Soft Error Vulnerability Assessment of the Real-Time Safety-Related ARM Cortex-R5 CPU

Xabier Iturbe, Balaji Venu, Emre Ozer
ARM Research, Cambridge, UK
{xabier.iturbe, balaji.venu, emre.ozer}@arm.com

Abstract—**This paper presents the results collected in a series of fault injection experiments conducted on a modern commercial embedded ARM Cortex-R5 processor, which is extensively used in real-time safety-related embedded applications. The paper aims to be a comprehensible study on how faults propagate through this CPU as they turn into errors at the core boundaries. The main goal of this study is to identify the most vulnerable parts in the micro-architecture of the ARM Cortex-R5 CPU. The long-term objective is to propose and decide how to protect these vulnerable parts without impacting the characteristic features of ARM processors: energy-efficiency and high-performance.**

I. Introduction

A soft error, or Single Event Upset (SEU), is defined as a transient piece of incorrect machine state [1]. It occurs when a transient fault in logic, i.e., Single Event Transient (SET), is latched in a flip-flop, or when the value stored in a memory cell is inverted as a result of electrical noise or charged particle strikes. Soft errors due to energetic particle strikes are typically provoked by high-energy neutrons from cosmic radiation [2].

The susceptibility of modern processors to soft errors is dramatically increasing with the exponentially growing device count per chip as the process nodes become ever smaller (i.e., Moore's law). Device scaling in the VLSI era results in 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 sequential element (i.e., flip-flops and memory cells) [3]. Thus, 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 decrease [2]. All in all, soft errors have been recognized as one of the most important limits for digital electronic reliability. They have descended from space altitudes, where they have been confined for decades, and now threat millions of processors operating at the heart of terrestrial safety-critical equipment [4].

Several solutions have been proposed to mitigate soft errors over the years. These solutions range from using specialized rad-hard process technology [5] to architecture and circuit level fault-tolerance techniques, but are dominated by the use of redundant components: transistors and metal layers at the process level [6], [7], flip-flops and Error Correction Codes (ECCs) at the circuit level [8], [9], CPU cores and fault supervision circuits at architecture levels [10]-[14], as well as software processes and virtualized partitions at the software/hypervisor level [15], [16].

However, the redundant components used in fault-tolerant processors result in area and power overheads and usually reduce the achievable performance. A deeper knowledge of how soft errors affect the processor and propagate through it would be very helpful to pave the way for new area and energy efficient mitigation techniques that do not compromise the performance. This paper is aimed at explaining the effect of soft errors on a commercial mainstream processor currently used in terrestrial real-time and safety-related embedded applications: the ARM® Cortex®-R5. The paper is based on the results collected in a series of extensive fault injection experiments and investigates the fault propagation process through this CPU's micro-architecture, covering from the fault occurrence to the fault manifestation in the CPU boundary ports.

The remaining of the paper is organized as follows. Section II introduces the ARM Cortex-R5 CPU and outlines its micro-architecture. Then, Section III describes the fault-injection experiments that have been conducted and Section IV discusses the obtained results, relating these to the main components in the Cortex-R5 micro-architecture. Finally, Section V draws the final conclusions of the paper.

II. The ARM Cortex-R5 Processor

The ARM Cortex-R class embedded processors offer energy-efficiency, high performance, highly deterministic behaviour, and built in safety features. They are specifically designed for deeply embedded real-time applications, such as automotive. Namely, they include Tightly Coupled Memories (TCMs) to store instruction and data for bounded real-time response, such as interrupt handling routines, and use a micro Snoop Control Unit (uSCU) to maintain data coherency with the data cache when dealing with DMA transactions in real-time streaming applications. Cortex-R5 processors also include a Low-Latency Peripheral Port (LLPP) to carry out low-latency communications with external peripherals through AXI and/or AHB buses.

ARM Cortex-R processors are usually configured as multi-core processors containing two CPU cores. In safety-critical applications, they are typically used in Dual-Core Lock-Step (DCLS) configuration, where the two CPU cores share primary core's TCMs, caches and peripheral ports. The primary CPU runs 2 clock cycles ahead the secondary CPU to minimize the possibility that both CPUs are affected by a soft error at the same time. The outputs of the two CPUs are compared at every clock cycle to detect errors (i.e., divergences).

