Robustness analysis of different AES implementations on SRAM based FPGAs

Uli Kretzschmar, Armando Astarloa, *Member, IEEE*, Jesús Lázaro, Unai Bidarte, Jaime Jiménez
Department of Electronics and Telecommunications
University of the Basque Country
Bilbao, Spain

***Abstract*—Common features for comparing AES implementations are the latency and throughput of the module as well as its resource requirements. This work evaluates the robustness against punctual errors in the FPGA caused by SEUs or other effects for a variety of AES implementations in order to provide a possible additional feature differentiating various architectures. The AES implementations included in this work span from a speed of more than one Mcycle for one encryption to 16 cycles per encryption. A fault injection flow is executed on the different implementations in order to determine their robustness against these punctual errors.**

I. Introduction

The Advanced Encryption Standard (AES) is a widely used means to encrypt and decrypt data which is going to be transmitted or received on a channel considered to be unsecure. But the need for encryption is not only limited to data transfer, but it is also necessary for other tasks like storing data in a secure way.

The Data Encryption Standard (DES) was a popular choice for such tasks for a long time. But the continuously shrinking of process technology together with advancements in computer- and FPGA architectures and their execution speed made it possible to crack the 56 bit key of DES using brute force with many PCs, like in the *Deschall-Project* [1], or FPGAs like in the *COPACOBANA* [2] codebreaker machine. As the successor of DES, the advanced encryption standard was selected in an open call for proposals. AES, which is a subset of the Rijndael algorithm, supports a block size of 128 bit, which is also called state, and key sizes of 128, 192 and 256 bit. Depending on the key size a different number of rounds needs to be executed, where one normal round consists of four steps (Subbytes, Shiftrows, Mixcolumns and Addroundkey). The complete AES algorithm is described in detail in [3]. It was selected from many other proposals because of its high security combined with a low effort of implementation in soft- and hardware.

When implementing AES in harsh environments effects of radiation induced faults have to be considered. These faults are so called Single Event Upsets (SEU) and are a well known effect in SRAM based FPGA architectures. These SEUs are especially relevant for in orbit applications but become more and more relevant to ground level applications as well, which are not exposed to a high degree of radiation. But the advantages of FPGAs such as short time-to-marked, low engineering and device costs for low volume productions and the possibility of reconfiguring the device in the field (or in orbit) make FPGAs especially interesting for usage in radiation exposed environments like communication- or observation-satellites. And these applications do have an inherent need for data-protection. This work uses fault injection to investigate the SEU susceptibility of different types of AES implementations. But the results gained by fault injection are not only applicable for estimating a designs robustness against configuration SEUs, but they can also serve as an estimation on the designs behaviour when other effects like ageing affect the FPGAs resources or routing.

The paper is structured as follows: Section II gives an overview over different methods of implementing AES on a FPGA and different methods of measuring and estimating the susceptibility of designs against SEU. Section III serves as an overview on the used fault injection flow for gaining emulation results for different AES implementations. The AES implementations evaluated in this work are introduced in section IV together with the used test setups in section V. Finally section VI presents the SEU injection results followed by a conclusion.

II. AES implementations and fault injection

A. Different AES implementations

When implementing AES the first decision which needs to be taken is whether to implement the algorithm in software together with a CPU or whether to implement it in hardware as a dedicated IP-core. This decision normally is driven by throughput- and latency needs, by possible reuse of an existing CPU and by area- and resource constraints.

Using dedicated IP-cores for implementing AES it is possible to achieve the highest data throughput and the lowest latency values. But these are not the only possible points of optimization. There are various other goals for optimization like for example speed/area trade-off [4], security [5] and optimization on current consumption [6]. In AES hardware implementations the most costly component is the sbox [7] responsible for the Subbytes step of each round and parts of the key-expansion. If an implementation uses more sboxes it is usually faster in terms of cycles per encryption but also bigger. This together with the fact, that the sbox has the main responsibility for the security of AES makes this element of

AES IP-cores a wide field of research. High throughput AES implementations use pipelining to further improve the number of encryptions possible in a certain time interval.

