# Increasing Reliability Using Adaptive Cross-Layer Techniques in DRPs: Just-Safe-Enough Responses to Reliability Threats

Johannes Maximilian Kühn, Oliver Bringmann, and Wolfgang Rosenstiel

## 1 Introduction

The broad deployment, as well as the increasingly difficult manufacturing of in-spec semiconductors long make reliable operation and failures across the lifetime of an embedded system one of the industry's main concerns. Since ever-increasing demands do no longer allow us to resort to "robust" technologies, other means than semiconductor technology have to fill the gap left by cutting-edge technologies without resorting to unrealistic mainframe like protection mechanisms. As the operation scenarios become ever more challenging as well (edge computing, intelligent IoT nodes), hardware architects are faced with ever tighter power budgets for continuously increasing compute demands. We, therefore, proposed to exploit the architectural redundancies provided by potent, yet energy efficient massively parallel architectures, modeled using Dynamically Reconfigurable Processors (DRP). Using DRPs, we built an extensive cross-layer approach inspired by the overall project's approach as laid out in [1]. Following the idea of cross-layer reliability approaches, we built interfaces reaching from software layers right down to the transistor level mainly through computer architecture, allowing us to address both the varying reliability requirements and the significant computational demands of prospective workloads.

Figure 1 shows an overview of the layers this project targeted as described in the previous paragraph. While a strong focus has been on architecture, the project's aim was to use computer architecture to connect to the layers above and

J. M. Kühn (✉)
Preferred Networks Inc., Tokyo, Japan
e-mail: kuhn@preferred.jp

O. Bringmann · W. Rosenstiel
Eberhard Karls Universität Tübingen, Tübingen, Germany
e-mail: oliver.bringmann@uni-tuebingen.de

J. Henkel, N. Dutt (eds.), *Dependable Embedded Systems*, Embedded Systems,
https://doi.org/10.1007/978-3-030-52017-5_5

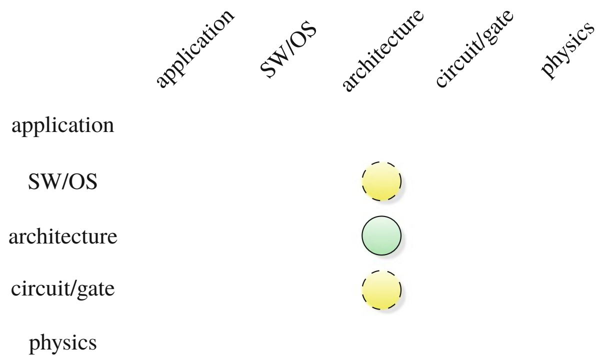


**Fig. 1** Main abstraction layers of embedded systems and this chapter's major (green, solid) and minor (yellow, dashed) cross-layer contributions

below. We show how DRP architectures can leverage their inherent architectural redundancies to realize various degrees of reliable computing. On one end of the spectrum, we highlight how triple modular redundancy (TMR) and duplication with comparison (DWC) compute modes can be realized to actively secure computations without permanently binding hardware resources and with only slight hardware overheads. On the other end of the spectrum, we show how fault-free operation can be passively ascertained by periodically testing SoC components. Both, active and passive concepts together with the architectural redundancies allow for graceful degradation by pinpoint failure detection and subsequently dynamically remapping applications. Once established, both graceful degradation and low-cost TMR for critical parts of applications can be used to make specific operations in processor cores reliable by using the DRP or the demonstrated concepts as a reliable pipeline within a processor core.

A central point of the proposed methods is an overarching cross-layer approach [1], tying together these methods from the software layers (Application, Operating System) to all hardware layers below down to the semiconductor through the concepts introduced by our DRP architecture. To enable a reach down to the circuit level, we exemplarily used the extensive Body Biasing capabilities of Fully Depleted Silicon on Insulator (FDSOI) processes as a means for transistor-level testing and manipulation. This access down to the transistor level enables continuous monitoring of the precise hardware health and thereby not only reactive measures in case of hardware failure but also proactive measures to prevent system failure and prolong system lifetime if the hardware starts exhibiting signs of wear. Access to the device state also multiplies the reliability and system health options on the software layer. With previously having the choice of using TMR/DWC to minimize the error probability, we also show how DVFS with Body Biasing can offer both high power but highly reliable over spec versus ultra-low-power but risky computing modes. These modes' long-term effects further multiply the set of operation modes, e.g., slowing down or speeding up degenerative effects such as Hot

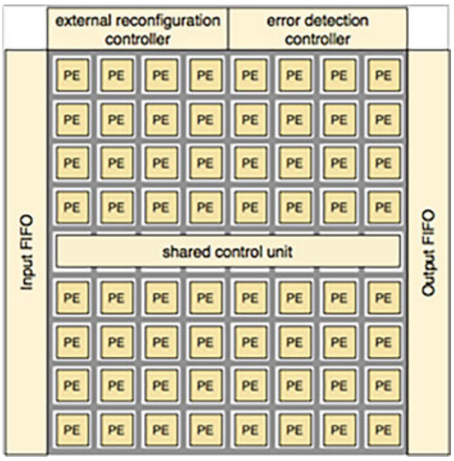
Carrier Injection (HCI) or Negative Bias Temperature Instability (NBTI). However, with access to actual transistor parameters, the proposed approach also indicates that even permanent degeneration such as HCI can be temporarily overcome [2] to prolong system lifetime long enough to extend the graceful degradation period beyond conventional physical limits. Or to put it in the spirit of the parallel NSF effort [3], by opportunistically filling the technology gap using cross-layer methods, there are more means to approach and exploit the hardware's sheer physical limits.

