A Portable and Fault-Tolerant Microprocessor Based on the SPARC V8 Architecture

Jiri Gaisler

Gaisler Research, 411 08 Göteborg, Sweden

jiri@gaisler.com

Abstract

The architecture and implementation of the LEON-FT processor is presented. LEON-FT is a fault-tolerant 32-bit processor based on the SPARC V8 instruction set. The processors tolerates transient SEU errors by using techniques such as TMR registers, on-chip EDAC, parity, pipeline restart, and forced cache miss. The first prototypes were manufactured on the Atmel ATC35 0.35 µm CMOS process, and subjected to heavy-ion fault-injection at the Louvain Cyclotron. The heavy-ion tests showed that all of the injected errors (> 100,000) were successfully corrected without timing or software impact. The device SEU threshold was measured to be below 6 MeV while ion energy-levels of up to 110 MeV were used for error injection.

1. Introduction

In 1997, the European Space Agency (ESA) completed the development of a 32-bit microprocessor for embedded space-flight applications, denoted ERC32 [1]. The ERC32 is based on the Cypress CY601 SPARC V7 processor and is now being used in several space projects, including the control computers of the International Space Station. To meet the mission requirements for projects beyond year 2000, the development of a new and improved processor denoted LEON was started in 1998. This paper presents the design goals, architecture, built-in fault-tolerance functions and initial test results of this processor. The LEON project was started by ESA under the Douglas Marsh fellowship and the ESA Technology Research Programme (TRP), and is now continued by Gaisler Research under ESTEC contract 15102/01/NL/FM.

2. Background

European microprocessors designed for space applications have typically had three main design drivers; radiation-hardness, performance and development cost. The objectives have been to within a limited budget, develop a device that can withstand the space radiation environment and provide the highest possible performance on the given semiconductor process. Other aspects, such as software compatibility, system integration, component cost, design reuse and future evolution have been less emphasized. However, the conditions under which on-board computers, and subsequently processors, are being developed are rapidly changing. The increased complexity and volumes of new satellites brings forward requirements for reduced cost, higher integration, use of c*ommercial-off-the-shelf* software and higher performance. The increasing development pace of microelectronic technology provides a further complication; the life-time of semiconductor processes is decreasing to a point where it is becoming difficult to guarantee the long-term component support required by many space programmes. In addition, demand for military components have decreased significantly since the end of the *Cold War,* reducing the number of foundries providing radiation-hard components and manufacturing services.

The objective for the LEON processor is to meet the requirements for performance, availability and low cost by the use of commercial standards, design techniques and semiconductor technology. The following design goals were defined for the processor:

- *Use of commercial semiconductor process*. To reduce cost and increase performance, it should be possible to implement the processor on commercial, single-event upset (SEU) sensitive semiconductor processes.

- *Portability*. To guarantee long-term availability, the processor should be portable across wide range of semiconductor processes with minimum cost and effort, while maintaining functionality and performance.

- *Modularity*. The processor implementation should allow reuse in *system-on-a-chip* (SOC) designs.

- *Scalability*. The processor should be usable in both low-end and high-end applications with minimum hardware and software overhead.

- *Standard interfaces*. The processor should have standardized interfaces to simplify system integration and to reuse commercial cores, components and tools.

- *Software compatibility*. The processor should be compatible with both the currently used ERC32 software development tools and COTS software packages.

The target performance is 100 MIPS (peak) at 100 MHz, with a power consumption of less than 1 Watt.

