# Resilience Articulation Point (RAP): Cross-layer Dependability Modeling for Nanometer System-on-chip Resilience

Herkersdorf, Andreas; Aliee, Hananeh; Engel, Michael; Glaß, Michael; Gimmler-Dumont, Christina; Henkel, Jörg; Kleeberger, Veit B.; Kochte, Michael A.; Kühn, Johannes M.; Mueller-Gritschneder, Daniel; Nassif, Sani R.; Rauchfuss, Holm; Rosenstiel, Wolfgang; Schlichtmann, Ulf; Shafique, Muhammad; Tahoori, Mehdi B.; Teich, Jürgen; Wehn, Norbert; Weis, Christian; Wunderlich, Hans-Joachim

Elsevier Microelectronics Reliability Journal Vol. 54(6-7) June-July 2014

doi: http://dx.doi.org/10.1016/j.microrel.2013.12.012

**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, voltage variations 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.

Preprint

# Resilience Articulation Point (RAP): Cross-layer Dependability Modeling for Nanometer System-on-chip Resilience

Andreas Herkersdorf[a], Hananeh Aliee[c], Michael Engel[b], Michael Glaß[c], Christina Gimmler-Dumont[g], Jörg Henkel[d], Veit B. Kleeberger[a], Michael A. Kochte[h], Johannes M. Kühn[f], Daniel Mueller-Gritschneder[a], Sani R. Nassif[e], Holm Rauchfuss[a], Wolfgang Rosenstiel[f], Ulf Schlichtmann[a], Muhammad Shafique[d], Mehdi B. Tahoori[d], Jürgen Teich[c], Norbert Wehn[g], Christian Weis[g], Hans-Joachim Wunderlich[h]

[a] *Technische Universität München*
[b] *Technische Universität Dortmund*
[c] *Universität Erlangen-Nürnberg*
[d] *Karlsruher Institut für Technologie*
[e] *IBM, Austin Research Laboratory*
[f] *Universität Tübingen*
[g] *Technische Universität Kaiserslautern*
[h] *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, voltage variations 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.

*Keywords:* Cross-layer SoC resilience, probabilistic dependability modeling, SRAM error models, critical charge, transient soft errors, permanent aging defects, error abstraction, error transformation, system-level failure analysis, resilience articulation point

## 1. Introduction / Motivation

Nanometer feature size CMOS technologies are susceptible to a variety of dependability threats affecting all abstraction layers of a 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 hardware/software abstraction layers. Consequently, countermeasures have been elaborated to conquer these various error symptoms at each 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-sized transistors 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 typically results in much less area overhead and may achieve the same result.

To effectively tackle these challenges while not compromising any 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

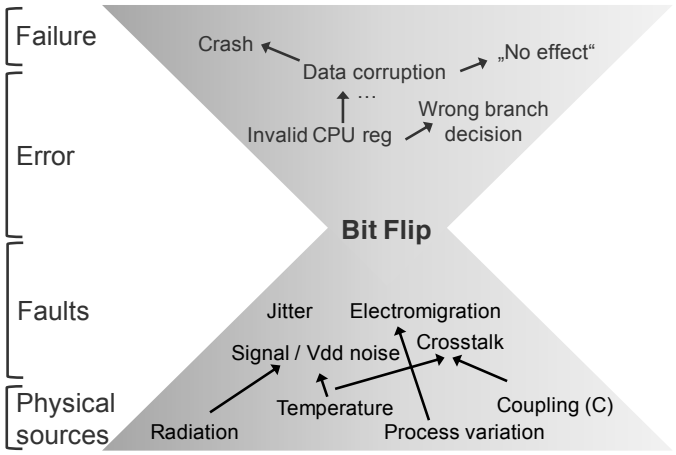


Figure 1: Cross-layer representation of faults, errors, and failures with bit flip as Resilience Articulation Point

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.

## 2. 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. eliminated), it will manifest with a certain probability as a permanent or transient single- or multi-bit signal invalidation, modeled by a probabilistic error function $\mathcal{P}_{\text{bit}}$. 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 (see Fig. 1).

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 error functions 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 3 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 4 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 5 and 6 describe how the RAP bit flip model is propagated towards higher abstraction levels up to the software application layer.

## 3. 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 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 modeling of different components (e. g., SRAM or latches) and different errors (e. g., soft errors or timing violations).

