Microprocessors and Microsystems 37 (2013) 981–1001

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

DeSyRe: On-demand system reliability ☆

I. Sourdis [a,*], C. Strydis [f], A. Armato [g], C.S. Bouganis [e], B. Falsafi [c], G.N. Gaydadjiev [a], S. Isaza [f], A. Malek [a], R. Mariani [g], D. Pnevmatikatos [d,i], D.K. Pradhan [b], G. Rauwerda [h], R.M. Seepers [f], R.A. Shafik [b], K. Sunesen [h], D. Theodoropoulos [d], S. Tzilis [a], M. Vavouras [e]

[a] *Computer Science and Engineering Dept., Chalmers University of Technology, Sweden*
[b] *Computer Science Dept., University of Bristol, UK*
[c] *Computer and Communication Sciences, EPFL, Switzerland*
[d] *Institute of Computer Science (ICS), Foundation for Research and Technology – Hellas (FORTH), Greece*
[e] *Electrical and Electronic Engineering Dept., Imperial College London, UK*
[f] *Neurasmus B.V., The Netherlands*
[g] *Yogitech SpA, Italy*
[h] *Recore Systems B.V., The Netherlands*
[i] *Electronic and Computer Engineering Department, Technical University of Crete, Greece*

ARTICLE INFO

Article history:
Available online 11 September 2013

Keywords:
Fault-tolerance
System-on-chip
Reconfigurable hardware
Medical systems

ABSTRACT

The DeSyRe project builds on-demand adaptive and reliable Systems-on-Chips (SoCs). As fabrication technology scales down, chips are becoming less reliable, thereby incurring increased power and performance costs for fault tolerance. To make matters worse, power density is becoming a significant limiting factor in SoC design, in general. In the face of such changes in the technological landscape, current solutions for fault tolerance are expected to introduce excessive overheads in future systems. Moreover, attempting to design and manufacture a totally defect-/fault-free system, would impact heavily, even prohibitively, the design, manufacturing, and testing costs, as well as the system performance and power consumption. In this context, DeSyRe delivers a new generation of systems that are reliable by design at well-balanced power, performance, and design costs. In our attempt to reduce the overheads of fault-tolerance, only a small fraction of the chip is built to be fault-free. This fault-free part is then employed to manage the remaining fault-prone resources of the SoC. The DeSyRe framework is applied to two medical systems with high safety requirements (measured using the IEC 61508 functional safety standard) and tight power and performance constraints.

1. Introduction

In the coming nanoscale era, chips are becoming less reliable, while manufacturing fault-free chips is becoming increasingly more difficult and costly [1,3]. Prominent causes for this are the shrinking device features, the sheer number of components on a given area of silicon, as well as the increasing complexity of current and future chips. It is expected that a significant number of devices will be defective already at manufacture time and many more will degrade and fail within their expected lifetime [2]. Furthermore, process variations as well as the increasing number of soft errors introduce additional sources of errors for future chips.

The ITRS targets a constant defect rate (1395 defects/m^2) in order to keep the chip yield constant [6]. Such a target is expected to substantially increase the chip manufacturing cost of future semiconductor technologies. Alternatively, chips need to be designed to tolerate an increasing number of defects in order to maintain a high yield. Apart from defects at manufacture time, aging effects are becoming more severe leading to more permanent and intermittent faults during the lifetime of a chip. Transistors degrade faster; while the degradation rate is further accelerated by the heavy testing processes (e.g. burn-in). Aging is expected to shorten SoC lifetime and to be a significant source of errors in technologies beyond 16-nm [1]. Process variations cause devices to operate differently than expected; such variations are random dopant fluctuations, heat flux, as well as lithography problems due to the shrinking geometries. Currently, on-chip clock frequency and total power consumption present variations up to 30% and 50%, respectively, across different parts of a single chip; it is projected that variations will only become more severe in the future and worst-case, deterministic design will be insufficient and unable to deliver reliable systems. Finally, as transistor count increases, the number of soft errors on a chip (i.e. transient faults)

☆ The DeSyRe Project is supported by the European Commission Seventh Framework Programme, grant agreement n° 287611. www.desyre.eu

* Corresponding author.
E-mail address: sourdis@chalmers.se (I. Sourdis).

0141-9331/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.micpro.2013.08.008

grows exponentially [2]. For example, by the 16-nm generation, the failure rate will be almost 100-fold higher that at 180-nm [2]; current fault-tolerance techniques such as simple check-pointing will, then, incur prohibitively high energy and performance costs.

As feature size continues to shrink and chips become less reliable, the cost for delivering reliable chips is expected to grow for future technology nodes. The price in system power and energy consumption, in performance degradation, and in extra resources, is getting higher in order to perform redundant computations in time or in space. However, it is a well-known fact that power consumption is becoming a severe problem, while performance no longer scales very well (mostly due to power-density limitations). To reduce some of the above costs, *DeSyRe aims at reliable systems containing and tolerating unreliable components rather than targeting totally fault-free systems*. Our goal is to describe a new, more efficient design framework for SoCs, which provides reliability at lower power and performance cost.

Although the above technology trends make the design of future SoCs harder, one of them can be turned to our advantage. As shown in Fig. 1, the increasing power density limits the gate density. In a few years, significant parts of a chip will be forced to remain powered-down in order to keep within the available power budget [14]. *In DeSyRe, we capitalize on this observation and propose to exploit the aforementioned unused resources to offer flexibility and reconfigurability on a chip*. Until now, reconfigurable hardware had a significant resource overhead; however, as explained above, this limitation no longer exists as on-chip resources are becoming "cheaper". A dynamically reconfigurable hardware-substrate can provide an excellent solution for defect tolerance; it can be used to adapt to faults on demand, isolate and correct defects, as well as to provide spare resources to substitute defective blocks. In the DeSyRe project, we intend to use such a reconfigurable substrate and combine it with system-level techniques to provide adaptive and on-demand reliable systems.

The remainder of the paper is organized as follows: in Sections 2 and 3, we describe the DeSyRe design framework and system architecture, respectively. Sections 4 and 5 summarize the DeSyRe fault-tolerance techniques and reconfigurable substrate. In Sections 6 and 7, we present the medical applications and the baseline SoCs used in DeSyRe. Finally, we draw our conclusions in Section 8.

2. The DeSyRe Design Framework

Systems-on-Chip comprise multiple design levels, ranging from top-level running software to hardware components and all the way down to the elementary technological (transistor) substrate. Although increasing design complexity has so far been kept in check by partitioning the design in horizontal levels (e.g. software, hardware, technology), this approach is rapidly becoming inefficient due to the increasing degree of cross-level sophistication required for modern embedded-system design. Knowledge of levels both above and below a specific level is becoming imperative for building functional systems. This implies a vertical, cross-cutting, level partitioning with designers having knowledge of what transpires above and below their respective levels of expertise.

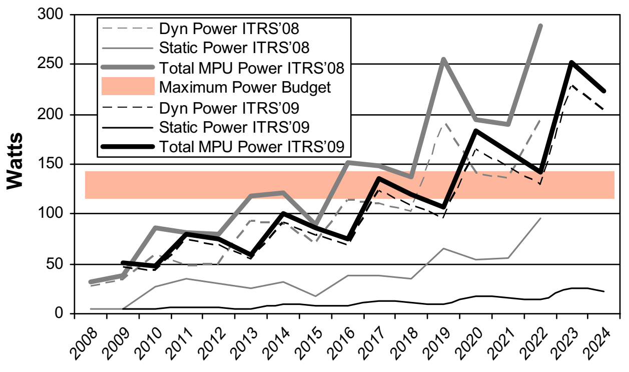


Fig. 1. Dynamic, Static and total power consumption of a Microprocessor chip vs. the maximum available power budget, based on the ITRS 2008 and 2009 projections.

To address this pragmatic need, the DeSyRe framework is partitioned across two orthogonal design dimensions: a *physical* and a *logical abstraction*. The physical partitioning is based on the different technological substrates (with different fault densities) used in the various parts of the framework and is mostly of interest to experts closer to the technology level. The logical partitioning considers the same framework from the viewpoint of functionality (i.e. which part of the system does what) and should mostly interest experts closer to the architecture and system level.

2.1. Physical Partitioning

Fig. 2 illustrates the physical partitioning of the DeSyRe SoC. The design area is physically divided into a fault-free (FF) section providing overall system management and a fault-prone (FP) section providing the actual system functionality. The motivation for this partitioning is reducing the chip cost: Designing a totally fault-free system is expensive, thus the FF section should be small and lightweight.

Fault-Free section: The FF section is required to provide centralized, system-wide control of the SoC, aiming to provide Quality of Service (QoS) attributes such as performance, low power consumption, resource utilization and fault tolerance. The various techniques through which this will be achieved involve an efficient combination of:

- Online fault tolerance.
- Runtime task scheduling, while being aware of task characteristics such as urgency and safety–criticality.
- Resource allocation, under varying availability of computational resources.
- Reconfiguration schemes to achieve flexible and defect-tolerant operation.

Fault-Prone section: The FP section is under the direct control of the FF section. It contains various components realized in the

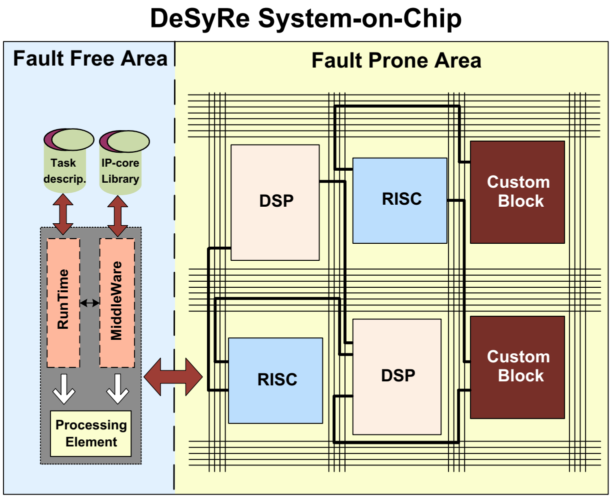


Fig. 2. DeSyRe SoC physical partitioning with a fault-free section for SoC management and a fault-prone section for SoC functionality.

DeSyRe reconfigurable substrate. The components implement the main system functionality based on the target application (domain). They are required to exhibit, among others, self-checking and self-correcting properties, working in tight synergy with those of the FF section. To this end, the various components should be equipped with their own self-checking and -correcting mechanisms, albeit under (in)direct control of the FF section.

2.2. Logical Partitioning

The logical partitioning organizes the DeSyRe SoC in three main layers. Fig. 3 depicts the layers from bottom to top: components, middleware and runtime system. This subdivision is based on the abstraction level involved and the tasks handled by each layer.

Components: The bottom layer deals with fault-tolerance issues of each component (i.e. unit which delivers a specific functionality) in the FP section, individually. In other words, it is responsible for providing component-level (intrinsic) fault tolerance. The system is composed of multiple heterogeneous components located at the fault-prone section. The design of these components takes into account the requirements of the DeSyRe system. The components are able to autonomously detect faults that might appear in them and possibly correct a subset of these detected faults (in other words, they exhibit self-checking and self-correcting properties). To deal with the aforementioned faults, each component further has a certain degree of flexibility, by being able to support task migration – that is, to receive a new task or transfer a running task elsewhere. Those features, of course, require interfacing with the above logical layers, to be able to send status information and receive commands concerning possible modifications.

For the list of faults with which a component deals intrinsically, any detection and/or correction action from the component side has to be transparent to the upper layers. In case there is partial recovery – affecting functionality and QoS constraints – or no correction whatsoever, the upper layers have to be notified about the new status of the component. Component-level fault tolerance, as opposed to centralized techniques, provides recovery schemes with the advantage of lower latency. On the other hand, these schemes are, as a matter of fact, less efficient than the ones supported by the upper logical layers.

Middleware: The second layer is responsible for the hardware synthesis and reconfiguration of the components in order to provide correctly functioning underlying hardware to the upper layer. In this layer we develop mechanisms for the dynamic reconfiguration of hardware resources implemented in the (fine- and/or coarse-grain) reconfigurable FP section. The Middleware manages the components as black boxes which need to be interconnected, isolated, (re)placed in order to deliver a functioning hardware platform ready to be used for running the tasks that the runtime system dictates. To one extent, the Middleware makes all its actions transparent to the Runtime system in order to provide well-functioning hardware; that is in order to ensure the reconfiguration process is performed correctly and installs a correct new configuration.

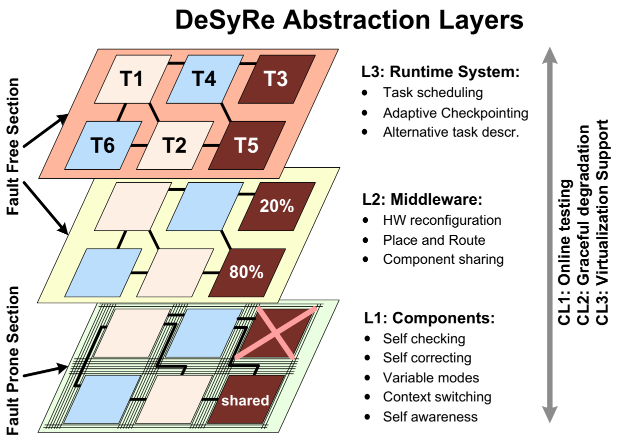


Fig. 3. Logical partitioning of a DeSyRe SoC: Components layer (SoC functionality), Middleware layer and Runtime-System layer (SoC management).

Runtime System: The third layer deals with run-time issues of the system; its basic functionality is to schedule tasks to components, to ensure the best quality of service for the soft real-time portion of the applications, and to adapt the system in the presence of faults. To deal with faults, the Runtime System collects system health status information for all system components to establish a system health map. Using the health map together with a description of the application tasks and their requirements, the Runtime System identifies a global assignment of tasks to the various system resources that satisfies the application requirements and achieves the best possible performance from the (possibly faulty) processing cores in the fault prone area. The Runtime System functionality is realized in the FF section of the SoC (running on a GPP). The main challenge for the runtime system is to be adaptive so to conform to the underlying reconfigurable hardware and to use it efficiently.

In addition to the above logical layers, DeSyRe is involved with three *distributed tasks* that span across the three logical layers, since all three of them need to deal with and support them:

- **Online Testing**: Dynamically scheduled tests for fault detection and diagnosis, in order to constantly update the components' status.
- **Graceful Degradation**: Loss of performance and/or functionality is preferred to a system crash.
- **Virtualization**: Tasks will be able to be executed on different, heterogeneous components as a solution to component unavailability due to faults.

These distributed tasks will be further addressed in Section 4.

