to excessive current densities or temperature hotspots, possibly in combination with manufacturing process variations.

Depending on the where and when such faults occur within an SoC, they either have no effect at all on the SoC behavior (because the fault is masked by other circuit conditions), cause an erroneous function output or data structure corruption or, in the worst case, result in a system crash.

While all of the above referenced faults originate at the low-level process or CMOS technology layers, the resulting errors and failures manifest at, and may propagate through, all HW/SW abstraction layers. Consequently, there are countermeasures to conquer these various error symptoms at every abstraction layer. However, it is not clear upfront, which fault type or error is most effectively tackled at what abstraction layer and by what form of countermeasure. Detecting and correcting an error directly at the level where it occurred may be possible but may not be the most efficient mean. For example, hardening SRAM cells against radiation-induced bit flips by means of using larger transistor comes at the expense of an area increase for each and every SRAM cell within the memory array. Applying information redundancy techniques in form of error detection and correction coding (ECC) during memory write/read operations results in much less area overhead and achieves the same result.

To effectively tackle these challenges while not compromising with performance targets, the ability to model and evaluate the various faults and errors at and across all SoC abstraction layers is a necessity. It is the declared objective of the German Research Foundation (DFG) Priority Program SPP1500 "Dependable Embedded Systems" to develop new cross-layer design methods and architectures for coping with reliability, performance degradation and increasing power dissipation issues when migrating to new CMOS technology nodes [1].

The proposed Resilience Articulation Point (RAP) method is the result of several working group meetings among SPP1500 partners and aims at provisioning a probabilistic error modeling and bottom-up error abstraction / transformation framework to characterize errors at different SoC hardware and software layers.

II. Resilience Articulation Point (RAP) Model

The RAP model is based on three principal pillars: First, the hypothesis that whatever physical phenomenon is the root cause for a fault, if it is not masked (i.e. eliminates itself), it will manifest as a permanent or transient single- or multi-bit signal invalidation $\mathcal{P}_{bit}$ (see Fig. 1). Second, cross-layer dependability optimization requires probabilistic methods for reliability modeling in order to cope with, abstract and quantify the impact of complex low-level fault exposures at higher levels. Third, transformation functions $\mathcal{T}_L$ convert probabilistic error functions $\mathcal{P}_L$ at abstraction level L into probabilistic error functions $\mathcal{P}_{L+i}$ at level(s) $L+i$ $(i \geq 1)$.

In graph theory, an articulation point is a vertex that connects sub-graphs in a biconnected graph, and whose removal would result in an increase of the number of connecting arcs within the graph. Translated to our domain of dependability challenges in SoC systems, spatially and temporally correlated bit flips represent the single connecting vertex between lower layer fault origins and the upper (hour glass) layer error and failure models of HW/SW system abstraction.

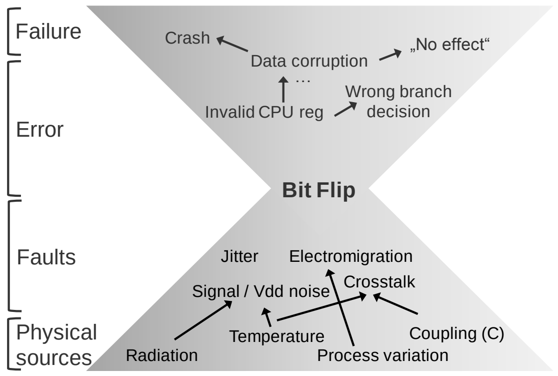


Fig. 1. Cross-layer representation of faults, errors, and failures with bit flip as Resilience Articulation Point

Error functions for different fault origins (radiation, aging, crosstalk or thermal hotspots, to name a few) and error transformation functions (such as for determining silent data corruption (SDC) or detected uncorrectable error (DUE) rates in microprocessor designs) are vital for the expressiveness of a RAP-based dependability assessment. However, it is not the intention of RAP (and beyond its abilities) to consider error and transformation functions to be an integral part of RAP. Neither is RAP a tool to develop such functions. RAP rather provides a framework where different fault origins, each being expressed as probabilistic bit errors for a particular signal, can be accumulated to represent an error function covering several physical shortcomings. Even when this accumulation and individual error models are approximate, they relief the SoC designer with expertise at higher abstraction levels from the details of the technological and device level aspects of SoC. This concept is applicable at each abstraction level including and above the bit or signal level.

Cross-layer approaches are suggested in related work as feasible techniques to enhance reliability of complex systems ([2],[3]). RIIF [4] proposes a standard language to foster exchange of reliability information and models among components at different levels and different EDA tools. Fault and error modeling in the space and time domain has a long tradition in the LSI testing community. The generalized conditional line flip model [5] allows specification of Boolean and temporal activation conditions. Excessive process variations may cause test invalidation of delay tests which threatens product quality. Probabilistic fault modeling aims to quantify the quality of the test and final product w.r.t. the parameter space in spite of high uncertainty of variations [6].