### 3.1. 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].

### 3.2. 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.

### 3.3. 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 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.

### 3.4. 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.

## 4. 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].

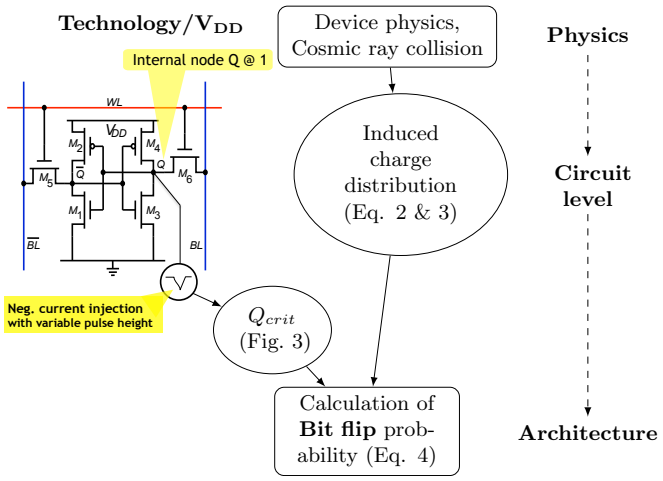


Figure 2: SRAM Single Event Upset Model

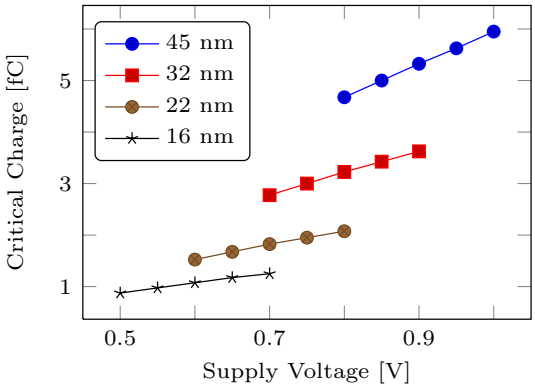


Figure 3: Dependence of $Q_{\text{crit}}$ on $V_{DD}$ for different technologies for a 6T SRAM cell.

### 4.1. SRAM Bit Errors

One common example where bit flips are encountered in a chip is an SRAM cell. We will show in this section how we can model the bit flip probabilities in an SRAM array by using the generic model from Section 3.

A bit flip in an SRAM cell occurs for example when a particle strike induces enough charge on a point within the cell to cause a flip in the cell's content. Thus, a bit flip model for this effect requires the critical charge to flip a cell as well as 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 the critical charge $Q_{\text{crit}}$ on environmental conditions (Fig. 3). 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. This results in a discrete model for the critical charge $Q_{\text{crit}}$ dependent on environmental and design parameters. Fig. 3 shows the $Q_{\text{crit}}$ dependence on the different supply voltages.

Fig. 4 shows the distribution of $Q_{\text{crit}}$ for varying process parameters in a 14 nm FinFET technology for a 6-transistor and

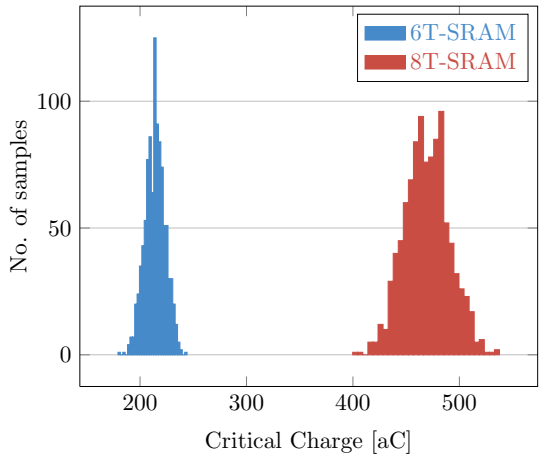


Figure 4: $Q_{\text{crit}}$ distribution for 6T and 8T architectures in 14 *nm* FinFET technology (No. of samples: 1000)

an 8-transistor FinFET technology. According to [11] we modeled mask offsets, line edge roughness, random dopant fluctuations, metal gate granularity, and oxide thickness variations. Obviously, 8T SRAM cells require more injected charge to flip the cell content, which makes them more resilient against soft errors. The obtained distributions can be used to introduce parametric yield as a parameter in the model.

