Reliability Analysis Reloaded: How Will We Survive?

Robert Aitken
ARM Incorporation
San Jose, CA 95510, USA
rob.aitken@arm.com

Görschwin Fey
German Aerospace Center
28359 Bremen, Germany
goerschwin.fey@dlr.de

Zbigniew T. Kalbarczyk
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
kalbarcz@illinois.edu

Frank Reichenbach
ABB Corporate Research
1375, Billingstad, Norway
frank.reichenbach@no.abb.com

Matteo Sonza Reorda
Politecnico di Torino
10129 Torino, Italy
matteo.sonzareorda@polito.it

***Abstract*—In safety related applications and in products with long lifetimes reliability is a must. Moreover, facing future technology nodes of integrated circuit device level reliability may decrease, i.e., counter-measures have to be taken to ensure product level reliability. But assessing the reliability of a large system is not a trivial task. This paper revisits the state-of-the-art in reliability evaluation starting from the physical device level, to the software system level, all the way up to the product level. Relevant standards and future trends are discussed.**

I. Introduction

Concepts for reliability and in particular reliability analysis of systems have been a hot topic in research and industry ever since the introduction of highly integrated systems over 40 years ago. Now, reliability problems are expected to increase rapidly for future technology nodes triggered by different sources [1] and by the increased adoption of electronic systems in safety-critical applications. Process variations in the production process may further exacerbate this issue.

While production processes become more accurate considering absolute measures, the relative inaccuracy compared to the component's size is increasing. One consequence are transistors with a wide range of threshold levels in a single circuit resulting in slightly faster or slower operating logic circuitry [2]. This may result, e.g., in delay errors under certain operating conditions of a device. Increasing sensitivity to the omnipresent environmental radiation is another issue. In the past some errors induced by radiation have been observed infrequently while systems in space missions are specified to be radiation resistant already. Shrinking feature sizes result in sensitivity to radiation with lower energy causing more radiation induced events like *Single Event Upsets* (SEUs) even at sea level. These issues may manifest as transient faults resulting in soft errors or as permanent faults resulting in a change of the functionality of the system. Relatively simple traditional fault models like, e.g., the stuck-at or gate-delay fault model, may be insufficient to address all these issues. But handling all types of physical faults individually will prove infeasible in practice. Moreover, understanding the effects of faults, and in particular identifying those faults that could be able to create serious consequences to the product mission is a key point which may become very hard to face when complex systems are considered. Regulations and standards already started addressing these issues and industry is facing growing interest in how to deal with them in practice. The sheer complexity of today's and future's "more Moore" and "more than Moore" systems as well as the huge universe of potential faults requires reliability analysis to be revisited from this new perspective.

Design margins are a traditional way to handle reliability issues at one level of abstraction, so they can be ignored at a higher level. But margins are only acceptable up to a certain cost level. Otherwise, failures have to be handled at higher levels in the system. Thus, reliability evaluation has to take the whole system stack into account.

This paper covers reliability analysis at the different levels of abstraction. Section II views at the above issues at the lowest layer from an industrial perspective, i.e., how we can find models for device level reliability that may be used for determining the reliability of a system. Section III reviews state-of-the-art techniques to evaluate the reliability of an operating system. Finally, the dependencies between safety standards and the reliability evaluation of the final product are addressed in Section IV. Section V draws conclusions.

II. Reliability Evaluation at the Device Level and Its Impact on Design

When determining the reliability of a system, knowing the respective parameters of the underlying devices is mandatory. An accurate device-level evaluation requires a deep under-

978-3-9815370-0-0/DATE13/©2013 EDAA

standing of the fault mechanisms including aging, stress, process variation etc. The underlying devices of integrated systems are the gates, the transistors, and the physical connections. We will focus on techniques to evaluate the device-level reliability of a given production technology or product family and discuss the typical flow used in practice to derive usable figures.

Degradation at the device level can be broadly divided into two groups: physical mechanisms, due to inherent defects and/or progressive degradation, and electrical mechanisms, which are transient in nature and do not permanently damage the device. Both must be responded to in a highly reliable system, but only the former needs a permanent workaround.

A. Physical Degradation Mechanisms at the Device Level

1) Dielectric Breakdown: This discussion is based on Choudhury et al [3]. Current generation devices are prone to *Time-Dependent Dielectric Breakdown* (TDDB), also known as soft oxide breakdown, during their lifetime. A soft oxide breakdown begins when interface "traps" begin to form in the gate oxide. At first, the traps are non-overlapping and thus do not conduct. As more traps are formed, they start to overlap and this may result in a resistive conduction path from gate to channel. Once the conduction channel is formed, more traps appear due to thermal damage. These new traps cause the conduction channel to become wider and hence more current flows leading to even higher temperature. This thermal runaway condition leads to a catastrophic failure known as *Hard Oxide Breakdown* (HBD).

There are three phases of gate oxide wearout [4]. The traps start to form at the end of the first phase, known as the time to soft breakdown. During the second phase (SBD phase), the traps move around and the leakage current fluctuates randomly. In this phase, the device is still functional but drifts in energy and delay. Once the conduction path is formed in the oxide, the device enters the third phase (HBD phase). In the HBD phase, the leakage current is exponentially higher and the fault is termed catastrophic. The transition from the beginning of soft oxide breakdown to hard breakdown is not abrupt, and the gate leakage current starts to progressively increase long before hard breakdown occurs. A variety of design approaches exist to both monitor and mitigate TDDB [5], [6].

2) NBTI/PBTI: Bias temperature instability occurs when a negatively-biased gate is stressed (PFET case for NBTI; for NFETs the effect happens with positive bias) at high temperature. Several models have been proposed to explain the precise mechanism [7]. Most of these involve the formation of traps in the gate oxide due to diffusion of hydrogen and subsequent increase in gate threshold voltage (Vt) leading to reduced performance over time. Unlike other aging effects, BTI can be partially offset by "healing" – when the device is oppositely biased, some of the traps collapse. Designers can thus mitigate the effects of BTI by balancing bias states. This can be quite challenging, depending on the portion of the design, but many efforts have been published [8], [9].

