Towards a Flexible Fault-Tolerant System-on-Chip

C. Albrecht, R. Koch, T. Pionteck
Institute of Computer Engineering
University of Lübeck
surname@iti.uni-luebeck.de

P. Glösekötter
Fachbereich
Elektrotechnik und Informatik
Fachhochschule Münster
gloesek@ieee.org

Abstract

Today's technology advances still lead to increased integration densities and higher clock rates. However, these advances are more and more accompanied by continuously increasing drawbacks such as intra-die-variation, temperature dependencies, and device degradation mechanisms. Besides the classic measure mips-per-watt, the overall system reliability is steadily becoming more important. In this context, device reliability has become a critical aspect of system-on-chip (SoC) design. To cope with this challenge, we divide the SoC itself into different architectural layers. Each layer is tailored individually to the specific SoC needs in terms of fault-tolerance. At the same time, we derive a comprehensive method of how to account for all layer dependencies in an efficient manner and yet enable error detection and correction mechanisms at system level. In particular, error detection is predominantly established at lower levels, whereas required error correction mechanisms are applied at higher system levels.

1 Introduction

Continuous CMOS scaling allows for an increasing number of transistors per die. This enables the design of more and more powerful systems such as system-on-chip (SoC) at reduced manufacturing costs, an increased clock frequency and lower power consumption. Adverse effects are high intra-chip variabilities of transistor parameters, an increased impact of temperature as well as degradation and radiation [2]. While for former technologies these effects could be neglected, reliability issues now become more and more important.

There is a wide variety of techniques usually applied to cope with runtime faults such as triple modular redundancy (TMR), yet they all result in a significant area and power overhead. Taking TMR as an example, this technique requires three instances of one hardware module and a majority voter to detect and mask faults. Fault detection and correction is done at the same time at register transfer layer of a SoC. Yet, the question is whether fault detection and correction can be divided among different SoC abstraction layers and how it can be achieved. Single event upsets (SEU), for example, can be detected at the technology layer by monitoring the transistor bias current. An error detection can be signalled to a higher layer where appropriate actions can be taken to correct this error. This mechanism also allows for a flexible reaction on faults, depending on the degree of fault-tolerance required for a certain application. Taking the idea of a universal SoC architecture as a hardware platform, a single system architecture can be used for different application areas with a configurable degree of reliability.

In order to systematise this approach, a SoC is divided into several layers. All layers include different fault-tolerance mechanisms based on their specific place in the layer model. By combining the fault-tolerance techniques of different layers, an efficient and reliable design regarding run-time errors, design errors and production errors can be realised. Such a SoC architecture would allow for adaptation with regard to the question: How much reliability is mandatory for a specific application? A key technique to suit this requirement are interfaces between neighbouring layers to inform about the needed level of reliability. Additionally, in order to report detected faults in the opposite direction interfaces have to be established so that error detection and correction are principally performed independently.

In the following, the layered SoC model is introduced in Section 2. Section 3 summarises possible errors and faults of SoCs and sketches origin and effects. Subsequently, methods for fault detection and recovery are assigned to each layer and their interoperability is explained by means of examples from literature in Section 4. Finally, before a conclusion is given in Section 6, the interoperability of all layers is is discussed with regard to image processing applications in Section 5.

2 SoC Layer Model

The generic layer model divides a system-on-chip into at least four layers. Reconfigurable architectures include a fifth layer that allows for the implementation of almost arbitrary applications on the same device. These devices often are field-programmable gate arrays (FPGAs). Static system-on-chip designs do not include this feature.

Table 1. SoC layer model from top to bottom

Layer	Description
Application Layer	(user) interface, system specification and constraints
Architectural Layer/ Middleware	components/functional modules, bus, network-on-chip, software/software abstraction layer
Configurable/ Programming Layer (optional)	configuration memory (SRAM, Flash)
Register/Logic Layer	SRAM, ROM, latches, logic gates
Technology Layer	CMOS

The set-up of the model is given in table 1. The optional layer is greyed out. In the following all layers are described with regard to their specific functionality and fault scenarios:

Technology Layer. Device reliability has become a critical aspect of CMOS technology. Smaller device dimensions and the introduction of new materials, such as copper/low-κ dielectric, high-κ gate dielectrics, and metal gate electrodes, have a significant impact on CMOS device reliability. For instance, for gate lengths smaller than 50 nm, the total number of dopant atoms is less than 100 in the device depletion region. As the standard deviation of fluctuations is equal to the square root of the number of dopant atoms, the $\pm 3\sigma$ boundaries becomes extremely large when the gate length is scaled down to 30 nm [11].

