Offset DMR: A Low Overhead Soft Error Detection and Correction Technique for Transform-Based Convolution

Pedro Reviriego, Chris Bleakley, Juan Antonio Maestro, and Anne O'Donnell

Abstract—A novel concurrent soft error detection and correction scheme is introduced for parallel hardware implementations of transform-based convolution. The proposed technique is based on the structure of radix-2 Fast Fourier Transforms (FFT) of length 2^n where n is an integer. The scheme can provide up to 100 percent detection and correction of isolated single soft errors in the convolution at the cost of little more than double the system area, rather than triple, as is required when using conventional Triple Modular Redundancy (TMR).

Index Terms—Error detection and correction, transform-based convolution, signal processing, FFT.

1 Introduction

An increasingly important problem in digital circuit design is the detection and correction of soft errors [1]. These transient errors, arising, for example, from single radiation events, can cause stored data bits to be corrupt until new data is written to the memory location. The errors manifest themselves as bit flips at the output of a register or combinational logic gates. Traditionally, soft errors have been a major concern for space applications due to circuit exposure to high radiation levels. With the introduction of smaller device geometries and new process technologies, soft errors are now becoming an issue for ground-level applications as well [1]. Several methods have been proposed to protect circuits from the effects of soft errors. These range from the use of specific manufacturing techniques to the addition of redundancy at the system level to detect and correct the errors [2].

Convolution and filtering are among the most common functions in Digital Signal Processing systems. These functions can be efficiently implemented for long sequences using the Fast Fourier Transform (FFT) [3]. The advantage of transform-based convolution and filtering is that convolution is reduced to multiplication in the transform domain, significantly reducing computational complexity. Therefore, transform-based convolutions are widely used in many signal processing applications.

The use of convolution in many applications and the cost effectiveness of transform-based convolution implementations provide a strong motivation to study fault-tolerant techniques for transform-based convolutions. This is the objective of this paper. The errors introduces a new low overhead soft error detection and correction scheme for parallel hardware implementations of transform-based convolution. The scheme uses a redundant convolution module, the input to which is offset by a number of samples from the original. The module in error is determined by analysis of the mismatch pattern at the module outputs. The scheme is designed for FFT-based Overlap Add convolutions with transform length 2^n where n is an integer. For large transform lengths, the area of the proposal tends to two-thirds of the area of a conventional Triple Modular Redundancy (TMR) implementation.

The remainder of the paper is structured as follows: Section 2 describes previously published research in the area of soft and permanent errors for transform-based convolution. Section 3 describes background theory on fast convolution by means of transforms. The proposed soft error detection and correction scheme is introduced in Section 4. Section 5 compares the area of the proposed Offset Double Modular Redundancy (DMR) scheme with that of TMR. A brief conclusion is given in Section 6.