3. DeSyRe System Architecture

The DeSyRe system architecture is designed to fit the physical partitioning and logical layers described in the previous section. Fig. 4 illustrates the Software and Hardware modules that constitute the DeSyRe SoC. The software modules are running on a processing unit located in the fault free part, and implement the runtime system optimizations, the middleware functionality for reconfiguration, as well as centralized software implemented fault-tolerance mechanisms. At the fault-prone part, each component has a wrapper to interface with the fault-free part of the system and to locally support functionalities pertinent to reconfigurability and fault-tolerance.

The main hardware components of the system are instances of a simple, MIPS-like, 5-stage RISC processor (SiMS), of a fixed-point high performance DSP processor (Xentium), and some custom application specific components. Each of these cores is equipped with local checkers, detecting and reporting faults, and some degree of reconfigurability. The interconnection mechanism is a Network on Chip utilizing a 32-bit bidirectional datapath and XY-routing. Each component is provided with a wrapper, which locally supports critical tasks by the following means: Component reconfiguration is controlled from the wrapper; test vector generation and response analysis for online testing can be performed locally; a watchdog is used to detect unresponsive components; the state of components can be observed and controlled to facilitate

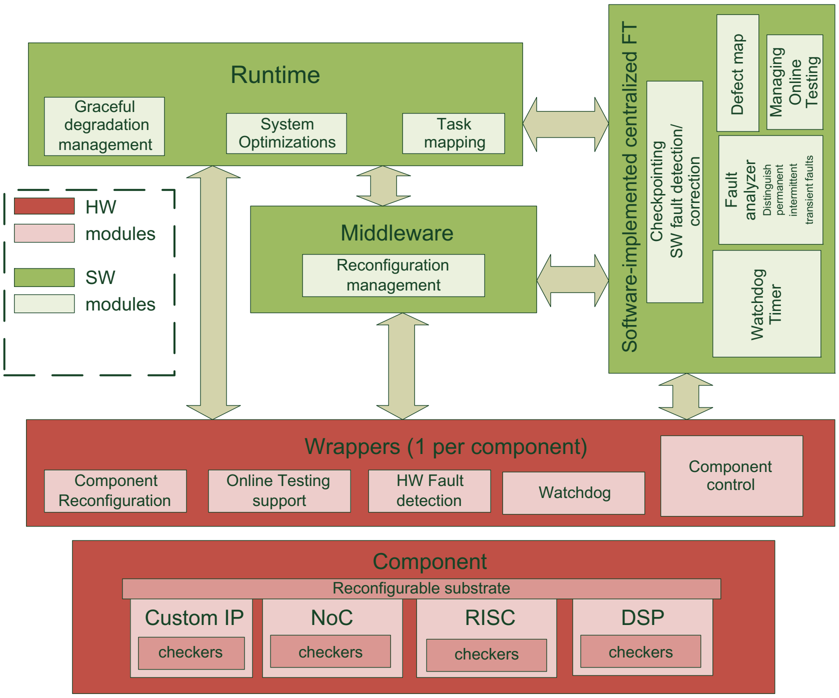


Fig. 4. DeSyRe System Architecture, illustrating the primary Hardware and Software modules.

both check-pointing and online testing; finally, faults detected in the local checkers are recorded and reported to the upper system layers.

The system hardware as it is described above, is being managed by software running on the fault-free part of the system. A Runtime system constantly assesses the status of the hardware components and makes decisions on the overall system configuration: The system is dynamically optimized based on the availability of components, the presence of defects, the workload and possibly other constraints. Graceful degradation is employed whenever new permanent effects are discovered to be taking place. The decisions taken can involve remapping of the tasks to the available components or even modification of the set of components through hardware reconfiguration. In the latter case, a Middleware module is used, that will reconfigure hardware as requested, having knowledge of the reconfigurable resources available at the time.

The overall system status, which is the raw input with which the Runtime works, is monitored by the Software Fault Tolerance layer. This part of the system employs software routines for the following functions: Saving check-points and using them whenever transient faults dictate rollback of system operation; analyzing and classifying the faults detected in hardware as transient, intermittent or permanent; deciding on the need of and managing the execution of online testing to help with the aforementioned fault classification; accumulating the signals from the different components' watchdogs and using them to keep track of (un)responsive resources. A defect map is also kept by the Software Fault Tolerance layer, to record all relevant information about defect-free and defective hardware parts.

In the remainder of this section we discuss specific system-level parameters and consideration in DeSyRe.

3.1. Execution and programming model

The envisioned DeSyRe systems consist of multiple and potentially heterogeneous Processing Elements (PEs). Mapping DeSyRe applications to such a substrate, requires that computations are partitioned and assigned for execution to these processing blocks. In addition to the basic code execution, we need to address the application and system reliability requirements. At the software level, one important functionality necessary to deal with potential execution errors (stemming from hardware faults) is task re-execution. To be able to re-execute correctly the tasks, we need to be able to save and restore their processing state. After investigating alternatives, we concluded that an efficient way to address these requirements is to adopt a *Task-Based* execution model, in order to simplify the system design without loss of generality. This decision affects the overall design in two ways: (a) the programming model specifies how the application developers should write their code to be used in the DeSyRe platforms, and (b) it sets the requirements for the DeSyRe Runtime System (RS).

Task-based programming models require that the programmer defines – via explicit language constructs of code annotation – portions of code that will be wrapped as a "task", which will be the unit of scheduling work for execution. In addition, to allow the exploitation of parallelism, task inputs, outputs and in-outs (variables that are read and written by the same task) are declared as such in the source code. Doing so enables the compiler to statically analyze dependencies and determine the possibility of parallel execution of tasks. The same information can be also used dynamically when the static analysis is insufficient to determine the dynamic independence of tasks.

Several task-based approaches have been investigated and shown to be efficient in high-performance computing environments e.g. [16]. We use a similar approach to the OMPSs [16], but more restricted in the sense that not all the OMPSs features are adopted in our programming model. In DeSyRe, the programmer will have to follow the task-based programming model, i.e. divide the application into different tasks, and annotate them with the task definition directives, so that the task's inputs, outputs, and in-outs are identified. With the code annotation of all application tasks, the Runtime System can efficiently identify any data dependencies, and schedule tasks in correct sequence without the need for synchronization. In addition, the Runtime System can also take into account other constraints i.e. energy consumption, real-time constraints, etc. In the DeSyRe context, dependency checking will be done dynamically to match the dynamic nature of the system that changes due to faults. Thus adopting a task-based execution model has the following consequences:

- Inter-task communication can happen only at the task boundaries.
- Checkpoints need only be saved at the task-boundaries, and as a consequence, domino effects during rollback are avoided.
- Architectural processor (component) state needs not to be stored or transferred to another processor during task-migration, since migrating the task description and its inputs is sufficient to re-execute. In this way, task re-execution in case of faults is simplified.

DeSyRe adopts a Partitioned Global Address Space (PGAS) [27], where the fault-free processor and all PEs can have access to any address. Specific memory regions are (physically) local to PEs, and can be used efficiently for their tasks execution. Moreover, part of the global memory address space will be used as a global system storage (GSS) region. The latter keeps common software data structures of the Runtime System such as task dependency and application variable tables, which are accessible by all PEs and the fault-free processor, in order to control task scheduling and execution.

3.2. System state management

Storing reliably the system state is a major issue under the presence of hardware faults and SEUs. Migration and checkpointing of tasks require maintaining the state reliably upon spawning tasks, updating the state upon completion, and migrating the state to a subsequent task. While conventional memory array error detection/correction techniques incur high latency and energy overhead in the presence of high fault rates, there is a number of emerging techniques that enable detection and correction at much lower power. We are investigating the performance, energy and reliability tradeoffs of a number of techniques to overcome memory vulnerability for checkpointing/migrating task state at each PE of the DeSyRe SoC. We are considering a diverse set of fabrication technology nodes to determine the performance, energy, and area overhead for a task with a given execution time (and rate of memory updates), while varying the fault rate. We are also studying idempotency analysis as a way to reduce updated state during task execution time. In particular we have considered decoupled codes [17,18] to protect memory arrays in one node vs. parity-protected redundant memory arrays placed on different two distinct nodes across the NoC.

Our initial results indicate that energy-efficient, decoupled codes offer a superior performance, energy and area trade off over maintaining duplicate copies across the NoC for near-term and medium-term technology nodes. However, duplicate copies across the NoC have the added benefit of protecting against NoC failure and disconnected nodes. According to these initial results, we consider a fault-free substrate that can send the task state to a compute node on the fault-prone substrate upon task spawn, as shown in Fig. 5. This state is the (architecturally-visible) input and output to the task. The state can be directly communicated across subsequent task spawns among the compute nodes and need not be maintained on the fault-free substrate. This practically confirms the suitability of the task-based programming model chosen for the DeSyRe architecture.

(1) Task re-execution

The aforementioned task-based execution model and system state-management and checkpointing approach, have been implemented in evaluated in a Microblaze-based MPSoC mapped to a Virtex6-based ML605 development board. The MPSoC consists of a master Microblaze (MMB) to host the runtime system, and 7 workers (WMBs) for tasks execution. We implemented a micro-benchmark that performs a reduction tree of matrix multiplications. Starting with 24 matrices, we performed a set of 12 concurrent matrix multiplications. As soon as they are done, a reduction phase begins with each pair being multiplied to produce a new matrix, and after 5 such steps, a single output matrix is calculated. The total number of multiplications is 24, requiring in total 6 steps due to the limited number of worker cores, and therefore the maximum theoretical speedup is 24/6 = 4×.

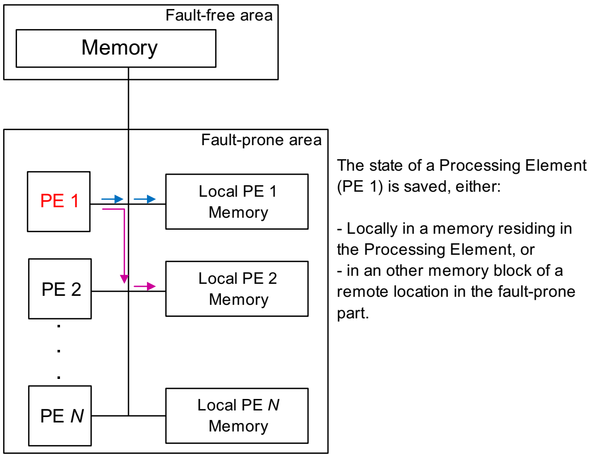

Fault-free area
Memory
Fault-prone area
PE 1
PE 2
PE N
Local PE 1 Memory
Local PE 2 Memory
Local PE N Memory
The state of a Processing Element (PE 1) is saved, either:
- Locally in a memory residing in the Processing Element, or
- in an other memory block of a remote location in the fault-prone part.

Fig. 5. State management in a DeSyRe system.

On the left portion of Fig. 6 we plot the execution times of the cascaded matrix multiplications micro-benchmark execution times on a single-processor (without runtime support) baseline system, 1-worker, and 7-worker MPSoCs. In addition, we consider three fault error rates 4%, 20% and 41% of the tasks being faulty. Moreover, in the second case, we assume that WMB4 becomes permanently damaged after 5 task executions; hence the runtime detects it and immediately migrates its remaining tasks to other active workers. With a 5 × 5 matrices size, the 7-worker MPSoC performance is low compared to the baseline due to the tasks scheduling overhead. However, it outperforms the baseline one when the array sizes become 15 × 15 or larger and scales to almost 3.95× compared to the baseline when the matrix sizes become 35 × 35.

We also measured the energy consumption for the same runs. According to the Xilinx XPower utility, the baseline and 7-worker systems require approximately 4.1 W and 4.45 W of applied power. Fig. 6 (right part) shows the MPSoC energy consumption for all considered cases. Compared to the baseline and for matrices size is 5 × 5, the task scheduling overhead increases the overall execution time, hence also the consumed energy, which is 16× higher. The same applies also for all fault rates considered, due to the transient and permanent faults occurrence. However, when the task workload increases, the overall execution time is significantly reduced compared to the baseline one, hence consumption becomes eventually as low as 27% of the baseline energy when there are no faults. Finally, under the largest task workload (35 × 35 matrix size) energy consumption still remains less than the baseline one for both 4% and 20% fault rates, while under a 41% error rate energy consumption is 15% higher compared to the baseline one.

3.3. Virtualization, context switching and task migration support

The selected task-based execution model and state management functionality simplifies the task of context switching, task

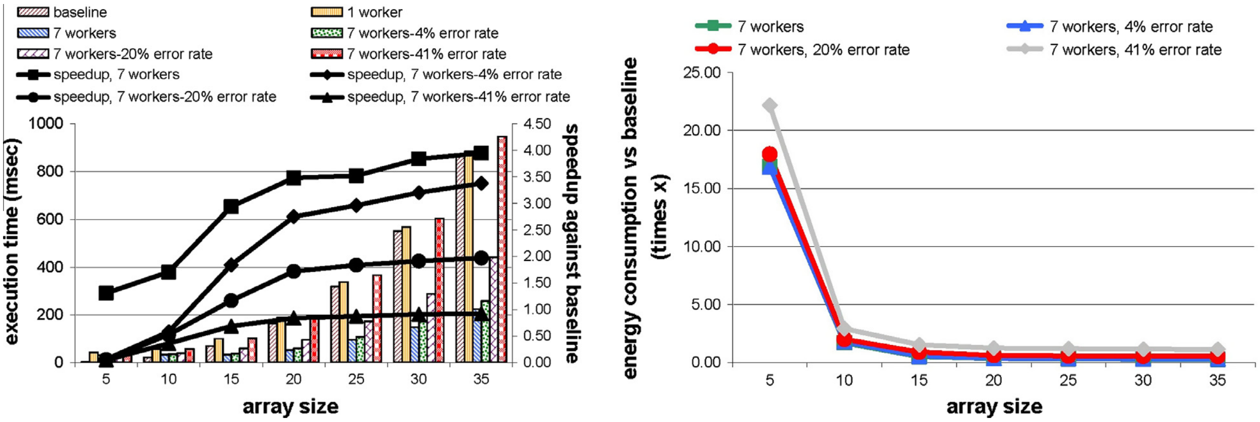


Fig. 6. Task-re-execution in the presense of faults: The matrix tree-multiplications micro-benchmark execution time and speedup (left), and energy consumption (right) against the baseline system.

migration and as a consequence virtualization support of a DeSyRe system. The granularity of atomic operation in DeSyRe is established at the task input/output boundaries and there is no need to save the internal state of the component that runs the task. As with the state management, DeSyRe tasks will execute until completion before the context switching, task re-execution, or task migration will be performed. Under this scenario, only the input (and in some cases the output) data of a task need to be saved and when required restored. While task migration between homogeneous processing elements is fairly straightforward, the DeSyRe Runtime System will also support migration to different types of processing elements. To support task migration in heterogeneous resources, the DeSyRe Runtime System has to be provided with the corresponding task versions (native binaries) for multiple types of resourses available in the system. In addition, we assume that all heterogeneous components support all the necessary native data types so there is no need for data conversions. Also, as the state of tasks is saved only on task boundaries, and at the level of the input variables, differences in the internal state of the various components (i.e. # of registers, and architectural state in general) do not affect task migration. The selected approach toward context switching and virtualization will obviously increase the pressure on the memory footprint of the application, and may be challenging in case of embedded, real-time systems. On the other hand, removing the need of saving the heterogeneous internal states of the components will remove completely the additional traffic, reduce the system complexity and provide a glueless support for task migration across components with different architectures. The advantages of the latter and the requirements of the DeSyRe applications validated our choices.