The remainder of the paper is structured as follows: Section III introduces the basic assumptions of our probabilistic error modeling under environmental, process and system state related constraints. A realistic SRAM circuit was used as example in section IV to calibrate the analytical model with real hardware for the fault scenario of radiation induced bit flips (soft errors). This is followed by a generalization of the SRAM fault model towards combinatorial and sequential logic circuits. Section V and VI describe how the RAP bit flip model is propagated towards higher abstraction levels up to the software application layer.

III. The Lower Half Of The Hour Glass

The task of an error model at the lower levels is to describe the probability of an occurring bit error as a function of parameters that may change during system design or operation.

We propose to model the error probability $\mathcal{P}$ of a bit by an error function $\mathcal{F}$ of three parameter vectors: Environmental and operating conditions $\mathcal{E}$, design parameters $\mathcal{D}$, and (error) state of correlated bits $\mathcal{S}$.

$$\mathcal{P} = \mathcal{F}(\mathcal{E}, \mathcal{D}, \mathcal{S}) \tag{1}$$

This generic model has to be adapted to every circuit component and fault type independently. This enables then the modelling of different components (e. g. SRAM or latches) and different errors (e. g. soft errors or timing violations).

A. *Environmental and Operating Conditions $\mathcal{E}$*

Almost all the functionality of a circuit is dependent on its environmental conditions. Device temperature and supply voltage values determine the electrical properties of all components in the circuit. Circuit age changes electrical properties such as threshold voltage. Other possible parameters include clock frequency or neutron flux density.

These parameters represent an interface to either user decisions or other models in the design process. For example, in a simplified analysis supply voltage might be a fixed value, while in a more detailed analysis it might come from some more advanced model [7].

B. *Design parameters $\mathcal{D}$*

During the design stage several decisions have to be made. For example, shall arithmetic adders follow a ripple-carry or carry-lookahead architecture (enumerative decision)? What technology node to choose (discrete decision)? How much area should one SRAM cell occupy (continuous decision)?

This allows the designer to make trade-offs between different decisions which all influence the error probability.

C. *Correlated (Error) States $\mathcal{S}$*

To model the dependence of the error probability on location, circuit state, and time it might be necessary to include several state variables.

These state variables lead to a model which is built from conditional probabilities $\mathcal{P}(b_1|b_2)$, where the error probability of the bit b_1 is dependent on the state of the bit b_2.

For example, the failure probability of one SRAM cell depends on the error state of neighboring SRAM cells due to the probability of Multi Bit Upset (MCU) [8]. For an 8T

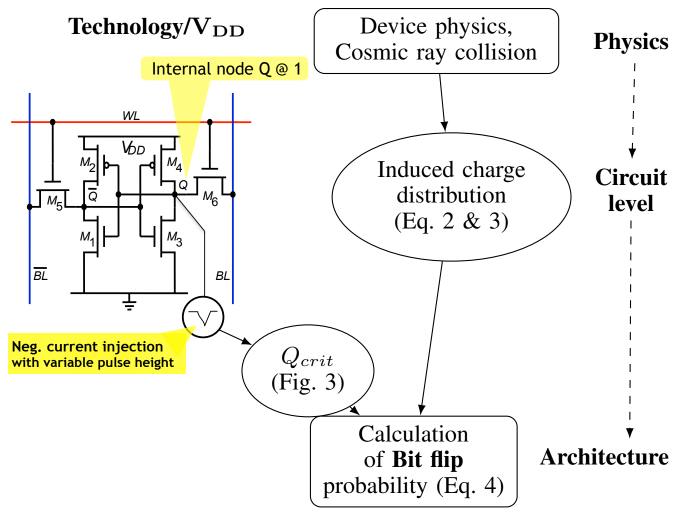
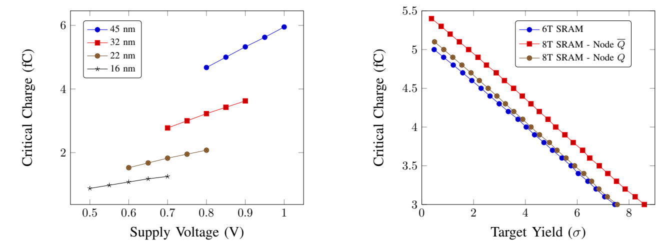

Technology/V_DD
Device physics, Cosmic ray collision
Physics
Internal node Q @ 1
WL
VDD
Induced charge distribution (Eq. 2 & 3)
Circuit level
Neg. current injection with variable pulse height
Q_{crit} (Fig. 3)
Calculation of Bit flip probability (Eq. 4)
Architecture

Fig. 2. SRAM Bit Flip Model



(a) Dependence on V_{DD} for different technologies

(b) Dependence on target yield (in sigma) for different architectures

Fig. 3. Critical Charge Dependence

SRAM cell it also depends on the stored value of the SRAM cell as the bit flip probability of a stored one is different from a stored zero.