*3) Hot Carrier Injection (*HCI*):* HCI occurs when an energetic ("hot") carrier physically damages the drain of a device due to its acceleration through the electric field of a gate. Reduced operating voltages reduce the number of hot carriers, and junction engineering can mitigate its effects [10] but in general it is difficult for designers to influence HCI. Instead, expected time to failure can be calculated using Arrhenius equations and accounted for in reliability budgets.

*4) Electrostatic discharge (*ESD*):* ESD can be thought of as an extreme form of dielectric breakdown. An excess of charge enters the circuit (e.g. via a spark at a pin) and this charge must be shunted safely into the ground network before it can damage a device oxide. ESD protection is a field unto itself and includes both direct shunting mechanisms (e.g. snapback devices) as well as electrical design rules for safety (never connect the gate of a transistor directly to a power or ground rail). A good overview of ESD design practices can be found in Voldman [11].

*5) Electromigration (*EM*):* EM is the physical movement of metal in a carrier, particularly a narrow one, as a result of high current density. Shrinking geometries have moved this problem from one of mild interest around high drive buffers to something that must be carefully checked throughout a design. Careful design of power supply distribution networks, especially the power buses within standard cells, is key to successfully coping with EM.

B. Electrical Degradation Mechanisms at the Device Level

1) Soft Errors: Soft errors are radiation induced faults which result from a particle strike, either by an alpha particle from impurities in packaging material or a neutron from cosmic rays. When particles strike the silicon substrate they create hole-electron pairs which are then collected by PN junctions via drift and diffusion mechanisms. This collected charge creates a transient current pulse and, if it is large enough, it can flip the value stored in one or more state saving elements (bit cell, latch etc.) When particle strike happens in combinational logic, the result is a glitch which can then propagate to a latch where it could be captured. Embedded SRAMs are especially vulnerable to SEUs due to the small size of each bit cell and its small node capacitances. The soft error rate per bit cell has stabilized in recent technologies, but the rate per area increases [12]. Soft errors are mainly mitigated through improved circuit design and the use of error correcting codes.

2) Noise: Noise arises from several sources, including power supply fluctuations (e.g. IR drop), clock jitter, and crosstalk (capacitive coupling between nets). Physically, noise is disruptive to circuit operation and can in the worst case lead

to erroneous behavior. In most cases, noise is dealt with at the design level by adding margin.

*3) Random Telegraph Noise (*RTN*):* Recently, random telegraph noise (RTN) has emerged as a reliability issue [13]. Traps in device oxides are randomly filled and emptied, leading to shifts in threshold voltage and drain current. These are most important in small devices, but have the potential to cause failures in designs with low margin.

4) Variability: Although variability is not strictly a degradation mechanism, it can lead to electrical failures both directly, when device parameters exceed limits accounted for during design, and indirectly, when a design is pushed close to its margins and subsequent degradation from one of the other mechanisms pushes it over. As a result, quantifying variability, both deterministic and random, is key to successful design-for-reliability.

C. Design Level Reliability Modeling

When designing at the transistor level, it is important to be able to quantify the effects of the various reliability mechanisms discussed in the previous section in order to be able to successfully mitigate them. In many cases, this can be accomplished through guardbanding. For example, if it is determined that the collective worst-case effect of aging will be a 10% upward shift in Vt, then standard cells and memories can be characterized for that point and then used as part of sign off. Similarly, an error correcting code together with a known or estimated soft error rate can be used to predict (and subsequently minimize) the failure rate of a system.

Other transient effects are more challenging because their magnitude is difficult to predict. In-line noise can be modeled (e.g. using CCS-noise in Liberty [14] and considered as part of static timing analysis, but the sources of the noise (power supply fluctuations, crosstalk, etc.) are notoriously difficult to model accurately, so designers typically make conservative estimates in order to ensure silicon functionality.

In this way, contributions to overall reliability can be calculated and then summed together to give an overall margin for the design in both performance and power consumption. As a general rule, performance budgets require more detailed accounting than power budgets, since a 1% slowdown could cause a part to fail, whereas a 1% increase in the power budget will most likely contribute only to a minor increase in operating temperature or decrease in battery life. In performance budgeting, setup time changes are less critical than hold time changes, since the former can be compensated for by slowing down the clock rate.

As transistors are combined into standard cells, memory, logic blocks, cores, and systems, reliability effects are by necessity abstracted into higher level changes in behavior. Multiple physical effects can be grouped into an aging model, for example, and guardbands can be developed for variability and noise. These abstractions can only be as good as their underlying data, so careful understanding and accounting should take place.

Table I
SPATIAL AND TEMPORAL CLASSIFICATION OF VARIATION

	Static	Slow-Changing	Fast-Changing
Global	Inter-die process variations Aging (BTI, TDDB, EM)	Die-level VDD variation Ambient temperature variation	Clock jitter IR drop Ldi/dt
Local	Random dopant fluctuation	Temperature hot spots	Coupling noise Local clock jitter

D. Workloads and Reliability

Many of the mechanisms discussed in the previous section are activity dependent. BTI includes a "healing" component when a transistor is not active. TDDB happens only when an oxide is stressed. Neutron strikes will not cause soft errors when a memory is powered down. This activity dependence is very important when making reliability predictions for a given product. A product in constant operation is much more likely to suffer a reliability fail than one that is rarely used. Similarly, some portions of a design are much more likely to be stressed than others. In some cases, the stressed devices are obvious (clock networks, for example), while others are more subtle: in many designs (e.g. processors) memories tend to contain more 0s than 1s throughout normal operation, meaning that 2 of the 6 devices in each SRAM bit cell undergo much more TDDB related stress than the others, in the worst case leading to unbalanced bit cells and failing read/write operations. Consideration of workloads can result in substantial changes to product reliability [15].

