Efficient Fault Tolerant SHA-2 Hash Functions for Space Applications

Marcio Juliato Catherine Gebotys Reouven Elbaz
Department of Electrical and Computer Engineering
University of Waterloo
200 University Avenue West
Waterloo, ON, Canada, N2L 3G1
+1 (519) 888-4567
{mrjuliat,cgebotys,reouven}@uwaterloo.ca

Abstract—Satellites are extensively used by public and private sectors to support a variety of services. Considering the cost and the strategic importance of these spacecrafts, it is fundamental to utilize strong cryptographic primitives to assure their security. However, it is of utmost importance to consider fault tolerance in their designs due to the harsh environment found in space, while keeping low area and power consumption. Therefore, this paper proposes novel fault tolerant schemes for the SHA-2 family of hash functions and analyzes their resistance to SEUs. Results obtained through FPGA implementation show that our best fault tolerant scheme for SHA-512 uses up to 32% less area and consumes up to 43% less power than the commonly used TMR technique. Moreover, its memory and registers are 435 and 175 times more resistant to SEUs than TMR. These results are crucial for supporting low area and low power fault tolerant cryptographic primitives in satellites.

TABLE OF CONTENTS

1. INTRODUCTION

Satellites are extensively used by public and private sectors to support communication services, conduct scientific experiments, provide navigation and meteorological support, or increase homeland security. Some countries also employ commercial satellites for military communications [1]. The Consultative Committee for Space Data Systems (CCSDS) has highlighted in [2] that advances in technology allow for complex attacks to be easily carried out against satellites, making security an important goal in satellite designs. Considering the cost and the strategic importance of these spacecrafts, it is not recommended to assure their security by relying on the uniqueness and obscurity of their designs. Actually, due to the lack of appropriate security, some satellites have already been compromised [3], [4], [5]. Threats to satellites can pose severe risks to communications infrastructures [1], [6] and architectures must be designed to provide security services such as authentication, data integrity and confidentiality, thus increasing the security of those spacecrafts.

Security architectures [7], [8] have been recently proposed to authenticate communications between ground stations and spacecrafts or to provide security services to data processed on-board. To do so, cryptographic primitives are utilized, such as hash functions for integrity checking/authentication and block encryption for confidentiality. These architectures encounter implementation challenges to protect the security mechanisms from the harsh environment found in space. Radiation coming from space can hit satellites' circuitry and cause errors known as Single Event Upsets (SEUs) [9]. SEUs are forms of soft-errors, in the sense that they cause dynamic bit flips but are not damaging to the hardware.

Spacecrafts include both ASICs and a variety of FPGAs in their subsystems. ASICs and non-volatile FPGAs (Anti-Fuse or Flash) offer an increased SEU resistance compared to FPGAs based on Static Random Access Memory (SRAM) technology; the construction of the SRAM cells make them more sensitive to SEUs. In some cases a single bit-flip in a configuration element of an SRAM FPGA is able to disrupt the entire functioning of the design implemented in the chip. However, the low production volume of satellites makes FPGAs an attractive alternative to reduce non-recurring engineering costs. In addition, their reconfigurability allows post-launch updates and patches to the satellite hardware. Regarding more specifically SRAM FPGAs, they provide both high density and high speed therefore resulting in a good trade-off between performance and flexibility. Thus, to overcome the issue of SEUs occurring in configuration elements, designers utilizing SRAM FPGAs implement methods like read-back, CRC checking and reconfiguration. These methods basically

Date: April 30, 2009.

consist in reading the configuration from the FPGA, checking its correctness using CRC, and then reconfiguring the device with a correct bitstream. Device hardening is another technique to make FPGAs more resistant against radiation.

Considering that the hardware description is protected against SEUs by either the underlying technology (i.e. ASICs and non-volatile FPGAs) or by the aforementioned techniques, the present work focuses on protecting the data processed by the device. Indeed, the issue of SEUs in registers and memory persists for all kinds of FPGAs as well as for ASICs. Therefore, whatever the underlying technology is, we assume that the configuration of the device is safe regarding errors. Thus, the proposed techniques target error detection and correction on the data processed within a device.

This work focuses on fault tolerant design of hash functions for space applications. Hash functions are employed in satellite systems in many different ways. They are used as building blocks in authentication schemes like digital signatures [10], [11], and Hash-based Message Authentication Codes (HMAC) [12]. These cryptographic primitives allow for the integrity checking of data received from a ground station to assure that they were not accidentally or maliciously compromised. Furthermore, it could also be employed in invasion detection and recovery schemes [7]. By using those schemes, it becomes possible, for example, to determine whether an attacker, who may have broken into the satellite, has tampered with the system's program memories or FPGA configuration. Considering the properties of hash functions and their applications it is clear that a single bit-flip can have disastrous consequences like provoking unjustified satellite reset or intrusion alert. Therefore, SEUs should be properly addressed to guarantee the correct and reliable operation of spacecrafts employing cryptographic algorithms. For that purpose, fault-tolerant designs for hash function are required.

In this paper we investigate fault tolerant architectures for the family of Secure Hash Algorithms (SHA) [13] which is the most commonly used hash function in integrity checking and authentication architectures. More specifically, the SHA-2 family of hash functions is recommended to be adopted by the Consultative Committee for Space Data Systems (CCSDS) as a standard for space systems [14]. Thus, we propose novel fault-tolerant solutions for SHA-2 that combine existing techniques, i.e. Triple Modular Redundancy (TMR) and Hamming Codes (HC), in order to offer several performance and cost trade-offs. Moreover, while the applicability of these solutions are independent of the underlying technology, we provide a detailed study on power consumption and area based on FPGA implementations. We also carry out an analysis to determine the robustness of each scheme against SEUs, and we show that the proposed solutions offer a better resistance to bit-flips when compared to the traditional TMR. For instance, the best scheme proposed for SHA-512 consumes 32% less area and consumes up to 43% less power than TMR. Furthermore, compared to TMR, its memory and registers are, respectively, 435 and 175 times more resistant to SEUs.

The remainder of this paper is organized as follows. Related works are presented in Section 2. Section 3 describes the SHA-2 algorithms, while Section 4 introduces non-fault tolerant designs for SHA-2. The fault tolerant architectures proposed in this work are presented in Section 5 and their robustness against SEUs are evaluated in Section 6. Section 7 reports our experimental results in term of power, area and frequency of operation, and provides a comprehensive comparison of the proposed schemes applied to the SHA-2 family of hash functions. Our conclusions are presented in Section 8.

2. Related Work

With the growing worldwide demand for satellite-based services, the dependence on these spacecrafts tends to increase. Consequently, in case of a satellite failure, the risk of losses tends to go higher over the years. As a result, the disruption of satellite services, whether intentional or not, can have a major economic impact. This context motivated the United States General Accounting Office to issue a report [1] presenting several threats to satellite systems. Its conclusion stresses that the security of commercial satellites should be more fully addressed in order to achieve higher levels of protection for the country's critical infrastructure. CCSDS has also highlighted [2] the importance of including security in space missions. Actually, several proposals have been made for CCSDS to standardize the use of strong cryptographic mechanisms for integrity checking, authentication and encryption [14], [15].

Several security architectures have been proposed for satellites to remotely manage their hardware configuration from the ground station [16], and for purposes of key distribution [17] and management [18], [19], [20]. All of those works make use of cryptographic primitives such as hash functions to achieve their goals, but none of them considers the harsh environment surrounding spacecrafts and more specifically the resistance to SEUs. In [7] only, a security scheme for key recovery in satellites is proposed with SEU-resistant hardware. The authors, however, employ existing techniques for fault tolerance like TMR and HC, but do not specifically investigate new approaches for fault tolerance.

Hardware implementations of hash functions have been proposed by several works. In [21] a single chip implementation of SHA-384 and SHA-512 based on FPGAs is introduced. A SHA-256 processor is presented in [22], also employing FPGAs for its implementation. In [23], [24] the whole SHA-2 family is implemented in FPGAs and compared in terms of area, frequency of operation and throughput. Some other works [25], [26], [27] provide further comparisons of FPGA-based implementations of hash functions. Also, some optimizations based on pipelining, loop unrolling, operation rescheduling and hardware reutilization are proposed in [28],

[29]. Previous research has extensively addressed the implementation of the SHA-2 family of hash functions. However, none of them consider the resistance against SEUs and are therefore not appropriate for space applications. Only in [30] error detection was considered for SHA-512 in FPGAs. Since this approach employs parity prediction for the internal hash function operations, it is therefore unable to correct errors, as it is proposed in this paper.

Fault-tolerant designs of cryptographic primitives have mainly been proposed for block encryption algorithms like the Advanced Encryption Standard (AES) [31]. In [8] a fault detection and correction capabilities are included into AES implemented in FPGAs. SEUs are detected in each round transformation by using parity prediction, and corrected through the use of Hamming codes applied to the round data matrix. Another approach is due by [32], in which single bit-flips in the substitution box of the AES algorithm are detected by using look-up tables and parity prediction.

The hardware designs of chips is made resistant to SEUs through different techniques like radiation hardening, which is the case of Actel's Anti-Fuse [33] and Flash [34] FPGAs. Anti-Fuse devices provide higher reliability compared to Flash ones, but their main drawback is that they can be configured only once. Xilinx also provides radiation-hardened SRAM-based FPGAs [35] to meet the requirements of space applications. SRAM-based FPGAs are available in higher densities than the Anti-Fuse and Flash counterparts, but they are more sensitive to SEUs because of the features of SRAM cell itself. Altera also proposes a strategy to protect the configuration of SRAM-based FPGAs against SEUs at runtime. Some Altera FPGAs employ built-in Cyclic Redundancy Check (CRC) circuitry [36]. Although the aforementioned strategy is able to check the internal FPGA's configuration, it does not detect errors in the user data stored or being processed within the device. Other designs can employ ASICs to achieve high reliability of their hardware implementations at the cost of very high non-recurring engineering costs. However, whatever the underlying technology is, the user data being processed within the device remain unprotected against SEUs.

