Towards a Secure and Reliable System

Portolan Michele, Leveugle Régis

TIMA Laboratory 46, Avenue Félix Viallet - 38031 Grenoble Cedex – FRANCE
Michele.Portolan@imag.fr; Regis.Leveugle@imag.fr

Abstract. In this article we describe a system based on a 32-bit processor, Leon, complete with security features offered by a specific cryptographic AES IP. Hardening is done not only on the principal hardware components but on the operating system as well, with attention for possible interaction between the different levels. The cryptographic IP is protected too to offer good resistance against, for example, fault-based attacks.

1 Introduction

Nowadays the application range of computing systems is steadily increasing, going well beyond classical applications, like for instance desktops or mainframes. This usually comes together with the search for extreme miniaturisation and low power, meaning that the resulting electronic circuits can be seriously affected by transient faults, once relegated to especially harsh environments (space, nuclear facilities, etc…). It is therefore important to provide hardening as soon as possible in the development process to assure good performances and fault coverage with contained costs. The usual hardening techniques, based on post-synthesis netlist modifications, can be really demanding in terms of overhead and, especially, development and verification time. One solution is to harden the design already at conception time, acting on the behavioural high-level description (usually in VHDL). These interventions can be a little more complex than their gate-level homologues, but have the great advantage of being directly verifiable in behavioural descriptions, for example through simulation, which is much lighter than post-synthesis validations. The result is the possibility of validating at an early stage the robustness of the protected system, instead of waiting after synthesis. Moreover it is possible to exploit the configurability features of these high-level languages to obtain solutions directly applicable to a large range of possible configurations, while gate-level hardening must be individually done for each netlist. The contribution of this paper is to illustrate ona practical case the feasibility and efficiency of this approach.

In this article we firstly introduce the concept of transient faults in section 2, pointing out the most common causes and fault models. Then we introduce a secure embedded system, detailed in section 3, whose hardening is then discussed in section 4. Section 5 evaluates the robustness level obtained, while lastly section 6 points out future developments.

2 Natural and Provoked Faults

Transient faults are perturbations of the normal operation of the circuit that if not properly handled could cause erroneous results, with serious consequences. There can be many causes, like for instance the impact of a charged alpha particle causing an ionization of the channel, or the crosstalk between two adjacent wires. Electromagnetic fields could also alter the thresholds of transistors, while strong noises on the input signals may cause incorrect information to be sampled. Miniaturisation means higher density of the circuit, making it more sensitive to particle impacts and crosstalks, while low power means low working voltages and currents, making electromagnetic perturbation and noise effects more important too. Handheld devices for instance are especially exposed: on the one hand battery life (low power) and size (miniaturisation) can become key factors for the success of a product, while on the other hand they are even more sensitive to interferences while working near to other components. Anyone, for instance, has experienced the problems a mobile phone ringing can provoke to a stereo or a television.

Regardless to their cause these faults can be usually modelled as unwanted transitions either in the combinatorial logic (the result of the operation is not correct) or in the sequential logic (the value stored in a flip-flop is modified): if a single signal is modified they are called Single Event Transients (SETs) and Single Event Upsets (SEUs) respectively. They alone

can model a great part of natural faults, while combining them together it is always possible to investigate more complex phenomena that involve errors with higher multiplicity.

As sensitive information is increasingly stored in portable devices (addresses, telephone numbers, credit card identifiers, etc …) there is also a rising need for security inside systems, often equipped with cryptographic units. In later years the emergence of fault-based attacks on cryptographic systems has cleared a point: reliability is important for secure systems too. The principle of a fault-based attack is to deliberately cause a fault in the system in order to obtain sensitive information that should have been kept secret. A famous example is the attack on the RSA algorithm implemented with the Chinese Reminder Theorem exposed by Lenstra in [1]: with one or two well-timed faults the circuits directly yields the secret key.

There is a great range and variety of fault-related attacks on secure systems: a complete panorama is out of the scope of this article, so we will just give a brief summary. In general terms attacks can be divided in three categories, based on their approach to the system they want to violate: invasive attacks, non-invasive attacks andsemi-invasive attacks.