3. LEON architecture

The LEON processor is based on a newly developed SPARCV8-compatible [6] integer unit, featuring a 5-stage pipeline, hardware multiply and divide units, dual co-processor interfaces, and separate instruction and data buses (Harvard architecture). The reason to choose the SPARCV8 architecture is two-fold: to maintain software compatibility with ERC32, and to avoid any licensing issues regarding a re-implementation of the instruction set (SPARC is an open architecture and can be used without license). A high-speed AMBA [7] AHB bus is used for data transfer between the caches and the external memory controller. A low-speed AMBA APB bus is used to attach on-chip simpler peripherals such as timers, uarts, interrupt controller and I/O ports. A block diagram can be seen in figure 1 below:
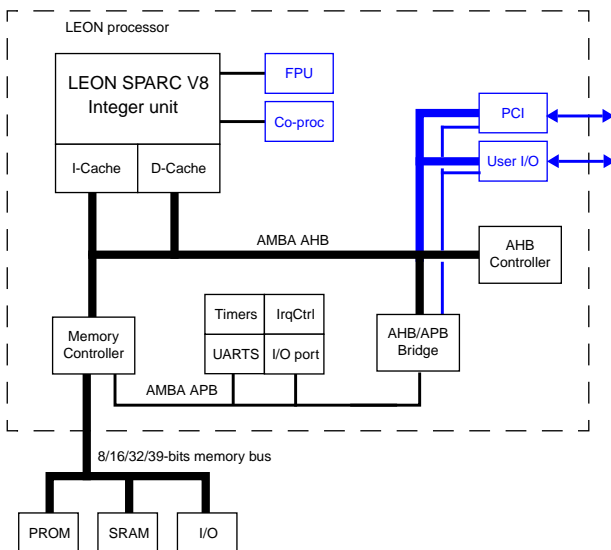

Figure 1: LEON block diagram

4. Adaptation to the space environment

4.1 Design goals

One of the main design goals for LEON was to be able to use a single-event upset (SEU) sensitive semiconductor process, while maintaining correct operation in the space environment. Extensive tests on ERC32 [3] showed that error-*detection* is not enough to maintain correct operation without using spare computers or voting. To avoid the large overhead of spare units, the decision was taken to implement on-chip fault-tolerance to both detect *and* remove SEU errors. To be able to use unmodified COTS software, the error-removal must be fully software transparent, and with no (or negligible) performance impact. To maintain portability, the fault-tolerance functions must be implemented directly in VHDL without relying on SEU-hardened technology cells.

4.2 Initial analysis

For the first LEON design, only SEU errors originating from a direct hit in a register or memory element were taken into account. Tests have shown that SEU errors in combinational logic can propagate to a register on a clock edge, but the probability of such events is low [4].

Studying the distribution of sequential cells in the LEON design, they form three groups: single-port ram cells used for cache memories (tag and data), dual or triple-port rams used for the processor register file, and synchronous D-flip-flops used for all remaining storage such as state machines, pipeline registers and status/control functions. Different types of SEU protection has been used for these three groups, based on their usage in the system.

4.3 Cache memories

Large cache memories are vital for high performance, and are typically in the critical (timing) path of a processor. This is indeed true for LEON which due to portability reasons uses standard synchronous ram cells rather than specially-designed cache rams. To minimize complexity and timing overhead, the cache rams are provided with simple error-detection in form of one or two parity bits for each tag or data word. The parity bits are written simultaneously with the associated tag or data, and checked on each access. If a parity error is detected during a cache access, a cache miss is forced and the (uncorrupted) data is fetched from external memory. The data cache uses write-through policy, and a second copy of the data is thus always available. No timing penalty occurs since parity checking is performed in parallel with tag checking.

In dense ram blocks, it is possible that one SEU hit can cause multiple errors [10], typically in adjacent cells. If the bits in the ram block are geometrically organized as a matrix with one word per line, a multiple error could occur in the same word and potentially not be detected by one parity bit (a single parity bit can only detect odd number of errors). To handle such ram cells, LEON can be configured to use two parity bits per tag and data word, one for odd and one for even data bits. A double error in any adjacent cells can then be detected. In rams with data words geometrically interleaved (i.e. no adjacent bits belongs to the same word), one parity bit is sufficient to detect double errors.

