Fault-Tolerant Platforms for Automotive Safety-Critical Applications

M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli
PARADES EEIG
Via San Pantaleo 66
Rome, Italy
mbaleani, aferrari, leon, alberto@parades.rm.cnr.it

Maurizio Peri, Saverio Pezzini
ST Microelectronics
Via C. Olivetti 2
Agrate Brianza, Italy
maurizio.peri, saverio.pezzini@st.com

ABSTRACT

Fault-tolerant electronic sub-systems are becoming a standard requirement in the automotive industrial sector as electronics becomes pervasive in present cars. We address the issue of fault tolerant chip architectures for automotive applications. We begin by reviewing fault-tolerant architectures commonly used in other industrial domains where fault-tolerant electronics has been a must for a number of years, e.g., the aircraft manufacturing industrial sector. We then proceed to investigate how these architecture could be implemented on a single chip and we compare them with a metric that combines traditional terms such as cost, performance and fault coverage with flexibility, i.e. the ability of adapting to changing requirements and capturing a wide range of applications, an emerging criterion for platform design. Finally, we describe in some details a cost effective dual lock-step platform that can be used as a single fail-operational unit or as two fail-silent channels trading fault-tolerance for performance.

Categories and Subject Descriptors

B.8 [**Performance and Reliability**]: Performance Analysis and Design Aids; C.0 [**General**]: System architectures; C.1.2 [**Processors Architectures**]: Multiple Data Stream Architectures (Multiprocessors)—*Interconnection architectures (e.g., common bus, multi-port memory, crossbar switch), Multiple-instruction-stream, multiple-data-stream processors (MIMD)*; C.3 [**Special-Purpose and Application-Based Systems**]: Microprocessor/microcomputer applications, Real-time and embedded systems; C.4 [**Performance of Systems**]: Design studies, Fault-tolerance, Reliability, Availability, and Serviceability; C.5.4 [**Computer System Implementation**]: VLSI Systems

CASES'03, Oct. 30–Nov. 2, 2003, San Jose, California, USA.

General Terms

Design, Reliability

Keywords

system-on-a-chip, fault-tolerant, VLSI, multi-processor, automotive, safety critical

1. INTRODUCTION

1.1 Electronics in the Car

The introduction of digitally controlled combustion engines with fuel injection and digitally controlled anti-lock brake systems (ABS) in the late 70's was just the first step towards a pervasive use of electronics in the car. Close synergy between mechanics and electronics yields several benefits that we can measure in terms of better fuel economy, better vehicle performance in adverse conditions, driver assisting functions such as ABS, traction control (TCS), electronic stability control (ESP), and brake assistant (BA) and safety features such as collision warning and even automatic collision avoidance systems.

To design cars with better performance and higher level of safety, engineers must substitute mechanical interfaces between the driver and the vehicle with electronic systems. These systems are common in the aerospace industry and are generically called X-by-wire systems. X-by-wire systems consist of a driver's operating unit (throttle pedal, brake pedal, gear selector, steering wheel) whose electrical output is processed by micro-controllers that manage the power-train, braking and steering activities via electrical actuators. Throttle-by-wire, shift-by-wire, and driver assistance systems have been used successfully for many years.

1.2 Fault-Tolerance Requirements

The fault-tolerance requirements in cars are minimal since the scenario considered is the single-fault one [8]. In the case of the previously mentioned electronic sub-systems, either a mechanical backup exists, or a fail-safe mechanism is guaranteed by mechanical sub-systems in the event of electronic failures. In the case of throttle-by-wire, the throttle spring system provides a reduced engine speed in the event of electronic failure. Similarly, when electronic braking functions (ABS, TCS, ESP, BA) fail, the brake system behaves like a

conventional one providing a mechanical backup. Mechanical backups relieve electronics of stringent fault-tolerance requirements, but they are costly, heavy, potentially critical in the case of accident (e.g. steering column) and somehow limit the potential of electronics in terms of performance and flexibility. Moving to X-by-wire systems without mechanical backup (brake-by-wire, steer-by-wire), as it is done today in fly-by-wire aircraft, will require to build highly reliable and fault-tolerant electronic systems. Indeed, the real challenge is to build these fault-tolerant systems with hard real-time requirements for mass market and at a reasonable cost.

With respect to fly-by-wire systems in the aerospace, automotive electronic systems present some distinctive features in terms of safety requirements. Particularly, a safe state can be reached easier and faster in the event of hazardous failures. For automobiles, standstill or low speed at a non-hazardous place represent a safe state. This fail-safe condition propagates differently to electronic components according to their hazard severity for failures and the inherent fault-tolerance possibilities due to mechanical backups and/or intrinsic redundancy. In the case of a brake-by-wire system, the hazard severity is mitigated by the intrinsic redundancy of the braking system (four braking wheels).