When implementing AES on a CPU a great number of AES implementations is available in software. There are different optimizations for different types of CPUs. A well known implementation for IA32 type processors and basically all processors having a data width of 32 bit is shown in [8]. The implementation [9] represents just one optimized software for a microcontroller having less then 32 bit as well as [10] an assembler software for the 8 bit PicoBlaze tiny microprocessor for Xilinx FPGAs. Software implementations normally have a lower throughput compared to hardware implementations. But they have practically no additional hardware penalty if there is already a CPU in the system, which can be used for the AES calculations. There is only additional memory required for program code storage. But even if a tiny microcontroller like the PicoBlaze [11] is added to the design exclusively for AES calculations, this still results in a typically very low area consumption compared to dedicated AES hardware implementations.

Some researches finally propose Application Specific Instruction Processors (ASIP) [12] which try to combine the benefits of both solutions by including specific instructions to existing CPU architectures specially for accelerating AES. Like this it is possible to get a solution of a size comparable to a software implementation on a tiny microcontroller but with significantly higher throughput rates.

B. SEU robustness estimations

There are different ways of analysing the robustness of a SRAM FPGA based design against SEUs. Since exposing the device under test to actual radiation is an expensive and poorly controllable way of injecting SEUs there are works using other ways of simulating the effects of SEUs in the configuration memory. One way of injecting the SEUs in run-time is shown in [13]. In this method the so called SEU controller is used in conjunction with the ICAP port, which is described in [14], of Virtex-5 FPGAs. One or more bits of the configuration memory are flipped via the ICAP port during the runtime, which is equivalent to a SEU. The benefit of this methodology is that SEUs occur some time after device startup, just like in SEUs provoked by radiation. A drawback of this method is, that the SEU controller requires additional resources on the FPGA, which are vulnerable to SEUs and by this might distort the robustness measurement itself.

Another way of injecting SEUs into the configuration memory of a FPGA is to manipulate the bitstream prior to the programming and then writing a modified bitstream to the device. This manipulation can be done on a host PC as in [15], [16] or using additional hardware as in [17]. Bitstream manipulation on the PC is easy to implement, does not require additional resources on the FPGA under test, which avoids adding critical bits not belonging to the AES implementation, and allows evaluating the effects of SEUs in the configuration of the device under test.

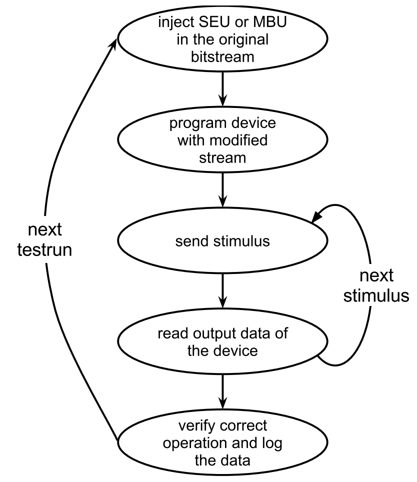
inject SEU or MBU in the original bitstream
program device with modified stream
send stimulus
read output data of the device
verify correct operation and log the data
next testrun
next stimulus

Fig. 1. SEU injection and test-flow

For evaluating the correct operation of the device, data considered significant is put out and compared to a reliable source. For AES this significant data obviously is the ciphertext. The source for comparison could be a golden run like in [16], a golden device as used in [18] or by the results of AES software.

III. SEU TEST FLOW

This work uses the SEU test flow as described in [15]. A flow chart of this test flow is shown in Fig. 1. Its basic idea is to program the device with a bitstream, where a SEU or MBU was injected beforehand and to check whether the design is still behaving as expected after programming it to the FPGA. The checking of the devices correct operation despite of the presence of a SEU or MBU is done in iterations each of them called testrun or run.

This test flow will be described here in more detail for completeness sake.

A. SEU injection and device programming

Since the bitstream generated by the Xilinx *bitgen* generator is in a well defined format, it is easily possible to invert one or more of its bits that are going to be written to the configuration SRAM.

