The Arm Triple Core Lock-Step (TCLS) Processor

XABIER ITURBE, Ikerlan
BALAJI VENU and EMRE OZER, Arm Ltd., UK
JEAN-LUC POUPAT, Airbus Defence & Space, France
GREGOIRE GIMENEZ, Dolphin Integration, France
HANS-ULRICH ZUREK, Atmel Corp., Germany

The Arm Triple Core Lock-Step (TCLS) architecture is the natural evolution of Arm Cortex-R Dual Core Lock-Step (DCLS) processors to increase dependability, predictability, and availability in safety-critical and ultra-reliable applications. TCLS is simple, scalable, and easy to deploy in applications where Arm DCLS processors are widely used (e.g., automotive), as well as in new sectors where the presence of Arm technology is incipient (e.g., enterprise) or almost non-existent (e.g., space). Specifically in space, COTS Arm processors provide optimal power-to-performance, extensibility, evolvability, software availability, and ease of use, especially in comparison with the decades old rad-hard computing solutions that are still in use. This article discusses the fundamentals of an Arm Cortex-R5 based TCLS processor, providing key functioning and implementation details. The article shows that the TCLS architecture keeps the use of rad-hard technology to a minimum, namely, using rad-hard by design standard cell libraries only to protect the critical parts that account for less than 4% of the entire TCLS solution. Moreover, when exposure to radiation is relatively low, such as in terrestrial applications or even satellites operating in Low Earth Orbits (LEO), the system could be implemented entirely using commercial cell libraries, relying on the radiation mitigation methods implemented on the TCLS to cope with sporadic soft errors in its critical parts. The TCLS solution allows thus to significantly reduce chip manufacturing costs and keep pace with advances in low power consumption and high density integration by leveraging commercial semiconductor processes, while matching the reliability levels and improving availability that can be achieved using extremely expensive rad-hard semiconductor processes. Finally, the article describes a TRL4 proof-of-concept TCLS-based System-on-Chip (SoC) that has been prototyped and tested to power the computer on-board an Airbus Defence and Space telecom satellite. When compared to the currently used processor solution by Airbus, the TCLS-based SoC results in a more than 5× performance increase and cuts power consumption by more than half.

CCS Concepts: • **Computer systems organization → System-on-a-chip**; *Embedded systems*; *Reliability*; *Availability*; *Processors and memory architecture*;

Additional Key Words and Phrases: Arm, soft error resilience, space avionics, safety-critical

The research described in this article has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No. 640354. Airbus DS, Arm, Atmel and Dolphin are part of this consortium and are collaborating to bring this innovative technology into the European space industry. Xabier Iturbe was funded by the European Union's FP7 Marie-Curie international outgoing fellowship program with agreement No. 627579.

Authors' addresses: X. Iturbe, Ikerlan, Arizmendiarrieta Pasalekua 2, Arrasate-Mondragon 20500, Basque Country (Spain); email: xiturbe@ikerlan.es; B. Venu and E. Ozer, Arm Ltd., 110 Fulbourn Road, Cambridge CB19NJ, UK; email: {balaji.venu, emre.ozer}@arm.com; J.-L. Poupat, Airbus Defence and Space, 1 Boulevard Jean Moulin, Elancourt 78990, France; email: jean-luc.poupat@airbus.com; G. Gimenez, Dolphin Integration, 1bisA Chemin du Pre Carre, Meylan 38240, France; email: gregoire.gimenez@dolphin.fr; H.-U. Zurek, Atmel Automotive GmbH, Parkring 4, Garching bei München 85748, Germany; email: ulrich.zurek@atmel.com.

0734-2071/2019/06-ART7 $15.00
https://doi.org/10.1145/3323917

ACM Reference format:
Xabier Iturbe, Balaji Venu, Emre Ozer, Jean-Luc Poupat, Gregoire Gimenez, and Hans-Ulrich Zurek. 2019. The Arm Triple Core Lock-Step (TCLS) Processor. *ACM Trans. Comput. Syst.* 36, 3, Article 7 (June 2019), 30 pages. https://doi.org/10.1145/3323917

1 INTRODUCTION

The susceptibility of modern processors to radiation-induced soft errors is dramatically increasing with the exponential growth of transistor count per chip. Small technology nodes use lower operating voltages, which in turn reduce the energy necessary to provoke a voltage pulse at the output of a logic gate, or to invert the value stored in a flip-flop or memory cell (Degalahal et al. 2006). Therefore, lower-energy particle strikes that did not pose any threat in past technology generations can induce soft errors in future generations. To make things worse, the rate of particle strikes increases exponentially as the energy level of the particles decreases (Amort et al. 2011; Johnston 2000), and cumulative radiation can eventually result in permanent damage in the chip, that is, latch-ups or hard errors (Hargrove et al. 1998).

Soft errors have been recognized as one of the most important limits for digital electronic reliability (Normand 1996). They threaten millions of processors operating at the heart of terrestrial safety-critical equipment, ranging from cars to pacemakers, and provoke important economic loses and brand reputation damage when they affect other non-safety-critical equipment, such as space exploration avionics, telecom satellites, and data center servers.

Several solutions have been proposed to mitigate soft errors. These range from using specialized rad-hard process technology (BAE Systems 2008) to architecture and circuit level fault-tolerance techniques, but are dominated by the use of redundant components. Redundant transistors and metal layers are used at the process level (Calin et al. 1996; Lin et al. 2011), flip-flops and Error Correction Codes (ECCs) at the circuit level (Aeroflex Gaisler 2015; Ghahroodi et al. 2013), CPU cores and fault supervision circuits at architecture levels (Arm Ltd. 2011; Infineon Tech. 2012; Kuschel et al. 2010; Maxwell Tech. 2013), as well as software processes and virtualized partitions at the software/hypervisor level (Jeffery and Figueiredo 2012; Resch et al. 2013). However, redundant components at the circuit and process levels limit the achievable CPU performance, namely, the highest usable clock frequency (Ghahroodi et al. 2013; Vorago Tech. 2017).

Currently available space-grade processors are usually based on decades-old designs manufactured with rad-hard process technology, which is very expensive, power hungry, and slow (Aeroflex Gaisler 2015; Atmel Corp. 2004; BAE Systems 2008; Lockheed Martin 1995; Moog Broad Reach 2015). For instance, the RAD750, which is one of the most used processors in ongoing space missions today, including NASA's Curiosity Mars rover, shows five to six orders of magnitude better in radiation tolerance than its commercial PowerPC-750 counterpart, at a reported cost of around $200,000 (in 2002) and with a maximum usable clock frequency of only 200MHz, which limits the maximum achievable performance to only 400 DMIPS (Rhea 2002). This performance limitation is slowing down the adoption of new architectures in space applications, such as ARINC 653, and is preventing the transition from channel-switching to packet-switching telecom satellites (Poupat et al. 2017). In order to deal with this performance crisis, some space agencies and satellite manufacturers have started to consider using COTS processors (ESA/ESTEC 2018a; Iturbe et al. 2016a; NASA 2015; Pignol 2010; Poupat et al. 2017).

Another driving force for using COTS processors in space might well be the "New-Space" companies that look to reduce the traditional prohibitive costs of space applications and open new profitable markets for them (Vernile 2018). Namely, SpaceX, OneWeb, and others have announced a billionaire investment to set up the largest satellite constellations in history to serve as the

backbone for the Internet of Things (IoT) (ESA/ESTEC 2018b). These constellations, which will mainly operate in Low Earth Orbits (LEO), demand for affordable space-qualified processors avoiding as much as possible the use of rad-hard process technology.

On Earth, where exposure to radiation is limited, high-performance and energy-efficient COTS Arm processors are currently used in many terrestrial safety-critical applications conforming to fail-safe standards such as ISO 26262 and IEC 61508. The reliability level required by these standards is achieved using Dual Core Lock-Step (DCLS) architectures (e.g., Arm Cortex-R (Arm Ltd. 2011)), which integrate two twin CPUs that execute the same program simultaneously and compare the outputs at all times to detect errors (i.e., divergences). Upon the detection of an error, the CPUs are either reset or rolled back to a previously saved correct state (i.e., checkpoint) (Gizopoulos et al. 2011). While virtually no errors can escape undetected from the DCLS architecture, a single error can still cause a critical computation deadline to be missed due to the delay introduced by the error recovery process, typically in the range of milliseconds. This is an important limitation for fail-functional applications, such as autonomous vehicles, which need to recover from errors without degrading the system operation (Iturbe et al. 2018; Koopman and Wagner 2016).

This article discusses in detail the Arm Triple Core Lock-Step (TCLS) processor concept, which has already been outlined in Iturbe et al. (2016c), and demonstrates its use in a prototypic TRL4 System-on-Chip (SoC) that has been tested in the laboratory to power an Airbus Defence and Space (D&S) telecom satellite. The TCLS processor adds a third CPU with the dual objective of allowing critical computations to continue executing in the event of individual CPU errors (i.e., using the two remaining functional CPUs), and accelerate the subsequent error recovery process. In fact, the TCLS implements a quick, transparent and reliable mechanism to recover from soft errors, thus relaxing and even removing the need to use rad-hard technology in space applications, and paving the way for building fail-functional sub-systems in terrestrial applications. The proof-of-concept TCLS processor discussed in this article uses Arm Cortex-R5 CPUs (Arm Ltd. 2011) and can recover from individual CPU errors within microseconds. The article also explains a novel high resilience CPU mode to execute high-criticality software routines using the least possible amount of micro-architecture components, thus increasing the resilience of the TCLS processor without impacting the achievable performance.

The remainder of the article is organized as follows. Section 2 introduces related work in fault-tolerant and rad-hard processors. Then Section 3 describes the Arm Cortex-R family of DCLS processors, which is the baseline for the TCLS processor, and discusses the most vulnerable structures within the Cortex-R5 CPU micro-architecture. Afterward, Section 4 explains the TCLS architecture and describes its functioning, putting an emphasis on the error detection and recovery mechanism. Section 5 then describes the TCLS-based SoC that has been prototyped for use in Airbus D&S telecom satellites and provides results for both FPGA and ASIC implementations. Finally, Section 6 concludes the article.

2 RELATED WORK

Space is a harsh environment where electrons, protons, and heavy ions present in galactic cosmic rays and cosmic solar flares as well as trapped in Van Allen belts provoke undesired effects to electronic circuits (Normand 2000). These effects include Total Ionizing Dose (TID), which refers to the accumulation of ionizing dose deposition over time that eventually damages the semiconductor material, and Single Event Effects (SEEs) or soft errors, caused by instantaneous high ionizing dose deposition when a highly energized particle such as a heavy ion strikes a transistor. The circuit type and semiconductor technology as well as the localization and amount of deposed charge by the radiation particle defines if a SEE is triggered and the type of that SEE. The most common type

Table 1. Summary of Rad-Hard Processors and SoCs for Space Missions

Processor	Year	# of Cores	Arch.	SEU/bit-day	DMIPS/MHz	max. DMIPS	mW/MHz	max. W
TSC695	1997	1	SPARCv7	1E-8	0.8	20	40	1
RAD6000	1997	1	PowerPC	1E-10	1.0	35	25	5
RAD750	2001	1	PowerPC	1.6E-10	2.0	400	25	5
AT697	2003	1	SPARCv8	1E-5	0.9	100	10	1
BRE440	2009	1	PowerPC	6.8E-5	2.0	300	50	8
GR712RC	2009	2	SPARCv8	unknown	1.3	280	15	1.5
UT700	2014	1	SPARCv8	5.2E-7	1.2	233	40	4
GR740	2016	4	SPARCv8	unknown	1.8	1,700	24	6
RAD5545	2016	4	PowerPC	2E-9	3.0	5,600	38	17.7

of SEE is the Single Event Upset (SEU). Smaller technology nodes are naturally more resilient to TID, but more sensitive to SEE (Amort et al. 2011).

There are two main radiation hardening methods that are used in integrated circuits destined for space applications: Rad-hard by Process (RHBP) (Dawes et al. 1976) and Rad-Hard by Design (RHBD) (Amort et al. 2011; Lacoe et al. 2000; Scholastique and Hili 2017). RHBP consists in using advanced semiconductor manufacturing processes and materials for mitigating radiation effects, including Silicon on Insulator (SOI) (Romanko and Clegg 2005) and Silicon on Sapphire (SOS) technologies (Schlesier 1974). These special processes require a big economic investment, and do not allow the dense integration scale that can be achieved using the standard and much cheaper commercial semiconductor processes, which are driven by the consumer marketplace. On the other hand, RHBD provides radiation tolerance at the expense of a larger area by using standard cells that include redundant transistors (e.g., Dual-Interlocked Storage Cells—DICE (Berg 2013)), guard rings around p-wells and n-wells (Shaneyfelt et al. 1998), or modifying the geometry of the transistors (e.g., annular, ring, and edgeless transistors (Anelli et al. 1999)). Although RHBD libraries can still be manufactured using commercial semiconductor processes, they are usually combined with RHBP in space-grade circuits to achieve the highest radiation tolerance levels.

Table 1 summarizes the most important processors and SoCs that have been successfully used in space missions so far (Ginosar 2012), as well as the latest generation of space-grade processors and SoCs that have been recently released but not yet debuted in space missions. The space-proven processor list includes BAE RAD6000 (Lockheed Martin 1995), BAE RAD750 (BAE Systems 2008), Atmel TSC695/ERC32 (Atmel Corp. 2004), Atmel AT697/LEON-2FT (Atmel Corp. 2011), Cobham-Aeroflex UT700/LEON-3FT (Aeroflex Gaisler 2015), Broad Reach BRE440 (Moog Broad Reach 2015), and Cobham-Gaisler GR712RC/LEON3-FT (Cobham 2016). Note that all these processors are manufactured using rad-hard process technology and offer very limited computation capabilities compared with non-space COTS devices. In fact, all of them are 32-bit single-core processors, except for the Cobham-Gaisler GR712RC/LEON3-FT dual-core SoC. The latest generation of space-grade SoCs, which have yet to be used in a real mission, include the 32-bit quad-core Cobham-Gaisler GR740/LEON4-FT (Hjorth et al. 2015) and the 64-bit quad-core BAE RAD5545 (BAE Systems 2017). The latter RAD5545 SoC promises unprecedented performance for space applications at the price of a dramatic increase of energy consumption, which will bring major problems for use in small-medium spacecrafts and exploration rovers where the processor power is constrained to 12 W (Doyle et al. 2014). Meeting the power budget in these systems will require reducing the clock frequency or the number of active cores at the same time, in both cases impacting the performance.

