esa

DOCUMENT

document title/ titre du document

System Knowledge Based Protection Techniques Against SEEs caused by space radiation for Fast Fourier Transforms

prepared by/*préparé par*

reference/*réference*
issue/*édition*
revision/*révision*
date of issue/*date d'édition*
status/*état*
Document type/*type de document*
Distribution/*distribution*

European Space Agency
Agence spatiale européenne

ESTEC
European Space Research and Technology Centre - Keplerlaan 1 - 2201 AZ Noordwijk - The Netherlands
Tel. (31) 71 5656565 - Fax (31) 71 5656040 www.esa.int

FFT_FT_Preyes

APPROVAL

Title *Titre*	System Knowledge Based Protection Techniques Against SEE caused by radiation effects for Fast Fourier Transforms	issue *issue*	revision *revision*

author *auteur*	María del Pilar Reyes Moreno	date *date*

approved by *approuvé par*	date *date*

CHANGE LOG

reason for change */raison du changement*	issue/*issue*	revision/*revision*	date/*date*

CHANGE RECORD

Issue: Revision:

reason for change/*raison du changement*	page(s)/*page(s)*	paragraph(s)/*paragraph(s)*

Table Of Contents

1 SCOPE

The scope of this document is to present some protection techniques against *Single Event Effects* (SEEs) based on the *System Knowledge* for one specific type of *Digital Signal Processing* (DSP) circuits as *Fast Fourier Transform*. These error mitigation techniques show a different way to protect digital designs with respect to the traditional systematic soft error mitigation mechanisms as TMR, FTMR or EDAC Codes. However, as it could be extracted from the rest of the document no optimal fault tolerant FFT implementations (in area cost terms) have been found due to the extremely regular and symmetric structure of the FFT.

2 TERMS AND ACRONYMS

SEE-Single Event Effects
SET-Single Event Transients
SEU-Single Event Upsets
TMR- Triple Modular Redundancy
FTMR- Functional Triple Modular Redundancy
EDAC- Error Detection and Correction Codes
ECC-Error Correction Codes
MBU- Multiple Bit Upset
FTU-FT-Unshades
DSP- Digital Signal Processing
DIT- Decimated in time
DIF- Decimated in frequency
ABFT-Algorithm Based Fault Tolerant
CED- Concurrent Error Detection

3 INTRODUCTION

The effects of radiation on microelectronic circuits have a number of consequences that impact the design of devices that operate in certain environments [ShFl04]. One type of these effects are Single Event Effects (SEEs), which cause changes in the values of flips-flops or combinational logic [DoMa03]. In order to mitigate the effects of SEEs, a number of techniques can be used at the physical level (device size and structure) [BaBM00]. In addition to those techniques, redundancy can be introduced in the design so that it can detect and correct SEEs, the proposed protection techniques exposed in this document are in this category. Single Event Effects (SEEs) can be classified in Single Event Upsets (SEUs) and Single Event Transients (SETs), depending on whether the change in the digital value occurs in a permanent storage element or in a combinational element. Single Event Transients (SETs) would only result in a persistent error if they propagate to the input of a storage element and they meet the setup and hold constraints there [DoSS04].

To deal with SEUs, a common hardware or space redundancy approach is Triple Modular Redundancy (TMR), which triplicates the flip-flops in the design and adds logic to vote in case of conflict. If it is necessary to deal with SETs, Functional Triple Modular Redundancy (FTMR), which also triplicates the combinational logic, can be used [Hanb02].

One advantage of both TMR and FTMR is that they are general techniques that can be applied to most digital circuits. However, this comes at a high cost in terms of circuit area and power and more so for FTMR. As they can be applied to any digital design, they can be used to protect Digital Signal Processing circuits against SEEs, as filters and Fast Fourier Transforms (FFT) designs, the kind of circuits on which this work is focused, but, as it has just been mentioned, the use of TMR or FTMR will incur in an expensive extra area overhead. Another possible solution to deal with SEEs on electronic devices could be using the *System Knowledge* concept, that consists in employing some structural or computational characteristics of the circuit under protection to obtain a fault tolerant implementation with lower extra area overhead than some of the common error mitigation techniques used in space applications as, for example, FTMR, XTMR. Therefore, in this document the analysis of different error mitigation techniques based on the concept of *System Knowledge* and applied to protect in place FFT radix-2 implementations against non destructive effects caused by radiation in space environments, as SEEs, will be performed. A comparison of these techniques with the FTMR protection in terms of area cost and fault coverage is also carried out.

As it will be seen in the rest of the document, it is difficult to obtain an optimal fault tolerant FFT implementation (in terms of area cost) although the *System Knowledge* concept is used, because of the low redundancy of the Discrete Fourier Transform (DFT) computation by mean of the FFT and its regularity or symmetry, which has a good computational efficiency thanks to this redundancy elimination. However, the solutions proposed here can be used as an alternative to FTMR in some applications (those that satisfy the requirements specified for each proposed error mitigation technique). Besides, it has been proved that the *System Knowledge* it is a good methodology that can

be used to protect efficiently other kind of DSP applications, as FIR, adaptive filters or communication system interleavers [RRMR07a], [RRMR07b], [RRMR07c], [RRMR06], and [RRMR08]. It could be also used to protect other type of DFT implementations, as the sliding DFT. As it will be extract from *Related Work* (section 5) finding a good FT-FFT design with low extra area overhead continues being an open problem.

In the rest of the document a brief introduction related to the DFT/FFT computation process and its main properties will be introduced (because the proposed protection techniques are based on the *System Knowledge* concept and the designer needs to know some characteristics about the device under protection). Next, some related work with other fault tolerant FFT implementations is shown in order to justify the work carried out, followed by the presentation of the error mitigation techniques proposed in this document, for an specific type of FFT circuits. Then, fault tolerance and area cost results are illustrated and compared with the obtained when protecting with FTMR, using a simple case study. The fault coverage analysis is accomplished by mean of using two different fault injection tools (FT-UNSHADES and SST tool). Finally, the main conclusions of this work and some proposals for the future will be presented in section 9.

4 DFT: BASICS, PROPERTIES AND IMPLEMENTATIONS

As the proposed fault tolerant techniques for FFT circuits are based on the *System Knowledge* concept, we need to know the basics of the DFT transforms, its properties and the most common implementations in order to understand the content of the rest of the document.

The time and frequency domains are alternate ways of representing signals. The Fourier Transform is the mathematical relationship between these two representations. Therefore, the Discrete Fourier Transform is the mathematical procedure used to determine the harmonic (frequency) content of a discrete signal sequence.

The DFT pair is defined in equations (1) and (2)

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{kn} \quad k = 0,1,2,...N-1 \qquad (1)$$

$$x[n] = \frac{1}{N} \cdot \sum_{k=0}^{N-1} X[k] \cdot W_N^{-kn} \quad n = 0,1,...N-1 \qquad (2)$$

$$W_N^{kn} = e^{-j\frac{2\pi}{N}kn} \qquad (3)$$

where W_N^k are the twiddle factors (complex exponential), *x[n]* is the time domain signal and *X[k]* the frequency domain representation, in this definition both *x[n]* and *X[k]* are assumed to be complex [OpSh75].

4.1 DFT hardware implementations

The DFT transform has a lot of hardware implementations, some of the most common ones are the *Decimation in Time* (DIT) and *Decimation in Frequency* (DIF) *Fast Fourier Transforms* (FFT). These implementations are based on a simple butterfly module and on a pipeline mode operation, and they need all input samples available to start the FFT computation. In the case of radix-2 FFT, each butterfly module compute two DFT output sequences. Therefore, for an *N-point DFT* calculation we will need *N/2* 2-point butterflies and $log_2(N)$ pipeline stages. Figures 1 and 2 illustrate the 8-point FFT radix-2 DIT and DIF implementations (see Appendix A for an explanation of how the FFT computes the DFT, for the case of a DIT implementation). In order to have a good performance of the FFT, the value of *N* must be a power of two.

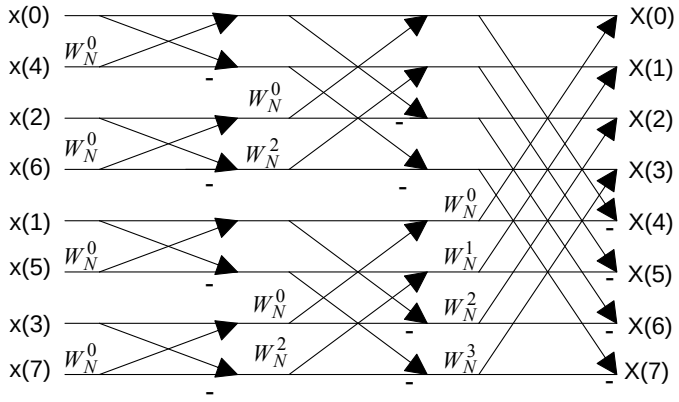

Figure 1. 8-point FFT Decimated in Time radix-2 implementation.

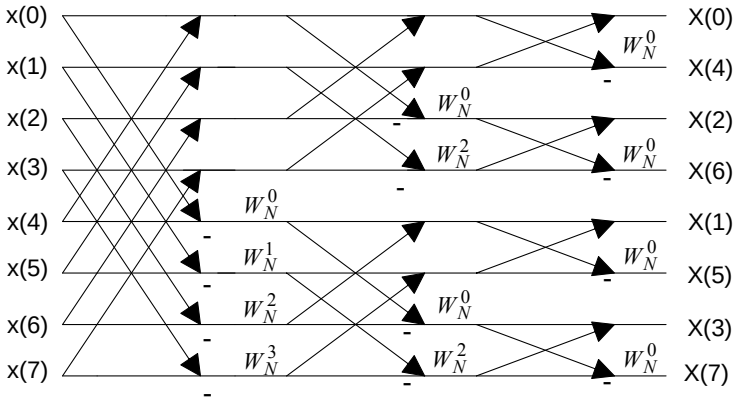

Figure 2. 8-point FFT Decimated in Frequency radix-2 implementation.

The implementations shown in figures 1 and 2 are only two examples of the multiple FFT forms. And the FFT is one of the most computationally efficient implementation of the DFT.

But, for applications whose throughput does not need to be very high, other DFT implementations can be used, as direct calculation using equations (1) and (2), Goertzel algorithm which is based on a recursive filter implementation or the Sliding DFT. Goertzel filter has lower area cost than the direct calculation and it computes a single complex DFT spectral bin value for every *N* input time samples (equation (4) represents the z-domain transfer function of Goertzel filter). Another possible interesting implementation could be the sliding DFT whose bin output rate is equal to the input data rate, on a sample by sample basis, with the advantage that it requires fewer computations than Goertzel algorithms for real time analysis [JaLy03].

$$H_G(z) = \frac{1 - e^{-j\frac{2\pi k}{N}} z^{-1}}{1 - 2\cos(\frac{2\pi k}{N}) z^{-1} + z^{-2}} \qquad (4)$$

The mentioned implementations are only examples of the multiple variations of the DFT hardware solutions.

4.2 DFT properties

In this section some of the most useful (for our purpose) DFT properties are exposed.

A. Linearity of the Fourier Transform

The Fourier Transform is linear, that is, it possesses the properties of homogeneity and additivity. Homogeneity means that a change in amplitude in one domain produces an identical change in amplitude in the other domain. In a mathematical form the linearity property can be expressed as follows:

$$\begin{aligned} x3[n] &= a \cdot x1[n] + b \cdot x2[n] \\ X3[n] &= a \cdot X1[n] + b \cdot X2[n] \end{aligned} \qquad (5)$$

B. Symmetry and periodicity properties of the twiddle factors

As twiddle factors are complex exponentials they have some properties that have been used, for example, to reduce the computational complexity of implementing the DFT by means of the FFT.

Periodicity property:

$$W_N^{k+N} = W_N^k \qquad (6)$$

Symmetry property is mathematically represented by equation (7).

$$W_N^{k+\frac{N}{2}} = -W_N^k \qquad (7)$$

C. Parseval's relation

Parseval's relation represents the energy conservation theorem what means that the energy of the input signal must be equal to the energy of the output signal.

$$\sum_{i=0}^{N-1} |x[i]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2 \qquad (8)$$

5 RELATED WORK

In critical missions, as the space ones, some levels of fault tolerance must be integrated to ensure that the results of a digital circuit in general and one FFT design in particular are valid.

The main points to optimize when designing a fault tolerant FFT circuit are:

- The degradation of the original performance should be minimal.
- Minimal error detection latency.
- The hardware overhead should be minimal.
- The detection schemes need to be able to cover small magnitude errors.

In this section we review the previous work related with some of the most interesting fault tolerant FFT implementations, most of them use a similar approach to the *System Knowledge* one to perform their protection. The presented techniques include error detection and/or error location and correction mechanisms.

5.1 Use of the FFT structure or properties to detect/correct faults

If we try to eliminate the SEE when it occurs we need to protect each FFT butterfly module. One idea, proposed by Lombardi et al. in [LoMu92], consists in using some interesting properties of the butterfly (see the basic operation of a butterfly module in figure 3) to include a fault detection mechanism as it is illustrated in the following equations.

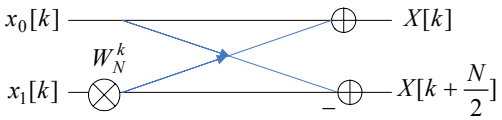

Figure 3. Radix-2 butterfly operation.

