Connecting Different Worlds – Technology Abstraction for Reliability-Aware Design and Test

Ulf Schlichtmann*, Veit B. Kleeberger*, Jacob A. Abraham[§], Adrian Evans[¶], Christina Gimmler-Dumont[†], Michael Glaß[‡], Andreas Herkersdorf*, Sani R. Nassif[‖], and Norbert Wehn[†]
*TU München [†] TU Kaiserslautern [‡] FAU Erlangen-Nürnberg
[§] UT Austin [¶] iROC Technologies [‖] IBM Research

***Abstract*—The rapid shrinking of device geometries in the nanometer regime requires new technology-aware design methodologies. These must be able to evaluate the resilience of the circuit throughout all System on Chip (SoC) abstraction levels. To successfully guide design decisions at the system level, reliability models, which abstract technology information, are required to identify those parts of the system where additional protection in the form of hardware or software countermeasures is most effective. Interfaces such as the presented Resilience Articulation Point (RAP) or the Reliability Interchange Information Format (RIIF) are required to enable EDA-assisted analysis and propagation of reliability information. The models are discussed from different perspectives, such as design and test.**

I. Background and Motivation

Reliability has always been a major concern in digital IC design. When a System on Chip (SoC) fails to meet necessary reliability targets, the consequences can be severe. For example, just recently, EETimes reported that most probably a single unprotected bit flip caused a car accident which resulted in the death of a person [1].

Traditionally, the responsibility of ensuring proper system reliability essentially fell on manufacturing process technology. While process innovations such as high-k dielectrics, metal gates, or FinFET transistors put emphasis on reliability, the burden of ensuring system reliability increasingly must be addressed during the system design process. Today, this required reliability typically is achieved by general safety margins and overdesign during the design process. By trading off area, power and performance, design techniques can be used to mitigate imperfections at the process level.

Unfortunately, scaling evolved quicker than our capability to economically maintain reliability problems at a low level [2]. Hence, in order to obtain hardware which can be trusted to be fully reliable, increased margins were added during the design process. As a result, new process technologies offered only diminished benefits to product designers. One approach to mitigate this trend and improve design techniques is to minimize the amount of design margins that are applied. Examples for these improvements include: use of realistic worst-case design instead of corner-based design [3], statistical extensions to timing analysis [4], or optical proximity correction in mask generation [5].

But still, there were points in the design, where margins and technology improvements alone were not enough to meet the required specification. Especially in memories, which are the most susceptible parts of the design, many more improvements were needed. Hence, additional components were added to raw memory, such as modular redundancy, row and column replacements, and error correcting codes (ECC) [6].

While all of these techniques and improvements helped designers in the past to maintain reliable hardware, we are now facing a point where these improvements alone may not be enough anymore. In fact, the circuit's transistors start to become extremely unreliable and maintaining this conservative design method might become again too costly [7,8]. In addition, this problem affects all kinds of applications, from exascale computing [9] to embedded systems [10].

To overcome these problems, the task of ensuring resilience has to be distributed over the full hardware/software stack. Each abstraction level has its own techniques for increasing reliability. On the lower levels, technology innovations, overdesign, and use of margins helps to ensure reliability. Moving up the abstraction stack, other techniques, such as ECC or redundancy are available. The variety of protection and fault-tolerance mechanisms increases even further when moving to the software level, where techniques such as checkpointing [11] or control-flow checking [12] exist.

Each of these techniques has its own associated cost and benefits in terms of power, performance, area, and resulting improvement in reliability. Additionally, each abstraction level also offers some inherent resilience. For example, not all instructions are equally vulnerable to errors. In a branch decision, only the least significant bit matters; in an addition or multiplication, the most significant bit might be the most vulnerable; and in an OR operation, all bits might have equal influence. Putting the processed data in an application context adds further masking mechanisms. For example, in multimedia, some additional noise on the processed patterns might still be tolerable [13].

978-3-9815370-2-4/DATE14/©2014 EDAA

II. Cross-Layer Reliability

Cross-layer approaches have been suggested as desirable techniques to enhance the reliability of complex systems [14,15]. In recent years, several authors outlined major challenges for resilient designs and proposed approaches to cross-layer reliability [16–19]. In order to turn cross-layer reliability into a widely-accepted, mainstream solution, several important challenges have to be addressed.

Today, information about an IC's reliability status is essentially still contained at lower technology and circuit-levels, where a deep understanding for the ongoing technology effects and possible mitigation techniques exists. Exposing these problems to higher levels introduces several major challenges. Information about reliability issues that was gained at the technology level has to be abstracted and relevant information must be made available to higher levels. At the same time designers at higher abstraction levels (e.g. SW developers) must be given some knobs which they can use to tune a circuit's or system's reliability.

One central issue is to define the correct interface to propagate reliability information up in the design hierarchy while considering important information for influencing these reliability issues. This interface has to be designed in a way that all important aspects of reliability can be exposed to higher levels. For example, recent studies show that errors and failures have a certain degree of temporal and spatial correlation in real-world systems. Thus, this kind of information has to be handled by the selected interface which is used for reliability information propagation [20,21].