D. The Error Function $\mathcal{F}$

The error function $\mathcal{F}$ finally takes the three parameter sets $\mathcal{E}$, $\mathcal{D}$, and $\mathcal{S}$ and returns the corresponding bit error probability.

The error function is unique for a specific type of fault and for a specific circuit element. It might be possible to express the error function by simple analytical formulas. On the other hand, the error function might also require a non-closed form representation, e.g. a timing analysis engine or a circuit simulator.

IV. Examples for Low-Layer Error Models

In the following sections we describe bit flip error models of SRAMs and combinational or sequential logic cells. Similar methods were presented in [9].

A. SRAM Single Event Upsets

One common example where bit flips are encountered in a chip is an SRAM cell. We will show in this example how neutron induced bit flips can be modeled in an SRAM array by using the generic model from Section III. A bit flip in an SRAM cell occurs when a particle strike induces enough charge on a point within the cell to cause a flip in the cell's content. Thus, an SRAM bit flip model requires the critical charge to flip a cell as well a distribution describing the probability of charge injection (see Fig. 2).

The critical charge which is required to flip a cell can be characterized for a given cell architecture using SPICE simulation [10]. Variation of environment temperature or cell supply voltage introduces a dependence of critical charge Q_{crit} on environmental conditions (Fig. 3a). The dependence on design parameters can also be characterized in a similar way, and the influence of cell area can be modeled by varying the size of the transistors inside the SRAM cell. The dependence on target yield can be found by adding a worst-case analysis to the characterization as described in [11]. This results in a discrete model for the critical charge Q_{crit} dependent on environmental and design parameters (Fig. 3) which may additionally be approximated by some analytical function.

For the second part of the model in Fig. 2 we need the probability that a charge which is larger than the critical charge is injected by a particle strike. The probability that a neutron strikes the cell can be modeled by a Poisson process [12]:

$$P\left(N(T)=k\right)=\exp\left(-\Phi\cdot A\cdot T\right)\frac{(\Phi\cdot A\cdot T)^k}{k!} \qquad (2)$$

This equation expresses the probability that k neutrons hit an area A during the time interval T which is exposed to a neutron flux Φ. These neutrons are then uniformly distributed over the SRAM area and may only cause an error if they hit the critical area of one of the cells.

Once the neutron strikes the critical area of the cell it may generate electron-hole pairs, which have the potential to change the charge stored on the capacitances inside the chip. We assume in the following that the probability distribution of injected charges due to a neutron strike follows an exponential distribution [13]:

$$f_Q(Q_{injected})=\frac{1}{Q_s}\exp\left(-\frac{Q_{injected}}{Q_s}\right) \qquad (3)$$

The parameter Q_s is the charge collection slope due to one neutron strike, which is technology dependent [10]. The probability P_{SEU} of a cell flip, and thus a bit error $P_{bit}(\vec{x},t)$, can then be composed from the critical charge of the cell Q_{crit} (Fig. 3) and equation 3:

$$P_{SEU}(Q\geq Q_{crit}|\text{Node Q}=1)=\int_{Q_{crit}}^{\infty}f_Q(Q)dQ \qquad (4)$$

With increasing integration density the probability of Multi Bit Upsets increases. Possible reasons for this include the successive hit of multiple storage nodes by the same neutron, shared charge to adjacent cells, or parasitic bipolar transistors in the case of bulk technology [14]. To correctly account for Multi Bit Upsets we therefore have to add error state variables to the model. For this we first have to characterize the occurrence probability of given shapes [15]. Using these occurrence probabilities we can account for Multi Bit Upsets using conditional probabilities which determine the probability that an adjacent cell is upset given the upset of spatially close cells [8].

B. Combinatorial and Sequential Logic

When a neutron strikes a combinatorial or sequential logic block within the SoC, it will result in a charge separation within the semiconductor substrate material which may lead to a voltage pulse on a signal wire line (see signal A in Fig. 4).