4.4 Processor register file

The SPARC architecture uses registers in windows of 32 (16 overlapping). A typical 8-window implementation contains 136 32-bit integer registers and 32 32-bits floating-point registers. Most SPARC instructions uses two source and one destination operand, and the LEON processor therefore uses a three-port register file. To detect SEU errors, each word can be protected using one parity bit, two parity bits or a (32,7) BCH [9] checksum. The protection bits are generated in the write stage of the pipeline and written together with the corresponding data. The register file is read in the decode stage, but checking is done in the execute stage in parallel with instruction execution to avoid timing penalties in the decode stage. If a correctable error is detected (fig. 2C), the pipeline is flushed and the erroneous operand data is corrected and written back to the register file (instead of the erroneous instruction result). The pipeline is then restarted at the point of the failing instruction. The restart operation is identical to taking a trap with the exception that a jump is made to the address of the failed instruction rather than to a trap vector. The time for the complete restart operation takes 4 clock cycles, the same as for taking a normal trap. If an uncorrectable error is detected (fig 2D), a register error trap is generated. The implementation overhead for the pipeline restart is small since the logic for normal trap handling is used. Figure 2B shows the pipeline behavior for a normal trap.

The register file has two read-ports, and two error-detection units are implemented. The register file has one write port, and only one correction unit is needed. This means that if more than one correctable error occurs, the instruction will be restarted once for each error, correcting and storing one register value each time. In worst-case, a double-store instruction that use four individual registers can be restarted up to four times, correcting one register value at a time.

Most standard-cell ram-libraries do not include three-port rams, and the register file is then implemented as two parallel two-port rams with the write-ports connected together. In such case, the cheaper parity coding can be use to not only detect errors but also to correct them. Both two-port rams have the same content, and when a parity error is detected on one read port (i.e. in one memory) error correction is performed by copying the value from the error-free memory to the failed memory. During the copy operation, the (presumed) error-free ram is also checked for errors, if an error is found an uncorrectable error trap is generated.

Although the implemented protection scheme (parity or BCH) has no timing impact, performance of double-store instructions can be slightly affected. This is because the write-buffer in the data cache will delay the request of the memory bus one clock cycle in order not to start any memory store cycle before the second store data word has been checked and (potentially) corrected.

A. Normal execution

FETCH	INST1	INST2	INST3	INST4				
DECODE		INST1	INST2	INST3	INST4			
EXECUTE			INST1	INST2	INST3	INST4		
MEMORY				INST1	INST2	INST3	INST4	
WRITE					INST1	INST2	INST3	INST4

B. Normal trap operation (INST2 trapped)

FETCH	INST1	INST2	INST3	INST4	INST5	FLUSH	TA1	TA2
DECODE		INST1	INST2	INST3	INST4	FLUSH		TA1
EXECUTE			INST1	INST2	INST3	FLUSH		
MEMORY				INST1	TRAP	FLUSH		
WRITE					INST1	TRAP		

C. Regfile error detection/correction (INST2 restarted)

FETCH	INST1	INST2	INST3	INST4	INST5	FLUSH	INST2	INST3
DECODE		INST1	INST2	INST3	INST4	FLUSH		INST2
EXECUTE			INST1	CHECK	INST3	FLUSH		
MEMORY				INST1	CORR.	FLUSH		
WRITE					INST1	UPDATE		

D. Uncorrectable regfile error, error trap

FETCH	INST1	INST2	INST3	INST4	INST5	FLUSH	TA1	TA2
DECODE		INST1	INST2	INST3	INST4	FLUSH		TA1
EXECUTE			INST1	CHECK	INST3	FLUSH		
MEMORY				INST1	ERROR.	FLUSH		
WRITE					INST1	TRAP		

Figure 2: Pipeline operation during traps and errors

4.5 Flip-flops

The processor contains approximately 2,500 flip-flops, used for temporary storage and state machines. To protect against SEU errors, each on-chip register can be implemented using triple modular redundancy (TMR). The flip-flops are continuously clocked, and any SEU register error will automatically be removed within one clock cycle while the output of the voter will maintain the correct (glitch-free) value. To further increase robustness, each of the three lanes of the TMR registers can have separate clock-trees (figure 3). An SEU hit in one clock-tree can therefore be tolerated even if the data of a complete lane of 2,500 registers is corrupted. On the following clock edge, all errors will be removed when new data is clocked in. An SEU hit in the (single) clock pad is not tolerated since it will propagate to all three clock-trees. However, the large feature size and capacitance in the main clock buffer makes such an event unlikely.
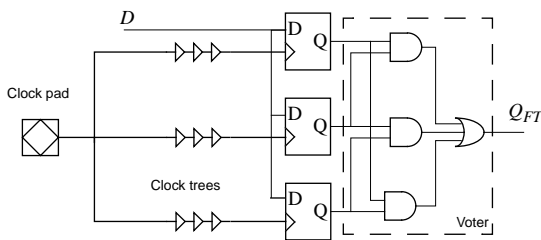