Further, it is identified by the International Technology Roadmap for Semiconductors (ITRS), that beyond the 90 nm node, single event transients (SET) severely impact field-level product reliability, not only for embedded memory, but for logic as well [5]. Besides, threshold voltage change caused by negative bias temperature instability (NBTI) for the PMOS transistor has become the dominant factor to limit the lifetime [15]. Thus, the efficiency with which a precise variability characterisation can be performed is going to become important. Therefore, it will be no longer sufficient to do it once at the stage of technology bring up. New measures of design optimisation are needed for continuous re-evaluation. Hence, the probabilities of failure have to be reported to the next SoC Layer [8].

Register/Logic Layer. At this layer the system is seen as a combination of memory and logic elements. The memory elements divide in SRAM, ROM and latches, each with a different SEU tolerance [14]. For larger memory elements several methods for error detection and correction exist, while combinational circuits are often unprotected. On the one side this is due to the assumption that combinational circuits are less vulnerable to SEUs than memories, on the other side this is due to the large overhead required to harden combinational logic. Yet, with further decreasing of features sizes, logic elements will become as vulnerable as memory elements [14]. Another problem arises from the fact that strict separation between the failure rate of memory elements and combinational logic is not always applicable. For example, when data in a register file are protected by EDAC (error detection and correction) codes, faults affecting the read/write logic are not covered. In particular for multi-port register files the area required for combinational logic exceeds the area used for the actual memory elements [1].

Configuration/Programming Layer (optional). Hardware platforms providing a configuration layer most often use SRAM to store the configuration code. Each memory cell is assigned to a part of the logic or routing boxes so that interconnect and function can be configured by the contents of the memory. Unfortunately, radiation also effects these memory cells. Changes in the configuration memory also change the function or interconnect configured by it. These faults can be detected and avoided by redundancy or detected by tests and corrected by loading a new configuration into the memory [3]. If this procedure fails a permanent fault is assumed.

Middleware/Architectural Layer. This level includes component-based architectures with communication infrastructure such as buses or network-on-chips and functional units. Furthermore, it also includes soft units such as protocols and middleware used as abstraction layers implemented in software to provide programming interfaces for data exchange within a device. In the functional units any kind of fault might occur. Detection and avoidance are performed by usual techniques such as redundancy and test mechanisms. The communication parts may use codes, checksums, and even routing mechanisms to ensure a safe data transfer.

Application Layer. The specific semantics of all combined components is introduced and the target application is implemented at the top layer. Today most systems use software at the highest layer to provide a user interface and establish a more or less complex abstraction of the underlying device.

3 Faults

Faults can arise at different stages: at design time, at production time or at run time. Because of the non-restricted application area for system-on-chips the variety of potential effects of faults is huge.

The design of a SoC is in general a complex issue that includes a lot of perils and pitfalls. Thus, even a SoC design carefully manufactured and tested may include faults. These design faults often are wrongly interpreted specifications for rarely appearing use cases.

Production errors are unfortunately independent of a design. Thus, any effort to build a clean design can be thwarted. Here, the technology layer has a huge impact. Beside impurities disturbing e.g. interconnects on the die, downscaling and the introduction of new materials influence the device reliability. In particular, downscaling leads to more and more complex production processes and expressing their complex non-linear realities via design rules is becoming difficult at best [7]. The Design/Technology interface information bandwidth needs are skyrocketing. Another result of downscaling is the growing impact of device degradation. For instance, as already mentioned, threshold voltage change caused by NBTI has become the crucial factor to limit the life time [15].

Transient faults are mainly caused by outer conditions such as temperature and radiation. The latter in particular is responsible for single event upsets. SEUs have different impact on the logic state. If a part of a memory cell is struck, the stored value can be inverted. Depending on the semantics of the memory cell, processed data or configuration data is changed. While the first leads to a wrong result with generally unpredictable consequences the latter impacts on the functionality. That is why this kind of SEU is called single event functional interrupt (SEFI). Instead of storage elements the logic can be hit, too. In this case, it is called a single event transition. In addition changing transistor behaviour also results in different timing properties and, thus, failures on the register/logic layer where synchronisation and timing properties play an important role. Based on Monte Carlo simulation, Dao et al. [4] examined the variation of the propagation time due to process variability. While statistical variations of set-up and propagation times in critical paths significantly influence the maximum chip performance, a violation of the hold time in short FF-logic-FF paths leads to a chip failure due to a generation of races in the pipeline. Race conditions are caused by the combination of short paths, clock skew, and jitter between sending and receiving FFs, and process variations.