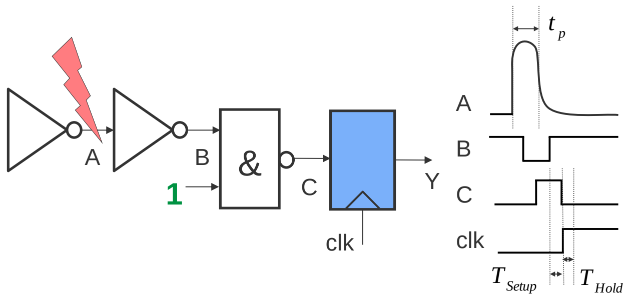


Fig. 4. Bit flip as result of SEU or SET

The temporal width of the voltage pulse again depends on the energy of the particle, the technology feature size, the capacitive load of the signal, the supply voltage (in other words, on the $\mathcal{E}$, $\mathcal{D}$, and $\mathcal{S}$ parameters). However, the voltage pulse only results in a functional error (i.e. a false bit value latched into the following register stage affecting signal Y), if the pulse propagates from the location of occurrence to the register stage on a combinatorially sensitized path and overlaps with the critical time window $\Delta T_{crit} = T_{Setup} + T_{Hold}$ around the active clock edge. Otherwise, the pulse will be masked out and thus, never be noticed. The probability for a bit error $P_{bit}(\vec{x}, t)$ within sequential logic again is spatially (where in the combinatorial net did the strike occur) and temporally (what is the combinatorial path delay between strike location and register input) correlated with the fault and, with the probability P_{sense} to have a sensitized path to the register, approximated as:

$$P_{bit}(\vec{x}, t) \approx \frac{T_{Setup} + T_{Hold} + t_p}{T_{clk}} \cdot P_{sense} \cdot P_{SET} \quad (5)$$

Signal Y in Fig. 4 can be considered as an individual bit of a data word in a sequential data path pipeline or a bit within a state vector of a control FSM. Upsets on clock trees would result in multiple (hundreds of) erroneous register contents (data / control word corruptions). Clock tree upsets can be modeled as transient bit flips too, but will occur significantly less likely as clock buffers are usually hardened by multiple sequential nMOS and pMOS transistors in the buffer / inverter designs. A signal degradation on an i/o bit will result in an interface (control) error at a higher layer of abstraction and is also in line with the RAP model. In consequence, we now have a probabilistic bit flip model for combinatorial and sequential logic, interconnect wires, external interfaces and memory arrays, and thus cover all fundamental functional building blocks of SoCs or computing architectures.

V. The Upper Half of the Hour Glass

Bits or individual signals are meaningful targets for describing errors at the transistor, logic gate and RTL levels of abstraction. At higher layers, compounds of multiple signals/bits, referred to as data, control or address words, FSM state vectors, variables, interfaces or data structures, are more intuitive and descriptive, particularly for software developers (see Fig. 5). On the other hand, a memory data word is nothing but a bundle of multiple (say 32) consecutive memory cells or memory bits. Thus, when assuming individual bit errors $P_{bit}(\vec{x}, t)$ in space $\vec{x}$ and time t within memory cells to be independent, one can determine the approximate $P_{word}(\vec{x}, t)$ error probability by the following concrete transformation function T_{bit}:

$$P_{word}(\vec{x}, t) = T_{bit} \circ P_{bit}(\vec{x}, t) = 1 - \prod_{x_i \in \vec{x}} (1 - P_{bit}(x_i, t)) \quad (6)$$

The derivation of word error probabilities under consideration of correlated data bits and interleaving is also possible. We refer to [16] for a more complete discussion of this more complex case.

When operand variables of arithmetic operations are stored in an SRAM memory array, then $P_{word}(\vec{x}, t)$ describes the probability with which these variables contain erroneous data. In case the same variables are kept in the CPU register file, then a different $P_{word}(\vec{x}, t)$ describes the trustworthiness of the contents of the register file. The two $P_{word}(\vec{x}, t)$ probabilities are different because the technological (D) and state-related constraints (S) of SRAM arrays and a register files are different. $P_{word}(\vec{x}, t)$ can also incorporate potential dependability countermeasures applied at word level abstraction layers (e. g. ECC detecting and correcting up to k bit errors per word). In case of ECC protection, only $N > k$ accumulated bit errors within one and the same data word and between two consecutive write refreshes will result in a word / variable error. $N < k$ bit flips per data word remain invisible (i.e., are masked) for the software layer. Hence, an ECC protected memory has a different $P_{word}(\vec{x}, t)$ than a non-protected memory, although both may have the same bit level $P_{bit}(\vec{x}, t)$.

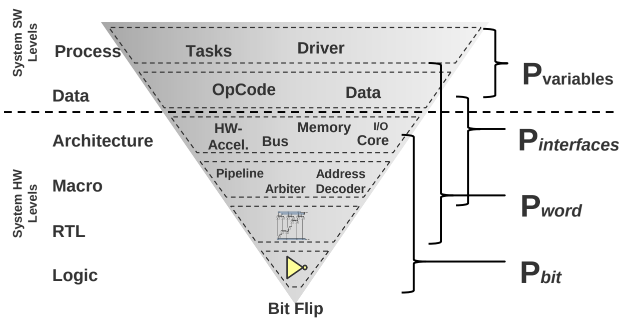


Fig. 5. Abstraction and transformation of bit flips in higher system model layers