In this paper we focus on the Cortex-R5 processor, configured in DCLS. This processor delivers up to 1.67 DMIPS/MHz performance and 62 DMIPS/mW efficiency when implemented in 28nm HPM commercial process technology. Cortex-R5 CPUs are at the heart of TI Hercules TMS570 microcontrollers [17] and in the Xilinx UltraScale MPSoCs [18].

A. *ARM Cortex-R5 CPU Micro-Architecture*

Fig. 1 depicts the main components in the Cortex-R5 micro-architecture and the percentage of the total CPU sequential elements they use (in red colour). In this figure, we use the prefix IF_x to refer to the interface that a given component x exposes in the CPU core boundaries.

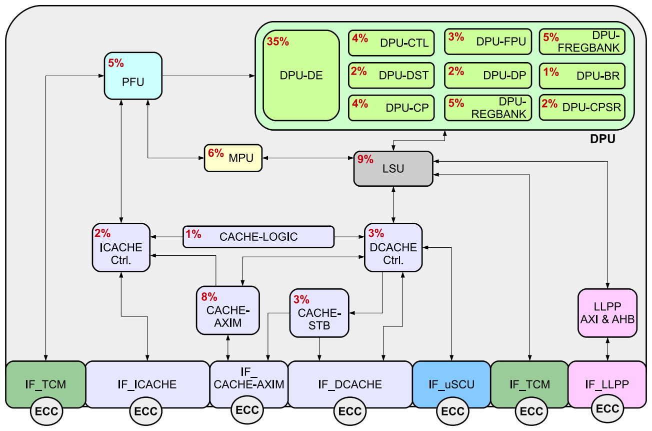


Fig. 1: Cortex-R5 micro-architecture (Adapted from [12])

The Cortex-R5 CPU has an 8-stage pipelined in-order dual-issue 32-bit micro-architecture with 5 execution paths [12]. It implements the ARMv7-R instruction set and also supports Thumb-2 for high code density. As the Cortex-R5 is designed for safety-related applications, it includes a number of hardware mechanisms to cope with soft and hard errors. Caches and TCMs are protected against soft errors using parity checking and ECC (i.e., SECDED), and a small cache memory is included to replace up to 64 TCM memory-cells when these get permanently damaged.

There are 5 CPU 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. In this study, we use 16 KB instruction and data cache memories.

The Pre-Fetch Unit (PFU) obtains instructions from the I-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). 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. A pre-decode stage is used to accommodate branch prediction and an instruction queue that keeps the DPU fed with instructions.

This study breaks the DPU down into 10 major 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 coprocessor instructions and implements the CP15 and CP14 registers used to configure TCMs, caches, breakpoints and watchpoints. 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 (i.e., not associated with a specific data-path).

III. Fault Injection Methodology

This section describes the main aspects of the fault injection methodology that has been used to assess the soft error vulnerability of the Cortex-R5.

A. *Benchmark Applications*

We use 12 applications from the EEMBC's AutoBench benchmark suite that characterize the most common operations in safety-critical automotive applications [19]. These include a Controller Area Network (CAN) `canrdr01`, tooth-to-spark (i.e., locating the engine's cog when the spark is ignited) `ttsprk01`, angle-to-time conversion `a2time01`, road speed calculation `rspeed01`, pulse-width modulation `puwmod01` and table lookup and interpolation `tblook-01`, as well as some of the most representative algorithms which are becoming increasingly important for sensors used in engine knock detection, vehicle stability control, and occupant safety systems. These algorithms use matrix mathematics `matrix01`, Fast Fourier Transforms (FFT and iFFT) `aiifft01/aifftr01`, Finite and Infinite Impulse Response (IIR and FIR) filters `aifirf01/iirflt01` and Discrete Cosine Transforms (DCT and iDCT) `idctrn01`. In order to reduce the total execution time, we run only 10 iterations of each of the application benchmarks, taking between 50,000 and 475,000 clock cycles to be completed. Prior to executing the benchmarks, two routines initialize the Cortex-R5 processor and the heap memory segment within approximately 30,000 clock cycles.

B. *Terminology, Definitions and Assumptions*

In this paper we use the dependability terminology presented in [20]. In a DCLS processor this terminology can be applied as shown in Fig. 2. 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 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. Although this is a very conservative and pessimistic approach, this level of preventive dependability is required in safety-critical applications which are our focus in this study.