Transient faults occurring at layers above the technology layer are often propagated through the hierarchy and have their origin in one of these reasons.

Table 2 summarises faults and their causes per layer. At least all these aspects have to be taken into account while designing a new SoC.

Table 2. Fault summary for SoC layers.

Layer	Faults and/or Causes
Application Layer	behaviour, design, specification
Architectural Layer/ Middleware	protocol specification, transmission error, data error, timing error
Configurable/ Programming Layer (optional)	SEFI
Register/Logic Layer	SEU, SET
Technology Layer	device degradation, production errors, timing errors, SEU, SET

4 Methods and Interfaces

The countermeasures that can be taken against the occurence of faults are manifold. Mechanisms for fault detection and recovery can be incorporated at every level. Basic techniques are structural, informational, and time redundancy. Structural redundancy utilises the replication of components to perform the same task multiple times in parallel to ensure the result by majority vote. Informational redundancy extends components by functions to detect and avoid faults, e.g. numbers could be represented by binary codes instead of simple dual representation. Finally, time redundancy reuses the same components for multiple executions of the same computation sequentially and determining a result by majority vote again. With regard to a specific application a general fault-tolerant system might implement a lot of reliability functionalities which are hardly ever needed, but consume power and also time for computation. The overhead could be reduced by passing on information about the degree of reliability actually required by the application and by combining different techniques of different layers. Especially, the fault detection can be performed at the lower levels whereas the upper levels know whether a fault is worth to be corrected or not.

The goal to achieve an appropriate reliability level of the complete SoC system demands adaptation capabilities

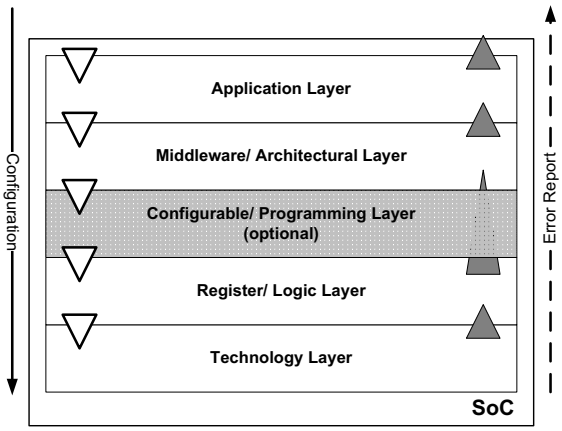

Figure 1. Layer model of a generic System-on-Chip.

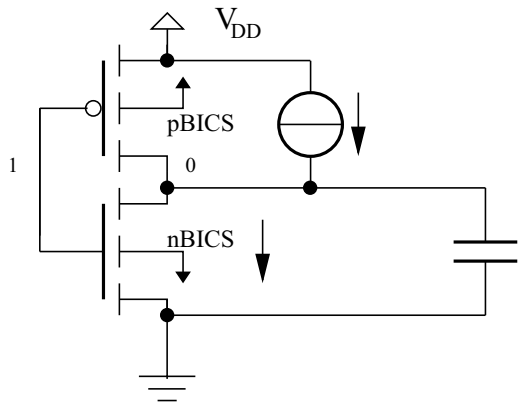

Figure 2. Transient current source models the charge deposition mechanism.

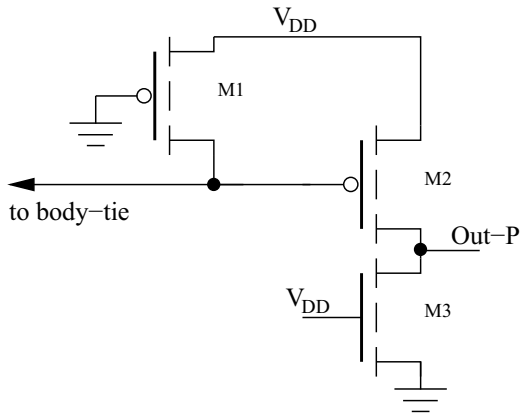

Figure 3. A bulk-BICS for SET detection. This pBICS detects particle strikes at PMOS transistor.

at any level. The overall knowledge is present within the application so that interfaces for informing lower layers about current reliability needs are demanded in a top-down manner. The system-on-chip layer model in figure 1 shows two sorts of uni-directional interfaces. On the left side configuration interfaces for configuring reliability level of the lower layer, and on the right side interfaces for error reports to be evaluated at the upper layer are sketched. Particularly, the detection of errors caused by transient faults is principally easier at lower levels whereas an appropriate reaction to such an error can be determined more reasonably at a higher level of abstraction.