From figure 3 the next equations can be set

$$\begin{aligned} X(k) &= x_0(k) + W_N^k \cdot x_1(k) \\ X(k+N/2) &= x_0(k) - W_N^k \cdot x_1(k) \end{aligned} \tag{9}$$

Therefore,

$$X(k) + X(k+N/2) = 2 \cdot x_0(k) \tag{10}$$

If $x(k)$ are complex signals, then:

$$x_0(k) = A + jB \tag{11}$$

$$x_1(k) = C + jD \tag{12}$$

$$\mathrm{Re}[X(k)] = A + (C\cos\alpha + D\sin\alpha) \tag{13}$$

$$\mathrm{Re}[X(k+N/2)] = A - (C\cos\alpha + D\sin\alpha) \tag{14}$$

$$\mathrm{Im}[X(k)] = B + (D\cos\alpha - C\sin\alpha) \tag{15}$$

$$\mathrm{Im}[X(k+N/2)] = B - (-C\sin\alpha + D\cos\alpha) \tag{16}$$

$$\begin{aligned} \mathrm{Re}[X(k)] + \mathrm{Re}[X(k+N/2)] &= 2A \\ \mathrm{Im}[X(k)] + \mathrm{Im}[X(k+N/2)] &= 2B \end{aligned} \tag{17}$$

Therefore, the sum of the real parts of the butterfly outputs is equal to the double of the real part of the upper input signal to the butterfly, a similar result is observed for the imaginary parts. This operation can be used to check errors in the adders. However, it does not detect errors in multipliers, so extra logic must be introduced to perform the error detection mechanism in these modules. This technique has a high extra area overhead (each butterfly is 50% bigger than the non protected version). Besides, it only detects faults but not locate them directly and extra delay and logic for fault location and FFT reconfiguration purposes should be added. However, it can detect any single fault in a component block per module (butterfly) in comparison with other techniques that only detect single faults during the complete FFT computation. The extra area overhead ratio for this error detection technique is approximately $N \cdot \log_2 N / 2 \cdot N \cdot \log_2 N$. Figure 4 illustrates the fault tolerant mechanism proposed by Lombardi et al. [LoMu92].

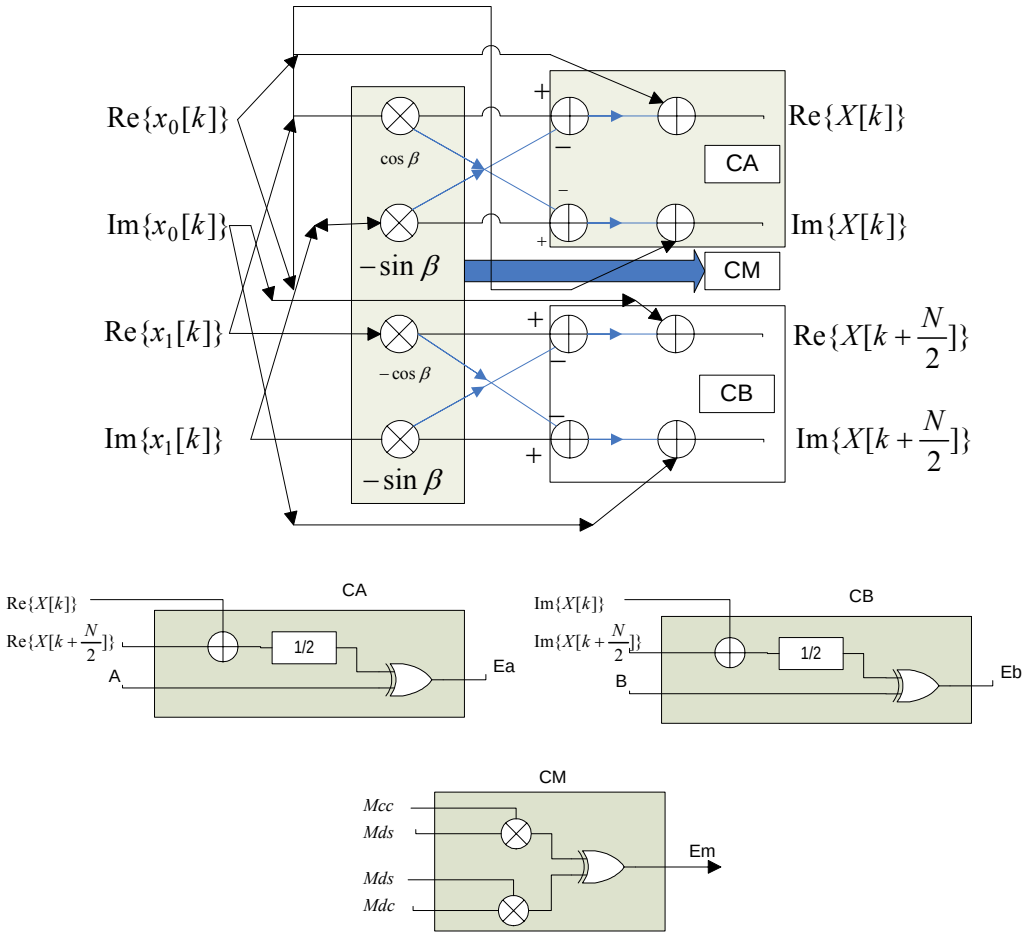

Figure 4. FFT module with concurrent error detection [LoMu92].

Other fault tolerant FFT implementations based on the butterfly architecture and several designs for easy testability have been also proposed and analyzed in several works [LLHW00], [FeML93], [AnSa91], [LuWK93], [WuCh93], [LuSH05].

5.2 Use of the DFT properties to detect/correct faults

In [JoAb88] Jou et al. propose an *Algorithm Based Fault Tolerant* implementation (ABFT) as *Concurrent Error Detection* (CED) mechanism (the ABFT solution comprises one encoder, one decoder and a TSC (Totally Self Checking) Comparator). They also introduce a fault location and correction by retry mechanism. Therefore, their fault tolerant implementation combines space redundancy for fault detection and time redundancy for fault location and correction. The CED algorithm proposed in [JoAb88] consists in using the linearity and rotation properties of the DFT to perform a checksum scheme that detects almost all possible single functional errors in the FFT circuit. Equation (18) illustrates the checksum used to detect errors in the FFT operation.

$$aX + bX^1 = A_N(ax + bx^1) \qquad (18)$$

Where a and b are scalars, X and X^1 are the transformed values of x and x^1, and the upper index ^{1} indicates that x^1 is the one step rotated version of x, A_N represents the twiddle factors matrix. (They select the values 2 and 1 for a, b scalars, respectively). Considering some simplifications they conclude that their proposed error detection mechanism has a low area overhead ratio with respect to the non protected FFT version, approximately $2/\log_2(N)$, and one delay overhead ratio of 1 more stage or cycle. They also propose a fault location and correction (data retry, reconfiguration) mechanism that incurs in a maximal error latency of $logN/3$. The proposed scheme is shown in figure 5. The disadvantage of this scheme is that different encoding schemes are needed for one-dimensional and two dimensional FFT's, and that the encoding/decoding mechanism are in serial with the FFT computation. Therefore, the FFT results and the encoder/decoder ones will have a high round off error and the fault coverage of this system will be influenced by this situation.

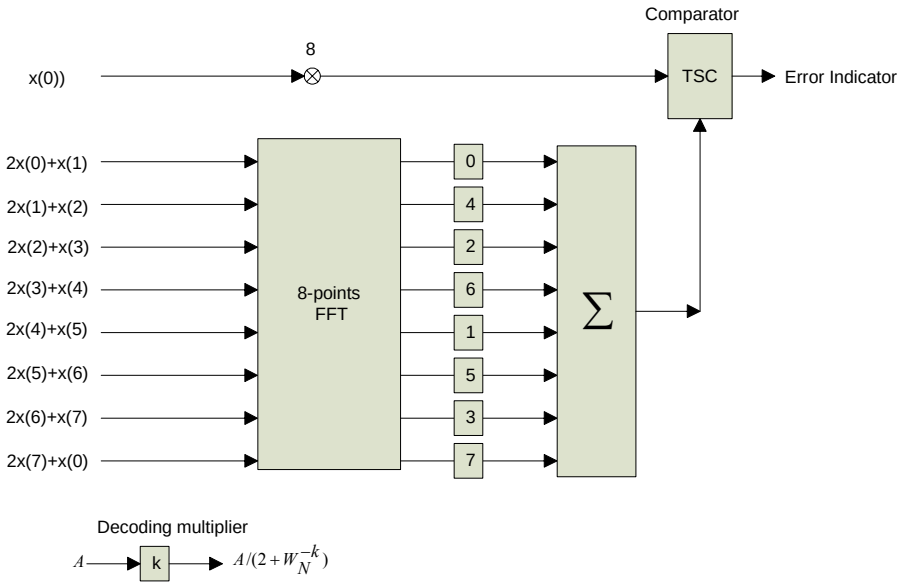

Figure 5. CED scheme proposed for FFT designs in [JoAb88]

Other ABFT fault tolerant FFT implementation was proposed in [WaJh94] by Wang et al., where the authors present a solution with a fault coverage near to 100% over the complete FFT computation process. They propose to use encoding and decoding schemes that weight all outputs that can be affected by the same error with different values, so that the final addition is erroneous in case of fault in any stage of the FFT computation. Due to the regularity and symmetry of the FFT computation, if one fault affects different final outputs, these erroneous signals have the same error magnitude but with alternate sign and their total addition could not detect some errors. Using the notation presented in the following expressions (for a single dimension FFT) equations (20) and (21) illustrate the proposed ABFT *Concurrent Error Detection* scheme by Wang et al.

$$\begin{bmatrix} X(0) \\ X(1) \\ \dots \\ X(N-1) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & \dots & W^{N-1} \\ \dots & \dots & \dots & \dots \\ W^0 & W^{(N-1)} & \dots & W^{(N-1)(N-1)} \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(1) \\ \dots \\ x(N-1) \end{bmatrix} \tag{19}$$

$$\begin{aligned} \vec{X}^T &= F_N * \vec{x}^T \\ \begin{bmatrix} \vec{X}^T \\ X(N) \end{bmatrix} &= \begin{bmatrix} F_N \\ \vec{w}_s \end{bmatrix} * \vec{x}^T \end{aligned} \tag{20}$$

$$\begin{aligned} &\vec{r}_w = (W_3^0, W_3^1, \dots, W_3^{N-1}) \quad where W_3^q = e^{-j\frac{2\pi q}{3}} \\ &\vec{w}_s = \vec{r}_w * F_N \quad X(N) = \vec{r}_w * \vec{X}^T \end{aligned} \tag{21}$$

The input encoder generates the input weighted checksum and the output decoder generates the output weighted checksum, both of them are equal in a fault free environment, so in a fault-free scenario equation (21) will be true. In the input side *N* complex multipliers are need for the encoding process and in the output side only two complex multipliers will be needed.

In [OhY094] and [OhY095] different CED schemes (including fault correction) are presented, based on ABFT solutions. They select distinct linear weight factors to minimize the possibility of masking when an error occurs. Notice, as it has been commented before, that the FFT circuit is regular and symmetrical. Therefore, an error in the DFT computation can most likely produce an output checksum different from an input checksum when each input data is assigned a different weight equally distributed between 0 and 1, while the computation occurs through the balanced symmetric FFT design. The authors assure that a high error coverage with low false alarm rate is obtained by applying the linear weight factors to the checksum.

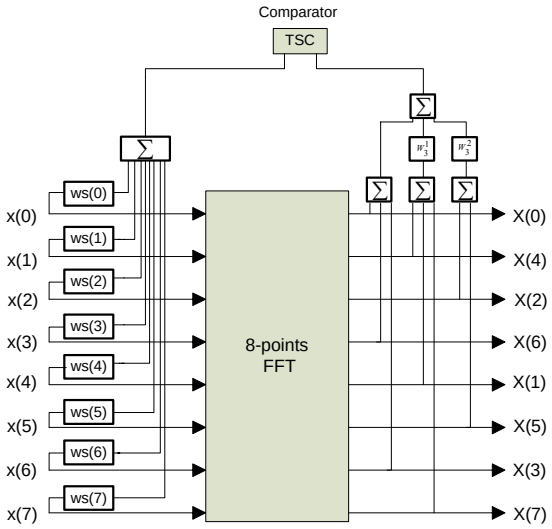

Figure 6.The weighted checksum CED scheme for an 8-point FFT [WaJh94]

Other ABFT systems have been presented in [TaHa93] and in [CiJG07], in this last reference the author propose a solution to detect and correct faults in software FFT implementations used in space applications, the authors present one of the experiments in the Space Technology 8 mission of NASA's New Milennium Program.

Another CED scheme that uses some of the properties of the computation algorithm implemented by the DFT, as Parseval's Theorem is proposed in [ReBa90], see equation (8), this kind of checksum, referred as SOS (Sum of Squares) is valid for both one-dimensional and two-dimensional FFTs. In this case, the ratio area overhead needed to implement this error detection scheme is *3N* real multipliers and at most *3N* real adders, where *N* is the number of input samples to the FFT circuit. The butterfly operation on complex numbers involves four multiplications and six additions of real numbers. Hence, the hardware overhead is $O(N/N \cdot log_2 N)$, or $O(1/log_2 N)$. The throughput of the system is not affected by the error checking hardware since the pipelining delay is limited by the complex butterfly operation rather than the real multiplications involved in the error checking hardware. The sum of squares of the elements may be large in magnitude and would require a word length larger than that used for FFT computation.

In [ChMa88] Choi et al. propose a method to achieve fault tolerance by introducing a redundant stage for an FFT design. In this proposal, a *Concurrent Error Detection* mechanism called *recomputing by alternate path* detects errors during normal operation. Because it is based on recalculation, this scheme requires a time overhead of 100% and hence it greatly reduces the system throughput.

6 PROPOSED FAULT TOLERANT FFT IMPLEMENTATIONS

In this section a number of fault tolerant FFT implementations to deal with SEUs and SETs based on the *System Knowledge* concept will be proposed and analyzed in order to compare them with the FTMR protected version of the same in place-DIT FFT design.

The fault tolerant implementations can be classified in two categories: those which are for a specific number of points N of the FFT, and those which are independent of N.