Within this project, we also faced the challenge of how such cross-layer approaches can be realistically validated and evaluated. While Software layers down to the RTL level allow, e.g., fault injection through instrumentation or emulation, the computational effort quickly becomes too large for realistically sized test samples. Furthermore, going below the gate level offers an entirely new set of challenges, both calling for appropriate solutions. For the layers from Software to RTL, we chose to implement the entire system as a prototype on an FPGA. For this FPGA, in turn, we developed a precise fault-injection mechanism so we could emulate the entire SoC with specific faults present. For the gate level and below, we devised a mix of SPICE simulations, and for body bias effect evaluation we ran in-silicon evaluations at the laboratory of Professor Amano at Keio University.

This chapter is structured as follows. Since reliability threats and how such threats surface has been covered in the general introduction, Dynamically Reconfigurable Processors are briefly introduced. The next section directly dives into how the inherent architectural redundancy can be put to use to increase the reliability of computations, as well as how to test these techniques. In the following two sections, the focus then shifts to both ends of the abstraction layers by focussing on how to infer the device state at the transistor level and potentially also recovering from a faulty state using body biasing together with how decisions on the software or operating system level affect the transistor level. The last technical section before wrapping up then brings all levels together by highlighting the interplay between each layer and the synergistic gain thereby achieved.

## 2 Dynamically Reconfigurable Processors

Dynamically reconfigurable architectures, or short DRP, are a sub-category of so-called coarse-grained reconfigurable architectures (CGRA). Similar to Field Programmable Gate Arrays (FPGA), CGRAs are reconfigurable architectures; however, in contrast to FPGAs, CGRAs are reconfigurable on a far coarser level. That is, while FPGAs can efficiently map per-bit configurability, CGRAs only allow reconfigurability on word-sized units. While this restriction makes CGRAs unfavorable for random bit logic, CGRAs possess a far greater area and energy efficiency as the logic overhead for reconfigurability per bit is far lower. DRPs add the concept of dynamic reconfiguration to CGRAs by having on-chip memories for multiple configurations, or contexts, as instructions are often called in DRPs. As the keyword instruction already hints, DRPs resemble much more simple processors


external reconfiguration controller
error detection controller
Input FIFO
Output FIFO
PE
shared control unit
**Fig. 2** Exemplary DRP instance with additional controllers and I/O buffers

than classical reconfigurable architectures, hence this reconfiguration mechanism is also often referred to processor-like reconfiguration.

DRPs at the point of writing date back more than 25 years which makes an exhaustive overview unfeasible. Instead, three different cited surveys shall give both a historical, functional, and up-to-date introduction to the field. De Sutter et al. [4] take a processor-centric view on CGRA architectures using the concept of instruction slots, that is logic where instructions can be executed. These units are connected using a simple form of interconnect like, e.g., nearest neighbor interconnect, and all have shared, or as De Sutter et al. describe them, distributed register files.

On the other hand, Hideharu Amano defines CGRAs and DRPs from a general hardware perspective. In [5], he defines a DRP to be an array of coarse-grained cells as depicted in Fig. 2, so-called PEs, consisting of one or multiple ALU and/or functional units (FU), a register file and a data manipulator [5]. The third and last survey cited for the purpose of an encompassing definition takes a similar approach as the authors of this chapter. In [6], Kiyoung Choi characterizes CGRA and by extension also DRPs via configuration granularity. All authors' definitions encompass an array of PEs and possess dynamic reconfiguration or processor-like execution and thus DRPs as architectural concept range from small reconfigurable DSP like blocks to many-core processors.

In theory, this allows the generalization of findings obtained in DRPs to be extended to far more complex brethren. In practice, however, the definition is restricted by precisely the architectural complexity as DRPs aim to be more energy efficient in more specialized fields other than, e.g., GPGPUs. This becomes also apparent in the general lack of complex caches and big register files, as well as simplistic, spatial interconnects that reduces both register file accesses and long and energy inefficient data transfers [4, 5]. For the purpose of this research project, this minimalism was a welcome attribute as it allowed an abstraction of far more complex architectures while maintaining generality. For this reason, we refer to the cited surveys [4–6] for comprehensive coverage of concrete DRP architectures.

## 3 Exploiting Architectural Redundancy for Increased Reliability

### *3.1 Realizing Low-Cost TMR Using PE Clusters*

Among the most apparent aspects of DRPs is their regular structure. One of the first investigations published in [7] therefore sought to utilize the structural redundancy to increase DRPs' reliability by implementing the quasi-gold standard of fault-tolerance, triple modular redundancy (TMR). The biggest issue of TMR and also the reason why it is only used in critical systems is the prohibitively high cost, i.e., everything that is secured through TMR is triplicated. These triplicated copies then have to perform the exact same operation, and at given checkpoints or most commonly at the block level of the covered component, the outputs are compared. If an error surfaced, the correct result, as well as the faulty component, are determined through a majority vote. The big drawback of this technique is the high cost, both in circuit size since three copies are required, as well as in power consumption as all have to perform the same operation all the time. This makes TMR unviable for all but the most critical applications. With reconfigurable hardware, such as DRPs, however, hardware resource can be dynamically allocated. Given the addition of error detection components, the penalty of TMR can be severely reduced as resources do not have to be committed in a hard-wired fashion, but can be reassigned temporally, or, TMR could be dynamically used for specially flagged parts of a program only.