Aerospace applications have traditionally used techniques based on modular redundancy for mitigating SEUs. For instance, TMR together with FPGA reconfiguration is proposed in [37] to fully protect systems from SEUs. TMR can be very costly, though. It requires the triplication of all architectural elements along with a voter, thus demanding considerable amounts of resources. Attempts were made in [38] to reduce the costs associated with fault mitigation by only applying TMR to the most critical components of a design.

In contrast to previous research, we propose novel fault-tolerant designs of the SHA-2 family of hash functions. Therefore, we investigate original combinations of TMR and Hamming codes (HC) that support not only error detection, but also correction. We explore different levels of granularity involving TMR and HC, and analyze all the implementations in terms of area, frequency of operation, throughput and power consumption. Additionally, this research provides an analysis of the resistance of each new scheme against SEUs and compare them to the traditional TMR.

3. SHA-2 ALGORITHMS

In this section, the SHA-2 family of hash algorithms [13] is presented. It comprises four algorithms, namely, SHA-224, SHA-256, SHA-384, and SHA-512. SHA-224 and SHA-256 have several commonalities, thus in the following description SHA224/256 is a shortcut that refers to both hash functions. Similarly, SHA384/512 refers to both SHA-384 and SHA-512. Moreover, the datapath width in bits of these functions is denoted by D; D is equal to 32 bits for SHA224/256 and to 64 bits for SHA384/512.

These algorithms are one-way hash functions able to process messages of up to 2^{64} bits for SHA224/256 and 2^{128} bits for SHA384/512. The output of the algorithms is a message digest, which varies in size depending on the algorithm used. For instance, SHA-224 produces a 224-bit message digest, SHA-256 produces a 256-bit message digest, and so on. These algorithms can be divided into two computational parts, denoted preprocessing and hash computation.

Preprocessing

SHA-2 algorithms take as input blocks of different sizes. The preprocessing stage first splits the original message in N blocks, namely $M^{(1)}, M^{(2)}, ..., M^{(N)}$. SHA224/256 process 512-bit blocks of data, whereas SHA384/512 process 1024-bit blocks. Then, padding is performed if the message length is not a multiple of the underlying block size. Next, eight initial hash values, $H_0^{(0)}, ..., H_7^{(0)}$, are set. Each algorithm uses a distinct set of initial hash values given in [13].

Hash Computation

The entire computation of the message digest is based on operations over D-bit words. The SHA-2 algorithms comprise sixty four message schedule words ($W_0, ..., W_{63}$), eight working variables (a, b, c, d, e, f, g, h), and eight hash values ($H_0^{(i)}, ..., H_7^{(i)}$). In addition, SHA224/256 and SHA384/512 use respectively sixty four ($K_0, ..., K_{63}$) and eighty D-bit constants ($K_0, ..., K_{79}$), as specified in [13]. Furthermore, six logical functions are also employed, and are shown below. The operations $ROTR^n(x)$ and $SHR^n(x)$ are rotation and shift of x by n bits to the right.

SHA-224 and SHA-256:

$$\begin{aligned}
Ch(x,y,z) &= (x \wedge y) \oplus (\bar{x} \wedge y), \\
Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z), \\
\textstyle\sum_0(x) &= ROTR^2(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x), \\
\textstyle\sum_1(x) &= ROTR^6(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x), \\
\sigma_0(x) &= ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x), \\
\sigma_1(x) &= ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x).
\end{aligned}$$

SHA-384 and SHA-512:

$$\begin{aligned}
Ch(x,y,z) &= (x \wedge y) \oplus (\bar{x} \wedge y), \\
Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z), \\
\textstyle\sum_0(x) &= ROTR^{28}(x) \oplus ROTR^{34}(x) \oplus ROTR^{39}(x), \\
\textstyle\sum_1(x) &= ROTR^{14}(x) \oplus ROTR^{18}(x) \oplus ROTR^{41}(x), \\
\sigma_0(x) &= ROTR^{1}(x) \oplus ROTR^{8}(x) \oplus SHR^{7}(x), \\
\sigma_1(x) &= ROTR^{19}(x) \oplus ROTR^{61}(x) \oplus SHR^{6}(x).
\end{aligned}$$

Further, for each message block i, $1 \leq i \leq N$, a four-step digest round is performed as follows:

Step 1: Initialize the eight working variables

$$\begin{aligned}
a &= H_0^{(i-1)}, \quad b = H_1^{(i-1)}, \quad c = H_2^{(i-1)}, \quad d = H_3^{(i-1)}, \\
e &= H_4^{(i-1)}, \quad f = H_5^{(i-1)}, \quad g = H_6^{(i-1)}, \quad h = H_7^{(i-1)}.
\end{aligned}$$

Step 2: Prepare the message schedule

$$W_t = \begin{cases} M_t^{(i)}, & 0 \leq t \leq 15 \\ \sigma_1(W_{t-2}) + W_{t-7} + \sigma_0(W_{t-15}) + W_{t-16}, & 16 \leq t \leq j-1. \end{cases}$$

The number of words processed by the message scheduler is given by j. Actually, j corresponds to the number of iterations performed by the algorithm. For SHA224/256 $j = 64$, whereas for SHA384/512, $j = 80$.

Step 3: For $t = 0$ to $j - 1$ do:

$$\begin{aligned}
&T_1 = h + \textstyle\sum_1(e) + Ch(e,f,g) + K_t + W_t, \\
&T_2 = \textstyle\sum_0(a) + Maj(a,b,c), \\
&h = g, \\
&g = f, \\
&f = e, \\
&e = d + T_1, \\
&d = c, \\
&c = b, \\
&b = a, \\
&a = T_1 + T_2,
\end{aligned}$$

Step 4: Compute the i^{th} intermediate hash value $H^{(i)}$:

$$\begin{aligned}
H_0^{(i)} &= a + H_0^{(i-1)}, \quad H_1^{(i)} = b + H_1^{(i-1)}, \\
H_2^{(i)} &= c + H_2^{(i-1)}, \quad H_3^{(i)} = d + H_3^{(i-1)}, \\
H_4^{(i)} &= e + H_4^{(i-1)}, \quad H_5^{(i)} = f + H_5^{(i-1)}, \\
H_6^{(i)} &= g + H_6^{(i-1)}, \quad H_7^{(i)} = h + H_7^{(i-1)}.
\end{aligned}$$

After processing all N blocks of message M, the final message digest is obtained by concatenating the hash values $(H_0^{(i)}, ..., H_7^{(i)})$. More precisely, the message digest for each algorithm is given by the concatenations shown below. The concatenation of words is represented by the symbol $||$.

SHA-224:

$$H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}||H_5^{(N)}||H_6^{(N)}.$$

SHA-256 and SHA-512:

$$H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}||H_5^{(N)}||H_6^{(N)}||H_7^{(N)}.$$

SHA-384:

$$H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}||H_5^{(N)}.$$

4. Hardware Design

In this section a non-fault tolerant hardware design for SHA-2 algorithms is presented; this design is used in following sections to support the description of the evaluated fault-tolerant techniques. It basically consists of shift-registers, logical operations, D-bit adders, and a memory to store the algorithm's initialization values and constants. Similarly to hash hardware implementations mentioned in Section 2, we do not perform message padding in hardware. Our main focus is on the hash computation datapath.

The architectural elements of the SHA-2 implementation, as shown in Figure 1, can be divided into four main blocks: Intermediate Hash Computation, Compressor, Message Scheduler, and Constants Memory. The constants memory utilizes the FPGA's RAM blocks. Since this design does not involve any kind of fault tolerance, it is referred to as NoFT.

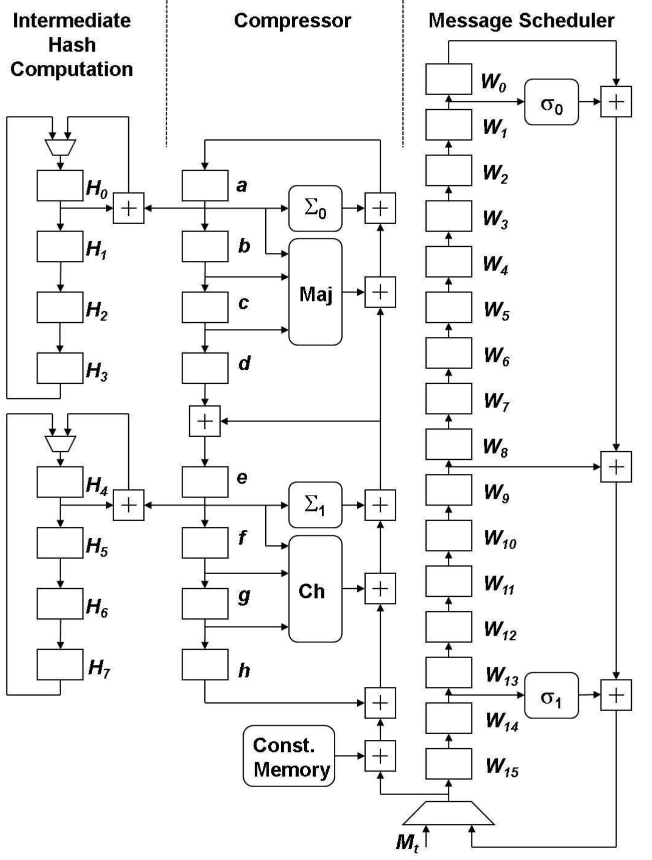

Figure 1. SHA-2 architectural elements.