E. Trends

Reliability effects can be thought of as lying along two axes: geographical (local versus chip wide) and temporal (slow changing versus fast changing), as shown in Table I. A system must be robust enough to handle fast changing, local situations such as soft errors or local clock jitter. Effects with longer durations or larger area effects can also be handled adaptively, i.e., the system can identify that a change has occurred and adjust itself accordingly, by boosting voltage or lowering clock frequency, for example.

Over the next few process generations, we expect to see some of these effects get worse. Among these are virtually all of the physical failure mechanisms described earlier, so it is vital for designers to account for reliability in a complete and correct fashion, accounting for expected usage profiles. Other elements are more directly under designer control, including noise, soft error, and ESD tolerance. Error detection and/or correction can help. Finally, there are some effects that may improve over time. Fully depleted devices, such as FinFETs,

for example, should have lower random variation than their bulk predecessors.

Adaptive systems may be the best way forward, and while they remain challenging to implement for a variety of reasons that are beyond the scope of this paper, they are very likely to be the most successful design-for-reliability approach in future technologies.

But still handling all types of reliability issues at the device level neither be feasible nor economically viable. Therefore the careful assessment of device level reliability measures provides the input required as a solid base for system level reliability evaluation.

III. Reliability Evaluation at the System Level

The dependability of a computing system and consequently the services it provides to the end user, depends to large extent on the failure-resilience of the underlying *Operating System* (OS). Understanding a system's sensitivity to errors and identifying error propagation patterns and single points of failure are thus of primary importance in selecting a computing platform and in assessing tradeoffs involving cost, reliability, and performance.

In the recent years several approaches were proposed to evaluate robustness of OSs, e.g., [16], [17], [18]. These studies address robustness at the user visible interfaces and constitute an important component in OS benchmarking. It is however, critical to quantify the impact of a broad range of faults that occur in the processor, memory, I/O, and the network interfaces of the underlying hardware and the corresponding system and application software. Sound fault/error injection methods and tools are essential when quantifying these metrics directly or indirectly (as we do by associating performance loss and the severity of faults). While there are many outstanding issues in terms of – *how, where and when* – faults/errors should be injected so that OSs can be evaluated and compared, there is strong evidence (as exemplified on evaluation of real systems) that fault/error injection should be an integral part of system benchmarking procedures.

A comprehensive benchmark requires evaluating different approaches in assessing system dependability and using this knowledge to define sound procedures, methods, and tools to enable experimental OS benchmarking. While external characteristics are satisfying and acceptable in performance benchmarking, this is not the case for dependability benchmarking. Rather a combination of a "black box" and a "white box" approach has emerged.

The remaining of this section presents the use of software implemented fault injection to conduct experimental studies and to derive quantitative dependability measures. Doing so we can stress a broad range of system components including OS code, data and stack sections, and processor registers. The systematic approach allows:

- Assessing fault severity, efficiency of detection, and recovery mechanisms under variable workloads (real or synthetic applications), thus quantifying coverage and ability of the system to recover.
- Measuring the detection latency, which is of particular importance in characterizing chances of errors to: (i) propagate between system components and (ii) escape beyond the containment boundaries defined by a computing node, e.g., fail silence violations or silent data corruption.
- Exercising (with faults) critical execution paths within a code to pinpoint the error sensitive system components/locations. This, in turn, provides a feedback to the developers on ways for integrating enhancements.

In order to demonstrate strength of the fault injection based system evaluation we discuss experience and lessons learned from assessing error sensitivity of the Linux Kernel executing on PowerPC (G4) and Pentium 4 (P4) processors [19], [20]. The goal is to introduce methodology and illustrate the type of results that can be obtained to enable: (i) comparing the Linux kernel behavior under a broad range of errors on two target processors and (ii) understanding how architectural characteristics of the target processors impact the error sensitivity of the OS. Two target Linux-2.4.22 systems are used: the Intel Pentium 4 (P4) running RedHat Linux 9.0 and the Motorola PowerPC (G4) running YellowDog Linux 3.0.

A. Evaluation of OS Dependability

Due to the size of the OS, it is impractical to target the entire OS code for error injection. Instead, a more practical approach is to focus on the most important (critical) subsystems and the most frequently used functions. This information is obtained by system profiling while running benchmark programs which are used to ensure sufficient kernel activity to trigger injected errors.

1) Methodology: An approach for sound experimental assessment (benchmarking) of OSs includes:

- methods to stress the system, i.e., to generate runtime errors;
- procedures to specify a set of measurements in terms of error types, error frequency, and workloads;
- metrics to quantify dependability attributes of the OS; and
- tools to set up and carry on experiments, collect and analyze the measurement data, and calculate the dependability metrics.

2) Error Injection Environment: Software-implemented error injection is a common method employed for experimental assessment of computing system dependability. NFTAPE [21], a software framework for conducting fault/error injection experiments, is used to conduct the tests in the presented example study.