Similar, the vulnerability of data, address and control word transports on on-chip buses, or the data transformation within the combinatorial logic blocks of a CPU data path or FSM control structure can be expressed as $P_{word}(\vec{x}, t)$. At higher abstraction layers, the units of words, or compounds of words referred to as interfaces (or data structures), substitute bits or signals. The corresponding $P_{interface}(\vec{x}, t)$ models are derived from the error probabilities at bit-level $P_{bit}(\vec{x}, t)$ or word-level $P_{word}(\vec{x}, t)$, plus additional knowledge on the internal IP block architecture and topology. In other words, the D and S constraints at the respective abstraction layers represent a transformation model between $P_{bit}(\vec{x}, t)$, $P_{word}(\vec{x}, t)$ and $P_{interface}(\vec{x}, t)$.

A. Divide and Conquer

Once we can describe the dependability exposure of a complex SoC by probabilistic functions for data bus words and operand variables, higher layer SoC behavior (HW architecture and SW layers) can again be investigated without maintaining the complete set of lower layer models. $\mathcal{P}_{word}$ or $\mathcal{P}_{interface}$ are adequate representatives of the lower layer errors. They can be considered as adequate error injection means at, e.g. architecture or system software levels, thereby replacing complex lower layer models.

Abstraction level specific probabilistic error models and transformation functions can be used for propagating error models towards higher abstraction levels. Mathematically, this can be expressed by the following equation and is graphically depicted in Fig. 6:

$$\mathcal{P}_{L+i} = \mathcal{T}_L(\mathcal{E}_L, \mathcal{D}_L, \mathcal{S}_L) \circ \mathcal{P}_L \qquad (7)$$

Transformation functions can stretch one or several abstraction levels. The SRAM data word example from Eq. (6) dealt with two consecutive abstraction levels. Section VI below will show cases where transformations cover multiple levels, from bit to architecture level and word to application software level, respectively. Abstraction levels not only have specific transformation functions, but also level specific environmental, design and correlated state parameters. Externally imposed workloads and fault exposure patterns contribute to the environmental dimension, abstraction level related design structures and templates to the design and state related parameters. Dynamic program flow is considered through the workload (environmental $\mathcal{E}_L$ parameters) and, thus, affect the error model at higher abstraction level(s).

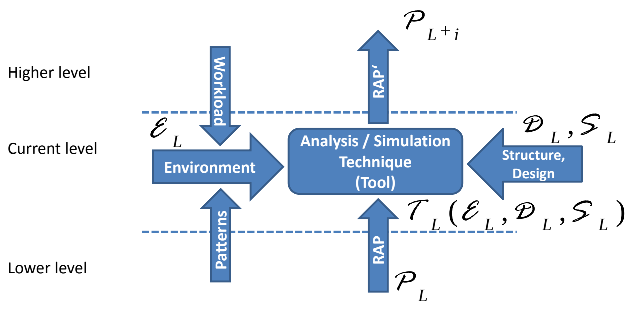


Fig. 6. Error transformation / propagation in the upper half layers

VI. Transformations, Tools and Applications for the Upper Half

Transformation functions are essential for raising the level of abstraction and obtaining application-specific results out of RAP-based dependability analysis. However, the transformation functions themselves are not an integral part of RAP. In the following sections we provide examples of how transformation functions can be used in the context of RAP-based system analysis and drive architecture related design decisions.

A. Data and Instruction Vulnerability Analysis

A large number of embedded software applications can tolerate certain errors with negligible output quality degradation. Nevertheless, errors leading to significant output deviations or even system crashes have to be corrected mandatorily. Flexible error handling requires meta data to indicate the vulnerability of a given data object or code sequence to errors. This can be accomplished by providing reliability annotations in application source code, e.g., by means of a binary classification into "critical" and "non-critical" objects. This information is propagated throughout the application using static analysis and source or binary code transformations.

The SPP1500 FEHLER project uses this meta data at compile time to map data and instructions to components of appropriate reliability [17]. In addition, for errors manifesting at runtime, meta data is used by the operating system to determine the appropriate error correction method considering current resource availability. Lower level RAP word error models $P_{word}(\vec{x}, t)$ can be used to decide *if*, *when*, and *how* to correct a given error. This can significantly help to assess the overhead required for error correction under different workloads (E parameter) at runtime. Extensions to the analysis and related meta data will help to consider additional design (D) trade-offs, e.g., between output quality, real-time constraints, and energy consumption.