Figure 3: TMR register with separate clock trees

4.6 External memory

The external memory is protected using an on-chip EDAC (error-detection and correction) unit. The EDAC implements a standard (32,7) BCH code, correcting one and detecting two errors per 32-bit word. Error-detection and correction is done during the re-fill of the caches without timing penalties. The caches operate in parallel with the processor, and parts of a cache line can be filled on speculation without being requested by the processor. It is therefore not always desirable to signal an uncorrectable EDAC error to the processor since the failed data might in fact never be used. To solve this problem, the caches implement sub-blocking [8], with one valid bit per 32-bit data word. If an uncorrectable error is detected, the corresponding valid bit is not set, but the remaining of the cache line is still filled. If the processor accesses the failed data (or instruction), a cache miss will occur since the valid bit is not set. The data is re-fetched and the memory error is propagated to the processor which will take a corresponding data/ instruction error trap. The taken trap will then always be

'precise', and software recovery by roll-back/roll-forward is simplified since no instructions have been executed beyond the error.

4.7 Master/Checker mode

The processor implements a checker mode, which allows two LEON processors to operate concurrently in master/checker configuration. In the checker mode, all outputs are disabled and the internal values which should have been driven are compared to the values which are driven by the master device. A discrepancy will assert an error output on the checker device. This feature can be used in applications with extremely high requirements on error-detection, but is mostly used during SEU testing [5]. Note that the correction of register file or cache memory errors will also result in a master/checker error since the execution in the two processor will be skewed. This limits the usage of the master/checker configuration, since a reset is necessary to synchronize the two processors.

4.8 Software considerations

The data in caches and register file is only checked for errors when accessed, and the probability of undetected multiple errors will increase if stored data is not regularly used. Most tasking kernels (such as VxWorks or RTEMS) writes all active register windows to the stack on each task switch, which will automatically correct any latent errors in the process. Error build-up in the register file should thus not be a problem. The caches are not flushed on task switches, but with a reasonably large program, all cache contents should be regularly replaced or accessed, and any latent errors over-written. There is however no easy way of determining for how long a cache line might stay active in the cache without being accessed. In small programs, a cache flush could therefore periodically be performed to force a refresh of all cache contents.

5. Implementation

5.1 Design style and portability

The LEON processor is implemented as a high-level VHDL model, fully synthesisable with common synthesis tools such as Synopsys, Synplify and Leonardo. The model is extensively configurable through a configuration package. Options such as cache size and organization, multiplier implementation, target technology, speed/area trade-off and fault-tolerance scheme can be set by editing constants in the configuration package.

The only technology-specific cells used in the design are ram mega-cells for cache memories and register file, and

pads. To keep the design portable, a package with wrappers is made for each technology, providing a uniform interface between the processor and the custom technology cells. Porting LEON to a new target technology only consists of making a new wrapper package for the target technology. The design is fully synchronous, has only one clock, and uses only registered inputs and outputs. This makes synthesis simple, and improves portability further.

5.2 Synthesis overhead - Atmel ATC25

Making the model highly configurable makes it possible to quickly analyze the impact of the fault-tolerance functions. In table 1 below, the synthesis results for an FPU-less LEON configuration is shown for the Atmel ATC25 (0.25 µm) CMOS technology. The the core area is given for a standard configuration versus a configuration with fault-tolerance. The two configurations are functionally identical but the fault-tolerant version uses TMR on all flip-flops, 2 parity bits on the cache rams, and 7-bit BCH code on the register file.The area overhead for the LEON core without ram blocks is around 100%. This was expected since a TMR cell is approximately 4 times the size of a normal flip-flop (3x flip-flops + voter), and a non-TMR configuration uses 20% of the area for flip-flops. The overhead including ram cells is only 39% since the overhead for parity and BCH checkbits is lower. This particular configuration is heavily pad-limited on a 0.25 µm process, and if manufactured, the area overhead at chip level would in fact be 0%. The timing penalty for the fault-tolerant version is the extra delay through the TMR voter, approximately two gate-delays or 8% of the cycle time.