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 from the environment strikes the cell can be modeled by a Poisson process [12]:

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

This equation expresses the probability that the number $N(T)$ neutrons hitting an area $A$ during the time interval $T$ which is exposed to a neutron flux $\Phi$ is $k$. The neutron strike may be followed by the generation of 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]:

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

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

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

With increasing integration density the probability of Multi Bit Upsets (MBU) increases. Possible reasons for this include the successive hit of multiple storage nodes by the same neutron or shared charge to adjacent cells [14]. To correctly account for

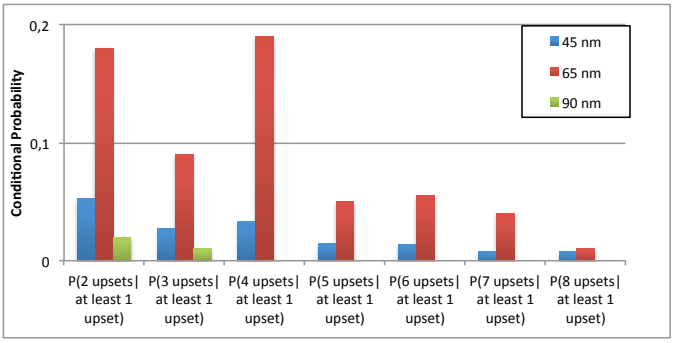


Figure 5: Conditional probabilities for multiple upsets dependent on the first single event upset for different technologies [16, 8].

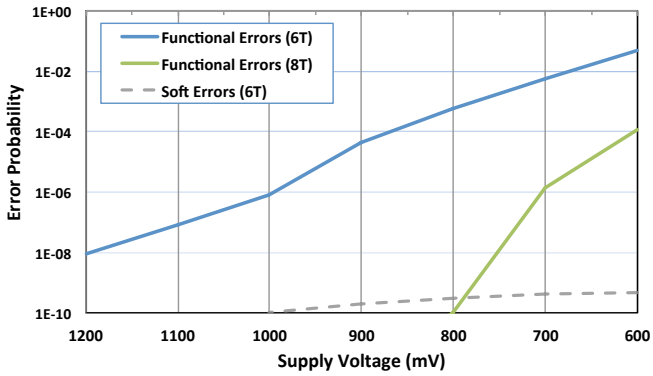


Figure 6: SRAM error probabilities for 6T and 8T cells in the presence of voltage drops for a 65 nm technology. The soft-error rate relates to a storage period of 10 $\mu s$.

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]. Figure 5 shows the conditional probabilities of having $n$ additional upsets given a erroneous bit in an SRAM cell array due to MBU for different technologies.

SRAM cells can also experience intrinsic errors, such as functional or delay errors. The probability of these errors is strongly related to the supply voltage at which the cell is operated. Thus, voltage drops in the system are also a possible source of additional errors. Figure 6 shows these error probabilities for 6T and 8T SRAM cells for a 65 nm technology, which were characterized by extensive Monte Carlo simulations [17].

For comparison the error probability due to soft errors is also shown which is much smaller in this case. Thus, if stringent power requirements exist—as it is for example the case in low power systems—the error rate might be dominated by voltage drop related errors. The error probabilities shown in Fig. 6 can be well approximated for 65 nm by linear or piece-wise linear functions in semi-logarithmic representation [18]:

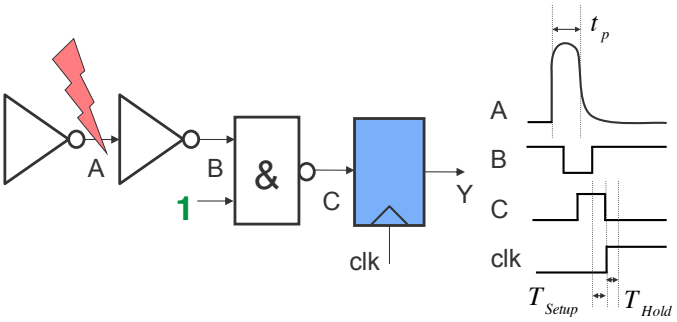


Figure 7: Bit flip as result of SEU or SET.

$$P_{\text{6T,cell fail}} = 10^{-11.7 \cdot V_{DD}/\text{V}+5.6} \tag{5}$$

$$P_{\text{8T,cell fail}} = \begin{cases} 10^{-20 \cdot V_{DD}/\text{V}+7.8} & \text{if } V_{DD} \leq 0.7 \text{ V} \\ 10^{-40 \cdot V_{DD}/\text{V}+21.8} & \text{if } V_{DD} > 0.7 \text{ V} \end{cases} \tag{6}$$

### 4.2. 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. 7).

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 in Equation (1)). 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 $\Delta T_{\text{crit}} = T_{\text{Setup}} + T_{\text{Hold}}$ around the active clock edge. Otherwise, the pulse will be masked out and thus, never be noticed. The probability for a bit error $P_{\text{bit}}(\vec{x}, t)$ 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 $P_{\text{sense}}$ to have a sensitized path to the register, approximated as:

$$P_{\text{bit}}(\vec{x}, t) \approx \frac{T_{\text{Setup}} + T_{\text{Hold}} + t_p}{T_{\text{clk}}} \cdot P_{\text{sense}} \cdot P_{\text{SET}} \tag{7}$$

Signal Y in Fig. 7 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.