On Earth, SEEs are less frequent and hence there is no need for using any rad-hard method at all (Atmel Corp. 2016). Instead, DCLS is the most commonly used architecture to meet the safety

integrity requirements imposed by the various functional safety standards, such as IEC 61508 and ISO 26262. Lock-step has the great advantage of achieving the best error detection with small impact on the usable maximum clock frequency. However, DCLS still results in increased area and power overheads and can only detect errors manifested in the CPU output ports. Internal architectural states (e.g., registers) are not checked directly, only indirectly if they influence an output state.

Lock-step architectures with three or more chips enable the use of proven COTS computing technology in space applications (after completing a qualification process) (Pignol 2010). For example, the NASA Orion spacecraft uses three redundant computers, each consisting of two PowerPC-750FX processors running in lock-step (Whitwam 2014), and the on-board computer in NASA's CALIPSO spacecraft uses four PowerPC-603 processor chips (DeCoursey et al. 2006). However, although the PowerPC-603 processors can run up to 240MHz, they can be clocked only at 160MHz to meet the power constraints in the spacecraft. This illustrates the limitations brought about by outdated processor technology. In Saito et al. (2001), the authors describe the on-board computer of the JAXA INDEX satellite that uses three Hitachi SH-3 COTS microcontrollers and a centralized voter implemented in a space-qualified FPGA. This computer needs as long as 2s to recover from an error. In Hillman et al. (2003), the authors describe a similar computing platform using three PowerPC-750FX microprocessor chips and also a centralized voter implemented on a rad-tolerant FPGA. This computing platform has been commercialized by Maxwell Technologies under the name SCS750, and is able to deliver 1,800 MIPS and recover from an error in about 1ms (Maxwell Tech. 2013). Central in all these approaches is the scrubbing routines that periodically remove errors in the memories and redundant processors. The Computer Space Processor (CSP) designed by the NSF Center for High-performance Reconfigurable Computing (CHREC) relies on a COTS Xilinx Zynq SoCintegrating a 7-Series FPGA fabric and a dual-core Arm Cortex-A9 processor—to perform critical computations, and supervised by rad-hard devices (i.e., reset and watchdog circuits) (Rudolph et al. 2014). The CSP processor will be part of future NASA missions, including the Space test Program-Houston 5-ISS SpaceCube experiment (Wilson et al. 2014) and the Compact Radiation BElt Explorer (CeRES) heliophysics CubeSat (Kanekal et al. 2014). Likewise, GomSpace commercializes a small rugged computing module (NanoMind Z7000) for on-board payload data processing also powered by a Xilinx Zynq SoC (GomSpace A/S 2017).

Arm-based lock-step microcontrollers and SoCs are very popular in terrestrial safety-critical applications thanks to their superior performance and energy efficiency. The TI Hercules family of 32-bit microcontrollers are based on Arm Cortex-R4 and R5 CPUs configured in DCLS and are intended to simplify functional safety certification for IEC 61508 SIL 3 and ISO 26262 ASIL D levels (Texas Instruments 2014). The Infineon Tricore family of 32-bit microcontrollers is based on Arm Cortex-M0 and M4 configured in DCLS and mainly targets automotive and industrial applications (Infineon Tech. 2012). Renesas has recently unveiled the R-Car H3 SoC to pave the way for driverless and autonomous cars. This SoC includes four Arm Cortex-A57 CPUs, four Arm Cortex-A5 CPUs, and two Arm Cortex-R7 CPUs configured in DCLS (Renesas 2015). NXP (now acquired by Qualcomm) offers a wide Arm-based portfolio of microcontrollers for ISO 26262 compliant automotive and IEC 61508 compliant industrial applications (NXP 2012). Following the path started with the Zynq SoC, Xilinx has integrated a dual/quad-core Arm high-performance Cortex-A53 processor, an Arm safety-related Cortex-R5 DCLS processor, and a reconfigurable FPGA fabric in a SoC device (Hansen 2016).

An important additional advantage of Arm processors is the vast software ecosystem and huge user base backing them. For instance, Arm defines and maintains a Cortex Microcontroller Software Interface Standard (CMSIS) to enable consistent device support and simple software interfaces among all Arm-centric microcontrollers developed by silicon partners. CMSIS simplifies

software reuse and reduces the learning curve for microcontroller developers as well as the time to market for new Arm-based devices. This contrasts with the small penetration of currently available space-grade processor architectures (i.e., SPARC and PowerPC), which has impeded the complete development of the space sector. At the dawn of the "NewSpace" era, space applications still remain the domain of a small number of players.

These four features of Arm processors, energy efficiency, high performance, wide software ecosystem, and popularity, have attracted both NASA and ESA, as well as other aerospace and silicon companies to develop new computing solutions using Arm COTS technologies. Namely, NASA has put its attention on Cortex-A CPUs and is working in collaboration with the Goddard Space Flight Center (GSFC), the Air Force Research Laboratory (AFRL), and Boeing on designing a next-generation high-performance space flight dual quad-core processor using Arm Cortex-A53 CPUs (64-bit, 2.3 DMIPS/MHz) (NASA 2015). On the other hand, ESA is trying to move toward more modular, cost-effective and reusable space avionics, where functionalities currently centralized in the main on-board computer would be distributed throughout a number of simpler modules powered by a Cortex-M-based microcontroller (ESA/ESTEC 2018a). Companies like Cobham, Atmel, Texas Instruments, Vorago, and Renesas, are already developing Cortex-M and R-based microcontrollers for space (e.g., Atmel Corp. (2016)), and the Vorago VA10820 Cortex-M0-based microcontroller is already been used in several spacecraft applications, either as a main controller on small satellites or as a watchdog/safety monitor to a more powerful device such as an FPGA (Vorago Tech. 2017). However, most of the solutions described here rely to some extent on RHBP techniques, which impacts the cost, performance, and energy consumption.

The Arm TCLS processor described in this article is an experimental and pre-commercial design created by Arm Research to allow cost-efficient Arm-based computing in space applications. Namely, TCLS implements the redundant CPUs within the same chip to keep the best power-to-performance ratio, and uses exclusively commercial semiconductor processes to allow the design to scale with the shrinking commercial technology nodes and reducing manufacturing costs compared to currently available full rad-hard solutions (i.e., RHBD + RHBP). Furthermore, TCLS decouples the CPUs from memory to ease and speed up the error recovery process, which can be completed within microseconds, and implements a number of innovations to make the whole design more resilient and effective.

3 THE BASELINE: ARM CORTEX-R DCLS PROCESSORS

As introduced above, Arm Cortex-R class embedded processors are widely used in microcontrollers and SoCs for terrestrial safety-critical applications, and are starting to be seen as an alternative to deal with the performance limitations and energy inefficiency of current space-qualified processors (Poupat et al. 2017).

In safety-critical applications, Cortex-R processors are typically used in DCLS configuration, where two identical CPU cores share primary core's TCMs, caches, and peripheral ports. The outputs of the two CPUs are compared at every clock cycle to detect and isolate errors, preventing them from propagating to memories and input/output interfaces. In order to enable detection of common-source faults, such as glitches in the clock tree, the primary CPU runs two clock cycles ahead of the secondary CPU. This time diversity ensures that a common-source fault does not affect the two CPUs in the same way because each CPU is in a different architectural state. Cortex-R processors implement a number of features with a focus on dependable real-time performance (Arm Ltd. 2011), including (1) Tightly Coupled Memories (TCMs) to store instruction and data for bounded real-time response, (2) a micro Snoop Control Unit (uSCU) to maintain data coherency with the data cache when dealing with DMA transactions in real-time streaming applications, and (3) Low-Latency Peripheral Port (LLPP) to carry out fast communications with external peripherals

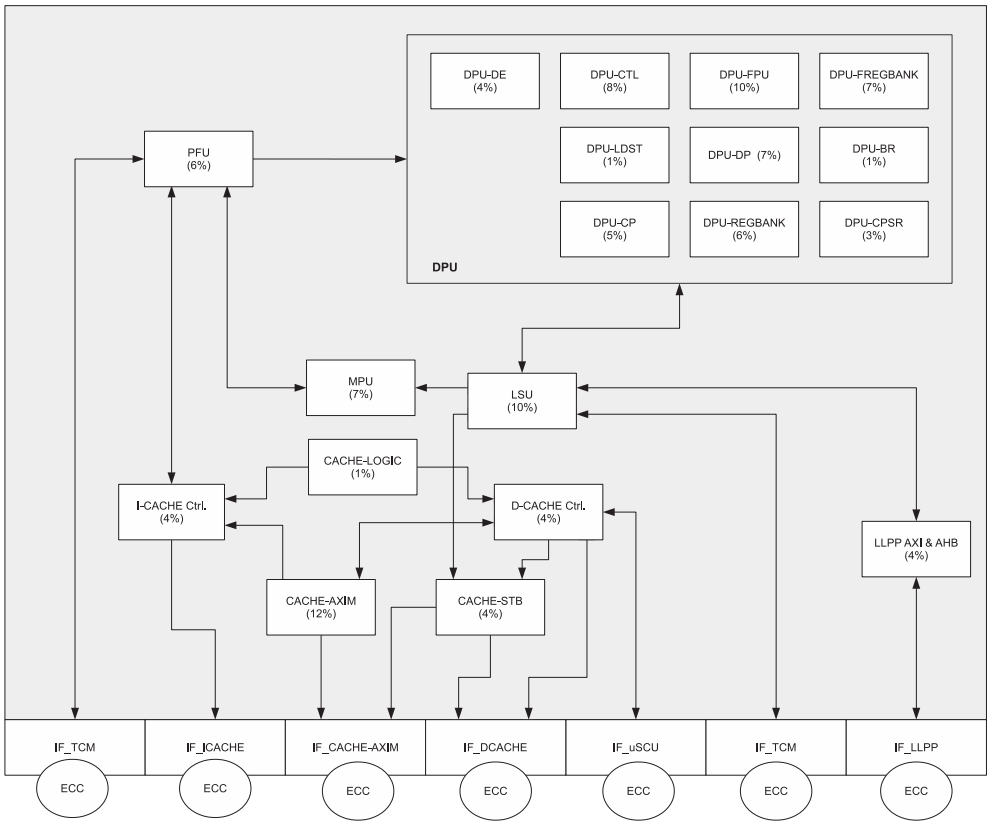


Fig. 1. Arm Cortex-R5 micro-architecture and micro-components.

through AXI and AHB on-chip buses. On-chip busses and memories, including caches and TCMs, are protected with built-in safety features.

3.1 The Cortex-R5 CPU Micro-Architecture

The Cortex-R5 CPU has an eight-stage pipelined in-order dual-issue 32-bit micro-architecture with five execution paths (Arm Ltd. 2011). It implements the ARMv7-R instruction set and also supports Thumb-2 for high code density. Figure 1 depicts the Cortex-R5 micro-architecture and the percentage of the total CPU sequential elements used by each micro-component (i.e., flip-flops and memory cells). In this figure, we use the prefix IF x to refer to the interface that a given micro-component x exposes in the CPU boundaries.

There are 5 CPU micro-components related to the cache memories: (1) the data cache controller (DCACHE), (2) the instruction cache controller (ICACHE), (3) the common logic for both controllers (CACHE-LOGIC), (4) a master AXI bus interface to access the main memory (CACHE-AXIM), and (5) the cache store buffer (CACHE-STB), which includes four 64-bit independent slots where data is buffered prior to being written in the main memory or data cache. Interestingly, the cache store buffer is not used when writing data in TCMs. In this study, we use 16KB instruction and data cache memories. Caches and TCMs are protected against soft errors using parity checking and ECC (i.e., SECDED). A small cache memory is also included to replace up to 64 TCM memory cells when these get permanently damaged.

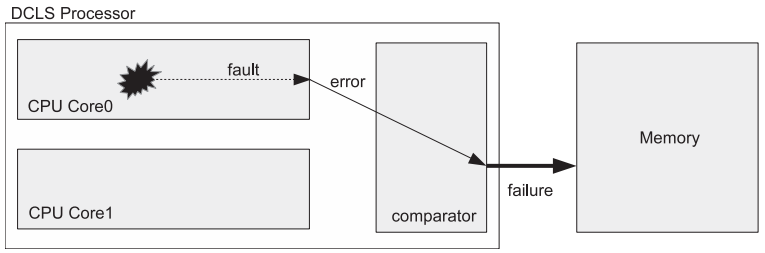


Fig. 2. Fault propagation and dependability terminology in Cortex-R5 DCLS processor.

The Pre-Fetch Unit (PFU) retrieves instructions from the instruction cache, TCM, or main memory and predicts the outcome of branches in the instruction stream to increase performance (e.g., conditionals, loops, and function returns). A pre-decode stage is used to accommodate branch prediction and an instruction queue is implemented to keep the CPU pipeline fed with instructions during branches. The Memory Protection Unit (MPU) captures and aborts non-authorized (i.e., illegal) memory accesses and the Load Store Unit (LSU) manages all load and store operations efficiently.

The Floating Point Unit (FPU) is fully integrated within the Data Path Unit (DPU) pipeline (i.e., not a separate co-processor) and there is a hardware Multiply and Accumulate (MAC) unit and a divider for high-performance calculations. We have broken the DPU down into 10 major micro-components. The (1) DPU-REGBANK and (2) DPU-FREGBANK are the integer and floating-point register files, respectively. The visible registers at a given point of time depend on the processor mode at that time (e.g., user, privileged). The (3) DPU-CPSR includes the Current Program Status Register (CPSR) and five Saved Program Status Registers (SPSRs) to be used by exception handlers. The (4) DPU-BR is responsible for the majority of the PFU interface. The (5) DPU-DE implements most of the DPU logic, including the main instruction decoder logic, the logic to detect dual instruction issue restrictions (e.g., instructions with the same destination), the logic to generate the Program Counter (PC), the instruction queue logic, and the logic to translate the register numbers into physical registers depending on the mode of the processor. The (6) DPU-CP includes the logic for dealing with the co-processor instructions and implements the CP14 and CP15 registers used to configure TCMs, caches, breakpoints, and watch-points. The CP15 registers also hold the configuration related to the MPU, such as the number of memory regions defined (12 in this study) and the base addresses, sizes, and memory types of these regions. The (7) DPU-DP instantiates the main data-path modules involved in execution stages, such as the Arithmetic Logic Unit (ALU), MAC, and divider, and also contains the multiplexors to select data to and from those pipelines. The (8) DPU-FPU contains all the registers, and the data-path structures for the FPU pipeline. The (9) DPU-LDST pipelines the various control signals required for load/store transactions with the LSU. Finally, the (10) DPU-CTL implements all the control signals that are generic to the DPU pipeline, that is, not associated with any specific data-path.