The protection techniques proposed here are presented for in place DIT FFT implementations (high-throughput) that means that the memory used for the input signals is reused to store the intermediate and final results of the FFT computation. Therefore, input, output data and intermediate FFT results are written in the same memory [OpSh75], [Lyon01]. However, these fault tolerant techniques can be extrapolated to other kinds of FFT implementations with minimal changes.

Consider one small (*N* reduced) in place FFT implementation where input data directly used by the FFT core is written one by cycle and the FFT computation is carried out using a pipelined mode operation. In this situation, if sum of the memory write operation time plus the FFT computation one is smaller than the minimal time interval between consecutive SEEs we can use the implementations referred as *Checksum-based Fault tolerant FFT* and *Checksum-based Fault tolerant FFT enhanced* commented in section 6.1.

For large FFTs (and in general for *N* points FFTs, *N* not limited to any constraint), different schemes based on a combination of space and time redundancy are presented in section 6.2.

It must be noted that we have assumed the module level fault model in this work. This model includes all possible functional errors that can occur in a butterfly, errors in the inputs, outputs, adders and multipliers. When a fault appears in a butterfly, the resulting error can be modelled as an additive error at one of the input or one or two (errors in multipliers) of the outputs ports of the module. [JoAb88].

6.1 Fault tolerant FFTs based on checksums

Taking into account the previous consideration related with the time interval between consecutive erroneous bit flips in the design. We can consider the possibility of performing (in parallel to the memory write operation) the calculation of some checksums that we will need to decide if the FFT results are correct. This mechanism will be used as correction and fault masking logic, using as CED mechanism a duplicate version of the FFT circuit.

These techniques are based on the *System Knowledge* concept because they use some properties/characteristics of the DFT computation to carry out the error mitigation. Also, they follow a similar protection mechanism to the used in the ABFT solutions commented on *Related Work* (section 5) in the sense that some checksums are computed to detect/locate any possible bit flip in the butterfly module (in multipliers, adders, registers). Therefore, as these SEE detection and correction mechanisms are based on the comparison of values that are calculated by distinct ways, the checksums on one hand and the normal FFT computation results on the other hand, these values will have different precision errors due to the quantization and round off or truncation and these differences should be taken into account to analyze the real fault coverage of these fault tolerant implementations.

There are several ways to accomplish the fault coverage analysis of these types of fault tolerant techniques, by means of analytical expressions or through exhaustive fault injection campaigns. In this document, an exhaustive set of fault injection campaigns has been performed on the proposed fault tolerant protection techniques based on checksums and the main results are exposed in section 8.1.

6.1.1 CHECKSUM- BASED FAULT TOLERANT FFT

The error mitigation technique presented in this section will be referred as *Checksum-based fault tolerant FFT.*

6.1.1.1 General Description

If we consider an in-place FFT with a reduced value of *N,* (in our case, memory write operation and FFT computation times must be lower than the time interval between consecutive SEEs or bit-flips to assure a good fault tolerant effectiveness) we can use the fault tolerant implementation shown in figure 7 for an 8-point FFT. In this figure three main modules are shown:

- Two FFT parallel cores are used to detect any possible bit flip on the FFT computation.
- A checksum module that represents the combinational and sequential logic used to compute some simple checksums from the input signals of the FFT that are needed to decide what FFT is the erroneous one and correct it, in case of SEE occurrence.
- A comparator that
  - Compares the outputs of the two FFT cores and
    - If no differences are found, the two FFT computations are assumed to be correct and the output of one of them is selected to be connected to the output interface of the design. Note that input data must be correctly protected to guarantee that indeed the output is correct. For this specific example, two in place parallel FFTs, the data processed by each FFT core are stored in different storage cells but the problem of how input data are written in these different storage structures correctly is assumed to be resolved previously.

- If differences are found, one error has occurred in one of the two FFT computations. In this situation, the checksums previously calculated are used to find which FFT core has not been impacted by the single event and connect its results to the output interface of the device.

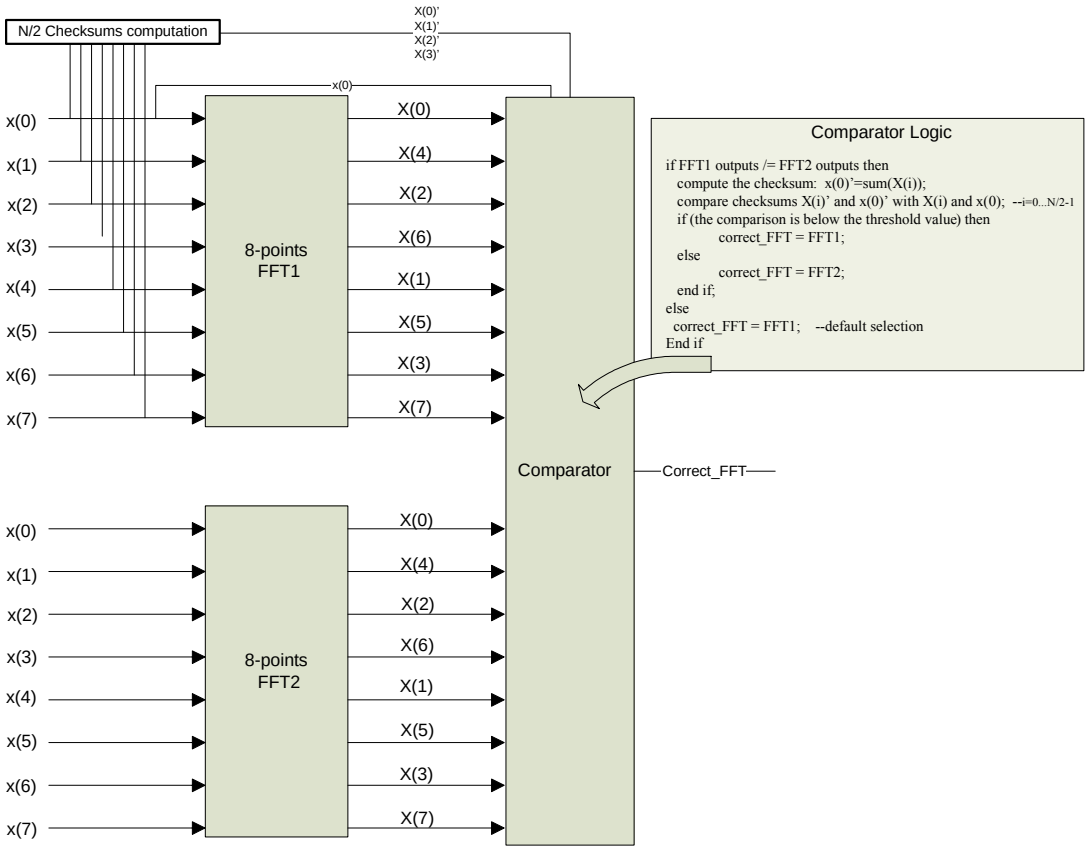

Figure 7. *Checksum- based Fault tolerant FFT* scheme for an 8-point FFT

The traditional CED mechanisms based on checksums (see section 5) to detect errors in FFTs, compute two kind of checksums from the FFT inputs and outputs and compare them to detect the maximal number of possible errors in any butterfly module. These checksums are computed using extra multipliers, adders and additional logic. The main difference with these CED schemes is that in our proposal we compute the minimal number of *Simple Concurrent Error Detection Checksums* to assure that the maximal number of possible errors in the butterfly model can be determined. Note that we consider a maximal number of possible errors detected because, as we have just commented before, even if a check is shown to detect all faults in the system in infinite precision, the

performance of any algorithm-based check is essentially limited by the round off/truncation errors, and due to finite-precision errors some hardware faults cannot be simply detected. Section 8.1 introduces a fault coverage analysis of the checksums based fault tolerant techniques by means of exhaustive fault injection campaigns.

As *Simple Concurrent Error Detection Checksum* we understand the computation of one specific DFT output, using any of the possible implementations, as direct calculation or Goertzel algorithm, for example.

From the same property that is used to compute the DFT by mean of the FFT (DIT), the symmetry of the twiddle factors (see equation (7)), we can infer that we need *N/2* different S*imple Checksums* to detect all possible errors in a FFT computation, with the exception of errors in one of the outputs on the last step of the FFT computation, to detect these errors an extra *Simple Checksum* will be needed (the computation of the value of the input sample *x[0],* in our case). Figure 8 shows the number of twiddle factors that we need to compute 64-point (left image) and 8-point (right image) FFTs, related to the minimum number of output signals of the FFT that we need to calculate as *Simple Concurrent Error Detection Checksums.* Concretely, the first *N/2* FFT outputs are calculated using the twiddle factors shown in figure 8 in the last stage of the FFT computation (see figure 1), and these outputs will be the ones that the *Simple Concurrent Error Detection Checksums* should compute to detect all possible errors on the FFT calculation (except in last stage).

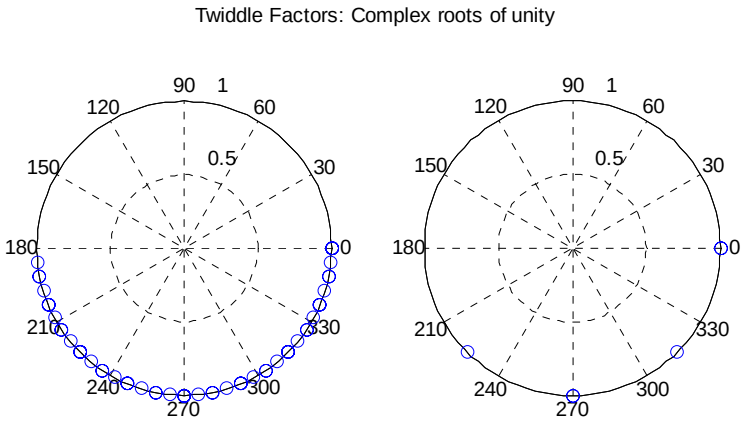

Figure 8. Twiddle factors needed to compute specific FFTs (64-point and 8-point FFTs)

The extra area overhead of the *Checksum-based fault tolerant FFT* consists of one extra FFT parallel core, and one checksum module that needs *N/2-2* complex multipliers (the twiddle factors corresponding to 0 and 270 degrees do not need multipliers) and *N/2* complex adders, some extra registers are also needed to store the value of the *Simple Checksums*. Besides extra logic is also needed to detect errors in the last stage of the FFT computation, this process is carried out by mean

of an extra S*imple Concurrent Error Detection Checksum,* the sum of the FFT outputs (equal to *x[0]* input). Equations (22) and (23) illustrate the *Simple Checksums* needed to detect all possible single errors in an *N*-point FFT, using direct calculation.

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{kn} \quad k = 0,1,2,\ldots N/2 - 1 \qquad (22)$$

$$x[0] = \frac{1}{N} \cdot \sum_{k=0}^{N-1} X[k] \qquad (23)$$

6.1.1.2 Conclusions and restrictions

Most of the ABFT solutions presented in *Related Work* (section 5) do not take into account the possibility of errors in the checksum calculation used to detect or locate the error. In this technique the checksums are not protected against SEEs either because with the assumption that only one SEE (SEU or SET) can occur during the time needed to write the memory and compute the FFT (for N-point FFTs with N low). In this scenario, if the error occurs in one of the checksums, the outputs of the two FFT cores will be identical and the checksums will not be used to decide which is the correct FFT.

To enhance the fault coverage of this technique, some extra logic can be included into the comparator so that, when differences are found between the outputs of the two FFT cores, the nearest value to the checksums is selected as the correct one, in place of using one threshold to compare the checksum values with the outputs of one of the FFT cores. However the fault coverage of this last solution will not be 100% in real applications limited by round off errors, as it has been tested using the FTU hardware fault injection emulator.

Although several cases of multiple errors can be detected and corrected using this protection technique (as for example several errors on the computation process of one of the two FFT cores) it deals only with SEEs that cause single bit-flips. The protection against MBUs (or SEEs that generate MBUs) or consecutive errors in a time interval lower than the specified before (time needed to write the data and compute the FFT) is not considered. A similar situation applies to the FTMR protection, but in this last case, one error can be tolerated each clock cycle.

Extrapolation of this fault tolerant technique to any type of other FFT implementation must analyze some characteristics that are implementation dependent as for example, high or low throughput FFT, if it is in place or not, if it uses intermediate buffers or FIFOs to compensate different operation speeds…

However, the extra area overhead of this error mitigation technique based on checksums might be lower than the area cost increment of the FFT protected version using FTMR (see section 8.2) in most of the high-throughput implementations, but in any case, area overhead is

architecture/implementation dependent and it should be analyzed for each specific case. One possible exception for the error mitigation technique proposed here could be, for example, a low-throughput FFT implementation with one single butterfly 'running' across the memory with all *N* data points. In this situation, it might be less overhead to triple the butterfly instead of doubling it and adding the checksum calculation (but it may be found a checksum based mechanism adjusted for this type of implementations).

Therefore, we could try to reduce the area cost of this protection method by means of using the solution proposed in section 6.1.2.

6.1.2 CHECKSUM- BASED FAULT TOLERANT FFT ENHANCED

6.1.2.1 General Description

From figure 8 we can note that the complex twiddle factors are equally distributed over the unity circle, therefore, we could think about reducing the number of multipliers combining the *N/2* checksums that we need to detect all internal possible faults in the FFT computation two by two. In order to decrease the number of multipliers (and the number of registers needed to store the checksums), these combinations must be made between those FFT outputs which basic twiddle factors have a certain relation.