## 5. The Upper Half of the Hour Glass

Bits or individual signals are meaningful targets for describing errors at the transistor, logic gate and RT 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. 8). 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 $P_{\text{bit}}(\vec{x}, t)$ in space $\vec{x}$ and time $t$ within memory cells to be independent, one can determine the approximate $P_{\text{word}}(\vec{x}, t)$ error probability by the following concrete transformation function $T_{\text{bit}}$:

$$P_{\text{word}}(\vec{x}, t) = T_{\text{bit}} \circ P_{\text{bit}}(\vec{x}, t) = 1 - \prod_{x_i \in \vec{x}} (1 - P_{\text{bit}}(x_i, t)) \tag{8}$$

The derivation of word error probabilities under consideration of correlated data bits and interleaving is also possible. We refer to [19] for a more complete discussion of this more complex case.

When operand variables of arithmetic operations are stored in an SRAM memory array, then $P_{\text{word}}(\vec{x}, t)$ 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 $P_{\text{word}}(\vec{x}, t)$ describes the trustworthiness of the contents of the register file. The two $P_{\text{word}}(\vec{x}, t)$ probabilities are different because the technological ($D$) and state-related constraints ($S$) of SRAM arrays and a register files are different. $P_{\text{word}}(\vec{x}, t)$ 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 $P_{\text{word}}(\vec{x}, t)$ than a non-protected memory, although both may have the same bit level $P_{\text{bit}}(\vec{x}, t)$.

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 $P_{\text{word}}(\vec{x}, t)$. At higher abstraction layers, the units of words, or compounds of words referred to as interfaces (or data structures), substitute bits or signals. Corresponding interface related probability functions $P_{\text{interface}}(\vec{x}, t)$ are derived from the error probabilities at bit-level $P_{\text{bit}}(\vec{x}, t)$ or word-level $P_{\text{word}}(\vec{x}, t)$, 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 $P_{\text{bit}}(\vec{x}, t)$, $P_{\text{word}}(\vec{x}, t)$ and $P_{\text{interface}}(\vec{x}, t)$.

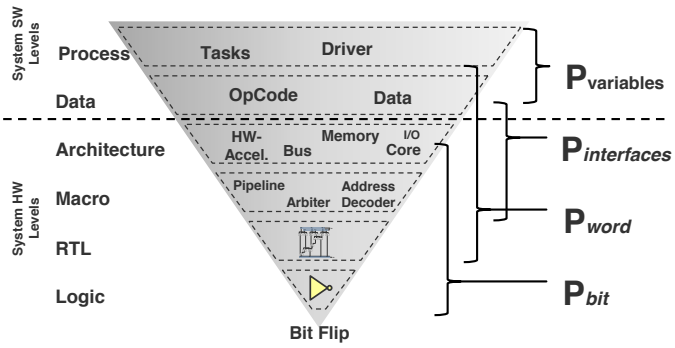


Figure 8: Abstraction and transformation of bit flips in higher system model layers

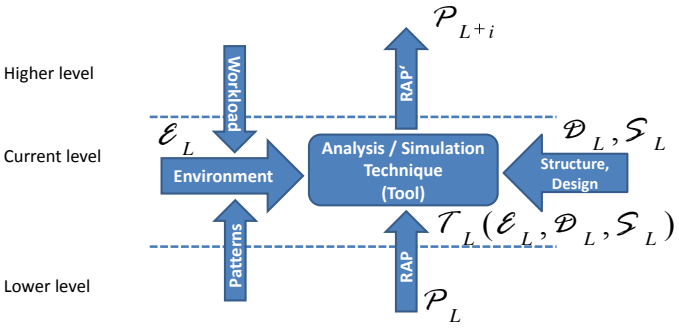


Figure 9: Error transformation / propagation in the upper half layers

### 5.1. 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 (hardware architecture and software layers) can again be investigated without maintaining the complete set of lower layer models. $\mathcal{P}_{\text{word}}$ or $\mathcal{P}_{\text{interface}}$ 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. 9:

$$\mathcal{P}_{L+i} = \mathcal{T}_L(\mathcal{E}_L, \mathcal{D}_L, \mathcal{S}_L) \circ \mathcal{P}_L \tag{9}$$

Transformation functions can stretch one or several abstraction levels. The SRAM data word example from Eq. (8) dealt with two consecutive abstraction levels. Section 6 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 parameters $\mathcal{E}_L$) and, thus, affect the error model at higher abstraction level(s).