The message scheduler's registers $W_0, ..., W_{15}$ are initialized by shifting in the first 16 words M_t of the message M; this processing takes 16 clock cycles. Simultaneously, the constants memory provides the initialization values for the working variables $(a, ..., h)$. Initial hashes $(H_0, ..., H_7)$ are also set within this period of time. After that, the compressor employs the current values of $a, ..., h$, as well as W_t and K_t to determine the new values of $a, ..., h$. As described in Step 3 of Section 3, this is performed in t iterations and is controlled internally by a counter. Again, SHA224/256 utilizes 64 iterations, while SHA384/512 uses 80 iterations. In each of these iterations, registers $W_0, ..., W_{15}$ and $a, ..., h$ are shifted in the direction of the arrows shown in Figure 1.

In the end of t iterations, the intermediate hash computation must be performed. This operation could be executed in one clock cycle, but it would require eight adders for that. However, in order to save implementation area, only two adders are utilized. This way, the computation of the intermediate hash is spread over the last 4 iterations by computing two additions per clock cycle. More precisely, the additions are performed when $t = 60, ..., 63$ for SHA224/256. For instance, in SHA224/256, when $t = 60$, H_3 and H_7 are computed, when $t = 61$, H_2 and H_6 are computed, and so on. The same strategy is followed by SHA384/512, but the additions are executed when $t = 76, ..., 79$.

In case of a multi-block message, a new execution cycle initiates with 16 more words M_t being shifted into the module. Then, the same procedure described above is executed. For the last message block, read operations are performed to shift out the message digest. Specifically, SHA-224, SHA-256, SHA-384 and SHA-512 require, respectively, 7, 8, 6, and 8 read operations.

The total memory requirement to store the constants K_t and $H_0^{(0)}, ..., H_7^{(0)}$ is 2304 bits for SHA224/256, and 5632 bits for SHA384/512, as shown in Table 1. Given the variables $W_0, ..., W_{15}$, $a, ..., h$ and $H_0, ..., H_7$, the total register requirements are 1024 bits for SHA224/256, and 2048 bits for SHA384/512, as listed in Table 2.

Table 1. Memory Requirements

Scheme	SHA224/256 (bits)	SHA384/512 (bits)
NoFT	2304	5632
FullTMR	6912	16896
TMR&HCMem	2736	6248
TMRRegs&HCMem	2736	6248
HCAllRegs	2736	6248
HCMainRegs	2736	6248

Considering the number of iterations involved in a hash computation, it is clear that a single bit-flip in a memory that provides constants or input blocks, or in registers propagating intermediate values, can be devastating for the applications using the hash function (e.g., data integrity checking, authentication). In order to make hash function designs appropriate for space applications, error detection and correction schemes must be incorporated so that SEUs do not compromise its normal operation. This issue is addressed in Section 5.

Table 2. Register Requirements

Scheme	SHA224/256 (bits)	SHA384/512 (bits)
NoFT	1024	2048
FullTMR	3072	6144
TMR&HCMem	3072	6144
TMRRegs&HCMem	3072	6144
HCAllRegs	1035	2060
HCMainRegs	1216	2272

5. Error Detection and Correction Schemes

In this paper, we propose four new fault-tolerant schemes that combine two existing techniques for error detection and correction: modular redundancy and information redundancy. Modular redundancy is usually employed as the traditional TMR: three identical modules are instantiated and their outputs are sent to a voter. The voter is then in charge of majority voting the three intermediate results coming from modules. After masking out a potential incorrect result, the voter outputs the final result. Notice that if two modules fail, the voter is unable to properly select the correct intermediate result, i.e. it tolerates up to one module failure. The second technique employed is based on HCs. The Hamming codes used in this work have Hamming distance of three, which means that they are capable of either detecting double bit-flips or detecting and correcting a single bit-flip. Since we are concerned with error correction, we are using HCs to detect and correct single bit-flips.

In total, five fault-tolerant schemes are considered. The first one is the regular TMR that we called *Full Triple Modular Redundancy*. The four schemes we introduce in this paper are named *TMR with Shared Encoded Memory*, *TMR for Registers and Shared Encoded Memory*, *Encoding/Decoding All Registers with Hamming Codes*, and *Encoding/Decoding Main Registers with Hamming Codes*. They are hybrid solutions based on the two techniques aforementioned. The goal is to decrease implementation area and power consumption, as well as to make designs more resistant against SEUs. Alternatives based on *checkpoint and restart* were not considered due to their potential negative impact in the module's speed and power consumption.

To show the improvements offered by these new solutions, we implemented these schemes for every hash function of the SHA-2 family (i.e. SHA-224, SHA-256, SHA-384, and SHA-512). In the following, when we refer to SHA-2, we imply the four functions belonging to the family; when distinction between functions is required we specifically name the hash function. We define a terminology for Hamming codes: (w,v), where v is the number of data bits, and w is the number of data bits along with parity bits. Furthermore, Tables 1 and 2 summarizes the memory and register requirements discussed below.

Full Triple Modular Redundancy

As described above, triple modular redundancy consists in triplicating the circuit and using a voter to determine the output. In this work, three SHA-2 hardware modules are instantiated and share the same inputs as depicted in Figure 2. This scheme is referred as `FullTMR` for short. During implementation of `FullTMR`, special attention was paid to the design partitioning in order to avoid the synthesizer merging common circuitry and registers, which would lead to misleading synthesis results. `FullTMR` is used only as a reference model for the comparisons performed in the next sections.

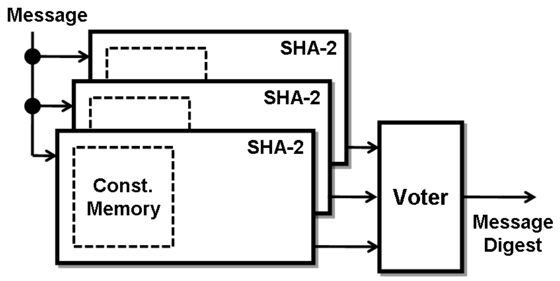

Figure 2. FullTMR block diagram

Since `FullTMR` uses three instances of the `NoFT` module, it triplicates the memory and registers requirements. Precisely, the memory requirements of `FullTMR` is 6912 bits for SHA224/256, and 16896 bits for SHA384/512. The register requirements is now 3072 bits for SHA224/256, and 6144 bits for SHA384/512. An advantage of `FullTMR` is that it includes fault-tolerance without a big impact on the module's speed, since it employs three `NoFT` modules working in parallel. On the other hand, a drawback is the big area penalty imposed by the use of replication. Thus, other schemes are proposed to reduce the memory requirements and to achieve smaller implementation area and lower power consumption.

Regarding SEU resistance, `FullTMR` has three times as much memory and registers as the `NoFT` module. Considering that these memories do not employ any fault-tolerant mechanism, a single bit-flip in one of these memories compromises the processing of the entire `NoFT` module. A second bit-flip in any other location of the other two memories, compromises the entire `FullTMR` module. The same failure condition applies to the registers; one bit-flip in a register of one of the modules, and a second bit-flip in any of the registers of the other two modules. Considering that spacecrafts may have mission lifetimes reaching decades, it is very likely to have multiple bit-flips in the memory elements. As a result, it is very important to protect memory against SEUs while reducing memory requirements; this is the idea behind the scheme presented in the following section.

TMR with Shared Encoded Memory

In order to scale down the number of memory bits used, as well as the power consumption, the constants memory could be shared among the three SHA-2 modules. This scheme is named `TMR&HCMem`, is depicted in Figure 3. However, the common memory needs to be protected against SEUs. This is accomplished by encoding the memory of SHA224/256 with a (38,32) Hamming code, whereas SHA384/512 employ a (71,64) Hamming code. For each memory read, a Hamming decoder detects and corrects any potential bit flip, thus sending the correct value to modules.

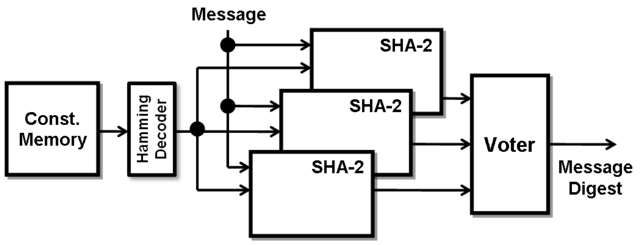

Figure 3. TMR&HCMem block diagram.

The use of a single memory in `TMR&HCMem` decreases the memory overhead compared to `FullTMR`, even when parity bits are attached to each memory element. As a result, the memory requirements of `TMR&HCMem` is 2736 bits for SHA224/256, and 6248 bits for SHA384/512. The consequence of using HC, though, is the inclusion of the Hamming decoder between the memory and modules. This decoder implies a longer critical path of the circuit and thus decreases its frequency of operation. The register requirements is exactly the same as in `FullTMR`, i.e. 3072 bits for SHA224/256 and 6144 bits for SHA384/512.

On the other hand, the main advantage of encoding the memory is that it now tolerates up to one bit-flip in each of its memory elements. In order to have a failure, two bit-flips must happen in the same memory element. Hence, the encoded memory is less likely to fail compared to the triplicated, unprotected memory in `FullTMR`. A deeper analysis of the probability of memory failure is provided in Section 6. Due to its better resistance to bit-flips, encoded memory is used in all of the following schemes introduced next.