The following equations show the new proposed scenario:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{kn} \quad k = 0,1,2,\ldots {}^{N}\!/_{2} - 1 \tag{24}$$

$$X[0] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{o} = \sum_{n=0}^{N-1} x[n] \tag{25}$$

$$X[1] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{1n} \tag{26}$$

$$X[2] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{2n} \tag{27}$$

$$\ldots$$

$$X\left[\frac{N}{2}-1\right] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{({}^{N}\!/_{2})n-n} = \sum_{n=0}^{N-1} x[n] \cdot W_N^{({}^{N}\!/_{2})n} \cdot W_N^{-n} = \sum_{n=0}^{N-1} x[n] \cdot (-1)^n \cdot (W_N^n)^* \tag{28}$$

Therefore, we could combine the next expressions

$$X[0] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{o} = \sum_{n=0}^{N-1} x[n] \tag{29}$$

$$X\left[\frac{N}{4}\right] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{(N/4)n} = \sum_{n=0}^{N-1} (-j)^n \cdot x[n] \qquad (30)$$

$$X[k] + X\left[\frac{N}{2} - k\right] = \sum_{n=0}^{N-1} x[n] \cdot (W_N^{kn}) + \sum_{n=0}^{N-1} x[n] \cdot (-1)^n \cdot (W_N^{kn})^* \quad k = 1,2,..., \frac{N}{4} - 1 \qquad (31)$$

If $W_N^{kn} = e^{-j\frac{2\pi}{N}kn} = \cos\left(\frac{2\pi}{N}kn\right) - jsen\left(\frac{2\pi}{N}kn\right)$, then,

$$\begin{aligned} X[k] + X\left[\frac{N}{2} - k\right] = & \sum_{n=0}^{N-1} x[n] \cdot \cos\left(\frac{2\pi}{N}kn\right) - j \cdot x[n] \cdot sen\left(\frac{2\pi}{N}kn\right) + \\ & \sum_{n=0}^{N-1} (-1)^n \cdot x[n] \cdot \cos\left(\frac{2\pi}{N}kn\right) + j \cdot (-1)^n \cdot x[n] \cdot sen\left(\frac{2\pi}{N}kn\right) \quad k = 1,2,..., \frac{N}{4} - 1 \end{aligned} \qquad (32)$$

$$\begin{aligned} & X[k] + X\left[\frac{N}{2} - k\right] = \sum_{n=0}^{N-1} x[n] \cdot \cos\left(\frac{2\pi}{N}kn\right) \cdot (1 + (-1)^n) + j \cdot x[n] \cdot sen\left(\frac{2\pi}{N}kn\right) \cdot ((-1)^n - 1) \\ & k = 1,2,..., \frac{N}{4} - 1 \end{aligned} \qquad (33)$$

The benefit of these combinations is that the number of multipliers is reduced to *N/4-2* and the number of extra registers used to store the *simple checksum values* or their combinations is also reduced to *N/4* registers of a certain length (with a similar precision to the input samples).

But, when using this fault detection scheme based on *simple checksum combinations*, not all errors in the butterfly module can be detected. For example, consider the case of an 8-point FFT and one error in the multiplier of the second butterfly in the second stage of the computation (see figure 1). In this situation, the error will be propagated to the FFT outputs *X [1], X [3], X [5]* and *X [7]*, with alternate sign, that means that

$$\begin{aligned} X[1]' &= X[1] + e \\ X[3]' &= X[3] - e \\ X[5]' &= X[5] + e \\ X[7]' &= X[7] - e \end{aligned} \qquad (34)$$

Where *X[i]'* represents the erroneous output and *X[i]* is the non-erroneous value of the *i-th* output, being *e* the error magnitude.

In this example, the simple checksums or/and combinations will be *X [0], X [1] + X [3]* and *X[2]* (see equations 29, 30 and 31). Therefore, in the final comparison to detect errors the addition of the FFT outputs *X [1]'+ X [3]'* will be equal to the previously calculated checksum *X [1] + X [3]*, so

the error will not be covered with these type of checksum combinations. This problem could be resolved by weighting the checksum addition with different values a*X[k]+b*X[N/2-k]*, to continue using the advantage of the multipliers reduction, we can consider the basic combination of *X[k]+2*X[N/2-k]* (*a=1* and *b=2*) that may be implemented without multipliers by means of shifting operations. With this last modification, equation (31) will be changed by equation (35)[1].

$$X[k]+2\cdot X\left[\frac{N}{2}-k\right]=\sum_{n=0}^{N-1}x[n]\cdot(W_N^{kn})+2\cdot\sum_{n=0}^{N-1}x[n]\cdot(-1)^n\cdot(W_N^{kn})^* \qquad k=1,2,...,\frac{N}{4}-1 \quad (35)$$

6.1.2.2 Conclusions and restrictions

The ratio computational overhead (in comparison with the non protected FFT) of the commented implementations will be 1 and 1/2 for the *Checksum Based Fault Tolerant FFT technique* and for the *enhancement* using the checksum combinations, respectively (See area cost results in section 8.2. for the specified case study 7). Also, the extra logic for comparisons, registers and adders is less than the corresponding for the FTMR protection. However, when using the proposed solutions to accomplish fault detection and correction (adding a duplicate version of the FFT computation module) they have a very high extra area overhead although it may be still lower than for FTMR protection for high-throughput implementations (but it is implementation dependent in a general sense, and this should be analyzed for each specific design).

The proposed implementations assume also a minimal time interval between consecutive SEE to assure a complete fault coverage, if for example one bit-flip occurs in one of the checksum accumulators and then another SEE flips the value in one line of the FFT network the system will not be able to decide correctly what is the FFT free of faults, in this sense we could try to protect the checksums using some kind of parity (Hamming codes) or triplication (TMR) but as they are proposed for small in place FFTs (reduced time to write the memory and compute the FFT, approximately equivalent to the minimal time interval between consecutive SEEs), this considerations has not been taken into account.

It should be noted that the maximal frequency operation of these error mitigation techniques, for implementations whose outputs are available at the same cycle, is lower than the obtained for other fault tolerant methods (as FTMR) because of the *Simple Checksum* used to compute the addition of all FFT outputs at the end of the computation.

[1] This fault tolerant implementation and the previous one (exposed in section 6.1.1) has been modelled and tested in matlab for different random input streams, non-random inputs and considering all possible butterfly errors for infinite precision and they two have full error coverage.

If an extrapolation of these solutions to FFTs with large *N* is analyzed, the protection (for example using TMR and Hamming) of the memory used to store the input data and the checksums has to be considered. Besides, the word length of input, output and checksum data should be increased to maintain the same precision on the FFT results. In this situation, the techniques will allow one single error each FFT computation, that means that the minimal time interval between consecutive faults could be reduce to the time needed to perform the FFT computation (approximately), addressing, of course, a higher extra area cost. This technique (for large FFTs) has not been implemented because it seems to have not any advantage with respect to the protected implementation by means of FTMR.

6.2 Fault tolerant FFTs combining time and space redundancy

One fault tolerant technique is proposed in this section for situations in which some extra delay can be tolerated between the SEE (SEU or SET) occurrence and its correction. Therefore, the error latency in these FFTs fault tolerant implementations will be higher that the one obtained for the FFT protected version with FTMR. However, the number of points of the FFT (the value of *N*) under protection is not restricted to any value in particular.

Using the benefit of TMR protection for the memory that contains the input data, we can use one of the triplicate memories as back up of the previous (error free) stage during the pipelined FFT computation. Once a fault is detected, the previously stored stage is restored and the last computation is retried. This fault tolerant protection mechanism has some error latency cycles and low extra area cost if we assume that the memory used for the input data is TMR protected. It will be known as *Fault Tolerant FFT based on Recalculation.*

6.2.1 FAULT TOLERANT FFT BASED ON RECALCULATION

6.2.1.1 General Description

If the memory used to store the input data is TMR protected and the FFT implementation is an in place one (similar to the previous FFT designs protected), a fault tolerant FFT implementation that combines space and time redundancy can be performed as follows:

- During the memory write/read operations the data are TMR protected.
- During the FFT computation (one butterfly stage that executes $log_2(N)$ times):
  - Two parallel FFT cores are used to detect all possible faults in the computation process.
  - Two of the TMR protected memories store the intermediate and final results during the FFT computation.
  - The third TMR protected memory is used to store the results of the last error free FFT computation step (this module is used as back up memory).

Therefore, as it is an in place FFT, where the memory used to store input data is also reused for the intermediate and output results, two of the triplicate memory modules used to protect the initial data (inputs for the FFT circuit) should be used to store the computation results of the two parallel FFTs, and the third memory module could save the results of the last non erroneous stage of the FFT computation. In this scenario, in case of SEEs occurrence over one specific FFT data path and computation step the restoration of the error free results of the previous stage (that are stored in the module used as backup memory) can be made and the recalculation of the last erroneous computation can be launched then. At the end of the FFT processing, the TMR protection for the final output results can be carried out again.

Figure 9 illustrates a simple diagram and a brief logic description of the proposed solution. As it can be seen in the figure the main modules of the proposed protection technique referred to as *Fault Tolerant FFT based on recalculation* are:

- Two FFT cores that perform the same computation in parallel in order to detect any fault during the FFT calculation
- Input/Output data Memory with TMR protection
- C&R (Comparison and Retry) module compares the FFT outputs at the end of each pipelined step of the computation. In case of error it launches the restoration of the results of previous step and retries the last computation.

This technique will have an error latency that depends on the parallelism of the structure but the higher the parallelism is the lower the error latency of the proposed implementation is, in case of faults. The minimal time interval between consecutive SEEs (SEUs and SETs) to assure a correct operation of this technique is 1 cycle. Some MBU patterns can be tolerated too, concretely, all MBUs that affect the results of the two FFT computations in a different way will be detected during the comparison process (at the end of each FFT step). Different outputs will trigger launch the recalculation process (in this faulty scenario the memory used to store the previous results should be error free). MBUs only in the memory used as back up memory can be tolerated too, because in this case the outputs of the two FFT computations should be identical. Furthermore, if we assume that TMR protection is accomplished for the input data memory, the hardware overhead for this protection technique includes a replication of the data path of the FFT and a comparator that performs the C&R logic shown in figure 9 and logic to copy back up data in case of error. (See fault coverage and area cost results in section 8).

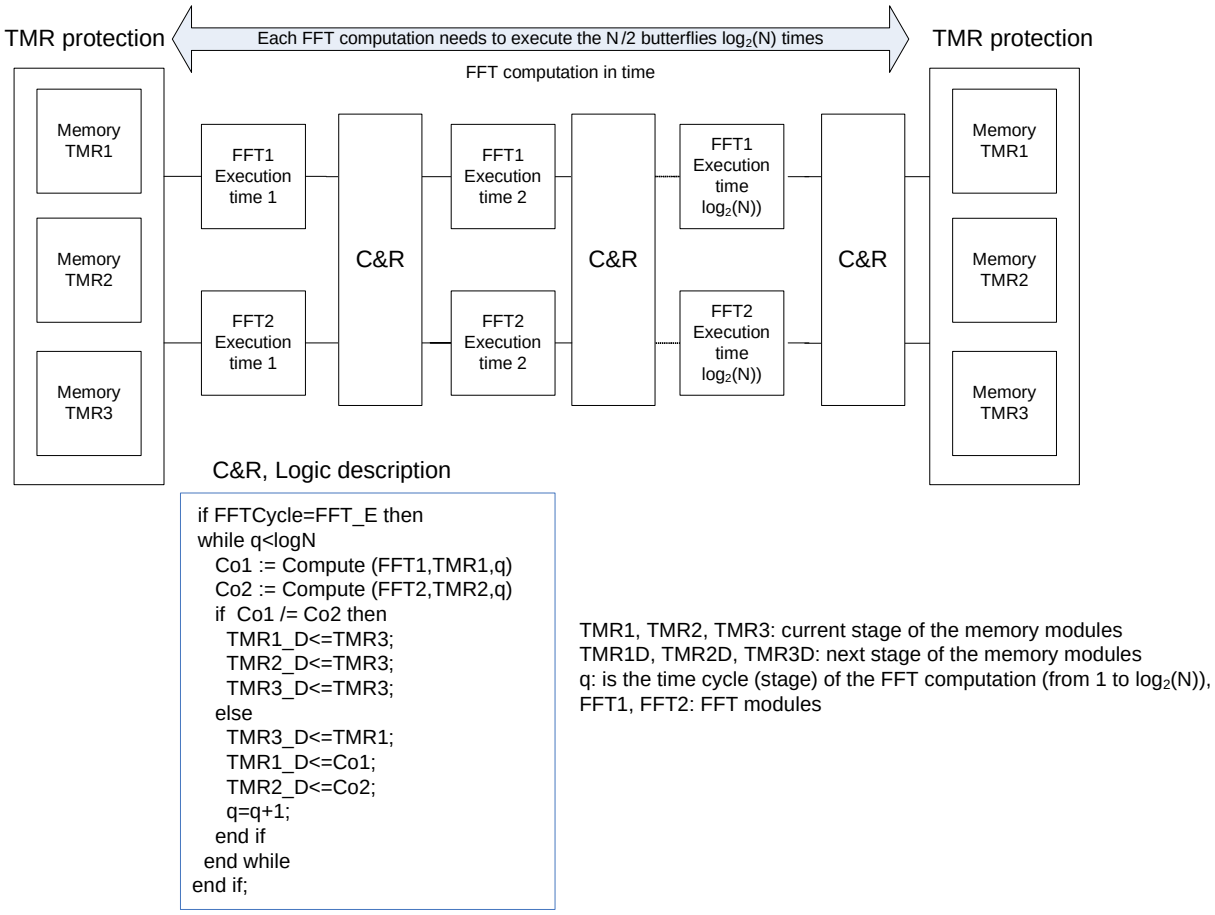

Figure 9. Fault tolerant FFT based on back up memory

7 CASE STUDY

The protection techniques proposed have been implemented to protect an 8-point FFT circuit that was used for several radiation test analysis performed by Saab Ericsson Space under different ESA contracts. The related results can be found in reports [Stur04] and [Stur06].