Figure 3 depicts a simplified representation of the Flexible Error Handling Module. It consists of an actual data error detection module, containing a three-input comparator. The comparator results are fed to the voter and the timing error detection. The voter determines the correct results through a majority vote and feeds the correct channel selection to the multiplexer which then forwards the result that

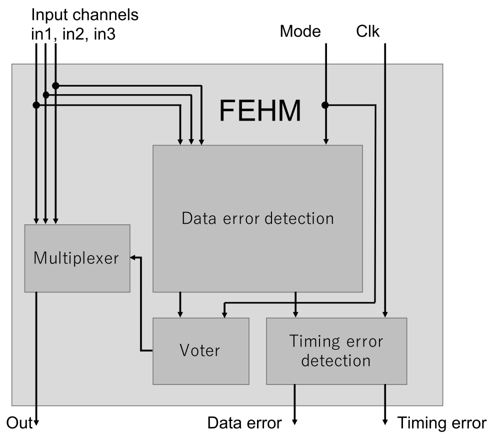


**Fig. 3** The flexible error handling module (FEHM)

is now presumed to be correct to the next PE or out of the DRP. The timing error detection samples the comparison results in a double buffer on `Clk` the clock signal as well as on a slightly delayed clock signal. If the double buffer's contents on each sample are not the same, a timing error occurred and will be appropriately signaled. Similarly, if not all comparison results are equal in the first place, it will raise a data error signal. The entire module's functionality is controlled using the `Mode` signal. Using this signal, the FEHM can be turned off, to Duplicate with Comparison (DWC) mode or to full TMR mode.

This switch is central to the original goal of attaining TMR at lower cost: By making the mode signal part of the instruction word, not only does this free up TMR resources when TMR is not required, but it also allows for some degradation to DWC. Evaluations of this low-cost TMR evaluation showed that even if it is used in relatively primitive DRP architectures with very fine-grained data words, the additional hardware amounts for approximately a 6% increase in area. The power consumption, on the other hand, increased by about 7.5% which can be attributed to the constantly used XOR-OR trees and double buffers used for comparison and timing error detection.

## *3.2 DRPs as Redundancy for CPU Pipelines*

CPUs as central control units in SoCs take a vital role and thus are of great interest for reliability. However, at the same time, they are among the most difficult components to harden against any type of fault if blunt and costly instruments such as TMR are avoided. The extreme degree of dynamism and control involved in CPUs make static redundancy schemes like TMR virtually mandatory if an error-free operation needs to be guaranteed. But if some tradeoffs are permissible, dynamic redundancy schemes can be alternatively used. Such tradeoffs can be for example an absolute time limit until recovery has to complete. In both cases, however, some form of spare component is required.

While DRPs will not be able to take over a CPU's main functions, they certainly could serve as spare compute pipeline [8], thus reducing the parts that need to be hardened using conventional methods. Placing a DRP into a processor's pipeline is not a novel idea such as [9] or [10] demonstrated and makes much sense from an acceleration point of view. However, as this chapter shall highlight, they might be a good pick concerning reliability as well. When used as a static redundancy as depicted in Fig. 4 (left), DRPs can make use of their structural redundancies to provide for additional samples computed in parallel to realize true TMR. The low-cost TMR method proposed in the previous section, on the other hand, can add an additional level of reliability so that the DRP's results can be trusted and false-positives effectively prevented. As dynamic redundancy or as a spare, the DRP can take over functionality if an error has been detected using other means as depicted in Fig. 4 (right). The viability of this approach has been validated in a model implementation inspired by ARM's Cortex-M3 microcontroller. This serves

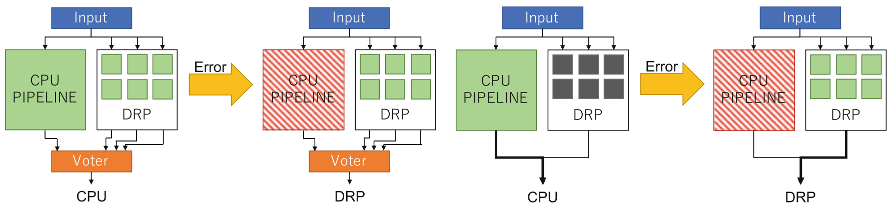


**Fig. 4** DRP serving as static redundancy (left) and as dynamic redundancy (right)

as an interesting choice as ARM has its line of cores for safety-critical applications, the so-called ARM Cortex-R series with support for dual-core lock-step [11]. The results of this study as published in [8] showed that as long as support for division units is omitted in the DRP, the area overhead is far lower than the 100% overhead of an additional core, however, while of course leaving out other components to be secured separately. In this particular study, a 2 by 2 PE array, that is 4 PEs have been integrated into the CPU pipeline. Additionally, instructions and infrastructure to utilize the DRP have been added. Comparing the incurred overheads to a single-core implementation without any reliability measures, the area overhead for an implementation without hardware implemented division amounted to 20%. While this might not be an entirely fair comparison, division implementations in DRPs have a greater impact due to the far greater number of processing elements.

## 3.3 *Dynamic Testing*

In contrast to critical applications, SoCs often also accommodate non-essential functionality. For these applications, running all parts in TMR mode might be wasteful, yet a certain temporal assurance would be desirable. For example, in case of infotainment, brief dysfunction might be tolerable, but if functionality cannot be restored within a given amount of time, actual damage ensues. To avoid TMR or DWC for all applications and to implement time and probability based levels of reliability, we proposed a dynamic testing scheme for reconfigurable hardware.

Dynamic testing or also often called online testing as defined by Gao et al. [12] describes a testing method where for a known algorithm implemented in a certain component, input samples, and associated output samples are obtained and then recomputed separately. If the recomputation's results match the output samples, no error is present. If there is a mismatch, an error of the tested component is assumed.

Specifically using DRPs for dynamic testing has a big advantage: the choice between utilizing the temporal and spatial domains. Instead of competing with applications for resources on the DRP, dynamic testing resources can be allocated temporally and inserted interleaved with applications' instructions to be executed in a time-multiplexed fashion. By moving and interleaving into the time domain, testing becomes slower. However, for most non-critical applications, a couple of seconds before a system returns to a functioning state can be tolerated. Furthermore, the spatial domain allows alternating the compute units used to recompute the samples, further making false-positives less likely apart from the error checking conducted during TMR usage.