TMR for Registers and Shared Encoded Memory

Given our concern in protecting only the data being processed, a further optimization to `TMR&HCMem` is to move the redundancy to the register level instead of keeping it at the modular level. This scheme is called `TMRReg&HCMem`. More precisely, instead of triplicating the entire SHA-2 module, only one SHA-2 module is used, but all its registers are triplicated, as illustrated in Figure 4. The register requirements is exactly the same as in `FullTMR` and `TMR&HCMem`, i.e. 3072 bits for SHA-224 and SHA-256, and 6144 bits for SHA-384 and SHA-512. But now, in order to mask out registers errors, one voter is used for each trio of registers. In total, 32 voters are needed, resulting in higher implementation area.

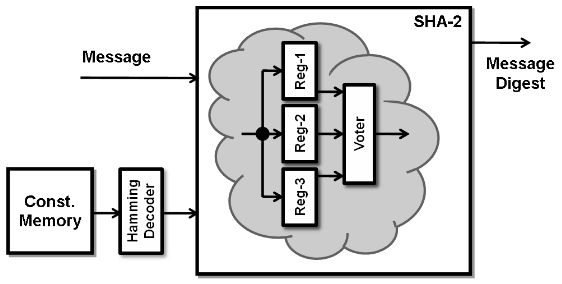

Figure 4. TMRReg&HCMem block diagram.

Similarly to `TMR&HCMem`, this approach uses an encoded memory to keep SHA-2 constants protected against SEUs. Given the Hamming decoder and the multiple voters used, lower frequencies of operation are expected compared to `FullTMR` and `TMRReg&HCMem`. The main advantage of this scheme, however, is that errors are masked in every clock cycle. In other words, the design fails if two bit-flips occur in the same register and in the same clock cycle. Since this is very unlikely to happen, a very high protection against SEUs in register is achieved with this scheme, as is more formally discussed in Section 6.

Encoding/Decoding All Registers with Hamming Codes

As an alternative to replicating registers and modules, a scheme named `HCAllRegs` is proposed. In this scheme, Hamming codes are used in place of TMR to protect the registers contents. The goal is to detect and correct potential bit-flips in each clock cycle of SHA-2 functions. In order to achieve that, register contents are encoded/decoded on every clock cycle before each write/read operation. As a consequence, registers are kept encoded all the time and therefore protected against bit-flips. This scheme is depicted in Figure 5, where the shaded and dark areas represent Hamming encoders and decoders.

In order to reduce the number of parity bits used, all the 32 registers were merged and treated as a single register. For example, if SHA224/256 encoded their thirty two 32-bit registers separately using a (38,32) Hamming code, a total of 192 parity bits would have been necessary. By treating the registers as one 1024-bit register, a (1035,1024) Hamming code can be used and only 11 parity bits are needed. Likewise, if SHA384/512 encoded their thirty two 64-bit registers separately using a (71,64) Hamming code, a total of 224 parity bits would be needed. When the registers are merged into a single 2048-bit register, a (2060,2048) Hamming code can be employed, thus demanding only 12 parity bits. Notice that, although Figure 5 show each register associated with an encoder and a decoder, a single encoder and a single decoder is employed. In sum, the register requirements are 1035 bits for SHA224/256, and 2060 bits for SHA384/512.

The main disadvantage of this approach is that the use of Hamming encoders and decoders for all registers increases the critical path of the SHA-2 modules and thus reduces the

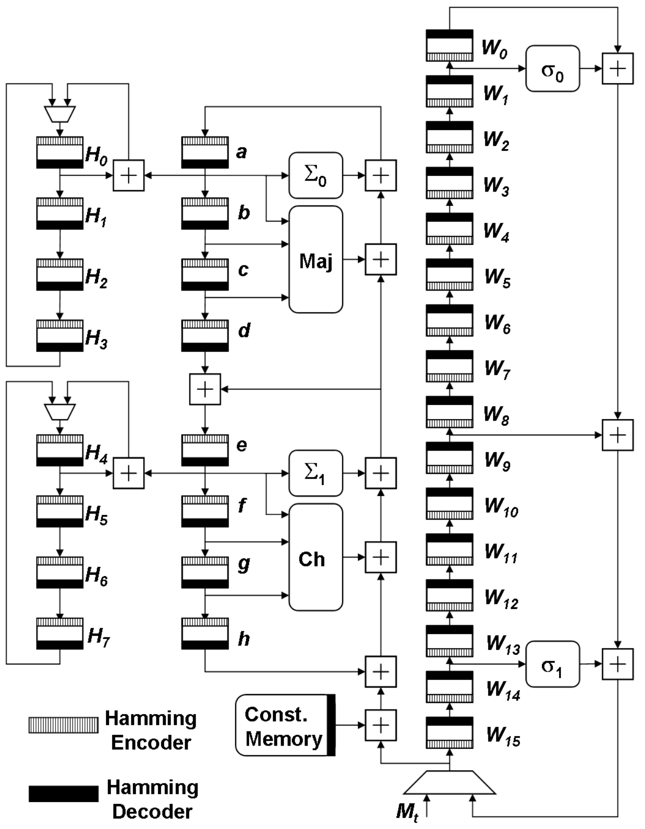

Figure 5. HCAllRegs block diagram.

frequency of operation of the design. However, since the merged register is decoded and re-encoded in every clock cycle, it will only have a failure if two bit-flips happen in the same clock cycle. As a consequence, this scheme provides a high protection against SEUs, as described in Section 6.

Encoding/Decoding Main Registers with Hamming Codes

Since `HCAllRegs` keeps all the registers always encoded, they are protected against SEUs all the time. Although `HCAllRegs` offers a high resistance against SEUs, that comes at the cost of employing large Hamming encoders and decoders to detect and correct errors in every clock cycle. This can be translated to a higher demand on implementation area and power consumption. In this context, it would be interesting to achieve a better trade-off between SEU protection, area and power consumption. This trade-off is explored in the scheme described in this section.

By analyzing Figure 5, it can be noticed that, in a given algorithm iteration, only registers $a, ..., h$, H_0, H_4, W_0, W_1, W_9, and W_{14} are involved in the SHA operations. Thus, those are the only registers that need to be decoded for fault detection and correction, while all the rest can remain encoded. Analogously, only a, e, H_0, H_4, and W_{15} are updated with new values. So, only these registers must be re-encoded. This approach, which encodes and decodes only the main registers is named `HCMainRegs`, and depicted in Figure 6.

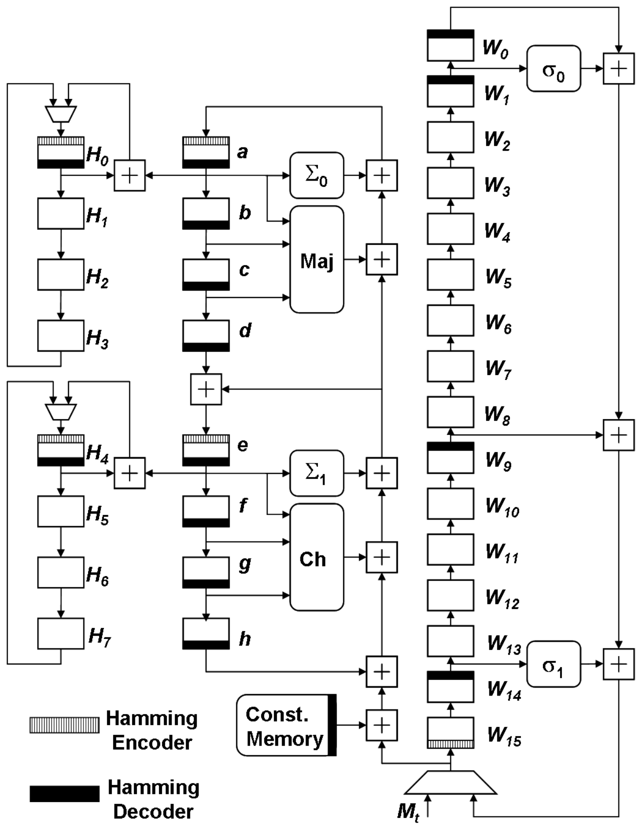

Figure 6. HCMainRegs block diagram.

In this scheme, registers are not merged, but have their own Hamming encoder/decoder (when they are needed). Differently from `HCAllRegs`, `HCMainRegs` uses separate encoders and decoders. More precisely, SHA224/256 encode their thirty two 32-bit registers separately using a (38,32) Hamming code, demanding 192 parity bits. Similarly, SHA384/512 encode their thirty two 64-bit registers separately using a (71,64) Hamming code, and uses 224 parity bits. The register requirements are 1216 bits for SHA224/256, and 2272 bits for SHA384/512.

From Figure 6 it is easy to observe error detection and correction is only performed when reading registers $a, ..., h$, H_0, H_4, W_0, W_1, W_9, and W_{14}. The data present in these registers have been waiting for a number of clock cycles until their use in the SHA-2 operation. The number of clock cycles may vary from 1 (for $a, ..., h$) to 60 (for H_0 and H_4) in the case of SHA224/256 or 70 in the case of SHA384/512. A failure will occur if two bit-flips happen in the same data word ($a, ..., h$, H_0, H_4, W_0, W_1, W_9, W_{14}) while they are in their idle period. This case is detailed in Section 6.

Similarly to `HCAllRegs`, `HCMainRegs` uses an encoded memory to keep the constants protected against SEUs. Besides, it uses encoders and decoders in its datapath, which will certainly increase the circuit's critical path. As a result, lower frequencies of operation are expected for `HCMainRegs` compared to the ones achieved in `FullTMR`.

6. Evaluation of Robustness against SEUs

In addition to factors such as implementation area and power consumption, it is important to evaluate the robustness of the proposed schemes against SEUs. A qualitative analysis, as done in Section 5, give us a good idea on how resilient each scheme is. However, it is appropriate to conduct a quantitative evaluation of the strategies proposed in this work, so that we can better compare them with the traditional schemes such as TMR.