1.3 Fault-Tolerant Design Approaches

Any form of fault-tolerance is based on redundancy that can be spatial, or temporal, or pertaining to information. Redundancy alone does not guarantee fault-tolerance. On the contrary, redundant structures show a higher fault arrival rate compared to a non-redundant system. It is of paramount importance how redundancy is managed. One of the most important issues is the definition of fault-containment regions (FCR's) i.e. "collection of components that operate correctly regardless of any arbitrary logical or electrical fault outside the region" [9] and whose faults do not cross region boundaries. This generally requires the use of independent power and clock sources, the electrical isolation of interfaces and may also require physical separation in order to avoid common-mode failures. These fault-tolerance requirements call for multi-chip/multi-package solutions, at least for fail-operational structures and are apparently clashing with current silicon technology trends. In fact, recent advances in device integration and IC packaging make the implementation of complete systems on a single chip (SoC's) or in a single package (SiP's) not only viable but also cost effective. Indeed, the most hazardous drive-by-wire applications will deploy redundant distributed architectures implementing fail-operational configurations. Nonetheless, single chip fail-operational architectures may be of important value if we account for the occurrence of soft errors.

1.4 Contributions

In this paper, we focus on fault-tolerant system-on-a-chip platforms. We begin by describing the issues in system-on-a-chip fault-tolerant design (Section 2). We then present in Section 3 several alternatives for an automotive control platform trading performances for system integrity, i.e. the probability of the system satisfactorily performing the required functions under all the stated conditions within a stated period of time. In analyzing each of the alternatives, we leverage the great deal of literature inherited from traditional safety related disciplines (railway, aircraft, space, military, and nuclear systems), we consider the unique safety requirements and features of automotive systems, and we harmonize all these aspects with the current silicon technology trends. The main contributions of this paper come from the analysis of a system-on-a-chip implementation (Section 4) and the effective design space exploration presented in Section 5, where the necessary trade-offs between cost, performance and system integrity are discussed within an automotive domain application context. The results show that a great deal of flexibility in performance versus system integrity can be achieved at an acceptable cost.

2. FAULT-TOLERANCE IN SOC

Single-chip solutions cannot be divided up into fault containment regions, i.e. they are subject to common mode failures. Common mode failures occur because of faults whose occurrence causes the failure of all replicas. For instance, duplication cannot provide any protection against a common mode failure that stops both CPUs, e.g. a failure on the power supply. In a single-chip implementation, usual sources of common mode failures are the clock tree, the power supply and the silicon substrate.

Design techniques can be adopted to separate clock and power distribution or to make them more robust and/or clock and power supply monitors [3] can observe the correct operation and signal any deviation.

On the other hand, as pointed out before, we cannot prevent other common mode failures as those caused by mechanical faults (e.g. faults hitting the packages or the substrate) from affecting system operation. If the probability of these events is not small enough to satisfy the safety requirements, the only solution available is replicating the entire chip keeping the replicas physically separated.

Although single-chip implementations of electronic systems suffer from common modes of failure, the use of redundancy within the system itself is still to be considered a fundamental design practice. The higher level of integration bears as a side effect a non-negligible increase in the probability of occurrence of permanent and transient faults. In fact, the reduced pitch and wire width, due to the higher integration level, significantly increase the probability of occurrence of an undesired short or open circuit, causing a permanent fault in the system.

Similarly, transient faults, also called single event upsets (SEU's), are due to electrical noise or external radiations [2]. Radiation can, directly or indirectly, induce localized ionization events capable of upsetting internal data states. While the upset causes a data error, the circuit itself is undamaged and the system experiences a transient fault. The data upsetting is called *soft error* and the rate at which these events occur is called *soft error rate* (SER). Soft errors are already of great concern in devices built in current technologies [13] and will become a major issue as device sizes continue to shrink not only for memory components but also for combinational logic [1, 14].

While the occurrence of a permanent fault may impair or even stop the correct functionality of the system, soft errors caused by transient faults often drastically reduce the system availability. As a matter of fact, it is often the case that soft error avoidance is strongly required to maintain the system availability at an acceptable level.

Due to the transient nature of these faults, temporal redundancy techniques can be used to implement fault detection and masking. Triple execution and majority voting

represents a static temporal redundancy technique that can be used to mask any single soft error. A dynamic technique can be realized deploying error detection (e.g. via duplication and comparison) and recovery via checkpointing and roll-back.