Cho et al. studied the accuracy of fault injection methods on different abstraction levels in [22]. They concluded that the estimation results of high-level fault injection methods might be inaccurate by up to a factor of 45 if they do not rely on realistic distributions gained from lower levels. Even more severely, without the proper fault distribution, these methods have no guarantee to be either optimistic or pessimistic.

Standardization is another major issue for mainstream adoption of cross-layer reliability. For widespread adoption, commercial CAD support is essential. Such support requires standardized file formats and languages to express the reliability information. This is essential in order to distribute the reliability analysis across the design flow and propagate information across the hierarchy.

In the remainder of this paper, Section III introduces the *Resilience Articulation Point* (RAP) model as an interface for propagation of reliability information between different abstraction levels. Section IV provides details on the *Reliability Interchange Information Format* (RIIF), which deals with format standardization for reliability information exchange. Section V provides real-world examples for the application of these methods. Section VI discusses design and test perspectives on these challenges and proposed solutions. Finally, Section VII concludes the paper.

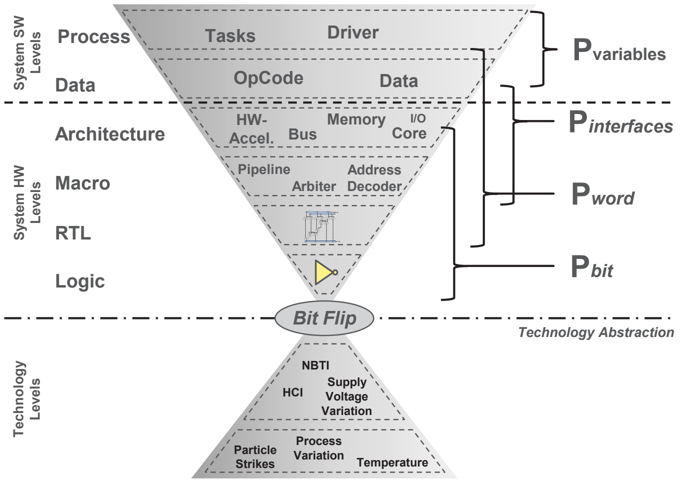

Fig. 1. Cross-layer representation of faults, errors, and failures with bit flip as Resilience Articulation Point

III. Resilience Articulation Point (RAP)

The RAP model [23,24] is based on the following three concepts: (I) It is assumed 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 bit error, modeled by a probabilistic error function $\mathcal{P}_{\text{bit}}$. (II) To efficiently cope with, abstract from, and quantify the impact of lower-level faults onto higher abstraction levels of complex SoCs, the cross-layer reliability modeling and optimization techniques should follow probabilistic methods. (III) 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 the world of resilience evaluation within SoC models, bit flips represent the single connecting vertex between lower-layer fault origins and the upper layer error and failure models of the HW/SW system abstraction (see Fig. 1). Error functions for different fault origins (e.g. radiation, aging, crosstalk, or thermal hotspots) 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 in the scope nor can it be the intention of the RAP model to consider all possible error and transformation functions to be an integral part of RAP. RAP rather provides a framework where different fault origins, each being expressed as probabilistic bit-error functions for a particular functional signal, can be accumulated to represent an error function that covers several physical shortcomings. By approximation and accumulation of individual lower-level error models, RAP provides effective means for technology abstraction at different SoC abstraction levels. This relieves the SoC designer with expertise at higher abstraction levels from the necessity to comprehend the

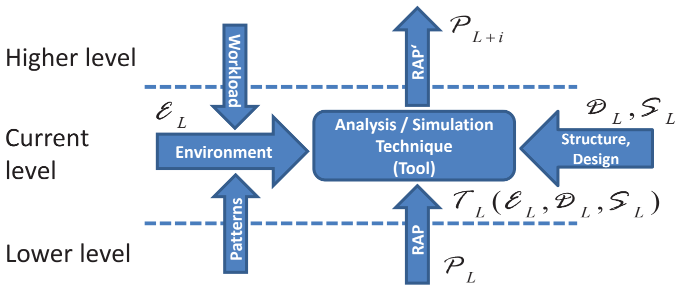

Fig. 2. Error transformation / propagation in the upper half layers

details of the SoC at the device or even technological level.

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 [25] allows the 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 at quantifying the quality of the test and final product w.r.t. the parameter space in spite of high uncertainty of variations [26].

The task of an error function $\mathcal{F}$ at the lower abstraction levels is to describe the probability of an occurring bit error $\mathcal{P}$ as a function of environmental and operating conditions $\mathcal{E}$, design parameters $\mathcal{D}$, and (error) state bits $\mathcal{S}$; all of which are parameters that may change during system design and run time:

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

This generic model for an error function is unique for a specific type of fault and for a specific circuit element. Hence, it has to be adapted to every circuit component and fault type independently.