The evaluations conducted in this section analyze the probability of failure of two main elements: memory and registers. We assume that there is one scheme per device (ASIC or FPGA), and that its implementation is spread uniformly over the device resources. Therefore, we define the following terms:

M : Total memory resources in bits,
R : Total number of registers in bits,
m : Used memory resources in bits, and
r : Used registers in bits.

In order to simplify the discussions, we assume that all designs are performing at the same frequency of operation. We consider the computation of 1 hash for our evaluations. We further assume that the hardware devices are in the same environment and subject to the same bit-flip rate. Then, we define:

μ : Bit-flip rate per memory bit per clock cycle,
ρ : Bit-flip rate per register bit per clock cycle, and
n : Period of time in which bit-flips may occur expressed in number of clock cycles.

Given that all schemes based on TMR and Hamming codes tolerate one bit-flip, we analyze the condition for failure, which is to have two bit-flips corrupting the SHA-2 computation. Then, we define the following terms:

$P(F_M)$: Probability of a memory failure,
$P(F_R)$: Probability of a register failure,
$P(X_1)$: Probability of the first bit-flip in X, and
$P(X_2)$: Probability of the second bit-flip in X,
where X can be either memory or register elements.

From these definitions it is possible to determine the probability of memory and register failures for each of the schemes presented in Section 5.

Full Triple Modular Redundancy

The traditional TMR triplicates the memory requirements of a non fault tolerant module. In order to have a failure in `FullTMR`, it is necessary to have one bit-flip in one of the memories, and a second bit-flip in any location of the other two memories. Let the total memory usage of TMR be m bits out of M bits available in the device. First analyzing the case of a first bit-flip, we have that the probability of a particle hitting one memory element on-chip is $1/M$. Given that m bits are used for TMR, the probability of having this particle hitting one memory element of the TMR design is m/M. Further, the three modules take n clock cycles to compute a SHA-2 operation. Moreover, while in operation their memory elements suffer a bit-flip rate of μ per clock cycle. Thus, the probability for the first bit-flip is given by $P(M_1) = mn\mu/M$. Assume that a bit-flip happened in the memory of one of the modules; this corrupted memory occupies $m/3$ bits of the total memory requirements. So, in order to have a failure, another bit-flip must happen in the remaining $2m/3$ bits of the other two healthy modules. The probability of this event to happen is then $2m/(3M)$. Consequently, the probability for the second bit-flip is given by $P(M_2) = 2mn\mu/(3M)$. Since both events must occur, the final probability to have a memory failure in `FullTMR` is

$$P(F_M) = 2m^2n^2\mu^2/(3M^2). \tag{1}$$

The probability of register failure in `FullTMR` is defined exactly the same way as the memory one. A failure will happen with the occurrence of one bit-flip in a register of one of the modules, and a second bit-flip in any of the registers of the other two modules. The total register usage of TMR is r bits out of R bits available. Given that n clock cycles to compute a SHA-2 operation, and that the chip suffers a register bit-flip rate of ρ per clock cycle, the probability of the first bit-flip in a register is $P(R_1) = rn\rho/R$. Furthermore, the probability of the second bit-flip is given by $P(R_2) = 2rn\rho/(3R)$. As a result, the final probability of having a register failure in `FullTMR` is

$$P(F_R) = 2r^2n^2\rho^2/(3R^2). \tag{2}$$

TMR with Shared Encoded Memory

In order to achieve a higher protection level for the memory, each memory position was encoded using Hamming codes. The condition for failure in `TMR&HCMem` is to have two bit-flips happening in the same memory element. Let the total encoded memory be m bits out of M bits available. The probability of a particle to hit the encoded memory element is m/M. Since `TMR&HCMem` takes n clock cycles to compute a SHA-2, and considering a bit-flip rate of μ per clock cycle, the probability of the first memory bit-flip is $P(M_1) = mn\mu/M$. Given that each memory position is individually encoded, the condition for failure is to have a second bit-flip in the same memory location that received the first bit-flip. Assume that each encoded memory location utilizes l bits, and that the first bit-flip happened in one of these l bits. Then, a failure will happen if a second bit-flip happens in the remaining $(l-1)$ bits, so that probability of that occurring is $(l-1)/M$. As a consequence, the probability of the second bit-flip is given by $P(M_2) = (l-1)n\mu/M$. Thus, as a result, the final probability of having a memory failure in `TMR&HCMem` is

$$P(F_M) = m(l-1)n^2\mu^2/M^2. \tag{3}$$

Since `TMR&HCMem` is still a TMR-based scheme for registers, the probability of having a register failure is exactly the same as in `FullTMR`, i.e. is given by Equation (2).

TMR for Registers and Shared Encoded Memory

`TMRRegs&HCMem` uses the same encoded memory as in `TMR&HCMem`. Therefore the probability of a memory failure is given by Equation (3). Despite the fact that `TMRRegs&HCMem` is also a TMR-based scheme, the redundancy is implemented at the register level. Thus, the probability of register failure is slightly different from the other TMR schemes. In order to have a register failure in `TMRRegs&HCMem`, two bit-flips must occur in the same register and in the same clock cycle.

Consider a total register usage of r bits out of R bits available, and a register bit-flip rate of ρ per clock cycle. Notice that in `FullTMR` and `TMR&HCMem`, errors occurring in the middle processing must wait n clock cycles to be masked out by the voter. Now, in `TMRRegs&HCMem`, errors are masked in every clock cycle for each trio of registers. So, the bit-flip analysis is now performed in a single clock cycle. As a result, the probability of the first register bit-flip is $P(R_1) = r\rho/R$. The failure condition for `TMRRegs&HCMem` is to have two registers corrupted in the trio, so that a voter would not be able to decide for the correct result. Thus, assume that a trio of registers occupies t bits, and that one of these registers suffered a bit-flip. The probability of having a second register corrupted in the trio, is $2t/(3R)$. Hence, the probability of the second bit-flip to happen is given by $P(R_2) = 2t\rho/(3R)$. Thus, the final probability of register failure in `TMRRegs&HCMem` is

$$P(F_R) = 2tr\rho^2/(3R^2). \tag{4}$$

Encoding/Decoding All Registers with Hamming Codes

Even though `HCAllRegs` adopts a totally different fault-tolerant technique, it uses the same encoded memory as in `TMR&HCMem`. Then, the probability of having a memory failure is given by Equation (3). To save parity bits, this scheme merges all the registers before encoding. Then, the resulting Hamming-encoded register can be treated as a single element occupying r bits out of R bits available in the hardware device. Since this single register is decoded and re-encoded in every clock cycle, a failure will occur only if two bit-flips happen in the encoded register during the time frame of 1 clock cycle.

The probability of having a bit-flip in the encoded register is r/R. By considering a single clock cycle period to have a bit-flip and a bit-flip rate of ρ per clock cycle, the probability of the first register bit-flip is $P(R_1) = r\rho/R$. Assuming that a first bit-flip happened, the condition for failure is to have a second bit-flip in the encoded register. Now, there are $(r-1)$ bits that may suffer a bit-flip. As a consequence, probability of having a second bit-flip in the encoded register is $P(R_2) = (r-1)\rho/R$. Subsequently, the final probability of register failure in `HCAllRegs` is

$$P(F_R) = (r^2 - r)\rho^2/R^2. \qquad (5)$$

Encoding/Decoding Main Registers with Hamming Codes

Although `HCAllRegs` is an optimization of `HCAllRegs` which reduces the register requirements, it also uses the same encoded memory as `HCAllRegs` and `TMR&HCMem`. Thus, the probability of having a memory failure is given by Equation (3). In `HCMainRegs`, though, all registers are encoded individually through the use of Hamming codes.

The condition for a register failure is to have two bit-flips in the same register ($a, ..., h$, H_0, H_4, W_0, W_1, W_9 or W_{14}) while they are in their idle period. Suppose that all encoded registers employ r bits out of the R bits available in the hardware device. Then, the probability of a bit-flip in any bit of the encoded registers is r/R. Further, assume a bit-flip rate of ρ per clock cycle. In this scheme the registers are kept encoded all the time, but are not decoded and re-encoded in every clock cycle. Instead of that, they have a idle period of time, which we define as i. More precisely, i is the maximum number of clock cycles without performing error detection and correction on the data present in the registers. Therefore, the probability of the first bit-flip in a register is $P(R_1) = ri\rho/R$. To have a failure, a second bit-flips must occur in the same encoded register during i clock cycles. Furthermore, consider that a first bit-flip has already occurred in an encoded register, and that each encoded register uses g bits. Then, the second bit-flip must happen in one of the $(g-1)$ remaining bits. Thus, the probability of having a second bit flip in the same encoded register is $P(R_2) = (g-1)i\rho/R$. As a result, the final probability of register failure in `HCMainRegs` is

$$P(F_R) = r(g-1)\rho^2 i^2/R^2. \qquad (6)$$

Device-Specific Probability of Failure

In order to employ the aforementioned equations to perform an analysis of device-specific probability of failure, it is necessary to consider the parameters defining the target hardware device, such as the total number of registers (R) and memory (M) available. Given that we use an Altera CycloneII EP2C35F672C6 FPGA to obtain our experimental results, the same device was used to conduct the analysis in this section. Moreover, information on the design of each scheme must also be known. The memory and register requirements are shown in Tables 1 and Tables 2, respectively. The specific values for all variables defined while determining the probability equations are listed in Table 3.

Memory Analysis—The memory usage (m) for each scheme is listed in Table 1. By using Table 3 and the equations provided in this section, it is possible to determine, in terms of μ^2, the probability of memory failure of all fault tolerant schemes. The results for all schemes are organized in Table 4.