While these two aspects make DRPs appealing for such testing schemes, time-multiplexing restricting testing to time-windows $T_{TW}$ and further mapping into the temporal domain slowing down testing by a scaling factor $s$ in combination with the probabilistic nature of error occurrence and detection make any estimation rather difficult. Therefore, Monte Carlo simulations can be used to estimate the behavior of dynamic testing accounting for all DRP specific aspects. For example, aspects such as reconfiguration overhead $T_{OV}$ which has to be deducted from time-windows $T_{TW}$ as well as scaling factors which reduces the number of samples that can be computed within one $T_{TW}$ to detect a fault with an observation probability of $q$.

Consider Fig. 5, depicting a feasibility plot to detect a fault with an observation probability of $q = 10^{-5}$ and a reconfiguration overhead of 1 ms. The goal in this experiment was to detect such a fault within 2 s. The red striped regions indicate that here, it would take more than 2 s to detect the fault, whereas shades from white (fastest) to black indicate increasing detection latency $DL$. This result shows that even if the temporal domain is massively utilized at e.g. $s = 77$, the deadline of 2 s is still met at $DL = 1.7$ s with a time-window of 2 ms for computations thus allowing to use spatially extremely compact mappings for fault detection. This compaction also allows to further share the DRPs resources to conduct periodic checks of the

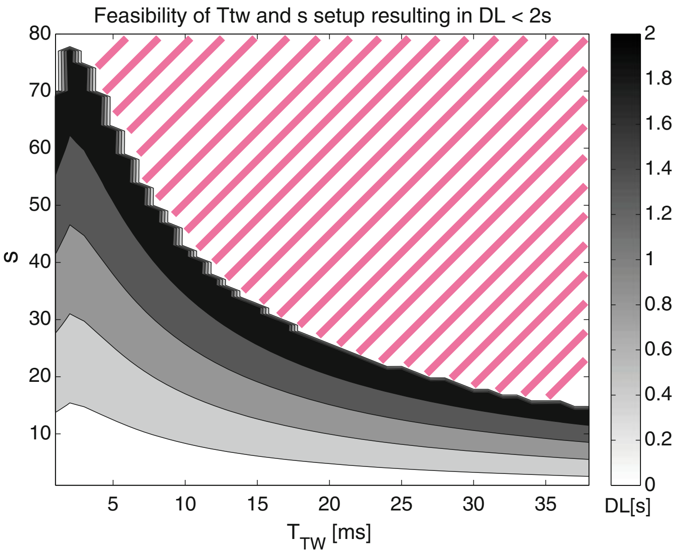


**Fig. 5** Dynamic testing feasibility for a detection latency $DL$ of 2 s by scaling factor $s$ and time-window size for a fault with observation probability $q = 10^{-5}$, $T_{OV} = 1$ ms, a clock frequency of $F = 100$ MHz and $2\sigma$ confidence

entire surrounding SoC, expanding the reach of a reliable DRP to other system components as well.

## *3.4 Dynamic Remapping*

Having various reliable ways to detect errors is vital as any reaction to a false-positive would just turn any reliability mechanism against itself. With low-cost TMR and dynamic testing, we have ways to detect errors and in the TMR case even to mask them. However, once a permanent fault is present and errors surface, TMR degrades to DWC, and dynamic testing is also limited to reasserting the error's presence over and over again. As DRPs are a class of reconfigurable hardware, to restore proper functionality, the applications have to be mapped anew avoiding faulty components. To do this, however, the remapping method and sufficient mapping resources are required.

In case of the FEHM equipped DRP used for our studies, two dimensions of redundancies can be utilized to run the application on unaffected PEs of the DRP. (1) spatially moving the application part of one faulty PE to a fault-free unused PE and (2) temporally adding the application part to an unaffected PE which is used for other application parts but still has the capacity to accommodate this part. As in DRPs the amount of instructions that can be stored and executed without external reconfiguration is limited, compensating for one or more faulty PEs can be a challenge in highly utilized scenarios. However, even if utilization is not critical, just moving parts around on the DRP will yield sub-optimal results, which is why the application mapping, that is resource allocation and scheduling needs to be rerun. This task, however, needs to be run on the SoCs CPU without obstructing normal operation.

To reduce the work-load of the SoC's CPU, we proposed an incremental remapping algorithm in [13]. First, the architecture graph is adjusted by removing the faulty components. Then, from this architecture graph, we extract a subgraph containing the affected PE and its vicinity. Similarly, the application graph is used to extract a subgraph containing only the application nodes mapped to the affected nodes in the architecture subgraph. With these two subgraphs, the mapping is then attempted as exemplarily depicted in Fig. 6. The mapping algorithm will try to first utilize the spatial dimension before resorting to the temporal dimension, i.e. prolonging execution time. If both dimensions do not have the resources to accommodate the application subgraph on the nodes of the pruned architecture subgraph, the architecture subgraph is enlarged by adding further neighboring nodes and remapping is retried until a new mapping has been found or the process fails altogether. If the process succeeds, the application now can run again without any errors occurring, even in non-TMR modes.

This prioritization of subgraph size over runtime, i.e., increasing subgraph size only if both dimensions cannot accommodate the application subgraph is arbitrary and other tradeoffs might be preferable. In this specific case, the priority was CPU

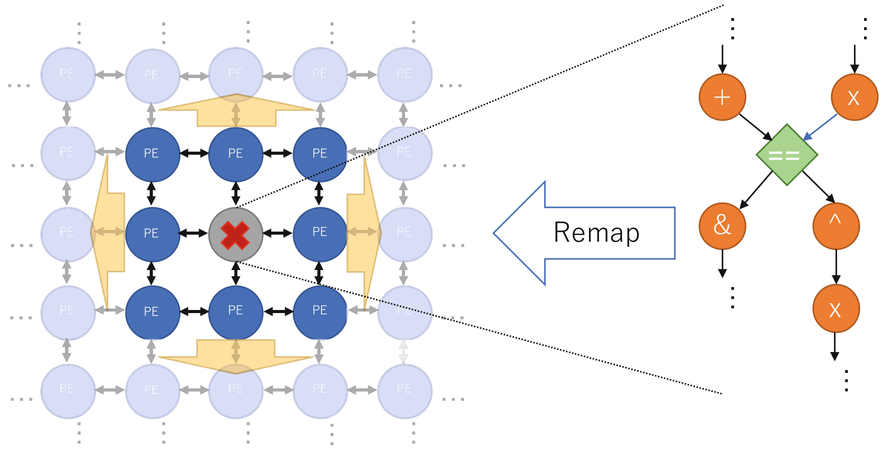


**Fig. 6** Incremental remapping flow on architecture and application subgraphs to avoid faulty components. Starting on the direct neighborhood first, expanding if resources do not suffice

usage minimization, and therefore runtime and memory usage were prioritized by using the smallest subgraphs first at the expense of increased runtimes of the new mappings. For real-world applications, this needs to be carefully weighted as increased runtimes might not be viable.