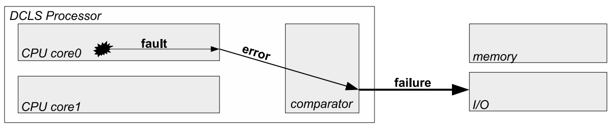


Fig. 2: Fault, error and failure in a DCLS processor

A fault can remain dormant, affect an idle component (i.e., 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 [1]. 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 [21]).

Our fault injection experiments are intended to identify the most sensitive sequential elements in the CPU's architecture (i.e., software-visible registers) and micro-architecture (i.e., all remaining state elements). We consider the worst-case scenario where all the sequential elements in the processor could potentially be affected by soft errors, either by latching a SET or as a result of a high-energy particle direct strike. We assume the same Soft Error Rate (SER) for every sequential element in the CPU. We also assume that the CPU behaviour with respect to soft errors is statistically similar during the execution of the benchmark applications.

C. *Fault Injection Experiments*

We inject faults in the sequential elements of the primary CPU core and rely on the DCLS comparator logic to detect error situations. If the injected fault does not result in an error, the architecture state of the two Cortex-R5 CPUs is compared at the end of the benchmark to detect dormant faults. These are also marked as errors following a safety-critical oriented worst-case conservative approach.

We simulate the Cortex-R5 DCLS processor using the Synopsis VCS. This allows us to obtain accurate fault injection results without incurring the high economic costs of irradiation tests and the intractable device level simulation time (i.e., electrical simulation). Our RTL simulation only covers logic gate masking and micro-architecture level masking mechanisms, thus resulting in a worst-case dependability analysis which is still adequate for safety-critical applications.

In each fault injection experiment a single fault is injected in a given sequential element at a given instant of time. Since it is intractable to inject a fault in each of the tens of thousands sequential elements existing in a Cortex-R5 CPU at every execution clock cycle, the benchmark execution time (excluding the initialization routines) is divided into 64 equally-sized intervals, and for each of these intervals a random fault injection instant is selected. This represents on average one fault injected per few hundreds of clock cycles, involving more than 1.5 million fault injection experiments per benchmark. Hence, we randomly sample the time domain along the entire benchmark execution, rather than conducting a complex lifetime analysis of each sequential element, as typically done when computing the Architectural Vulnerability Factor (AVF) [22].

IV. Fault Injection Results

A. *Overall Error Rate*

Fig. 3 breaks down the total observed CPU error rate in each benchmark for the components that show the highest error contribution. As expected, the components that provoke most of the errors in the CPU are the register bank files: DPU-REGBANK and DPU-FREGBANK (when using floating-point operations), each of which accounts for at least 25% of the total CPU errors in all benchmarks. Note that a fault in these structures directly affects the data being processed, which is a much more straight forward error propagation mechanism than in other CPU components, where faults are more likely to affect data-path logic, and hence are prone to be masked before propagating to data. This straight error propagation mechanism also dominates the CACHE-STB, LSU and CACHE-AXIM, which all are equipped with data buffers.

The DPU-DE is the second ranked component in terms of the absolute number of CPU errors it provokes. This can be explained by the fact this component is potentially involved in the execution of every single instruction, regardless of its type and its execution path. Although the DPU-CTL is also involved in the execution of most instructions, its error rate contribution is lower because it is considerably smaller than