Invasive attacks are based on physical modifications of the system, like inserting probes, cutting lines or short-circuiting some signals.

Non-invasive attacks do not physically modify the system but try to infer sensitive information either by observing its normal behaviour (side-channel attacks) or by perturbing it with the insertion of voluntary faults (fault-based attacks). Side-channel attacks exploit information that "leaks out" of the system by comparing runs on different loads of data. The most important are Timing Attacks (TA, time difference in calculations with different data), Simple and Differential Power Attacks (SPA and DPA, power consumption analysis) and Electromagnetic Attacks (EMA, analysis of the electromagnetic emissions by the system). On the other hand fault-injection attacks presume the active intervention of the hacker who deliberately inserts a fault at a given instant to disrupt the normal execution of the cryptographic algorithm. If the injection instant is well calibrated the result can be quite effective, as previously illustrated for RSA.

Faults can be injected in a variety of ways, usually either by inserting glitches on the power source or the clock or by flashing the device (either the entire chip with a photolamp flash or more selectively with a laser beam).

Lastly semi-invasive attacks are between the two: sometimes in order to perform attacks it is necessary to prepare the circuit, usually decapsulating it from the protective resin to expose the target area to the light. As the chip itself is not modified such attacks are conceptually similar to the non-invasive ones.

The attention is put more on selective attacks provoking limited errors because large-scale ones can be easily blocked by sensors put on the chip that reset the system when, for example, a flash of light is detected. These devices on the other hand become useless against, for instance, a laser focalised on specific point of the system. This category of faults can once more be modelled in terms of SETs and SEUs, so the protection from natural and provoked faults can be merged. Note that when the fault model requires more than one error occurring simultaneously it may be possible in some cases to account for as many concurrent SETs/SEUs as needed.

In the following we will therefore focus on protection against SETs and SEUs, that may be either natural or provoked, without distinguishing further the two cases. Some multiple error configurations are implicitly also covered by the proposed implementation, but a complete protection against multiple-bit errors will not be discussed here.

3 Proposed Prototype System

3.1 System Overview

The system we propose, illustrated in Fig. 1, is based on a reconfigurable RTL processor core, Leon 2 [2], equipped with a cryptographic coprocessor implementing AES to provide security. The system runs eCos, an embedded Operating System (OS) by Red Hat [3], thus obtaining a complete platform well representative of Systems on Chip (SoCs). A real application could be obtained directly implementing it, while vice versa any typical single-processor system could easily be retraced to our example, making it possible to apply the proposed strategy on it.

3.2 Leon processor

Leon is a SPARC V-8 compliant 32-bit RISC processor that is freely available on the net ([2]) as a VDHL synthesisable model covered by the GNU GPL. Such a format allows the processor to be freely tailored-out to suit the needs of the

particular application, as well as being target of reconfiguration operations like those proposed in [4]. A block diagram of the processor is reported in Fig. 2. The centre of the processor is a 5-stage Integer Unit (IU) implemented with cache-level Harvard architecture (i.e. separate caches for data and instruction, but common central memory). Around that block many entities can be instantiated to adapt the processor to the designer needs, notably an optional multiplier, a Debug Support Unit (DSU) to allow on-line debug through a serial link, a Memory Management Unit (MMU), an SDRAM controller, a Floating Point Unit (FPU), support for a Coprocessor (CP), PCI and Ethernet interfaces.

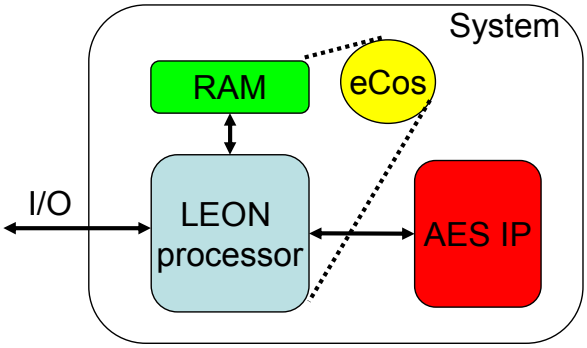

Fig. 1. Scheme of the proposed prototype system: the three main components (Leon processor, AES cryptographic IP and Operating System) and their interconnections are shown.