## *3.5 Testing Reliability Schemes in Hardware*

One of the big challenges of hardware manufacturing and particularly of implementing hardware-based countermeasures to reliability issues is testing and verification. Given the enormous number of input vectors and states, exhaustive testing via simulation is entirely unfeasible. While big commercial hardware emulators allow for a much greater design size and ease of use, they are also very costly. For small to medium-sized designs, FPGAs offer a sweet spot for prototype implementations. While simulations allow for easy fault injection but very slow simulation speeds, FPGAs offer speeds close to ASIC implementations but fault injection was virtually unfeasible.

To develop a prototyping platform, the Gaisler LEON3 SoC [14] served as a template into which the hardened DRP has been integrated. Parallel to this effort, different techniques for FPGA fault injection have been studied [15], culminating in the Static Mapping Library (StML) approach [16]. While instrumentation, i.e. RTL level insertion of faulty behavior allows unlimited choice in fault type and temporal behavior, it also requires for the RTL to be recompiled after each change. As the entire compilation and mapping process of our SoC took more than 4 h, this approach was abandoned. On the other hand, directly inserting faults into FPGA

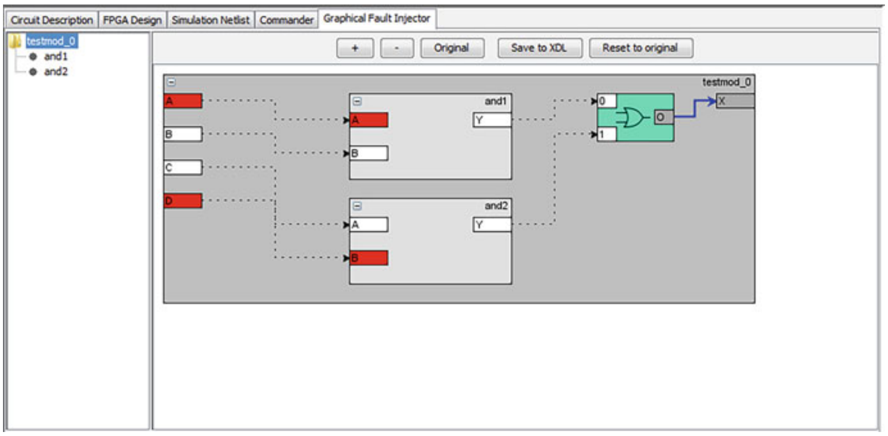


**Fig. 7** StML GUI view of a sample logical AND component with fault injectable ports in each module

mappings or even the bitstream offers a simplistic way to create faulty versions of an FPGA mapping, but it offers no control over the type or location of the injected fault. This approach would not even guarantee that the FPGA mapping would behave in a faulty manner. Ideally, the exact fault location should be specifiable on RTL level to fully qualify the efficiency of the proposed architectural methods. To realize this, different intermediate results were utilized, primarily the FPGA's simulation netlist containing both RTL level structural information and FPGA mapping names in combination with the Xilinx Design Language (XDL) file containing the concrete FPGA mapping. By establishing a bidirectional link between the simulation netlist and the XDL file, StML enabled to pinpoint ports of module's implementation right down to the logic level to insert a stuck-at-zero or stuck-at-one fault. As the placed and routed XDL file can be directly altered, the only remaining step after fault injection is bitstream generation. A user-friendly GUI (Fig. 7) offering graphical representations of the implementation as well as a powerful command line interface allowed for both smooth experiment and extensive testing. Using this approach, we were able to reduce the fault-injection experiment time from hours to below 5 min, with most experiments done in below 2 min.

To showcase the viability of the proposed techniques, low-cost TMR, dynamic testing, dynamic remapping, and the FPGA prototype combined with the fault injection techniques have been successfully demonstrated at ICFPT in 2013 [17].

## 4 Device-Level State and Countermeasures

Below the architectural level, we studied opportunities to determine the state of semiconductor devices. Additionally, we also considered specific device-level

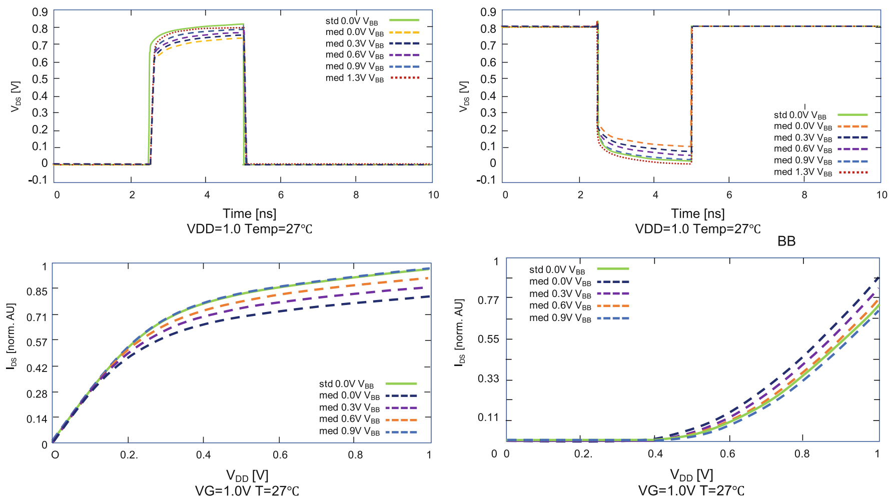


**Fig. 8** Transient simulation for a single switching NMOS (upper left) and a single PMOS (upper right), as well as DC sweep of $I_{DS}$ over $V_{DD} = V_{GS}$ for a NMOS (bottom left) as well as a PMOS transistor (bottom right), all using a medium degeneration model

countermeasures and their effects to put hardware into a more reliable state for tasks that require higher levels of reliability.