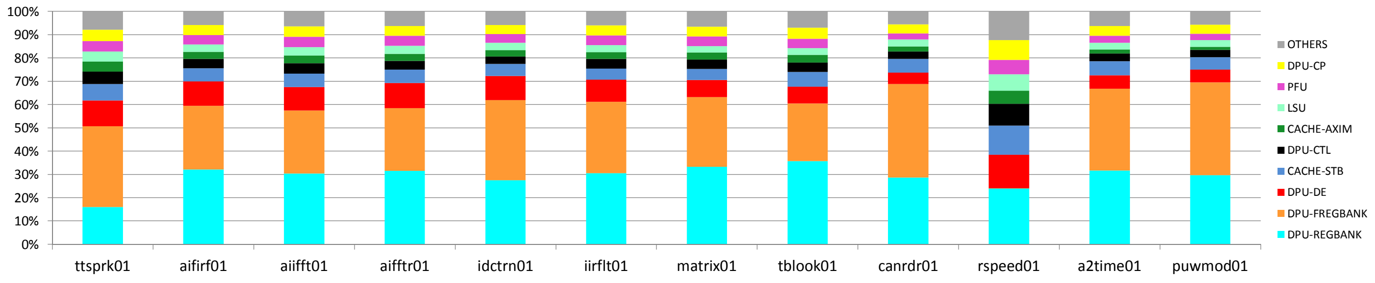
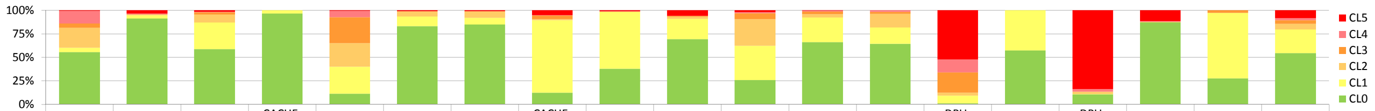


Fig. 3: Error rate by benchmark and by CPU component



	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 TOTAL
CL5	5	47	31	0	4	8	6	98	3	34	23	36	0	608	0	1045	111	0	2059
CL4	159	12	12	0	45	0	0	9	1	0	12	76	10	160	0	32	6	0	534
CL3	55	8	49	0	187	2	2	73	1	4	68	286	12	253	0	0	3	5	1008
CL2	250	4	166	0	169	37	37	21	1	15	311	314	90	33	0	32	0	0	1480
CL1	53	50	556	5	196	75	41	1454	213	126	399	2406	105	96	354	6	2	130	6267
CL0	651	1285	1165	144	76	596	483	234	133	404	284	6149	393	14	473	132	839	52	13507

Fig. 4: Number (in table) and percentage (in chart) of critical elements in each CPU component

the DPU-DE. On the other hand, the components instantiated in a specific CPU execution path show a notable lower error rate as they remain idle for a longer time and faults are masked every time new data is loaded in the pipeline. The most notable example is the DPU-FPU, which remains idle when executing integer code, and thus shows the lowest error rate among all CPU components (less than 0.3%).

The PFU, which lies in the early stages of the CPU pipeline where there is a single execution path, also shows a significant contribution to the overall CPU error rate. 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.

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 component and can also be seen in a lesser extent in the DPU-CPSR.

B. CPU Component Criticality Analysis

We have categorized 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), level 5 indicates that a fault in that element always results in an error, and level 1 to 4 cover different error rates in 25% increments each (CL1: 1-25%; CL2: 25-50%; CL3: 50-75% and CL4: 75-99%). Each sequential element is classified in the highest CL shown in any of the benchmarks tested, and all functionally-equivalent components (e.g., same data vector register bits) are classified in the same CL, equal to the highest CL among them.

Fig. 4 shows the number and percentage of sequential elements classified in each CL in each CPU component. Note that more than half of the total sequential elements in the CPU never provoke any error, and more than 70% of the errors are provoked by around only 2,000 sequential elements. Since this represents less than 10% 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.

The register files solely account for up to 80% of the total CL5 elements in the CPU. The floating-point register file contributes with around 30% 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 the CPU 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) [23].

Other components that buffer data for a significant amount of time, such as the CACHE-STB and the CACHE-AXIM, also have a very reduced number of non-critical CL0 components, while this does not happen in the components that store data for only a few clock cycles, such as the LSU and DPU-LDST.

Control registers tend to be classified in CL5 as they have the potential to affect a large number of the instructions executed by the processor, directly or indirectly, and thus provoke errors in the processed data. This is the case of CP15/CP14 in DPU-CP, SPSR/CPSR in DPU-CPSR and the CP15-related structures in MPU. On the other hand, datapath registers in the CPU pipeline, which have a very short lifetime, tend to be classified in more relaxed criticality levels. For instance, the DPU-DP and DPU-FPU execution units do not have any sequential element classified in CL4 and CL5. Indeed, note that instructions persist longer in the instruction queue in the DPU-DE than they take to execute in the DPU-DP and DPU-FPU. Together with the instruction queue, the