TABLE 1. LEON synthesis results on Atmel ATC25

Module	Area (mm^2)	Area incl. FT	Increase
Integer unit (+ mul/div)	0.86	1.61	87%
Cache controllers	0.17	0.35	105%
Peripheral units	0.45	0.90	100%
Register file (136x32)	0.19	0.24	26%
Cache mem. (16 Kbyte)	2.42	2.59	7%
Total	4.09	5.69	39%

5.3 LEON-Express

The first silicon implementation of LEON was made on the Atmel ATC35 process. The device was code-named 'LEON-Express' and was manufactured in January 2001. A standard-cell library without any SEU hardening features was used. The chip is roughly 40 mm^2 and operates at 50 MHz (military temperature, worst-case conditions). The purpose of the LEON-Express device was to validate the

operation of the LEON processor and demonstrate the implemented fault-tolerance techniques, it will not be commercialized. Figure 4 shows the floorplan. The device is pad-limited and to minimize manufacturing costs, only three metal layers were used.
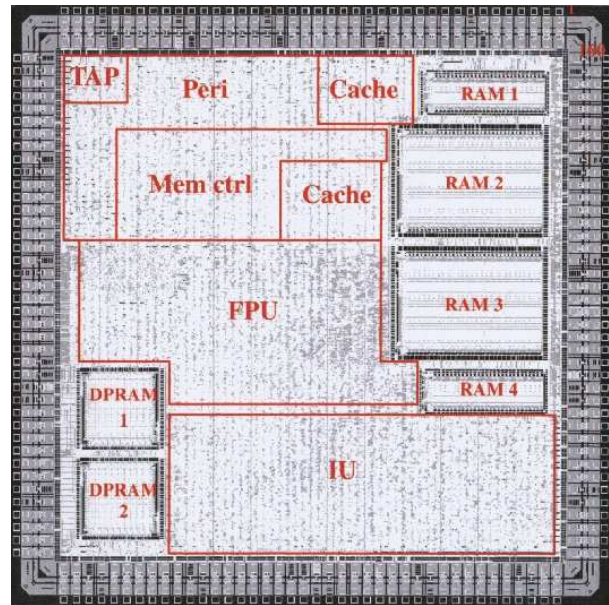

Figure 4: LEON-Express floor-plan

6. Heavy-ion error injection

To test the SEU protection methods, the LEON-Express device was submitted to heavy-ion error injection at the Louvain Cyclotron Facility in Belgium. The purpose of the tests was to measure the SEU sensitivity of the device to ions at different energy levels, and to assess the efficiency of the fault-tolerance logic. A test board with two LEON devices was designed, connected in master/checker mode. The built-in master/checker comparators of LEON makes it possible to run the device at full speed and yet compare the outputs on each clock cycle. Figure 5 below shows the SEU test board (checker device not mounted).

Three type of test programs were used: IUTEST that continuously checks the register file and caches memories for errors, PARANOIA that checks the FPU operation, and CNCF which is based on real spacecraft navigation software. Each test program is self-checking and calculates a checksum of all operations that are made. The register file and cache memories are provided with on-chip error-monitoring counters that increment automatically after each corrected SEU error. The test software continuously reports the value of these counters to an external host computer that

counts the number of errors in each ram type, as well as any software checksum errors.

During the heavy-ion injection, the master device was submitted to the ion beam while the compare error signal from the slave was monitored for compare errors. When a compare error is detected, the current software cycle is completed and the checksum is verified to control that correction has been done successfully. The error counters are also inspected to verify that the compare error originated from a correction operation and not from an undetected (and uncorrected error).

The first round of tests were made with effective Linear Transfer Energies (LET) between 6 and 110 MeV, using ion fluxes from 75 - 400 ions/s/cm^2. During each run, 10E5 particles were injected into the device. The number of resulting errors are shown in table 2 below. The ITE stands for instruction cache tag error, IDE for instruction cache data error, DTE for data cache tag error, DDE for data cache data error, and RFE for register file error.