The following sections will show appropriate methods and techniques for each level and give examples for their application. Further on, the requirements for the interface design of inter-layer communication is described. The configuration interface, e.g., enables redundancy techniques mentioned above. Conversely, the error report interface passes on information about the current fault situation.

4.1 Technology Layer

Since technology has been evolving continuously, featuring smaller and smaller transistors, soft errors on devices occur more frequently. Connecting a built-in current sensor (BICS) in the design bulk of a digital system the sensitivity for detecting transient upsets in combinational and sequential logic [8] can be increased.

If an energetic particle strikes a sensitive region in a semiconductor device, the resulting electron hole pair generation can cause a transient current that may alter the logical state of the circuit. The charge deposition mechanism produces a transient pulse that lasts until the deposited charge is conducted away via open current paths to V_{DD} or ground, returning the logic node to its original state. If pulse amplitude is high enough and pulse duration is long enough, the pulse may propagate through subsequent circuit stages and change the results of a computation. Hence, pulse amplitude and duration are key parameters for evaluation of circuit sensitivity to SEU.

For instance, the drain junction of the reverse biased p-transistor in Figure 2 tends to be vulnerable. As current flows through the pn-junction of the struck transistor, the transistor in the on-state (n-transistor) conducts a current that attempts to balance the current induced by the particle strike. If the current induced by the particle strike is high enough the on-transistor cannot balance the current and a voltage change at the node will occur. This voltage change lasts until the charge is conducted away by the current feed through the on-transistor.

Since under normal circuit operation the bulk current is

very small if compared to the current generated by a particle strike, this approach allows the bulk-BICS (Figure 3) to distinguish SETs from internal logic signals and consequently detect them. As a consequence, the bulk-BICS may be used to detect transient faults in digital combinatorial logic and mixed signal circuits, as well as in memories [17].

An advantage of this design is its negligible static power consumption, a key feature for VLSI. This bulk-BICS detects an SET due to an energetic particle strike at the reverse biased drain junction of PMOS transistors (Figure 2) in the off state. The body-ties of the PMOS transistors being monitored are connected to V_{DD} through transistor M1 of the bulk-BICS. To ensure proper connection of the body-ties to V_{DD}, the PMOS transistor M1 has a large W/L ratio. L is the channel length and W the channel width. The PMOS transistor M2 also has a large W/L ratio, while the NMOS transistor M3 has a small W/L ratio. Under normal operation (no particle strike) the current flowing through M1 is negligible, and the gate of M2 is at V_{DD}. As a consequence, the output Out-P of the bulk-BICS is at logic zero. If a particle strike occurs, current flows through M1, and the gate voltage of M2 slightly drops. The bulk-BICS amplifies this voltage change, and the output voltage at Out-P rises, changing its value from logic zero to logic one. Out-P flags the occurrence of an SET to the upper system level where error handling techniques may then be implemented at the circuit or system level, aiming to avoid that an SET turns into a faulty behaviour. A similar design (nBICS) may be used to detect particle strikes at the drain of NMOS transistors in the off state [8, 17].

The interface to the technology layer enables the upper layers to configure the capabilities of this layer according to the actual needs within a certain context. At design time, mechanisms as described above can be enabled or disabled while at run time there is the possibility to power up or power down parts of the design such as the underlying circuits of a redundant register. More advanced designs can also provide an interface to dynamic voltage-scaling and dynamic frequency-scaling capabilities. In the opposite direction the technology layer merely signals to the upper layers whether a fault has been detected and possibly corrected in a certain part of the design or not.