3.2 Vulnerabilities in the Cortex-R5 CPU Micro-Architecture

This subsection studies the effect of soft and hard errors in the Cortex-R5 processor and identifies the most sensitive CPU structures. The study covers error propagation through the CPU micro-architecture and manifestation in the CPU output ports, and extends a previous study that has been published in (Iturbe et al. 2016b).

We use the dependability terminology presented in Avizienis et al. (2004). As shown in Figure 2, in a DCLS processor this terminology can be applied as follows. A fault (e.g., soft error) is active

when it produces an error, which is manifested as a different value in one of the internal CPU's output ports. A failure occurs when an error (e.g., wrong computed data) is propagated outside the processor, either to the I/Os or to the memories. In the Cortex-R5 DCLS processor, failures are prevented by rolling the processor back to a safe state when a mismatch in any of the ports of the two lock-step CPUs is detected.

A fault can remain dormant (i.e., latent), affect an idle component (i.e., a component that contains no valid data), or can be masked at circuit or micro-architecture levels, preventing it from being propagated to the CPU output ports. Circuit level masking occurs when a transient pulse is gated from all possible target sequential elements (e.g., ANDed with a 0), attenuated by subsequent combinational logic levels, or does not arrive within the capture window of the target sequential element. Micro-architecture level masking occurs when the erroneous data is overwritten before use, masked in subsequent logic operations, or simply has no effect in the computation (e.g., dead instructions).

3.2.1 Fault Injection Experiments. Given the lack of specific benchmarks for space applications, we used seven benchmarks from the EEMBC's AutoBench suite that characterize the most common operations in safety-critical automotive applications. These include Controller Area Network (CAN) `canrdr01`, tooth-to-spark `ttsprk01` (i.e., locating the engine's cog when the spark is ignited), road speed calculation `rspeed01`, pulse-width modulation `puwmod01`, and table lookup and interpolation `tblook01`. The remaining two benchmarks deal with matrix mapping operations `matrix01` and Finite Impulse Response (FIR) filters `aifirf01`, which are becoming increasingly important for sensors used in engine knock detection, vehicle stability control, and occupant safety systems. Note that these operations can be considered to be similar to those typically performed on-board a telecom satellite.

We simulated the Cortex-R5 DCLS processor using the Synopsis VCS tool, which allowed us to obtain accurate fault injection results at the RTL level without incurring the high economic costs of irradiation tests, and avoiding the intractable device level simulation time (i.e., electrical simulation). In fact, in order to reduce the simulation time in the fault injection experiments, we ran only 10 iterations of each benchmark, taking between 125,000 and 782,000 clock cycles to be completed. We did not use higher level abstraction models in our experiments (e.g., Gem5) because of the accuracy loss (Kaliorakis et al. 2017). Namely, micro-architecture models do not allow fault propagation delays to be measured with precision of clock cycles as they do not consider non-architectural registers in the CPU pipeline. Secondly, micro-architecture models do not model the CPU output ports and hence they do not allow one to observe manifested errors in the CPU boundaries, which is one of the objectives of our fault-injection experiments. We have used this information to design a more effective error recovery mechanism in the TCLS, as explained in Section 4.2.

We considered the worst-case scenario where all sequential elements in the processor could be affected by errors with the same probability. Hence, we injected faults in all the sequential elements in the primary CPU core and relied on the DCLS comparator logic to detect error situations. *Soft faults* were injected by inverting the value stored in the sequential elements in the CPU for a clock period, while *hard faults* were simulated by forcing the signal values at their opposite levels for the rest of the benchmark execution (i.e., stuck-at faults). If the injected faults did not manifest in the CPU boundaries, the architectural state of the two CPUs (i.e., register files, CPSR, SPSRs, CP14/CP15, and MPU registers) were compared at the end of the benchmark execution to detect latent errors. Therefore, this methodology slightly overrates the vulnerability of architectural elements by considering all latent errors in them; however, not doing the final comparison would result in a vulnerability underestimation as many of the value mismatches in architectural

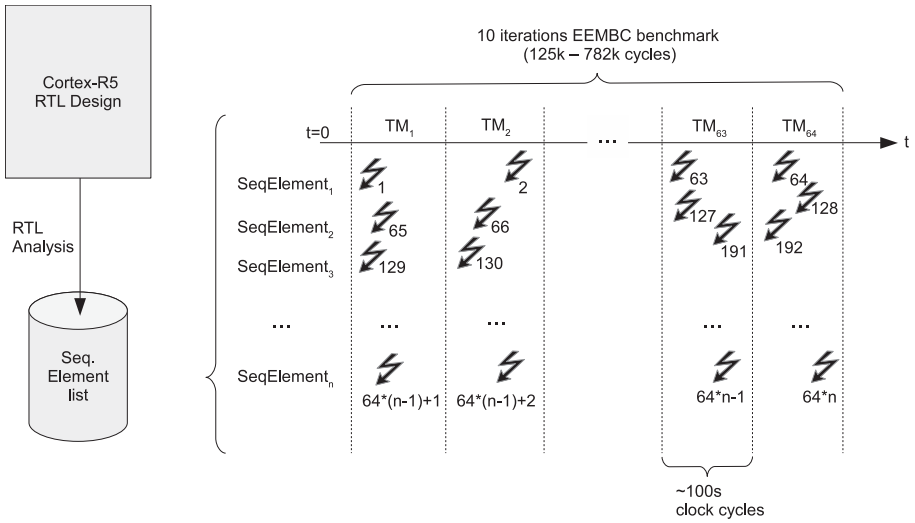


Fig. 3. Fault injection experiments.

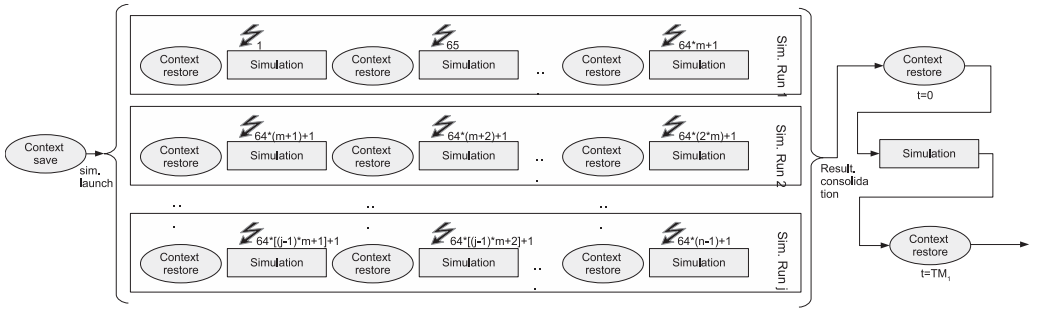
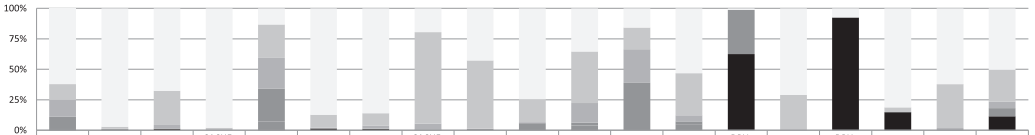


Fig. 4. Fault injection simulation runs.

elements would likely propagate to the CPU output ports and manifest as errors if the simulation continued to run. The vulnerability overrating effect is more apparent when faults are injected in later stages of the program execution as there is less time left for them to be masked. On the other hand, the results are more accurate when errors are injected in early stages of the program execution. For instance, early injected faults in a non-used register can still be masked when the register is later used and an actual value is written to it.

In each fault injection experiment a single fault was injected in a given sequential element at a given instant of time. For the sake of simplicity and aiming at covering the widest behavior of the CPU, the benchmark execution time was divided into 64 equally sized time intervals (TM_x), and for each of these intervals a fault injection instant was randomly selected, as shown in Figure 3. This means that up to 64 different faults were injected in each CPU sequential element at different time intervals, each of them in a different fault injection experiment (shown as a subindex number in Figure 3). On average one fault was injected per few hundred clock cycles, and in total, more than 1 million fault injection experiments were conducted per benchmark.

The fault injection experiments were executed in the Arm Research Cluster at Cambridge, UK. In order to reduce the overall simulation time, incremental simulation was used. As shown in Figure 4, the simulation context was saved at the beginning of each new fault injection interval



	PFU	MPU	LSU	CACHE-LOGIC	CACHE-STB	DCACHE	ICACHE	CACHE-AXIM	DPU-BR	DPU-CPSR	DPU-CTL	DPU-DE	DPU-LDST	DPU-REGBANK	DPU-FPU	DPU-FREGBANK	DPU-CP	DPU-DP	CPU
CL0	640	1034	1044	185	91	528	425	366	94	330	431	94	109	14	1181	80	631	728	8005
CL1	134	16	434	4	186	62	50	1405	123	83	515	106	71	0	475	6	29	422	4121
CL2	142	6	41	0	171	4	12	89	1	3	196	165	10	0	7	0	0	22	869
CL3	95	3	5	0	185	2	1	1	0	0	28	232	5	224	0	0	0	0	781
CL4	16	0	5	0	44	0	1	12	2	26	40	0	10	128	0	0	3	0	287
CL5	2	6	16	0	4	8	5	0	0	0	8	0	0	608	0	1045	114	0	1816

Fig. 5. Soft errors: Number (in table) and percentage (in chart) of critical elements in each Cortex-R5 micro-component.

and then restored multiple times in each subsequent fault injection experiment, thus reducing the span of time to simulate. Likewise, several fault injection experiments were packed together (m in Figure 4) into a smaller number of simulation runs (j) with the dual objective of keeping VCS initialization time to a minimum and exploiting execution parallelism in the Arm cluster. The exact number of fault injection experiments that were packed together in each simulation run was chosen, based on the simulation time of each particular benchmark, with the objective of making the experiments run concurrently for the longest span of time (typically: $50 < m < 100$). We achieved a peak simulation speed greater than 30,000 fault injection experiments per hour.

3.2.2 Soft Error Analysis. Up to 5.6% of the injected soft faults resulted in an error manifested in the CPU output ports. Likewise, up to 14.72% of the fault injection experiments resulted in a latent error affecting the CPU architectural state, but they did not manifest in the CPU output ports.

As expected, the micro-components that provoke most of the errors in the CPU are the register files: DPU-REGBANK and DPU-FREGBANK, which jointly account for at least 75% of the total CPU errors. Note that a fault in these structures directly affects the data being processed, which is a much more straightforward error propagation mechanism than in other CPU micro-components where faults are more likely to affect data-path registers, and hence are more likely to be masked. Moreover, register files are vulnerable to faults for longer periods of time than the micro-components in data-paths, which are only vulnerable while the data-path is active. The most notable example of this is the DPU-FPU, which remains idle when executing integer code, and thus shows the lowest error rate among all CPU micro-components (less than 0.3%). A straight error propagation mechanism similar to that of register files also dominates the CACHE-STB and CACHE-AXIM, which are equipped with data buffers.

Errors also occur when a fault affects a global control register and corrupts the CPU configuration/functioning. This error propagation mechanism dominates the DPU-CP micro-component and can also be seen in a lesser extent in the DPU-CPSR.

Finally, the case of the PFU is very interesting as most of the faults affecting it are benign, resulting in branch mispredictions that only impact the performance, but they are still considered errors in a lock-step processor and reduce the availability of the system.

3.2.3 CPU Micro-Component Soft Error Vulnerability Analysis. Figure 5 categorizes each sequential element in the CPU into five Criticality Levels (CLs). Level 0 means that a fault in that element never provokes an error (i.e., non-critical), whereas level 5 indicates that a fault in that element always results in an error. Levels 1–4 cover different error rates in 25% increments each (CL1: 1–25%; CL2: 25–50%; CL3: 50–75%, and CL4: 75–99%). Each element is classified in the highest CL shown in any of the benchmarks tested, and all functionally equivalent elements (e.g., same data vector register bits) are classified in the same CL, equal to the highest CL among them.

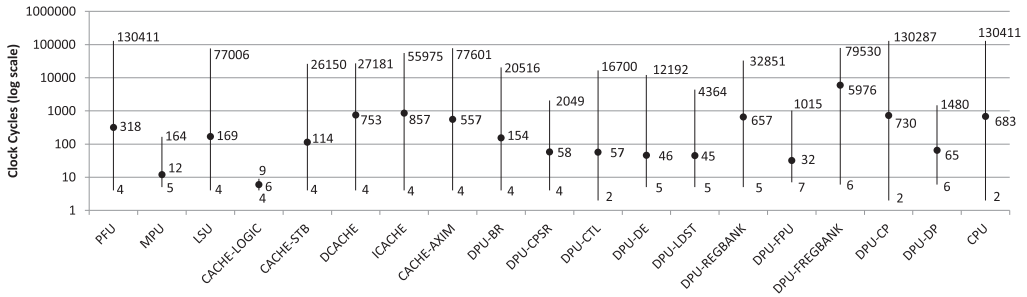


Fig. 6. Error manifestation time (max., avg., and min.).

Note that more than half of the total sequential elements in the CPU never provoke an error (i.e., CL0), and more than 70% of the errors are provoked by around only 2,000 sequential elements. Since this represents less than 15% of the total sequential elements in the Cortex-R5 CPU, we believe there are good opportunities for a notable reliability improvement with small area and power consumption penalties. These results are used in Section 4.3 to propose a high resilience CPU mode that minimizes the number of vulnerable sequential elements used in the processor pipeline when executing high-criticality routines.