Table 3. Variables used for Resistance Analysis

Variable	SHA224/256	SHA384/512
M (bits)	483840	
R (bits)	33216	
n (clock cycles)	64	80
i (clock cycles)	60	76
l (bits)	38	71
t (bits)	96	192
g (bits)	38	71

From Equation (1), it results that the probabilities of memory failure of `FullTMR` are 557.28x10$^{-3}\mu^2$ and 5202.99x10$^{-3}\mu^2$, respectively, for SHA224/256 and SHA384/512. Since `TMR&HCMem`, as well as all the remaining schemes, use a Hamming-encoded memory, their probability of memory failure are given by Equation (3). More precisely, the Hamming-encoded memory decreases the probability of memory failure to 1.77x10$^{-3}\mu^2$ and 11.96x10$^{-3}\mu^2$, respectively, for SHA224/256 and SHA384/512.

Table 4. Probability of Memory Failure

Scheme	SHA224/256 (x10$^{-3}\mu^2$)	SHA384/512 (x10$^{-3}\mu^2$)
FullTMR	557.28	5202.99
TMR&HCMem	1.77	11.96
TMRRegs&HCMem	1.77	11.96
HCAllRegs	1.77	11.96
HCMainRegs	1.77	11.96

Table 5. Normalized Memory Resistance

Scheme	SHA224/256	SHA384/512
FullTMR	1	1
TMR&HCMem	314.63	435.15
TMRRegs&HCMem	314.63	435.15
HCAllRegs	314.63	435.15
HCMainRegs	314.63	435.15

In order to have a better picture of the resistance provided by the Hamming-encoded memory, a normalized analysis is performed taking as reference the triplicated memory of `FullTMR`. From Table 5, it is possible to conclude that, by encoding the memory, the memory resistance against SEUs of SHA224/256 is increased 314.63 times. This increase is even higher for SHA384/512, i.e. their memory become 435.15 times more resistant by using Hamming codes instead of TMR.

Register Analysis—Similar analysis for the registers can be done by utilizing the register usage (r) listed in Table 2 along with Table 3. Equation (2) provides us the probability of register failure for `FullTMR` and `TMR&HCMem`. When SHA224/256 and SHA384/512 employ those schemes, the probabilities of register failure are 23356.97x10$^{-3}\rho^2$ and 145981.04x10$^{-3}\rho^2$, respectively. The probabilities

Table 6. Probability of Register Failure

Scheme	SHA224/256 ($\mathbf{x10^{-3}\rho^2}$)	SHA384/512 ($\mathbf{x10^{-3}\rho^2}$)
FullTMR	23356.97	145981.04
TMR&HCMem	23356.97	145981.04
TMRRegs&HCMem	0.18	0.71
HCAllRegs	0.97	3.84
HCMainRegs	146.81	832.61

Table 7. Normalized Register Resistance

Scheme	SHA224/256	SHA384/512
FullTMR	1	1
TMR&HCMem	1	1
TMRRegs&HCMem	131072	204800
HCAllRegs	24079.65	37972.36
HCMainRegs	159.1	175.33

of register failures for all schemes are listed in Table 6. Now, if `TMRRegs&HCMem` is used, Equation (4) determines that the probabilities of register failures are 0.18x$10^{-3}\rho^2$ and 0.71x$10^{-3}\rho^2$, respectively, for SHA224/256 and SHA384/512. Furthermore, when `HCAllRegs` is adopted, Equation (5) determines that the probabilities of register failure for SHA224/256 and SHA384/512 are, respectively, 0.97x$10^{-3}\rho^2$ and 3.84x$10^{-3}\rho^2$. Lastly, Equation (6) determines the probabilities of failure for SHA224/256 and SHA384/512 when using `HCMainRegs`, which are, respectively, 146.81x$10^{-3}\rho^2$ and 832.61x$10^{-3}\rho^2$.

If a normalized analysis is performed taking as reference `FullTMR`, it is easy to observe, from Table 7, that `TMRRegs&HCMem` increases the register resistance of SHA224/256 and SHA384/512 by a factor of 131072 and 204800, respectively. Another scheme with a good register protection is `HCAllRegs`. Precisely, that scheme increases the register resistance of SHA224/256 and SHA384/512, respectively, by a factor of 24079.65 and 37972.36. If `HCMainRegs` is employed, the register resistance is 159.1 and 175.33 times higher, respectively, for SHA224/256 and SHA384/512.

7. Experimental Results

In order to better analyze the advantages of each scheme proposed in Section 5 it is necessary to evaluate them in terms of implementation area, throughput, frequency of operation, and power consumption. Although all the schemes discussed here can be implemented using both ASICs and FPGAs (Flash, Anti-Fuse, SRAM), we selected an SRAM FPGA capable of perform CRC checks automatically: an Altera CycloneII EP2C35F672C6 FPGA. Hence, we described the proposed schemes using VHDL, and performed the FPGA implementation. The tool employed in the description, synthesis, simulation and power estimation of all hardware modules was QuartusII [39] version 7.2, service pack 1.

We conducted the synthesis targeting low implementation area and low power consumption. In order to minimize the power consumption even further, we performed a two-step synthesis and simulation procedure. Once the first synthesis and simulation were complete, a signal activity file was created. This file was then fed back to the tool allowing for better synthesis optimization, thus leading to additional power savings. The data shown in Tables 8, 9, and 10 reflect the results of the final synthesis and simulation. Table 11 shows dynamic power resulting from the simulation of the modules at their maximum frequency of operation (F_{max}), whereas Table 12 provides the normalized power estimates with all modules running at 33.33MHz.

Area Results

Implementation area is measured in terms of the number of logic elements (LEs) used to implement a given scheme in the FPGA. According to Table 8, the non-fault tolerant (`NoFT`) SHA-224 occupies 1559 LEs. As expected, in comparison with the `NoFT`, `FullTMR` occupies slightly more than three times as much area (4709 LEs). Now, considering the first design adopting shared encoded memory, `TMR&HCMem`, it is easy to observe that it uses slightly more area than `FullTMR` (4843 LEs). Supposedly, the `TMRRegs&HCMem` design should be smaller than `TMR&HCMem`, given that the SHA-2 datapath is not triplicated. However, due to the number of voters used, LEs are used exclusively for implementing all the logic needed. It utilizes 6234 LEs, which means four times the area of `NoFT`, and 1.3 times the area of `FullTMR`. The `HCAllRegs` design is quite inefficient in terms of area (6507 LEs), i.e. it is more than four times bigger than `NoFT`, and has 1.38 times the size of `FullTMR`. Finally, a design with relatively small implementation area (3617 LEs) is `HCMainRegs`. `HCMainRegs` employs only 2.3 times as much area as `NoFT`, and employs approximately 3/4 of the area of `FullTMR`.

By analyzing Table 8, it is possible to observe certain trends in the SHA-2 area utilization. `FullTMR` occupies about 3 times as much area as `NoFT`, while `TMR&HCMem` is slightly bigger than `FullTMR`. Further, both `TMRRegs&HCMem` and `HCAllRegs` employs, on average, 3.9 times more area than `NoFT`, and thus are more inefficient than `FullTMR`. `HCMainRegs` provides the higher area efficiency. On average, it utilizes 2.2 times as much area as `NoFT`, and less than 3/4 of the area of `FullTMR`.

Frequency Results

The frequency of operations of the schemes considered in this work is presented in Table 9. By analyzing the SHA-224 results in that table, it is possible to notice that the fault-tolerant scheme with higher frequency of operation is `FullTMR`. Due to the high parallelism involved it can operate at 73.05MHz. Because `TMR&HCMem` uses a Hamming decoder between the memory and the TMR part of the module, its critical path is impacted negatively. As a consequence, it operates at a

Table 8. Area Results

Scheme	**SHA-224 (LEs)**	**SHA-256 (LEs)**	**SHA-384 (LEs)**	**SHA-512 (LEs)**
NoFT	1559	1591	3275	3341
FullTMR	4709	4807	9896	10084
TMR&HCMem	4843	4919	10161	10288
TMRRegs&HCMem	6234	6232	12379	12377
HCAllRegs	6507	6494	12419	12365
HCMainRegs	3617	3657	6914	6897

Table 9. Frequency Results

Scheme	**SHA-224 (MHz)**	**SHA-256 (MHz)**	**SHA-384 (MHz)**	**SHA-512 (MHz)**
NoFT	76.24	79.18	61.65	60.58
FullTMR	73.05	71.69	59.99	60.66
TMR&HCMem	47.44	46.93	38.71	38.80
TMRRegs&HCMem	45.41	45.28	39.08	35.58
HCAllRegs	28.52	28.93	17.42	20.73
HCMainRegs	42.92	42.93	37.73	35.03

Table 10. Throughput Results

Scheme	**SHA-224 (Mbps)**	**SHA-256 (Mbps)**	**SHA-384 (Mbps)**	**SHA-512 (Mbps)**
NoFT	443.58	455.51	612.91	590.80
FullTMR	425.02	412.42	596.41	591.58
TMR&HCMem	276.01	269.98	384.85	378.39
TMRRegs&HCMem	264.20	260.49	388.52	346.99
HCAllRegs	165.93	166.43	173.19	202.17
HCMainRegs	249.72	246.97	375.10	341.63

lower frequency than `FullTMR`, i.e, 47.44MHz. Besides a memory decoder, `TMRRegs&HCMem` has also its frequency of operation impacted negatively by voters, so that it operates at 45.41MHz. Since `HCAllRegs` performs an encoding and a decoding operation in each clock cycle, its frequency of operation is dramatically reduced to 28.52MHz. By encoding and decoding the main registers only, as it is done in `HCMainRegs`, a frequency of operation of 42.92MHz is achieved.