Instruction Vulnerability Index (IVI) [18] and Instruction Masking Index (IMI) [19] are alternative approaches providing probabilistic estimations/quantifications for the vulnerability, masking of application software at various granularities, i.e. instruction, basic block, and function. The IVI model (Eq. (8)) quantifies the spatial and temporal vulnerabilities of different types of software instructions in different microarchitecture/RTL-level pipeline stages $c \in C$ of a given processor according to their area A_c and error probability $P_E(c)$ [18]. It jointly considers the effects of faults in different processor components (spatial), during the execution of different instructions (temporal), types of errors, (non-)critical instructions, and vulnerable bit analysis. The error probability $P_E(c)$ for each pipeline component is obtained using the HW-level reliability methods like EPP [20], CEP [21] and CLASS [22]. These techniques provide probabilistic analysis of error propagation from error site ($P_{bit}(\vec{x}, t)$ or $P_{word}(\vec{x}, t)$) to the reachable primary outputs using topological traversal of the netlist. Moreover, the correlation in propagated errors to multiple outputs as well as multi-cycle propagation of latent errors in flip-flops and memories are handled by these techniques. The correlation coefficient method is adopted to obtain error probabilities and correlations of primary outputs due to particle strike at internal nodes. A_c is obtained from the hardware synthesis results.

$$IVI_i = \frac{\sum_{\forall c \in C} IVI_{ic} \times A_c \times P_E(c)}{\sum_{\forall c \in C} A_c} \tag{8}$$

IVI_{ic} denotes the vulnerability at a processor component c (with an architecturally-defined size β_c) is given as the product of its vulnerable periods in that processor component (υ_{ic}) and vulnerable bits affecting the Correct Execution ($\beta_{c(v)}$), as shown in Eq. (9). A_c and $\beta_{c(v)}$ capture the spatial vulnerability, while υ_{ic} captures the temporal vulnerability.

$$IVI_{ic} = \frac{\upsilon_{ic} \times \beta_{c(v)}}{\sum_{\forall c \in C} \beta_c} \tag{9}$$

$\beta_{c(v)}$ is obtained using the program-level analysis of vulnerable bits [18], bit error probabilities (Eq. (10)) and their correlations [21][22]. The above discussion on IVI model illustrates that how hardware- and program-level error analysis can be combined to accurately estimate the reliability at higher system layers.

$$P_{bitAVG}(i \mid j) = \frac{P_{bitAVG}(ij)}{P_{bitAVG}(j)} \tag{10}$$

IVI can then be used to derive the vulnerabilities at function, task, and application program level.

As IVI captures the probability of an error, IMI captures the software properties of how probable is that this error will ultimately propagate to the visible program output. Hence, IMI provides a transformation function T covering one or multiple abstraction layers.

B. Tool Perspective

The concept and advantages of RAP as a basic model for reliability considerations in the MPSoC domain can enhance existing and guide future design and analysis tools. As outlined in Sec. IV, RAP at the level of bit flips has the potential to close the gap between (a) lowest-level techniques that are aware of physical effects and (b) numerous higher-level techniques, e. g., from the fault-injection domain like [23]. The latter are agnostic of physical causes but rely on a mathematical description of an error as a discrete deviation from the expected state occurring deterministically or statistically. Given that the single bit flip is the smallest functional error unit, no inherent abstraction prevents the model from being generally applicable by already neglecting certain aspects. But RAP not only has the potential to bridge between those two worlds, but it also enhances the heterogeneous higher-level tool landscape by means of providing a transformation scheme as a step towards a cross-layer tool flow. As indicated in Sec. V-A, the concept behind RAP enables: (a) An abstraction from concrete fault models, in particular, it may even already serve as an abstraction for several concurrent fault models, and (b) different causes of errors can be composed and provided to the next higher level of abstraction. Modeling errors as flips in bits, words, interfaces, or variables is individually covered in existing simulation-based and analytical analysis tools. Here, RAP may serve as an intermediary between existing analysis tools and techniques; a step towards solving the problem of cross-layer analysis as, e. g., discussed in [24]. This cross-layer concept behind RAP will also be reflected in the implementation of a recent concept for cross-layer reliability analysis presented in [25].

C. Architectural Layer Application Example

Dynamic Functional Verification (DFV) on Coarse Grained Reconfigurable Architectures (CGRA) is a low-cost method to detect faults in SoCs by computing samples of SoC components on a fault tolerant CGRA [26]. The method provides fault detection deadlines which are met with specified probabilities. Specification of these deadlines as well as the desired confidence to meet the latter allows DFV to be optimized for the actual demand. CGRAs support this optimization through temporal and spatial mapping. By mapping these components into the temporal domain, they are deliberately slowed-down by the factor s to only calculate as many samples as are absolutely required to have detection latency DL, the time from fault occurrence to fault detection, meet the deadline with the desired confidence. The usage of fault tolerant CGRAs [27] ensures that the information thereby acquired is reliable.