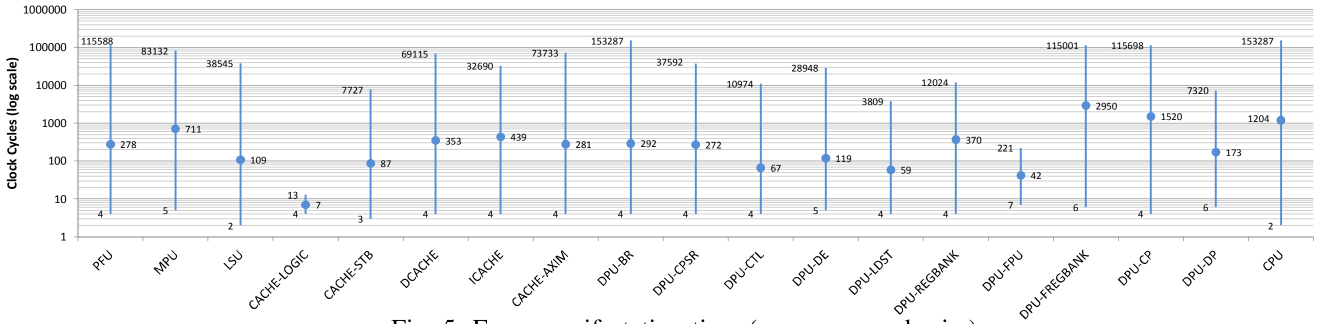


Fig. 5: Error manifestation time (max. avg. and min.)

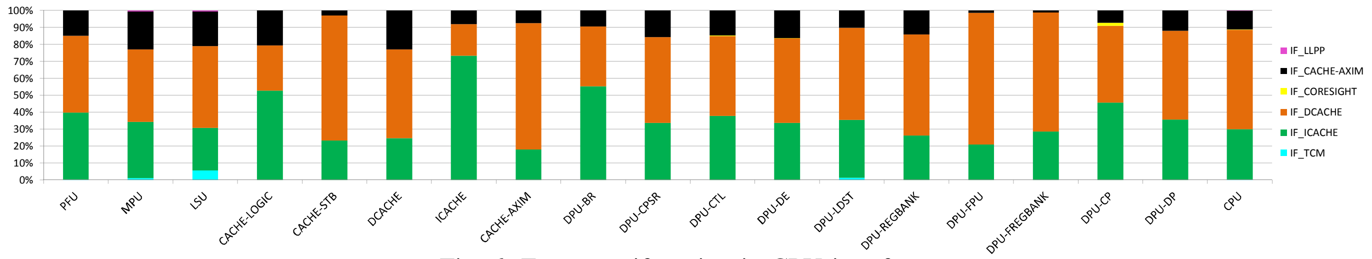


Fig. 6: Error manifestation in CPU interfaces

PC logic and the logic that translates register numbers into physical ones are the most critical structures in the DPU-DE component. In the DPU-CTL, the most critical structure is the exception handler.

In the PFU, the most critical structures are the return-address stack and the branch prediction table. However, since the branch predictor table has a considerable number of entries that are randomly accessed during the program execution, the error rate tends to be well dispersed along different CLs.

C. Error Propagation and Manifestation

Fig. 5 and Fig. 6 show the number of clock cycles that take the faults to propagate to the CPU boundary (i.e., error manifestation time) and the ports on which they are manifested. These figures allow us to trace the error propagation paths in the CPU micro-architecture and evaluate how straight forward is each of these paths.

It is interesting to note that errors manifest in only 65% of the CPU ports, with almost 80% of these errors manifesting in only 20% of the ports (see Fig. 7). The vast majority of errors manifest in the IF_DCACHE (almost 60%) and IF_ICACHE (about 30%). This is consistent with the results shown in previous sections, which indicate that data structures, such as register files and cache store buffer, are the most vulnerable components in the CPU micro-architecture. Note that faults affecting these data structures tend to propagate to the IF_DCACHE, which is the most commonly used CPU output data interface. Errors affecting data structures can also manifest in the IF_ICACHE as the processed data is also related with the instruction addresses issued by the CPU by means of comparison operators in the program. Besides, some structures in the CPU micro-architecture are exclusively related with the issued instruction addresses (e.g., PC logic in the DPU-DE) and therefore, faults affecting these components tend to manifest in the IF_ICACHE.