The former solution may result too demanding, particularly taking into account the hard real-time requirements of automotive applications. The latter technique may yield to unacceptable degradation in terms of the availability of the service provided by the system especially when the soft error rate (SER) is sensibly higher than the permanent fault rate. This is why we believe that some form of spatial redundancy must be employed at the chip level in order to mask or detect soft errors. While the error detection drastically simplifies the system roll-back and restart, error masking eliminate (or at least reduce) this need thus maintaining the provided availability at an acceptable level.

3. A SURVEY OF FAULT-TOLERANT MULTI-PROCESSOR ARCHITECTURES

Comprehensive surveys on redundant structures can be found in [10, 12, 4]. In this section we review the basic features of the various alternatives.

3.1 Lock-Step Dual Processor Architecture

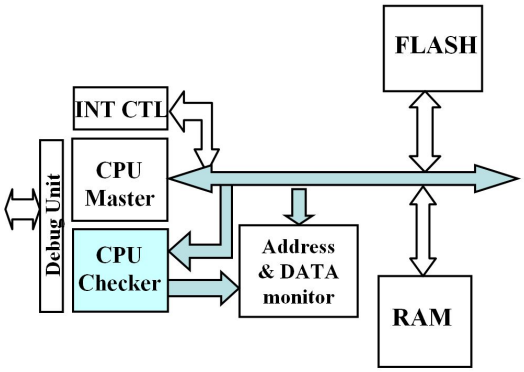

Figure 1: Lock-step dual processor architecture

In a lock-step architecture (Figure 1), two processors usually referred to as the *master* and the *checker*, execute the same code being strictly synchronized. The master has access to the system memory and drives all system outputs, while the checker continuously executes the instructions moving on the bus (i.e. those fetched by the master processor). The outputs produced by the checker, both addresses and data, feed the compare logic (monitor), consisting of a comparator circuit at the master's and checker's bus interfaces, that checks the consistency of their data-, address- and control-lines. The detection of a disagreement on the value of any pair of duplicated bus lines reveals the presence of a fault on either CPU without giving the chance to identify the faulty CPU.

The monitor is not capable of detecting bus and memory errors. These errors are in fact a source of common-mode failure causing both CPU to fail the same way. Bus and memory must hence be protected against faults deploying error detection (correction) techniques such as parity bits (Error Correcting Codes).

The lock-step architecture can be employed as a fail-silent node providing the capability of detecting any (100% coverage) single error (permanent or transient) occurring indifferently on the CPU, memory or communication sub-system. Error correcting codes are required when errors occurring on busses and memories turn out to be relatively frequent due to the occurrence of transient faults.

Due to the relatively low gate count of its logic, the interrupt controller can be designed to guarantee the required robustness using ad-hoc techniques or it can be provided with fault-tolerance mechanisms such as self-checking hardware.

3.2 Loosely-Synchronized Dual Processor Architecture

In a loosely-synchronized architecture (Figure 2) two CPUs run independently having access to distinct memory subsystems. A real-time operating system running on both CPUs handles interprocessor communication and synchronization and is responsible for error detection (e.g. by means of cross-checks), correction and containment (e.g. memory protection).

A subset (possibly all) of the tasks executed by the processors are defined as *critical*. The image of critical tasks is duplicated on both memories. Critical tasks are executed in parallel as software replicas and their outputs are exchanged after each run on a time triggered basis. Both processors are responsible for checking their consistency. A mismatch indicates a fault on the CPU, memory or communication sub-system and prevents outputs from being committed.

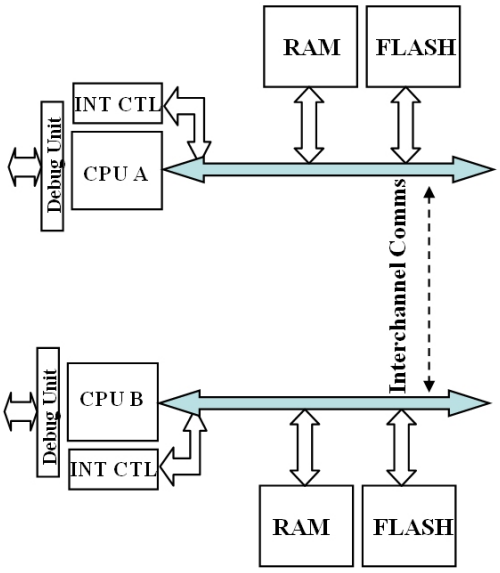

Figure 2: Loosely-Synchronized Dual-Processor architecture

When a cross-check mismatch has been detected, predefined computation, called *sanity-check*, or other self-testing techniques can be used on both processors to identify the faulty component. If the check succeeds, i.e. the sanity-check on only one processor detects a fault condition, the

system may degrade its operation (instead of halting) using the healthy component only. Notice that, if the sanity-check coverage is less than 100%, there is no guarantee that the operational CPU be fault-free. In fact the probability that both CPU experience a fault might be non-negligible since the fault rate must be integrated over a time window starting from the last known fault-free condition.