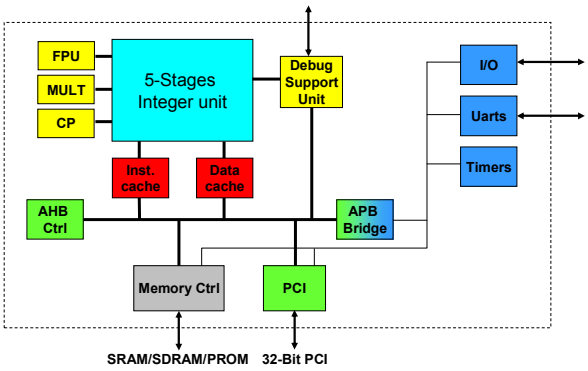

Fig. 2. Leon 2 Processor Block Diagram: note in particular in the upper left-hand corner the Integer Unit and the two caches (instruction and data).

From our point of view Leon offers many interesting points:

- it is well-known, widely tested and centrally maintained, offering a good applicability range;
- the GNU GPL allows us (almost) complete freedom of intervention on the source code;
- configurability is a key feature of the project;
- the core is build around two AMBA-compliant buses [5] (an AHB and an APB), making it easy to add new components.

We exploited these characteristics to enhance the processor and add new features, such as hardening and cryptographic capabilities. Leon has been developed with space applications in mind and so has already undergone hardening stages, resulting in the Leon-FT [6] that is also the base for a commercial product by Atmel, named AT697E [7]. Anyway its sources are not public, making impossible to directly use it: we decided to develop our own protections using the Leon-FT as an inspiration and comparison model.

3.3 AES Cryptographic IP

Portable (and therefore ubiquitous) systems are being used nowadays also as terminal for secure transactions, like for example biometric identification devices or bank payments. An example can be the Thumbpod presented in [8], where a Leon is coupled to a fingerprint scanner and an AES core to obtain a portable biometric identification unit. High connectivity capability (usually wireless) is becoming a standard requirement and, as recent years have shown, this too cannot come without proper security, especially when it comes to communications between peer devices. Unfortunately the two most classical wireless communication standards, Bluetooth and IEEE 802.11, have both great problems in that field, as can be seen for example in [9]: a good way to counter weaknesses in the standard is to boost up security in devices.

Security features have been added to our system by providing an optional cryptographic IP, realising AES encryption/decryption. Plugged on the APB bus of the Leon it permits fast secure operations, like for instance digital signatures or transactions: with that equipment our system is up to the challenge, remaining open to new cores implementing different algorithms. The core can be configured to support keys of 128, 192 or 256 bits (or all three) and the S-Boxes can be either hardwired in ROM or implemented in RAM (for Virtex II only). The IP has been synthetised with Leonardo for various configurations, the most representative of which are reported in Table 1 for the ROM-based implementation and in Table 2 for the RAM-based one.

Table 1. Synthesis Results for AES IP on Virtex-II, S-Boxes implemented in ROM

Key Size (bits)	Clock Cycles (Coding)	Clock Cycles (Decoding)	Max Clock (Mhz)	Hardware complexity	
				FG	RAMs
128	70	130	44.29	8160	0
192	82	122	45.85	9460	0
256	104	142	42.86	9836	0
All	***	***	45.23	11692	0

***: depends on key size, equivalent to precedent results

Table 2. Synthesis Results for AES IP on Virtex-II, S-Boxes implemented in RAM

Key Size (bits)	Clock Cycles (Coding)	Clock Cycles (Decoding)	Max Clock (Mhz)	Hardware complexity	
				FG	RAMs
128	580	642	56.66	4454	16
192	584	624	47.98	5752	16
256	610	654	56.21	6124	16
All	***	***	47.28	7610	16