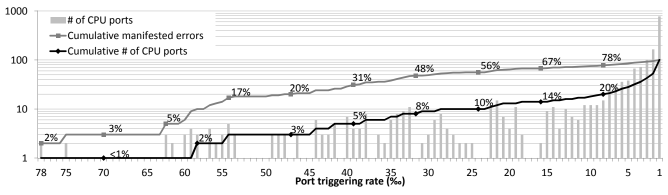
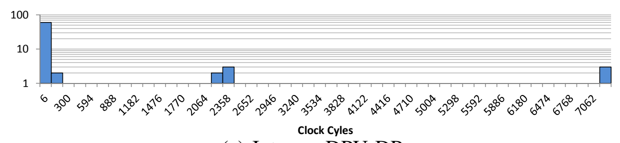


Fig. 7: Error manifestation port histogram

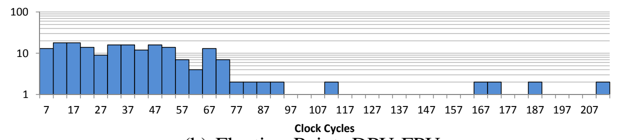
Only 10% of the total CPU errors are manifested in the IF_AXIM, which reflects the smaller activity rate in this interface; i.e., it is only used after a cache miss. A small fraction of the errors manifest in the IF_TCM, which is significantly less used than the caches and main memory, and a negligible amount of errors manifest in the IF_LLPP interface. Finally, a few number of errors manifest in the debugging interface IF_CORESIGHT. Namely, the errors that corrupt the configuration/functioning of the CPU pipeline, such as errors affecting the DPU-CP component.

As shown in Fig. 5, all of the CPU components have a minimum error manifestation time equal or less than 7 clock cycles, whereas the average error manifestation time in the whole CPU is 1,204 clock cycles. Indeed, it is interesting to see the high variability of the error manifestation time across the different CPU components, which can be as high as tens of thousands clock cycles.

As expected, errors propagate more quickly when they directly affect the 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 the 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 [24]. Therefore, the DPU-REGBANK, which is typically involved in more immediate computations, has significantly shorter error manifestation time than the DPU-FREGBANK, as shown in Fig. 5. The exception are some long-life integer variables, such as indexes in iterative loops. Errors affecting these variables tend to manifest in the CPU ports after thousands of clock cycles, which contrasts with the tens of hundreds clock cycles needed by the vast majority of errors affecting the DPU-DP (see Fig. 8a). 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. The latter component deals only with data variables, and therefore shows a uniform error manifestation time distribution within a short time span (see Fig. 8b).



(a) Integer: DPU-DP



(b) Floating-Point: DPU-FPU

Fig. 8: Error manifestation time histogram in execution units

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 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, control register centric components (e.g., DPU-CP and MPU) show one of the highest average error manifestation time, which points to an intricate error propagation path of faults that corrupt the CPU configuration/functioning.

V. Conclusions

This paper has assessed the soft error vulnerability of the ARM Cortex-R5 CPU, which is currently extensively used in safety-related real-time applications. The paper has conducted an intensive fault injection campaign on this CPU while running a set of representative automotive benchmarks to study how faults are propagated through the CPU micro-architecture and manifested as errors at its boundaries. We conclude that less than 10% of the sequential elements in the Cortex-R5 CPU account for more than 70% of the errors, and these errors are manifested in only 65% of the CPU output ports. The most critical structures in the CPU micro-architecture are: (1) register files, (2) cache store buffer, (3) branch prediction table, (4) branch return-address stack, (5) exception handler logic, (6) instruction queue, (7) PC logic, (8) register translation logic, as well as some control registers, such as (9) CP14/CP15s and (10) CPSR/SPSRs. These results suggest that an important reliability improvement could be achieved by protecting only a few hundreds of sequential CPU elements. The results reported in this paper could also help developing more accurate models to estimate the AVF in complex CPUs, such as ARM Cortex-A.