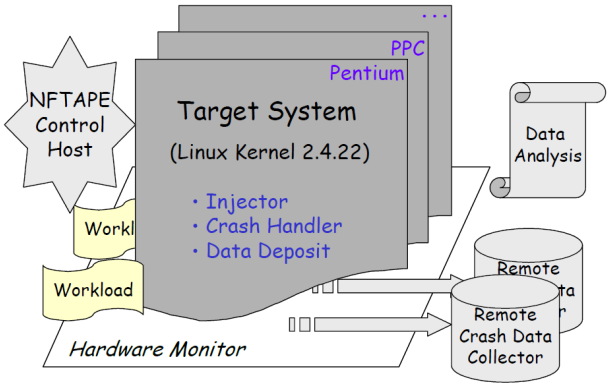

Figure 1. Error injection environment

Single-bit errors are injected into kernel stacks, kernel code sections, kernel data structures, and CPU system registers while running benchmark programs. The NFTAPE error injection environment, shown in Figure 1, consists of (i) kernel-embedded components (injectors, crash handlers, and data deposit module) for different architectures, (ii) a user-level NFTAPE control host, which prepares the target addresses/registers (to be injected), starts the workload program, and logs injection data for analysis, (iii) a hardware monitor (e.g., a watchdog card) to detect system hangs/crashes in order to provide auto reboot if needed, and (iv) a remote crash data collector that resides on the control host computer to receive data on system crashes and on detection latency.

a) Error Model: The error model assumed is not contingent upon the error origin, i.e., an error could have occurred anywhere in the system – the disk, network, bus, memory, or CPU. Single-bit errors are injected into the instructions of the target kernel functions, the stack of the corresponding kernel process, the kernel data structures, and the corresponding CPU's system registers. Previous research on microprocessors [22] has shown that most (90-99%) of device-level transients can be modeled as logic-level, single-bit errors. Data on operational errors also show that many errors in the field are single-bit errors [23].

While in a well-designed system multiple mechanisms for protecting against errors may be available (e.g., parity, *Error Correcting Code* (ECC), or memory scrubbing), errors still exist. Errors could, for example, be timing issues due to hardware/software problems, to a noise source such as undershoot or overshoot, or to noise on the address bus that results in the wrong data being written to/read from the memory. In the latter case, the data may be unaltered due to the address bus noise, but the wrong location is accessed. Memory errors and system register errors are used to emulate the diverse origins and impact of actual errors.

As indicated by manufactures, logic failure rates may erode the efficacy of ECC in designs. Hardened logic libraries or schemes to mask logic sensitivity (redundancy on critical paths, spatial and/or temporal) may be needed to account for this deficiency.

Table II
OUTCOME CATEGORIES

Outcome Category	Description
Activated	The corrupted instruction/data is executed/used.
Not Manifested	The fault is activated but it does not cause a visible abnormal impact on the system.
Fail Silence Violation	Either OS or application allows incorrect data/response to propagate out.
Crash	Operating system stops working, e.g., bad trap or system panic.
Hang	System resources are exhausted, resulting in a non-operational system, e.g., deadlock.

Four attributes characterize each error injected: (i) *trigger*, i.e., when an error is injected, (ii) *location*, i.e., where an error is injected, e.g., memory or CPU registers, (iii) *type*, i.e., what to corrupt, e.g., a single bit in a data item, and (iv) *duration*, i.e., how long the injected fault/error persists, e.g., a bit is flipped in the target to emulate the impact of a transient event.

b) Outcome Categories: Outcomes from error injection experiments are classified according to the categories given in Table II.

3) Sample Results: Analysis of the obtained data indicates significant differences between the two platforms in how errors manifest and how they are detected in the hardware and the OS.

a) Fault/Error Manifestation: The error activation rates are generally similar for both processors; however, the manifestation rates for the Pentium 4 are about twice as high as compared with the G4 platform. The fault injection based approach allows quantifying the observed differences and similarities. For instance, for stack errors, the manifestation rates are 56% for P4 versus 21% for G4. A similar trend is observed in the case of an error in the kernel data (66% for the P4 versus 21% for the G4). The observed difference between the two platforms can be explained by the disparity in the way they use memory. The G4 processor always operates on 32-bit wide data items, while the P4 allows 8-bit, 16-bit, and 32-bit data transfers. As a result, the sparseness of the data can mask errors. For example, the larger presence of unused bits in data items means altering any unused bit is inconsequential, even if the corrupted data instruction is used. The more optimized access patterns on P4 increase the chances that accessing a corrupted memory location will lead to problems. Though less compact, fixed 32-bit data and stack access make the G4 platform less sensitive to errors.

b) Crash Latency: An interesting observation can be made in term of crash latency. *Crash Latency (Cycles-to-Crash)* is defined as the number of CPU cycles between error activation and the actual crash. Typically, latency includes three stages, as shown in Figure 2.

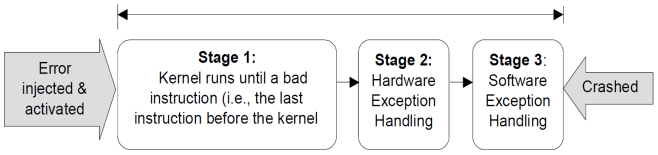

Figure 2. Definition of Cycles-to-Crash

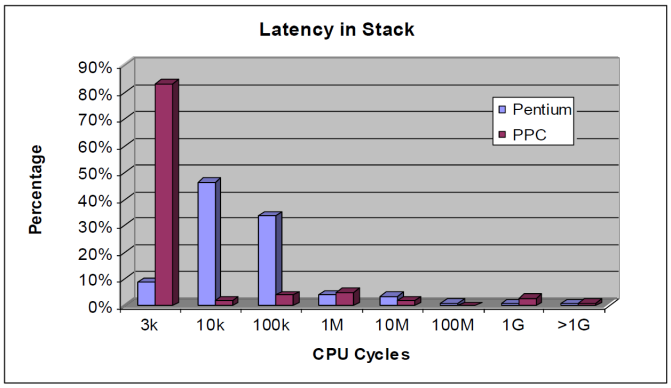

Figure 3. Crash Latency Distribution for Stack Injection

Figure 3 depicts the crash latency distribution for errors injected into the stack on the two target platforms (i.e., P4 and G4 processors). The data show that the majority (80%) of stack errors on the G4 platform are short-lived (less than 3,000 cycles). On the P4 platform, the majority (80%) of stack errors result in longer crash latency (3,000 to 100,000 cycles). The primary reason for this disparity is the way the two platforms handle exceptions. For example, the kernel on the G4 platform provides quick detection of stack overflow errors, while the kernel on the P4 architecture converts stack overflow events into other types of exceptions (e.g., *Bad Paging*), resulting in inherently slower detection.