By observing the results for the other SHA-2 algorithms in Table 9, it is possible to realize that `FullTMR` is in fact the fault tolerant scheme that provides higher frequencies of operation. Although, `TMR&HCMem` and `TMRRegs&HCMem` have similar frequencies of operation, the former is slightly faster than the latter. Further, these two modules are followed quite closely by `HCMainRegs`. The slower scheme in all cases, though, is `HCAllRegs`.

Throughput Results

For the purpose of simulation and throughput estimation, we consider the hash of only one block of message. The block size is 512 bits for SHA224/256, and 1024 bits for SHA384/512. More precisely, the throughput is defined as: $message\ block\ size/(\#cycles/F_{max})$. In order to compute a message digest, SHA-224, SHA-256, SHA-384 and SHA-512 take, respectively, 88, 89, 103, and 105 clock cycles. The number of clock cycles reported include the complete SHA-2 processing, as well as the time spent writing/reading data to/from the module.

As shown in Table 10, the frequency of operation has a strong influence on the throughput, i.e. the higher the frequency, the higher the throughput. In fact, SHA-224 using `FullTMR` has highest throughput among the fault tolerant modules. Its throughput (425.02Mbps) is 96% of the `NoFT` throughput (443.58Mbps). Further, the throughput of `TMR&HCMem` (276.01Mbps) and `TMRRegs&HCMem` (264.20Mbps) achieve, respectively, 62% and 60% of the `NoFT` throughput. `HCAllRegs` presents a relative low throughput (165.93Mbps), which is 37% of the `NoFT` throughput. Moreover, `HCMainRegs` presents a throughput of 249.72Mbps, which is comparable with the ones of `TMR&HCMem` and `TMRRegs&HCMem`. This represents 56% of the `NoFT` throughput.

Following the same analysis, the throughput of the SHA-256, SHA-384 and SHA-512 using `FullTMR` are, on average,

Table 11. Dynamic Power Consumption Results

Scheme	SHA-224 (mW)	SHA-256 (mW)	SHA-384 (mW)	SHA-512 (mW)
NoFT	78.17	81.73	147.59	143.95
FullTMR	222.04	228.27	430.52	426.18
TMR&HCMem	146.58	148.35	297.13	294.73
TMRRegs&HCMem	149.89	144.9	261.6	252.22
HCAllRegs	267.94	290.2	511.42	541.54
HCMainRegs	125.04	126.18	249.84	241.63

Table 12. Normalized Dynamic Power Consumption Results

Scheme	SHA-224 (mW)	SHA-256 (mW)	SHA-384 (mW)	SHA-512 (mW)
NoFT	36.06	34.96	83.9	81.69
FullTMR	103.57	107.18	244.3	241.83
TMR&HCMem	108.66	108.95	256.82	255.01
TMRRegs&HCMem	115.73	111.28	266.55	243.53
HCAllRegs	-	-	-	-
HCMainRegs	99.84	101.07	224.32	233.31

higher than 90% of the `NoFT`. On average, `TMR&HCMem`, `TMRRegs&HCMem` and `HCMainRegs` have similar relative throughput. Precisely, they achieve respectively 62%, 60% and 57% of the `NoFT` throughput.

Power Results

Power consumption is another important factor to be analyzed in space systems. Table 11 reports the dynamic power consumption of the implementations performing one hash computation at their maximum frequency of operation. The SHA-224 `NoFT` design consumes 78.17mW. When `FullTMR` is used, its power consumption is 2.8 times higher, i.e. 222.04mW. This is an expected result, given that it triplicates the SHA-224 datapath. By using an encoded memory, `TMR&HCMem` and `TMRRegs&HCMem` consume 1.9 times as much power as `NoFT`, respectively, 146.58mw and 149.89mW. Moreover, `HCAllRegs` consumes 267.94mW, i.e. 3.4 times more power than `NoFT`. The scheme with the least power consumption (125.04mW) is `HCMainRegs`. It consumes 1.6 times as much power as `NoFT`.

The overall power increase of `FullTMR` is slightly less than 3 times the power of `NoFT`. In addition, `TMR&HCMem` uses on average twice as much power as `NoFT`, whereas `TMRRegs&HCMem` uses 1.9 times more power than `NoFT`. The power increase caused by `HCMainRegs` is on average 1.6 times than the `NoFT` one.

In order to provide a fair comparison among the implementations, we performed a normalized power estimation by running all the designs at a common frequency of 33.33MHz. Given that `HCAllRegs` has a frequency of operation lower than 33.33MHz, they were not included in this comparison. At 33.33MHz, the throughput of SHA-224, SHA-256, SHA-384, and SHA-512 are 193.94, 191.76, 331.39, and 325.08Mbps, respectively.

The normalized power results are listed in Table 12. The non-fault tolerant (`NoFT`) SHA-224 implementation consumes 36.06mW of dynamic power to generate a message digest. Further, SHA-224 `FullTMR` consumes 103.57mW, which is slightly less than three times the dynamic power of `NoFT`. Furthermore, `TMR&HCMem` and `TMRRegs&HCMem` normalized power consumptions are, respectively, 108.66mW and 115.73mW. This represents increases of 3 and 3.2 times in the power consumption of `TMR&HCMem` and `TMRRegs&HCMem`, respectively, compared to `NoFT`. Once again, `HCMainRegs` provides the lowest power consumption among all fault tolerant schemes, which is 99.84mW. In other words, `HCMainRegs` consumes 2.8 times as much power as `NoFT` at 33.33MHz.

By looking at the normalized power consumption of SHA-256, SHA-384 and SHA-512 in Table 12, one can conclude that `FullTMR`, `TMR&HCMem` and `TMRRegs&HCMem` uses, on average, 3 times as much normalized power as `NoFT`. `HCMainRegs`, in turn, leads to an average normalized power increase of 2.8 times, compared to `NoFT`.

Discussion

This section highlights the most important improvements brought by the proposed schemes compared to TMR. The benefits of using an encoded memory is twofold. First, as shown in Table 1, it uses 60% less memory than `FullTMR`. Second, as listed in Table 5, the memory becomes 314.63 times more resistant against SEUs in SHA224/256, and 435.15 times more resistant in SHA384/512.

`TMRRegs&HCMem` offers the highest level of protection against SEUs. More precisely, as listed in Table 7, the registers become 131072 times more resistant against SEUs in SHA224/256, and 204800 times more resistant in SHA384/512 when compared to `FullTMR`. However this scheme employs 1.3 more area than `FullTMR` to achieve higher levels of resistance to SEUs.

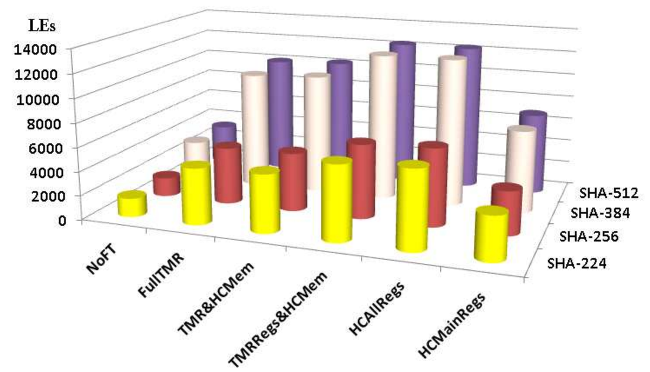

Figure 7. Comparison of Implementation Area.

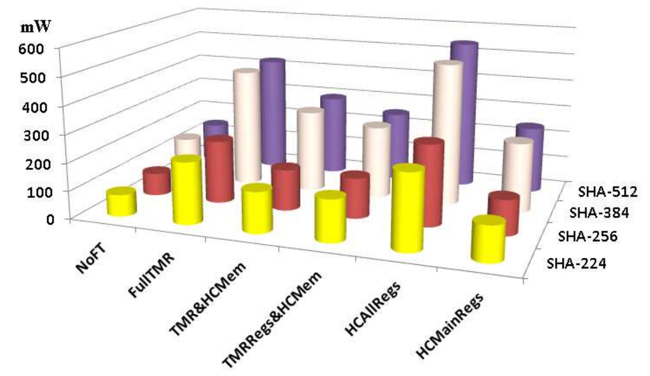

Figure 8. Comparison of Power Consumption.

`HCAllRegs` also offers a high protection against SEUs, but as can be noticed from Figures 7 and 8, that it is the most inefficient scheme in terms of area, throughput, and power consumption.

The best trade-off among implementation area, power consumption and protection against SEUs is achieved with `HCMainRegs`. As depicted in Figures 7 and 8, this scheme uses up to 32% less area and consumes up to 43% less power than `FullTMR`. Further, it employs up to 63% less registers than `FullTMR`. Additionally, when `HCMainRegs` is applied to SHA224/256 and SHA384/512 their registers become, respectively, 159.1 and 175.33 times more resistant against SEUs than when using `FullTMR`. Also, the memory of SHA224/256 and SHA384/512 become, respectively, 314.63 and 435.15 times more resistant than the one in `FullTMR`.

8. Conclusions

The paper proposes fault tolerant schemes for the SHA-2 family of hash functions, providing both error detection and correction. Although all schemes can be applicable to ASICs and FPGAs (SRAM, Flash, Anti-Fuse), we implemented them in an SRAM FPGA to perform their evaluation in terms of area, frequency of operation, throughput, and power consumption. Additionally, a comprehensive analysis of the resistance of the schemes against SEUs is performed.

For the sake of comparison we implemented the traditional TMR and we showed that this fault tolerance technique applied to SHA-2 hash function demands three times as much area resource as a non-fault tolerant approach. The proposed scheme named `HCMainRegs` provides a better trade-off for area and power consumption and improves the resistance to errors caused by SEUs. For instance, SHA-512 adopting `HCMainRegs` employs 6897 LEs and 6248 memory bits, and has a dynamic power consumption of 241.63mW. By comparing with `NoFT`, those results represent area and power increases of 2.1 and 1.7 times, respectively. However, `HCMainRegs` uses up to 32% less area and consumes up to 43% less power than `FullTMR`. Moreover, its memory and registers are, respectively, 435 and 175 times more resistant against SEUs than they would be by using `FullTMR`.