4.2 Register/Logic Layer

Classical approaches to detect and correct faults at register and logic layer are based on structural and informational redundancy. EDAC codes such as Hamming codes or Reed-Solomon codes are well suited to detect and correct bit errors in memories. Faults in combinational circuits can be detected by duplicating the circuit. TMR even offers the capability to correct faults. While informational redundancy cannot mask all faults in combinational logic and memory cells, full structural redundancy causes too much area and power overhead. By using the information of the technology layer the decision can be made whether and where an error has occurred, the third hardware instance and voter of a TMR implementation can be avoided as the lower level can signalise which hardware instance is effected by an SEU. If complete fault coverage is not required, a selective hardening can be done. Therefor, the susceptibility of individual gates to faults resulting in a system malfunction has to be evaluated by a fault propagation analysis. Faults in the speculation unit of a processor for example will not lead to wrong results but will only cause some performance penalties. In the same way the characteristics of data stored in memory elements has to be considered. Analysis in [16] showed that faults affecting the address fields for memory operations in a processor are less severe than faults in the valid bits throughout the pipeline.

The register/logic layer communicates the number of detected and possibly corrected faults within a period of time to the upper layers. The other way round, the upper layers can adjust the ability of this layer to detect and correct faults by requesting or disabling the use of redundant codes and registers.

4.3 Configuration/Programming Layer

The optional configuration/programming layer of this model is present only in case of an underlying run-time reconfigurable platform. This platform may provide fine-grained or coarse-grained reconfigurability. In both cases, the reconfigurability of the platform provides the basis for applying late-binding techniques.

There are two levels of binding which have to be considered: First, the functionality of the architectural modules defined at the middleware/architectural layer is bound to functional modules defined at this configuration/programming layer. These functional modules are, in turn, bound to particular resources of the reconfigurable platform, e. g., to configurable logic blocks (CLBs) of an FPGA or to the more complex processing elements (PEs) of a coarse-grained architecture.

Two different, yet compatible functional modules might, for example, differ in that one is optimised for speed while the other provides intrinsic fault-tolerance. By dynamically re-binding the functionality of architectural modules from one to the other of such two functional modules, the degree of fault-tolerance provided for specific parts of the system can be adapted at runtime and upon request by the upper layers.

On the other hand, by dynamically re-binding a functional module to platform resources, such a module can be relocated within the reconfigurable area of the underlying platform. This allows for circumventing faulty regions as

reported by the lower register/logic layer. Moreover, a periodical relocation of active modules can help reduce device degradation at hot-spots and, thus, provides a means for fault-prevention and prolongation of the lifetime of the device.

Employing the techniques outlined above, reconfiguration can be used

1. to replace throughput optimised versions of data processing modules with fault-tolerant versions and vice versa,
2. to group together functionally equivalent modules which were, until then, executing independent tasks to redundant modules for use in, e. g., a TMR scheme,
3. to prevent excessive wearout of hot-spots at the technology layer,
4. to actually activate spare modules specified in one of the adjacent layers,
5. to dynamically reconfigure the module interconnect with the objective of either enabling alternative communication paths or decreasing latency and increasing bandwidth.

The complexity of the upstream interface of the configuration/programming layer heavily depends on the flexibility offered by a particular implementation. Capabilites such as internally managed auto-relocation of modules or dynamic rebinding generally require an advanced controlling instance within this layer. In this case, a sophisticated protocol may be used for communication with the upper layers. In contrast, a simple implementation might hardly offer more than making, e. g., the internal configuration access port (ICAP) of an FPGA accessible to the architectural layer.

4.4 Middleware/Architectural Layer

The architectural layer may simply apply well-known redundancy techniques at component level such as TMR [10]. Further on, the modules to be used in a redundant scheme can be independently designed so that SEU and design errors are detected and corrected. A less area consuming approach is the addition of test functionalities which allow for testing a module, e. g., by means of test patterns. For instance, the network-on-chip CoNoChi uses a certain test-packet scheme for error detection and localisation [12]. Instead of huge area overheads for module replication, each module is extended to support processing of these packets. Coding schemes such as parity codes and multiple transmissions are usually used for on-chip interconnects to ensure a fault-free communication at this level. Another capability to be utilised at this layer is the task or process scheduling applying multiple heterogeneous components. Tasks may be mapped onto dedicated processing elements or realised as a software task on a processor. For a fault-free long-term processing checkpointing is used. Each checkpoint stores an intermediate result and the process state. In case a fault occurs, the computation can be set back to the last fault-free checkpoint and continue. Migrating tasks from hardware to software and vice versa is challenging and demands an accurate analysis of the different process models applied for hardware and software units. Approaches to realise a middleware providing this functionality exist [6].