No undetected errors or other anomalies occurred and a total of 4,500 errors were detected and corrected. The cross-section (SEU sensitive area) depends on software activity and ion LET, and the maximum (0.1 cm^2) was measured when running the IUTEST program using an LET of 110 MeV. Compared to the ram size of 10 mm^2, this means that 10% of the ram cell area is sensitive to SEU hits. The cross-section for the flip-flops could not be measured since no SEU monitoring capability is implemented in the TMR cells. Limiting the flux to 400 ions/s/cm^2 during the first round of tests was necessary to be able to count all errors accurately. A reset and re-initialisation of the test system takes around 50 milli-seconds, and must be significantly lower than the average error rate.

Having obtained data to determine the cross-section, additional test were made at an ion flux between 2,000 - 5,000 ions/s/cm^2 using an LET of 110 MeV. At these levels, 20 - 50 errors/second occurred in the ram cells. Several runs

Table 2: LEON-Express SEU error, runs of 10E5 particles

| TEST	LET	ITE	IDE	DTE	DDE	RFE	Total	X-sect
IU	5.86	10	35	13	44	0	102	1.02E-03
IU	8.27	16	58	12	84	6	176	2.04E-03
IU	14.1	24	124	31	113	27	319	3.19E-03
IU	14.1	33	142	32	124	35	366	3.66E-03
IU	28.2	50	227	59	245	44	625	6.25E-03
IU	55.9	50	170	53	220	42	535	5.35E-03
IU	110	83	318	103	440	71	1015	1.02E-2
PAR	14.1	30	65	13	12	25	145	1.45E-03
PAR	28.2	34	81	15	28	36	194	1.94E-03
PAR	55.9	45	94	27	31	45	242	2.42E-03
PAR	110	95	157	50	67	76	445	4.45E-03
CNCF	14.1	23	44	17	51	28	163	1.63E-03
CNCF	14.1	28	49	27	70	23	197	1.97E-03

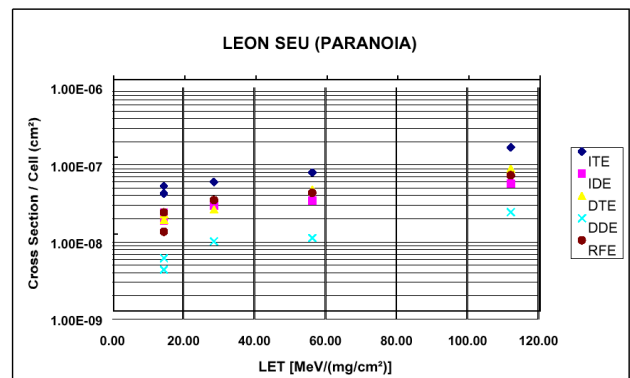Figure 5: LEON-Express SEU test board
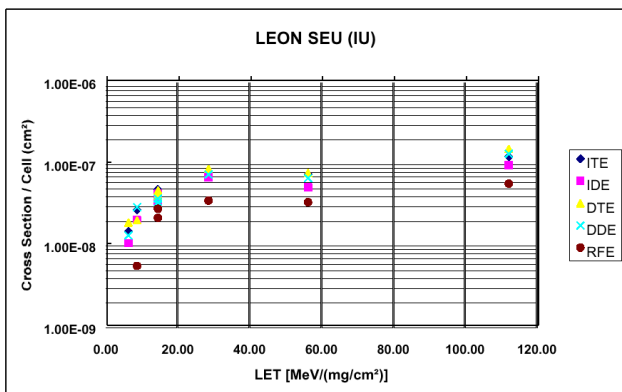
Figure 6: Cross-section vs. LET, IUTEST

Figure 7: Cross-section vs. LET, PARANOIA

were made using all three test programs, injecting 10E6 and 10E7 particles (10E4 - 10E5 errors) per test run. The CNCF and PARANOIA test programs executed without undetected errors, but the IUTEST showed on average 5 error traps or software failures per 10E7 particles. Ion fluxes below 2,000/s/cm^2 did not give any failures, and it is believed that the undetected errors were due to multiple-error build-up in the caches. Worst-case condition for ion flux in the space environment is many magnitudes lower, and this effect is thus not considered a problem. The cross-section per bit for the different ram types is plotted in figure 6 and 7.