As a result, `HCMainRegs` can successfully replace TMR for achieving fault tolerance in the SHA-2 family of hash functions. Besides, it provides higher levels of protection against SEUs, as well as favors low power and low implementation area, which are crucial in space applications. To the best of our knowledge, this is the first implementation and analysis of the SHA-2 family of hash functions providing both error detection and correction reported in the literature.

Acknowledgments

This research is supported in part by grants from NSERC and OCE. We would like to thank Kryptus Technologies for providing some golden models used to validate our architectures.

References

[1] "Critical infrastructure protection: Commercial satellite security should be more fully addressed," US General Accounting Office, Tech. Rep. GAO-02-781, 2002.

[2] "Security threats against space missions," CCSDS, Informational Report 350.1-G-1, October 2006.

[3] "Hackers seize UK military satellite," Online, The Landfield Group, March 1999, http://www.landfield.com/isn/mail-archive/1999/Mar/0001.html.

[4] S. Fleming, "Hacker infiltrates military satellite," Online, The Register, March 1999, http://www.theregister.com/1999/03/01/hacker_infiltrates_military_satellite/.

[5] "British hackers attack MoD satellite," Online, Telegraph Newspaper, March 1999, http://www.telegraph.co.uk/connected/main.jhtml?xml=/connected/1999/03/04/ecnhack04.xml.

[6] K. Poulsen, "Satellites at risk of hacks," Security Focus, October 2002.

[7] M. Juliato and C. Gebotys, "An approach for recovering satellites and their cryptographic capabilities in the presence of SEUs and attacks," in *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2008)*, June 2008, pp. 101–108.

[8] T. Vladimirova, R. Banu, and M. Sweeting, "On-board security services in small satellites," in *MAPLD 2005 Proceedings*, 2005.

[9] G. Messenger and M. Ash, *Single Event Phenomena*. Springer, 2006.

[10] A. Menezes, P. Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.

[11] D. Stinson, *Cryptography Theory and Practice*, 3rd ed. CRC Press, 2005.

[12] "The keyed-hash message authentication code (HMAC)," NIST, Federal Information Processing Standards Publication FIPS PUB 198, March 2002.

[13] "Secure hash standard (SHS)," NIST, Federal Information Processing Standards Publication FIPS 180-3, October 2008.

[14] "Authentication / integrity algorithm issues survey," CCSDS, Informational Report CCSDS 350.1-G-1, March 2008, green Book.

[15] "Encryption algorithm trade survey," CCSDS, Informational Report CCSDS 350.2-G-1, March 2008, green Book.

[16] B. Badrignans, R. Elbaz, and L. Torres, "Secure FPGA configuration architecture preventing system downgrade," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2008, pp. 317–322.

[17] I. Ingemarsson and C. Wong, "Encryption and authentication in on-board processing satellite communication systems," *IEEE Transactions on Communications*, vol. 29, no. 11, pp. 1684–1687, November 1981.

[18] A. Roy-Chowdhury, J. Baras, M. Hadjitheodosiou, and S. Papademetriou, "Security issues in hybrid networks with a satellite component," *IEEE Wireless Communications*, vol. 12, no. 6, pp. 50–61, 2005.

[19] M. Arslan and F. Alagoz, "Security issues and performance study of key management techniques over satellite links," in *Proceedings of the 11th Intenational Workshop on Computer-Aided Modeling, Analysis and Design of Communication Links and Networks*, 2006, pp. 122–128.

[20] D. Fischer, M. Merri, and T. Engel, "Key management for CCSDS compliant space missions," in *Proceedings of the International Conference on Space Operations (Spaceops 2008)*, May 2008.

[21] M. McLoone and J. V. McCanny, "Efficient single-chip implementation of SHA-384 and SHA-512," in *Proceedings of the 2002 IEEE International Conference on Field-Programmable Technology (FPT'02)*, 2002, pp. 311–314.

[22] K. K. Ting, S. C. L. Yuen, K. H. Lee, and P. H. W. Leong, "An FPGA based SHA-256 processor," in *Proceedings of the Reconfigurable Computing Is Going Mainstream, 12th International Conference on Field-Programmable Logic and Applications (FPL '02)*. Springer-Verlag, 2002, pp. 577–585.

[23] N. Sklavos and O. Koufopavlou, "On the hardware implementations of the SHA-2 (256, 384, 512) hash functions," *Proceedings of the 2003 International Symposium on Circuits and Systems (ISCAS'03).*, vol. 5, pp. V–153–V–156, May 2003.

[24] ——, "Implementation of the SHA-2 hash family standard using FPGAs," *The Journal of Supercomputing*, vol. 31, no. 3, pp. 227–248, 2005.

[25] T. Grembowski, R. Lien, K. Gaj, N. Nguyen, P. Bellows, J. Flidr, T. Lehman, and B. Schott, "Comparative analysis of the hardware implementations of hash functions SHA-1 and SHA-512," in *Proceedings of the 5th International Conference on Information Security ISC'02*. Springer-Verlag, 2002, pp. 75–89.

[26] I. Ahmad and A. S. Das, "Hardware implementation analysis of SHA-256 and SHA-512 algorithms on FPGAs," *Computers and Electrical Engineering*, vol. 31, no. 6, pp. 345–360, 2005.

[27] R. P. McEvoy., F. M. Crowe, C. C. Murphy, and W. P. Marnane, "Optimisation of the SHA-2 family of hash functions on FPGAs," *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures, 2006*, vol. 00, p. 6 pp., March 2006.

[28] H. Michail, A. Kakarountas, G. N. Selimis, and C. E. Goutis, "Optimizing SHA-1 hash function for high throughput with a partial unrolling study," in *PATMOS 2005 Proceedings*, 2005, pp. 591–600.

[29] R. Chaves, G. Kuzmanov, L. Sousa, and S. Vassiliadis, "Improving SHA-2 hardware implementations," in *Proceedings of the Cryptographic Hardware and Embedded Systems (CHES 2006)*, 2006, pp. 298–310.

[30] I. Ahmad and A. S. Das, "Analysis and detection of errors in implementation of SHA-512 algorithms on FPGAs," *The Computer Journal*, vol. 50, no. 6, pp. 728–738, 2007.

[31] "Advanced encryption standard (AES)," NIST, Federal Information Processing Standards Publication FIPS PUB 197, November 2001.

[32] S. Ghaznavi and C. Gebotys, "A SEU-resistant, FPGA-

based implementation of the substitution transformation in AES for security on satellites," in *Proceedings of the Tenth International Workshop on Signal Processing for Space Communications (SPSC 2008)*, October 2008.

[33] *RTAX-S/SL RadTolerant FPGAs*, 5th ed., Actel Corporation, October 2008.

[34] *Radiation-Tolerant ProASIC3 Low-Power Space-Flight Flash FPGAs*, Actel Corporation, September 2008.

[35] *Radiation-Hardened Virtex-4 QPro-V Family Overview*, 1st ed., Xilinx Inc., March 2008, DS653.

[36] *Error Detection and Recovery Using CRC in Altera FPGA Devices*, Altera Corporation, July 2008, AN 357.

[37] P. Blain, C. Carmichael, E. Fuller, and M. Caffrey, "SEU mitigation techniques for Virtex FPGAs in space applications," in *MAPLD 1999 Proceedings*, 1999.

[38] P. Samudrala, J. Ramos, and S. Katkoori, "Selective triple modular redundancy (STMR) based single-event upset (SEU) tolerant synthesis for FPGAs," *IEEE Transactions on Nuclear Science*, vol. 51, pp. 2957–2969, 2004.

[39] *Quartus II Version 8.0 Handbook*, Altera Corporation, May 2008.

BIOGRAPHY

Marcio Juliato received his bachelor's degree in Computer Engineering in 2003, and his M.Sc. degree in Computer Science in 2006, both from the University of Campinas, Campinas, Brazil. Marcio is currently a PhD candidate in Computer Engineering in the Dept. of Electrical and Computer Engineering, University of Waterloo, Waterloo, Canada. He is also a member of the Center for Applied Cryptography Research (CACR) at the University of Waterloo. His research interests include cryptography, computer architecture, fault tolerance, and efficient implementations of security mechanisms for embedded and space systems.

Catherine Gebotys received her B.A.Sc. degree in Engineering Science in 1982 and her M.A.Sc. degree in Electrical Engineering in 1984, both from University of Toronto, Toronto, Canada. She received her PhD degree in Electrical Engineering in 1991 from the University of Waterloo, Waterloo, Canada. She is currently a Professor in the Dept. of Electrical and Computer Engineering, University of Waterloo, Canada. Her research interests include embedded systems security, side channel (power/electromagnetic) analysis of cryptographic algorithms, and reconfigurable architectures.

Reouven Elbaz received his M.Sc. degree in Electrical Engineering in 2003 from Polytech'Montpellier, France, and his PhD degree in Electrical and Computer Engineering in 2006 from the University of Montpellier, France. During his PhD studies, he was also with STMicroelectronics within the Security Group of AST (Advanced System Technology) Rousset, France working on trusted computing in embedded systems. Then, he joined the Dept. of Electrical Engineering at Princeton University, USA, for eighteen months as a research associate. He is now research associate in the Dept. of Electrical and Computer Engineering, University of Waterloo, Waterloo, Canada. His research interests include trusted computing and security, cryptography, reconfigurable architectures, computer architecture and parallel processing.