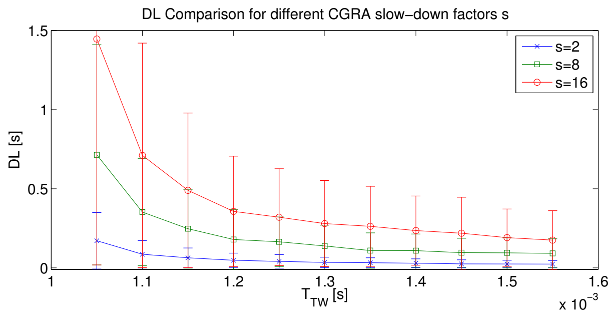


Fig. 7. DL Comparison with standard deviation δ between $s = \{2, 8, 16\}$ and $P(FD) = q = 10^{-5}$

However, with the capability to adjust DFV according to reliability goals, it is important to asses the initial reliability situation correctly. Prior to the RAP Model, this was mostly up to experience and experiments, both which left chances for under- and overestimating the fault occurrence probability $P(FO)$ and thus also its derivative, the fault detection probability $P(FD) = q$, which is limited by $P(FO)$ as upper boundary and which shall be assumed to be equal for simplification. In case of underestimation of $P(FO)$, the risk is deemed lower than it actually is and system stability is jeopardized. Overestimation of $P(FO)$ leads to excess checking and thus to a waste of computing power and energy. The RAP model provides a bit flip probability $\mathcal{P}$, enabling specific optimization for the actual reliability demand, preventing the aforementioned hazardous scenarios. The following example shall elucidate.

Based on the chart in Fig. 7 faults shall be detected within $0.5s$ with a confidence of 95.6%. If $P(FO) > \mathcal{P}$ is overestimated, a setup with CGRA slow-down factor $s = 2$ might be used, using a more resources than necessary. Underestimating $P(FO) < \mathcal{P}$ might lead to a solution using $s = 16$ which would prevent DFV from ever meeting its goal. But if $\mathcal{P}$ is known upfront through the RAP model, all this can be prevented. In this case, the optimization algorithm presented in [26] will suggest a solution of $s = 8$ and a time window of $T_{TW} = 1.30ms$ which will just meet the aforementioned demand.

VII. Summary

This paper presented the basic idea of the RAP model, which is intended to serve as the logical interface point for resilience analysis between lower (technology, circuit, device) levels of abstraction and higher levels of system implementation. The intention behind the development of the RAP model was to allow researchers at all levels of abstraction to be able to clearly and quantitatively describe the error and fault relationships between these levels in terms of probabilistic models and abstraction transformation functions. Thereby, detailed *implementation* and *technology related* aspects of the system are considered via the lower level models. This property allows the designer to globally optimize system resilience across all relevant abstraction levels.

The upper levels of the RAP framework are assigned abstract and meaningful "units of information" to characterize the data and control entities that are typically processed at the respective HW/SW levels. Probabilistic error function $\mathcal{P}_L$ at higher levels can be derived / transformed out of the probabilistic error function describing the lower level bit flips.

Acknowledgment

This research program is supported by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP1500 - spp1500.itec.kit.edu). We would also like to thank all partners within the priority program for their input and feedback.

References

[1] J. Henkel, L. Bauer, J. Becker, O. Bringmann, U. Brinkschulte, S. Chakraborty, M. Engel, R. Ernst, H. Hartig, L. Hedrich *et al.*, "Design and architectures for dependable embedded systems," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2011 Proceedings of the 9th International Conference on*. IEEE, 2011, pp. 69–78.

[2] H. M. Quinn, A. De Hon, and N. Carter, "Ccc visioning study: system-level cross-layer cooperation to achieve predictable systems from unpredictable components," Los Alamos National Laboratory (LANL), Tech. Rep., 2011.

[3] W. Robinson, M. Alles, T. Bapty, B. Bhuva, J. Black, A. Bonds, L. Massengill, S. Neema, R. Schrimpf, and J. Scott, "Soft error considerations for multicore microprocessor design," in *Integrated Circuit Design and Technology, 2007. ICICDT'07. IEEE International Conference on*. IEEE, 2007, pp. 1–4.

[4] A. Evans, M. Nicolaidis, S.-J. Wen, D. Alexandrescu, and E. Costenaro, "Riif-reliability information interchange format," in *On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International*. IEEE, 2012, pp. 103–108.

[5] H.-J. Wunderlich and S. Holst, "Generalized fault modeling for logic diagnosis," in *Models in Hardware Testing*, ser. Frontiers in Electronic Testing, H.-J. Wunderlich, Ed. Springer Netherlands, 2010, vol. 43, pp. 133–155.

[6] B. Becker, S. Hellebrand, I. Polian, B. Straube, W. Vermeiren, and H.-J. Wunderlich, "Massive statistical process variations: A grand challenge for testing nanoelectronic circuits," in *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2010, pp. 95–100.