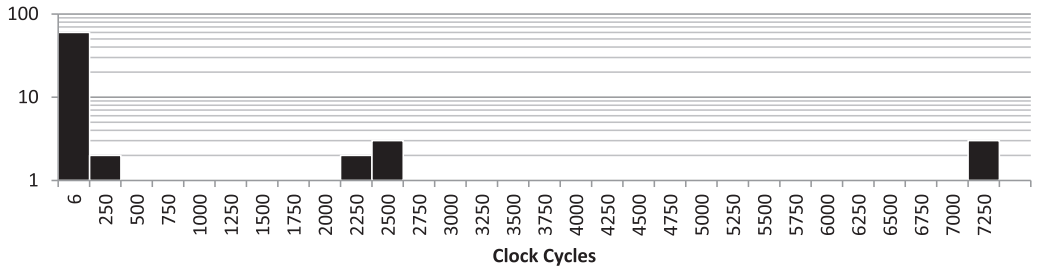
As shown in Figure 5, the register files solely account for more than 90% of the total CL5 elements in the CPU. The floating-point register file contributes with around 24% more CL5 elements than the integer register file, but on the other hand, the integer register file has the greatest combined number of CL4 and CL3 elements and the least number of non-critical CL0 elements among all CPU micro-components. This reflects the fact that integer registers are far more frequently used than floating-point registers (this is why there are very few CL0 elements), but tend to have shorter lifetimes (this is why some faults do not corrupt actual data, leading to fewer CL5 elements). Other micro-components where data remains buffered for a significant amount of time (e.g., CACHE-STB and CACHE-AXIM) also have a very reduced percentage of their elements classified in CL0, while this does not happen in the micro-components that store data for only a few clock cycles, such as the LSU and DPU-LDST.

Both DPU-CTL and especially DPU-DE show a low rate of CL0 elements, which is likely to be the result of the central role they play in executing every single instruction in the CPU. In fact, the most critical structure in the DPU-DE is the instruction queue. However, there are no CL4 nor CL5 elements in the DPU-DE. This can be due to the fact that not all instructions buffered in the instruction queue are finally executed by the CPU because of branches and some of the executed instructions are dead instructions that generate unused results. In the DPU-CTL, the most critical structures are the exception handler logic and the PC pipeline.

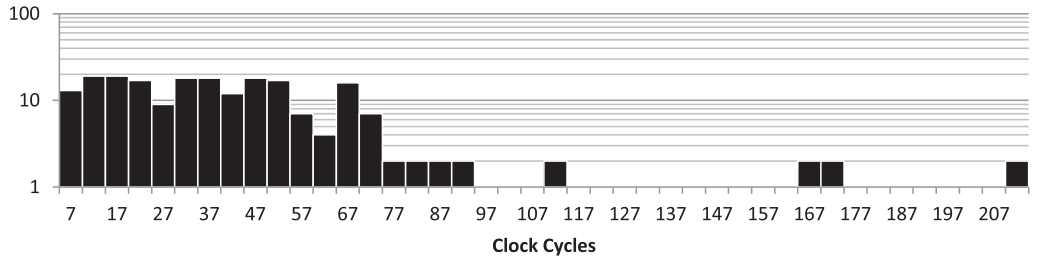
Control registers, such as CP14/CP15 registers in the DPU-CP, tend to be classified in CL5 as they have the potential to affect a large number of instructions, directly or indirectly, eventually resulting in processed data errors. On the other hand, data-path registers, which usually have very short lifetimes, tend to be classified in more relaxed criticality levels. For instance, the DPU-DP and DPU-FPU execution units do not have any element classified in CL3, CL4, and CL5.

The error rate in the PFU is well dispersed along different CLs, reflecting the random accesses to the branch predictor table during the program execution. The latter table and the return-address stack are the most critical structures in this micro-component.

3.2.4 Soft Error Propagation and Latent Fault Analysis. Figure 6 shows the maximum, average, and minimum number of clock cycles that take soft faults injected in each micro-component to propagate to the CPU output ports.



(a) Integer: DPU-DP



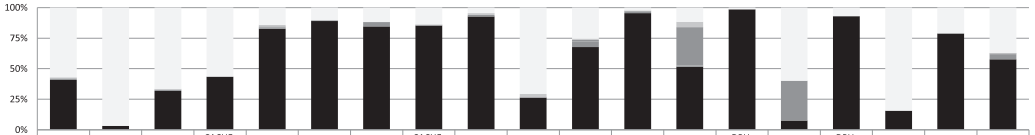
(b) Floating-Point: DPU-FPU

Fig. 7. Error manifestation time histogram in execution units.

Note that all of the CPU micro-components have a minimum error manifestation time equal to or less than seven clock cycles, whereas the average error manifestation time in the whole CPU is 683 clock cycles. Indeed, it is interesting to see the high variability of the error manifestation time across the different CPU micro-components, which can be as high as tens of thousands of clock cycles.

As expected, errors propagate more quickly when they affect processing data, which is the case in LSU, CACHE-STB, DPU-LDST, DPU-DP, and DPU-FPU. Notably, the error manifestation time is shorter in execution units, such as DPU-DP and DPU-FPU, than in data storage units, such as register files. The reason for this is that data in execution units is more likely to be transferred to the CPU outputs (e.g., memory) within a short time, while data stored in register files can remain there for a long time. This is especially the case in floating-point operations. For example, when performing a matrix multiplication, active matrix row and column values (or a subset of them) are typically stored in floating-point registers (Lam et al. 1991). Therefore, the DPU-REGBANK, which is typically involved in more immediate computations, shows significantly shorter error manifestation time than the DPU-FREGBANK. The exceptions are some long-life integer variables, such as indexes in iterative loops, which are stored in the DPU-REGBANK and processed in the DPU-DP. Errors affecting these variables tend to manifest in the CPU ports after thousands of clock cycles, which contrasts with the tens of clock cycles needed by the vast majority of the rest of the errors in the DPU-DP (see Figure 7(a)). Hence, these long-life variables increase the average error manifestation time in the DPU-DP, making it even greater than that in the DPU-FPU. In fact, the latter DPU-FPU micro-component deals only with data variables, and therefore shows a uniform error manifestation time distribution within a short time span in the range of tens of clock cycles (see Figure 7(b)).

Branch prediction related components (i.e., PFU and DPU-BR) show a great dispersion in error manifestation time, with the average in the range of hundreds of clock cycles and the maximum



	PFU	MPU	LSU	CACHE-LOGIC	CACHE-STB	DCACHE	ICACHE	CACHE-AXIM	DPU-BR	DPU-CPSR	DPU-CTL	DPU-DE	DPU-LDST	DPU-REGBANK	DPU-FPU	DPU-FREGBANK	DPU-CP	DPU-DP	CPU
CL0	587	1031	1027	106	97	62	59	257	10	312	313	15	24	14	995	80	655	247	5891
CL1	0	0	7	1	8	1	0	9	3	9	9	0	9	0	0	0	0	0	56
CL2	10	0	7	0	0	0	0	1	0	4	0	0	0	0	0	0	0	0	22
CL3	1	0	1	0	7	2	0	1	0	0	17	5	64	0	0	0	0	2	100
CL4	12	0	10	0	5	1	17	8	3	0	54	7	2	0	543	0	2	0	664
CL5	419	34	493	82	564	538	418	1597	204	117	825	570	106	960	125	1051	120	923	9146

Fig. 8. Hard errors: Number (in table) and percentage (in chart) of critical elements in each Cortex-R5 micro-component.

reaching tens of thousands of clock cycles. This emphasizes the randomness of the error propagation mechanism in these components, which highly depends on the arbitrary accesses to the branch prediction table in the PFU. Finally, the DPU-CP shows one of the longest error manifestation times, which suggests an intricate error propagation path of faults that corrupt the CPU configuration/functioning.

3.2.5 Hard Error Analysis. Up to 46% of the simulated hard faults resulted in errors manifested as deviations at the CPU output ports. On the other hand, the number of latent errors provoked by hard faults was negligible. As a result of a longer interference of stuck-at faults with the normal functioning of the CPU (i.e., a stuck-at fault can be seen as many soft faults injected at many different points of time), there is a polarization of the criticality of CPU elements. As shown in Figure 8, most of these elements are clustered in either CL5 (58%) or CL0 (37%) criticality levels, with the remaining 5% elements falling mostly in the CL4 and CL3 categories. As hard fault injection experiments are a superset of soft fault injection experiments, elements that showed some vulnerability to soft faults are definitely classified in CL5. It is interesting to see that the most vulnerable CPU micro-components to hard errors are not only those that store user or configuration data as occurred in soft error experiments, but also those which implement CPU control logic, such as CACHE-LOGIC, DCACHE, ICACHE, DPU-DE, DPU-BR, and DPU-DP.

4 THE ARM CORTEX-R-BASED TCLS PROCESSOR

As depicted in Figure 9, the TCLS processor integrates three identical Cortex-R5 CPUs running in lock-step and coordinated by a TCLS Assist Unit. The CPUs share a Vectorized Interrupt Controller (VIC), AMBA Network Interconnect (NIC) unit, and ECC-protected TCMs and instruction/data caches. In order to keep engineering and certification costs to a minimum, the Cortex-R5 CPUs are not significatively modified. Only a few little combinational circuitries are added to them to make their pipeline more resilient when executing high-criticality software routines, as explained in Section 4.3.

The instructions to execute by the TCLS processor are read from the shared instruction cache or TCM and distributed to the triplicated CPUs. At every clock cycle, the outputs delivered by the CPUs are majority-voted in the TCLS Assist Unit and forwarded to peripherals, memories, and I/O ports. Simultaneously, the CPU outputs are compared to detect errors. These can be *correctable*, when two of the CPUs deliver the same set of outputs, or uncorrectable, when all of the CPUs deliver a different set of outputs. In the highly unlikely case of *uncorrectable* errors, the TCLS transitions to a fail-safe operation state, in which it might force a global reset of the entire SoC where it is integrated. In the event of correctable errors, the TCLS initiates a CPU resynchronization process to correct the architectural state of the wrong CPU, as described in Section 4.2. The processor can be configured to carry out the CPU resynchronization either immediately or on-demand upon

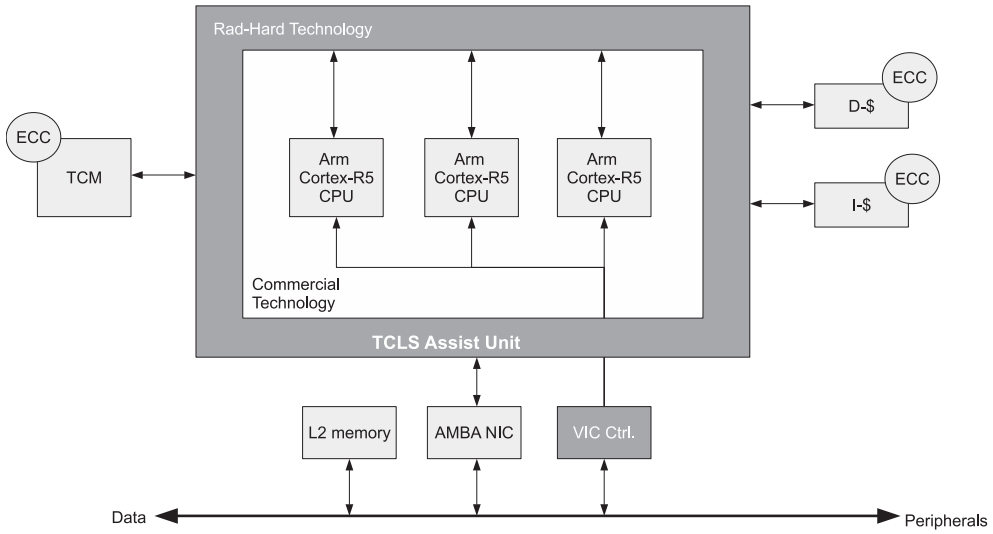


Fig. 9. TCLS architecture overview.

detecting an error. The latter on-demand recovery is driven by software and is useful to prevent the interruption of critical hard real-time tasks and maintain data integrity. Note that the TCLS can still work safely with only two functionally correct CPUs (in DCLS configuration).

In order to prevent the accumulation of latent soft errors, which might eventually result in uncorrectable error situations, the CPUs in the TCLS processor should be periodically scrubbed using the same CPU resynchronization procedure triggered in correctable error situations (i.e., preventive scrubbing). In fact, this is a recommended technique by the ISO 26262 safety standard to improve the Latent Fault Metric (LFM) in automotive applications. The CPU scrubbing interval can be configured by software.

The TCLS in conjunction with the ECC-protected TCM, where the error recovery routines are held, enable the creation of a *Reliable Root Node* that can be used to execute error detection and recovery routines (e.g., BIST routines) to protect other components in the SoC (e.g., peripherals and memories).

4.1 TCLS Assist Unit

Figure 10 shows the four main parts in the TCLS Assist Unit: (1) Majority voter, (2) Error detection logic, (3) CPU resynchronization logic, and (4) Control registers. As shown in the figure, the TCLS Assist Unit can also be configured to include support for injecting errors in selected CPU outputs, which is very useful to carry out chip manufacturing tests and real-time performance tests in error scenarios.

The most vulnerable parts, such as the CPU resynchronization logic and control registers, are protected against soft errors as described in Section 4.1.2. This is recommended by the ISO 26262 standard to improve the Single Point Fault Metric (SPFM) in terrestrial automotive applications. For space use, the TCLS Assist Unit is to be implemented using RHBD libraries, as described in Section 5.1. Since this unit represents only a fraction of the total TCLS processor, cost, performance, and power consumption overheads can still be kept to a minimum.