In order to guarantee the commitment of agreed outputs only, several mechanism may be employed. With *protected accesses*, only specific processes such as RTOS services or dedicated tasks are granted access to the system outputs. Alternatively, to prevent outputs from being committed before being cross-checked, *time guardians* can restrict CPU access to system outputs to a predefined time-window. The mechanism can be implemented in hardware, by synchronizing it with the cross-check executions, or handled by the RTOS. A third technique is based on the use of signatures. Each processor adds its own signature to the outputs of critical tasks and the receiver checks for both signatures before accepting the data.

According to the subset of critical task, the architecture can appear in several different configurations. At the one end, fully critical applications must be entirely replicated, thus requiring twice as much memory while providing the same performance as a single processor architecture. Actually, in this case, the lock-step architecture outperforms the loosely-synchronized one, for the performance overhead required for executing sanity- and cross-checks does not compare with the relatively low performance penalty due to the compare logic. At the other end, when only a few sporadic tasks are identified as critical, the two processors can run different tasks in parallel approaching the performance of a dual-processor architecture.

The execution of a function on both CPUs guarantees the detection of any error (100% coverage) occurring indifferently on one of the CPUs, busses or memories. Since busses and memories (at least for critical tasks) are replicated, no other form of redundancy (e.g. parity bits) is needed to detect errors on these components. Nevertheless, ECCs may be employed in the case of high memory (or bus) failure rate.

3.3 Triple Modular Redundant (TMR) Architecture

In the TMR configuration (Figure 3) three identical CPUs execute the same code in lock-step and a majority vote of the outputs masks any possible single CPU fault. The memory and communication sub-system faults can be masked employing ECC techniques.

Other TMR configurations can be devised. For example, three loosely synchronized cores (possibly heterogeneous) can replace the lock-step modules or a dynamic redundancy scheme with hot/cold standby can be deployed. These configurations are not considered in the present work since we believe they are outperformed in terms of cost, performance and fault-tolerance characteristics by the dual lock-step architecture presented below.

3.4 Dual Lock-Step Architecture

A configuration largely employed in multi-chip fault-tolerant systems consists of the combination of two fail-silent channels, each one consisting of a lock-step architecture as the one presented in Section 3.1, building up a single fail-operational

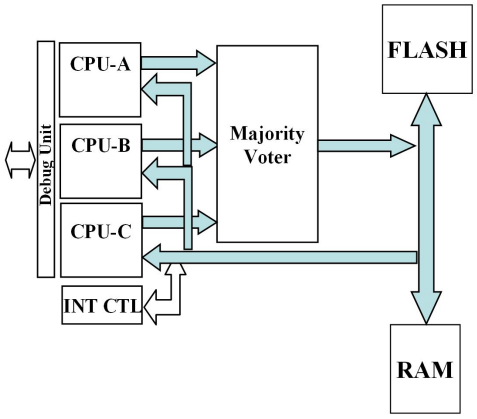

Figure 3: Triple modular redundant (TMR) architecture

unit (see Figure 4). In this case, the architecture provides fault-tolerance only for the replicated tasks, whose outputs are checked before being committed. Notice that software design errors can be prevented as well. In contrast to solution presented in Section 3.2, the execution of sanity-checks is no more required, since self-checking capabilities are already provided in hardware by means of duplication and yield a 100% fault coverage. Moreover, in degraded mode, while fault-tolerance cannot be guaranteed, the architecture still preserves fail-silence capabilities.

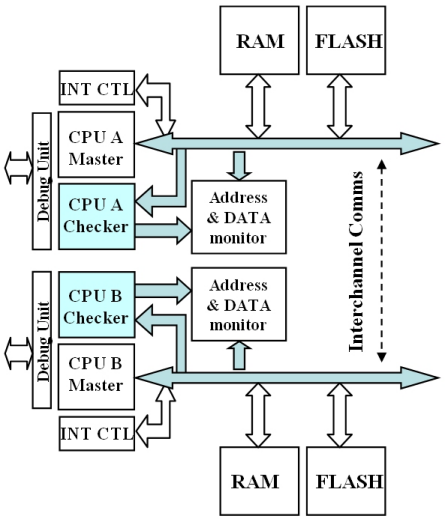

Figure 4: Dual Lock-Step architecture

4. SOC FAULT-TOLERANT ARCHITECTURE IMPLEMENTATION

The SoC implementation of the architectures presented in Section 3 opens the way to new opportunities and challenges. Due to the costs associated to the higher integration level, single-chip implementations should have enough flexibility to support a wide range of applications [6] in order to share

the silicon development cost across a set of different final electronic systems.