One can also observe that the crash latency distribution has a long tail and there are a non-negligible percentage of cases when the crash latency exceeds hundreds of millions of CPU cycles. *In other words, the processor can execute millions (or even billions) of instructions in presence of an active error before it finally crashes.* During this time errors can propagate causing the system to produce bad data or making incorrect decisions. To cope with such problems one needs to integrate low-latency error detection mechanisms at the system and/or application level [24], [25], [26]. It is worth noting that the crash latency can only be measured in the controlled experiments such as the one described here.

c) Crash Severity: Another interesting result comes from analyzing the *OS crash severity* in terms of potential downtime due to the failure. The severity of the crash failures resulting from the injected errors can be categorized into three levels: (1) *most severe* – rebooting the system after an error injection requires a complete reformatting of the file system on the disk and the process of bringing up the system can take nearly an hour; (2) *severe* – rebooting the system requires the user (interactively) to run *fsck* facility/tool to recover the partially corrupted file system, the process can take more than five minutes and requires user intervention; and (3) *normal* – the system automatically reboots, and the rebooting usually takes less than four minutes depending on the type of machine and the configuration of Linux.

While relatively few crash cases in the severe level category were observed, several cases required reformatting the file system. The availability impact of the most severe crashes is clearly of concern. For example, to achieve 5 nines of availability (5 min/year downtime) one can only afford about one failure classified most severe in 10 years, no more than one severe crash in two years, and a "normal" crash no more than once a year.

B. Trends

Lessons learned extend beyond the examples of studies presented in this section. The most generic observations can be grouped into two categories: (i) What is the unique value in employing fault/error injection to benchmark computing systems? (ii) What is expected from benchmarking tools?

1) Value in Employing Fault Injection:

a) Characterization of Crash Severity: It is a common assumption that crashes are benign and that there is a mechanism in a system that ensures that when the program encounters an error, the application will crash instantaneously. While many crashes are benign, severe system failures often result from latent errors that cause undetected error propagation resulting in file corruption (e.g., corruption of the OS image on the disk), remote process failures, or checkpoint corruption.

b) Measurement of detection latency and validation of crash-failure semantic: Assumption of crash failure semantic for a program or a system behavior is not good, if one cannot provide efficient mechanisms for rapid error detection to ensure that this assumption holds in practice. Measurement of detection latency must be an integral component of the benchmarking procedure. For example, we demonstrated that crash latency can be as large as hundreds of millions or billions of cycles.

c) Characterization of Recovery Latency: Measurement of recovery time is crucial to assess system downtime and hence, to quantify availability. Benchmarking must enable validation of system behavior in the case when multiple detection mechanisms are triggered due to same or propagated error.

2) Toolset and Benchmark Procedures:

a) Complexity: Benchmarking of fault tolerance requires complex procedures and tools (the process is far more complex than in the case of performance benchmarking). Often deploying the toolset is more time consuming than conducting the measurements.

b) Multiple Platforms (Hardware, OS): Using different tools one runs the risk that the benchmark measures the effectiveness of the tools rather than the dependability of the target system. A key to a wide acceptance of a benchmarking toolset is its portability across computing platforms.

c) Multiple Fault Models: Evaluation of complex systems benefits from injecting a wide variety of faults such as communication faults, bit-flip faults to memory and registers, and high-level faults specific to an application. The more diverse the fault set, the more can be learned about the target system.

While there is still a long way to go before we define the dependability benchmark for OSs, the approach discussed in this section shows a way towards this goal. The data gathered by these techniques is one example for evaluating the reliability of individual components in a full product.

IV. On Evaluating the Reliability of Industrial Products and the Impact of Safety Standards in Automation Industry

Both, high reliability and high availability (uptime) of industrial products are an absolute necessity in order to engineer sophisticated control systems in process and factory automation. Harsh environments with, e.g., *Electro-Magnetic* (EMC) disturbance, vibrations or dust put even more requirements on industrial devices and operational lifespans of up to 20 to 30 years are by far no exception in automation industry. This makes reliability analysis crucial in order to succeed on the market. In particular, the need for safety systems has been increasingly rising since the awareness for protecting people and environment is becoming eminent. Companies understood that prevention is more economical than reaction on accidents (like in the "Deepwater Horizon" case). *European Union* (EU) directives set strict requirements for the minimum level of health and safety. Reliability must be involved in both engineering of safety systems, but more than ever also in the development of safety products. For example, a typical safety system as the steam-boiler is comprised of pressure sensors (capable of detecting overpressure), a logical element like a *Programmable Logic Controller* (PLC - comparing sensor values with an upper threshold) and a valve (can be opened by the PLC to release steam) to avoid the steam-boiler's explosion. Sufficient safety must be given on engineering (system) level, which can be realized by using redundant architectures (e.g., 2 pressure sensors, 2 PLCs, 2 valves). There has been an ongoing trend that customers want to use safety certified products instead of redundancy on engineering level, because safety devices can be used in a single channel manner. This reduces hardware costs and overall maintenance costs while keeping the freedom to use safety devices from various vendors in one safety system. Such a solution assumes product certification and therefore a comprehensive reliability analysis at component level called *Component Failure Mode and Effect Analysis* (CFMEA) is required. This section will approach the handling of reliability for the *development of safety devices*, rather than for the engineering of safety systems.