7.1 Original FFT design

The FFT was chosen to achieve high proportion of both combinatorial and sequential logic. The flow for the FFT-module (see Figure 10) is described in the next points [Stur06]:

1. The in data shift register is loaded from DataIn by DataInEn inputs.
2. The Randomizer generates randomized data from the In Data Shift Register and stores it in Data Matrix.
3. A Fourier Transform is performed on the Data Matrix. The result is rewritten to the Data Matrix.
4. The sum of each result from the Fourier transform is added to the Sum of Fourier Register.
5. Step 2 and 4 is repeated until FFT has been performed on 1024 samples.
6. When all Fourier transforms have been performed, the Sum of Fourier Register is shifted out on DataOut output with DataOutEn asserted.

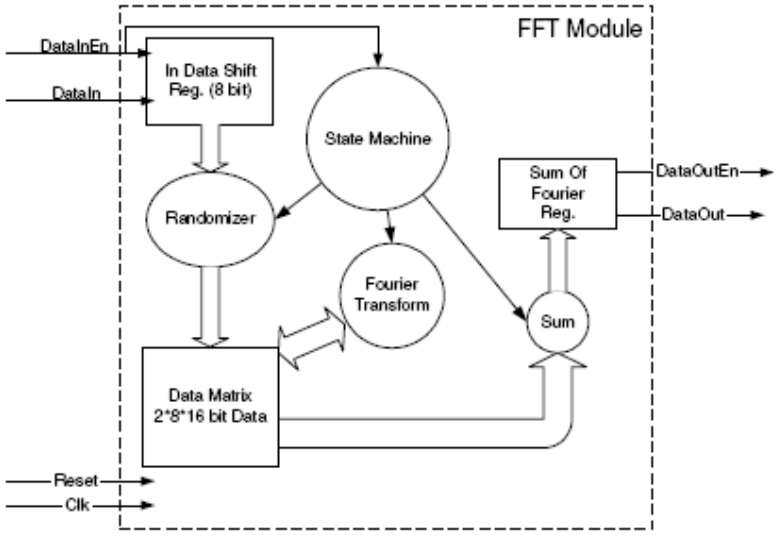

Figure 10. Dataflow of the original FFT design [Stur06]

7.2 Modified FFT design to be protected with the fault tolerant techniques

The original FFT circuit was modified to control the input stimuli generation (using matlab) for testing purposes and to record the output data in files for further analysis in matlab. The input and output data (not their sum) are written/read one by cycle during the memory write/read corresponding states. Besides this, some little modifications were performed to the butterfly modules to obtain the correct FFT outputs (see figure 11).

The data flow for the modified FFT circuit is:

1. The in data is loaded in the Data Matrix from DataIn by DataInEn inputs (from an Input data file generated in Matlab). The FFT stage associated with this operation will be referred to as *Gen_E* stage and it lasts 8 clock cycles, because one complex input value is written each cycle.
2. An 8-point in place Fast Fourier Transform is performed on the Data Matrix. The results are rewritten to the Data Matrix. This corresponds to the *Fft_E* stage that comprises 3 clock cycles.
3. The complex output results are written one by one each clock cycle and they are written in the Output Data file for further analysis in Matlab. This stage is the *Sum_E* one and it lasts 8 clock cycles.
4. Steps 1 to 3 are repeated until the FFT has been carried out on any specified number of samples (its maximal value is 32756), depending on the fault injection test performed.

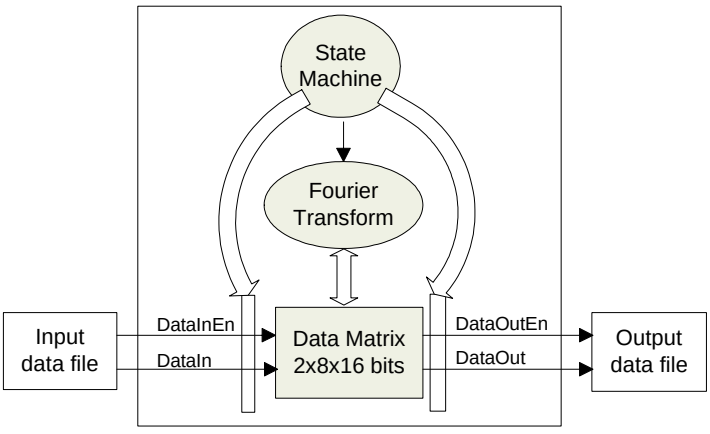

Figure 11. Dataflow of the modified FFT design

Other information related with the FFT data path is the following:

-
- Complex input signals are represented using fixed point arithmetic.
- A sign magnitude representation of 16 bits is used for of real and imaginary parts.
- As it is an 8 point FFT (a small FFT), the bit reverse function and the twiddle factors generation are accomplished by mean of LUTs.
- Several counters and a simple FSM are used to control the FFT computation.

8 RESULTS. FAULT COVERAGE AND EXTRA AREA COST

Results in terms of fault coverage and extra area cost of the fault tolerant implementations presented in section 6 and applied to the FFT introduced in *Case Study* (section 7) are illustrated in this section. These results are compared with the FFT protected version using FTMR to deal with SEUs and SETs.

8.1 Fault coverage

The accuracy of digital systems is limited because only a finite number of bits are available. As a result, round-off errors are unavoidable and care must be taken to prevent the overflow [OpSh75]. Due to the finite precision, when the same value is computed by different paths (as it is the case of the *checksum-base fault tolerant FFT* implementations proposed in sections 6.1.1 and 6.1.2) with different precision errors, the values obtained by these two paths may not be equal to each other, even if the FFT computation is free of any functional error caused by radiation effects, such as SEEs. One approach to solve such a problem is to allow a small difference (threshold),η, between the values compared. But there is a trade-off for selecting η. A small η will increase the false alarms in the system, which means, that the result of the comparison will be that one error has occurred when the system is error free, whereas a large η will reduce the fault coverage and errors in the FFT computation will be masked by the round-off ones. In this last situation, the error mitigation mechanism may not detect that an error has occurred in the system. For the checksum based fault tolerant FFT implementations presented in this work, false alarms and fault masking will be a problem because the use of the checksums is only triggered when faults are detected in one of the two FFT cores. In this situation, if the checksum comparison with one of the two FFTs outputs has a false alarm response or the fault is not covered the outputs of the protected system will be erroneous, as it will be analyzed in section 8.1.1. This will be the main problem of fault coverage of the checksum based proposed solutions.

For the reason commented, the study of how round off errors modify the theoretical fault coverage of one specific error mitigation technique based on checksums must be carried out by means of analytical error models or exhaustive fault injections. In this work, to prove the fault coverage of the different fault tolerant techniques, a number of fault injection campaigns and tools have been performed and used. A hardware fault injection emulator based on FPGAs, the FT-Unshades (FTU) v1.0 developed by the University of Sevilla (AICIA-GTE) under an ESA contract,

has been used for the exhaustive fault injection campaigns performed to the *Checksum based FT-FFT* implementations (section 6.1). This fault injector provides a test and analysis framework to check the design protections (TMRs and voters, EDACs, …) of an IP or ASIC, before the place and route and fabrication processes [AgBM04], [AgTo04]. Furthermore, a software fault injection has been used based on ModelSim simulations (using scripts derived from the SST tool [Gonz04], [RMRR07]) to carry out the fault tolerant analysis of the error mitigation techniques based on space and time redundancy presented in section 6.2.

8.1.1 Fault coverage of Checksum Based Fault Tolerant FFT techniques

As a consequence of finite precision, that introduces different round-off errors depending on the computation process used to obtain any specific value, some soft errors or SEEs will remain undetected. Therefore, we need to perform an exhaustive analysis of the system operation and its fault coverage in real environments, (real input signals, operation conditions, threshold values…). To do this, a number of fault injection campaigns have been performed on the two proposed error mitigation techniques based on checksums to determine their effectiveness against SEEs. In our techniques, as the *Simple Checksums* calculated are compared with the results of one of the two FFT cores only when faults are detected between the final results of the two parallel FFT cores (what means that a single event caused by radiation has impacted the device and generated one bit-flip), several cases could occur:

- If the comparison is performed between the *Simple Checksums* and the erroneous FFT results, several scenarios can occur:
  - The comparison detects that the FFT under study is erroneous and decides that the correct FFT is the other one, connecting its results to the output interface of the devices. In this scenario, the correct operation of the device is assured (*case A*).
  - No faults are found between the *Simple Checksums* and the erroneous FFT results because the magnitude of the difference between the checksums and the output is below the threshold value that takes into account the round off errors. In this case, as no faults are found, the outputs of the erroneous FFT will be connected to the output of the device. Therefore the system results will not be correct. This scenario corresponds to the one when a fault is not detected or covered (*case B*).

- If the comparison is performed between the *Simple Checksums* and the FFT results without errors:
  - If the comparison detects that the FFT results are erroneous, a false alarm is launched and the results of the other FFT core (the erroneous one) will be connected to the output interface of the device. The system outputs will be erroneous. In this situation, a false alarm (because the difference between the correct FFT outputs and the checksums is higher than the threshold, η) is the cause of the erroneous operation (*case C*).
  - If the comparison decides that the FFT results are correct, they will be connected to the output of the device, therefore the system will operate correctly (*case D*).

Therefore, there are two different scenarios of possible errors in the operation of the *Checksum based Fault Tolerant* implementations, errors not covered and false alarms (*cases B and C*).

To prove the analysis above, the fault coverage of each *Checksum Based Fault Tolerant* technique for in place DIT FFTs (presented in section 6.1 for the case study presented in section 7) has been tested by mean of using the FTU fault injector, using different input sequences of distinct lengths (at most 32768-sample lengths) and fault injection campaigns on any register of the design in all states of the FSM used to perform the FFT computation.

The different states controlled by the FSM are the following:

- *Idle_E*: default stage.
- *Gen_E*: In this state the input data are stored into the Data Matrix (see figure 11).
- *Fft_E:* computation of the FFT on the Data Matrix and save the outputs into Data Matrix.
- *Sum_E*: The fault tolerant system outputs the FFT results cycle by cycle.
- *Cnt32768_E:* To check at the end of each FFT computation if it was the last computation.

FAULT INJECTION ANALYSIS: FTU

Similar results from fault injection campaigns using the FTU tool have been obtained for the two techniques tested. Therefore, the results commented here are applicable to the following techniques:

- *Checksum based fault tolerant FFT*
- *Checksum based fault tolerant FFT enhanced (multipliers reduction)*

- Fault injection characteristics:
    - 10 different input signals of 32768 samples (input signals of random and Fourier series samples).
    - 1 SEE is injected during each simulation.
    - Each campaign is repeated until all interesting possibilities are explored. Therefore thousands of fault injection campaigns are performed to carry out each experiment.

Fault injection results are summarizes in the next points.

1. Bit-flips on *Idle_E* and *Cnt32768* states

Fault injection in all registers of data and control paths has been performed during Idle and *Cnt32768* Count1024 states

- Bit flips on data path registers: single errors on Data Matrix, Simple Checksums, data out and enable registers do not cause faults in the output of the design.

- Bit flips on FSM registers (control path): counters and registers of the simple FSM that controls the normal operation of the FFT processor are TMR protected and their correct operation has been validated by exhaustive fault injection on all possible registers.

2. Bit-flips in *Gen_E*, *Fft_E* and *Sum_E states*.

These states are the ones that control the FFT process and computation, writing the input data into the Data Matrix, processing the FFT and connecting the correct results to the output interface of the device.

- Bit flips on FSM registers (control path): Exhaustive fault injection campaigns in all registers (TMR protected) used to control the system operation (during the states *Gen_E, Fft_E* and *Sum_E*) have been performed. No faults are found in the outputs of the fault tolerant implementations studied.

- Bit flips on Data Path registers:

Errors in adders, multipliers and memory cells can be modelled as single, double, quadruple… faults in Data Matrix depending on the case and state of the FFT computation where the single event generates the bit flip.

Results of exhaustive fault injection campaigns in all data path registers and states commented are exposed in the following points:

- During the FFT computation (*Fft_E* state) single events on adders, multipliers and registers have been analyzed, (note that single events in one butterfly multiplier can be modelled as double errors at the two outputs of the butterfly module). Depending on the FFT computation step where the SEE occurs, one or several outputs of one of the two FFT cores (whose outputs are compared) will be erroneous. In this situation, at the end of the FFT computation, when the comparison is performed, the stored *Simple Checksums* are used to decide which FFT is the correct one. In this scenario, and considering bit flips during the *Fft_E* states several situations have been observed.

  - Single bit flips on the *Simple Checksums* do not cause any fault in the output of the device. The system operation is correct.

  - Single bit flips on Data Matrix registers (remember that there are two Data Matrix registers in the system, referred to as Data Matrix1 for the first FFT and Data Matrix2 for the second one). Suppose that the first FFT (Data Matrix 1) is the one compared with the *Simple Checksums*

- Bit flips on MSB registers on Data Matrix1 (from register 15 to register 5). The system operation is correct, and no faults are found in the output of the device (similar to *case A*).
- Bit flips on LSB registers on Data Matrix1 (from 4 to 0), some of them are not detected during the comparison process with the Simple Checksums (due to round off errors) and the fault is not covered. As a result of this situation, the outputs of the system are erroneous, but the error magnitude observed in the output of the device is very small (see figures 12, 13 and 14, similar to *case B*).
- Bit flips on any register on Data Matrix2. The system operation is correct, and no faults are found in the output of the device. This means that no false alarms (*case C* is not observed) are found in the operation of the device because the threshold value selected is large enough to not trigger this type of situations (the system behaviour in this scenario is similar to *Case D*).

o If the bit flip is injected when the FFT results are connected to the output interface, *Sum_E* state, the following scenarios have been arisen:

  - Single bit flips on the Simple Checksums do not cause any fault in the output of the device. The system operation is correct.