3.4. Graceful degradation

As already mentioned, the DeSyRe System on Chip is designed to be capable of managing the accumulation of faults in the system in a graceful manner – in other words to refrain from crashing and instead decide to sacrifice part of the system functionality and/or performance. In DeSyRe, Graceful Degradation can be achieved in three different ways:

- *Hardware Reconfiguration:* The Runtime can exploit the coarse- and fine-grain reconfigurability capabilities of the components, in order to tailor the set of working components to the application needs. Focusing on the coarse-grain, partially defective components will be used for tasks that they are still able to perform, while the fine-grain reconfigurable substrate will be utilized to deal with shortages of specific functional units (see also Section 6, Fig. 13).
- *Workload Adaptation:* The Runtime has the authority to adapt the workload of the system by dropping (low-priority) tasks or replacing tasks with other that have different processing requirements (possibly less computationally intensive and less efficient/accurate). One approach to this is to switch between different predefined modes of the application (i.e., normal mode, no extra features mode, emergency mode).
- *Task (re-)mapping:* Given the dynamic nature of both the available system resources and the software workload, the binding of tasks to resources will also have to be modified regularly. This can be also utilized to facilitate Graceful Degradation: If the number of available cores is reduced, the Runtime can either queue more tasks on the remaining cores and expect them to be carried out slower (performance degradation) or drop the least important tasks so that the rest are performed without performance loss (functional degradation).

3.5. Online Testing

Permanent faults, due to design, manufacturing or aging effects, will be detected in DeSyRe through online testing, scheduled centrally by the runtime system manager (see Fig. 7). To facilitate such testing, DeSyRe components will be designed with testing support architectures suitable for that particular component. These support architectures will include interfaces between runtime system manager and the local online testing support alongside the actual testing mechanism. In the following, online testing support architectures for Xentium DSP component, which is a crucial component in DeSyRe framework, is discussed in greater detail.

To facilitate online testing, Xentium incorporates wrapper design with online testing support. The testing support architecture contains interface with the runtime system manager via context switching mechanism (see Fig. 4). Since online testing will require exclusive access to Xentium's datapath and memory, the runtime system manager will enable context switching through local check-pointing at the component level. Moreover, the testing support architecture within the Xentium wrapper will contain test vector storage or generation mechanisms for software implemented built-in self test (SW BIST) and test vector analyzers accessed through the test access mechanism (TAM) as shown in Fig. 7. As can be seen, the online testing support architecture, which includes the test access mechanism and test vector

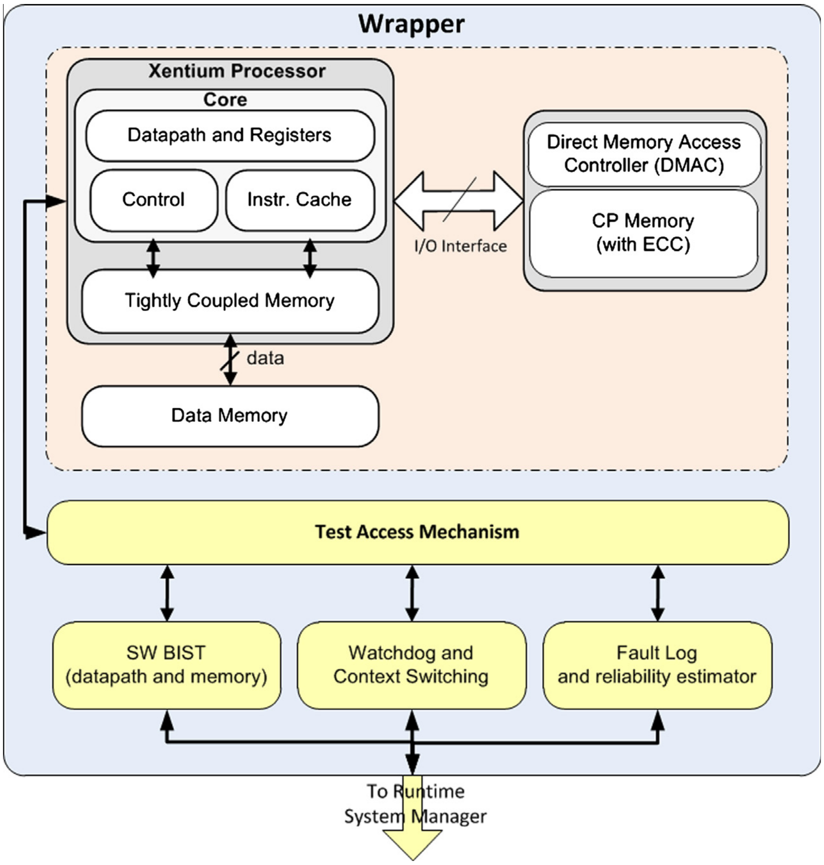


Fig. 7. DeSyRe online testing support in component wrapper.

storage/generation mechanism, is designed around the original Xentium DSP component.

Fig. 8 shows the detailed testing support architecture for Xentium DSP. The testing support architecture includes hardware/software based co-testing mechanism, co-ordinated by a central test access mechanism (TAM). The processor datapath is tested through a number of different test input vectors interfaced through the TAM. The responses from the processor datapath is then analyzed by the test output analyzer for detecting permanent faults. To test the memory devices built-in self-test (BIST) controller is incorporated. The BIST controller generates a number of memory access instructions in random locations and feeds these instructions through the TAM. The output of these instructions is then analyzed to detect the presence of one or more permanent faults. Using such BIST controller reduces the number of test vectors needed to be stored for effective fault detection coverage for such devices. The test vectors storage and the BIST controllers (Fig. 8) are designed with self-testing capability to ensure fault-free testing mechanism. More detailed description of the Xentium online testing mechanism is described in [26].

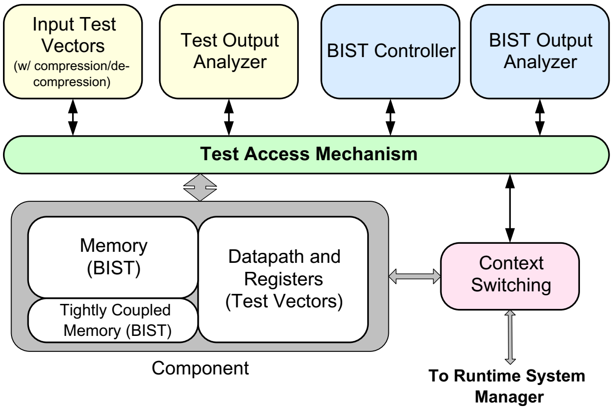

Input Test Vectors (w/ compression/de-compression)
Test Output Analyzer
BIST Controller
BIST Output Analyzer
Test Access Mechanism
Memory (BIST)
Tightly Coupled Memory (BIST)
Datapath and Registers (Test Vectors)
Component
Context Switching
To Runtime System Manager

Fig. 8. DeSyRe online testing support in component wrapper.

Apart from Xentium DSP a number of different other components are also being designed with appropriate testing support architectures. For example, error detection and correction coding is being incorporated in the network-on-chip (NoC) components with dynamic testing scheduling, RISC (SiMS) processors are being designed with instruction retry architecture [19], coarse- and fine-grained reconfigurable logics are being designed with appropriate redundancy logics, etc. To achieve the best possible permanent fault detection coverage, trade-off between the coverage and complexity of testing is currently being extensively studied.

4. DeSyRe Mechanisms for Fault Tolerance

The proposed framework is expected to cope with permanent, transient and intermittent faults using the above described logical layers and physical parts. Table 1 summarizes the DeSyRe mechanisms to detect and correct various fault types.

Permanent faults (defects), due to design, manufacturing or aging effects, are detected and diagnosed in DeSyRe through online testing scheduled centrally by the runtime system or distributed by self-checking mechanisms at each component. Hardware reconfiguration of the defective part is the first choice for correction; this involves isolation, and replacement of the defective part, performed by Middleware with possible high-level decisions made at the runtime system layer. When reconfiguration is not possible the tasks scheduled in the defective parts of the system are re-scheduled in other available components. To economize system

Table 1
DeSyRe techniques for tolerating various types of faults.

Fault Type	DeSyRe Detection mechanisms	DeSyRe Correction mechanisms
Permanent	• Online testing • Self-checking components	• Coarse- or fine-grain Reconfiguration (via Middleware, and runtime optimizations) • Task migration (possibly using alternative task descriptions) • Component sharing • Graceful degradation (through the above mechanisms, and with the assistance of runtime system optimizations)
Transient	• Software error detection codes (used together with adaptive checkpoints) • Self-checking components	• Adaptive check-pointing (i.e. rollback and recover). – Application-aware – Adapted to fault density • Self-correcting mechanisms at the components (i.e. ECCs)
Intermittent	• Same as transient, with some additional software mechanism to distinguish them from the transient faults	• Task migration (possibly using alternative task descriptions) • System Reconfiguration and Runtime optimizations • Adaptive check-pointing (i.e. rollback and recover)

resources, component sharing may be possible at the cost of performance degradation. The above will be achieved based on the graceful degradation policies of the system.

Transient faults (soft-errors) are detected in DeSyRe either centrally at the software-level using error detection (and correction) codes or locally at the components using checkers. Check-pointing mechanisms will be used to rollback to a known good state and recover from faults. Check-pointing will be adaptive to the application requirements and the fault density (i.e. adaptive frequency and placement of checkpoints) for energy efficiency. At the component level self-correcting mechanisms may be able to correct a sub-set of faults.

In DeSyRe, intermittent faults (i.e. periodic transient faults) are distinguished from transient faults in order to treat these differently. To do so, we add to the transient-fault detection mechanisms extra functionality to determine whether a fault is repeated. Then, task migration to other available components or hardware reconfigurations can be used for correction, while adaptive check-pointing and context switching mechanisms will be required to rollback and recover or to migrate a task.

Fault tolerance is a property that can be implemented in different ways. Redundancy (at different levels) is certainly the most popular way to achieve fault tolerance – however it is not the only way. For example, fault forecasting techniques (like memory scrubbing) inspect component sub-parts to find faults before they become failures; in case they detect a fault, they can repair (or reconfigure) the component sub-part and then avoid the failure. Fault repair is also a very common approach to achieve fault tolerance: in that sense, a CPU reset after a transient fault is a very straightforward way to achieve tolerance to transient faults.

In DeSyRe, the basic concept is to “instrument” or “wrap” the unreliable components and sub-components by local fault supervisors (FS) to become self-aware of their faults and be able to dispatch this information to the fault-free part.

We discuss next some specific fault-detection and correction mechanisms used in the DeSyRe applications SoCs.

4.1. Software Implemented Fault Tolerance

Central Software implemented fault tolerance in DeSyRe will be performed using the application check-pointing technique. The basic aim of this technique is to save the system state when no fault is detected or to roll back when one or more faults are detected. In this way, applications can overcome the impact of transient faults effectively. However, since this technique adds regular check-pointing or rollback intervals with the original execution times, performance overhead is caused. To incorporate a low-cost application check-pointing mechanism, DeSyRe will dynamically adapt the check-pointing frequency depending on the application reliability requirements during runtime. Such check-pointing mechanism will be implemented in DeSyRe using inter-component communication as shown in Fig. 4. Upon detection of one or more faults, the HW fault detection unit will communicate the approximate number and frequency of these faults to the Fault Analyzer unit within the software implemented fault tolerance manager. Using this information, the Fault Analyzer will then estimate and communicate the component reliability to the runtime system manager to adapt the check-pointing frequency accordingly.

To date significant progress has been made towards implementing software based fault detection and tolerance in DeSyRe. The fault detection and tolerance are both carried out through a software modification technique, followed by its evaluation in terms of performance and memory footprint costs. Fig. 9(a) shows the system architecture based on the Xentium DSP processor [7] enabling fault detection through software modification and correction through application check-pointing, while Fig. 9 (b) shows the software modification based technique employed on this system architecture. As can be seen, to enable application check-pointing technique, Xentium incorporates a fault tolerant memory (sufficiently error correction coded) interfaced via the memory access (DMA) unit (Fig. 9(a)).

With the given system architecture (Fig. 9(a), fault detection and tolerance is carried out using Software Modification Aided transient eRror detection Technique (SMART) in three steps. In the first step, high-level source-to-source conversion is carried out to replace the original data types to error detection enabler types. Such conversion essentially generates duplicate copies of the data and their computations to compare and detect the presence of transient faults. This is then followed by a low-level (i.e. assembly-level) software modification step to incorporate dynamic signature comparisons to detect transient faults in the software control flow during branch instructions. In this step, each branch label is preceded by inserting extra instructions to save the function of label address (s_b) in a temporary register. Then, further instructions are also inserted after each label to save the function of current label address (s_a) in another register. The previous and the new register values (i.e. s_b and s_a) are then compared to find any possible differences and thereby presence of one or more faults. Upon fault detection, rollback routines are initiated to avoid illegal control flow.