Probabilistic error models and transformation functions that are specific for a certain level of abstraction can now be used for propagating error models towards higher abstraction levels. Mathematically, this can be expressed by the following equation, graphically depicted in Fig. 2:

$$\mathcal{P}_{L+i} = \mathcal{T}_L(\mathcal{E}_L, \mathcal{D}_L, \mathcal{S}_L) \circ \mathcal{P}_L \tag{2}$$

Once we are able to describe the dependability exposure of a complex SoC at different abstraction levels by corresponding probabilistic functions $\mathcal{P}_L$ affecting, e.g., 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 to realize a sound technology abstraction. $\mathcal{P}_{\text{word}}$ or $\mathcal{P}_{\text{interface}}$ are adequate representatives of the lower layer errors. An analysis approach that follows a similar concept of cross-layer error propagation is presented in [27].

Thereby, transformation functions can stretch one or several abstraction levels: From bit to architecture level (e.g. SRAM bit flip) and register word to application software level (e.g. application code variable), 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 typically contribute to the environmental dimension, while design structures and templates on to the respective abstraction levels constitute 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).

IV. Reliability Information Interchange Format (RIIF)

The problem of modeling how transistor-level failures affect the operation of a complex SoC is challenging. It is important to note that this is primarily an industrial problem and thus, beyond the purely technical modeling challenges, certain practical issues must be seriously considered in order to provide a solution that will gain adoption. First, the approach must be supported by standards and a working file-format so that the companies involved in the design of an SoC can exchange reliability data effectively and develop tools that process the data. Today, the majority of silicon reliability analysis is done using spreadsheets which basically perform a sum of computed or estimated failure rates. Change will occur only if there is an economic incentive to adopt a new approach. Either the reduced effort due to a more streamlined methodology, or the potential for improved accuracy and reduced design margins, must be sufficient to incite the industry to evolve beyond the status quo.

RIIF is an application-specific language targeted at the problem of modeling failure propagation in SoCs. *RIIF* was first proposed in [28] and was further developed during a dedicated workshop at DATE'13 [29]. The goals of the *RIIF* language are:

- Enumerate and specify probability of failure events
- Relate the effect of environment (voltage, temperature) on failure rates
- Build composite models using simpler models
- Scale from the transistor-level to the system-level
- Provide a standard means for sharing reliability data
- Provide templates for standard classes of components
- Specify reliability targets that must be met

The basic unit in *RIIF* is the *component* which is similar to a Verilog module (VHDL entity), and can represent either a low-level entity such as a logic gate or a complex entity like a full SoC. With *RIIF*, only the faults are modeled, not the functionality. The user declares *failure_modes* within the *components*. Typical *failure_modes* would be a single bit error (SBE) in a memory component or a silent data corruption (SDC) in a CPU. The rate of occurrence of these events is expressed as a function of the parameters within the *component*, where typical parameters would be voltage or temperature for a transistor or the type of workload being executed for a CPU. A complex component can be modeled by referencing the failure rates in the lower level child components and expressing how those failure rates (for example a SBE or MBE in a RAM) propagate into high-level failure modes (such as SDC in a CPU). The equations that express the propagation take into account the mitigating effects of masking (e.g. AVF) or error correction.

The *RIIF* approach is pragmatic and industry focused. A file-format is proposed based on a defined grammar. Real-world issues have been considered. For example, there are many families of commodity components (e.g. DDR DRAMs, Flash, etc.) and users expect the reliability models from different vendors to be plug-and-play. The *RIIF* language offers an approach to templating, so that a basic model can be defined for a category of components. Each implementation can extend the template to define the failure rates in that device, however, the users benefit from a pre-defined interface defined in the template.

To conclude, there is a need for improved approaches for modeling the way faults propagate within complex SoCs. Fundamentally, this is a modeling problem and requires a flexible, but standardized approach, for building and exchanging models. Such an approach needs to be inclusive of currently known failure mechanisms in CMOS and easily extendible to new failure mechanisms in future implementation technologies.

V. Application Examples

This section provides three application examples, which deal with the idea of cross-layer reliability. In the first two examples the idea of the RAP framework is demonstrated using two system-level applications – an autonomous robot and a MIMO receiver. The last example demonstrates the RIIF reliability modeling language for a protected SRAM.

A. Autonomous Robot

Autonomous vehicles have a great potential in future public and industrial transportation systems. Because of their autonomous nature, safety and reliability are often a special concern for these systems – especially when they operate with humans in the same environment. In [30] a fault injection method based on Mixture Importance Sampling, which relies on the RAP principle (Fig. 1), is presented. This method is suitable to evaluate the effect of technology-level faults in a full system simulation. We utilized this approach in [31] to study the effect of neutron-induced soft errors in the data cache of a two-wheeled autonomous robot.