## 6. 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.

### 6.1. 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 [20]. 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 $P_{\text{word}}(\vec{x}, t)$ 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) [21, 22] and Instruction Masking Index (IMI) [23] 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. (10)) quantifies the spatial and temporal vulnerabilities of different types of software instructions in different microarchitecture/RT-level pipeline stages $c \in C$ of a given processor according to their area $A_c$ and error probability $P_E(c)$ [21]. 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 $c$ is obtained using the HW-level reliability methods like EPP [24], CEP [25] and CLASS [26]. These techniques provide probabilistic analysis of error propagation from error site ($P_{\text{bit}}(\vec{x}, t)$ or $P_{\text{word}}(\vec{x}, t)$) 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 a particle strike at internal nodes. $A_c$ is obtained from the hardware synthesis results.

$$IVI_i = \frac{\sum_{\forall c \in C} IVI_{ic} \times A_c \times P_E(c)}{\sum_{\forall c \in C} A_c} \tag{10}$$

$IVI_{ic}$ denotes the vulnerability at a processor component $c$ (with an architecturally-defined size $\beta_c$) is given as the product of its vulnerable periods in that processor component ($\upsilon_{ic}$) and vulnerable bits affecting the Correct Execution ($\beta_{c(v)}$), as shown in Eq. (11). $A_c$ and $\beta_{c(v)}$ capture the spatial vulnerability, while $\upsilon_{ic}$ captures the temporal vulnerability.

$$IVI_{ic} = \frac{\upsilon_{ic} \times \beta_{c(v)}}{\sum_{\forall c \in C} \beta_c} \tag{11}$$

$\beta_{c(v)}$ is obtained using the program-level analysis of vulnerable bits [21], bit error probabilities (Eq. (12)) and their correlations [25][26]. 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.

$$P_{\text{bit AVG}}(i \mid j) = \frac{P_{\text{bit AVG}}(ij)}{P_{\text{bit AVG}}(j)} \tag{12}$$

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.

### 6.2. 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 Section 4, 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 [27]. 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 Section 5.1, 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

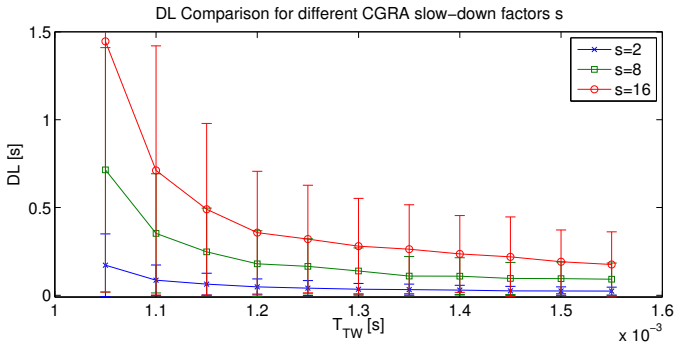


Figure 10: Write detection latency ($DL$) comparison with standard deviation $\delta$ between $s = \{2, 8, 16\}$ and $P(FD) = q = 10^{-5}$

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 [28]. This cross-layer concept behind RAP will also be reflected in the implementation of a recent concept for cross-layer reliability analysis presented in [29].

### 6.3. Architectural Layer 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 [30]. 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 [31] ensures that the information thereby acquired is reliable.

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. 10 faults shall be detected within $0.5s$ with a confidence of 95.6%. If $P(FO)$ is overestimated ($P(FO) > \mathcal{P}$), a setup with CGRA slow-down factor $s = 2$ might be used, using a more resources than necessary. Underestimating $P(FO)$ ($P(FO) < \mathcal{P}$) 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 [30] will suggest a solution of $s = 8$ and a time window of $T_{TW} = 1.30\ ms$ which will just meet the aforementioned demand.

### 6.4. System-level Analysis Examples

As outlined in Section 5.1, transformation functions are an integral part of the RAP concept to enable the propagation of low-level effects up to the highest level of abstraction. Therefore, RAP can be used to enhance reliability studies of full application systems. Utilizing RAP in this context allows to draw connections between occurring effects in an application system and the underlying fault mechanisms.

#### 6.4.1. Distributed Embedded Control Application

Often, it is of interest to not only consider a single source of unreliability but also other sources such as aging-based permanent faults to decide which effects have the highest impact and which counter measures to apply. A recent approach presented in [32] relies on transformation functions from lower levels, see Section 6.1, and considers soft errors at the level of tasks, i. e., $P_{\text{task}}(\vec{x}, t)$ as well as permanent defects of components as derived, e. g., in [29]. The automatic reliability analysis can be seen as the top-level transformation function to deliver what could be described as $P_{\text{system}}(\vec{x}, t)$. Based on success trees, a variant of the well-known fault trees, the proposed method not only considers multiple transient and permanent faults concurrently, but a carefully introduced structure of the success tree enables to track a system failure back to the critical effect. The result of such an analysis for a distributed embedded control application, cf. [33], is depicted in Figure 11. Shown is the classic reliability function, denoting the probability that the system works correctly until a respective point in time, as well as the ratio of permanent and transient faults with respect to being the critical effect that caused the system failure. In the beginning, most components work properly such that soft-errors are the dominant source of failure. Over time and with aging effects taking place, more and more components tend to be permanently defective, making permanent effects the dominant source of failure. As can be seen, the varying impact of the different effects over the system's lifetime can be investigated, enabling a careful and efficient application of counter measures with respect to the real impact of an effect and the targeted main mission time of a system.