The Xilinx bitstream format is packet oriented. There are two types of packets[14]. Each packet is partitioned into header and payload. In packets of type 1 the header defines how many words (32 bit) are going to be read or written to what register. Type 2 headers can be used following type 1 headers and allow specifying a bigger word count, while neither changing the read/write setting nor the destination register. To inject SEUs into the configuration memory only bits have to be changed, that are in the payload of a package that is written to the FDRI packet register of the Virtex-5 FPGA. It needs to be ensured, that the CRC checking is disabled when generating the bitstream using bitgen.exe.

In order to get a completely automatic test environment the programming of the FPGA needs to be automated as well. For Xilinx FPGAs this is achieved by using the command line interface of the *IMPACT* [19] programming tool.

B. Behaviour validation

The SEU injection test-flow used by this work utilizes a UART channel for both, sending a stimulus to the design and for receiving the designs calculation results on the host PC. The correct data can be determined by running the FPGA with the desired system without any SEUs in a golden run, or by using a software AES implementation.

Each so-called test-run consists of the following:

- Injecting the SEU or MBU and programming the device as described in the previous section
- Sending a stimulus consisting of 16 bytes to the device (which will automatically trigger the design to start the calculations)
- Receiving the designs response and comparing it to the expected value of the golden run
- Sending next stimulus (if multiple stimuli are used)

If the UART output of a run with a SEU or MBU effected bitstream is the same like in the golden run, the SEU or MBU positions are marked as non-critical. If the output differs, these locations are marked as critical.

Using UART as communication channel between the FPGA and the PC controlling the test flow causes a very small overhead in terms of logic. UART can be implemented easily using only little logic resources and by this it is secured, that a high percentage of the errors caused by SEUs result from SEUs in the application under test (the AES) and not errors in the UART.

A custom set of PHP scripts and C++ programs has been developed for the bitstream manipulation, programming of the device, UART validation and testflow management. Like this an automated testing of *n* SEU or MBU positions was possible without the need of manual interaction.

IV. Compared AES implementations

Different implementations with different properties were selected for robustness analysis. These implementations vary in their execution speed, resource requirements for implementing them on a FPGA and mostly in their architectures. Three AES specialized IP-cores were tested and one small microcontroller running a software for AES calculations. The following list contains all the implementations in order of their execution speed and resource requirements:

- H.V.Kampen code for PicoBlaze
- Implementation of J.Castillo
- Four sbox AES implemenation
- Implementation of H.Satyanarayana

The microcontroller based implementation together with the AES specific IP-cores from J.Castillo and H.Satyanarayana are publicly available. The so-called *four sbox* implementation was developed within this work to fill the gap in between the properties of the J.Castillo and H.Satyanarayana implementation.

A. H.V.Kampen implementation using the PicoBlaze

Obviously not only specialized IP-cores can be used to perform the AES algorithm, but it is also possible to use a microprocessor together with software. In this case the low area-consuming PicoBlaze microcontroller for Xilinx FPGAs is used together with an AES-software developed by H.V.Kampen [10].

The key-, input- and output-data are provided using the standard 8 bit input and output ports of the PicoBlaze microcontroller. This solution uses about 1.5 Mcycles for one encryption, which is significantly higher compared to the special IPs. But the area consumption of the PicoBlaze microcontroller is the lowest of all solutions evaluated in this work. The sbox is implemented in software by executing composite field arithmetic, which makes up for more than 99% of the PicoBlazes cycles for an encryption. A significant speed-up to about 10000 cycles could be achieved by implementing a sbox look-up-table, which requires additional RAM to be connected to the PicoBlaze. But within this work the original variant with the composite field sbox is used.

B. J.Castillo implementation

J.Castillos implementation of the AES algorithm [6] has the focus on a low area consumption. SystemC was used for the HDL design and a SystemC to Verilog translator delivers a synthesizable design.

The interface of this module consists of two 128 bit inputs for key- and state-data, as well as of a 128 bit output for the encryption result. In addition to that, there are separate load-signals for both key and state together with a ready-output for indicating the end of the encryption. Only one sbox is implemented due to the area optimized approach, and this sbox is not implemented as a look-up-table but using composite field arithmetics which leads to further area savings.

One encryption takes about 500 cycles, which is sufficient for many applications.