To control the potential performance overhead due to duplication of data and their computations, SMART incorporates two different inputs generated by the application profiler (Fig. 9(b)): data usage profile and normalized usage threshold (U_{NT}). The data usage profile gives the structured organization of the various variables and constants used in an application, together with their usage statistics. Using such data usage statistics, the tool only duplicates the original data types and operations of chosen

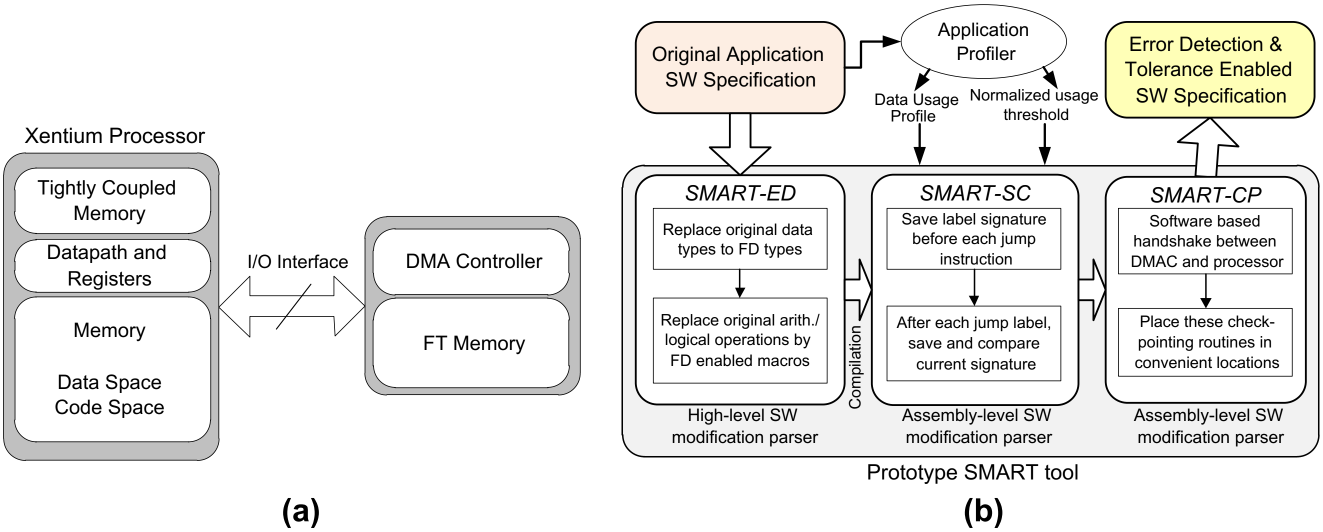

Xentium Processor
Tightly Coupled Memory
Datapath and Registers
Memory
Data Space
Code Space
I/O Interface
DMA Controller
FT Memory
(a)
Original Application SW Specification
Application Profiler
Data Usage Profile
Normalized usage threshold
Error Detection & Tolerance Enabled SW Specification
SMART-ED
Replace original data types to FD types
Replace original arith./ logical operations by FD enabled macros
High-level SW modification parser
Compilation
SMART-SC
Save label signature before each jump instruction
After each jump label, save and compare current signature
Assembly-level SW modification parser
SMART-CP
Software based handshake between DMAC and processor
Place these check-pointing routines in convenient locations
Assembly-level SW modification parser
Prototype SMART tool
(b)

Fig. 9. (a) DeSyRe system architecture enabling check-pointing mechanism in Xentium DSP, and (b) Software modification technique for transient fault detection and tolerance in Xentium DSP system.

variables or constants that have usage higher than normalized threshold (i.e. $U_N \geqslant U_{NT}$). Normalized threshold usage (U_{NT}) is calculated as a ratio of a variable's usage count to the maximum usage of any other variable. Details of how data/register usage is generated can be found in [25].

With the detected faults in first two steps (Fig. 9(b)), fault tolerance or correction is performed using software implemented application check-pointing technique enabled through further software modification as shown in Fig. 9. The technique employs a handshake based routine between DMA and processor units to initiate the check-pointing mechanism (since hardware interrupt or timing based supervision is not featured in the DSP processor).

To evaluate the effectiveness of the proposed fault detection and tolerance technique (Fig. 9(b)), a number of experiments are carried out in Xentium ISS environment [7] using different signal/image/data conditioning and processing benchmark applications to evaluate the effectiveness of the proposed SMART tool with system architecture (Fig. 9(a)). To demonstrate the effectiveness of the modified software description, initially full data duplication with $U_{NT} = 0$ is assumed. For each application, an approximate total of 5000 transient errors are injected in consecutive runs using single-event upset (SEU) based model (for fault injection details, see [26]). An arbitrary error probability of 10^{-12} (SEU per cycle) is assumed. Similar error injection has also been carried out in [15].

Table 2 shows the impact of SEU injections on these benchmark applications. The average execution cycles per application for a given test input sequence is shown in column 2, while the percentage of injected SEUs in different instructions are shown in columns 3–6 (determined after application profiling through output trace followed by error injection). Columns 7–8 compare the percentage of incorrect executions without and with SMART technique.

From Table 2 two major observations can be made. Firstly, it can be seen that depending on the nature of applications the percentage SEUs injected in different instructions can vary. However, the injected SEUs show a general trend of higher SEUs being injected in load/store (i.e. LDW/STW) and arithmetic/logical instructions (which is expected as DSP applications are highly computationally intensive in nature). For example, for memory and computationally intensive JPEG application the highest number (41%) of SEUs are injected in the load/store memory instructions, while about 34% SEUs are injected in the arithmetic/logical instructions. The branch instructions are subjected to the next highest percentage of SEUs injected, which varies depending on the nature of application. With the given SEUs injected, the second observation is that proposed SMART technique can significantly reduce the number of incorrect executions (by up to 80% in the case of FIR application) through software implemented error detection and tolerance technique (Fig. 9(b)). This is because, proposed technique can effectively detect the SEUs injected and mitigate the impact of these SEUs through application check-pointing, enabled through software modification (in SMART-CP). However, note that up to 15% of the cases can still lead to incorrect executions (in the case of WHT application) using SMART technique. This underlines the limitations of the proposed software-only technique.

To demonstrate the effectiveness of error detection through SMART technique, Fig. 10 shows the effective percentage of masked (i.e. errors with no effect), detected (both by SMART-ED and SMART-SC) and undetected SEUs through SMART technique in the benchmark applications with $U_{NT} = 0$ (i.e. full duplication of all data). From Fig. 10, two observations can be made. First observation is that maximum number of injected SEUs (up to 74% for WHT application) are detected through the proposed technique, while 17% SEUs can get masked, summing to an effective total of 91% errors being either detected or masked. Out of the detected SEUs, up to 85% are detected through the SMART-ED, while the rest of the SEUs are detected by SMART-SC. This is expected as SEUs injected in load/store and arithmetic/logical instruction dominate (Table 2). Note that up to 17% of the SEUs cannot be detected depending on where SEUs are injected.

The inclusion of data duplication, dynamic signature generation and check-pointing through software modification imply performance overheads. This is because SMART-ED generates controlled repetitions of computations through normalized usage threshold, U_{NT}, while SMART-SC requires signature generation and comparison cycles as well (Fig. 9(b)). Furthermore, SMART-CP requires processor to software-implemented check-pointing, which incurs further overheads.

To explore the performance overheads incurred due by SMART generated software modification, Fig. 11(a) shows the normalized performance overheads of different benchmark applications with varied duplication control through normalized usage threshold (U_{NT}). An approximate check-pointing interval of 10^6 cycles is chosen arbitrarily for the applications in comprehensive error injection environment. As can be seen SMART technique can result in

Table 2
Benchmark applications with injected SEUs in different instructions.

Application	Avg. Cycles/run	% SEUs in STW/LDW	% SEUs in Branch	% SEUs in Arith/Logical	% SEUs in Others	% Incorrect execution (w/o SMART)	% Incorrect execution (w/ SMART)
FIR	74,953,217	34.7	14.3	41.4	9.6	66	13
DWT	98,353,826	33.4	16.4	42.1	8.1	61	14
FFT	103,505,361	35.3	15.6	39.7	9.4	63	11
WHT	97,526,718	39.3	13.6	37.5	9.6	60	15
Viterbi	89,962,118	38.9	17.3	35.2	8.6	63	13
JPEG	311,822,971	41.3	15.9	34.5	8.3	59	12

a reasonable performance overhead for incorporating SEU detection and tolerance depending on the application and its nature of computation. For example, up to 83% performance overhead is incurred by Viterbi application (when all variables and data are duplicated, i.e. $U_{NT} = 0$). Note that although the proposed technique employs duplicate storage and computations, the performance overhead is contained. This is because the post modification compilation steps can employ instruction-level parallelism (a common feature in high-performance DSP processors) to reduce the performance overheads. Comparing the contributions of different SMART steps (Fig. 9(b)), it can be seen that the highest performance overhead (about 65% for the WHT decoding application) is caused by SMART-ED. SMART-CP is the second largest contributor for the performance overheads (with up to 13% overhead for the JPEG application). This can, however, vary depending on the approximate check-pointing interval assumed. SMART-SC contributes the lowest performance overheads for the given check-pointing interval when compared with the other steps. As expected, with higher normalized usage threshold, the duplication in the original application (through SMART-ED) can be controlled and the performance overheads show a trade-off. For example, when the duplication is only carried out for variables/constants that have more than 50% normalized usage (i.e. $U_N \geqslant U_{NT} = 0.5$), the performance overhead reduced to as low as 56% for DWT application. Similar trend can also be observed from other applications. However, this reduction of duplication comes with a decrease in the detection coverage as well. For example, in the case of FIR application, the detection coverage (detected and masked) can reduce to 64% when $U_{NT} = 0.5$ (compared to 91% when $U_{NT} = 0$).

Due to software modification, software descriptions can also expand with increased memory footprint. Fig. 11(b) shows the comparative memory footprints of different applications, showing contributions from different SMART steps. Since SMART-SC does not use memory resources, it is not shown. As expected, SMART-ED gives the highest memory overheads due to duplicated data components and related comparison instructions for detection of SEUs. Comparing the different applications, it can be seen that a reasonable 113% memory footprint overhead is caused by SMART technique in Viterbi and WHT applications, for example. This is because these applications use comparatively larger amount of data storage components (variables and data value holders) at high-level description, which is further duplicated by the proposed technique (SMART-ED). Further, it can also be seen that with controlled duplication through SMART-ED using normalized usage threshold, memory footprint can be reduced. This explains the trade-off between injected software redundancy overheads in terms of memory footprint and achievable error detection coverage, as expected.

Software implemented fault tolerance (SW FT) achieved through the above technique, however, has limitations of not being able to tolerate all possible faults, including permanent faults. Hence, SW FT will use various fault syndromes, such as double rollback, double watchdog timeout events and acceptance tests, to generate permanent fault syndromes. Upon detection of such syndromes, the fault analyzer module (Fig. 4) will request the runtime system manager to schedule appropriate online testing. More details of online testing can be found in Sections 4 and 5.

4.2. Transient Fault Tolerance for the SiMS RISC Processor

Fault tolerance in the SiMS processor is facilitated through duplicating the instructions on-demand using a single-event upset based fault model [19].Whenever a sequence of instructions is decided (on-demand) to be protected via instruction duplication, a dedicated register is set using a custom instruction. When decoded instructions are protected with the above mechanism, a hardware instruction-duplication unit duplicates this instruction in the next cycle. The result of the original instruction is stored in a dedicated register and, if the result of the "original" and the "duplicated" instruction are the same, the result is committed. If the results differ, then a fault is detected and the pipeline is flushed. The error is corrected by re-fetching the "original" instructions and resuming execution from the point of failure. An example is given in Fig. 12, where all control-flow instructions are made fault tolerant and, therefore, instruction A is duplicated.

The SiMS fault tolerance and its corresponding area, power, performance and energy overheads have been evaluated using a custom fault injector and a low-power encryption cipher, suitable for ultra-low-power systems, as benchmark. A number of duplication policies, duplicating *none*, *control-flow* instructions (jumps, branches), *all control-flow related* instructions (includes code-segments for e.g. loop-counter evaluation) or *all* instructions, has been devised to support various fault-tolerance requirements. In short, the previous list of policies offers increasingly higher fault tolerance. Fig. 13 depicts the percentage of injected soft faults manifesting as (control-flow) soft-errors for each of the duplication policies. It is shown that, even when no fault tolerance is added, only a small percentage (14.7%) manifests as a soft error, as the majority of faults are injected in idle parts of the processor. As expected, a duplication policy offering higher fault tolerance increases the number of corrected transient faults and, in case all instructions are duplicated, full fault tolerance to single-cycle transient faults is guaranteed.

Besides, by injecting duplicated instructions in a consequent cycle after the original, the switching activity of the core is lowered (due to less bit toggles), thus minimizing the power-consumption overhead. Furthermore, by allowing duplicate instructions to be executed instead of scheduled NOPs, the increase in execution time is minimal. The technique comes at an overhead of area (+23%) and, when offering the highest degree of soft-fault tolerance, power consumption (+17.5%), execution time (+42%) and energy consumption (+77%).

4.3. Dual Core Lock-Step (DCLS) Architecture based Fault Detection

One of the standard architectures used in the reliability and functional safety field is the Dual Core Lock-Step, i.e. two symmetrical processing units are contained on one die. The processing units run duplicate operations in lockstep (or delayed by a fixed period) and the results are compared. Any mismatch results in an error condition and usually a reset condition. However, for the

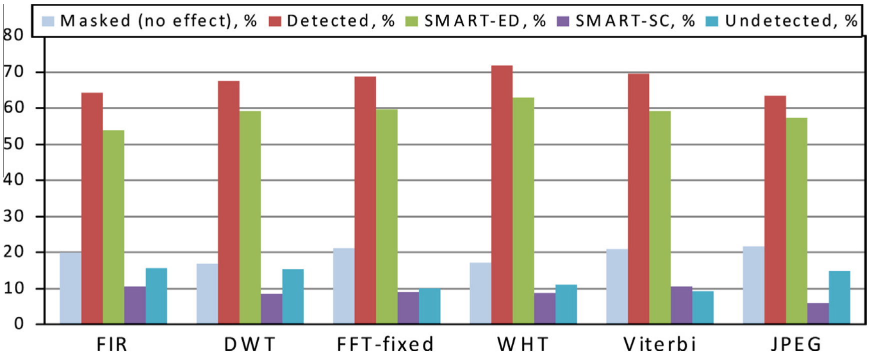


Fig. 10. SMART detection efficiency.

objective of DeSyRe system (i.e. going beyond fault detection), the standard implementations of DCLS are not able to identify which one of the two CPUs is faulty; consequently, even if the fault is in the slave CPU (so it is not a real critical scenario) the system is typically put in safe state and the availability of the system is completely lost. On the contrary, the “smart” DCLS architecture proposed in DeSyRe is an advanced solution to detect and identify failures, which replaces the standard comparator used in the traditional scheme and is able to determine which of the two cores is the faulty one, exploiting micro-architectural information extracted from the cores. As illustrated in 14, this new Advanced Comparator is an IP sitting in between the two processors (master and slave) taking control of the bus. It embeds cycle-by-cycle comparators, assuring the same basic functionality described for the standard DCLS, and it removes DCLS limitations adding features basically aimed to:

- detect which of the two cores is failing, if possible preserving the availability of the system (Identification Coverage);
- provide failure information to allow failure control techniques;
- swap the CPU master with the CPU slave when the default master is damaged.

The “smart” DCLS has been implemented for ARM Cortex-R4 and Cortex-R5 CPUs but it can support all different types of CPU including the Xentium. The implementation addresses not only the inner portion of the CPU core but also outer structures like caches (see Figs. 14–16).

Efficiency in the detection, identification and repair of faults has been measured by means of a fault injection campaign on both permanent and transient faults. Faults coverage results of the “smart” DCLS architecture are summarized in the following table (see Table 3).