4.1.1 Majority Voter and Error Detection Logic. The *majority voter* acts as an error propagation boundary, masking correctable errors and preventing them from propagating to the caches and

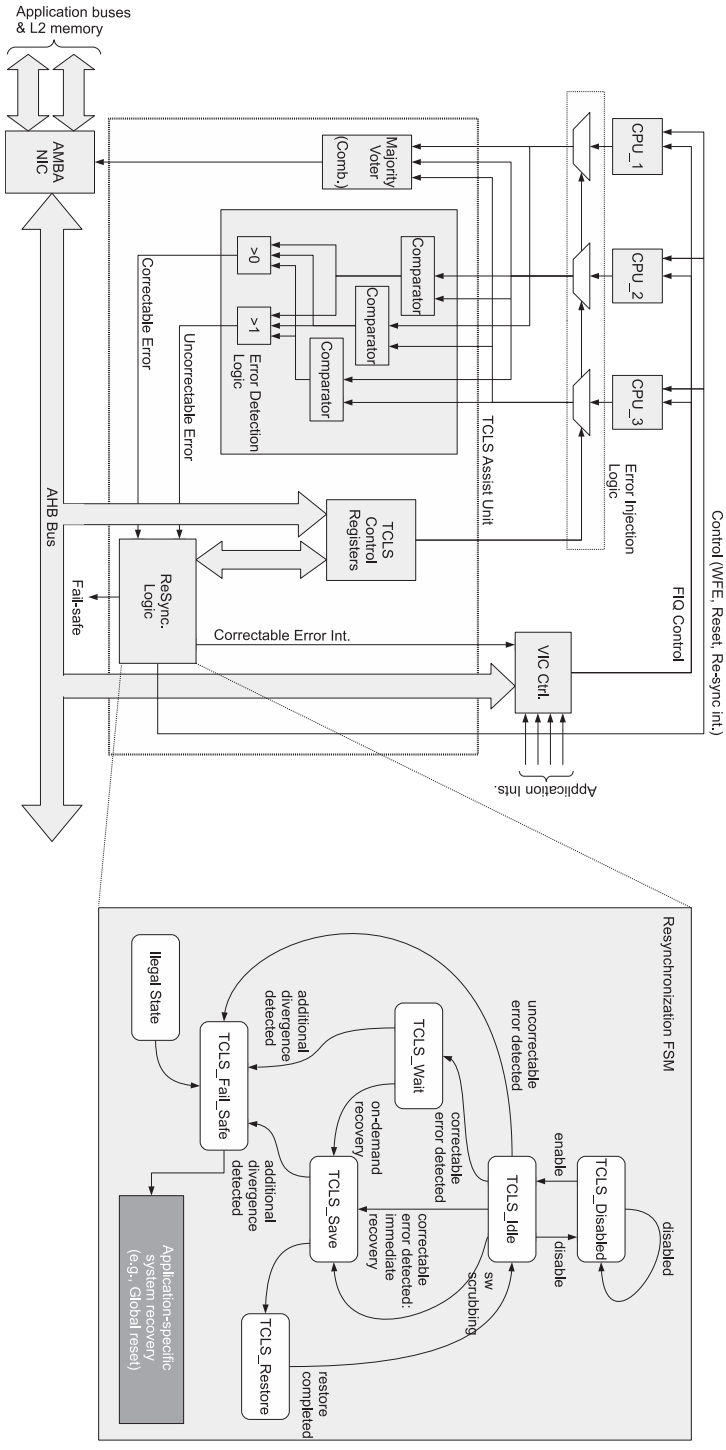


Fig. 10. Block diagram of the TCLS assist unit.

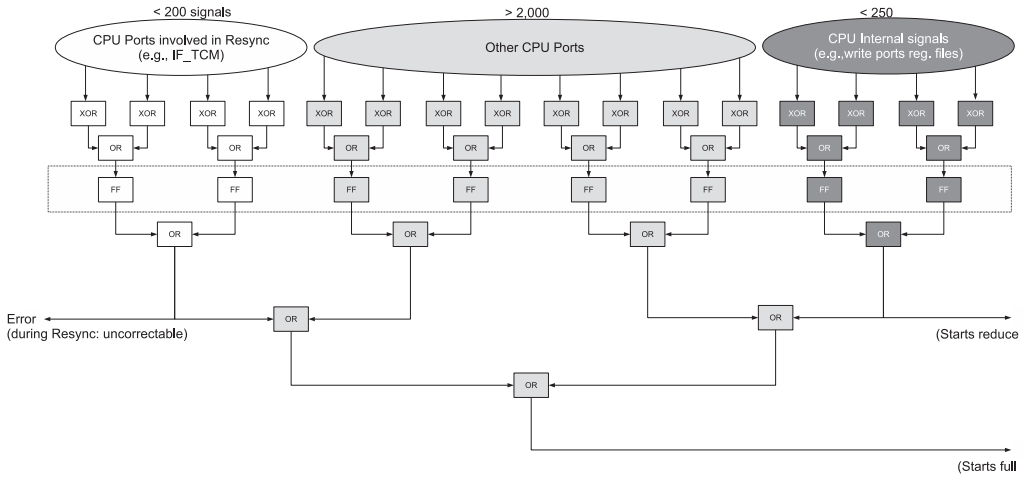


Fig. 11. Internal structure of the comparators in the TCLS error detection logic.

TCMs. This reduces the error recovery time as the memory state does not need to be restored. All CPU output ports are majority voted in TCLS.

The *error detection logic* identifies correctable and uncorrectable errors within the CPUs. In addition to the CPU output ports, a reduced number of internal CPU signals are also monitored to allow detection of errors before they propagate to other CPU parts. Upon detecting an error in a CPU internal signal, specific recovery actions for the affected CPU micro-component are started. When the on-demand error recovery mode is enabled, the erroneous CPU is immediately stopped (i.e., clock signal disabled) to prevent error propagation within it until starting the recovery actions. This is much quicker than waiting for the error to propagate to the CPU output ports and then correcting all architectural registers. The internal CPU signals to expose to the error detection logic are carefully chosen to increase the error detection coverage.

Register files, which account for more than 75% of the total errors in Cortex-R5 CPUs (see Section 3.2), offer an excellent opportunity to detect errors as most of the instructions executed by the CPUs involve accesses to them, either to read operands or to write results computed by the execution units. Likewise, the cache store buffer interface allows for detecting errors in execution units that corrupt data to be written directly to memory (i.e., without being previously stored in register files). Hence, the internal CPU signals that are exposed to the error detection logic are (1) the two 32-bit write ports in the integer and floating-point register files, and (2) the 64-bit data and 32-bit address ports in the cache store buffer interface. In total, we expose less than 250 internal CPU signals, which represents around a 9% increase in the total number of monitored signals.

As shown in Figure 11, there are three different groups of signals that are monitored by the TCLS error detection logic: (1) the CPU output ports, (2) the internal CPU signals discussed above, and (3) the CPU output ports in the TCM interface. These are less than 200 ports in the Cortex-R5 and play a major role in the CPU resynchronization process. In fact, these are the only ports that are monitored while conducting a CPU resynchronization in order to avoid benign errors that might be provoked by other CPU micro-components that are not exercised at these times (e.g., CACHE-AXIM).

The comparators in the error detection logic process each group of monitored signals in separate XOR chains, as shown in Figure 11. Each chain results in an error signal that is flagged when there is a mismatch in its input signals: (1) “Full_Resync” indicates errors in the standard CPU output ports,

(2) "Reduced_Resync" indicates errors in the CPU internal signals, and (3) "Uncorrectable" indicates uncorrectable errors in the TCM ports when carrying out a CPU resynchronization. Although not shown in Figure 11, the comparators also indicate the group of internal CPU signals where the error was detected (i.e., write ports in integer/floating-point register files or cache store buffer interface). Full_Resync and Reduced_Resync signals generated by the comparators are combined to detect correctable and uncorrectable error situations (see Figure 10). In correctable error situations, each of these signals triggers different error recovery actions in TCLS, as explained in Section 4.2.

Both error detection logic and majority voter are essentially combinational, but while the bitwise majority voter involves very few logic levels, the error detection logic involves vector-wise comparison of the thousands of CPU output ports and hence needs to be pipelined as shown in Figure 11. Since errors are reasonably rare, it is extremely unlikely that a fault affects a pipelining register during one of the particular clock cycles when it is active. Hence, most of the faults that affect these registers will result in false-positive detected errors.

4.1.2 CPU Resynchronization Logic and TCLS Control Registers. As shown in Figure 10, the *CPU resynchronization logic* is built around a six-state Finite State Machine (FSM) that interacts with the CPUs during the error recovery process. The FSM is implemented with the capability to jump to a defined recovery state if an illegal state is reached due to a soft error. This is achieved by encoding each FSM state using one-hot codes and directing the undefined states with multiple "1"s to the TCLS Fail Safe recovery state.

The functioning of the FSM is directed by the configuration stored in the *TCLS control registers*, which are accessible by software through a memory-mapped AHB slave peripheral interface. These registers are used, for instance, to enable/disable the TCLS configuration, configure the error recovery mechanism to be immediate or on-demand, or keep status information including the erroneous CPU id and the number of errors detected. Data integrity in the TCLS registers is protected with a technique that combines one-hot encoding and parity bits (Venu et al. 2016). Namely, the error protection bits are interleaved with data bits in the registers to maximize error detection coverage and an FSM-centric logic continuously checks these bits to detect upsets, which are automatically corrected within a few clock cycles. The latter FSM also uses one-hot encoding to cope with illegal state transitions provoked by soft errors.

4.2 TCLS Error Recovery Process: CPU Resynchronization

The CPU resynchronization process in the TCLS processor consists of two assembly routines: SAVE and RESTORE. The code of these two routines, as well as the data structures involved in the CPU resynchronization process, are stored in the ECC-protected TCM. This reduces the amount of CPU hardware and ports exercised while carrying out the resynchronization process to avoid unnecessary uncorrectable errors. As shown in Figure 1, TCMs are directly accessed from the LSU micro-component, circumventing the use of the cache store buffer in the CACHE-STB micro-component and the cache access interface, which comprises significantly more ports than the interface of the TCM.

SAVE is the Interruption Servicing Routine (ISR) to handle the fast interruption (FIQ) issued by the TCLS Assist Unit upon detection of a correctable error. When servicing this ISR, the CPU pipeline is automatically flushed and the PC saved. As discussed in Section 4.1.1, the action taken when executing SAVE ISR depends on the error signal generated by the comparators in the error detection logic (i.e., Full Resync or Reduced Resync).

If the error is manifested in the standard CPU output ports, the SAVE ISR pushes out the entire architectural state of the three CPUs, that is, register files and configuration registers. In Cortex-R5 CPUs this includes pushing out the value of 113 registers, which are passed through the majority

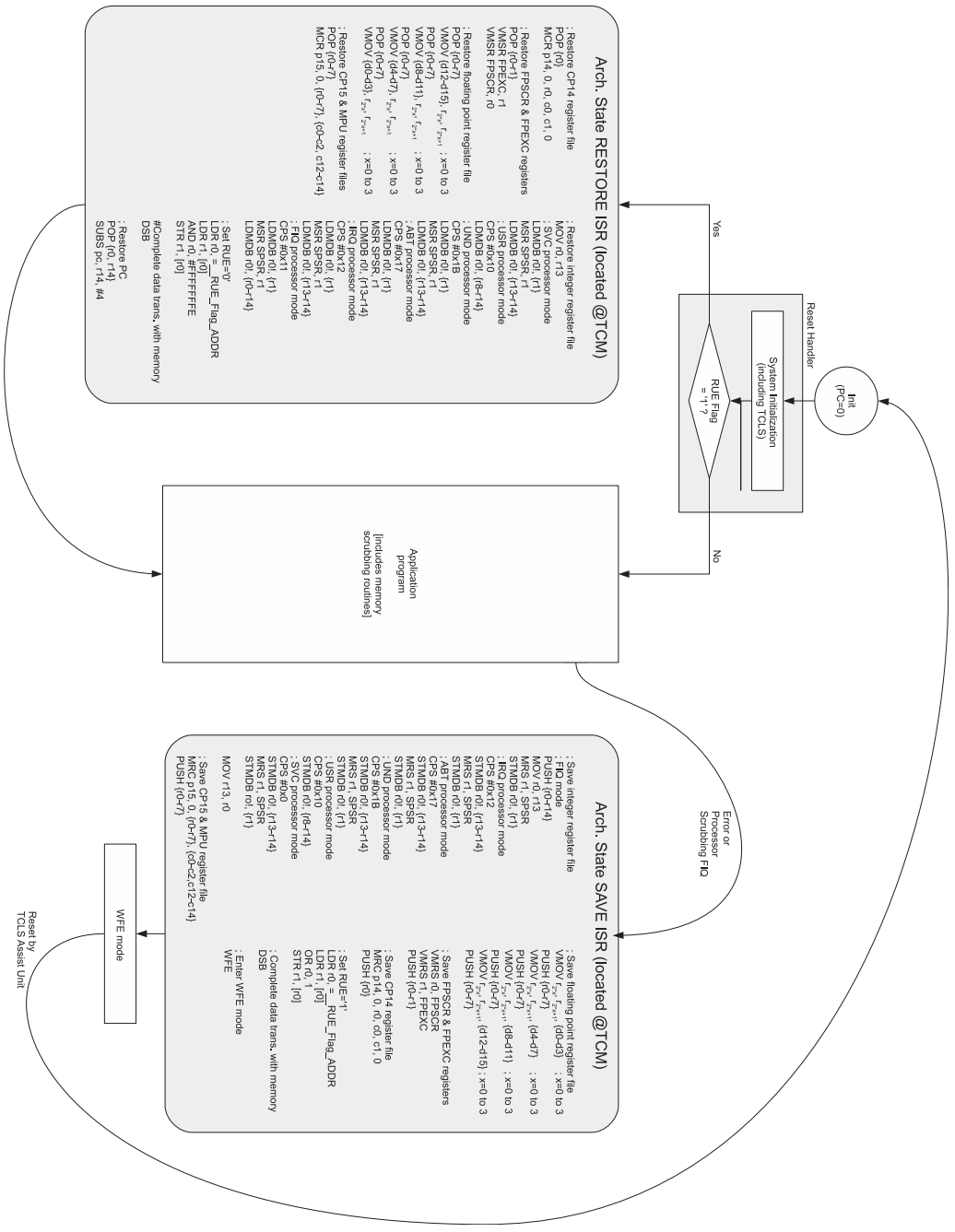


Fig. 12. *Full CPU Resynchronization* when error is detected in CPU output ports.

voter in the TCLS Assist Unit and stored in a Program Stack mapped to the ECC-protected TCM. We name this recovery process as full CPU resynchronization and is shown in Figure 12.