A. Reliability and Safety

Safety certification of products has been becoming a market differentiator, even though safety does not necessarily increase the uptime of the entire system, which is sometimes forgotten or also misunderstood by the customers. However, in order to certify the product, the vendor is obliged to handle the product's reliability in a proper manner. It has to be proven that the product achieves a certain *Probability of Failure on Demand* (PFD) and a certain *Probability of Failure per Hour* (PFH), which depends on the demand of the safety related function. In a typical automation system there are some *significant differences between reliability and safety*. Firstly, a reliable device should carry out its functionality for a given period of time, but the safety aspect considers also the consequence of failures. Secondly, reliability would be measured by detecting the first failure after start of operation, but a safety system could be subject to failures. However, only if the failure is dangerous and leads to a severe consequence it would be taken into account. For example, a soft error (transient error) could flip a bit in the memory, but the ECC detects and recovers the memory, triggers a warning, and the system proceeds without further effects.

In the following we explain how reliability evaluation driven by standards is performed.

1) Standards: The new EN ISO 13849-1:2009 ("Safety of machinery – Safety-related parts of control systems") [27] follows a probabilistic approach by quantifying the components reliability in a safety system. This has been creating hurdles in industry, since the previous standard EN 954-1 was based on architectural decisions without complex reliability calculations. EN 954-1 was more deterministic and in a way more straight forward. However, the approach gives the machinery builders more freedom on how they can design their products. Device builders end up quickly in the IEC 61508 Ed. 2. This is a generic standard for functional safety of electric, electronic and programmable safety related equipment (E/E/PE), IEC 61508 Ed. 2 [28]. The standard aims at specifying a set of methods, measures and procedures that device builders must conform to for claiming a certain *Safety Integrity Level* (SIL). SIL is generic in the sense that it can be used as a standalone standard, in addition to acting as basis for many applications. This standard is a basis for many more domain specific standards and will also be used in this section as a reference. Most of ABB's safety development work is following the IEC 61508 Ed. 2. Thus this section on automation industry will set the main focus here.

2) Safety Integrity Level: According to the process that is described in IEC 61508 Ed. 2 a first analysis must be carried out to identify potential hazards originating from the *Equipment Under Control* (EUC). A second analysis aims at assessing the severity of the consequences and the expected frequency of each hazard. This leads to corresponding risks. If the risk is within tolerable limits, risk reduction is not required. Risk reduction can be realized by physical protection (e.g., externally: simple barriers like fences that block access to moving parts) or internal logic (e.g., safety functions added to the EUC, like a simple watchdog timer). The level of risk reduction is specified in the so called SIL followed by a number indicating the order of quantitative and qualitative magnitude.

3) Safe Failure Fraction: Each safety instrumented function is comprised of subsystems, either type A, where the complete failure behavior is known a priori, or type B (i.e., a complex microcontroller). This affects the required *Safe Failure Fraction* (*SFF*) at each SIL. A non-redundant architecture type A subsystem (SIL 2), requires between 60% and 90% safe failure fraction, where for a type B subsystem this would be between 90% and 99%. In a *Failure Mode and Effect Analysis* (FMEA) all components are investigated on the basis of their failure modes – dangerous undetected failures and also safe failures. Given the failure rates for each failure, λ_s denotes the failure rate for a safe failure, λ_{DD} denotes the failure rate for a *Dangerous Detected* (DD) failure, and λ_{DU} denotes the failure rate for a *Dangerours Undetected* (DU) failure. This analysis allows calculating a first *SFF*, even if no diagnostics have been considered. The *SFF* can already be located within the required limits and therefore no diagnostics are needed. In general the *SFF* is defined as:

$$SFF = (\sum \lambda_s + \sum \lambda_{DD})/(\sum \lambda_s + \sum \lambda_{DD} + \sum \lambda_{DU})$$

If the *SFF* is too low, it can be increased by adding diagnostics being capable of detecting previously undetected dangerous failures. That reduces the number of DU failures and the likelihood of random failures decreases likewise. Another important value is the *Diagnostic Coverage* (DC), defined as:

$$\begin{aligned} DC \quad &= \quad \sum \lambda_{DD} / \sum \lambda_{Dtotal}, \\ &\text{where} \quad \sum \lambda_{Dtotal} = \sum \lambda_{DD} + \sum \lambda_{DU}, \end{aligned}$$

which gives the ratio between all dangerous detected failures and all dangerous failures.

4) Quality of Diagnostics and Fault Reactions: The diagnostic coverage and its effectiveness are two quality parameters to quantify the degree of risk reduction. In addition *Mean Time to Failure* (*MTTF*) is determined by the failure rate $\lambda(t)$ of a component. *MTTF* defines the safe execution of a component until the first failure occurs. If $\lambda(t)$ is constant then the *Mean Time between Failures* (*MTBF*) is defined as $1/\lambda$ and the *Mean Time to Repair* (*MTTR*) is defined as the average time between two losses with:

$$MTTR = MTBF - MTTF.$$

The fault reaction can be defined differently with respect to the domain the system is used in. In avionics, for example, the safe state is not to de-energize the EUC, but to keep the flight control system up as long as possible so that the plane can land safely. In industry the safe state is often realized by disconnecting power to a mechanical device, so that moving parts cannot unintentionally lead to accidents. When concepting the fault reaction one should also consider the impact on the availability (uptime) of the system.

B. Safety Related Reliability Analysis for Industrial Products

Reliability is handled covering the entire lifecycle whereas in each phase of the product lifecycle reliability is considered properly. A detailed description of parts of this process can be found in [29]. This section will summarize the proposal of [29] and list only the reliability related tasks:

1) Idea phase: Failure Mode and Effect Analysis; benchmarking, service life needs, reliability goals
2) Evaluation phase: Design assessment, reliability testing, Test-Analyze-and-Fix
3) Development phase: Design maturity testing, Testing of beta product, root-cause failure analysis and correcting, validation
4) Transition phase: Product screening, highly accelerated stress test, environmental stress screening, many others as thermal cycling, shock stabilization etc.
5) Production phase: Reliability monitoring, highly accelerated stress screening, environmental stress screening, many others as thermal cycling, shock stabilization etc.
6) Operation phase: Warranty plan development, field failure tracking, customer feedback, failure analysis, lessons learned etc.
7) End of life phase: Continuous improvement, highly accelerated stress audit etc.