When considering how to obtain information on the state at the device level, a transistors' threshold voltage $V_{TH}$ is a central variable [18] to consider. While of course, not all reliability phenomena manifest as an actual shift in $V_{TH}$, they can be modeled as such. For example, stuck-at faults are either a reduction to 0 V or $\infty V$ of $V_{TH}$ or even changes in the drive current and subsequent timing faults can be viewed as such. With the semiconductor world moving either towards FinFET or FDSOI technologies, we investigated the options of FDSOI processes such as ST Microelectronics Ultra Thin Body and Box Fully Depleted Semiconductor on Insulator (UTBB-FDSOI) technology [19]. While being a planar technology, it is manufactured in a triple well process, shielding the transistor body against the substrate using a diode in reverse direction. The transistor is manufactured using a fully depleted channel which allows for further scaling to compete with FinFET processes. One of the main advantages of FDSOI technologies is that the insulated transistor body allows for very high biasing voltages previously unfeasible as it would have shorted the transistor to the substrate. As this thin body with the thin box construction equipped with a separate body electrode acts as a second gate, it is ideal to adjust $V_{TH}$ dynamically after manufacturing. The adjustment of a transistor via this second gate is also called body biasing.

To study the possibilities to use body biasing to detect faults or even faults building up, SPICE level models have been considered. Figure 8 depicts the transient and DC analysis of a medium degeneration transistor-level model. The left side in Fig. 8 depicts an NMOS transistor whereas the right side depicts a PMOS transistor. In each graph, there are several plots: std $0.0V\,V_{BB}$, that is a perfectly functioning transistor without any body bias applied, med $0.0V\,V_{BB}$ transistor with a medium $V_{TH}$ increase and no body biasing, and then several variants of the defective transistor with increasing levels of body biasing. When comparing std $0.0V\,V_{BB}$ to med $0.0V\,V_{BB}$, it immediately becomes apparent that there is a significant gap in the rate at which the signal rises (top two graphs) and signal level, as well as a strong difference in drive current (bottom two graphs). The effects of a $V_{TH}$ (about 45 mV NMOS and 40 mV PMOS) shift of this magnitude are, of course, relative to the operating conditions. If e.g. a couple of such transistors would be used somewhere on a critical path within a high-performance circuit, it would surely fail. On the other hand, if the circuit is used far from timing limits or if only a single transistor is considered, the effect might be barely noticeable. Given an on-chip test circuit or a known critical path, they can be used in conjunction with body biasing to measure degenerative effects. To perform such post-manufacturing bias, ideally each chip should be tested after production with a sweep over body bias levels as described in [20], with the minimum body bias, that is the maximum reverse body bias (the circuit's timing is intentionally slowed down), at which the circuit checked out functional written to a non-volatile memory. Later on, this minimum bias point can be used as a reference, i.e., if the chip or the tested component needs a higher level of body bias corrected for temperature, then some degeneration occurred. If the circuit is designed with reasonable margins, a build-up until an actual fault occurs can be thereby detected.

Similarly, body bias also allows pushing the circuit back conforming to specification. The effect depicted in Fig. 8 would be catastrophic for any performance-oriented component. However, this medium degeneration case has been chosen specifically so that corrective measures can be taken without special electrical precautions, which is up to a $V_{BB}$ of 1.3 V in most processes. However, it should be noted that this also leads to significantly increased leakage levels and would be unfeasible for an entire chip. This being said, it neatly complements DRPs' architectural granularity, i.e. one PE would be coarse enough to mitigate the overheads of an individual body bias domain, yet it is small enough to keep the leakage overhead of strong forward biases down [21]. Additionally, finer steps of, e.g., 100 mV should be used to detect shifts in $V_{TH}$ early on.

## 5 Synergistic Effects of Cross-Layer Approaches

The question following from the previous section is whether to use architectural approaches or device-level countermeasures to achieve a certain reliability objective is an extremely complex and multivariate problem. Beyond the question whether or not to use a specific technique, there are additional variables such as time, i.e. when to use these techniques, extend, that is in what parts to use them and also in regard to criticality, what techniques and with which parameters could be used at all and to what end?

In a very insightful collaboration with the FEHLER project (chapter "Soft Error Handling for Embedded Systems using Compiler-OS Interaction"), their static analysis of program criticality provided powerful means to determine key portions for reliable execution at the application level [22]. By annotating source code with keywords indicating the respective criticality, only those parts marked as critical will be additionally secured using reliability techniques. It thus was not only a great fit for selective low-cost TMR on the hardened DRP, but beyond that offered a proof-of-concept of mixed-criticality applications along with the means to identify portions critical for reliability. In [22], the targeted application was an h264 decoder. As an entertainment application, the primary metric is whether the service is provided at a certain perceived quality level above which actually occurring errors are irrelevant as they are imperceivable.

On the other end of the scale, device state monitoring allows to assess the physical state of a SoC and also its progression over time. On the architectural level, low-cost TMR or DWC allows for continuous checking, whereas dynamic testing makes sure that errors are not left undetected indefinitely, both providing vital information to potential agents. However, as shall be explored below, reactive measures cannot be determined on one layer alone.