If the error is detected in the CPU internal signals, the SAVE ISR pushes only the affected set of architectural registers. More specifically, if the error is observed in the write ports of register

files, then only 30 integer registers or 32 floating-point registers are pushed out, majority voted, and stored in the Program Stack mapped to the ECC-protected TCM. If the error is observed in the cache store buffer interface, no register is pushed out, but memory barriers are executed to force incorrect data in the cache store buffer to be immediately drained to memory and not be reused by the instructions in the CPU pipeline. In the latter case, the error detection logic ignores the subsequent divergence observed in the CPU's output write data ports. This shorter recovery process helps increase the amount of time CPUs are performing active computation, thus improving the system availability. We name it as selective CPU resynchronization.

If an additional divergence of a CPU other than the one that triggered the resynchronization process is detected during the architectural state dump, the CPU resynchronization FSM in the TCLS Assist Unit transitions to the fail-safe state as the system is considered to be not trustable (i.e., uncorrectable error is assumed). This situation is indicated with the Uncorrectable error signal generated by the comparators in the error detection logic.

In a full CPU resynchronization, the CPUs enter the Wait for Event (WFE) low-power standby mode after executing the SAVE ISR. Simultaneously, the CPU resynchronization FSM activates an internal Resynchronization Under Execution (RUE) flag in the TCLS control registers. When the CPU resynchronization FSM observes the WFE signal, it issues a reset to the three CPUs to purge any soft error that might exist in their micro-architecture and wakes them up from the standby mode. When starting up, the CPUs check the RUE flag to distinguish between a normal reset and a reset provoked by a CPU resynchronization process. In the latter case, the CPUs branch to the RESTORE routine, which restores the previously saved architectural state from the TCM. Note that the last register to be restored is the PC, when returning from the routine, thus resuming the normal operation of the CPUs at the same point in the code where they were suspended to launch the CPU resynchronization process.

In a selective CPU resynchronization, the CPUs jump to the RESTORE routine directly from the SAVE ISR. Unlike in the full resynchronization, the CPU resynchronization FSM does not issue a global reset to the CPUs to preserve the content of their micro-architecture registers, which are believed to be correct. The RESTORE routine restores the required architectural registers back from the TCM. As in the full CPU resynchronization, the last register to be restored is the PC, thus resuming the normal operation of the CPUs. We have checked that most CPU errors are corrected by performing a selective CPU resynchronization. The only errors that are not corrected are those that affect the CPU configuration (e.g., CP14/CP15 and MPU registers), which occur in less than 5% of the error scenarios discussed in Section 3.2. To deal with these cases, a full CPU resynchronization (including a CPU reset) is carried out if an error is detected a short time after completing a selective CPU resynchronization.

Permanent hard errors in the CPUs may result in a live-lock situation where the TCLS Assist Unit forces continuous resynchronization operations. In order to deal with these locks, the TCLS Assist Unit implements a software-writable control register where one can specify the maximum number of CPU resynchronization attempts to be done within a certain period of time before considering that the CPU is affected by a permanent error. In this situation, the TCLS switches to DCLS mode and signals the need for replacement (useful in terrestrial applications).

Unlike in DCLS, where the error recovery is application-dependent (e.g., checkpoint-rollback), the recovery process in TCLS is automatic and transparent to the software. A full CPU resynchronization is completed within only 2,351 clock cycles (SAVE routine takes 1,171 clock cycles and RESTORE takes 1,180), and a selective CPU resynchronization needs less than 400 clock cycles. Hence, when running at 300MHz, a full CPU resynchronization takes less than 8μs, which is a great improvement compared to related solutions for space that need as much as 1ms (Maxwell Tech. 2013). In automotive applications, the quick, predictable, and transparent error recovery

mechanism available in TCLS could be used to build fail-functional Electronic Control Units (ECUs) (Iturbe et al. 2018).

4.3 *High-Resilience* CPU Mode for Executing High-Criticality Routines

Section 3.2 has identified the most vulnerable micro-components in the Cortex-R5 CPU. Namely, Figure 5 shows that DPU-DE, CACHE-STB, and PFU have a large number of sequential elements classified in high criticality levels (CL5 to CL3), which is only surpassed by that in the micro-components that include architectural structures (i.e., DPU-REGBANK, DPU-FREGBANK, and DPU-CP).

This section proposes a novel high resilience CPU execution mode that uses the bare minimum hardware in the critical DPU-DE, CACHE-STB, and PFU micro-components to increase error resilience of the Cortex-R5 pipeline when executing high-criticality routines, such as the CPU resynchronization in TCLS. This mode can also be used when carrying out check-points in a DCLS processor and memory scrubbing in any type of processor. This execution mode, however, does not protect register files and other architectural registers, such as CP14/CP15 and MPU registers. The high resilience mode is based on two principles: (1) keep the instructions and data in the ECC-protected TCM for the longest possible time without impacting the performance, and (2) disable or by-pass all structures in the CPU pipeline that are not strictly necessary for executing the high-criticality routines (e.g., performance-oriented CPU features).

When the CPU enters the high-resiliency mode, the instruction fetch rate is reduced to minimize the amount of time highly critical instructions remain unprotected in the CPU internal buffers (i.e., instruction queue, fetch, decode, and issue buffers). Due to the nature of the TCLS SAVE routine, where LD instructions are followed by ST and PUSH instructions, the performance penalty provoked by the reduced instruction fetch rate is only a few clock cycles. This is because the performance achievable by LD and ST operations is largely bound by the data-side throughput, and PUSH instructions generate plenty of data-side bandwidth whilst consuming little instruction-side bandwidth. Likewise, the TCLS SAVE routine contains few branches, and therefore significant buffering of instructions is not needed to keep the CPU pipeline full. Similar results are expected when using the high-resilience mode in check-pointing and memory scrubbing routines, which also follow the same LD-PUSH/ST instruction structure.

As shown in Figure 13, the Cortex-R5 hardware disabled in the high-resilience mode includes most of the slots in the instruction and issue queues in the DPU-DE (i.e., only two slots are made available), hardware debugging support (e.g., break-point registers), and branch prediction-related hardware in the PFU (e.g., branch prediction table). As previously discussed, it is advisable to only use TCMs for both data and instructions executed in the high-resilience mode, thus circumventing the cache store buffer as well. Note that larger CPUs are more likely to have a greater amount of logic that can be avoided. For instance, the dispatch queue size in an out-of-order CPU could also be reduced.

The hardware to be added in the CPU micro-architecture to support the high-resilience mode is extremely light, resulting in almost zero area and power consumption overheads. Namely, a set of multiplexers is added to drive the fetch rate control signal in the PFU micro-component and the instruction/issue queue signals in the DPU-DE micro-component. These multiplexers, as well as the debug enable signal, are commanded by a software-writable control bit in one of the CPU configuration registers in the DPU-CP micro-component. Note that the Cortex-R5 CPU already includes hardware support for enabling and disabling the branch predictor feature from software.

We have conducted a series of fault injection experiments to evaluate the resilience improvement brought about by the high-resilience mode. Namely, we have compared the error rate when injecting soft faults while the processor executes the TCLS SAVE routine without enabling this

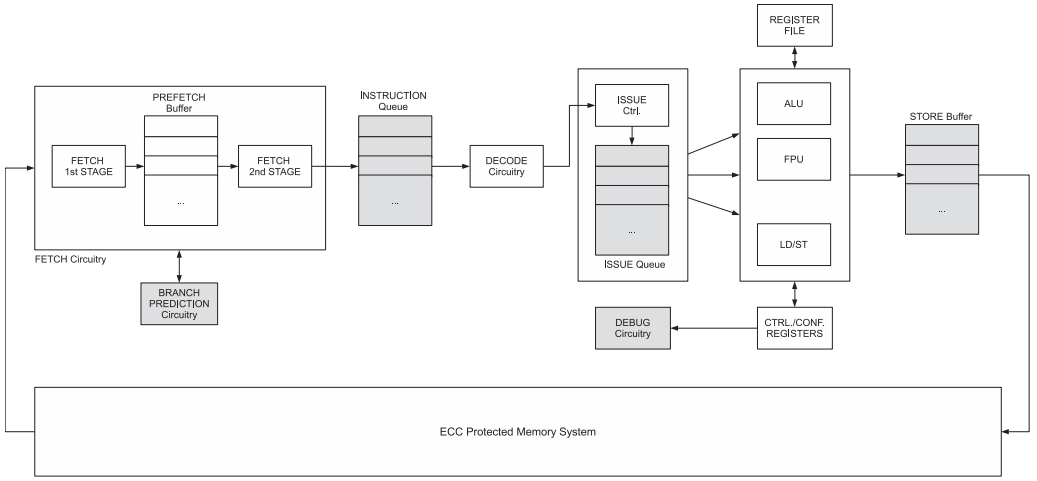


Fig. 13. Disabled structures in the CPU pipeline in the high-resilience mode (in gray).

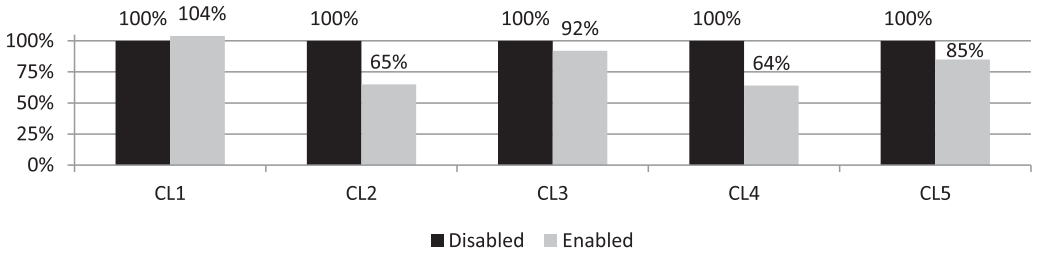


Fig. 14. Reduction in percentage of critical elements in the Cortex-R5 pipeline when enabling the high-resilience CPU mode.

mode (baseline) and when it is enabled. The fault injection conditions are the same as described in Section 3.2.1. The obtained results are shown in Figure 14 and confirm that the overall error rate is reduced by 11%. The major improvement is in the decoder, which shows only a 9% error rate when the high-resilience mode is enabled vs. 25% when this mode is disabled. As shown in Figure 14, the number of high-criticality elements in the CPU pipeline has also been significantly reduced when enabling the high-resilience mode. In fact, there is a migration from high criticality levels (CL2 to CL5) to low criticality levels, resulting in a slight increase in the number of CL1 elements. It is especially interesting to see the reduction of 15% in the number of CL5 elements.

4.4 Reliability Assessment of the TCLS Architecture

In order to test the improvement in reliability brought about by the TCLS solution, we repeated the soft error injection experiments described in Section 3, and compared the results with those in the single-core Cortex-R5 CPU. Besides injecting soft faults in each sequential element of one of the three CPUs in the TCLS architecture, we also injected soft faults in the TCLS Assist Unit. Our experiments showed that none of the faults injected in the CPU lead to a system error, and only around 2% of the injected faults in the TCLS Assist Unit resulted in incorrect functional behavior of the TCLS and eventually provoke a transition to the fail-safe state. This behavior proved that the techniques implemented to detect faults in the TCLS Assist Unit are effective. The number of elements in the TCLS solution that were found to be critical was limited to only 115 flip-flops,

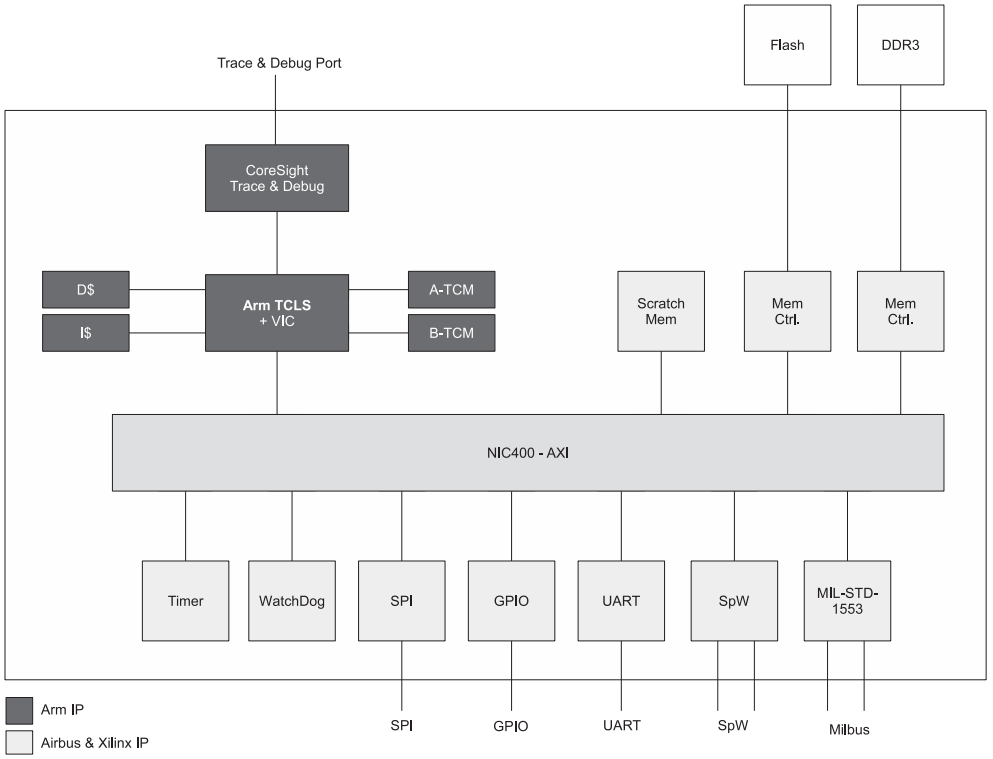


Fig. 15. General architecture of the TCLS-based SoC.