  - The SEE occurs in one of the Data Matrix1 or Data Matrix2 registers that has just been read (to the output interface). In this situation, the system will have a correct and normal operation.

  - The bit-flip is injected in one register (Data Matrix1 or Data Matrix2) that has not been read yet. This situation is similar to the commented before; in this case the error mitigation technique will locate the correct FFT in base on the checksums values and the threshold. Therefore it can occur that some small errors will not be detected and the outputs of the device have some small level of error (*cases A, B* and *D* observed for similar conditions to the shown in state *Fft_E)*.

o If the bit flip is injected when the input data are written on Data Matrix1 and Data Matrix2, that means during state *Gen_E*, similar scenarios to the shown for the state *Sum_E* can be arose, depending on if the SEE affect to registers that have been or not written with the new input data. Besides, single bit flips on the Simple Checksums do not cause any fault in the output of the device.

Table 1 summarizes the main conclusions of the fault injection results obtained assuming only single events.

SEEs location	Error propagation analysis
DataMatrix1 or DataMatrix2 (bit-flips on Data Matrix registers can model faults in multipliers, adders or storage cells)	• Errors during the writing process of the input data *(Gen_E* state) • Errors during the FFT computation (*Fft_E* state) • Errors during the read process of the FFT results (*Sum_E* state) In all of these cases the fault coverage and round off errors must be studied because the *Simple Checksums* (and the threshold used to support the different round-off errors of the distinct computation paths) are used to locate the correct FFT, *cases A,B* and *D* are found. *Case B* (associated with a fault not detected, the output of the system is erroneous) is only observed when injecting faults on the LSB bits of Data Matrix, situations in which the error magnitude observed in the system outputs is near to the error precision of the FFT computation.
Simple Checksums	Errors do not cause any fault in the output. In this situation the values of DataMatrix1 and DataMatrix2 (outputs of the two FFT parallel cores) will be identical and one of then will be selected as correct output.
Errors on the FSM registers	Errors on the FSM registers do not affect to the computation because all of them are protected using TMR in all proposed error mitigation techniques solutions

Table 1. SEEs location and brief analysis of their propagation and or correction on the checksum based fault tolerant techniques of section 6.1

Considering that all possible errors are equally likely and weighting them with the function that takes into account if each possible bit flip affects or not to the output (considering the state of the FSM when it occurs, if it occurs before or after writing/reading the input/output data, …), the fault coverage of the system is very close to 100%. The error magnitude observed in the outputs of the *Checksum Based Fault Tolerant FFTs* (in case the fault is masked by the checksums, i.e., only when the fault impacts on LSB bits of DataMatrix1 in the suitable states) is near to the error precision of the FFT computation itself. That means that in case of errors in the output, the difference between the correct output and the erroneous one is very small, for the case study presented the maximal value of this difference is around 2^{-13}. Figures 12, 13 and 14 show the error magnitude observed in the output of the fault tolerant systems analyzed when a fault on LSB bits of Data Matrix 1 is not detected (note that bit flips on equivalent bits of Data Matrix 2 do not cause faults in the output of the device).

Figure 12 illustrates the case of one SEU in bit 4 (bit 0 corresponds to LSB bit) on the imaginary part of the fifth register of Data Matrix 1 during the writing process of input data into DataMatrix1 (*gen_E* state) in clock cycle 15. As it can be seen, and due to the round off errors the error magnitude of the outputs of the system with respect to the correct ones (fault free results) is around 2^{-14}, similar to the round off error that affects to the normal FFT computation. Of course, this type of errors will be propagated to the FFT outputs only in the cases when the register affected by the SEU has already been written with the new input data.

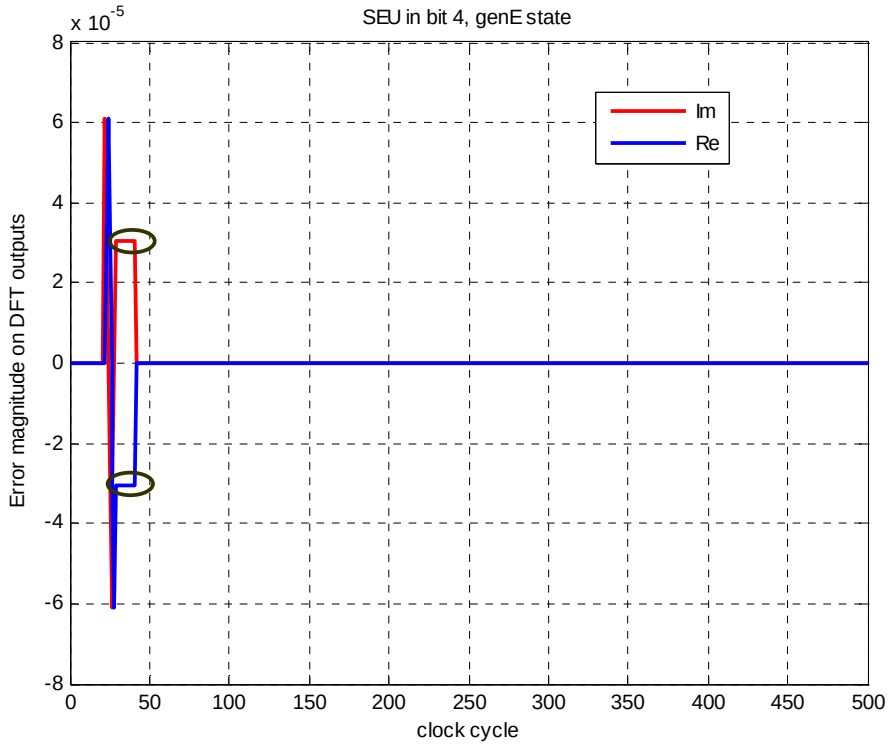

Figure 12. Error magnitude in the output of the *Checksum Based Fault Tolerant FFT Enhanced* when it is masked by the threshold used to address different round off errors (stage G*enE*).

When one FFT computation finishes and its results are output, the last useful value is maintained in the output of the device until the following FFT results are obtained, this is the reason why the marked errors (rounded in Figure 13) are kept until clock cycle 41. This can be eliminated by means of a reset operation of the output each time one complete FFT computation is finished (with low extra area overhead).

A number of errors on LSB bits of Data Matrix1 during *FFT_E* state can not be detected either. Figure 13 illustrates the outputs of the *Checksum Based Fault Tolerant FFT Enhanced* technique when one SEU is not detected and it is propagated to the output interface of the device. In this case, the SEU is injected on bit 2 of the imaginary part of the third Data Matrix1 register (cycle 38). As it can be seen in the figure, the error magnitude is also similar to the round off error estimated in the normal FFT computation, around 2^{-15}.

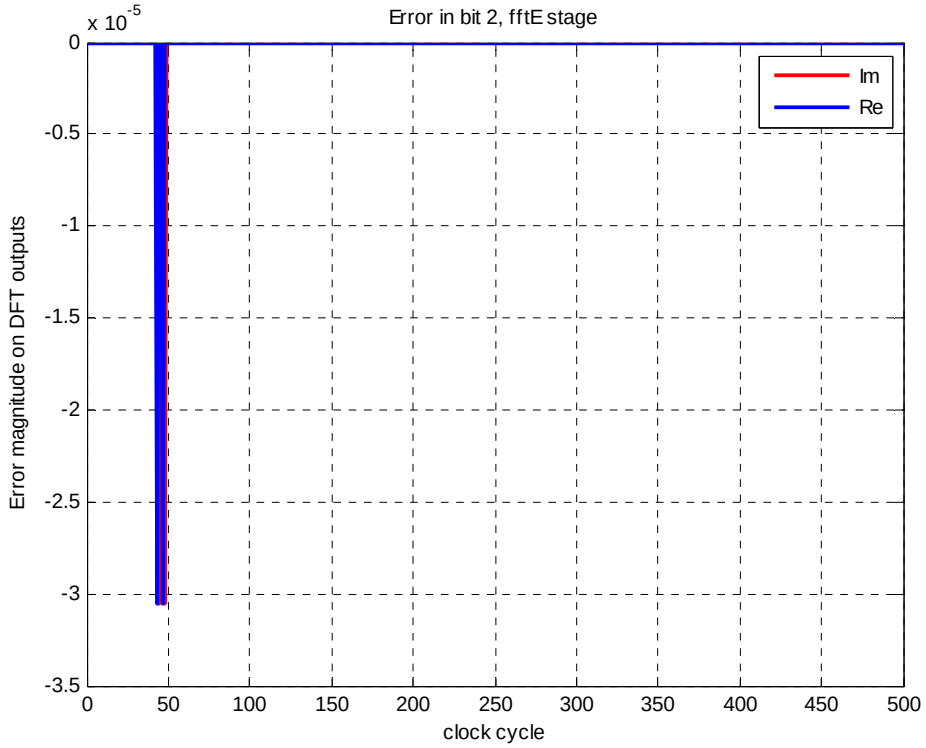

Figure 13. Error magnitude in the output of the *Checksum Based Fault Tolerant FFT Enhanced* when it is masked by the threshold used to address different round off errors (stage *Fft_E*).

Similar results can be observed when the SEE occurs in the *Sum_E* state, during the time interval used to read the FFT results to the output interface. Figure 14 illustrates the error magnitude in the output of the device (because the bit-flip has not been correctly detected by the detection and correction logic based on checksums) when the single event flips bit 2 on the imaginary part of the first register of Data Matrix1.

All results shown correspond to fault injection on imaginary parts on LSB Data Matrix1 registers of the *Checksum-based fault tolerant FFT enhanced technique* (presented in section 6.1.2) but similar solutions have been obtained when faults are injected on LSB bits of real parts. Equivalent results to those in the figures 12, 13 and 14 have been obtained for the *Checksum-based fault tolerant FFT* technique (illustrated in section 6.1.1).

It must be noted that when the SEE affects bit positions higher than 4 (bit 0 LSB bit) the error is always detected and corrected by the fault tolerant logic based on checksums. No false alarms have been found in any of the fault injection campaigns performed.

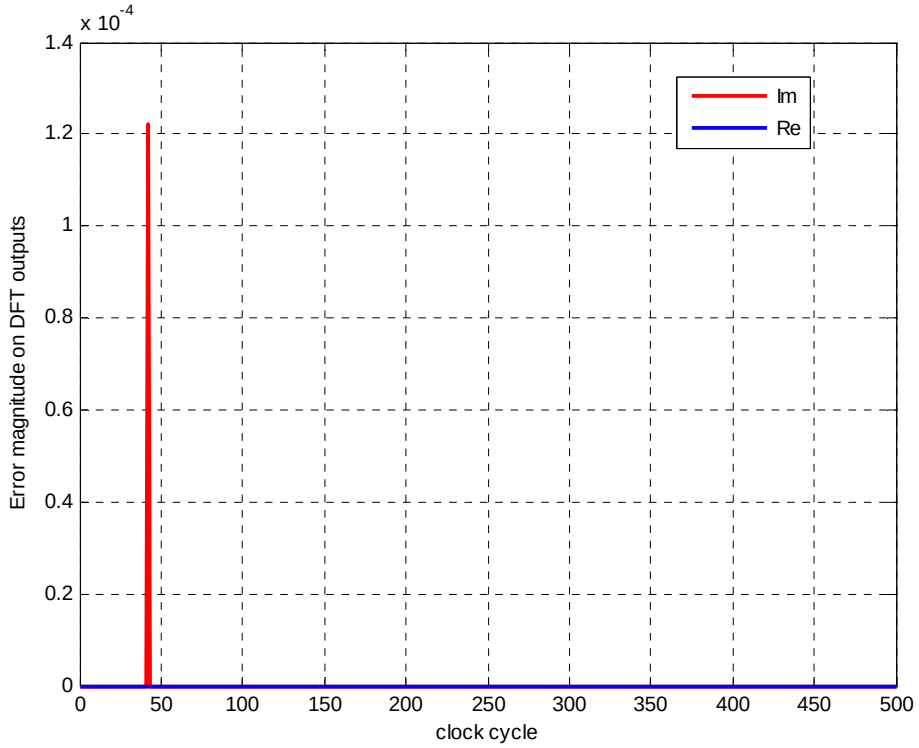

Figure 14. Error magnitude in the output of the *Checksum Based Fault Tolerant FFT Enhanced* when it is masked by the threshold used to address different round off errors (stage *Sum_E*).

Although the fault tolerance of *Checksum-based fault tolerant FFT* techniques proposed in this document is near to the ideal one (the maximal error magnitude observed in the output when the faults are masked by the round off errors is around 2^{-13}), the evolution of this capacity (the fault tolerance or fault coverage) when the FFT size increases must be analyzed.

In this case, the higher the number of points of the FFT, the lower is the fault coverage, i.e., the relation between the FFT size and the fault coverage of the checksum based approach is inversely proportional. The error coverage is reduced because of the accumulation or errors in the checksums values when the number of points increases. However, as these techniques are proposed for FFTs with a reduced number of points the fault coverage reduction of the fault tolerant techniques when the size increases is not a problem to consider in our specific case. Besides, although these techniques were used for FFTs with a large value of N the fault coverage can be enhanced by means of increasing the finite precision of the input samples (using more bits to represent their values). Increasing the number of bits of the inputs signals (1 bit each time) whenever the FFT size is duplicated could be a good option. For further analysis about how to calculate the fault coverage of one checksum based approach see [TaHa93].

As the floating point systems are less affected by round off errors than the fixed point ones, we expect that the fault coverage of the floating point system constructed by using the proposed schemes will be better or equal to the fix point system built.

8.1.2 Fault coverage of the protection techniques based on time and space redundancy

The fault tolerance analysis of the protection technique based on time and space redundancy that uses the third TMR memory as back up stage during the FFT computation (presented in section 6.2.1) has been carried out by means of software fault injection using the SST tool and ModelSim SE 6.1b.