Once the device-level state is known, this information can be used on every abstraction layer above. If for example degradation has been detected, this information can be used to minimize physical stresses by using a combination of supply voltage $V_{DD}$ and body bias $V_{BB}$ [2]. As proposed in the previous subsection, a concrete proposal is to counter $V_{TH}$ drift, that is usually $V_{TH}$ becoming larger, by using a forward body bias. As, however, Federspiel et al. found in [2], this will also increase effects like Hot Carrier Injection (HCI) stress which in turn can cause a decrease in drive current. This could lead to a feedback loop as with the method described in Sect. 4, this would appear like a $V_{TH}$ increase and cause more forward bias to be applied, further increasing HCI stress. Thus, such action needs to be a concerted effort on the operating system level with a full view of the system state and the resources available.

For this reason, countermeasures could encompass several different options from the set of available countermeasures with the primary distinction on lifetime extension or securing error-free functionality in the presence of faults. In both cases, it should be noted that both distinctions are only two different takes on graceful degradation. In case the primary goal of reactive measures is lifetime extension, measures which incur less physical stress should be taken. If e.g. the application allows for some degeneration of the service level such as the aforementioned h264 decoding, less effort can be spent on uncritical parts of an application or it could be mapped alongside another application on a DRP. If error-free functionality is the primary goal because, e.g., the application is critical and does not allow for any degeneration, there is a two-step cascade. If the application can be remapped to fault-free components, this should be prioritized. If the resources do not permit remapping or if no resources are left, the SoC can attempt to mitigate the fault through, e.g., a forward body bias at the expense of a potentially shortened lifetime.

In all cases, however, it is clear that information from the application layer, the operating system, i.e. knowledge about what else is running on the SoC, the

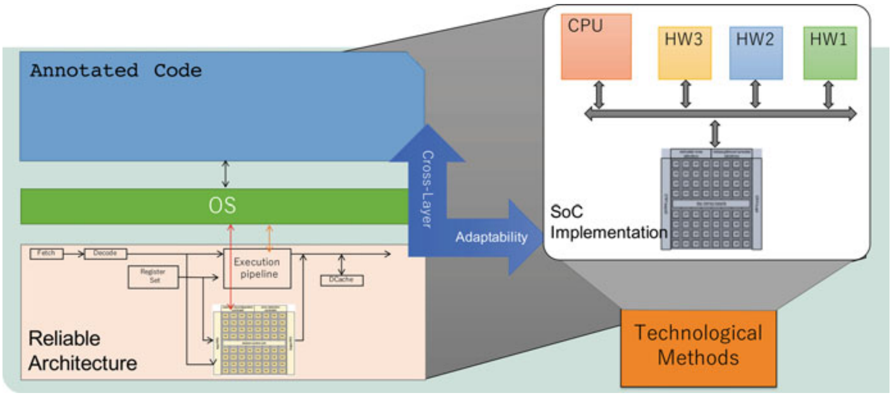


**Fig. 9** Using information of all abstraction layers to realize more reliable and efficient SoC

architectural layer, what resources are available, which resources are inoperable, etc., as well as the device level, are all key to determine the optimal response. In the specific example visualized in Fig. 9, we start at the application layer by assuming reliability annotated source code. Using this source code, an appropriate mapping for example with low-cost TMR onto the DRP can be determined. Additionally, the OS might then go ahead to issue its execution without any special circuit-level tuning, i.e. increasing supply voltage or forward bias to add timing margins. Similarly, a mapping onto the CPU pipeline could be more suitable where the OS then might opt for extra forward bias as some degradation has been previously detected and the application is realizing important functionality. Not only does such a cross-layer approach as visualized in Fig. 9 help to achieve the reliability objectives, but it also is capable of more than what can be achieved on one layer at a time [23]. In this concrete example, the incorporation of multiple layers and multiple methods at specific layers allows to tailor reliability measures to requirements. Device-level information enables the system to act proactively as many phenomena can be detected at this layer in the build-up phase. Once the device layer degenerates, actors such as body biasing allow a system to restore or prolong functionality in the presence of faults.

## 6 Conclusion

Over a generous 6-year period in which this project was funded, the possibilities to use DRPs for increased reliability were extensively studied and also tested in prototype implementations at a functional level. This research revealed that DRPs are not only well suited for tasks that require TMR like reliability, but they can be used in numerous ways to improve the reliability of entire SoCs as well. Their simple and efficient structure allowed to research new and efficient concepts such as dynamic remapping or body biasing for device-level sensing and countermeasures. While DRPs are still undeservingly viewed as a kind of fringe architecture concept, most of the insights gained through such architectures are easily transferable to multi- or many-core SoCs. This project showed that far more can be done in regard to reliability if multiple abstraction layers are considered in a cross-layer approach. While common wisdom still is to use TMR whenever software people use terms such as error-free or fault-tolerant, this project showed multiple options how to incorporate more specific application requirements and how to translate this into adequate reliability measures. Or in simpler terms, just-safe-enough responses to the reliability threats.

**Acknowledgments** As the funding agency of this project, the authors express their gratitude to the DFG for the generous funding throughout the project duration under RO-1030/13 within the Priority Program 1500. The authors also would like to thank STMicroelectronics for the cooperation on FDSOI technology. In regard to the exploration of technological aspects, the authors would also like to express their deep gratitude to the lab of Prof. Hideharu Amano at Keio University and its partnering institutions. It was a tremendous help to see to possibilities of FDSOI in silicon very early on. Finally, the authors would like to thank all the collaborating projects, persons, and especially all the alumni that were involved in this project for the fruitful discussions and interesting exchanges.