7. Alternative implementations

LEON is by no means the first implementation of an fault-tolerant RISC processor, previous implementations includes the IBM S/390 G5 [11] and the Intel Itanium [12]. The IBM processor solves error-detection by duplicating the complete pipeline until the last write-stage, in which the result from the two pipelines is compared. In case of a discrepancy, the state of the processor is not updated and the pipeline is restarted at the point of the falling instruction. The area overhead is similar to LEON, 100%. The IBM scheme is better in the sense that timing is not affected by a TMR voter and that all types of errors are detected, not only soft errors in register. The scheme is worse from a real-time point-of-view since restarting of the pipeline takes several thousand clock cycles. The scheme can also only be used where (functional) timing is not important, bus interfaces or timer units can not use this scheme without loosing their function.

The Intel implementation a mix of ECC and parity codes to detect and correct soft errors in caches and TLB memories. State machine registers are not protected.

8. Conclusion

Well-know error-detection and correction techniques such as parity, BCH and TMR have been used to implement an SEU-tolerant processor on a non-hardened semiconductor process. By choosing the appropriate detection and correction method for each specific memory type, the area overhead has been kept low. Fault-injection using heavy-ions has proved the efficiency of the fault-tolerance concept, although some anomalies were detected at high particle fluxes. The portable design style and simple synthesis method insures long-term availability and quick access to new semiconductor processes.

9. Future directions

During 2002, the LEON processor is planned to be implemented and manufactured on the Atmel ATC25 pro-

cess (0.25 μm CMOS). The ATC25 device will have larger caches than the ATC35 version, and include additional functions such as a PCI interface and an on-chip debug unit. The final die size will be around 20 mm^2 (pad-limited) and the device is planned to operate at 100 MHz.

Although no indications of combinational SEU errors were seen for the ATC35 device, the separate clock trees for the TMR cells makes it possible to form a pulse filter on the inputs to the flip-flops. By skewing the three clocks, any pulse shorter than the skew would only be latched by one of the flip-flops in the cell, and be removed by the voter. The feasibility of such a scheme will be further investigated.

10. Acknowledgements

11. References

[1] J.Gaisler, "Concurrent error-detection and modular fault-tolerance in an 32-bit processing core for space applications", FTCS-24, June 1994 (Austin, USA).

[2] J.Gaisler, "Evaluation of a 32-bit microprocessor with built-in concurrent error-detection", FTSC-27, June 1997 (Seattle, USA)

[3] J.Gaisler & T.Vardanega, "Lessons Learned from the Implementation of On-Board Tolerance to Physical Faults in Ada", The International Journal of Computer Systems: Science & Engineering, January 2000.

[4] P.Linden et al., "On latching probability of particles induced transients in combinatorial networks", FTCS-24, 1994.

[5] R.Koga et al., "Techniques of microprocessor testing and SEU rate prediction", IEEE Trans Nucl. Sci., NS-32, 1985

[6] "The SPARC Architecture Manual Version 8", SPARC International, Prentice Hall, 1992

[7] "AMBA Specification, version 2.0", ARM-IHI 0011A, ARM Limited, 1999.

[8] J.Handy, "The Cache Memory Book", Academic Press, 1993.

[9] C.L.Chen et al., "Error-correcting codes for Semiconductor Memory Applications: A-state-of -the-art-review", IBM J Res Development, page 124-132, March 1984.

[10] J.A.Zoutendyk et al., "Characterization of Multiple-Bit Errors from Single-Ion Tracks in Integrated Circuits", IEEE Trans Nucl. Sci,. December 1989.

[11] M.A.Check et al. "Custom S/390 G5 and G6 microprocessors", IBM Journal of Research and Dev., Vol 43, No 5/6, 1999.

[12] N.Quach, "High availability and Reliability in Itanium Processors", IEEE Micro, Vol 20, No 5, 2000.