Here, the term flexibility describes the capability of a silicon solution to correctly adapt to performance, cost and fault-tolerance requirements of a set of applications, after silicon production. The flexibility can be obtained via software configurability or programmability during software design or even run-time.

While, from the architectural standpoint, the simple architecture of the lock-step and the TMR solutions offer very small optimization margins, important variations of the loosely-synchronized and dual fail-silent architectures can be explored that provide significant improvement in both performance and flexibility. In this section we first examine such architectural variations and then provide a comparison of the single-chip implementation of the four architectures.

In contrast to multi-chip solutions, in a single-chip dual-processor architecture the memory sub-system can be shared between the processors at much lower cost. Since the two cores can run independently, the memory and communication sub-systems are likely to become a major performance bottleneck. For this reason, as we already proposed in [5], the memory sub-system is split into 4 banks (2 for code and data respectively) and the traditional bus is replaced by a more performant crossbar switch, which guarantees sufficient bandwidth between the processor and memory sub-systems.

As a result, while the sensitivity to failures only marginally changes due to the small amount of gates additionally needed to implement the shared memory sub-system, the flexibility of the architecture significantly improves. In fact, the physical memory partition does no more constrain the application size on each processor and unbalanced applications can be supported.

Figure 5 shows the single-chip loosely-synchronized dual-processor architecture, called *Shared-Memory (SM) Loosely-Synchronized Dual-Processor* in the sequel. Since the memory sub-system is shared between the processors, the duplication of critical code becomes a trade-off between system integrity, memory size and performance: while critical code takes up costly memory space, non-duplicated critical code, which must be executed on both cores, runs at half the speed of a single processor.

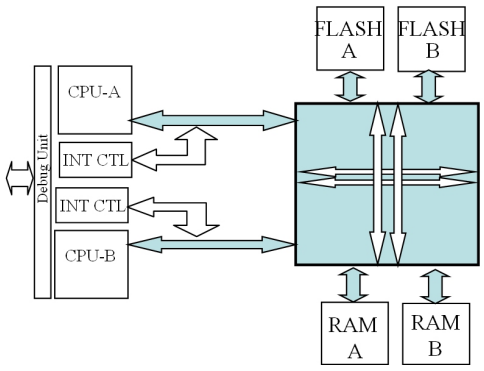

Figure 5: SM Loosely-Synchronized Dual-Processor architecture

The most promising opportunities in the single-chip implementation of the presented architectures come from the integration of the dual lock-step configuration, as it will be presently discussed. In the *SM Dual Lock-Step* architecture (Figure 6) the two fail-silent channels share the same memory sub-system. This solution largely enhances flexibility, since it covers the TMR solution (same fault-tolerance properties), while implementing the dual lock-step architecture. In fact, when fail-operational capability is required, the two channels can be arranged in lock-step mode, in which case the architecture provides masking capabilities of CPU's faults as in the TMR solution, otherwise they can be used as two completely parallel fail-silent channels providing double performance. Memories and buses are protected using ECCs in order to retain error masking capabilities on these components when operating in lock-step mode.

The lock-step and parallel modes are just the two extremes of a continuum of operating modes the architecture can be used in. For applications featuring just a few critical tasks, we can follow an approach similar to Section 3.2, by only loosely synchronizing the two fail-silent channels.

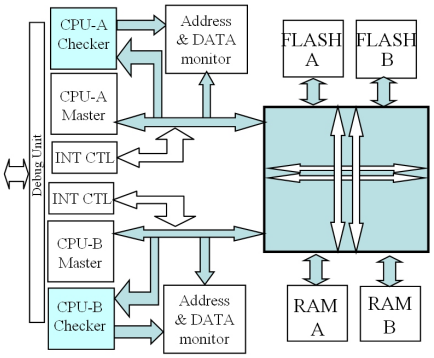

Figure 6: SM Dual Lock-Step architecture

5. IMPLEMENTATION ISSUES AND COMPARISONS

The design space exploration of the previously described single-chip architectures is carried out in a three-dimensional design space by analyzing the best trade-off between cost, performance and fault-tolerance capabilities. In order to support our assertions, we compare the performance and the fault-tolerance features of the different solutions and evaluate their cost on the basis of the area estimates derived from the current implementation of the architecture presented in [5].