C. Four sbox implementation

In order to get a compromise between area consumption and encryption speed an AES IP was developed within this work, which uses four sboxes. These are implemented as look-up-tables and are used for both, the key expansion and the sbox-step of the AES algorithm. The internal state machine uses 8 cycles for each of the 10 AES rounds, which leads to a execution time for one of 82 cycles for an AES encryption including one cycle for data input. Similar to the solution of J.Castillo, the interface of the *four sbox* implementation consists of two 128 bit inputs for providing key- and state-data, one 128 bit output for the encryption result, a single bit input for starting the encryption and a one bit output for signaling of the encryption end.

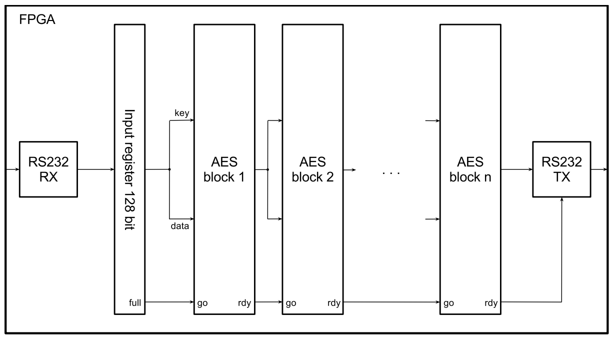

Fig. 2. SEU injection set up for the implementations J.Castillo, *four sbox* AES and H.Satyanarayana

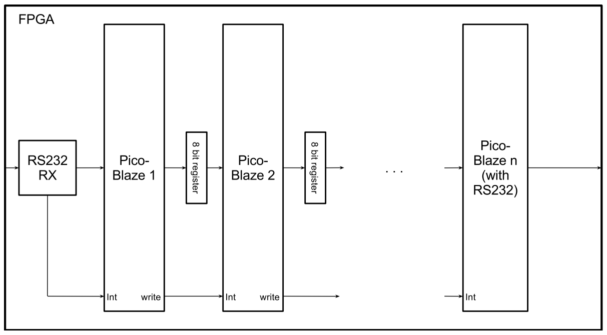

Fig. 3. SEU injection set up for AES on PicoBlaze

D. H.Satyanarayana implementation

The fastest AES implementation tested in this work is the implementation of H.Satyanarayana [20]. It uses 20 sboxes to be able to perform one AES-round in one clock cycle. In contrast to the other AES modules the implementation of H.Satyanarayana uses a 64 bit interface and two clock cycles to input key- and state-data. The data output is 128 bit wide and there is also a start-encryption input together with a encryption-ready output. One AES128 encryption needs 16 cycles using this implementation.

V. Test setup for AES testing

For error injection into the bitstream of a SRAM based FPGA it is recommendable to have a high device resource utilization. Like this there are less unused devices resources of the FPGA, where faults injected in corresponding configuration bits will most likely have no impact on the overall functionality. Since the used Virtex-5 device is significantly bigger than the AES implementations evaluated it is necessary to place multiple instances of the same implementation to fill the device.

A wrapper for the H.Satyanarayana implementation was developed to achieve a similar interface as the implementations of J.Castillo and the *four sbox* AES. This interface consists of a 128 bit key- and plaintext-input, a 128 bit ciphertext output, an encryption start and encryption-ready signal. Having this interface allows to build a test setup which fills the FPGA by chaining multiple instances of the same AES module. This chain of AES modules is shown in Fig. 2 and consists basically of three sections:

- The *RS232 RX* part together containing a 128 bit flipflop to hold the input data for the first AES module,
- Various chained *AES-cores*, which make out the majority of the device utilization and
- A *RS232 TX* part, which is used to send the result of the AES chain to the connected PC.

Within the SEU test flow the correct device functionality is validated by evaluating the AES encryption results of the chain for eight different AES input vectors, which means eight different stimuli. Each AES input vector is written by the PC via RS232 in 16 eight bit frames to the input register. As soon as this is filled the first AES module starts its encryption using the data of the input register as key as well as plaintext. When the first module finishes calculating the ciphertext, this output is used as key- and plaintext-input of the second module and so on. Finally when the last module finishes its calculations the 16x8 bit of ciphertext are send to the PC for checking against the expected result. The chains ciphertext to all eight stimuli is compared to the expected data and if a fault occurred the corresponding SEU or MBU is logged as critical.