#### 6.4.2. Autonomous Robot

Autonomous vehicles can revolutionize public and industrial transportation systems. For these systems often safety and reliability are a special concern, because of their autonomous nature – especially when the operate with humans in the same

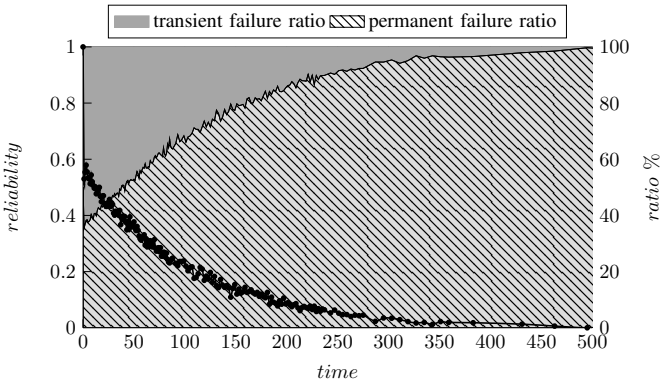


Figure 11: Analysis results of an embedded control application: The reliability function and the ratio of system failures induced by transient effects of the software tasks or permanent effects of the hardware over the lifetime of the system. In the beginning, most hardware components are fault free such that transient effects are a significant cause of system failure. Over time, the aging and wear-out results in permanent component faults, of course, reducing the impact of transient effects since there are less and less tasks being executed.

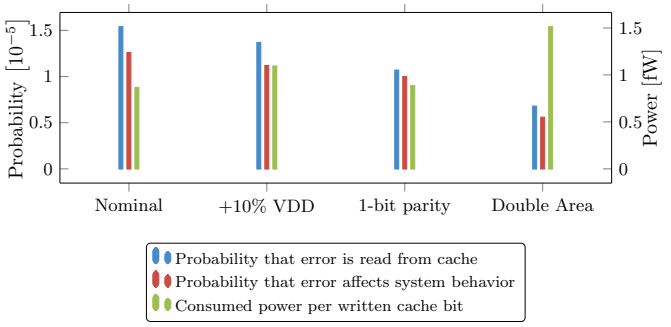


Figure 12: Trade-off between failure probability and power for different protection solutions for the data cache of an autonomous robot for a system runtime of 10 seconds.

environment. In [34], we studied the effect of soft errors in the data cache of an two-wheeled autonomous robot. Using a fault model as it was presented in Section 4.1, faults can be injected in the simulation environment. The robot in this simulation was modeled in SystemC/TLM and its environment in Java. To make the fault injection experiment feasible we used a Mixture Importance Sampling approach to simulate only relevant scenarios. Using this approach the fault probability of the whole system can be estimated with high confidence within 1,500 samples. This equals a cumulative simulation time of 2.5 days. We utilized this approach to test the efficiency of different protection solutions for the data cache in [18].

Figure 12 shows the system failure probability (i.e., the robot makes a failure in its movement), the probability that a erroneous bit is read from the cache, and the consumed power per written cache bit for different protection solutions. The probabilities were estimated for a system runtime of 10 seconds. Protecting the cache by hardening the cells with increased cell area or supply voltage reduces the probability that an erroneous bit is read from the cache. The system failure probability is similarily influenced. In contrast, the addition of a 1-bit parity protection with write-through mode behaves differently. With a

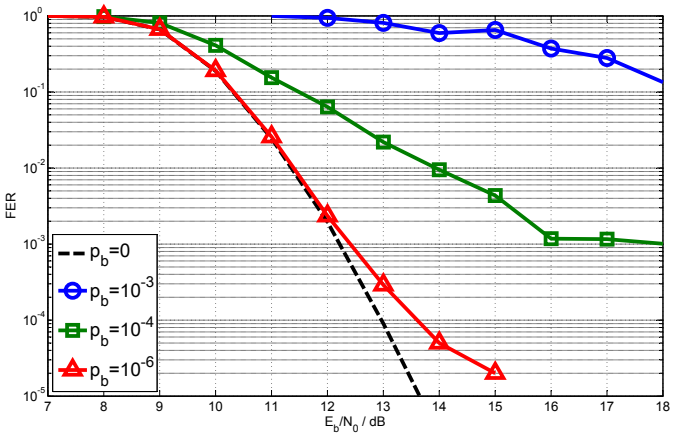


Figure 13: The system communication performance is gradually decreasing for random bit-flips in the channel information memory. $p_b$ depicts the bit error probability.