CPU	FLASH (1Mbit)	RAM (1KByte)	Bus (32bit)	ECC overhead
1	4.15	0.17	0.5	7/32

Table 1: Area of embedded memory components normalized to CPU footprint

Table 1 summarizes the area of memory components (both RAM and FLASH) and buses, normalized to the CPU foot-

	#CPU	CPU area	#RAM	RAM area	#FLASH	FLASH area	#Bus	Bus area	Area
Low-range									
Single CPU no ECC	1	1.0	1	2.0	1	8.0	1	0.5	11.5
Lock-step mem/bus ECC	2	2.0	1	2.4	1	9.8	1	0.6	14.5
SM Loosely-Sync mem ECC	2	2.0	2	4.9	2	19.5	2	1.0	27.4
TMR mem/bus ECC	3	3.0	1	2.4	1	9.8	1	0.6	15.8
SM Dual LS mem/bus ECC	4	4.0	1	2.4	1	9.8	2	1.2	17.4
Mid-range									
Single CPU no ECC	1	1.0	1	10.0	1	41.0	1	0.5	52.5
Lock-step mem/bus ECC	2	2.0	1	12.2	1	50.0	1	0.6	64.8
SM Loosely-Sync mem ECC	2	2.0	2	24.4	2	99.9	2	1.0	127.3
TMR mem/bus ECC	3	3.0	1	12.2	1	50.0	1	0.6	65.8
SM Dual LS mem/bus ECC	4	4.0	1	12.2	1	50.0	2	1.2	67.4

Table 2: Cost of different architectures for low-/mid-range X-by-wire systems

print. The approximated overhead due to the introduction of memory error correction code is also shown.

Table 2 shows the estimated area for each of the alternatives presented in Section 3 for typical low-range and mid-range X-by-wire systems.

The single CPU architecture can be considered as a reference design satisfying computational and memory requirements but not providing any fault-tolerance capability.

5.1 The Lock-Step Architecture

The lock-step architecture cannot provide any performance boost over the single processor solution, since the two cores are bound to execute the same code cycle by cycle. Rather, due to the introduction of the compare logic and the ECC coders/decoders in the critical path, the clock rate may be decreased.

However, with a relatively low area overhead (see Table 2), this solution provides a 100% fault coverage within an error detection time in the order of the clock period. This feature together with an error detection mechanism which is local to the CPU, memory and bus sub-systems eases fault diagnosis. Even if the lock-step configuration does not provide any degraded mode of operation, fault diagnosis can be very important to recover from transient faults.

Since both processors execute the same code, the lock-step configuration does not provide any protection against software design errors. On the other hand, hardware design diversity is partially supported (e.g. implementing the dual core as the combination of a hard-IP and a soft-IP) even if this might make core synchronization more complex.

5.2 SM Loosely-Synchronized Dual-Processor Architecture

In the SM loosely-synchronized dual-processor architecture the two CPUs can run independently having full access to the memory sub-system and system I/O. Since only critical tasks must be duplicated for safety requirements, we can trade *criticality* for system performance.

As the lock-step configuration, the SM loosely-synchronized architecture provides a 100% error detection when running full-critical applications. However, this requires roughly twice as much memory space to accommodate the duplicated code. Memory footprint is mostly responsible for the huge area overhead[1] as shown in Table reftab:results. Moreover, fault diagnosis is complicated by the longer error detection time, proportional to the check execution period, and by the fact that error detection only performed on selected outputs. Nonetheless, in contrast to the lock-step solution, the SM loosely-synchronized architecture has the ability of supporting both hardware and software design diversity and provides a degraded mode of operation.

Both configurations presented above provide no fault masking mechanism, except for the possible implementation of ECCs on buses and memories. This may be a major drawback especially in the case of a high transient fault rate.

5.3 Triple Modular Redundant Architecture

The TMR configuration represents a "low-cost" solution. In fact, the area overhead over the lock-step architecture is as low as 9% and 1.5% for low- and mid-range systems respectively. However, it also inherits almost all of the features and flaws of the lock-step architecture. Excepting its unique capability of masking any single fault, at the cost of an additional CPU, it offers a 100% error detection coverage within a single clock period. On the other hand, performances are limited to those of a single processor architecture, and neither degraded mode nor support for software design diversity are provided. Actually we can view the TMR as operating in degraded mode when it is working with just two healthy CPUs: in this condition the TMR system operates as a fail-silent node.

5.4 SM Dual Lock-Step Architecture

The SM dual lock-step architecture combines the advantages of the SM loosely-synchronized solution in terms of flexibility with the fault masking capabilities provided by the TMR architecture.