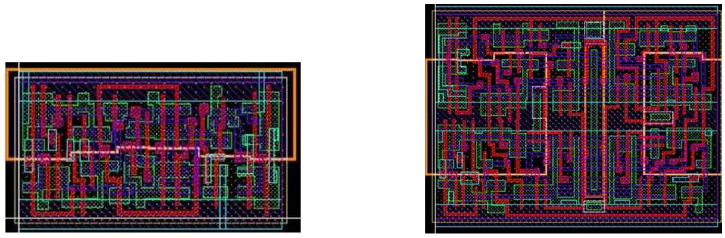
Fig. 16. Memory map of the TCLS-based SoC.

which is a considerable reduction compared to the more than 7,800 potentially vulnerable elements identified in the single-core Cortex-R5 CPU, as shown in Figure 5. However, we have to note here that these results can only be considered to be orientative as real radiation conditions might well provoke multiple simultaneous bit upsets or other effects that have not been simulated in our fault injections experiments, such as transients in the combinational logic.

5 TCLS-BASED SOC PROTOTYPE FOR TELECOM SATELLITE CASE STUDY

The Arm TCLS processor described in this article was used in a prototypic SoC computer for an Airbus D&S telecom satellite. This SoC was implemented on a Xilinx Virtex-7 XC7VX485T FPGA and tested in a laboratory environment.

Figures 15 and 16 show the general architecture and memory map of the prototyped SoC, respectively. The SoC is provided with 128Mb flash and 1GB DDR3 memories. The TCLS is coupled with 512kB on-chip scratchpad memory, which acts as a local-store memory, 64kB ECC-protected



(a) C65 flip-flop cell (b) S65 flip-flop cell

Fig. 17. ST Commercial C65 vs RHBD S65 libraries.

data/instruction cache memories, and two TCMs to achieve hard real-time performance (TCM-A: 256kB and TCM-B: 512kB). The SoC includes standard peripherals, such as a timer, watch-dog, SPI, and UART, as well as other peripherals typically used in space applications, such as SpaceWire (SpW) and MIL-STD-1553 (Milbus) buses to interact with off-chip avionics modules. All peripherals are interconnected through an Arm NIC-400. The prototyped SoC works at 100MHz and delivers approximately 160 DMIPS performance, consuming around 44,000 Slice LUTs (58% of the available LUTs in the FPGA), 245 I/Os (35% of the available pins), and uses about 17Mb on-chip memory (46% of the available Block-RAMs).

A test software was developed to check that all blocks in the SoC are functioning correctly at ambient temperature in standard laboratory conditions. This software includes classical benchmark test sequences dedicated to test the ability of the system to handle hard real-time operations and routines typically executed in telecom satellite applications, such as telemetry and telecommand handling.

The error recovery mechanism of TCLS was successfully tested by injecting faults in the CPU output ports using the fault injection logic implemented in the TCLS Assist Unit (see Section 4.1) while the system was executing the aforementioned telecom satellite routines. We conducted a statistical analysis on the availability improvement that could be achieved by TCLS in a representative 10-year telecom satellite mission operating in LEO orbit with a solar activity of 8 solar flare days, which is a usual case for estimations. This analysis showed that the quick error recovery in TCLS can reduce the system downtime by a 1,000× factor compared with a DCLS solution, where a global reset is always required to recover from errors. Availability is in fact a central requirement in most space missions, especially in telecom satellites with huge direct costs per downtime hour and important indirect economic losses due to reputation damage.

5.1 Toward a Radiation Immune ASIC Implementation

We implemented different versions of the SoC described in the previous section (without on-chip scratchpad memory) to compare the Power, Performance, and Area (PPA) numbers. Two STMicroelectronics' 65nm sets of libraries were used in these implementations: (1) C65 library contains commercial non-rad-hard elements only, whereas (2) C65-SPACE RHBD library (further shortened into S65), which has been derived from the former commercial library, using a modified design of the standard cells to improve the overall soft error rate in space radiation environments (Scholastique and Hili 2017). The difference in design, and hence size, of a flip-flop cell in the two libraries can be seen in Figure 17. Namely, S65 library uses bigger geometries and adds a redundancy scheme to flip-flops to mitigate soft errors, as well as C-elements to the clock tree buffers to filter glitches. To compensate for the timing degradation due to radiation hardening, the voltage threshold in the S65 library is lowered compared to that in C65.

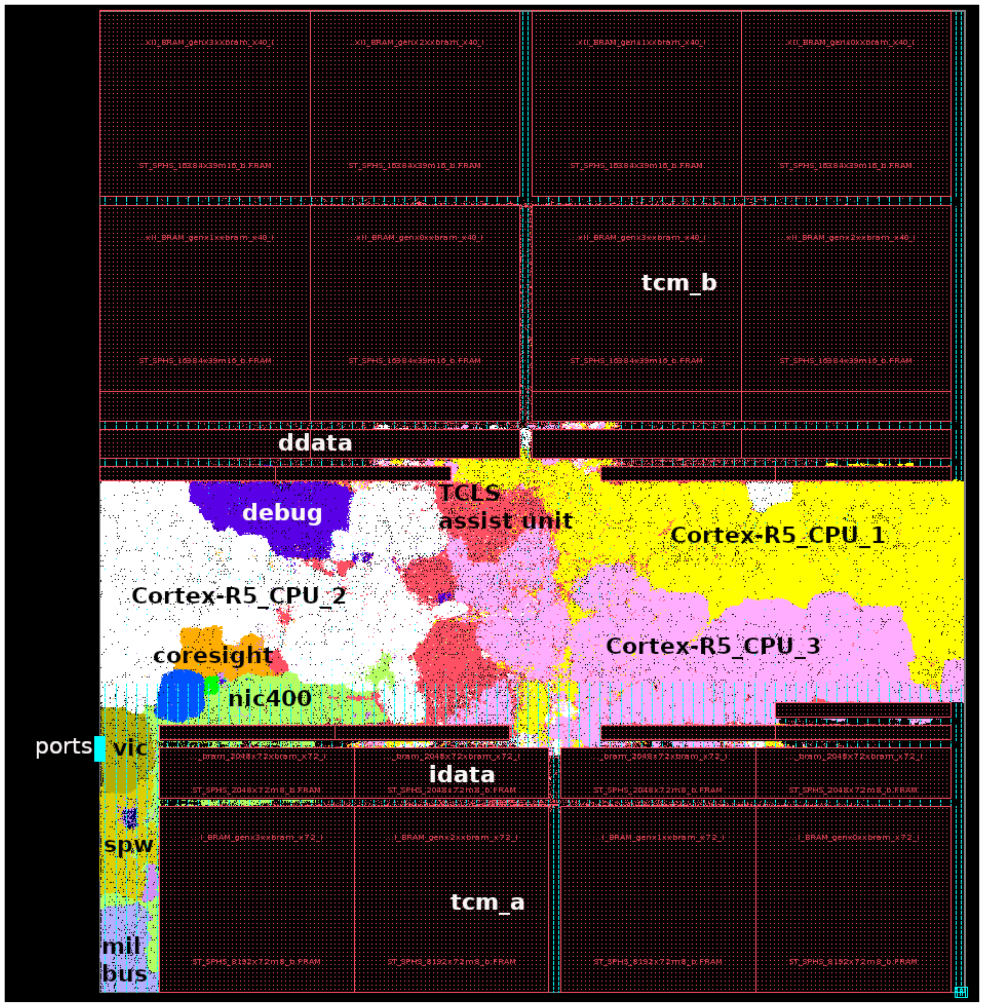


Fig. 18. Floorplan of the TCLS-based SoC using commercial C65 standard cell libraries.

A first baseline implementation is not powered by TCLS, but by a single-core COTS Cortex-R5 CPU and uses exclusively S65 libraries. In this implementation, the cache memories and TCMs represent approximately 85% of the circuit area and the Cortex-R5 CPU core accounts for more than 50% of the logic area. The second implementation uses exclusively C65 libraries to implement TCLS. The third implementation uses S65 libraries in the critical TCLS Assist Unit and C65 libraries in the rest of the SoC components, including the Cortex-R5 CPUs, peripherals, and TCMs. Note that since the hardening is done at the level of cell libraries, i.e., RHBD, both rad-hard and commercial libraries can be combined in the same design because they both use the same design rules of C65 (although S65 uses bigger geometries and adds redundant elements). We used the same methodology and Quality of Results (QoR) targets in all implementations to ensure fair comparison between them.

Figure 18 shows the floorplan of the TCLS-based SoC implemented with C65 standard cells. The arrangement of components in this implementation is very similar to the other three implementations. The TCLS and debug logic are located in the central part of the chip, the cache memories

Table 2. PPA Results (% are Overheads w.r.t. the Baseline Single-Core Implementation Using S65 RHBD Libraries)

		Single-core S65 (Baseline)	TCLS C65		TCLS C65 & S65	
Performance		359.8MHz	314.3MHz	(−14%)	311MHz	(−16%)
		≈600 DMIPS	≈525 DMIPS	(−14%)	≈520 DMIPS	(−15%)
Power	Core only	491.3mW	530mW	(+7%)	553.1mW	(+13%)
	Total (incl. TCMs)	721.6mW	758.8mW	(+5%)	781.9mW	(+9%)
Area	Core only	2.86mm^2	3.64mm^2	(+27%)	3.8mm^2	(+33%)
	Total (incl. TCMs)	12.14mm^2	12.92mm^2	(+6%)	13.08mm^2	(+8%)

and TCMs span the top/bottom parts, and the peripherals are mapped to the bottom left side. The peripherals that take small chip area are not labeled in this figure (e.g., UART, SPI, timer, and watchdog); they are only represented with different colors. Table 2 shows the PPA numbers extracted from the place-and-route databases of the three implementations.

The TCLS solution results in similar PPA numbers as the baseline single-core implementation using S65 RHBD libraries. Namely, TCLS' overall impact on the chip area (<10%) is relaxed because peripherals, cache memories, and TCMs are not triplicated. The power consumption overhead varies between +5% and +9% when using exclusively C65 and when combining C65 and S65 libraries, respectively. It is important to note the impact on the achievable maximum clock frequency in the TCLS solution, which is reduced by around 15%, due to the complexity overhead brought by the triplicated CPU cores and the TCLS Assist Unit. A deeper analysis shows that the critical path in the single-core implementation is a memory to register connection located in the ECC of the data cache, while in the TCLS implementation the critical paths are in the standard cells area.

5.2 Discussion

The degradation in PPA numbers of the Arm TCLS design with selective RHBD radiation hardening (or not using rad-hard technology at all) does not seem to be prohibitive for building "best-in-class" space-qualified SoCs. Namely, the overhead in area and energy consumption of TCLS compared to a RHBD single-core processor is less than 10% when integrated in an SoC that implements relevant functions and peripherals in satellite applications. Note that these overhead numbers will decrease as the SoCs become more complex and integrate more on-chip functions and memory. In these SoCs, the TCLS can be useful to safely execute the routines and mechanisms necessary to ensure the integrity of the entire system, handling error situations in the other components. The TCLS also provides an effective error detection and recovery mechanism to increase system availability, which is not available in single-core rad-hard processors, and is of utmost importance in telecom satellites and autonomous vehicles.

The Arm Cortex-R5 TCLS-powered SoC prototype discussed in this work results in a more than 5× performance increase compared to the currently used SoC computer in Airbus D&S telecom satellites, named SCOC3, which is based on the Gaisler LEON3-FT processor (Koebel and Coldefy 2010) and manufactured using Atmel 180nm RHBP technology (Atmel Corp. 2005). Likewise, the TCLS-based SoC reduces the power consumption to less than half. Compared to the brand new BAE quad-core rad-hard RAD5545 SoC (BAE Systems 2017), the Arm Cortex-R5-based TCLS solution delivers 10× less performance but also consumes 30× less energy. Compared to the latest LEON4-FT-based Cobham-Gaisler quad-core GR740 processor, the TCLS solution provides 3× less

performance and 10× improvement in energy consumption. We are now looking on improving the performance of TCLS while still keeping its energy consumption within the power budget advised for space applications (Doyle et al. 2014), which in some cases is well below the consumption of rad-hard space-grade SoCs such as RAD5545. An on-going research effort in this line consists in replacing the Cortex-R5 CPUs with the newer and more powerful Cortex-R52 CPUs (2.16 DMIPS/MHz), and integrating an embedded on-chip FPGA (eFPGA) to accelerate custom signal processing algorithms (Poupat 2017).

While the improvement in energy consumption is inherent to the efficiency of the Arm architecture and the achievable performance depends on the specific CPU used, the TCLS solution allows one to leverage the continuous scaling of commercial process libraries, resulting in competitive manufacturing costs and a good prospect for further improvements in energy efficiency and performance with every new technology node. Namely, the CPUs in TCLS can be implemented using state-of-the-art commercial libraries with small additional costs, but it will require a huge effort and money to be invested to develop equivalent rad-hard libraries for the same technology nodes. Cost is in fact a critical aspect to consider by the "NewSpace" companies that are looking for ultra-reliable components at automotive pricing for building their satellite mega constellations. The TCLS also implements a number of soft error mitigation techniques that could be sufficient when radiation levels are relatively low, such as in LEO orbits, removing completely the need for rad-hard process technologies.

6 CONCLUSIONS

This article has introduced the Arm Triple Core Lock-Step (TCLS) processor using Cortex-R5 CPUs, and has discussed its integration into a proof-of-concept SoC to power the on-board computers in next-generation Airbus D&S telecom satellites. The TCLS processor can detect errors in the CPUs and recover automatically from most of them within microseconds, thereby reducing system downtime by a factor of 1,000× compared with COTS Cortex-R5 DCLS processors in a representative telecom satellite 10-year mission operating in LEO orbit. This article proposes to constrain the use of rad-hard technology, namely, only using RHBD libraries to implement the critical parts of the TCLS processor, such as the error recovery mechanism. Since the potentially critical parts represent only around 4% of the TCLS processor—we have identified only 2% of the TCLS elements to be critical by means of fault injection (i.e., 115 flip-flops)—this approach permits one to keep the performance and energy consumption overheads introduced by rad-hard technology to a minimum. The remaining 96% of the processor can benefit from the continuous scaling of commercial standard cell libraries and the entire solution leverages the commercial semiconductor technology breakthroughs. Although we have checked that the TCLS processor can run at a maximum clock frequency slightly lower than COTS processors used in commercial safety-critical terrestrial applications, it can still deliver around 5× more performance than the space-qualified processors currently used in Airbus D&S satellites, which implement register triplication at the circuit level. Furthermore, the TCLS consumes considerably less energy than currently available space-qualified processors, meeting the power budgets advised for future avionics systems. The TCLS solution described in this article paves the way for the adoption of COTS Arm technology (and its vast software ecosystem and user base) in the space sector as it can be potentially used with any type of Arm CPUs, including more powerful R-class cores (e.g., R52) and performance-oriented A-class families. The transition to using Arm technology in space is in fact led by both NASA and ESA that are currently developing Arm-based processors using RHBP techniques to protect against space radiation effects. Hence, TCLS is the first initiative for enabling cost-efficient computing in space, which might well be appealing by "NewSpace" companies that are looking to optimize costs and generate profit. After having qualified the TCLS solution up to TRL4 (i.e.,

technology validated in the lab), the next objective will be to test it on a space environment on-board an experimental CubeSat. Beyond space, key applications for the TCLS processor on Earth might well be in future autonomous cars and data center servers, in both cases using exclusively commercial standard cell libraries.