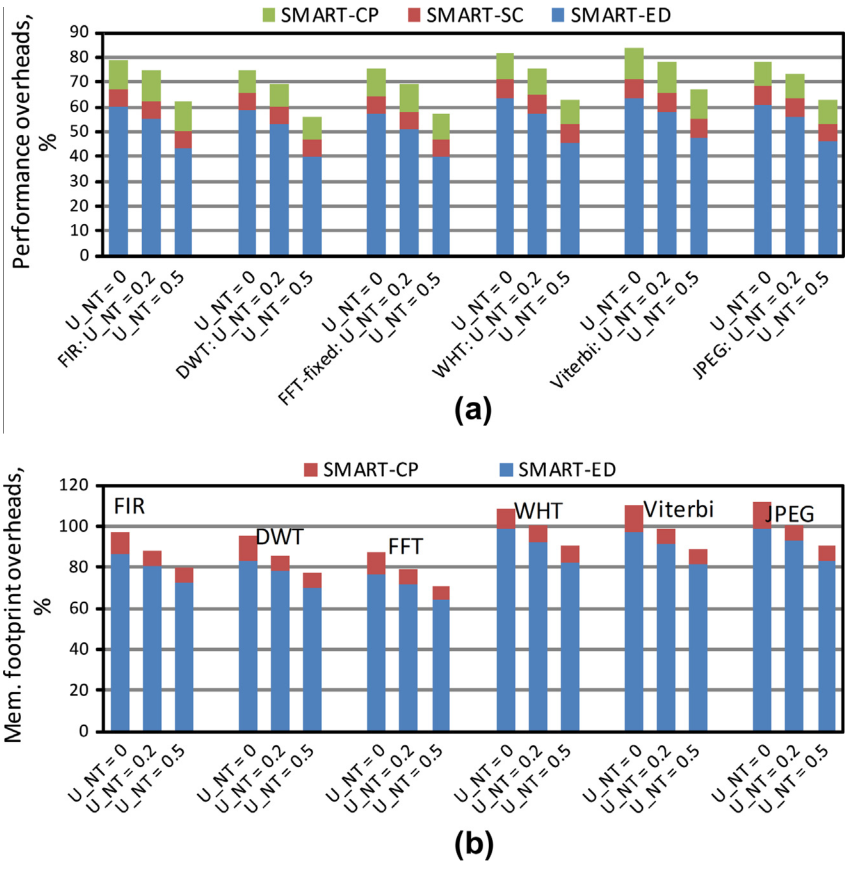


Fig. 11. (a) SMART performance overheads, and (b) SMART memory footprint overheads.

Another important characteristic of the "smart" DCLS is the ability to quickly detect, isolate and identify the fault that may affect one of the two CPU cores, so guaranteeing the faster reaction and reconfiguration. In general, as shown in the next figure, each of the operation steps of "smart" DCLS can be associated to a time interval:

- DTI = Detection Time Interval
- ISTI = Isolation Time Interval
- IDTI = Identification Time Interval
- RTI = Restoration Time Interval

Performance results achieved by "smart" DCLS are shown in the table (see Table 4).

The total overhead in terms of gate counts of "smart" DCLS (including also 4 instances of the "CCF detector" described in paragraph E) is in the range of 150–200 Kgates depending on the specific CPU.

4.4. Component Wrapper

Typically, in a safety–critical system, the alarm information shall be communicated to the System Supervisor (the fault-free part of the DeSyRe system) using a dedicated Hardware fault detection wrapper and a communication channel implemented using separate safe interconnections and transporting diagnostic information. This way the diagnostic and mission channels are kept separate, resulting in a more safe and robust system according to norm recommendations. This dedicated bus connects all the detectors implied in system. Moreover, having a dedicated communication channel to convey diagnostic information results in a faster transfer of critical alarms toward the system supervisor.

The HW fault detection wrapper provides to the fault-free part the following information:

- Alarm_Error. These alarms are related to the dangerous faults
- Alarm_Warning. These alarms are related to event near dangerous fault (e.g. Common Cause Failure detector provides warning signals which indicate that the silicon behavior got closer to violating constraints)
- Alarm_Info. These signals provide information about the coherency of the results in order to identify problems inside the detector.

4.5. Common Cause Failure (CCF) Detection

Detection of dependent failures and especially common-cause failures is crucial to allow identification and proper repair. In fact, *common-cause failures* are one of the most important failure modes when diagnostic mechanisms are implemented in the same silicon device. There are different types of common cause failures, like temperature variation and noise into power supply and the clock. It is well known that high temperature operation compromises long-term reliability and impacts circuit performance such as the timing characteristics of the circuit, causing timing violations. Similar timing effects can be obtained in case of noise affecting power supply and clock network. In order to avoid the standard measures to detect common-cause failure, the solution studied and implemented in DeSyRe consists in a CCF detector which can be instantiated according to the position of the expected redundancies. The CCF detector is a completely independent checker. It includes special structures used to understand when the circuit approaches timing-violating operating conditions. In order to make the CCF detector sensible to CCF problems, some internal paths are configurable during synthesis (e.g. by means of delay chain) to achieve the same critical path than the critical path in the IP under detection. The internal paths are redundant to distinguish real CCF problems from problems of the CCF detector itself. The CCF can be used in conjunction with Dual Core Lock-Step architectures (DCLS) explained in the previous section.

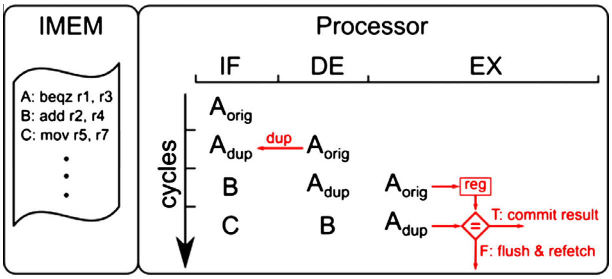


Fig. 12. Transient fault tolerance in RISC (SiMS) processor using instruction duplication and retry technique.

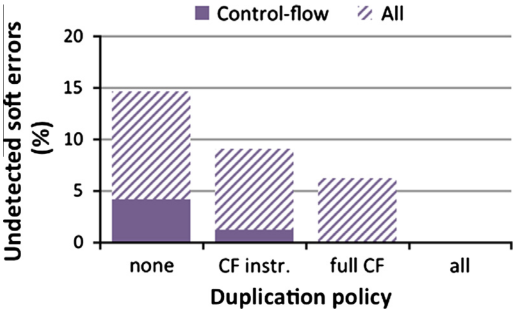


Fig. 13. Soft-error manifestation in the SiMS core.

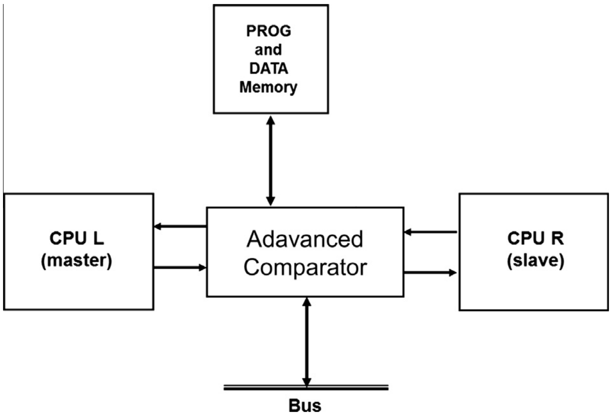


Fig. 14. The advanced comparator in the DCLS scheme.

5. DeSyRe Reconfigurable Substrate

The DeSyRe framework relies significantly on a flexible and dynamically reconfigurable hardware substrate to isolate, replace and (when possible) correct design and manufacturing defects as well as other permanent faults due to aging. In previous works, the design choice was either coarse- or fine-grain granularity of substitutable units; these are units that can be replaced when defective. In the first case, the substitutable unit can be an entire sub-component (e.g. a microprocessor's pipeline stage), while in the latter case an FPGA logic cell. There are tradeoffs between these two alternatives. Coarse-grain approaches are less defect-tolerant – fewer defects can have large impact to the system – but lead to solutions that are more power and silicon efficient. Fine-grain

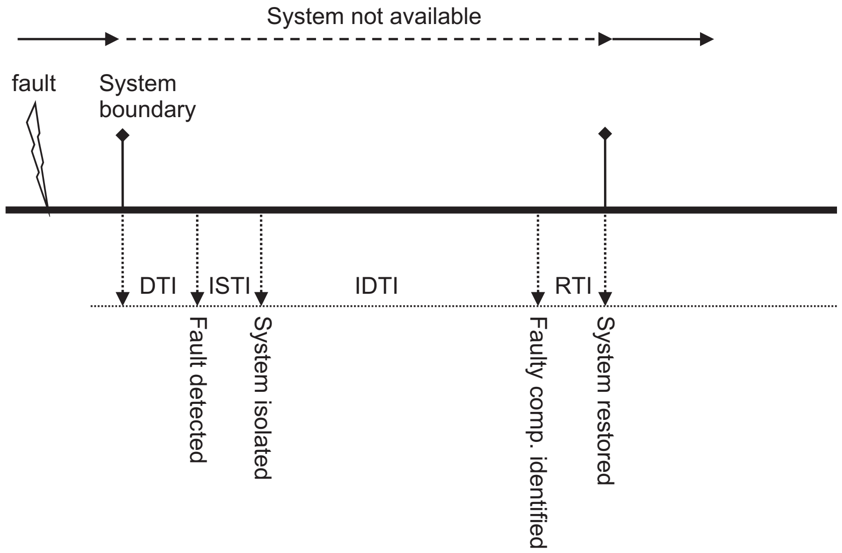


Fig. 15. The operation steps of "smart" DCLS (and related timing).

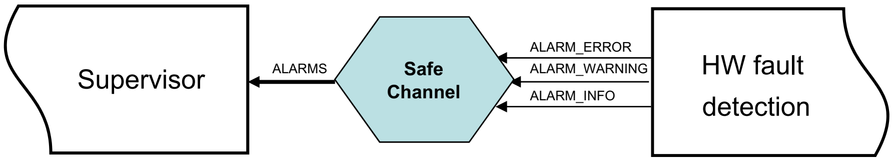


Fig. 16. Communication between HW fault detection wrapper of a component and the Supervisor at the fault-free part of the DeSyRe system.

Table 3
Coverage results of the "smart" DCLS.

Coverage	Value (%)	Note
Diagnostic coverage	99.9	Coverage is guaranteed also for faults that could affect the small portion of "smart" DCLS that could directly affect the mission function (e.g. the CPU switch)
Dependent failures detection	$\beta_{IC} = 23$	Achieved thanks to the Common-Cause Failure detector (see paragraph E). β_{IC} is an semi-quantitative method introduced by IEC 61508 standard to measure dependent failures coverage
Identification coverage by HW	60	
Identification coverage by HW + SW	90	Achieved by running a SW Test Library
Latent faults coverage	90	Latent faults coverage of "smart" DCLS stand-alone

Table 4
Performance results of the "smart" DCLS.

Step	Value	Note
DTI	~3 clock cycles	The fault is detected as soon as it appears at CPU output
ISTI	~20 clock cycles	
IDTI	~300 μs @ 160 MHz	Average IDTI when fault is identified by "smart" DCLS HW
	~2 ms @ 160 MHz	Average IDTI when fault is identified by "smart" DCLS SW
RTI	~150 clock cycles	This is the case in which a reset is not needed.
	Depends on the overall circuit architecture	This is the case in which a reset is needed. The reset time depends on the specific MCU

approaches can tolerate a larger number of defects, but utilizing an FPGA-like substrate introduces performance, power, and cost overheads.

One of the primary challenges in the DeSyRe project is to investigate the architecture of the underlying hardware substrate for the DeSyRe SoC. DeSyRe explores a granularity mix of fine- and coarse-grain underlying hardware in order to provide increased defect-tolerance without giving away significant parts of the system performance and power efficiency. Fig. 17 depicts such an example with two DeSyRe RISC components. In this example, each RISC processor is divided in smaller sub-components (implemented in fixed hardware), i.e. pipeline stages, functional units, etc., surrounded by reconfigurable interconnects/wires. In the absence of defects, the sub-components 'S' will form the RISC component. However, in case a sub-component is defective, it can be isolated using the reconfigurable interconnects, and subsequently be replaced either by an identical unused neighboring sub-component (S), or by a functionally equivalent instance (S′) implemented in fine-grain reconfigurable hardware.

5.1. Benefits and overheads of reconfigurability

In order to have a better understanding of aforementioned tradeoffs, an analytical study has been performed for an array of processors such as the ones depicted in Fig. 17. In the presence of permanent faults, we evaluated 5 different cases of processor arrays with fine-grain and/or coarse-grain reconfigurablity compared to no reconfigurability at all. In this study, it is assumed that a fixed silicon area is available which can accommodate a certain number of cores; as expected different choice or reconfigurability has

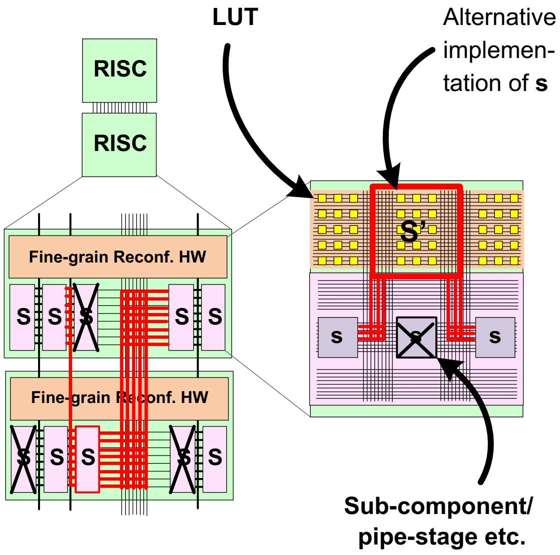


Fig. 17. The novel DeSyRe flexible/reconfigurable hardware substrate.

different area overheads and therefore different number of cores fit in the given area. The 5 cases of multicore arrays are as follows:

1. A baseline multiprocessor array design with no reconfiguration which fits 9 cores.
2. Coarse-grain reconfigurable array providing stage-level reconfigurability, which fits 6 cores.
3. Fine- and Coarse-Grain design, similar to the one depicted in Fig. 17, which fits 4 cores and a fine-grain substrate that can be used to configure 4 pipeline stages.
4. Restricted coarse-grain reconfigurable array which provides cores with interchangeable stages within two clusters of 4 and 3 cores, respectively. This constraint reduces the area overhead fitting 7 cores in the given area.
5. Similar to the previous creating clusters of pairs of cores, allows fitting 8 cores in the given area, although restricting the options of reconfiguration.

Considering a defect rate that varies between 10% and 90% probability to have a defect in a pipeline stage, we measured the probability of each one of the above cases to provide at least 3 working cores. Fig. 18 shows the result of this evaluation. For low defect rates the probability of delivering 3 working cores is almost 1 for all cases. However, as the defect rate increases, and above 40% chance to have a defect in a pipeline stage, the reconfigurable cases are more defect tolerant than the baseline. More specifically, the simple coarse-grain reconfigurability may offer up to 10 times better probability than the baseline, the clustered coarse-grain approach is the best guaranteeing at least 80% probability to have 3 working cores even in very high defect rates. Finally, the mix of fine and coarse grain reconfigurable substrate is somewhere in between the other reconfiguration options mainly due to its excessive area overhead which allows to fit only 4 cores in the given area.