This section addresses reliability and safety processes likewise. In the design phase a first FMEA at component level helps to understand the gap between the actual failure rate, without any hardware changes or software diagnostics and the targeted *SFF*. The Component FMEA could use a worst case scenario by handling every failure as dangerous, thus leading to a severe consequence. A failure mode is the way in which a component fails "functionally" at component level. In a second iteration the effects of failures modes are assessed where some failures might not be dangerous and therefore do not contribute to the

overall failure rate. A final iteration, usually corresponding to the *Failure Modes, Effects and Diagnostic Coverage Analysis* (FMEDA), should also include possible failure mitigations like hardware or software diagnostics. With that, all failures that are defined as "dangerous detected" can be subtracted from the overall failure rate until the target *SFF* has been reached. The next step would be to implement the stated diagnostics and prove their diagnostic effectiveness and coverage by, e.g., fault injection tests. The key challenge is to define a good combination of software and hardware diagnostics that helps reaching the target, but also keeps the development costs to a minimum, otherwise the price of the product would not be competitive.

1) Failure Mode and Effect Analysis: The process of an FMEA would typically include:

- Identification of all components that are relevant for the safety functions and their failure rates
- Classification of the impact of failure modes into safe and dangerous
- Identification of diagnostic functions that can detect dangerous failures
- Final calculation of *SFF* and DC

For a logical element (e.g., a complex system with a microcontroller) in a SIL 2 system typically 15% of the system's failure rate is allocated. The target rate for *Failures in Time* for the logical element must be below 150 FIT, where 1 FIT= 1 failure per 10^9 hours. The 15% is widely accepted by industry and certification bodies. A challenge in an FMEDA is to use meaningful data, which can be only determined experimentally for very new components. For known components there exist various failure rate databases to use [30], [31], [32]. Mixing FIT rates from various databases is forbidden in order not to risk that only the "good" values are considered by the manufacturer.

2) Saving Maintenance costs: Reliability data are often selected based on worst case scenarios, if no real field data exist, because in safety the worst case always defines the upper bound. Usually, the reliability of components in a device will persist over the entire lifetime. However, this is different on engineering level where the reliability of devices can be readjusted if real field data are collected over a long period of time. In particular, today's safety systems include rich functionality sets for measuring various condition parameters at run-time. A typical example is a safety device that is used on an oil-rig where, at least some decades ago, the environmental impact was rather unknown (e.g., how does vibration impact the failure rate?). With modern wireless vibration sensors the wear-out of devices can be readjusted, because the frequency and the amplitude of vibrations are constantly measured. The reason why customers are interested in avoiding the worst-case scenario is because that leads to smaller prove test intervals. Prove test intervals are defined for all field devices, which did not set this interval to its lifetime. This maintenance process leads to significant costs as a part of the overall maintenance. We mentioned that products can have a lifetime of sometimes more than 20 years and, if real field data are collected within this time, the worst case calculation can be recalculated on the basis of field data being collected. This reduces the prove test interval iteratively and therefore also the maintenance costs. Several research projects deal with lifecycle reliability handling.

3) Increasing the Reliability of Industrial Devices: We have already mentioned that there are ways to increase the reliability of an industrial product:

1) All those components contributing with high failure rates should be, if possible and economically feasible, replaced by others with smaller failure rates.
2) Components with unacceptable failure rates can be replicated so that if one component fails, the other can take over. This is typical for the power supply.
3) The failure of components can be detected by diagnostic measures in hardware (e.g., a watchdog timer) or software (e.g., a memory test, that checks for stuck-at-errors).

For the last item it has to be considered that hardware elements lead to higher component costs and more software diagnostics lead to higher development costs, so an optimum has to be found. This is normally the most difficult task, since the development and certification effort for software can be only roughly estimated.

C. Future Challenges

While traditional safety architectures were often comprised of simple hardware circuits and/or reasonably simple microcontrollers, today several research projects explore the usage of multicore processors, since they would provide some obvious advantages:

- By partitioning and virtualization non-safe and safe software could run on the same processor. This leads to
  - hardware cost savings, because only one powerful multicore processor has to be used,
  - more flexibility, because configuration and software upgrades can be done primarily in software,
  - easier migration to other platforms, and
  - a "one platform for all principle" for reduced software maintenance costs.
- Increased hardware performance
- Higher flexibility, because a traditional two-microcontroller solution is harder to update.

Despite all advantages, using multicores implies also many unsolved problems. Proving sufficient diagnostic coverage and

effectiveness becomes increasingly difficult, since multicore processors are very complex and their functionality grows with every new version. Moreover, actual semiconductor manufacturing processes lead to critically small dimensions where soft-errors (transient errors) become significantly dominant. The main challenges for multicore systems are to prove sufficient interference freeness between safe and non-safe functionality, to handle the common cause failure effects, and to utilize the processors performance properly.

In summary, design of reliable safety products is driven by recent standards, modern hardware platforms, the impact of systematic and random failures, and constantly growing software size. We discussed techniques to evaluate and assess the reliability and advised how to bridge the gap between requirements in the safety standards as well as their practical implementations in order to reduce development and certification costs.

V. Conclusions

Overall, reliability evaluation of a product has to consider the different layers of abstraction. Firstly, this is necessary to understand which failures may affect the functionality at a higher level or which failures are safely handled already. Secondly, such an approach provides meaningful data for assumptions like failure rates of components that are required when going from the physical level all the way up to assessing product reliability. Appropriate analytic models and scalable automation tools continue to be the challenges in assessing reliability of future products.

References

[1] ITRS Working Group, *International Technology Roadmap for Semiconductors 2011*. ITRS, 2011, available at http://www.itrs.net.