## References

1. Herkersdorf, A., Aliee, H., Engel, M., Glaß, M., Gimmler-Dumont, C., Henkel, J., Kleeberger, V.B., Kochte, M.A., Kühn, J.M., Mueller-Gritschneder, D., et al.: Resilience articulation point (RAP): cross-layer dependability modeling for nanometer system-on-chip resilience. Microelectron. Reliab. **54**(6–7), 1066–1074 (2014)
2. Federspiel, X., Angot, D., Rafik, M., Cacho, F., Bajolet, A., Planes, N., Roy, D., Haond, M., Arnaud, F.: 28 nm node bulk vs FDSOI reliability comparison. In: 2012 IEEE International Reliability Physics Symposium (IRPS) pp. 3B–1. IEEE, Piscataway (2012)
3. Gupta, P., Agarwal, Y., Dolecek, L., Dutt, N., Gupta, R.K., Kumar, R., Mitra, S., Nicolau, A., Rosing, T.S., Srivastava, M.B., et al.: Underdesigned and opportunistic computing in presence of hardware variability. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **32**(1), 8–23 (2012)
4. De Sutter, B., Raghavan, P., Lambrechts, A.: Coarse-grained reconfigurable array architectures. In: Handbook of Signal Processing Systems, pp. 427–472. Springer, Berlin (2019)
5. Amano, H.: A survey on dynamically reconfigurable processors. IEICE Trans. Commun. **89**(12), 3179–3187 (2006)
6. Choi, K.: Coarse-grained reconfigurable array: Architecture and application mapping. IPSJ Trans. Syst. LSI Des. Methodol. **4**, 31–46 (2011)
7. Schweizer, T., Schlicker, P., Eisenhardt, S., Kuhn, T., Rosenstiel, W.: Low-cost TMR for fault-tolerance on coarse-grained reconfigurable architectures. 2011 International Conference on Reconfigurable Computing and FPGAs, pp. 135–140. IEEE, Piscataway (2011)
8. Lübeck, K., Morgenstern, D., Schweizer, T., Peterson, D., Rosenstiel, W., Bringmann, O.: Neues Konzept zur Steigerung der Zuverlässigkeit einer ARM-basierten Prozessorarchitektur unter Verwendung eines CGRAs. In: MBMV, pp. 46–58 (2016)
9. Mei, B., Vernalde, S., Verkest, D., De Man, H., Lauwereins, R.: ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix. In: International Conference on Field Programmable Logic and Applications, pp. 61–70. Springer, Berlin (2003)

10. Hoy, C.H., Govindarajuz, V., Nowatzki, T., Nagaraju, R., Marzec, Z., Agarwal, P., Frericks, C., Cofell, R., Sankaralingam, K.: Performance evaluation of a DySER FPGA prototype system spanning the compiler, microarchitecture, and hardware implementation. In: 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 203–214. IEEE, Piscataway (2015)
11. Arm Cortex-R Series Processors. https://developer.arm.com/products/processors/cortex-r. Accessed 8 March 2019
12. Gao, M., Chang, H.M., Lisherness, P., Cheng, K.T.: Time-multiplexed online checking. IEEE Trans. Comput. **60**(9), 1300–1312 (2011)
13. Eisenhardt, S., Küster, A., Schweizer, T., Kuhn, T., Rosenstiel, W.: Spatial and temporal data path remapping for fault-tolerant coarse-grained reconfigurable architectures. In: 2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, pp. 382–388. IEEE, Piscataway (2011)
14. LEON3 Processor. https://www.gaisler.com/index.php/products/processors/leon3?task=view&id=13
15. Schweizer, T., Peterson, D., Kühn, J.M., Kuhn, T., Rosenstiel, W.: A fast and accurate fpga-based fault injection system. In: 2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines, pp. 236–236. IEEE, Piscataway (2013)
16. Peterson, D., Bringmann, O., Schweizer, T., Rosenstiel, W.: StML: bridging the gap between FPGA design and HDL circuit description. In: 2013 International Conference on Field-Programmable Technology (FPT), pp. 278–285. IEEE, Piscataway (2013)
17. Kühn, J.M., Schweizer, T., Peterson, D., Kuhn, T., Rosenstiel, W., et al.: Testing reliability techniques for SoCs with fault tolerant CGRA by using live FPGA fault injection, In: 2013 International Conference on Field-Programmable Technology (FPT), pp. 462–465. IEEE, Piscataway (2013)
18. Maricau, E., Gielen, G.: CMOS reliability overview. In: Analog IC Reliability in Nanometer CMOS, pp. 15–35. Springer, Berlin (2013)
19. Pelloux-Prayer, B., Blagojević, M., Thomas, O., Amara, A., Vladimirescu, A., Nikolić, B., Cesana, G., Flatresse, P.: Planar fully depleted SOI technology: The convergence of high performance and low power towards multimedia mobile applications. In: 2012 IEEE Faible Tension Faible Consommation, pp. 1–4. IEEE, Piscataway (2012)
20. Okuhara, H., Ahmed, A.B., Kühn, J.M., Amano, H.: Asymmetric body bias control with low-power FD-SOI technologies: modeling and power optimization. IEEE Trans. Very Large Scale Integr. Syst. **26**(7), 1254–1267 (2018)
21. Kühn, J.M., Ahmed, A.B., Okuhara, H., Amano, H., Bringmann, O., Rosenstiel, W.: MuCCRA4-BB: a fine-grained body biasing capable DRP. In: 2016 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS XIX), pp. 1–3. IEEE, Piscataway (2016)
22. Schmoll, F., Heinig, A., Marwedel, P., Engel, M.: Improving the fault resilience of an H.264 decoder using static analysis methods. ACM Trans. Embed. Comput. Syst. **13**(1s), 31:1–31:27 (2013). http://doi.acm.org/10.1145/2536747.2536753
23. Herkersdorf, A., Engel, M., Glaß, M., Henkel, J., Kleeberger, V.B., Kochte, M., Kühn, J.M., Nassif, S.R., Rauchfuss, H., Rosenstiel, W., et al.: Cross-layer dependability modeling and abstraction in system on chip. In: Workshop on Silicon Errors in Logic-System Effects (SELSE) (2013)