Figure 3 shows the system failure probability (i.e., the robot makes a failure in its movement), the probability that an erroneous bit is read from the cache, and the percentage of errors that are masked by the application intrinsic resilience. The probabilities were estimated for a system runtime of 10 seconds. Protecting the cache by

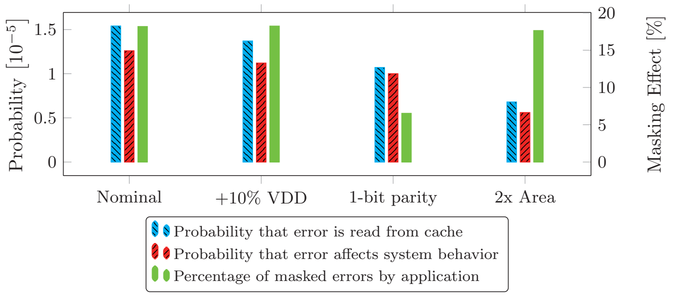

Fig. 3. Reliability improvement in different data cache protection solutions.

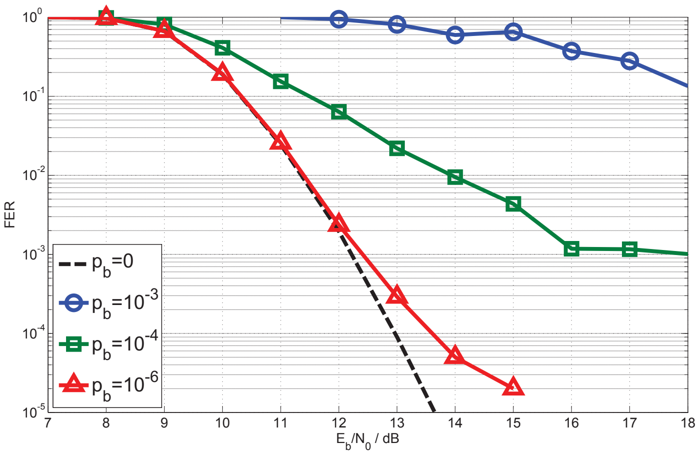

Fig. 4. System communication performance for different bit error probabilities p_b in the channel information memory.

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 similarly influenced. Hence, nominal and hardened caches have a similar error-masking behavior on application level, although providing overall different system failure probabilities. In contrast, the addition of a 1-bit parity protection with write-through mode behaves differently. With a 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. 3, 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. This results in a lower percentage of application error masking. This leads to a case where we overprotect the data cache, as we protect many errors which would be anyway masked by the system architecture. The usage of a fault model directly derived from the technology level provides here a possibility to gain such insights and make decisions which are optimal for the protection of the overall system.

B. Iterative MIMO-BICM Receiver

Multiple-input multiple-output (MIMO) systems are used to increase the data rate of wireless communication systems. They belong to the most advanced systems in the next generation communication standards. Their implementation complexity is challenging. Up to 50% of the receiver area are consumed by different system memories which store the data between the processing elements. We studied the influence of memory errors on the system-level communications performance in [32]. We assumed that the memory errors result from supply voltage drops, which occur regularly during power state switching. According to the RAP model shown in Fig. 1, we model the influence of changes in the supply voltage V_{DD} as a bit-flip probability $P_{cell\,fail}$ in the memory cells. This influence can be modeled for 6-transistor (6T) and 8-transistor (8T) memory cells according to the following equations [31,33]:

$$P_{6T\,cell\,fail} = 10^{-11.7(1/V)*V_{DD}+5.6} \tag{3}$$

$$P_{8T\,cell\,fail} = \begin{cases} 10^{-20(1/V)\cdot V_{DD}+7.8} & \text{if } V_{DD} \leq 700mV \\ 10^{-40(1/V)\cdot V_{DD}+21.8} & \text{if } V_{DD} > 700mV \end{cases} \tag{4}$$

Referring to Fig. 2, equations (3) and (4) represent the RAP from the lower technology level. By means of system simulations, we can propagate this information to the next level in the form of a system frame error rate (FER). The simulations take the system architecture into consideration. Thereby, memories containing complex-valued data have shown the highest susceptibility. The memory for the channel information belongs to this group of critical memories. Figure 4 shows the system FER for different bit error probabilities p_b in this memory. The system shows an algorithmic error resilience for $p_b \leq 10^{-6}$. For larger error rates, the performance is gradually decreasing.

Several resilience actuators exist which can mitigate the effect of hardware errors on the system performance. We analyzed the robustness of different resilience actuators against voltage drops in [31]. No action has to be taken as long as there is a high hardware reliability, i.e. voltage drops of no more than 200mV. For a decreased reliability in which voltage drops up to 300mV occur, we can react on the application level 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-level resilience actuators cannot provide the necessary resilience. On the architectural level, 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 level 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 500mV. However, the increase in area and power is in both cases permanent.