REFERENCES

Aeroflex Gaisler. 2015. *UT700 32-bit Fault-Tolerant SPARC V8/LEON 3FT Processor.* Technical Report.

T. Amort, W. Snapp, J. Evans, J. Popp, M. Cabanas-Holmen, and E. Cannon. 2011. 90nm RHBD ASIC design capability. In *Proceedings of the Military and Aerospace Programmable Logic Devices Workshop.*

G. Anelli, M. Campbell, M. Delmastro, F. Faccio, S. Floria, A. Giraldo, E. Heijne, P. Jarron, K. Kloukinas, A. Marchioro, P. Moreira, and W. Snoeys. 1999. Radiation tolerant VLSI circuits in standard deep submicron CMOS technologies for the LHC experiments: Practical design aspects. *IEEE Transactions on Nuclear Science* 46, 6 (1999), 1690–1696.

Arm Ltd. 2011. *Cortex-R5 and Cortex-R5F. Technical Reference Manual.* Technical Report.

Atmel Corp. 2004. *Rad-Hard 32-bit SPARC Embedded Processor TSC695F.* Technical Report.

Atmel Corp. 2005. *ATC18RHA Rad-Hard 0.18m CMOS Cell-Based ASIC Family for Space Use.* Technical Report.

Atmel Corp. 2011. *Rad-Hard 32-bit SPARC V8 Processor AT697F.* Technical Report.

Atmel Corp. 2016. Space components vs parts for automotive applications. In *Proceedings of the European Space Components Conference.*

A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr. 2004. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 1, 1 (2004), 11–33.

BAE Systems. 2008. *RAD750 Radiation-Hardened PowerPC Microprocessor.* Technical Report.

BAE Systems. 2017. *RAD5545 Multi-Core System-on-Chip Power Architecture Processor.* Technical Report.

M. Berg. 2013. Revisiting dual interlocked storage cell (DICE) single event upset (SEU) sensitivity. In *Proceedings of the Microelectronics Reliability and Qualification Working Meeting.*

T. Calin, M. Nicolaidis, and R. Velazco. 1996. Upset hardened memory design for submicron CMOS technology. *IEEE Transactions on Nuclear Science* 43, 6 (1996), 2874–2878.

Cobham. 2016. *GR712RC Data Sheet.* Technical Report.

W. R. Dawes, G. F. Derbenwick, and B. L. Gregory.1976. Process technology for radiation-hardened CMOS integrated circuits. *IEEE Journal of Solid-State Circuits* 11, 4 (1976), 459–465.

R. DeCoursey, R. Melton, and R. R. Estes. 2006. Non-radiation hardened microprocessors in space-based remote sensing systems. In *Proceedings of the SPIE Europe Remote Sensing Conference.*

V. Degalahal, R. Ramanarayanan, N. Vijaykrishnan, Y. Xie, and M. J. Irwin. 2006. *Effect of Power Optimizations on Soft Error Rate.* Springer, Chapter 1, 1–20.

R. Doyle, R. Some, W. Powell, G. Mounce, M. Goforth, S. Horan, and M. Lowry. 2014. High performance spaceflight computing (HPSC) next-generation space processor (NGSP): A joint investment of NASA and AFRL. In *Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space.*

ESA/ESTEC. 2018a. Statement of Work: Arm-based MCU (TEC/2016.43). Retrieved March 6, 2018 from http://emits.sso.esa.int/emits/owa/emits_online.showao?typ1=7593&user=Anonymous.

ESA/ESTEC. 2018b. This is the Year Internet from Space gets Really Serious. Retrieved January 7, 2019 from http://www.universetoday.com/138210/year-internet-space-gets-really-serious.

M. M. Ghahroodi, E. Ozer, and D. Bull. 2013. SEU and SET-tolerant Arm Cortex-R4 CPU for space and avionics applications. In *Proceedings of the Workshop on Manufacturable and Dependable Multi-core Architectures at Nanoscale.*

R. Ginosar. 2012. Survey of processors for space. In *Proceedings of the Data Systems in Aerospace Conference.*

D. Gizopoulos, M. Psarakis, S. V. Adve, P. Ramachandran, S. Siva, S. Hari, D. Sorin, A. Meixner, A. Biswas, and X. Vera. 2011. Architectures for online error detection and recovery in multicore processors. In *Proceedings of the Conference on Design, Automation and Test in Europe.*

GomSpace A/S. 2017. *NanoMind Z7000 Datasheet On-board CPU and FPGA for Space Applications.* Technical Report.

L. Hansen. 2016. *Unleash the Unparalleled Power and Flexibility of Zynq UltraScale+ MPSoCs - WP470.*

M. J. Hargrove, S. Voldman, R. Gauthier, J. Brown, K. Duncan, and W. Craig. 1998. Latchup in CMOS technology. In *Proceedings of the IEEE International Reliability Physics Symposium.*

R. Hillman, G. Swift, P. Layton, M. Conrad, C. Thibodeau, and F. Irom. 2003. Space processor radiation mitigation and validation techniques for an 1,800 MIPS processor board. In *Proceedings of the European Conference on Radiation and Its Effects on Components and Systems.*

M. Hjorth, M. Aberg, N. J. Wessman, J. Andersson, R. Chevallier, R. Forsyth, R. Weigand, and L. Fossati. 2015. GR740: Rad-hard quad-core LEON4FT system-on-chip. In *Proceedings of the Data Systems in Aerospace Conference.*

Infineon Tech. 2012. *Tricore: Highly Integrated and Performance Optimized 32-bit Microcontrollers for Automotive and Industrial Applications*. Technical Report.

X. Iturbe, D. Keymeulen, P. Yiu, D. Berisford, R. Carlson, K. Hand, and E. Ozer. 2016a. *On the Use of System-on-Chip Technology in Next-Generation Instruments Avionics for Space Exploration*. Springer, Chapter 1, 1–22.

X. Iturbe, B. Venu, J. Jagst, E. Ozer, P. Harrod, C. Turner, and J. Penton. 2018. Addressing functional safety challenges in autonomous vehicles with the arm triple core lock-step (TCLS) architecture. *IEEE Design and Test Magazine* 35, 3 (2018), 7–14.

X. Iturbe, B. Venu, and E. Ozer. 2016b. Soft error vulnerability assessment of the real-time safety-related Arm Cortex-R5 CPU. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*.

X. Iturbe, B. Venu, E. Ozer, and S. Das. 2016c. A triple core lock-step (TCLS) Arm Cortex-R5 processor for safety-critical and ultra-reliable applications. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*.

C. M. Jeffery and R. J. O. Figueiredo. 2012. A flexible approach to improving system reliability with virtual lockstep. *IEEE Transactions on Dependable and Secure Computing* 9, 1 (2012), 2–15.

A. H. Johnston. 2000. Scaling and technology issues for soft error rates. In *Proceedings of the Annual Research Conference on Reliability*.

M. Kaliorakis, D. Gizopoulos, R. Canal, and A. Gonzalez. 2017. MeRLiN: Exploiting dynamic instruction behavior for fast and accurate microarchitecture level reliability assessment. In *Proceedings of the 2017 Annual International Symposium on Computer Architecture*.

S. Kanekal, A. Jones, B. Randol, D. Patel, E. Summerlin, E. Gorman, G. Crum, G. D. Nolfo, N. Paschalidis, S. Heyward, and S. Riall. 2014. CeREs: A compact radiation belt explorer. In *Proceedings of the AIAA/USU Conference on Small Satellites*.

F. Koebel and J. F. Coldefy. 2010. SCOC3: A space computer on a chip. In *Proceedings of the Conference on Design, Automation and Test in Europe*.

P. Koopman and M. Wagner. 2016. Challenges in autonomous vehicle testing and validation. In *Proceedings of the SAE World Congress Conference*.

T. Kuschel, R. Mariani, and H. Shigehara. 2010. A flexible microcontroller architecture for fail-safe and fail-operational systems. In *Proceedings of the HiPEAC Workshop on Design for Reliability*.

R. C. Lacoe, J. V. Osborn, R. Koga, S. Brown, and D.C. Mayer. 2000. Application of hardness-by-design methodology to radiation-tolerant ASIC technologies. *IEEE Transactions on Nuclear Science* 47, 6 (2000), 2334–2341.

M. D. Lam, E. E. Rothberg, and M. E. Wolf. 1991. The cache performance and optimizations of blocked algorithms. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*.

S. Lin, Y. B. Kim, and F. Lombardi. 2011. A 11-transistor nanoscale CMOS memory cell for hardening to soft errors. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 19, 5 (2011), 900–904.

Lockheed Martin. 1995. *RAD6000 Radiation Hardened 32-Bit Processor*. Technical Report.

Maxwell Tech. 2013. *SCS750 Single Board Computer for Space*. Technical Report.

Moog Broad Reach. 2015. *BRE440 Rad-Hard CPU*. Technical Report.

NASA. 2015. *Statement of Work (SOW) for the Development of the High Performance Space Computing (HPSC) Processor*. Technical Report.

E. Normand. 1996. Single event upset at ground level. *IEEE Transactions on Nuclear Science* 43, 6 (1996), 2742–2750.

E. Normand. 2000. Radiation Effects in Spacecraft and Aircraft. Retrieved January 7, 2019 from http://lws.gsfc.nasa.gov/documents/mission_requirements/normand_020900.pdf.

NXP. 2012. *Automotive Solutions Setting the Pace for Innovation*. Technical Report.

M. Pignol. 2010. COTS-based applications in space avionics. In *Proceedings of the Conference on Design, Automation and Test in Europe*.

J. L. Poupat. 2017. DAHLIA system-on-chip. In *Proceedings of the IEEE International Conference on Space Mission Challenges for Information Technology*.

J. L. Poupat, B. Leroy, and T. Helfers. 2017. TCLS arm for space. In *Proceedings of the Conference on Data Systems in Aerospace*.

Renesas. 2015. *Main Specifications of the R-Car H3 SoC*. Technical Report.

S. Resch, A. Steininger, and C. Scherrer. 2013. Software composability and mixed criticality for triple modular redundant architectures. In *Proceedings of the SASSUR International Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems*.

J. Rhea. 2002. BAE Systems moves into Third Generation Rad-Hard Processors. Retrieved January 7, 2019 from http://www.militaryaerospace.com/articles/print/volume-13/issue-5/news/bae-systems-moves-into-third-generation-rad-hard-processors.html.

T. Romanko and B. Clegg. 2005. SOI Eases Radiation-Hardened ASIC Designs. Retrieved January 7, 2019 from https://www.design-reuse.com/articles/10962/soi-eases-radiation-hardened-asic-designs.html.

D. Rudolph, C. Wilson, J. Stewart, P. Gauvin, A. George, H. Lam, G. Crum, M. Wirthlin, A. Wilson, and A. Stoddard. 2014. CHREC space processor: A multifaceted hybrid architecture for space computing. In *Proceedings of the AIAA/USU Conference on Small Satellites*.

H. Saito, Y. Masumoto, T. Mizuno, A. Miura, M. Hashimoto, H. Ogawa, S. Tachikawa, T. Oshima, A. Choki, H. Fukuda, M. Hirahara, and S. Okano. 2001. Piggy-back satellite for Aurora observation and technology demonstration. *Acta Astronautica* 48, 5 (2001), 723–735.

K. M. Schlesier. 1974. Radiation hardening of CMOS/SOS integrated circuits. *IEEE Transactions on Nuclear Science* 47, 6 (1974), 152–158.

T. Scholastique and L. Hili. 2017. A 65nm Hardened ASIC Technology for Space Applications. Retrieved January 7, 2019 from http://indico.esa.int/indico/event/165/contribution/13/material/1/1.pdf.

M. R. Shaneyfelt, P. E. Dodd, B. L. Draper, and R. S. Flores. 1998. Challenges in hardening technologies using shallow-trench isolation. *IEEE Transactions on Nuclear Science* 45, 6 (1998), 2584–2592.

Texas Instruments. 2014. *Hercules TMS570 Microcontrollers*. Technical Report.

B. Venu, E. Ozer, X. Iturbe, and A. Robinson. 2016. A fail-functional automotive CPU subsystem architecture for mitigating single point of failures. In *Proceedings of the IEEE International Workshop on Automotive Reliability and Test*.

A. Vernile. 2018. *The Rise of Private Actors in the Space Sector*. Springer.

Vorago Tech. 2017. *VA10820 - Radiation Hardened Arm Cortex-M0 MCU Datasheet*. Technical Report.

R. Whitwam. 2014. NASA's Orion Spacecraft runs on a 12 Year-old Single-Core Processor from the iBook G3. Retrieved January 7, 2019 from https://www.geek.com/chips/nasas-orion-spacecraft-runs-on-a-12-year-old-single-core-processor-from-the-ibook-g3-1611132/.

C. Wilson, J. Stewart, P. Gauvin, J. MacKinnon, J. Coole, J. Urriste, A. George, G. Crum, E. Timmons, J. Beck, T. Flatley, A. Wilson, M. Wirthlin, and A. Stoddard. 2014. CSP hybrid space computing for STP-H5/ISEM on ISS. In *Proceedings of the AIAA/USU Conference on Small Satellites*.

Received October 2017; revised June 2018; accepted November 2018