Reliability improvements cannot only be achieved in hardware but also in software. By protecting program memory with software EDAC and scrubbing, the reliability of a program can be improved by several orders of magnitude compared to an unprotected memory [13]. At the same time software implementations provide much higher flexibility than hardware solutions. By adapting the coding scheme and data interleaving to the characteristics of the data at runtime, software solutions might even outperform hardware solutions in the case of multiple bit errors [13]. A practical way to reduce the overhead of fault tolerant systems is to perform only fault correction at the architectural level while leaving the responsibility for fault detection to the lower layers. When a fault is signalised by a lower layer, fault correction can be done by repeating the computation or, for example, by flushing a processor pipeline. Additional fault tolerance can be achieved by adding control and data redundancy. Data redundancy can be achieved by duplicating each variable and applying each write operation to both copies. If the variables differ, an error has occurred. Methods providing control redundancy include the labelling of basic blocks and procedures, and the repeating of conditional control instructions. With these methods the error rate can be reduced by a factor of 4 [9].

The middleware/architectural layer provides an architecture-specific interface to the application layer. Depending on the complexity of the architecture and the degree of flexibility offered for the configuration of fault-tolerance properties, this range extends from configuration options which allow to statically enable or disable TMR at design time to message interfaces which, at run-time, accept commands accurately describing the current minimum and maximum fault-tolerance requirements. For example, an architecture can provide an interface to an application which allows the latter to set the number of detected faults within certain parts of the architecture which will cause the system to be halted as a precaution against serious malfunction.

4.5 Application Layer

The application layer allows for almost everything to enhance the level of reliability. The designer can include any technique and utilise each feature provided by the target platform to achieve a reliable design as far as all constraints are met. At this layer the application can be completely implemented in a redundant manner e.g. to utilise TMR combined with design diversification in order to counter design and run-time errors at once. The application of fault-tolerance techniques is almost unlimited. Therefore, the considerations set the focus on the reliability level required on a set of applications utilised in different domains.

5 Discussion

Typical fault-tolerance techniques are mainly restricted to a single layer where they try to detect, correct and avoid errors. The effort to achieve any of these goals differs. In some layers detection is easily performed whereas correction is difficult and expensive. Therefore, the combination of different techniques at different layers could reduce the effort to build a reliable SoC. In addition, the SoC should be adaptable to the needs of the application. Tolerable faults do not need to be corrected at any level.

At least, the distribution of fault detection and correction among two layers is applied with success. For instance, a combination of configuration and architectural layer is often used for FPGA designs. The Venus Express Monitoring Camera is set up utilising the reconfigurability features of an FPGA-based SoC. Osterloh et al. [10] achieved an appropriate and sufficiently robust design for space requirements. They applied configuration memory scrubbing at the configuration layer and triple modular redundancy at the architectural layer. Combining both techniques a tolerable rate of non-correctable radiation-based errors of 3.9 per year is achieved.

For building reliable SoCs, masking effects between layers may also be exploited. Faults at a lower layer do not necessarily have an impact on the higher layers. In [16], the impact of soft errors induced in RAM cells and latches of a superscalar, dynamically-scheduled pipelined processor is examined. Fewer than 15% of single bit errors result in errors visible at the software level. As, in turn, half of these visible errors are masked by the software, 9 out of 10 transient faults are prevented from affecting correct execution of the program.

For a certain system, the combination of fault detection and correction mechanisms of different layers for diverse system components strongly depends on the application area and data dependencies within functional blocks. In general, a classification of the reliability requirements into three classes, low, medium and high can be done. A low reliability for system components means that no special precautions are taken to detect and correct faults of any types. Categorising an application area into the low reliability class presumes that a faulty system behaviour is tolerable for the environment. On the other side, a high reliability indicates that all possible efforts are done in order to avoid and mask system faults. Application areas not tolerating any faults are also categorised into this class. In between is the medium reliability class. Here, the reliability to be achieved is constrained by an acceptable hardware, performance and power trade off. The interdependencies between application area, processing tasks and reliability class can be exemplified by means of systems for image processing.

In mobile service robots, orientation and object identification is often done by analysing camera pictures. The reliability requirements for these image processing steps depend on the frequency and resolution at which pictures are taken. When pictures are taken at a low frequency or the resolution is very low, pixel faults can have a significant impact on the accuracy of orientation. In this scenario, the image processing tasks can be put into the medium reliability class. When pictures are taken at a higher frequency or the resolution is increased, the impact of a single pixel fault is less severe. Albeit the image processing tasks remain the same, for this application scenario a low reliability class can be used. Finally, any medical application should be put into the class requiring high reliability efforts because data loss or data faults can generally not be accepted.