C. RIIF Protected SRAM Model

Correctly modeling the effective failure rate of even a simple system can quickly become complicated when multiple failure modes are considered and when mitigation techniques mask some of the errors. In [34], Sánchez-Macian et al. developed a RIIF model for an SRAM with

```
component SRAM; // SRAM with two failure modes
 // Design Parameters
 parameter SIZE: int := 512 * 1024 * 1024;
 parameter WIDTH : int := 32;
 parameter M = ( SIZE / WIDTH );
 // Operating Parameters
 parameter TEMPERATURE: float;
 parameter VOLTAGE : float;
 // Failure modes
 fail_mode : SBU; // soft error
 assign SBU'unit = FITS;

 fail_mode : SHE; // hard error
 assign SHE'unit = FITS;
endcomponent
...
component SEC_PROTECTED_SRAM extends SRAM;
 parameter T_SCRUB : time; // scrub interval
 assign T_SCRUB=express_time("hours",24);
 parameter BUILD_T : time; // manufacturing time
 parameter POWERUP_T : time; // power−up time
 fail_mode SDC; // data corruption
 assign SDC'rate = ( pow(SBU'rate,2) *
     get_time_since( POWERUP_T, "hours" )/M +
    pow(SHE'rate,2) *
       get_time_since (BUILD_T, "hours" )/M +
   (SBU'rate * SHE'rate / M) * (
      get_time_since( POWERUP_T, "hours" ) +
          get_time_since( BUILD_T, "hours" ) ) );
...
endcomponent
```

Fig. 5. RIIF SRAM Model

M words. The model assumes that the memory is subject to transient single bit upsets (SBUs), at a rate of λ_{SBU} and to permanent single bit hard errors (SHEs) at a rate of λ_{SHE}. It is assumed that this memory is protected by a single error correct (SEC) ECC code. If two bits in a word are wrong (two SBUs, two SHEs or an SBU and SBE), the result is silent data corruption.

SBEs accumulate in the memory starting from the time the system is powered-up, t. Hard errors (SHEs) accumulate in the memory starting at the time the system was built, t'. It is shown that the rate of data corruption with this system is:

$$\lambda_{SEC} = \frac{\lambda_{SBU}^2 \cdot t}{M} + \frac{\lambda_{SHE}^2 \cdot t'}{M} + \lambda_{SBU} \cdot \frac{\lambda_{SHE} \cdot t'}{M} + \lambda_{SHE} \cdot \frac{\lambda_{SBU} \cdot t}{M} \tag{5}$$

It is common practice to use scrubbing [35] to prevent error accumulation. If the scrub period is denoted by T_S, then the rate of data corruption is given by:

$$\lambda_{SEC} = \frac{\lambda_{SBU}^2 \cdot T_S}{2 \cdot M} + \frac{\lambda_{SHE}^2 \cdot t'}{M} + \lambda_{SBU} \cdot \frac{\lambda_{SHE} \cdot t'}{M} + \lambda_{SHE} \cdot \frac{\lambda_{SBU} \cdot T_S}{2 \cdot M} \tag{6}$$

These equations can be coded in *RIIF* as shown in Fig. 5. First, a base *component* for an SRAM is declared. This model defines the rates of SBEs and SHEs as a function of voltage and temperature. Then this *component* is extended to model the effects of ECC and scrubbing.

Currently, an initial version of a tool to process RIIF models [36] is available and the results of simulating these

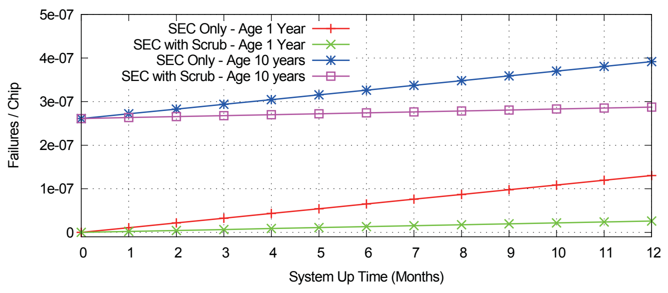

Fig. 6. Results of Simulating SRAM RIIF Model

models have been plotted in Fig. 6. From the results, we see that with scrubbing, the failure rate grows slowly with time, as the probability of an SBU and an SHE occurring in the same word is small. Without scrubbing, of course, the failure rate grows quickly with time, as the SBUs accumulate in the memory.

This example shows that even a simple system requires a time-variant model with voltage and temperature dependencies. Without a standard and scalable modeling language it is difficult for industry to accurately predict the reliability of complex silicon systems.