For our experiments, we considered as a case study processor the SiMS RISC of the DeSyRe SoCs. For this example, the 40% defect rate corresponds roughly to 4 defects per million transistors. Modeling aging effects in 32 nm technology it is estimated that up to 2–6 transistors per million can be damaged in 5 years SoC lifetime only due to the NBTI (Negative Bias Temperature Instability). In this modeling, temperatures 90–130 °C were considered, and a 25–95% duty cycle, while a transistor is considered damaged when having less than 40% of its initial voltage threshold. In general, we can state that the area overhead of the considered reconfigurability varies between 10% to 2×, while the performance overheads are between 15% and 50% in the coarse grain case and about 5× in the fine-grain case.

5.2. Dynamic Reconfiguration

In order to utilize the reconfiguration capabilities mentioned above, the Middleware performs dynamic reconfiguration based on the decisions made by the System Optimizations module of the Runtime System. More specifically, the System Optimizations module decides the appropriate configuration of each component based on various objectives and system constraints (i.e. real-time constraints, power consumption requirements) and sends a request to the Reconfiguration Management module to configure a component of the system.

An example of coarse grain reconfiguration between two identical components is given below, referring to Fig. 13. Both components have faulty stages in the coarse grain part and cannot be utilized under the existing configuration. The System Optimizations module sends a request to the Middleware to configure both components by bypassing faulty parts in order to recover a component that ensembles correct functionality. In general, the steps for the hardware reconfiguration process are:

1. The System Optimizations module decides the appropriate configuration of the components, given the existing faults and system constraints.
2. The System Optimizations module requests from the Reconfiguration Management to perform configuration of the two components and transmits a high level description of the required component configuration.
3. The Reconfiguration Management module processes the above request, notifies the wrapper of the target components about the scheduled reconfiguration, and provides the required information (i.e. configuration stream) to the wrapper (and specifically to the Component Reconfiguration module – see Fig. 4).
4. The Component Reconfiguration modules of the target components are then responsible for:
   - Loading the new configuration stream into the components.
   - Checking the status of the new configuration.
5. The wrapper passes the status of the reconfiguration to the Reconfiguration Management module and the Reconfiguration Management module passes on the message about successful or not reconfiguration to the System Optimizations module.

6. Baseline Implementation Platform

The DeSyRe applications are implemented on a heterogeneous multi-core SoC template depicted in Fig. 19. The DeSyRe baseline SoCs consist of multiple heterogeneous processing cores, such as Xentium DSP cores [7], on-chip memory blocks, RISC processors, and custom accelerator blocks interconnected by a Network-on-Chip (NoC). Fault-tolerance extensions to existing NoC approaches [8,9] as implemented in [10,11], respectively, are considered in DeSyRe. The heterogeneous baseline SoC is implemented in the fault-prone part of the DeSyRe system. Moreover, part of the DeSyRe system is realized on a separate general-purpose processor, which is considered to be the fault-free part of the system.

In the next section, two (medical) applications are described that are implemented based on the DeSyRe SoC framework: The artificial-cerebellum SoC consists of a RISC processor, a Custom-IP and multiple Xentium processors to perform the computationally intensive tasks of modeling brain-cells. The artificial pancreas will be more lightweight; it includes a RISC processor for control, a

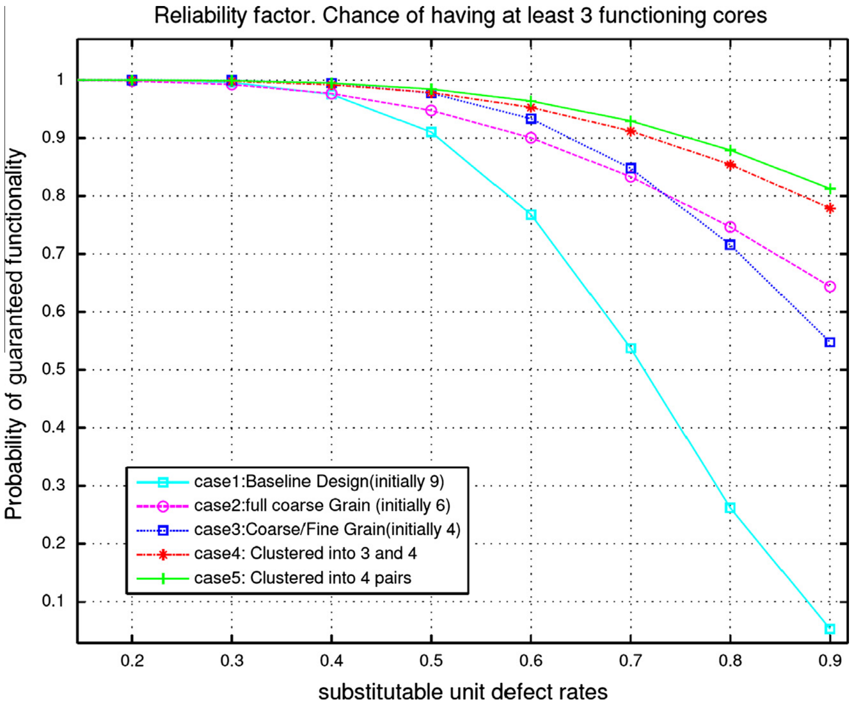


Fig. 18. Effect of introducing Coarse/fine grain reconfigurability: Performance comparison for five different scenarios of a Multiprocessor array.

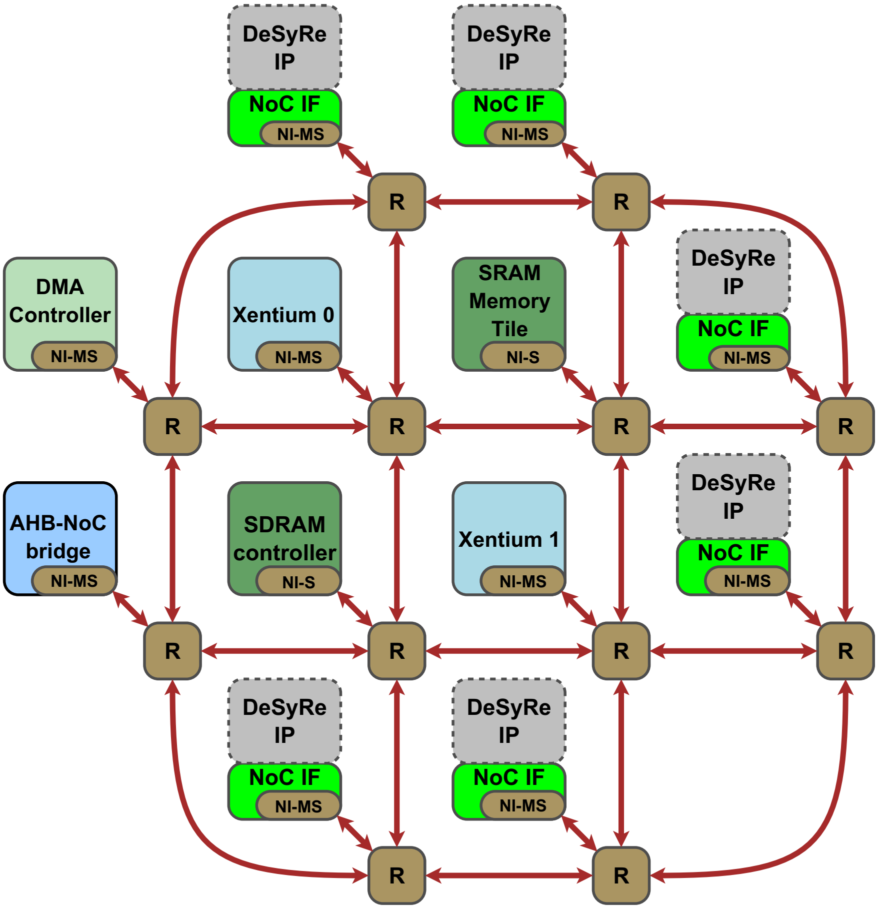


Fig. 19. DeSyRe SoC template with two Xentium DSPs, memories and custom (DeSyRe) IP blocks.

Custom-IP for calculating the insulin rates and a Xentium processor for encryption and data-logging.

In the remainder of this section we describe in more detail each component of the baseline implementation platform.

6.1. Reconfigurable fault-tolerant NoC

The main communication infrastructure in the DeSyRe SoC is provided by a NoC. All Xentium processing tiles, processor, memory resources and other accelerator cores are accessible via the NoC. Moreover, the NoC can be used to (re)configure cores in the SoC, and to access scan chains (at run-time for on-line testing) and memory Built-In Self-Test units of e.g. the Xentium tiles.

A key feature of NoCs is their predictable performance. Besides, well-designed NoCs allow disabling inactive parts of the network, which is essential for energy-efficiency and dependability. With respect to creating a reliable and fault-tolerant SoC architecture, the NoC is a crucial building block. Different levels of reliability can be identified in the SoC, which require large involvement of the NoC:

- On the *SoC level*, processor tile errors and memory tile errors can occur. Upon errors in the SoC, defective processor cores or memories in the SoC will be disabled and another (reconfigurable) processor core or memory part will be used to (re)execute a task; reconfiguring processor cores in the multi-core SoC or using different dedicated memory parts requires that data channels in the NoC are re-routed.
- On the *NoC level*, connection errors, channel errors, router errors or link errors can occur which are due to errors induced in the NoC elements.

On the system-level, the most important requirement of the NoC is that data can be re-routed in the multi-core SoC. Hence, the routing mechanisms should be based on dynamic routing schemes in the NoC routers. Upon detection of e.g. core errors, the system should be capable of re-routing channels from a source tile to a different destination tile in the multi-core SoC. Also, on the system-level, the individual building blocks of the SoC should provide means to detect that e.g. an error occurred in a processing tile of the multi-core SoC. After detection of an error on system-level, run-time software should take immediate action in order to reconfigure the multi-core SoC.

6.2. Xentium DSP

The Xentium core is a 32-bit fixed-point DSP core designed for high-performance embedded signal processing. The VLIW architecture of the Xentium features 10 parallel execution slots and includes support for Single Instruction Multiple Data (SIMD) and zero-overhead loops. High-performance and energy-efficiency are achieved by optimizing parallel operation at instruction level. The core modules of the Xentium tile are the Xentium core, the tightly coupled data memory and a NoC interface as shown in the block diagram in Fig. 20. A default instance of the Xentium DSP tile contains 16-kB tightly coupled data memory and 8 kB instruction cache. The size of the data and instruction memories is configurable at design-time of the multi-core SoC.

6.3. SiMS Processor

The SiMS core is a MIPS-like, 5-stage (Fetch, Decode, Execute, Memory, Writeback) processor, optimized for low-power operation. Low-power consumption is currently achieved by a minimalistic design (e.g. by omitting advanced architectural features such as branch predictor of forwarding unit) but ongoing efforts focus on architecture- and compiler-based enhancements. The processor has 16 32-bit registers, a 16 kB Instruction Memory (16-bit ISA, 24 instructions) and a 16kB Data Memory (32-bit data words). The SiMS core is integrated as one of the DeSyRe IP modules in the DeSyRe SoC, depicted in Fig. 19.

6.4. Custom-IP blocks

The Custom-IP blocks for the two applications each consist of one main computing unit and several smaller hardware blocks, used for data-buffering and communication to and from the NoC. The computing hardware blocks are implemented on a Virtex-6 LX 550-T with floating-point components from the FPLibrary [20], and are optimized for area. The custom-IP blocks are integrated in the DeSyRe SoC template in the DeSyRe IP place-holders.

The artificial-cerebellum compute block is a custom hardware implementation of the “soma” task (see Section 7). It is a dataflow circuit with complex floating-point arithmetic operations such as exponent, divisions and multiplications. It occupies 2577 FPGA slices and has a latency is 127 clock cycles.

The artificial-pancreas compute block is a custom hardware implementation of the insulin rate task (see Section 7). It is a state-machine controller with floating-point multiplications and additions. Depending on the input values its latency varies with a maximum of 42 clock cycles, while its area cost is 659 slices.

6.5. Memory tiles

The multi-core SoC is equipped with (on-chip) memory tiles, connected to the NoC. In Fig. 19 an SRAM memory tile as well as an SDRAM controller are integrated in the SoC template. Depending on the implemented application, the system designer could integrate on-chip or off-chip memory resources. It is key for scalable multi-core systems-on-chip to have sufficient distributed memory available in the SoC in order to avoid bottlenecks through single shared resources. In DeSyRe, the distributed memory in the SoC will adapt multiple functions, such as storage of DSP execution binary, elastic buffers, storage of checkpoints, storage of (intermediate) processing results, etc.

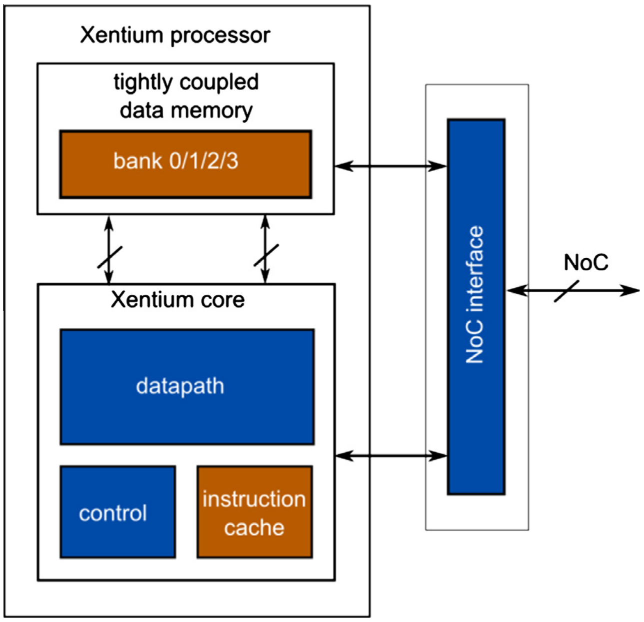


Fig. 20. Xentium DSP tile with NoC interface.

All communication in the DeSyRe SoC is memory-mapped over the NoC. In the course of the DeSyRe project, existing NoC approaches [8,9] will be considered and used to build a fault-tolerant NoC that satisfies the given requirements on tolerating SoC-level and NoC-level errors. Moreover, the DSP cores and memory tiles in the SoC will be extended with local checkers and (on-line) test wrappers to monitor the healthiness of the cores in the SoC at run-time. The ultimate goal is to have a reliable NoC that can be used for application data communication as well as for transferring (on-line) test information, such as test patterns, from test pattern generators to the test circuitry of a core, or to transfer (on-line) test results from the component-under-test (CuT) to a response analyzer. The test wrappers will be developed according to the IEEE1500 standard and are accessible by the NoC.