The test setup for the PicoBlaze microcontroller differs in some aspects. It is shown in Fig. 3. This setup consists basically of the same three sections, only some characteristics differ due to the 8 bit datawidth of the PicoBlaze.

- The *RS232 RX* part has only a 8 bit flipflop to hold the input data, since the sequence of UART bytes form the RS232 can be stored on the fly by the first PicoBlaze,
- The different *PicoBlazes* in the chain need a 8 bit flipflop to transfer the calculated ciphertext in portions of 8 bit. This is synchronized using interrupts and the data transfer is done using the input- and output ports.
- No special *RS232 TX* part is needed, since bitbanging can be used for RS232 transmit causing very little hardware overhead.

A test-run is equivalent to the setup mentioned above. But for the PicoBlaze microcontroller only the result of one AES input vector (one stimulus) is used to verify the correct operation of the device. This is due to the fact, that for a single AES encryption more than 1500000 clock-cycles are needed by the PicoBlaze. Like this it gets already tested in a high measure using a single stimulus.

VI. Test results

A detailed summary of the different AES architectures and corresponding test setups implemented on a Virtex-5 XC5VFX70T is given in table I.

In addition to AES implementation specific details like number of sboxes and cycles needed for encryption also implement also information dependent on the placing of the AES architectures on the actual FPGA can be found. This information contains the achieved size of the AES chain,

TABLE I
AES ARCHITECTURES IMPLEMENTATION RESULTS ON XILINX VIRTEX-5 FPGA

	H.V.Kampen	J.Castillo	*Four sbox* AES	H.Satyanarayana
CPU/IP-core	CPU	IP-core	IP-core	IP-core
Sboxes implemented	1	1	4	20
Cycles per single encryption	$>$ 1.500.000	508	82	16
Chain size n on Virtex-5 [nr. of AES modules]	211	38	35	22
Virtex-5 Slices	10.464 (93%)	11.067 (98%)	11.030 (98%)	10.378 (92%)
Virtex-5 LUTs	33.137 (73%)	42.161 (94%)	41.825 (93%)	36.330 (81%)
Virtex-5 Registers	17.754 (39%)	25.862 (57%)	9.959 (22%)	14.799 (33%)
Virtex-5 BRAM	129 (87%)	0 (0%)	0 (0%)	0 (0%)
Max. running speed on Virtex-5 [MHz]	189	149	154	271
Data throughput per AES module [Mbit/s]	0.016	37.5	241.0	2166.8
Accumulated AES data throughput [Mbit/s]	3.4	1427	8434	47671

the maximum possible clock speed, the data throughput of an individual module for a certain architecture and the accumulated throughput. Within this data no optimization was done for getting the best trade-off for area and speed. Even though it was aimed for a high device utilization the size of the AES chain does not represent the absolute maximum number of possible modules, which could be placed on the FPGA. This number only serves as an estimation on the size of the AES module by comparing it to the chain size of other implementations.

Table I also shows the significant AES calculation speed- (and by this throughput-) differences. This throughput varies from 16 kbit per second for the PicoBlaze running the software of H.V.Kampen to 2.17 Gbit per second of the H.Satyanarayana implementation. The accumulated AES data throughput parameter serves as an estimation on the implementation size/area trade-off. It can be seen that even operating 211 PicoBlaze AES modules in parallel has a lower performance than one implementation of J.Castillo.

For the four AES implemenations tested in this work, the fastest implementation from H.Satyanarayana also shows the best size/area trade-off. This is due to the fact, that the other implementations have a more complex state-machine as overhead for braking down an AES round into smaller subparts. Also the logic is used less continuously for the other solutions. For example the mixcolums logic is used only in four of the 8 cycles for a round in the *four sbox* solution. In contrast to that for the H.Satyanarayana implementation all logic is used in all cycles of the rounds, with exceptions in the first and last round.

The results of the robustness testing are summarized in Fig. 4. This bar chart summarizes the percentage of bits that where marked critical executing the SEU injection flow.