The two fail-silent channels can be used in different configurations trading fault-tolerance capabilities for performance and some area overhead. When the two cores execute the

[1]Notice that since memory is duplicated, ECC protection on memory is not needed for error detection. However it improves system availability assuming a relatively higher error rate affecting the memory sub-system than other system components. Without memory ECC the estimated area would be of 23.0 and 105.0 for low- and mid-range systems respectively. A similar argument holds for the communication sub-system.

same code in lock-step, they provide fault-tolerance capabilities. On the other hand, if the fail-silence property suffices for the application at hand, the two channels can operate completely independently and the architecture behaves like a "traditional" dual processor solution.

This great deal of flexibility comes at a relatively low price. In fact, if compared with the fault-tolerant TMR architecture, while the introduction of the 4th CPU yields a 10% overhead for low-range applications, the overhead falls down to just 2-3% for more memory demanding applications (i.e. mid-range). Notice that to cover software design faults[2] via design diversity, we need to double the memory footprint as done for the SM loosely-synchronized architecture. Also in this case, comparing the two alternatives, we come out with a modest increase in area, in the order of about 8% and 2% for low- and mid-range applications respectively.

5.5 Trade-off Analysis

As shown in Table 2 the SM loosely-synchronized architecture is by far the most area-demanding solution. The performance improvement over the lock-step architecture and its fault-tolerance characteristics do not justify the additional cost due to the large area overhead. Hence, the choice is restricted to the lock-step solution, the TMR configuration and the SM dual lock-step architecture (the single processor solution is out of the scope for it provides no error detection mechanism).

Both the lock-step and the TMR architectures cannot provide any performance improvement over the single processor solution, while representing "low-cost" solutions. On the other hand, the versatility (100% single fault-tolerance vs. fail-silence plus double performance) of the SM dual lock-step architecture allows to use the same platform for a wider range of applications, reducing engineering costs and significantly augmenting product volumes.

Based on the considerations above we strongly believe that the SM dual lock-step solution represents the best alternative between the four architectures considered in this paper for the given application range.

6. SM DUAL LOCK-STEP ARCHITECTURE IMPLEMENTATION DETAILS

The 4 CPUs used in the implementation of the dual lock-step architecture can be 4 ARM7 or ARM9 providing roughly 0.9 and 1.1 instruction per cycle respectively. In 0.13μm technology the maximum clock frequency can reach 80MHz and 200MHz. However, without the introduction of a more complex memory hierarchy (i.e. cache) the maximum clock frequency will be limited by the embedded memory access time which is expected to be about 60ns.

The 4 CPU are arranged in two lock-step pairs with the possibility of executing all 4 in a lock-step mode. One CPU for each couple drives the outputs to the memory or I/O subsystem while the other provides replicated outputs to a self-checking checker for duplication (or double rail) codes [11].

When the two pairs are used in lock-step mode to guarantee fault-tolerance, the system provides the same performances as a single CPU architecture (if we neglect the overhead introduced by fault-tolerant mechanisms). In this mode either outputs can be used to access system the memory and peripheral modules. Since each couple forms a fail-silent sub-system either both outputs are correct or one subsystem provides no output at all.

When used for parallel processing the architecture can double the performance while retaining the fail-silence property. Since in this operating mode accessing the system memory becomes the major performance bottleneck, we borrow many of the architectural principles introduced in [5] to overcome the same problem.

The memory sub-system provides 4 concurrent ports, two for SRAM and two for FLASH accesses. At the same time, a hierarchical interconnection scheme substitutes the traditional bus backbone. The two processor pairs, the I/O and memory ports are connected to a crossbar network. I/O devices share a high- or low-performance bus, according to their bandwidth requirements, which can be accessed by master devices (typically the two processor and a DMA) via the crossbar I/O ports.

We have to remark that other approaches exist that do not depend on the replication of the central processing unit. These approaches include concurrent checking methods based on coding techniques, on-line Built-In Self-Test (BIST) and Built-In Current-Monitoring (I_{DDQ}-Test). The AE11 microcontroller [3], specifically developed for safety-critical automotive applications, deploys a combination of these different techniques. These techniques yield a lower overhead than duplication at the expense of fault coverage, error detection latency and performances.

While on-line BIST and I_{DDQ} testing are extremely important for peripheral modules, we believe that duplication is the best approach for detecting/tolerating errors on the CPU sub-system, which is the focus of this paper. The AE11 controller deploys parity based coding techniques to detect errors on the CPU data path and signature monitoring [15] for the control path. These methods entail a complete redesign of the CPU and at least a major modification of the software tool-chain (e.g. a customized assembler). While the related development costs can be sustainable for an 8-bit CPU in 0.7μm technology, or at least comparable to the cost of duplicating the entire CPU, this is no longer the case for 32-bit architectures in 0.18-0.13μm technologies.