7. Applications

The DeSyRe framework will be applied to two medical SoCs, namely an artificial cerebellum and an artificial pancreas. Both applications are being developed by Neurasmus BV, one of the DeSyRe-project industrial partners. Currently, preliminary prototypes of the two applications have been implemented and validated on an FPGA board. In the second phase of the project, these two systems will be enhanced with the fault-tolerance features being developed under the DeSyRe project.

Although both DeSyRe medical applications exhibit high reliability requirements, the artificial-pancreas system is radically different from the artificial cerebellum, a difference which will help to demonstrate the diversity and flexibility of the DeSyRe framework. The artificial cerebellum system requires significant processing of complex mathematical operations in many concurrent tasks. On the contrary, the artificial pancreas requires – by comparison – few computations for processing and controlling sensors and actuators, and for interfacing (security, communication, compression) to a treating physician or the patient. It requires, however, more rigorous closed-loop control, and should be ultra-low-power and adaptive to new treatment descriptions.

7.1. Artificial Cerebellum

Neurasmus is developing an *artificial*, *real-time*, *cerebellar medical MPSoC for rescuing damaged parts of an actual*, *biological brain* suffering from various brain diseases stemming from loss of sensorimotor control, such as calcium-channelopathies, fragile-x and autism. The mid-term goal of Neurasmus is the development of a portable, artificial-cerebellum unit which can be carried to a home environment for patient use or to a lab environment for clinical experiments with the purpose of rescuing (i.e. replacing) parts of a failed or failing biological brain.

An artificial cerebellum effectively is a *closed-loop-control* system which will entail massive *processing tasks* as well as *real-time interfacing* to multiple recorded (input) and stimulated (output) neural structures. In its first generation, it is expected to be portable and has the following requirements:

- high reliability due to its medical application;
- high throughput (for replacing a large number of biological neurons);
- low latency (for “real-brain-time” processing);
- power efficiency for portability; and
- adaptability to different input patterns.

(1) The Inferior-Olive model

Within the DeSyRe project, Neurasmus has developed a prototype system that can accurately simulate one of the main structures of the olivocerebellar loop in the brain, i.e. the inferior olive. Neurons in the inferior olive are tightly coupled and produce synchronous and constant oscillations as well as controlled spikes that are believed to be essential for the fine-tuned motor-control ability of the cerebellar cortex. The inferior-olive model we have implemented has been originally developed by de Gruijl [22], one of our collaborators at the Netherlands Institute for Neuroscience. The model is an extension of the earlier model published by Schweighofer et al. [23] and is based on Hodgkin–Huxley differential equations [24]. These equations describe in detail the electrical activity that results from the combination of external changes in the electrical potential and the internal concentrations of the main chemical components involved in the transmission of neural-signals: calcium, potassium and sodium. The model divides the neuron into three anatomical compartments: dendrites, soma and axon. The soma is the cell body and the dendrites are extensions of the soma that receive the electro-chemical signals coming from other neurons. The axon is the neuron tail that transmits the electrical signals to other neurons. For every compartment, a few chemical channels are present in the model so as to contribute to the total compartment potential, as shown in Fig. 21. Furthermore, the computational model operates in a pipeline fashion to allow for the concurrent execution of the three compartments. In order to

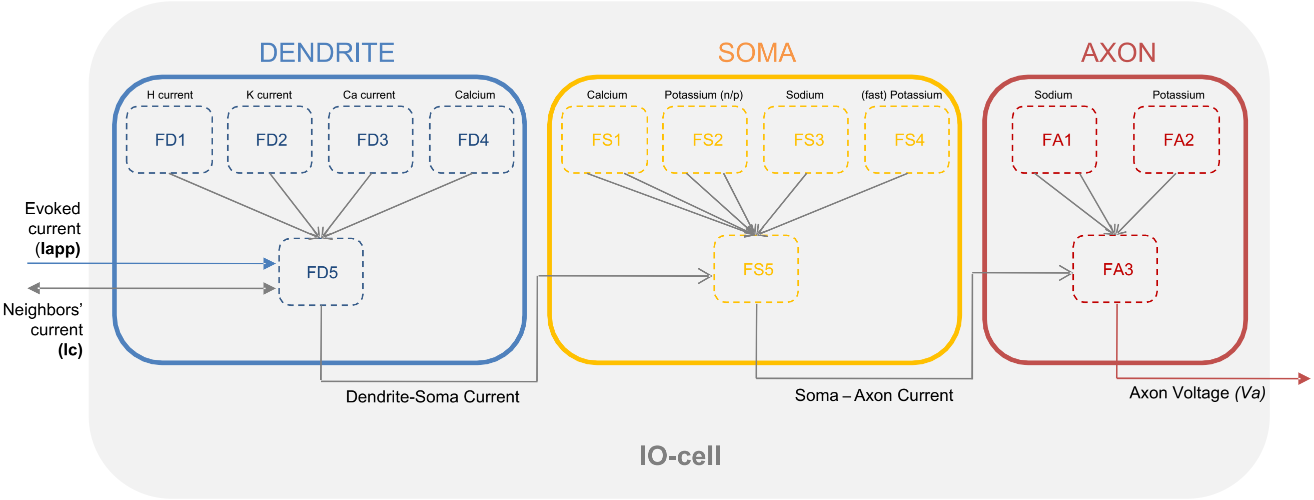


Fig. 21. Block diagram of the three-compartment model of a single inferior-olive neuron cell.

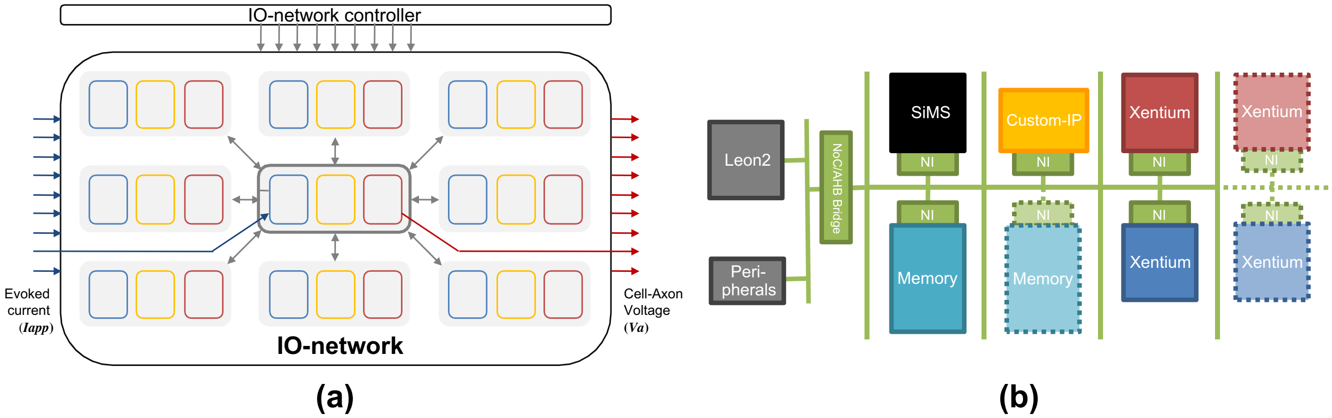


Fig. 22. Block diagram of (a) the IO-Network model and its mapping, and (b) onto the MPSoC architecture.

Table 5
Profiling of the three IO-cell compartments on the Xentium simulator.

	Dendrite	Soma	Axon
Execution cycles	35,860	64,421	33,008
% Of execution time	27%	48%	25%
(1) Task I/O data (bytes)	128/24	128/36	128/16

support real-time operation, all cells are required to compute one iteration of compartmental calculations within 50 µs.

(2) The IO-network model

Fig. 22 illustrates (a) the network-model architecture and (b) its MPSoC implementation. Every cell receives, through the dendritic compartment, the influence of its eight neighboring cells (thus modeling the biological gap-junctions). The dendrites in every cell also receive an externally evoked current *Iapp*. Conversely, the axon voltage *Va* of all cells is the system output. The system works in lock-step computing discrete output values that, when aggregated in time, contribute to form the electrical waveform response of the system. The IO-network controller module ensures the correct synchronization of the cell computations when multiple cells and compartments are being executed simultaneously.

(3) Application's mapping and implementation onto the MPSoC

The inferior-olive-model source code, initially available in Matlab, has been rewritten in C and then parallelized and ported to the DeSyRe baseline platform presented in Section 6. Fig. 22 shows the mapping of the application components onto the hardware cores. The controller block runs on the SiMS processor and is in charge of initializing the necessary data structures (especially the cell states) and, then, it runs the loop that guarantees the synchronization among the concurrent tasks. We define as task the execution of one instance of one of the cell compartments and, therefore, have three types of tasks: dendrite, soma and axon. Based on initial profiling results (see Table 5), we have identified the soma task as the most computationally intensive one. Therefore, the dendrite and axon tasks are mapped to Xentium processors, while the soma has been implemented in hardware as a custom IP block, as described in the previous section.

A shared-memory tile holds a large data structure that contains the state of all the cells in the network. The state of each cell is a collection of variables that represent, at every moment in time, the compartment's electrical potentials and the concentrations of the chemical elements modeled. In every time step, the controller issues three tasks per cell times the total number of modeled cells. Within DeSyRe, pre-recorded real neural-signal traces will be used as inputs for the evaluation of the proposed system.

The current prototype also includes an AHB subsystem with a Leon2 processor and a few peripherals for interfacing with a host PC. In the second stage of the project, the AHB subsystem can be used for implementing the fault-free part of the DeSyRe system. Furthermore, the dashed components in Fig. 22(a) indicate how the system is scaled by adding pairs of Xentiums and memory tiles if required. The abundant parallelism in the application, combined with the use of a NoC, allow the application performance to scale along. While the Custom-IP is designed to cope with the added performance pressure of tens of Xentium cores, the SiMS core is expected to saturate first. One solution we envision is the use of a clustered architecture with several SiMS cores. This architectural style is particularly interesting also from the biological point of view, since it matches the clustering of neurons that naturally happens in the brain.

7.2. Artificial pancreas

In some forms of diabetes, the pancreas cannot produce insulin sufficiently. The sheer number and increasing rates of such patients worldwide is a major incentive for developing a so-called "artificial pancreas" device. In essence, such a device is a closed-loop-control implant which samples the glucose levels in the blood stream and releases insulin as needed (see Fig. 23). Even though an actual, chronic artificial pancreas has not been developed yet, glucose-sensing implants have constantly increased in numbers and improved over the years [4,5].

Given the constantly increasing number of diabetic patients, Neurasmus envisions implementing the artificial pancreas implant as a DeSyRe-based, implantable system for automatically and accurately regulating blood glucose levels. In a fashion similar to the artificial-cerebellum application, this system is also subject to tight requirements of:

- high reliability due to its medical application;
- security (to avoid intruders);
- power efficiency for portability; and
- adaptability to different input patterns.

(1) Application's description

As shown in Fig. 23(a), the current system contains a module for processing sensory data and calculating the insulin dose, and a module for data logging that includes compression, encryption and checksuming. In developing this application, we adopt the

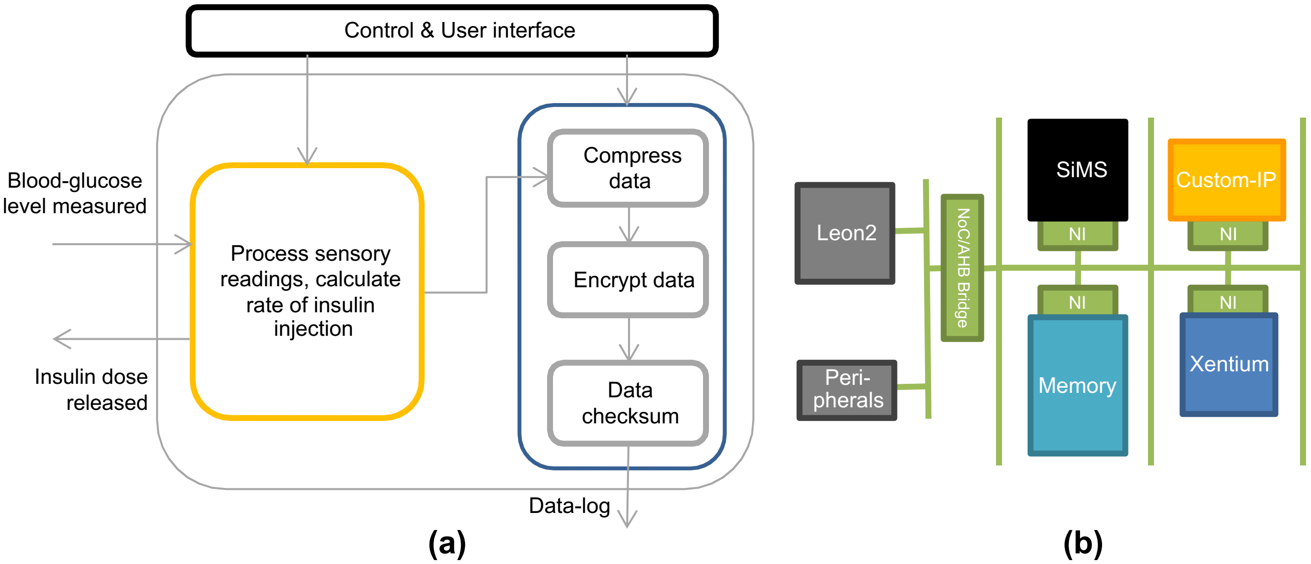


Fig. 23. Block diagram of (a) the artificial-pancreas application, and (b) its mapping onto the MPSoC architecture.

latest achievements in artificial-pancreas research [21,12,13], while improving on the state of the art by providing highly defect-tolerant devices, suitable for chronic implantation. The encrypted data logging feature is a further innovation in our system.

(1) Application's mapping

Fig. 23(b) illustrates the mapping of the artificial-pancreas application onto the MPSoC platform. The user interface and the system control runs on the SiMS processor. The control algorithm that decides on the insulin dose based on the glucose readings is implemented in custom hardware. Finally, the data-logging block is serviced by a Xentium processor.

8. Conclusions

The increasing need for fault tolerance imposed by the currently observed technology scaling introduces significant performance and power overheads. In our attempt to alleviate these overheads, the DeSyRe project will deliver a new generation of – by design – reliable systems, at a reduced power and performance cost. This is achieved through the following main contributions. Rather than aiming at totally fault-free chips, DeSyRe designs fault-tolerant systems built using unreliable components. In addition, DeSyRe systems are on-demand adaptive to various types and densities of faults, as well as to other system constraints and application requirements. A new dynamically reconfigurable substrate is designed and combined with runtime system software support in order to leverage on-demand adaptation, customization, and reliability at reduced cost. The above will result in a well-defined, generic and repeatable design framework for a large variety of SoCs. The proposed framework is applied to two medical SoCs with high reliability constraints and diverse performance and power requirements.

Acknowledgements

We would like to thank I. Pagkalos and P. Georgiou from Imperial College, London, for their contribution of the pancreas VHDL code (Biostator-II) and their support.