The SEU injection flow was executed with 16000 testruns for each of the four AES implementations injecting a single upset. For two bit MBUs and three bit MBUs 8000 testruns for each implementation were executed. This gives a total of 128000 testruns executed.

The lowest percentage of critical bits and by this the best robustness was achieved by the implementation of the PicoBlaze

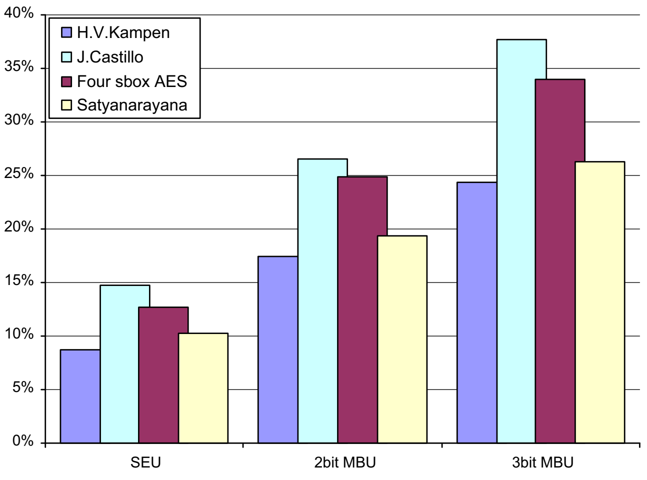

Fig. 4. Percentage of critical bits in fault injection

microcontroller together with the software of H.V.Kampen. An overall of about 9% of the 4000 single bit flips (SEUs) did lead to a wrong output of the cipher chain of this module. The reason for that is that the PicoBlaze is a general purpose microcontroller capable of executing any software. Running only an AES implementation will lead to the situation, where parts of the PicoBlazes logic will never be used by the program and by this SEUs in corresponding configuration memory will not have effects.

For the AES specific IP-cores it can be seen, that the fastest implementation (H.Satyanarayana) shows the best and the slowest implementation shows the worst robustness. The *four sbox* AES showed a number of critical bits in between the two other solutions. These observations can be explained by two factors:

- The slower AES solutions have a significant overhead caused by the state-machine and the necessary multiplexers for using only one or four sboxes for the 20 sbox lookups of each AES-round. These resources are not needed for the H.Satyanarayana implementation and by this no faults can be caused by this logic.

- An additional possible explanation are non-tested sbox entries. A 10 round AES128 encryption needs 200 $(10 rounds * 20 \frac{lookups}{round})$ sbox lookups. Using eight different stimuli the overall number of sbox lookups is 1600 per testrun and module. This leads to the fact, that for the implementation of J.Castillo the one available sbox gets tested 1600 times. In contrast to this for the H.Satyanarayana implementation with its 20 sboxes each sbox gets tested only 80 times. With a sbox consisting of $2^8 = 256$ possible inputs and the possibility of duplicates in the 80 sbox lookups some entries are not checked in each test-run.

From the SEU injection results it can be concluded, that in terms of robustness and resource requirements having a microcontroller like the PicoBlaze running AES software is a good solution. The only restriction to that is that the overall system must be able to handle the slow performance of this solution. AES specialized IP-cores offer faster encryptions and higher throughput, but also a slightly lower robustness against SEUs. In this work the robustnesses were comparable for these IP-cores, where the ones with less statemachine and multiplexing overhead showed slight advantages over area-optimized approaches.

VII. Conclusion

Within this work three freely available AES implementations of different performances where presented together with a fourth solution developed to fill a gap in the performance and area spectrum of the other three. Two test setups for microcontroller based AES calculations and IP-core based AES encryptions for SRAM based FPGAs were developed to be able to use a fault injection flow for SEU analysis. The results of this analysis showed, that implementing AES in a microcontroller results in a slightly higher robustness against configuration SEUs, but also in a significantly lower encryption performance.

Acknowledgments

This work has been partially supported by Ministerio de Ciencia e Innovación of Spain within the project TEC2010-21196-C02-01 and by the Department of Education, Universities and Research of the Basque Government within the fund for research groups of the Basque university system IT394-10.