Thus, the same image processing tasks have to be categorised into the high reliability class. With a flexible fault-tolerant SoC architecture, the same hardware architecture could be used for all application scenarios.

6 Conclusion

The given sample applications share a lot of similar image processing techniques although the reliability requirements differ strongly. A flexible fault-tolerant system-on-chip architecture allows the application to raise or to reduce the reliability level at different layers. So, the overhead caused by reliability can be reduced to a minimum. In return, power and performance can be adapted to the reliability level and may be saved in contrast to platforms without such a flexibility.

References

[1] J. A. Blome, S. Gupta, S. Feng, and S. Mahlke. Cost-efficient soft error protection for embedded microprocessors. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pages 421–431, Seoul, Korea, 2006. ACM.

[2] S. Borkar. Designing Reliable Systems from Unreliable Components: The Challenges of Transistor Variability and Degradation. *IEEE Micro*, 25(6):10–16, 2005.

[3] C. Carmichael and C. W. T. Tseng. XAPP988: Correcting Single-Event Upsets in Virtex-4 Platform FPGA Configuration Memory. Technical report, Xilinx, March 2008. v1.0.

[4] H. Q. Dao, K. Nowka, and V. G. Oklobdzija. Analysis of Clocked Timing Elements for Dynamic Voltage Scaling Effects over Process Parameter Variation. In *International Symposium on Low Power Electronics and Design*, pages 56–59, 2001.

[5] ITRS. *International Technology Roadmap for Semiconductors*, chapter Design, pages 6–7. ITRS, 2005.

[6] D. Koch, T. Streichert, S. Dittrich, C. Strengert, C. Haubelt, and J. Teich. An Operating System Infrastructure for Fault-Tolerant Reconfigurable Networks. In *Proceedings of the 19th International Conference on Architecture of Computing Systems (ARCS 2006), Frankfurt / Main, Germany*, pages 202–216, Frankfurt, Germany, Mar. 2006. Springer.

[7] S. Nassif. Model to Hardware Matching for Nanometer-scale Technologies. In *The Impact of Process Variability on Design and Test*. Design, Automation and Test in Europe, DATE, 2008.

[8] E. H. Neto, I. Ribeiro, M. Vieira, G. Wirth, and F. L. Kastensmidt. Using Bulk Built-in Current Sensors to Detect Soft Errors. *IEEE Micro*, 26(5):10–18, 2006.

[9] B. Nicolescu and R. Velazco. Detecting soft errors by a purely software approach: method, tools and experimental results. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 57–62 suppl., 2003.

[10] B. Osterloh, H. Michalik, B. Fiethe, and K. Kotarowski. SoCWire: A Network-on-Chip Approach for Reconfigurable System-on-Chip Designs in Space Applications. In *AHS '08: Proceedings of the 2008 NASA/ESA Conference on Adaptive Hardware and Systems*, pages 51–56, Washington, DC, USA, 2008. IEEE Computer Society.

[11] C. Piguet. *Low-Power CMOS Circuits*, chapter Evolution of Deep Submicron Bulk and SOI Technologies. Taylor & Francis Group, 2006.

[12] T. Pionteck, C. Albrecht, R. Koch, T. Brix, and E. Maehle. Design and Simulation of Runtime Reconfigurable Systems. In *11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, pages 1 – 4. IEEE, 2008.

[13] P. Shirvani, N. Saxena, and E. McCluskey. Software-implemented EDAC protection against SEUs. *Reliability, IEEE Transactions on*, 49(3):273–284, 2000.

[14] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi. Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic. In *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 389–398. IEEE Computer Society, 2002.

[15] R. Vattikonda, W. Wang, and Y. Cao. Modeling and Minimization of PMOS NBTI Effect for Robust Nanometer Design. In *Proceedings of the 43rd Annual Conference on Design Automation (DAC)*, pages 1047–1052, New York, NY, USA, 2006. ACM.

[16] N. J. Wang, J. Quek, T. M. Rafacz, and S. J. patel. Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, page 61. IEEE Computer Society, 2004.

[17] G. Wirth. Bulk Built in Current Sensors for Single Event Transient Detection in Deep-Submicron Technologies. *Microelectronics Reliability*, 48(5):710 – 715, 2008.