***: depends on key size, equivalent to precedent results

Both implementations offer good results, in particular the ROM-based one has really good timing performances. On the other hand the RAM-based version can reduce the occupied area by almost half and the maximal frequency is some 10% higher, but it causes a great timing loss in terms of cycles per operation (a factor varying from 6 to 8 depending on the implementation). Probably it pays heavily the access protocol used by the internal RAM the S-box are implemented in, called up for each calculation. Better results could be obtained using more performing or specially-tailored RAMs on ASIC targets. For example burst access could come useful if the data in the S-Box were cleverly placed. This version can nevertheless be interesting for applications where AES timing is not critical, like for instance an identification device (needing only a few calculations). More heavy time-critical applications, requiring for example a coder for data streaming, would prefer the ROM-style.

3.4 eCos Operating System

eCos is a multithread operating system having configurability and portability as its key features. On the one hand these features enabled us to easily modify it following our hardening strategy, as explained in section 4.3. On the other hand its good portability offered us a fully working Leon version, with the possibility of exploiting the other ports to instantly transfer it on another system if needed.

4 Robustness

4.1 Hardening Strategy

The system implements a protection scheme that has in the Operating System its fulcrum, but without forgetting the importance of decentralised control. In fact Fig. 3 shows the central place of the OS in the system architecture. This can be exploited to receive the detection/error signals generated by both hardware and software protection and handle them if the local hardening devices are not able to do it by themselves.

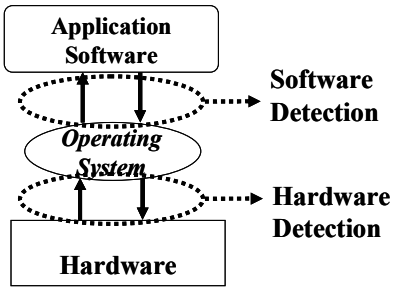

Fig. 3. General Protection Scheme. The Operating System's central position is exploited to make it the arbiter of fault tolerance exploiting detections implemented either in software or in hardware.

The following paragraphs (4.2 to 4.4) describe the protections implemented in the system that are then harmonised following our protection scheme. We can find 3 types of errors depending on the handling strategy: locally-handled, centrally handled or mixed. Locally-handled errors can be directly and easily corrected by the local hardening devices, like for example SEUs inside the cache memories or SETs in the execution pipeline (see 4.2). Centrally handled errors are errors that cannot be resolved locally and need to call up higher-level constructs to handle them. An example can be SEUs in the pipeline registers or SETs in cache controllers or in the detection-only pipeline (again 4.2) that can be used as trigger for a Context-Switch based rollback (see 4.3). Lastly mixed errors are errors that even if resolvable locally can be passed on to higher-level constructs for further treatment. An example are errors inside the cryptographic IP: even if the potential attack can be hindered by locally resetting the IP it could be useful to pass the detection information to the OS to, for example, erase other sensitive information from the memory or generating an alarm.

4.2 Leon hardening

In a processor usually the more sensitive units to assure correct operations are the Integer Unit (IU) and the caches. As Fig. 2 shows they are the central parts of Leon, present regardless of the chosen configuration: from now on we will refer to the Integer Unit and the cache system as the "processing unit" to highlight that role. Their impact on performances and execution correctness varies with the application, but techniques tested on them can be easily applied to other components. Moreover they represent two different entity types: the first one (the cache) with little logic and a lot of memory and therefore greatly affected by SEUs, the latter (the IU) mainly composed by logic, more touched by SETs, with the exception