The same input signals used for the test analysis performed to the *Checksums Based Fault Tolerant techniques* has been applied for the different test campaign performed. However, the validation of the *Fault Tolerant Technique based on Recalculation* has not been made as exhaustively as presented in the previous section for two main reasons:

- It is a time redundancy approach and a software fault injection (that takes more time to perform the experiments) have been used
- It is not an error mitigation scheme based on checksums comparisons and the fault tolerance of the system will be similar wherever the fault occurs (LSB or MSB bits)

The behaviour validated by means of software fault injection[2] is the following:

- TMR protection of Data Matrix during the *Gen_E* and *Sum_E* states of the FFT computation.
- Correct operation of the third memory module (use as back up device, saving previous fault free intermediate FFT results).

If differences between FFT intermediate or final results, the detection logic jumps to a new state, known as *correction_E* state that restores the previous fault free results and the values of the FSM registers to recalculate the last operation.

It has been proven that this technique needs additional clock cycles (implementation dependent) to perform the time redundancy, for the specific case study presented the technique implemented needs one cycle to restore the previous fault free state and another one to recalculate the last operation.

[2] Note that the FTU can not be directly used to check time redundancy schemes, because it works with fixed length test-benches and it can not cope with variable execution time in case of recalculation. However, some possible tricks may be used to employ this tool in time redundancy schemes if it is needed, as wait for a time which is long enough even in case of recalculation before reading out the output memory.

It has been proven too that this fault tolerant technique can protect the design against single faults per cycle like the FTMR approach, at the price of increased execution time in case of SEEs during the FFT computation (*Fft_E* state).

This type of systems should trigger a signal to the rest of the system when the fault is found, to notice that the next cycles will be used by the system to resolve the problem, which means that those cycles do not have useful information.

8.2 Area cost

Experimental results of area cost using XST ISE Xilinx synthesizer for FPGAs and Leonardo Spectrum for ASIC are shown in this section.

Synthesis results, in equivalent gates, of the three techniques proposed using Leonardo Spectrum from Mentor Graphics and TSMC 0.25u technology are shown in figures 15 and 16 for FFTs of 8 and 16 points (fixed-point data representation with 16 bits for imaginary and real parts, see section *Case Study*). As it can be observed, in figure 15, the extra area overhead of the *checksum-based protection techniques* proposed in comparison with the non protected version is too high (near to the cost of triplicate). Because of this reason these techniques may not be appropriated to ASICs designs, where triplication of the complete logic (sequential and combinational) is not usually considered (extra area cost too high). However, their area cost can not be compared with the protected version with TMR in sequential logic because this last method do not protect against errors in multipliers and adders (SETs converted in SEUs when they propagate to the input of a storage cell and they meet the setup and hold constrains there). Note that *Checksum based techniques* protect against faults in adders and multipliers. In this scenario, although it is impracticable in many real applications, the area cost of the proposed solutions should be compared with a systematic error mitigation mechanism with equivalent fault tolerance level as, for example, the triplication of the complete FFT module and the addition of some voting logic at the end of the comparison (its area cost, in equivalent gates, would be the triple of the area cost of the non protected version, approximately).

The *fault tolerant technique based on space and time redundancy* shows an extra area cost that might be assumed in some applications. In case of considering TMR protection of input data and, for greater and real FFTs, a rotation scheme to prevent from unnecessary data movements between memories its extra area overhead may be affordable, (note that logic described in figure 9 would change, input memories to FFT computation would be rotated). See figure 16, where no rotation improvement is used, for greater designs it should be used.

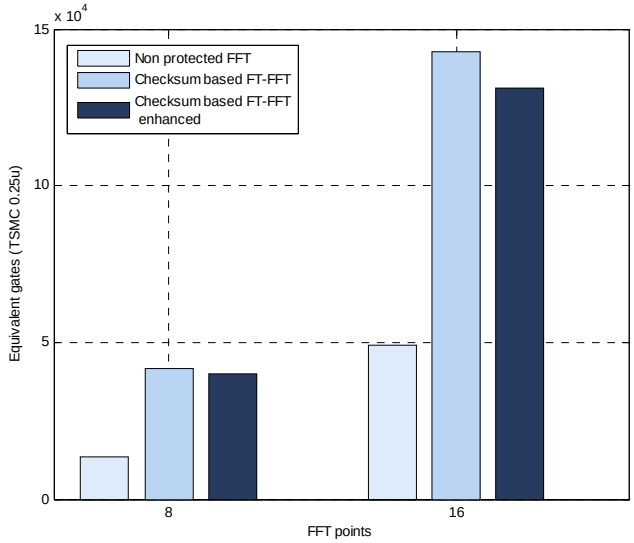

Figure 15. Synthesis results in equivalent gates of the protection techniques based on checksums. Comparison with the non-protected version.

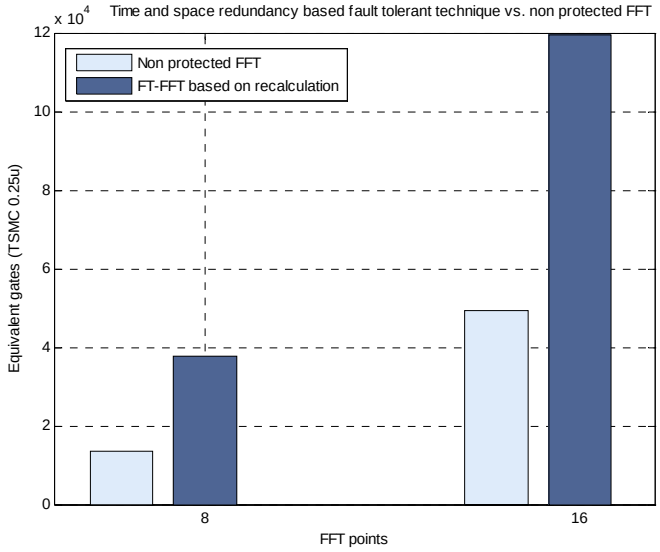

Figure 16. Synthesis results in equivalent gates of the protection technique base on space and time redundancy. Comparison with the non-protected version.

Table 2 shows the area cost results of the different *Checksum Based Fault Tolerant FFT Techniques* applied to an 8-point FFT of 16 bits fixed point representation for imaginary and complex parts. Also the area cost of the non protected version of the same 8-point FFT and the FTMR protected implementation are shown in the table. The results are illustrated in terms of number of different type of sub-modules inside the simple slice of the target device *xc2v6000-4ff1152 Xilinx Virtex* (number of LUTS, multipliers, flip-flops and total slices).

	Non-protected FFT	**Checksum-Based FT-FFT**	**Checksum-Based FT-FFT with multipliers enhancement**	**FTMR FT-FFT**
Number of FF	311	848	814	933
Number of 4 input LUTs	1731	6092	5563	7602
Number of MULT18X18	12	32	28	36
Number of Slices	896	3158	2889	3801

Table 2. Synthesis results of *Checksum-based Fault Tolerant FFT techniques*. Comparison with the non-protected and the FTMR protected version.

As it can be observed in Table 2, the FTMR protected version is the one whose extra area cost (the number of all types of resources is higher than the same resource in the proposed error mitigation techniques based on checksums) is the highest one. The relative area overhead is shown in Table 3, expressed as a percentage with respect to the area of the *Non-protected FFT*. As it can be observed the cost increment of the two proposed solutions to protect against SEE (SEU, SET) is lower than the obtained for the FTMR protected version.

	Non-protected FFT	**Checksum-Based FT-FFT**	**Checksum-Based FT-FFT with multipliers enhancement**	**FTMR FT-FFT**
Number of FF	-	172%	162%	200%
Number of 4 input LUTs	-	250%	221%	340%
Number of MULT18X18	-	167%	133%	200%
Number of Slices	-	252%	223%	324%

Table 3. Area cost increment with respect to the non-protected version for the *Checksum Based Protection Techniques* and the FTMR error mitigation mechanism.

Similar results in number of resources of each type are shown in Table 4 for the *Fault tolerant Technique based on Space and Time Redundancy* which uses one of the three memories used for TMR protection of the input and final data to back-up the results of the previous fault free FFT computation step. The relative area cost increment of this technique is also illustrated in Table 5, again as a percentage with respect to the non protected FFT. Again, the area cost of the proposed implementation is lower than the equivalent cost of the FFT protected by means of FTMR.

	Non-protected FFT	FTMR FT-FFT	Backup memory FT-FFT
Number of FF	311	933	933
Number of 4 input LUTs	1731	7602	5695
Number of MULT18X18	12	36	24
Number of Slices	896	3801	2966

Table 4. Synthesis results of the protection technique based on a backup memory. Comparison with the non-protected and the FTMR protected version

	Non-protected FFT	FTMR FT-FFT	Backup memory FT-FFT
Number of FF	-	200%	200%
Number of 4 input LUTs	-	340%	226%
Number of MULT18X18	-	200%	100%
Number of Slices	-	324%	231%

Table 5. Area cost increment with respect to the non-protected version for the Technique based on a backup memory and the FTMR error mitigation mechanism.

Similar results to the shown in tables 3 and 5 have been obtained for a 16-point FFT.

As the FFT implementation used is not a modular or scalable design (bit reversal and twiddle factors generation functions are not implemented, they are accomplished by means of LUT), the area cost increment of the techniques when the number of points of the FFT increases can be extrapolated using a mathematical approximation, taking into account that the area cost of the comparator is not considered in all schemes because its overhead is negligible as the FFT gets larger (however, for the FTMR protected version, an approximation that takes into account the area cost of the extra voters is used, but these voters are used for each register or combinational logic protected, their area cost when the size of the FFT increases is not negligible). Therefore, the ratio area overhead of each presented technique will be approximately:

- Extra Overhead of the *Checksum-Based FT-FFT*:
$$\frac{N}{2}\cdot A_i + \left(\frac{N}{2}-2\right)\cdot M_i + (N-1)\cdot A_o + \frac{N}{2}\cdot C + 1 \textit{extra FFT datapath} \qquad (36)$$
- Overhead of the *Checksum-Based FT-FFT enhanced:*
$$\frac{N}{4}\cdot A_i + \left(\frac{N}{4}-2\right)\cdot M_i + (N-1)\cdot A_o + \frac{N}{4}\cdot C + 1 \textit{extra FFT datapath} \qquad (37)$$
- Overhead of the *Backup memory FT-FFT*:
$$1 \textit{extra FFT datapath} \qquad (38)$$
- Overhead of the *Fault tolerant FFT based on recalculation by Goertzel*
$$A_i + M_i + C + 1 \textit{extra FFT datapath} \qquad (39)$$

- Overhead of the FTMR FT-FFT:

$$Aproximately\ 2.5\ times\ the\ area\ cost\ of\ the\ non-protected\ FFT\ version \quad (40)$$

Where:

- A_i and A_o are the 2-input adders used to perform the checksum calculation at the input and output sides, respectively.
- M_i and M_o represent the complex multipliers used for the checksums at the input and output sides, respectively.
- C is the number of storage registers (to store complex values with the same precision than the input samples) that the technique needs to store the checksums results.

As it can be concluded from the tables/figures shown in this section, related with the area cost of the fault tolerant implementations presented in section 6 for an FFT of 8 (16) points, all of them have an area overhead that is below the area cost of the FTMR protected version. However their area cost increment comparing with the non protected version is still quite large.

9 CONCLUSSIONS AND FUTURE WORK

A lot of fault tolerant implementations (similar to the presented in this work) against functional errors in FFT designs have already been proposed. However, the area cost of most of them is very large only for error detection purposes and extra area overhead or throughput reduction must be considered in case of including error correction mechanisms. Also, most of them are contained in the fault tolerant techniques known as ABFT solutions and based on the use of different checksum schemes to detect the faults. As it has been proved these types of fault tolerant techniques have the problem of a fault coverage level less than 100%.

Finding an optimal solution, in area cost and fault coverage terms, does not seem to be an easy question because of the redundancy reduction and symmetry of the FFT computation itself, that makes it difficult to find a simple fault tolerant scheme to protect the FFT computation against SEEs, although the *System Knowledge* concept is used. However, the idea of using some structure, inherent fault tolerant or computational characteristics of the system under protection to perform a quasi-optimal fault tolerant implementation has been employed for other DSP designs and could be applied to other type of DFT processing implementations as the Sliding DFT.

The effectiveness to deal with SEEs of one specific fault tolerant implementation must be analyzed by mean of fault injection. In this case, the fault coverage level of the techniques proposed has been analyzed using two different fault injection mechanisms, the FT-Ushades and the SST hardware and software fault injection tools, respectively.

The solutions proposed in this document could be used as an alternative to the FTMR protection (or other equivalent systematic error mitigation technique as XTMR) when some error latency cycles or a very small error can be assumed in the normal system operation.

Checksum based fault tolerant techniques proposed assume that the SEEs occurrence is low enough to assure a correct operation. Also, they have the problem that some faults are not simply detected due to round-off errors. Depending on the implementation, but for most of the high-throughput FFTs, the maximal frequency operation of the *Checksum based fault tolerant techniques* is reduced when comparing with the FTMR protected implementation (noticeable as the FFT size increases). However, for high-throughput implementations the area cost of the proposed techniques seems to be a bit lower than the obtained for FTMR.

The probability of occurrence of the small magnitude error observed at the output of the *Checksum based protection techniques* can be reduced comparing the results of the two FFTs with the *Simple Checksums* and selecting the FFT outputs with the smaller difference. In this case, some errors at the output of the system are still found and due to the extra area cost of this proposal it is not considered (because it does not resolve the problem completely).