parity protection errors inside the cache occur at the same probability as for the unprotected case, but only data words from the cache with an even number of errors contribute to system failures. Data words with an odd number of errors are fetched at the penalty of a cache miss from the better protected memory. According to Fig. 12, the probability that a faulty data word is read from the cache decreases as expected, while the overall system failure probability only slightly decreases compared to the unprotected cache. Thus, we can conclude that most of the protected data words (i.e., with an odd number of errors) are anyway masked by the system architecture. The usage of a fault model directly derived from the technology level provides here a possibility to make trade-offs between power and reliability while still being able to evaluate the efficiency of different protection solutions for the overall system.

#### *6.4.3. Iterative MIMO-BCIM Detector*

Multiple-antenna or MIMO systems have the potential to increase the data rate of wireless communication systems. They belong to the most advanced systems in the upcoming 4G and 5G communication standards, and their very high complexity is a challenge for any hardware implementation. In [35], we studied the effects of hardware errors in the system memories of a MIMO-BICM receiver on the system's communications performance because the memories consume a large amount of the systems area. We found out that especially the memories containing complex-valued data, i.e. the channel information and the received vectors, are very sensitive. Figure 13 shows the degradation of the communications performance when errors are injected in the channel information memory. Up to a bit error probability of $p_b = 10^{-6}$ the degradation is negligible for the typical frame error rates (FERs) of a wireless system. Afterwards, the performance decreases gradually with an increasing $p_b$.

We assume that the memory errors result from supply voltage drops which occur regularly during power state switching. According to Equations (5) and (6) each bit error probability $p_b$ corresponds to a specific voltage supply value. For instance a $p_b$ of $10^{-4}$ translates to a voltage value of 820 $mV$ for a 6T cell and 590 $mV$ for a 8T cell architecture. Several resilience actuators exist which can be applied for different degrees of hardware unreliability in order to mitigate the impact of the hardware errors on the system performance [18]. No action has to be taken as long as there is a high hardware reliability, i.e., voltage drops of no more than 200 $mV$. Within this area, the receiver shows an inherent algorithmic error resilience. For a decreased reliability in which voltage drops up to 300 $mV$ occur, we can react on the application layer by increasing the number of iterations in order to regain communications performance. For transient errors, this leads only to a temporary throughput degradation without loss of communications performance. When errors occur with a high probability $p_b > 5 \cdot 10^{-5}$, application-layer resilience actuators cannot provide the necessary resilience. On the architectural layer, the contents of the memory can be protected by a simple 1-bit error correction code. The resilience can be even further increased on technology layer by employing 8-transistor (8T) memory cells instead of 6-transistor (6T) cells resulting in a smaller implementation overhead. 8T memory cells can even tolerate voltage drops of 500 $mV$. However, the increase in area and power is in both cases permanent.

## 7. 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 functions $\mathcal{P}_L$ 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., De-

sign and architectures for dependable embedded systems, in: International Conference onHardware/Software Codesign and System Synthesis (CODES+ ISSS), IEEE, 2011, pp. 69–78.

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

[3] W. Robinson, M. Alles, T. Bapty, B. Bhuva, J. Black, A. Bonds, L. Massengill, S. Neema, R. Schrimpf, J. Scott, Soft error considerations for multicore microprocessor design, in: IEEE International Conference on Integrated Circuit Design and Technology, IEEE, 2007, pp. 1–4.

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

[5] H.-J. Wunderlich, S. Holst, Generalized fault modeling for logic diagnosis, in: H.-J. Wunderlich (Ed.), Models in Hardware Testing, Vol. 43 of Frontiers in Electronic Testing, Springer Netherlands, 2010, pp. 133–155.

[6] B. Becker, S. Hellebrand, I. Polian, B. Straube, W. Vermeiren, 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. doi:10.1109/DSNW.2010.5542612.

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

[8] S. Lee, S. Baeg, P. Reviriego, Memory reliability model for accumulated and clustered soft errors, IEEE Transactions on Nuclear Science 58 (5) (2011) 2483–2492.

[9] H. T. Nguyen, Y. Yagil, N. Seifert, M. Reitsma, Chip-level soft error estimation method, IEEE Transactions on Device and Materials Reliability 5 (3) (2005) 365–381.