of the internal registers. For what concerns protection techniques as a guideline we search for low-cost, low-impact solutions, avoiding extensive replication like TMR, concentrating on protection codes like parity, simple but representative.
The internal registers of each unit can be affected by SEUs: that is the reason why they are separately protected. A parity bit is calculated at writing time and checked at each cycle in parallel of normal computation, generating a validation signal. Being that calculation done in parallel with normal computation the critical path is usually not impacted, while the overhead is limited to some flip-flops to store parity and a little simple logic to calculate the parity (in double rail): that technique has been applied to the internal registers of the pipeline, obtaining SEU protection as shown in Table 3.
The other important SEU-prone components are cache memories. Their size imposes a RAM-based implementation, so a slightly different approach is needed. We chose to replicate the protection of Leon-FT proposed in [6] and taken up for [7]: each cache word is protected by two parity bits (one for the tag field, the other for the data), calculated at write time and checked at each read operation. A direct recovery is possible by generating a cache miss and reloading the faulty data from the central memory, supposedly free from the same transient fault. This assumption holds becaude a write-through policy is used, having therefore the central memory always up-to-date. As Table 3 shows that this approach is especially effective for FPGA-based implementations, where cache memories are mapped on internal RAMs: usually these blocks are not completely exploited, leaving some unused bits that can be directly used to store parity without any overhead.

The technique we adopted to protect the pipeline againts SETs is parity prediction, presented in [10] and extended in [11]: The idea is to add a parallel prediction component that just gives the prediction of the parity bit P_{Pre}. If L is the logic to protect, to be efficient the prediction logic L_P should be much simpler in terms of both complexity (otherwise a direct duplication would be better) and speed (not to impact the critical path). Anyway it is I general quite difficult, if not impossible, to manually design L_P starting from L, so a different strategy, illustrated in Fig. 4, is needed. A replica L' is created and its outputs are fed into the parity coder C (a XOR tree), obtaining the predicted parity bit P_{Pre}. L_P is then composed of L'+C and the synthesis tool will simplify it. To verify the code we will simply have to calculate P from L and compare it with P_{Pre}: the checker would better be self-checking, for example in double rail. As can be seen in Fig. 4, some logic is still added, so we have to expect some frequency losses, quite moderated if L_P is well optimized.

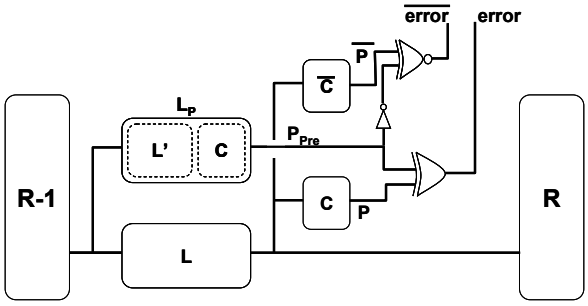

Fig. 4. Detection by parity prediction: the predicted parity bit P_{Pre} is obtained by the unit L_P=replica L'+C. The synthesis tool should be able to greatly simplify it, obtaining a much smaller overhead than with direct duplication

Tolerance can be easily achieved if one notices that almost all inputs/outputs are tied to internal registers. Fig. 5 shows the principle: in case of error the update of the registers is not validated and so at next clock cycle the logic will redo the same calculation, correcting the transient error.

That solution is a little more costly both in terms of frequency (outputs must wait error calculation before being validated) and of hardware (I/O must be buffered as they are not already saved in the internal registers). Anyway the overheads should remain much lower than with traditional replications.

This strategy has been manually applied to the pipeline, completely protecting it from SETs (parity prediction in the pipeline) and SEUs (parity check for internal registers), as can be seen in Table 4. It must be noted that synthesis is not done on the IU alone but on the higher-level entity: it is the only way to take into account the I/O buffers needed for recovery (see Fig. 5). This means that other little components are included too (notably the cache system), but they are kept constant in all configurations and their impact on overall results is not that noticeable.

The work done for the IU allowed us to define an automated algorithm to add parity prediction protection to any given entity that we successfully applied to the cache controllers (data and instruction), thus completing the protection of the cache system shown in **Erreur ! Source du renvoi introuvable.** with SET detection (parity prediction in the cache controllers).