For all the above reasons, extrapolating the *Checksum based techniques proposed* to FFTs with *N* large does not seem to be a good option. But, in case of need, several points should be considered as, for example: increasing the precision of the signal representation, protecting the checksums and the input data using TMR or EDAC codes, analyzing the error accumulation in checksum and FFT computations…

The *fault tolerant error mitigation technique based on the combination of space and time redundancy* (section 6.2) also has a lower extra area cost than the FTMR protected version but, if the SEE occurs during the FFT computation, the technique proposed needs some extra clock cycles to correct the erroneous behaviour. However, most of the time (when no errors are found during the computation) the throughput of this technique is 100%. As additional latency in the case of error (low probability of occurrence) is not necessarily a penalty[4] and this solution is an exact protection technique (in the sense that errors not covered due to round off are not presented) we consider that some future work can be proposed like:

- Develop the fault tolerant technique proposed using a more generic FFT application and analyze its extra area cost.
- Try to reduce the area overhead of the fault tolerant implementation performed, using different optimizations (similar to the mentioned before used to prevent from copying memories).
- Compare the area overhead with other type of systematic protections as XTMR, because FTMR seems to have fault tolerance problems[3].
- Carry out more exhaustive fault injection campaigns on it using a test system like FLIPPER to perform fault injection in configuration memory [ACDP07].

Other ideas for future work related to fault tolerant FFT implementations are the following:

- Try to use the redundancy of the data that the FFT processes (each specific FFT application should be analyzed to find the redundancy of the data used) to perform a fault tolerant implementation at higher levels of the design (application level).

- Apply the *System Knowledge* concept to other type of DFT hardware implementations, as the sliding DFT. In this case, similar protection to the proposed for FIR filters could be analyzed

[3] The selection of FTMR to compare the techniques proposed here was made to not consider fault tolerant schemes of any specific FPGA vendor.

[4] In fact, this is done also for microprocessors, where parity detection is concurrent to instruction execution, and only in case of a parity error, the pipeline is re-winded and the operation redone.

10 REFERENCES

[ACDP07] M. Alderighi, F. Casini, S. D'Angelo, S. Pastore, G.R. Sechi, R. Weigand, "*Evaluation of Single Event Upset Mitigation Schemes for SRAM based FPGAs using the FLIPPER Fault Injection Platform*", IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (ISSN: 1550-5774), Sep. 2007, pp. 105-113.

[AgBM04] M. Aguirre Echánove, V. Baena and F. Munoz, "*FT-UNSHADES Design Test and Analysis Tools*", May 2004.

[AgTo04] M. Aguirre Echánove and J. Tombs, "*FT-UNSHADES Design Preparation Tools*", May 2004.

[AnSa91] A. Antola and M. G. Sami, "*Testing and diagnosis of FFT arrays*," J. VLSI Signal Processing, no. 3, 1991, pp. 225–236.

[BaBM00] M.P. Baze, S.P. Buchner and D. McMorrow, "*A Digital CMOS Technique for SEU Hardening*", IEEE Transactions on Nuclear Science Vol. 47, No. 6, Dec. 2000, pp. 2603-2608.

[ChMa88] Y.-H. Choi and M. Malek, "*A Fault-Tolerant FFT Processor*", IEEE Transactions on Computers, Vol. 37, No. 5, May 1988, pp. 617-622.

[CiJG07] Grzegorz Cieslewski, Adam Jacobs, and Alan D. George, "*Fault-Tolerant 2D Fourier Transform with Checksum Encoding*", Aerospace Conference, 2007, IEEE, Vol. 3, No.10, March 2007, pp. 1-11.

[DoMa03] P.E. Dodd and LL. Massengill, "*Basic Mechanisms and Modeling of Single-Event Upset in Digital Microelectronics*", IEEE Transactions on Nuclear Science Vol. 50, No. 3, June 2003, pp. 583-602.

[DoSS04] P.E. Dodd, M.R. Shaneyfelt, J.A. Felix and J.R. Schwank, "*Production and Propagation of Single-Event Transients in High-Speed Digital Logic ICs*", IEEE Transactions on Nuclear Science Vol. 51, No. 6, Dec. 2004, pp. 3278-3284.

[FeML93] C. Feng, J. C. Muzio, and F. Lombardi, "*On the testability of the array structures for FFT computation*", J. Electronic Testing: Theory and Applications, vol. 4, Aug. 1993, pp. 215–224.

[Gonz04] D. Gonzalez-Gutierrez, "*Single Even Upset Simulation Tool Functional Description*", ESA Report TEC-EDM/DCC-SST2, July 2004.

[Hanb02] S. Hanbic, "*FTMR:Functional Triple Modular Redundancy*", ESA Report FPGA-003-001, Dec. 2002.

[JaLy03] E. Jacobsen, R.Lyons, "*The sliding DFT*", IEEE Signal Processing Magazine, March 2003, pp 74-80.

[JoAb88] Jing-Yang Jou and Jacob A. Abraham, "*Fault-Tolerant FFT Networks*", IEEE Transactions on Computers, Vol. 31, No. 5, May 1988, pp. 548-561.

[LLHW00] Jin-Fu Li, Shyue-Kung Lu, Shih-Arn Hwang and Cheng-Wen Wu, "*Easily Testable and Fault Tolerant FFT Butterfly Networks*", IEEE transactions on circuits and systems-II Analog and Digital Signal Processing, Vol.47, No. 9, Sep. 2000, pp. 919-929

[LoMu92] F. Lombardi, J.C. Muzio, "*Concurrent Error Detection and Fault Location in an FFT Architecture*", IEEE Journal of Solid State Circuits, Vo. 27, No. 5, May 1992, pp. 728-736.

[LuSH05] S. Lu, J. Shih and S. Huang, "*Design-for-testability and fault-tolerant techniques for FFT processors*", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 13, No. 6, June 2005, pp. 732-741.

[LuWK93] S.K. Lu, C.W. Wu and S.Y. Kuo, "*Enhancing testability of VLSI arrays for fast Fourier transform*", Proc Inst. Elect. Eng.—E, vol. 140, no. 3, May 1993, pp. 161–166.

[Lyon01] R.G.Lyons, "*Understanding Digital Signal Processing*", Ed. Prentice Hall, 2001. ISBN 0-201-63467-8.

[NaAB96] V.S.S. Nair, J.A.Abraham and P. Banerjee, "*Efficient Techniques for the Analysis of Algorithm-Based Fault Tolerance (ABFT) Schemes*", IEEE Transactions on Computer, vol. 45, No. 4, April 1996, pp. 499-503.

[OhYo94] C. G. Oh and H. Y. Youn, "*On concurrent error location and correction of FFT networks*", IEEE Transactions on VLSI Systems, Vol. 2, June 1994, pp.257–260.

[OhYo95] C. G. Oh, H. Y. Youn and V. K. Raj, "*An efficient algorithm-based concurrent error detection for FFT network*", IEEE Transactions on Computers, Vol. 44, Sep. 1995, pp. 1157-1162.

[OpSh75] A.V. Oppenheim, R.W.Schafer, "*Digital Signal Processing*", Prentice Hall, 1975. ISBN 0-13-214635-5

[OpWi97] A. V. Oppenheim, A. S. Willsky, " *Signals And Systems*", Ed. Prentice Hall, 1997. ISBN 0-13-651175-9

[ReBa90] A. Reddy and P. Banarjee, "*Algorithm-based fault detection for signal processing applications*", IEEE Transactions on Computers Vol. 39, No. 10, Oct. 1990, pp. 1304-1308.

[RMRR07] O. Ruano, J.A. Maestro, P. Reyes and P. Reviriego, "*A Simulation Platform for the Study of Soft Errors on Signal Processing Circuits through Software Fault Injection*", Proceedings of IEEE International Symposium on Industrial Electronics, June 2007.

[RRM007b] P. Reviriego, P. Reyes, J.A. Maestro, O. Ruano, "*System Knowledge-Based Techniques against SEUs for Adaptive Filters*", Proceedings of the RADECS 2007 Conference, Deauville (France), Sep. 2007.

[RRMO06] P. Reyes, P. Reviriego, O. Ruano, J.A. Maestro, "*Efficient Structures for the Implementation of Moving Average Filters in the Presence of SEUs using System Knowledge*", Proceedings of the RADECS 2006 Conference, Athens (Greece), Sep. 2006.

[RRMO07c] P. Reyes, P. Reviriego, J.A. Maestro, O. Ruano, "*A New Protection Technique for Finite Impulse Response (FIR) Filters in the Presence of Soft Errors*", Proceedings of IEEE International Symposium on Industrial Electronics, June 2007.

[RRMO08] P. Reyes, P. Reviriego, J.A. Maestro, O. Ruano, "*Fault Tolerance Analysis of Communication System Interleavers: the 802.11a Case Study*", Journal of VLSI Signal Processing Systems (in press)

[RRMR07a] P. Reyes, P. Reviriego, J.A. Maestro, O. Ruano, "*New Protection Techniques against SEUs for Moving Average Filters in a Radiation Environment*", IEEE Transactions on Nuclear Science (ISSN: 0018-9499), Vol. 54, No. 4, Aug. 2007, pp. 957-964.

[ShFl04] R.D. Schrimpf and D.M. Fleetwood, "*Radiation effects and soft errors in integrated circuits and electronic devices*", World Scientific Publishing, 2004 (ISBN: 981-238-940-7).

[ShSL03] B. Shim, N.R. Shanbhag and S. Lee, "*Energy-efficient soft error-tolerant digital signal processing*", Signals, Systems and Computers, 2003. Conference Record of the Thirty-Seventh Asilomar, Nov. 2003, pp. 1493-1497.

[Stur04] F. Sturesson, "*Application-like Radiation Test of XTMR and FTMR Mitigation Techniques for Xilinx Virtex-II FPGA*", 2004-11-03. D-P-REP-01272-SE

[Stur06] F. Sturesson "*Particle Test of Xilinx Virtex-II FPGA using XTMR Mitigation Technique*", 2006-09-20 Saab Ericsson Space. D-P-REP-01567-SE

[SuRe] J.L. Sung and G. R. Redimbo, "*Practical Algorithm-Based Fault Tolerant DFT System Implementation on an Hypercube Multiprocessor*"

[TaHa93] D.L.Tao, C.R.P. Hartmann, "*A Novel Concurrent Error Detection Scheme for FFT Networks*", IEEE Transactions on Parallel and Distributed Systems, vol.4, no.2, Feb. 1993, pp. 198–221.

[WaJh94] S. Wang and N.K. Jha, "*Algorithm-based fault tolerance for FFT networks*", IEEE Transactions on Computers Vol. 43, No. 7, July 1994, pp. 849-854.

[WuCh93] C.W. Wu and C.T. Chang, "*FFT butterfly network design for easy testing*", IEEE Trans. Circuits Syst. II, vol. 40, Feb. 1993, pp. 110–115.

APPENDIX A FFT RADIX-2 DECIMATION IN TIME

Let us consider the computation of the $N = 2^l$ point DFT by the divide-and conquer approach. If we split the N-point data sequence into two $N/2$-point data sequences $f_1(n)$ and $f_2(n)$, corresponding to the even-numbered and odd-numbered samples of $x(n)$, respectively:

$$\begin{aligned} f_1(n) &= x(2n) \\ f_2(n) &= x(2n+1), \quad n = 0,1,2,\ldots \tfrac{N}{2} - 1 \end{aligned} \tag{41}$$

Thus, $f_1(n)$ and $f_2(n)$ are obtained by decimating $x(n)$ by a factor of 2, and hence the resulting FFT algorithm is called a *decimation-in-time algorithm*.

Now the N-point DFT can be expressed in terms of the DFT's of the decimated sequences as follows:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) \cdot W_N^{kn} \quad k = 0,1,\ldots N-1 \\ &= \sum_{n\,even} x(n) \cdot W_N^{kn} \quad + \sum_{n\,odd} x(n) \cdot W_N^{kn} \\ &= \sum_{m=0}^{\left(\frac{N}{2}-1\right)} x(2m) \cdot W_N^{k2m} \quad + \sum_{m=0}^{\left(\frac{N}{2}-1\right)} x(2m+1) \cdot W_N^{k(2m+1)} \end{aligned} \tag{42}$$

But $W_N^2 = W_{N/2}$. With this substitution, the equation can be expressed as

$$\begin{aligned} X(k) &= \sum_{m=0}^{\left(\frac{N}{2}-1\right)} f_1(m) \cdot W_{\frac{N}{2}}^{km} \quad + W_N^k \cdot \sum_{m=0}^{\left(\frac{N}{2}-1\right)} f_2(m) \cdot W_{\frac{N}{2}}^{km} \\ &= F_1(k) + W_N^k \cdot F_2(k), \qquad k = 0,1,\ldots, N-1 \end{aligned} \tag{43}$$

Where $F_1(k)$ and $F_2(k)$ are the $N/2$-point DFTs of the sequences $f_1(m)$ and $f_2(m)$, respectively.

Since $F_1(k)$ and $F_2(k)$ are periodic, with period $N/2$, we have $F_1(k+N/2) = F_1(k)$ and $F_2(k+N/2) = F_2(k)$. In addition, the factor $W_N^{k+N/2} = -W_N^k$.

Hence the equation may be expressed as

$$\begin{aligned} X(k) &= F_1(k) + W_N^k \cdot F_2(k), \quad k = 0,1,...,\frac{N}{2}-1 \\ X\left(k+\frac{N}{2}\right) &= F_1(k) - W_N^k \cdot F_2(k), \quad k = 0,1,...,\frac{N}{2}-1 \end{aligned} \tag{44}$$

We observe that the direct computation of $F_1(k)$ requires $(N/2)^2$ complex multiplications. The same applies to the computation of $F_2(k)$. Furthermore, there are $N/2$ additional complex multiplications required to compute $W_N^k F_2(k)$. Hence the computation of $X(k)$ requires $2(N/2)^2 + N/2 = N^2/2 + N/2$ complex multiplications. This first step results in a reduction of the number of multiplications from N^2 to $N^2/2 + N/2$, which is about a factor of 2 for N large [OpSh75], [OpSh97].