VI. Test Perspectives

Testing integrated circuits after manufacture is a crucial step in ensuring the integrity of the systems which include them. This is particularly important for highly reliable systems which are assumed to be initially fault-free. The test process involves the application of input patterns to the chip which will expose any internal defects. If the tests are not perfect, a coverage factor reflecting the probability of an untested fault affecting the reliability should be included in the cross-layer representation. As circuits grew complex and exhaustive testing became impossible, fault models were used to target the tests. The "stuck-at" model for digital circuits is the most commonly used, and experience has shown that such tests detect many defects even if they are not correctly modeled as stuck-at faults. For specific circuits, such as memories, fault models were specified at the behavioral level and used to develop $O(n)$ tests for memories with n cells which could detect interactions between any pair of cells [37]. Such tests can be run periodically during operation to detect latent failures and to support bypass or repair of the faulty regions. In order to reduce test generation cost and application time, use of "design-for-test" techniques, such as *scan*, became common practice. Test access is provided to the internal storage elements in test mode by connecting them as part of a shift register [38]. This increases both the *controllability* and *observability* of the circuit, and simplifies test generation by only having to deal with combinational circuits. This concept has been extended to testing Systems on Chips (SoCs), and incorporated into standards. "Built-In Self Test" (BIST) techniques include pattern generators on the chip and logic to compress the responses into signatures, enabling tests to be applied at speed [39], and such techniques are commonly used in chips. The hardware overhead of BIST can be eliminated for processors by using processor instructions to generate tests and processor registers to store the signatures [40]. This technique, called native-mode or software based self test, has been used successfully in industry [41]. Such application-level tests are ideal for on-line periodic testing of the hardware, and information on any failures identified can be used to improve system-level reliability.

With advances in technology, including subwavelength lithography, random dopand fluctuations, hot-spots, and voltage variations, circuit behavior and the types of defects seen have changed. Experiments on chips have shown that some tests applied at slow speeds (such as when using scan) would not expose defects, but would do so when applied at the circuit operating speeds [42]. The "delay fault" model more accurately captures the results of these trends in digital circuits. Recent work on new design-for-test ideas to detect delay faults include on-chip delay lines to accurately measure path delays using scan tests and the skew between data and control signals (in memories, for example). Excessive delay on a path will result in the incorrect value stored in a flip-flop, similar to a bit-flip. The slacks on paths, which generally reduce with an increase in temperature or a decrease in the supply voltage, would be an indication of the operational reliability. The delay lines could be used to monitor slacks on paths to alert the system or application to potential failures.

The trend in SoCs, driven by consumer electronics, is to include analog and RF modules in addition to many digital cores. Analog and RF modules have to be tested for compliance with their specifications, such as bandwidth, signal-to-noise ratio and linearity. Attempts to develop structural fault models as in digital circuits have not been very successful for testing real chips. One approach embedded analog modules is to place them in loopback mode during the test, for example, a digital-to-analog converter feeding an analog to digital converter. Techniques have been developed for checking for the individual specifications of the converters from the results of the loopback test. A promising approach for efficiently testing for many specifications is *alternate test* [43] which exploits the fact that any parameter variations in the chip which affect the specifications will also affect other, easier to measure, parameters, such as current, amplitude of a signal, etc. Using carefully crafted tests which are sensitive to the possible parameter variations, the approach finds correlations between the measured signals and the specifications. After this initial training, high-volume tests can predict the specifications from the measured values with much lower cost. This approach has also been used successfully in industry.

There is ongoing research into developing on-chip circuits which can support analog and RF test. One example of testing RF circuits for high-frequency specifications is to embed sensors on the chip, [44]; chip measurements show that specifications which are difficult to test, such as linearity, can be predicted very accurately from the low-frequency output of the sensors, and thus reducing the cost of test.

Further research is needed for thoroughly testing complex SoC with embedded analog and RF modules. Extending the software-based self test ideas with powerful design-for-test circuitry on chip will help to make highly reliable systems of the future possible. How to balance the benefits of online test-methods against the required cost of such methods will be an important research topic. Also, test techniques to ensure reliability need to be compared in their effectiveness and their cost-benefit ratio to other techniques.

VII. Conclusion

Cross-layer techniques for improving the reliability of integrated circuits are fast moving from nice-to-have towards becoming a key enabler for reaping continued benefits from the scaling of CMOS process technologies. An essential key for the successful deployment of such cross-layer techniques are standardized interfaces that allow different levels of the design hierarchy to communicate seamlessly and efficiently. In this paper, we have described two such interfaces: the *Resilience Articulation Point* (RAP) model which serves as a conceptual interface for propagation of reliability information between different abstraction layers, as well as the *Reliability Interchange Information Format* (RIIF), which allows for standardized exchange of reliability information between different CAD tools. This has been complemented by a brief description of three case studies that utilize cross-layer techniques for modeling reliability and improving system performance. Importantly, also the test perspective on reliability has been explored.

Further research is required to enable cross-layer design to demonstrate its full potential and to become standard industry practice. This research will address on the one hand specific algorithms for efficient cross-layer analysis and optimization of design. On the other hand it needs to further develop and utilize standardized interfaces such as RAP and RIIF.

Acknowledgments

This work was supported in part by the ENIAC ELESIS project and by the German Research Foundation (DFG) as part of the priority program "Dependable Embedded Systems" (SPP 1500 – http://spp1500.itec.kit.edu).

References

[1] Junko Yoshida. Toyota Case: Single Bit Flip That Killed. *EETimes*, October 2013.

[2] Todd Austin, Valeria Bertacco, Scott Mahlke, and Yu Cao. Reliable Systems on Unreliable Fabrics. *IEEE Design & Test of Computers*, 25(4):322–333, July 2008.