[7] S. Nassif and O. Fakhouri, "Technology trends in power-grid-induced noise," in *Proceedings of the International Workshop on System-level Interconnect Prediction*, 2002, pp. 55–59.

[8] S. Lee, S. Baeg, and P. Reviriego, "Memory reliability model for accumulated and clustered soft errors," *IEEE Transactions on Nuclear Science*, vol. 58, no. 5, pp. 2483–2492, 2011.

[9] H. T. Nguyen, Y. Yagil, N. Seifert, and M. Reitsma, "Chip-level soft error estimation method," *Device and Materials Reliability, IEEE Transactions on*, vol. 5, no. 3, pp. 365–381, 2005.

[10] P. Hazucha and C. Svensson, "Impact of cmos technology scaling on the atmospheric neutron soft error rate," *IEEE Transactions on Nuclear Science*, vol. 47, no. 6, pp. 2586–2594, 2000.

[11] H. Graeb, *Analog Design Centering and Sizing*. Springer Amsterdam, 2007.

[12] J. Barth, C. Dyer, and E. Stassinopoulos, "Space, atmospheric, and terrestrial radiation environments," *IEEE Transactions on Nuclear Science*, vol. 50, no. 3, pp. 466–482, 2003.

[13] M. Zhang and N. Shanbhag, "Soft-error-rate-analysis (sera) methodology," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,*, vol. 25, no. 10, pp. 2140–2155, 2006.

[14] E. Ibe, S. Chung, S. Wen, H. Yamaguchi, Y. Yahagi, H. Kameyama, S. Yamamoto, and T. Akioka, "Spreading diversity in multi-cell neutron-induced upsets with device scaling," in *Proceedings of Custom Integrated Circuits Conference (CICC)*, 2006, pp. 437–444.

[15] D. Radaelli, H. Puchner, S. Wong, and S. Daniel, "Investigation of multi-bit upsets in a 150 nm technology sram device," *IEEE Transactions on Nuclear Science*, vol. 52, no. 6, pp. 2433–2437, 2005.

[16] S. Baeg, S. Wen, and R. Wong, "Sram interleaving distance selection with a soft error failure model," *IEEE Transactions on Nuclear Science*, vol. 56, no. 4, pp. 2111–2118, 2009.

[17] A. Heinig, V. J. Mooney, F. Schmoll, P. Marwedel, K. Palem, and M. Engel, "Classification-based improvement of application robustness and quality of service in probabilistic computer systems," in *Proceedings of ARCS 2012*, Munich, Germany, Mar. 2012.

[18] S. Rehman, M. Shafique, F. Kriebel, and J. Henkel, "Reliable software for unreliable hardware: Embedded code generation aiming at reliability," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2011, pp. 237–246.

[19] S. Rehman, M. Shafique, and J. Henkel, "Instruction scheduling for reliability-aware compilation," in *Proceedings of Design Automation Conference (DAC)*, 2012, pp. 1288–1296.

[20] S. Z. Shazli and M. B. Tahoori, "Obtaining microprocessor vulnerability factor using formal methods," in *IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, 2008, pp. 63–71.

[21] L. Chen and M. B. Tahoori, "An efficient probability framework for error propagation and correlation estimation," in *IEEE 18th International On-Line Testing Symposium (IOLTS)*, 2012, pp. 170–175.

[22] M. Ebrahimi, L. Chen, H. Asadi, and M. B. Tahoori, "Class: Combined logic and architectural soft error sensitivity analysis," in *Proceedings of 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2013, pp. 601–607.

[23] H. Schirmeier, M. Hoffmann, R. Kapitza, D. Lohmann, and O. Spinczyk, "Fail*: Towards a versatile fault-injection experiment framework," in *ARCS Workshops (ARCS)*, 2012, pp. 1–5.

[24] S. Mitra, K. Brelsford, and P. N. Sanda, "Cross-layer resilience challenges: Metrics and optimization," in *Proceedings of Design, Automation & Test in Europe (DATE)*, 2010, pp. 1029–1034.

[25] M. Glaß, H. Yu, F. Reimann, and J. Teich, "Cross-Level Compositional Reliability Analysis for Embedded Systems," in *Proceedings of the International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, 2012, pp. 111–124.

[26] J. Kühn, S. Eisenhardt, T. Schweizer, T. Kuhn, and W. Rosenstiel, "Improving system reliability using dynamic functional verification on cgras," in *Proceedings of the International Workshop on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART)*, 2012.

[27] T. Schweizer, A. Kuester, S. Eisenhardt, T. Kuhn, and W. Rosenstiel, "Using run-time reconfiguration to implement fault-tolerant coarse grained reconfigurable architectures," in *International Parellel and Distributed Processing Symposium Workshops (IPDPSW)*. Shanghai, China: IEEE, 05 2012.