References

[1] S. Borkar, Designing reliable systems from unreliable components: the challenges of transistor variability and degradation, IEEE Micro 25 (6) (2005) 10–16.
[2] P. Hazucha, T. Karnik, J. Maiz, S. Walstra, B. Bloechel, J. Tschanz, G. Dermer, S. Hareland, P. Armstrong, S. Borkar, Neutron soft error rate measurements in a 90-nm CMOS process and scaling trends in SRAM from 0.25-μm to 90-nm generation, in: Electron Devices Meeting, 2003, IEEE International Technical Digest (IEDM), December 2003.
[3] T. Austin, V. Bertacco, S. Mahlke, Y. Cao, Reliable systems on unreliable fabrics, IEEE Des. Test 25 (4) (2008) 322–332.
[4] M.C. Shults, R.K. Rhodes, S.J. Updike, B.J. Gilligan, W.N. Reining, A telemetry-instrumentation system for monitoring multiple subcutaneously implanted glucose sensors, IEEE Trans. Biomed. Eng. (1994) 937–942.
[5] P. Atanasov et al., Implantation of a refillable glucose monitoring-telemetry device, Biosens. Bioelectron. 12 (7) (1997) 669–680.
[6] International Technology Roadmap for Semiconductors. <http://www.itrs.net/>.
[7] Xentium()® DSP Core. <http://www.recoresystems.com/technology/>.
[8] P.T. Wolkotte, Exploration within the Network-on-chip Paradigm, PhD Thesis, University of Twente, Enschede, The Netherlands, January 2009.
[9] E. Bolotin et al., QNoC: QoS architecture and design process for network on chip, J. Syst. Architect. (50) (2004).
[10] T. Ahonen, et al., CRISP: Cutting Edge Reconfigurable ICs for Stream Processing, Reconfigurable Computing – From FPGAs to Hardware/Software Codesign, Springer, 2011.
[11] K.H.G. Walters, et al., Multicore SoC for on-board payload signal processing, in: NASA/ESA Conf. on Adaptive Hardware and Systems (AHS), 2011, pp. 17–21.
[12] M.E. Wilinska et al., Simulation environment to evaluate closed-loop insulin delivery systems in Type 1 diabetes, J. Diabetes Sci. Technol. 4 (1) (2010).
[13] C.W. Chia et al., Glucose sensors: toward closed loop insulin delivery, Endocrinol. Metab. Clin. N. Am. 33 (2004) 175–195.
[14] N. Hardavellas, M. Ferdman, B. Falsafi, A. Ailamaki, Toward dark silicon in servers, IEEE Micro 31 (4) (2011) 6–15.
[15] S. Campagna, M. Violante, An hybrid architecture to detect transient faults in microprocessors: an experimental validation, in: Proc. of Design, Automation & Test in Europe Conf. & Exhibition (DATE)}, 2012, pp. 1433–1438.
[16] D.E. Ayguade et al., Ompss: a proposal for programming heterogeneous multi-core architectures, IEEE Parallel Process. Lett. (2011) 173–193.
[17] Jangwoo Kim, Nikos Hardavellas, Ken Mai, Babak Falsafi, James Hoe, Multi-bit error tolerant caches using two-dimensional error coding, IEEE MICRO Comput. Soc. (2007) 197–209.
[18] Mehrtash Manoochehri, Murali Annavaram, Michel Dubois: CPPC: correctable parity protected cache, ISCA (2011) 223–234.
[19] R.M. Seepers, C. Strydis, G.N. Gaydadjiev, Architecture-level fault tolerance for biomedical implants, SAMOS (2012) 104–112.
[20] A VHDL Library of Parametrisable Floating-Point and LNS Operators for FPGA. <http://www.ens-lyon.fr/LIP/Arenaire/Ware/FPLibrary/>.
[21] I. Pagkalos, P. Herrero, M. El-Sharkawy, P. Pesl, N. Oliver, P. Georgiou, in: VHDL Implementation of the Biostator II Glucose Control Algorithm for Critical Care, Biomedical Circuits and Systems Conference (BioCAS), 10–12, 2011, pp. 94–97.
[22] J.R. De Gruijl, P. Bazzigaluppi, M.T.G. de Jeu, C.I. De Zeeuw, Climbing fiber burst size and olivary sub-threshold oscillations in a network setting, PLoS Comput. Biol. (2012).

[23] N. Schweighofer, K. Doya, M. Kawato, Electrophysiological properties of inferior olive neurons: a compartmental model, J. Neurophysiol. 82 (2) (1999) 804–817.

[24] A. Hodgkin, A. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve, J. Physiol. 117 (1952) 500–544.

[25] R.A. Shafik et al., System-level design optimization of reliable and low power multiprocessor system-on-chip, Microelectron. Reliab. 52 (8) (2010) 1735–1748.

[26] R.A. Shafik et al., Software modification aided transient error tolerance for embedded systems,16th Euromicro Conference on Digital System Design (Euromicro DSD/SEAA 2013), Santander, Spain 04–06 (2013) 8.

[27] K. Yelick, D. Bonachea, W. Chen, P. Colella, K. Datta, J. Duell, S.L. Graham, P. Hargrove, P. Hilfinger, P. Husbands, C. Iancu, A. Kamil, R. Nishtala, J. Su, M. Welcome, T. Wen, Productivity and performance using partitioned global address space languages, in: International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010, pp. 1–6.

Ioannis Sourdis was born in Corfu, Greece, in 1979. He is an assistant professor of Computer Engineering at Chalmers University of Technology in Sweden. His research interests are on the architecture and design of computer and networking systems, reconfigurable computing, interconnection networks, and fault-tolerant computing. He has a Dipl-Eng (‘02) and a MSc (‘04) in Electronic and Computer Engineering from the Technical University of Crete, Greece, and a PhD (‘07) in Computer Engineering from the Technical University of Delft, The Netherlands. He is a member of the IEEE and the ACM.

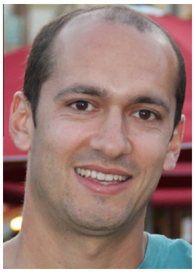
Christos Strydis studied Electronics & Computer Engineering at the Technical University of Crete, Greece, and in 2003 received his bachelor’s diploma (honors). In 2005 he obtained his M.Sc. degree (honors) in Computer Engineering from the Delft University of Technology, The Netherlands, with a minor in Biomedical Engineering. In 2011 he obtained his Ph.D. degree in Computer Engineering from the Delft University of Technology. Currently, he is a Postdoctoral researcher at the Erasmus Medical Center, the Netherlands, and is also chief engineer with Neurasmus BV, The Netherlands. His interests revolve around the topics of computer architecture, dependable systems, low-power embedded systems, high-performance neuroscience applications and biomedical microelectronic implants.

Antonino Armato, PhD is R&D Engineer at Yogitech SpA. He received the Laurea Degree in Electronic Engineering from the University of Messina, Italy and the Ph.D. degree in Bioengineering from the University of Pisa, in 2005 and 2011 respectively. Currently, he is pursuing his research work mainly at the R&D division at Yogitech SpA. His research interests are in mathematical modeling of faults on silicon and VLSI architectures to detect faults in integrated circuits and systems. He is author of papers on peer-review journals, contributions to international conferences and chapters in national books.

Christos-Savvas Bouganis (S’01–M’03) is a Senior Lecturer with the Electrical and Electronic Engineering Department, Imperial College London, London, U.K. He has published over 40 research papers in peer-referred journals and international conferences, and he has contributed three book chapters. His current research interests include the theory and practice of reconfigurable computing and design automation, mainly targeting digital signal processing algorithms. He currently serves on the program committees of many international conferences, including FPL, FPT, DATE, SPPRA, and VLSI-SoC. He is an Editorial Board Member of the IET Computers and Digital Techniques and the Journal of Systems Architecture. He has served as the General Chair of ARC in 2008 and the Program Chair of the IET FPGA designers’ forum in 2007.

Babak Falsafi is a professor of computer and communication sciences at EPFL, and the founding director of EcoCloud, an interdisciplinary research center targeting robust, economic, and environmentally friendly cloud technologies. He has a PhD in computer science from the University of Wisconsin Madison. He is a fellow of IEEE and a senior member of the ACM.

Georgi Gaydadjiev is a professor at the Department of Computer Science and Engineering at Chalmers University of Technology. His research interests include computer systems design, advanced computer architecture and micro-architecture, reconfigurable computing, hardware/software co-design and computer systems testing. He is a Senior IEEE and ACM member.



Sebastian Isaza studied Electronic Engineering at the University of Antioquia, Colombia, and graduated in 2004. In 2006 he obtained his MSc. degree in Embedded Systems Design at the University of Lugano, Switzerland. In 2011 he obtained the PhD. degree in Computer Engineering from the Delft University of Technology, the Netherlands. During 2012 he was a Postdoc researcher at the Erasmus Medical Center, the Netherlands. Currently he is an assistant professor at the University of Antioquia, Colombia. His research interests are in parallel computing for high-performance brain modeling and bioinformatics applications.

Alirad Malek was born in 1983. He received his B.S in electrical engineering from K.N. Toosi University of Technology, Tehran, Iran in 2006 and the MSc degree in System-on-Chip form Lund University of technology, Lund, Sweden in 2010. He has been working towards the Ph.D. degree in Chalmers University of technology, Computer science and engineering department, Gothenburg, Sweden, since 2011. His research interests are Fault-Tolerant system design, Interconnection Networks, Multicore Architectures and Reconfigurable Computing.

Riccardo Mariani holds a Ph.D. in Microelectronics from the University of Pisa. Before founding YOGITECH, he was Technical Director in Aurelia Microelettronica, CAD Laboratory and Team Director in CAEN Microelettronica, Digital Design Responsible and CAD Laboratory Coordinator in Centro TEAM, Digital Design Consultant in Italtel Center of Parma University. Riccardo won SGS-Thomson and Enrico Denoth Best Engineering Award. He is IEEE member and he has authored many papers related to High-Reliability Circuits, Design for Testability, Advanced Design Techniques and Asynchronous Circuits.

Dionisios Pnevmatikatos is a Professor and former Chair of the Electronic and Computing Engineering Department, Technical University of Crete and a Researcher at the Computer Architecture and VLSI Systems (CARV) Laboratory of the Institute of Computer Science, FORTH in Greece. He received his B.Sc. degree in Computer Science from the Department of Computer Science, University of Crete in 1989 and M.Sc. and Ph.D. degrees in Computer Science from the Department of Computer Science, University of Wisconsin-Madison in 1991 and 1995 respectively. His research interests are in the broader area of Computer Architecture, where he investigates the Design and Implementation of High-Performance and Cost-Effective Systems, Reliable System Design, and Reconfigurable Computing. He is currently the Coordinator of the FASTER project (ICT, STREP) and has participated in numerous national and European research projects.

Prof. Dhiraj K. Pradhan (Fellow of IEEE, ACM and JSPS) is currently a Professor in the Department of Computer Science at University of Bristol (UK). He has over 30 years of industrial and academic research experience in reliable and low power design and VLSI CAD tools development. His long and distinguished career began at the IBM System Development Division, in 1972. Later he served various other academic institutions including Texas A&M, Stanford, etc. Over the years, he has published more than 450 refereed papers and edited/authored/co-authored at least 12 research monographs and books. He is internationally renowned for his research, innovations and seminal contributions, for which he has earned numerous awards and fellowships. He regularly participates in the technical/steering committees of major international conferences and is an active member of the editorial committees of reputed journals.

Gerard Rauwerda obtained his MSc degree in Electrical Engineering at the University of Twente (The Netherlands) in 2002. He received his PhD degree from the University of Twente in January 2008 for his thesis titled "Multi-Standard Adaptive Wireless Communication Receivers – Adaptive Applications Mapped on Heterogeneous Dynamically Reconfigurable Hardware". He worked for Ericsson and later at the University of Twente (The Netherlands), and he was a visiting researcher at Atmel Germany. He has over 10 years' experience in wireless communications systems and reconfigurable embedded systems. He is co-founder of Recore Systems and currently engaged as Chief Technology Officer.

Robert Seepers is a PhD student at Erasmus Medical Center and an engineer at the research and development company Neurasmus BV, The Netherlands. His research interests include fault-tolerant design and low-power optimizations for implantable, biomedical devices. He has BSc ('08) and an MSc ('11) in electronic and computer engineering from Delft University of Technology, The Netherlands.

Dr. Rishad A. Shafik (member of IEEE and IET) is currently working as an academic research staff at the University of Bristol, UK. Prior to joining Bristol in Oct 2011, he worked at the University of Southampton, UK, as a research fellow. He received his PhD and MSc degrees from the same in 2010 and 2005, and his BSc degree in electronic engineering from IUT (Bangladesh) in 2001, respectively. He has published more than 35 refereed research articles in leading international conferences and journals. Dr Rishad is also actively engaged in various conference technical programme committees, journal editorial committees and consultancy services.

Kim Sunesen has studied at Aarhus University, Denmark, Université Louis Pasteur Strasbourg, France, and the Technical University of Munich, Germany. He holds a PhD. in computer science from Aarhus University. He manages research and development projects for Recore Systems' digital signal processing platform chips. His responsibilities include both hardware and software developments projects. He has more than eleven years of experience working in the semiconductor and EDA/Case tooling industry in The Netherlands, France and Denmark. Kim Sunesen is R&D project manager at Recore Systems

Dimitris Theodoropoulos (S'06) was born in Athens, Greece. In 2003 and 2006 he obtained his Diploma (5-year degree) and M.Sc degree respectively from the Electronic and Computer Engineering dept. at the Technical University of Crete, Greece. In 2007, he joined the Computer Engineering dept. of the Delft University of Technology, the Netherlands, where he worked towards his Ph.D and participated in the "hArtes" FP6 European project. In 2011, he joined the Computer Architecture and VLSI Systems group at the Foundation for Research and Technology – Hellas (FORTH) in Greece, where he is working as a post-doc researcher for the DeSyRe FP7 European project. He is a member of the Technical Chamber of Greece from 2003. His current research interests include: reconfigurable computing, embedded systems, and computer architecture.

Stavros Tzilis was born in Iraklio, Crete, Greece in 1982. He studied Electrical Engineering in Democritus University of Thrace, located in Xanthi, Greece, obtaining his Engineering Diploma in 2006. He went on to obtain his MSc in Computer Engineering from Delft University of Technology in 2010. Today he is pursuing PhD studies in Chalmers University of Technology, located in Gothenburg, Sweden, focusing on Fault Tolerance and Graceful Degradation. He has also worked for the Microelectronics Section of the European Space Agency.

Michail Vavouras is a Research Assistant and PhD Candidate in the Circuits and Systems Research Group, Department of Electrical and Electronic Engineering, Imperial College London, UK. His current research interests include reliability and fault tolerance techniques for reconfigurable computing (FPGAs)