References

[1] J. A. Blome, S. Gupta, S. Feng anf S. Mahlke, *Cost-Efficient Soft Error Protection for Embedded Microprocessors*, Proc. of the Intl. Conf. on Compilers, Architectures and Synthesis for Embedded Systems, 2006.
[2] A. H. Johnston, *Scaling and Technology Issues for Soft Error Rates*, Proc. of the Annual Research Conf. on Reliability, 2000.
[3] V. Degalahal, R. Ramanarayanan, N. Vijaykrishnan, Y. Xie and M. J. Irwin, *Effect of Power Optimizations on Soft Error Rate*, IFIP Series on VLSI-SoC, Springer, pp. 1-20, 2006.
[4] E. Normand, *Single Event Upset at Ground Level*, IEEE Trans. on Nuclear Science, Vol. 43, No. 6, pp. 2742-2750, 1996.
[5] BAE Systems, *RAD750 Radiation-Hardened PowerPC Microprocessor*, 2008.
[6] S. Lin, Y. B. Kim and F. Lombardi, *A 11-transistor Nanoscale CMOS Memory Cell for Hardening to Soft Errors*, IEEE Trans. on Very Large Scale Integration Systems, Vol. 19, No. 5, pp. 900-904, 2011.
[7] T. Calin, M. Nicolaidis and R. Velazco, *Upset Hardened Memory Design for Submicron CMOS Technology*, IEEE Trans. on Nuclear Science, Vol. 43, No. 6, pp. 2874-2878, 1996.
[8] Aeroflex Gaisler, *UT700 32-bit Fault-Tolerant SPARC V8/LEON 3FT Processor*, 2015.
[9] M. M. Ghahroodi, E. Ozer and D. Bull, *SEU and SET-Tolerant ARM Cortex-R4 CPU for Space and Avionics Applications*, Proc. of the Workshop on Manufacturable and Dependable Multi-core Architectures at Nanoscale, 2013.
[10] Maxwell Technologies, *SCS750 Single Board Computer for Space*, 2013.
[11] X. Iturbe, B. Venu, E. Ozer and S. Das, *A Triple Core Lock-Step (TCLS) ARM Cortex-R5 Processor for Safety-Critical and Ultra-Reliable Applications*, Proc. of the IEEE/IFIP Intl. Conf. on Dependable Systems and Networks, 2016.
[12] ARM, *Cortex-R5 and Cortex-R5F. Technical Reference Manual*, 2011.
[13] Infineon, *Tricore: Highly Integrated and Performance Optimized 32-bit Microcontrollers for Automotive and Industrial Applications*, 2012.
[14] R. Mariani, T. Kuschel and H. Shigehara, *A Flexible Microcontroller Architecture for Fail-Safe and Fail-Operational Systems*, Proc. of the HiPEAC Workshop on Design for Reliability, 2010.
[15] S. Resch, A. Steininger and C. Scherrer, *Software Composability and Mixed Criticality for Triple Modular Redundant Architectures*, Proc. of the Intl. Conf. on Computer Safety, Reliability and Security, 2013.
[16] C. M. Jeffery and R. J. O. Figueiredo, *A Flexible Approach to Improving System Reliability with Virtual Lockstep*, IEEE Trans. on Dependable and Secure Computing, Vol. 9, No. 1, pp. 2-15, 2012.
[17] Texas Instruments, *Hercules TMS570 Microcontrollers*, 2014.
[18] Xilinx, *UltraScale Architecture and Product Overview*, 2016.
[19] http://www.eembc.org/benchmark/automotive_sl.php
[20] A. Avizienis, J. C. Laprie, B. Randell and C. Landwehr, *Basic Concepts and Taxonomy of Dependable and Secure Computing*, IEEE Trans. on Dependable and Secure Computing, Vol. 1, No. 1, pp. 11-33, 2004.
[21] J. A. Butts and G. Sohi, *Dynamic Dead-instruction Detection and Elimination*, Proc. of the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems, 2002.
[22] S. S. Mukherjee, C. T. Weaver, J. Emer, S. K. Reinhardt and T. Austin, *Measuring Architectural Vulnerability Factors*, IEEE Micro, Vol. 23, No. 6, pp. 70-75, 2003.
[23] P. Montesinos, W. Liu and J. Torrellas, *Using Register Lifetime Predictions to Protect Register Files against Soft-Errors*, Proc. of the IEEE/IFIP Intl. Conf. on Dependable Systems and Networks, 2007.
[24] M. D. Lam, E. E. Rothberg and M. E. Wolf, *The Cache Performance and Optimizations of Blocked Algorithms*, Proc. of the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems, 1991.