[3] Helmut E. Graeb. *Analog design centering and sizing*. Springer, 2007.

[4] Sachin Sapatnekar. *Timing*. Springer, 2004.

[5] Alfred Kwok-Kit Wong. *Resolution enhancement techniques in optical lithography*, volume 47 of *Tutorial Texts in Optical Engineering*. SPIE Publications, 2001.

[6] Andrei Pavlov and Manoj Sachdev. *CMOS SRAM circuit design and parametric test in nano-scaled technologies: process-aware SRAM design and test*, volume 40 of *Frontiers in Electronic Testing*. Springer, 2008.

[7] Sani R. Nassif, Veit B. Kleeberger, and Ulf Schlichtmann. Goldilocks failures: Not too soft, not too hard. In *International Reliability Physics Symposium (IRPS)*, 2012.

[8] Sani R. Nassif, Nikil Mehta, and Yu Cao. A resilience roadmap. In *Conference on Design, Automation & Test in Europe (DATE)*, 2010.

[9] Robert F. Service. What It'll Take to Go Exascale. *Science*, 335(6067):394–396, January 2012.

[10] Jörg Henkel, Lars Bauer, Joachim Becker, Oliver Bringmann, Uwe Brinkschulte, Samarjit Chakraborty, Michael Engel, Rolf Ernst, H Hartig, Lars Hedrich, et al. Design and architectures for dependable embedded systems. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, 2011.

[11] George Bosilca, Rémi Delmas, Jack Dongarra, and Julien Langou. Algorithm-based fault tolerance applied to high performance computing. *Journal of Parallel and Distributed Computing*, 69(4):410–416, 2009.

[12] Ramtilak Vemu and Jacob A Abraham. CEDA: Control-flow error detection through assertions. In *International On-Line Testing Symposium (IOLTS)*, 2006.

[13] Melvin A Breuer, Sandeep K Gupta, and T.M. Mak. Defect and error tolerance in the presence of massive numbers of defects. *IEEE Design & Test of Computers*, 21(3):216–227, 2004.

[14] Heather M Quinn, Andre De Hon, and Nick Carter. CCC visioning study: system-level cross-layer cooperation to achieve predictable systems from unpredictable components. Technical report, Los Alamos National Laboratory (LANL), 2011.

[15] William H. Robinson, Michael L. Alles, Theodore A. Bapty, Bharat L. Bhuva, Jeffery D. Black, Alfred B. Bonds, Lloyd W. Massengill, Sandeep K. Neema, Ronald D. Schrimpf, and Jason M. Scott. Soft error considerations for multicore microprocessor design. In *International Conference on Integrated Circuit Design and Technology (ICICDT)*, 2007.

[16] Subhasish Mitra, Kevin Brelsford, and Pia N. Sanda. Cross-layer resilience challenges: Metrics and optimization. In *Conference on Design, Automation & Test in Europe (DATE)*, 2010.

[17] Naresh R Shanbhag, Rami A Abdallah, Rakesh Kumar, and Douglas L Jones. Stochastic computation. In *Design Automation Conference (DAC)*, 2010.

[18] Nicholas P. Carter, Helia Naeimi, and Donald S. Gardner. Design Techniques for Cross-Layer Resilience. In *Conference on Design, Automation & Test in Europe (DATE)*, 2010.

[19] Vijay Janapa Reddi, David Z Pan, Sani R Nassif, and Keith A Bowman. Robust and resilient designs from the bottom-up: Technology, CAD, circuit, and system issues. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2012.

[20] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. DRAM Errors in the Wild: A Large-Scale Field Study. In *ACM SIGMETRICS*, 2009.

[21] Edmund B. Nightingale, John R. Douceur, and Vince Orgovan. Cycles, cells and platters: an empirical analysis of hardware failures on a million consumer PCs. In *Conference on Computer Systems*, 2011.

[22] Hyungmin Cho, Shahrzad Mirkhani, Chen-Yong Cher, Jacob A Abraham, and Subhasish Mitra. Quantitative evaluation of soft error injection techniques for robust system design. In *Design Automation Conference (DAC)*, page 101, 2013.

[23] Andreas Herkersdorf, Michael Engel, Michael Glaß, Jörg Henkel, Veit B. Kleeberger, Michael Kochte, Johannes M. Kühn, Sani R. Nassif, Holm Rauchfuss, Wolfgang Rosenstiel, Ulf Schlichtmann, Muhammad Shafique, Mehdi B. Tahoori, Jürgen Teich, Norbert Wehn, Christian Weis, and Hans-Joachim Wunderlich. Cross-Layer Dependability Modeling and Abstraction in System on Chip. In *Workshop on Silicon Errors in Logic - System Effects (SELSE)*, 2013.