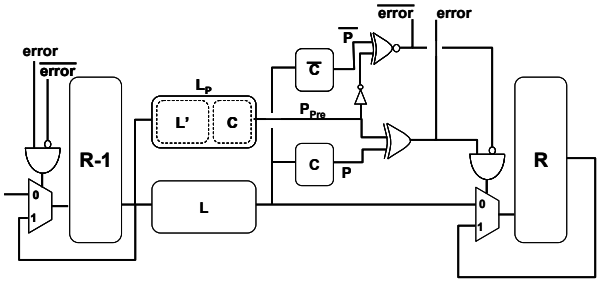

Fig. 5. Recovery by Parity Prediction: values in the registers are updated only if parity coding is respected, otherwise they keep the old value and calculation is done again, correcting the transient error.

The last protected element that is the general registers bank (not indicated in Fig. 2, but present in the more detailed Fig. 5). It simply is an array of memory elements, usually implemented directly as dual-port RAMs (the high number of 32-bit registers needed by the SPARC architecture makes flip-flop implementation infeasible). That means that they can be easily protected with the same strategy used for caches (see previous paragraph), so their protection does not add any new element to our analysis but simply improve coverage against SEUs.

Table 3. Synthesis results on Virtex II for cache memories and cache controllers hardening

	Slices		4 - LUTs	
Normal	833	--	1455	--
SEU Detection	833	(0%)	1507	(+3.6%)
SEU and SET Detection	918	(+10.2%)	1622	(+11.5%)
SEU Recovery	836	(+0.4%)	1511	(+3.8%)
SEU Recovery / SET Detection	940	(+12.8%)	1660	(+14.1%)

Table 4. Synthesis results on Virtex II for IU hardening (with cache system)

	Slices	**Slice Flip-Flops**	**4 – LUTs**	**Clk (MHz)**
Normal	2812	1258	4831	46.62
SET Detect	4696 (+67%)	1264 (+0.5%)	8319 (+72%)	45.99 (-1.4%)
SEU and SET Detect	4754 (+69%)	1289 (+2.5%)	8419 (+74%)	45.64 (-2.1%)
SEU Detect and SET Recovery	5239 (+86%)	1824 (+45%)	9274 (+92%)	25.50 (-45%)

4.3 OS-Level Hardening

The OS implements Checkpoint and Rollback (C&R) features for the tasks it runs following the scheme proposed in [12] and [13].

The base idea is to reuse the information stored by the OS at context-switch time (the operation of changing the task being executed by the system to achieve multitasking) to resume execution in case of error, as can be seen in Fig. 6. The state stored is usually incomplete, but it is possible to define clear application criteria (see above-mentioned articles for a complete analysis).

The result is a light, low-cost protection that enables recovery of any generic transient error detected during task execution. The hardware hardening operations detailed in the section above provide the detection signals that can be used to activate this scheme.

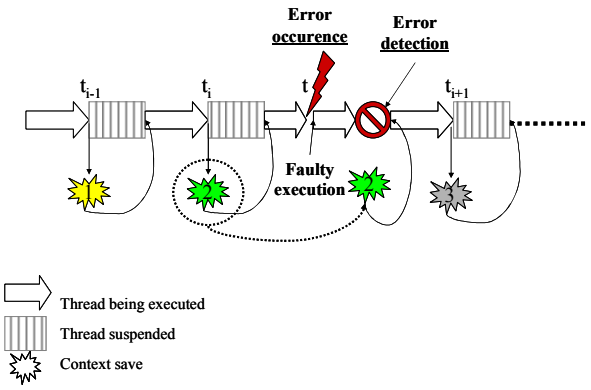

Fig. 6. Rollback to last context switch: information stored by the OS at context-switch time are also used to recover from an error, under some application hypotheses.

4.4 Cryptographic IP Hardening

The cryptographic IP is hardened to prevent fault-injection attacks. Once more we focus the attention on SETs and SEUs, adequately protecting the circuit.

SEUs may affect the sequential parts: the internal registers (noticeably the one containing the secret key) and the S-Boxes used to compute rounds (see [14] for details on the AES algorithm). Like for Leon caches each byte of the S-Boxes (implemented either in RAM or in ROM) is protected with a parity bit checked at each read operation. The secret key is protected in the same way, adding a parity bit for each byte, thus detecting SEUs changing its value.