[10] P. Hazucha, C. Svensson, Impact of CMOS technology scaling on the atmospheric neutron soft error rate, IEEE Transactions on Nuclear Science 47 (6) (2000) 2586–2594.

[11] V. B. Kleeberger, H. Graeb, U. Schlichtmann, Predicting future product performance: Modeling and evaluation of standard cells in FinFET technologies, in: ACM/IEEE Design Automation Conference (DAC), 2013, pp. 33:1–33:6.

[12] J. Barth, C. Dyer, E. Stassinopoulos, Space, atmospheric, and terrestrial radiation environments, IEEE Transactions on Nuclear Science 50 (3) (2003) 466–482.

[13] M. Zhang, N. Shanbhag, Soft-error-rate-analysis (sera) methodology, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 25 (10) (2006) 2140–2155.

[14] E. Ibe, S. Chung, S. Wen, H. Yamaguchi, Y. Yahagi, H. Kameyama, S. Yamamoto, 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, S. Daniel, Investigation of multi-bit upsets in a 150 nm technology sram device, IEEE Transactions on Nuclear Science 52 (6) (2005) 2433–2437.

[16] G. Georgakos, P. Huber, M. Ostermayr, E. Amirante, F. Ruckerbauer, Investigation of increased multi-bit failure rate due to neutron induced seu in advanced embedded srams, in: IEEE Symposium on VLSI Circuits, IEEE, 2007, pp. 80–81.

[17] I. Chang, D. Mohapatra, K. Roy, A priority-based 6T/8T hybrid SRAM architecture for aggressive voltage scaling in video applications, Transactions on Circuits and Systems for Video Technology 21 (2) (2011) 101–112.

[18] V. B. Kleeberger, C. Gimmler-Dumont, C. Weis, A. Herkersdorf, D. Mueller-Gritschneder, S. R. Nassif, U. Schlichtmann, N. Wehn, A cross-layer technology-based study of how memory errors impact system resilience, IEEE Micro 33 (4).

[19] S. Baeg, S. Wen, R. Wong, SRAM interleaving distance selection with a soft error failure model, IEEE Transactions on Nuclear Science 56 (4) (2009) 2111–2118.

[20] A. Heinig, V. J. Mooney, F. Schmoll, P. Marwedel, K. Palem, M. Engel, Classification-based improvement of application robustness and quality of service in probabilistic computer systems, in: Proceedings of ARCS 2012, Munich, Germany, 2012, pp. 1–12.

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

[22] S. Rehman, M. Shafique, J. Henkel, Instruction scheduling for reliability-aware compilation, in: ACM/IEEE Design Automation Conference (DAC), 2012, pp. 1288–1296.

[23] M. Shafique, S. Rehman, P. V. Aceituno, J. Henkel, Exploiting program-level masking and error propagation for constrained reliability optimization, in: ACM/IEEE Design Automation Conference, 2013, pp. 17:1–17:9.

[24] S. Z. Shazli, 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.

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

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

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

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

[29] M. Glaß, H. Yu, F. Reimann, J. Teich, Cross-level compositional reliability analysis for embedded systems, in: International Conference on Computer Safety, Reliability and Security (SAFECOMP), 2012, pp. 111–124.

[30] J. Kühn, S. Eisenhardt, T. Schweizer, T. Kuhn, W. Rosenstiel, Improving system reliability using dynamic functional verification on CGRAs, in: International Workshop on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART), 2012.

[31] T. Schweizer, A. Kuester, S. Eisenhardt, T. Kuhn, W. Rosenstiel, Using run-time reconfiguration to implement fault-tolerant coarse grained reconfigurable architectures, in: International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, Shanghai, China, 2012, pp. 320–327.

[32] H. Aliee, M. Glaß, F. Reimann, J. Teich, Automatic success tree-based reliability analysis for the consideration of transient and permanent faults, in: Design, Automation, and Test in Europe (DATE), 2013, pp. 1621–1626.

[33] M. Glaß, J. Teich, L. Zhang, A co-simulation approach for system-level analysis of embedded control systems, in: International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (IC-SAMOS 2012), 2012, pp. 355–362.

[34] V. B. Kleeberger, D. Mueller-Gritschneder, U. Schlichtmann, Technology-aware system failure analysis in the presence of soft errors by mixture importance sampling, in: IEEE Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems, 2013.

[35] C. Gimmler-Dumont, C. Brehm, N. Wehn, Reliability study on system memories of an iterative mimo-bicm system, in: International Conference on VLSI and System-on-Chip, IEEE, 2012, pp. 255–258.