Last but not least, the selected solution does not tie us to a particular CPU core manufacturer and architecture, leaving us the freedom of choosing the "best" (in terms of both performance and cost) CPU core for the application at hand [3].

7. CONCLUDING REMARKS

In this paper, we have proposed a single-chip solution, devised for fault-tolerant automotive applications, which is based on the use of two lock-step channels (4 CPUs overall), a cross-bar communication architecture and embedded memories.

The reasons behind this choice are manifold. From cost perspective, since the design is dominated by the embedded memory area (from 70% to even 94% of the overall area required for CPUs, memories and communication in-

[2]With respect to software design faults the dual lock-step architecture cannot provide fault-tolerance, since one pair of processors is simultaneously affected by the same fault and cannot detect it. Additional checks (e.g. alternate computation, consistency checks) can be deployed to preserve fault-tolerance capabilities.

[3]This aspect is underlined also in [7] and claimed to be a major advantage of the Delphi Secured Micro-controller.

frastructure) the introduction of the redundant CPUs affects marginally the overall cost.

The availability of two fail-silent channels, which can be used as lock-stepped or loosely synchronized computational units, provides a great deal of flexibility. This flexibility can be exploited in two different ways:

1. The dual lock-step platform fulfills the requirements of a large application space. In fact, it can be used for a class of applications that call for high computational power as well as for another class of applications that must satisfy more stringent safety requirements.

2. For each application, designers may still trade performance for fault-tolerance features. In fact, they have the freedom of moving from a single fail-operational channel with a 100% coverage of CPU, memory, and bus faults, to a double-performance fail-silent architecture, passing through a continuum of performance/fault-tolerance trade-off alternatives.

8. ACKNOWLEDGMENTS

This work has been partially supported by the European Community project IST-200138314 Columbus and by the NFS ITR CCR-0225610 Center for Hybrid Embedded Software Systems (CHESS) [4]. We would also like to thank Claudio Pinello, University of California at Berkeley, and Marco Carloni, University of Bologna, Italy, for their important contributions.

9. REFERENCES

[1] R. Baumann. The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction. In *Digest of the Internation Electron Devices Meeting IEDM'02.*, pages 329–332, 2002.

[2] R.C. Baumann. Soft errors in advanced semiconductor devices - part I: The three radiation sources. *IEEE Transaction on Device and Materials Reliability*, 1(1):17–22, Mar 2001.

[3] E. Böhl, Th. Lindenkreuz, and R. Stephan. The fail-stop controller AE11. In *Proceedings of the International Test Conference*, pages 567–577, Nov 1997.

[4] F. Cristian. Understanding fault-tolerant distributed systems. *Communications of the ACM*, 34(2):56–78, Feb 1991.

[5] A. Ferrari, S. Garue, M. Peri, S. Pezzini, L. Valsecchi, F. Andretta, and W. Nesci. Design and implementation of a dual processor platform for power-train systems. In *Proceedings of Convergence Conference*, October 2000.

[6] A. Ferrari and A. Sangiovanni-Vincentelli. System design: Traditional concepts and new paradigms. In *Proceedings of the International Conference on Computer Design*, October 1996.

[7] L. T. Fruehling. Delphi secured microcontroller architecture. In *Desing and Technologies for Automotive Safety-Critical Systems, SAE 2000 World Congress*, 2000.

[8] R. Isermann, R. Schwarz, and S. Stolzl. Fault-tolerant drive-by-wire systems. *IEEE Control Systems Magazine*, 22(5):64–81, Oct 2002.

[9] J.H. Lala and R.E. Harper. Architectural principles for safety-critical real-time applications. *Proceedings of the IEEE*, 82(1):25–40, Jan 1994.

[10] V.P. Nelson. Fault-tolerant computing: Fundamental concepts. *IEEE Computer*, 23(7):19–25, Jul 1990.

[11] M. Nicolaidis and B. Courtois. Strongly code disjoint checkers. *transcomp*, 37(6):751–756, Jun 1988.

[12] V.B. Prasad. Fault tolerant digital systems. *IEEE Potentials*, 8(1):17–21, Feb 1989.

[13] N. Seifert, Xiaowei Zhu, and L.W. Massengill. Impact of scaling on soft-error rates in commercial microprocessors. *IEEE Transactions on Nuclear Science*, 49(6):3100–3106, Dec 2002.

[14] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 389–398, 2002.

[15] K. Wilken and J.P. Shen. Continuous signature monitoring: Low-cost concurrent detection of processor control errors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(6):629–641, Jun 1990.

[4]Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).