[24] Andreas Herkersdorf, Hananeh Aliee, Michael Engel, Michael Glaß, Christina Gimmler-Dumont, Jörg Henkel, Veit B. Kleeberger, Michael Kochte, Johannes M. Kühn, Daniel Mueller-Gritschneder, Sani R. Nassif, Holm Rauchfuss, Wolfgang Rosenstiel, Ulf Schlichtmann, Muhammad Shafique, Mehdi B. Tahoori, Jürgen Teich, Norbert Wehn, Christian Weis, and Hans-Joachim Wunderlich. Resilience Articulation Point (RAP): Cross-layer Dependability Modeling for Nanometer

System-on-chip Resilience. *Elsevier Microelectronics Reliability*, 2014. Accepted for publication.

[25] Hans-Joachim Wunderlich and Stefan Holst. Generalized Fault Modeling for Logic Diagnosis. In Hans-Joachim Wunderlich, editor, *Models in Hardware Testing*, volume 43 of *Frontiers in Electronic Testing*, pages 133–155. Springer Netherlands, 2010.

[26] Bernd Becker, Sybille Hellebrand, Ilia Polian, Bernd Straube, Wolfgang Vermeiren, and Hans-Joachim Wunderlich. Massive statistical process variations: A grand challenge for testing nanoelectronic circuits. In *International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2010.

[27] Michael Glaß, Heng Yu, Felix Reimann, and Jürgen Teich. Cross-Level compositional reliability analysis for embedded systems. In *International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, 2012.

[28] Adrian Evans, Michael Nicolaidis, Shi-Jie Wen, Dan Alexandrescu, and Enrico Costenaro. RIIF - Reliability Information Interchange Format. In *International On-Line Testing Symposium (IOLTS)*, 2012.

[29] Adrian Evans and Oliver Bringmann. RIIF DATE 2013 Workshop: Towards Standards for Specifying and Modelling the Reliability of Complex Electronic Systems. http://riif-workshop.fzi.de/, 2013.

[30] Veit B. Kleeberger, Daniel Mueller-Gritschneder, and Ulf Schlichtmann. Technology-Aware System Failure Analysis in the Presence of Soft Errors by Mixture Importance Sampling. In *Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, 2013.

[31] Veit B. Kleeberger, Christina Gimmler-Dumont, Christian Weis, Andreas Herkersdorf, Daniel Mueller-Gritschneder, Sani R. Nassif, Ulf Schlichtmann, and Norbert Wehn. A Cross-Layer Technology-Based Study of How Memory Errors Impact System Resilience. *IEEE Micro*, 33(4):46–55, July/August 2013.

[32] Christina Gimmler-Dumont, Christian Brehm, and Norbert Wehn. Reliability Study on System Memories of an Iterative MIMO-BICM System. In *International Conference on Very Large Scale Integration (VLSI-SoC)*, 2012.

[33] Ik Joon Chang, Debabrata Mohapatra, and Kaushik Roy. A priority-based 6T/8T hybrid SRAM architecture for aggressive voltage scaling in video applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 21(2):101–112, 2011.

[34] Alfonso Sanchez-Macian, Pedro Reviriego, and Juan Antonio Maestro. Modeling Reliability of Memories Protected with Error Correction Codes with RIIF. In *RIIF DATE 2013 Workshop: Towards Standards for Specifying and Modelling the Reliability of Complex Electronic Systems*, 2013.

[35] Abdallah M. Saleh, Juan J. Serrano, and Janak H. Patel. Reliability of scrubbing recovery-techniques for memory systems. *IEEE Transactions on Reliability*, 39(1):114–122, 1990.

[36] [Online] Available:. http://code.google.com/p/java-riif-cli/.

[37] Ravindra Nair, Satish M. Thatte, and Jacob A. Abraham. Efficient algorithms for testing semiconductor random-access memories. *IEEE Transactions on Computers*, 100(6):572–576, 1978.

[38] Miron Abramovici, Melvin A Breuer, and Arthur D Friedman. *Digital systems testing and testable design*, volume 2. Computer Science Press, 1990.

[39] Edward J McCluskey. Built-in self-test techniques. *IEEE Design & Test of Computers*, 2(2):21–28, 1985.

[40] Jian Shen and Jacob A Abraham. Native mode functional test generation for processors with applications to self test and design validation. In *International Test Conference (ITC)*, 1998.

[41] Praveen Parvathala, Kaila Maneparambil, and William Lindsay. FRITS-a microprocessor functional BIST method. In *International Test Conference (ITC)*, 2002.

[42] Edward J McCluskey and Chao-Wen Tseng. Stuck-fault tests vs. actual defects. In *International Test Conference (ITC)*, 2000.

[43] Achintya Halder, Soumendu Bhattacharya, and Abhijit Chatterjee. Automatic Multitone Alternate Test Generation For RF Circuits Using Behavioral Models. In *International Test Conference (ITC)*, 2003.

[44] Chaoming Zhang, Ranjit Gharpurey, and Jacob A Abraham. Low cost RF receiver parameter measurement with on-chip amplitude detectors. In *VLSI Test Symposium (VTS)*, 2008.