SETs affecting the combinatorial logic are detected by timing redundancy. At the end of each round an inverse calculation is done: if the result is different from the original one an error is generated. The same principle is applied for the generation of the new key at each round. The good performances offered by the base IP (down to 70-100 clock cycles for a key size varying from 128 to 256 bits in ROM-style implementation) assure that the hardened IP's performance will be more than satisfying. Table 5 and Table 6 show the synthesis results for the hardened IP. Once the error has been detected the IP can either directly handle it by resetting itself (and stopping the possible intrusion) or generate an error signal toward the processor that will decide what to do. The situation is quite similar than before synthesis (see Table 1 and Table 2), with the ROM version having better performances in terms of cycles per operation but needing more area and slower frequency than the RAM-based one. It must nevertheless be noted how that timing gap between the two choices is shallower, reduced from some 10 Mhz to 1-2: the RAM implementation pays an heavier toll in terms of speed.

Table 5. Synthesis Results for hardened AES IP on Virtex-II, S-Boxes implemented in ROM

Key Size (bits)	Clock Cycles (Coding)	Clock Cycles (Decoding)	Max Clock (Mhz)	Hardware complexity	
				FG	RAMs
128	140	208	37.85	11136	0
192	156	178	40.27	12880	0
256	190	236	38.20	13761	0
all	***	***	39.63	15610	0

***: depends on key size, equivalent to precedent results

Table 6. Synthesis Results for hardened AES IP on Virtex-II, S-Boxes implemented in RAM

Key Size (bits)	Clock Cycles (Coding)	Clock Cycles (Decoding)	Max Clock (Mhz)	Hardware complexity FG	RAMs
128	636	698	42.94	6874	16
192	652	688	26,07	8542	16
256	688	748	43.76	9472	16
All	***	***	38.01	11321	16

***: depends on key size, equivalent to precedent results

5 Robustness Level

In this paragraph we will evaluate the good robustness level reached by the system presented in section 3 thanks to the hardening interventions detailed in section 4.

Taking as a reference Fig. 1 we can consider the AES block completely protected from transient faults, thanks to the characteristics of timing redundancy: it is almost impossible for the two runs to generate compensating errors. On the other hand some attention should be put on the state machine controlling it: faults causing an illegal jump to the last state could cause the validation of an illegal operation. It is the only situation that could cause an undetected error, as other unwanted state transitions would eventually cause wrong calculation results.

The Leon block is protected in its most important parts (see section 3.2): Fig. 7 shows a schema of the processor with the hardened components rayed. As can be easily seen the core of the Processing Unit is completely protected: the integer unit, the cache memories, the cache controllers and the register file are protected from SETs and SEUs. The limitations of a parity prediction scheme for combinatorial blocks is of course its ability to detect a transient fault only if it generates an odd number of bit errors on the outputs of the block. This can be ensured by using specific synthesis procedures [15], along with careful Placement&Routing . All operations inside the core are then protected.

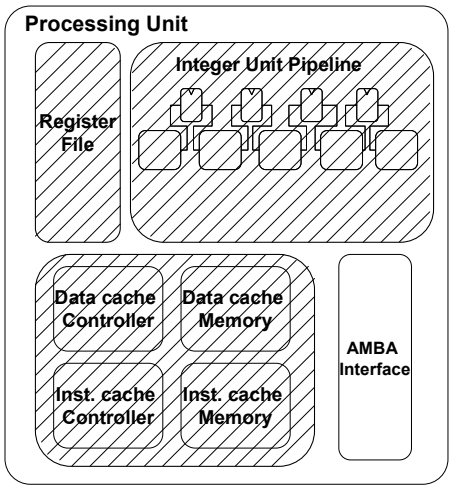

Fig. 7. Hardened components in the Leon processor. As can be clearly seen all components inside the Procesing Unit have been protected with our technique.

On the software side the OS gives Checkpoint and rollback capabilities to the software being run, correcting virtually any detected transient fault.