[2] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.

[3] M. Choudhury, V. Chandra, K. Mohanram, and R. Aitken, "Analytical model for TDDB based performance degradation in combinational logic," in *Design, Automation and Test in Europe*, 2010, pp. 423–428.

[4] H. Wang, M. Miranda, F. Catthoor, and W. Dehaene, "Impact of random soft oxide breakdown on SRAM energy/delay drift," *IEEE Trans. Device and Materials Reliability*, vol. 7, no. 4, pp. 581–591, 2007.

[5] J. McPherson, "Reliability challenges for 45nm and beyond," in *Design Automation Conf.*, 2006, pp. 176–181.

[6] E. Maricau and G. Gielen, "Computer-aided analog circuit design for reliability in nanometer CMOS," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 1, pp. 50–58, 2011.

[7] W. Wang, S. Yang, S. Bhardwaj, S. Vrudhula, F. Liu, and Y. Cao, "The impact of NBTI effect on combinational circuit: Modeling, simulation, and analysis," *IEEE Trans. VLSI Systems*, vol. 18, no. 2, pp. 173–183, 2010.

[8] R. Vattikonda, W. Wang, and Y. Cao, "Modeling and minimization of PMOS NBTI effect for robust nanometer design," in *Design Automation Conf.*, 2006, pp. 1047–1052.

[9] Z. Qi, J. Wang, A. Cabe, S. Wooters, T. Blalock, B. Calhoun, and M. Stan, "SRAM-based NBTI/PBTI sensor system design," in *Design Automation Conf.*, 2010, pp. 849–852.

[10] J. H. Stathis, M. Wang, and K. Zhao, "Reliability of advanced high-k/metal-gate n-FET devices," *Microelectronics Reliability*, vol. 50, no. 1, pp. 1199–1202, 2010.

[11] S. H. Voldman, *SD Basics: From Semiconductor Manufacturing to Product Us*. Wiley, 2012.

[12] S. Wen, R. Wong, M. Romain, and N. Tam, "Thermal neutron soft error rate for SRAMs in the 90nm-45nm technology range," in *IEEE International Reliability Physics Symposium*, 2010, pp. 1036–1039.

[13] T. Grasser, H. Reisinger, W. Goes, T. Aichinger, P. Hehenberger, P.-J. Wagner, M. Nelhiebel, J. Franco, and B. Kaczer, "Switching oxide traps as the missing link between negative bias temperature instability and random telegraph noise," in *IEEE International Electron Devices Meeting*, 2009, pp. 1–4.

[14] Synopsys Inc., "CCS noise technical white paper, version 1.1," 2005, available on opensourceliberty.org.

[15] E. M. et al, "Workload dependent aging simulation and optimization flow in sub-45nm industrial processor," in *Design, Automation and Test in Europe*, 2013.

[16] M. Auslander, D. Larkin, and A. Scherr, "The evolution of the MVS operating system," *IBM Journal of Research Development*, vol. 25, no. 5, pp. 471–482, 1981.

[17] J. Duraes and H. Madeira, "Multidimensional characterization of the impact of faulty drivers on the operating systems behavior," *IEICE Transactions on Information and Systems*, vol. E86-D, no. 12, pp. 2563–2570, 2003.

[18] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "Failure resilience for device drivers," in *Conference on Dependable Systems and Networks*, 2007, pp. 41–50.

[19] W. Gu, Z. Kalbarczyk, R. K. Iyer, and Z. Yang, "Characterization of linux kernel behavior under errors," in *Conference on Dependable Systems and Networks*, 2003, pp. 459–468.

[20] W. Gu, Z. Kalbarczyk, and R. Iyer, "Error sensitivity of the linux kernel executing on powerpc g4 and pentium 4 processors," in *Conference on Dependable Systems and Networks*, 2004, pp. 887–896.

[21] D. Stott, B. Floering, D. Burke, Z. Kalbarczpk, and R. Iyer, "NFTAPE: A framework for assessing dependability in distributed systems with lightweight fault injectors," in *Int'l Computer Performance and Dependability Symposium*, 2000, pp. 91–100.

[22] M. Rimen, J. Ohlsson, and J. Torin, "On microprocessor error behavior modeling," in *International Symposium on Fault-Tolerant Computing*, 1994, pp. 76–85.

[23] R. Iyer, D. Rossetti, and M.-C. Hsueh, "Measurement and modeling of computer reliability as affected by system activity," *ACM Transactions on Computer Systems*, vol. 4, no. 3, pp. 214–237, 1986.

[24] M. Hiller, A. Jhumka, and N. Suri, "On the placement of software mechanism for detection of data errors," in *Conference on Dependable Systems and Networks*, 2002, pp. 135–144.

[25] K. Pattabiraman, Z. Kalbarczyk, and R. Iyer, "Automated derivation of application-aware error detectors using static analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 1, pp. 44–57, 2011.

[26] K. Pattabiraman, G. Saggese, D. Chen, Z. Kalbarczyk, and R. Iyer, "Automated derivation of application-specific error detectors using dynamic analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 5, pp. 640–655, 2011.

[27] EN ISO 13849-1, *Safety of machinery, Safety-related parts of control systems*. International Organization for Standardization, 2008.

[28] IEC 61508, *Functional safety of electrical/electronic/programmable electronic safety-related systems*. International Electrotechnical Commission, 2010.

[29] D. Crowe and A. Feinberg, *Design for Reliability*, ser. Electronics Handbook Series. CRC Press, 2001.

[30] US Military Handbook 217F, *Military Handbook – Reliability Prediction of Electronic Equipment (Mil-Hdbk-217F)*. US Department of Defense, 1991.

[31] W. M. Goble, *Getting Failure Rate Data*. exida.com LLC, 2002, www.exida.com.

[32] SN 29500-1, *Failue rates of components*. Siemens AG, 2005.