References

[1] http://home.earthlink.net/˜rcv007/deschall.htm, "DESCHALL project homepage - Worlds First DES Crack," http://home.earthlink.net/˜rcv007/deschall.htm, 1997.

[2] S. Kumar, C. Paar, J. Pelzl, G. Pfeiffer, and M. Schimmler, "Copacobana a cost-optimized special-purpose hardware for code-breaking," in *Field-Programmable Custom Computing Machines, 2006. FCCM '06. 14th Annual IEEE Symposium on*, april 2006, pp. 311 –312.

[3] FIPS-197, NIST - Nacional Institute of Standards and Technology, "Announcing the ADVANCED ENCRYPTION STANDARD (AES)," http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, 2001.

[4] Rizk, M.R.M. and Morsy, M., "Optimized Area and Optimized Speed Hardware Implementations of AES on FPGA," in *Design and Test Workshop, 2007. IDT 2007. 2nd International*, dec 2007, pp. 207 –217.

[5] Shang, D. and Burns, F. and Bystrov, A. and Koelmans, A. and Sokolov, D. and Yakovlev, A., "High-security asynchronous circuit implementation of AES," *Computers and Digital Techniques, IEE Proceedings -*, vol. 153, no. 2, pp. 71 – 77, march 2006.

[6] J. Castillo, "A128/192 AES - Project: systemcaes," http://opencores.org, 2004.

[7] Namin Yu and Heys, H.M., "Investigation of compact hardware implementation of the advanced encryption standard," in *Electrical and Computer Engineering, 2005. Canadian Conference on*, may 2005, pp. 1069 –1072.

[8] Brian Gladman, "AES and Combined Encryption/Authentication Modes," http://gladman.plushost.co.uk/oldsite/AES/, 2011.

[9] Uli Kretzschmar, "AES128 A C Implementation for Encryption and Decryption; SLAA397A," http://www.ti.com, 2009.

[10] H.V.Kampen, "PicoBlaze Rijndael (AES-128) block cipher," http://www.mediatronix.com, 2003.

[11] K. Chapman, "PicoBlaze 8-Bit Microcontroller for Virtex-E and Spartan II/IIE Devices," Xilinx Application Notes XAPP213, http://www.xilinx.com, Feb. 2003.

[12] T. Good and M. Benaissa, "Very small fpga application-specific instruction processor for aes," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 53, no. 7, pp. 1477 – 1486, july 2006.

[13] Ken Chapman, "Virtex-5 SEU Critical Bit Information Extending the capability of the Virtex-5 SEU Controller," Xilinx Documentation SEU lounge, http://www.xilinx.com, Feb. 2010.

[14] Xilinx Corp., "Xilinx-5 FPGA Configuration User Guide," Xilinx Documentation UG191, http://www.xilinx.com, Aug. 2010.

[15] U.Kretzschmar, A.Astarloa, J.Lázaro, J.Jimeńez, and A.Zuloaga, "Automatic experimental set-up for robustness analysis of circuits based on tiny soft microprocessors implemented on sram fpgas," in *accepted at The International Symposium on System-on-Chip (SoC), 2011 13th IEEE Annual International Symposium on*, Nov 2011.

[16] Zachary K. Baker and Joshua S. Monson, "Fault Injection Into SRAM-Based FPGA For The Analysis Of SEU Effects," in *PROCEEDINGS 2003 IEEE International Conference on FieId-Programmable Technology (FPT)*, 2003.

[17] M.Alderighi and F.Casini and S.DÁngelo and M.Mancini and A.Marmo and S.Pastore, "A Tool for Injecting SEU-like Faults into the Configuration Control Mechanism of Xilinx Virtex FPGAs," 2003.

[18] Bernardi, P. and Reorda, M.S. and Sterpone, L. and Violante, M., "On the evaluation of SEU sensitiveness in SRAM-based FPGA," 2004.

[19] Xilinx Corp., "SEU Strategies for Virtex-5 Devices," Xilinx Documentation, http://www.xilinx.com, 2002.

[20] Hemanth Satyanarayana, "AES128 - Project: aes_crypto_core," http://opencores.org, 2004.