Lastly the RAM is not directly protected, but it is now possible to directly find on the market SEU-tolerant RAMs, usually exploiting Hamming codes.

6 Future developments

For what concerns security a secure RSA IP is being developed to be added to the system. With that addition it will be able to handle both symmetric secret-key (AES) and asymmetric public-key (RSA) operations, covering the two most important classes in cryptography algorithms. As an example it will be able to perform a PGP-like transaction: exchange of secret keys through a slow but secure public-key asymmetric RSA transaction to open a faster private-key symmetric transaction through AES (that requires both sides to now the secret key). Such a scheme can be well optimised, for instance, with a reconfiguration scheme like in [4]: each cryptographic IP can be an "ASIC-on-demand" and be loaded into the system only when needed.

For what concerns reliability a feasibility study is being conducted on the use of more complex correcting codes (Hamming, Reed-Solomon) to better protect values inside the register file, vital for the correctness of execution, or the possibility of using the proposed Checkpoint and Rollback scheme (see 4.3) to restore them in.

7 References

1. A. K. Lenstra, *"Memo on RSA Signature Generation In The Presence of Faults"*, private communication (available from the author), September 28, 1996
2. Gaisler Research Homepage, www.gaisler.com, regularly updated
3. Red Hat, eCos Homepage, http://sources.redhat. com/ecos/, updated on a regular basis.
4. C. Plessl, R. Enzler, H. Walder, J. Beutel, M. Platzner, L. Thiele, *"Reconfigurable Hardware in Wearable Computing Nodes"*, 6th International Symposium on Wearable Computers (ISWC) '02, pp 215-222, IEEE, 2002
5. AMBA Homepage, http://www.arm.com/ products/solutions/AMBAHomePage.html, regularly updated;
6. J. Gaisler, *"A Portable and Fault-Tolerant Microprocessor Based on the SPARC V8 Architecture"*, DSN 2002 pp 409-415, 2002
7. Atmel Rad-Hard Product portal, http://www.atmel. com/products/radhard/, regularly updated.
8. D. Hwang, P. Schaumont, Y. Fan, A. Hodjat, B. Lai, K. Sakiyama, S. Yang, I. Verbauwhede, "*Design Flow for HW/SW Accelleration Transparency in the Thumbpod Secure Embedded System*", 40th Design Automation Conference (DAC 2003), Anahiem, California, 2-6 June 2003
9. C.T. Hager, S.F. Midkiff, "*An analysis of Bluetooth security vulnerabilities*", Wireless Communications and Networking, 2003 (WCNC 2003), vol.3 pp1825- 1831, 16-20 March 2003
10. S.-B. Ko and J. –C. Lo, *"Efficient Realization of Parity Prediction Functions in FPGAs"*, Journal of Electronic Testing Theory and Applications (JETTA), Volume 20 Number 5 pp 489-499, October 2004
11. M. Portolan, R. Leveugle, *"A Highly Flexible Hardened RTL Processor Core Based on LEON"*, to be published at the Eight European Conference on Radiation and Its Effects on Components and Systems (RADECS 05), 19-23 September 2005
12. M. Portolan, R. Leveugle, "*Operating System Function Reuse to Achieve Low-Cost Fault Tolerance*", 10th International On-Line Testing Symposium 2004 (IOLTS 2004), Madeira, Portugal, 12-14 July 2004
13. M. Portolan, R. Leveugl*e, "A Context-Switch Based Checkpoint and Rollback Scheme*", XIX Conference on Design of Circuits and Integrated Systems (DCIS 2004), Bordeaux, France, 24-26 November 2004
14. NIST AES Homepage, http://csrc.nist.gov/CryptoToolkit/ tkencryption.html, regularly updated
15 N.A. Touba and E.J. McCluskey, *"Logic Synthesis of Multilevel Circuits with Concurrent Error Detection"*, IEEE Transactions on Computer-Aided Design, Vol. 16, No. 7, pp. 783-789, Jul. 1997