2 Related Work

The traditional approach to dealing with faults has been the use of Modular Redundancy (MR), which replicates the circuit such that errors can be detected when two modules are used and corrected when three identical modules are used. The first configuration is known as DMR and the latter configuration is known as TMR and is widely used for fault tolerance [2]. For some applications, such as signal processing, ad hoc protection schemes have been proposed [4], [5]. These are known as Algorithm-Based Fault Tolerance (ABFT) [6] techniques, in which fault tolerance is incorporated in to the algorithm at the system level. ABFT has, for example, been applied to the computation of the Fast Fourier Transform [7]. Various ABFT approaches have been proposed for fault-tolerant convolution. For example, in [8] cyclic error-correcting codes are used to implement fault-tolerant convolution. The proposed approach works for the direct implementation of the convolution but not for transform-based convolutions. This is a major drawback as the cost of the direct implementation is, in most cases, much larger than that of a transform-based implementation. In [9] an approach based on the use of a Residue Number System (RNS) for the computation is presented. In this case, the implementation has the same underlying structure as that of transform-based convolutions and therefore the cost can be competitive. The results in [9] show an overhead between 65 and 195 percent depending on the parameters of the convolution. These results compare favorably with the cost of TMR (200 percent). The use of two independent convolutions with different transform lengths has been recently proposed in [10]. The approach uses the error patterns associated with each convolution length to determine the module in error and perform the correction. The approach presented in this paper extends this idea but using the same transform length in both convolutions therefore simplifying the implementation. Also recently RNS approaches that relax some of the constraints imposed in [9], such as the need of the moduli to be coprime have been presented [11]. The use of Number Theoretic Transforms (NTTs) combined with RNS to provide fault-tolerant convolutions has also been recently addressed [12]. Other approaches that are not based on RNS have also been presented. For example, in [13] an approach based on computing partial results for arithmetic operations in parallel to correct errors and then merging the partial results to get a correct final value is presented and applied to the implementation of fault-tolerant convolution. This technique is a combination of temporal and spatial redundancy. Such a combination is needed as temporal redundancy alone cannot correct permanent failures. In [14], fault-tolerant techniques for bit modular implementations of convolution are studied. Finally, as transform-based convolution is composed of two transforms, methods for fault-tolerant transforms can be used, as proposed in [15], where concurrent error detection is used in both transforms to provide fault-tolerant convolution. The proposed scheme however only performs error detection. A number of alternative concurrent error detection schemes for FFTs

- *P. Reviriego and J.A. Maestro are with Universidad Antonio de Nebrija, C/Pirineos 55, Madrid E-28040, Spain. E-mail: {previrie, jmaestro}@nebrija.es.*
- *C. Bleakley and A. O'Donnell are with University College Dublin, Belfield, Dublin 4, Ireland. E-mail: chris.bleakley@ucd.ie, anneodonnell@eircom.net.*

Manuscript received 18 Feb. 2010; revised 4 May 2010; accepted 25 June 2010; published online 4 Apr. 2011.
Recommended for acceptance by B. Parhami.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2010-02-0117.
Digital Object Identifier no. 10.1109/TC.2011.80.

have also been proposed (see, for example, [7], [16], [17]) that could be used to protect convolutions in a similar way. An example is the use of an IFFT following the FFT and vice versa to detect errors. The IFFT computation would produce the original inputs and any mismatch would mean that an error has occurred. This approach is similar to DMR.

Most of the proposed schemes address both permanent faults or defects and soft errors. Manufacturing defects that cause continuous failure in the operation of some parts of the circuit [6] can significantly reduce yield, raising the cost of the devices. Permanent faults can occur due to circuit degradation over time and cause some elements to fail continuously. To deal with defects or permanent faults, some degree of spatial redundancy is needed as time redundancy operating on circuits with permanent failures does not provide error recovery. TMR and RNS-based approaches are based on spatial redundancy and therefore can deal with permanent failures. Soft errors on the other hand, cause only temporary failure of some elements in the circuit and therefore any scheme that can correct permanent faults can also correct soft errors of the same nature. Time redundancy schemes can deal with soft errors as long as the duration of the malfunction caused by the soft error is smaller that the recomputation time.

The scheme proposed in this paper is a combination of spatial and temporal redundancy that can correct soft errors. The proposed approach is based on using two standard transform convolutions shifted in time so that no sophisticated modification of the convolution implementation is needed. Another feature is that correction can be performed with little recomputation avoiding additional delay. The overhead of the proposed scheme is close to 100 percent, which is competitive with previous methods such as [9] given the simplicity of the proposed scheme.

The approach is an improvement on the one presented in [10] in which two independent convolutions of different lengths were used. In that case, as the error patterns at the convolution output depend on the length of the transform, the module in error can be found. Using Offset DMR, convolutions of the same length can be used resulting in simpler and more effective implementation. Conceptually, the idea is also related to that presented in [18] in which a set of data and the transform of that data using an invertible transform are sent over two communication channels. The inverse transform is performed on the received data to check if errors have occurred. In that case, the transform properties are used to find the channel in error based on the error propagation patterns of the transforms. In this case, the transforms are an integral part of the convolution and the different error patterns are obtained by time shifting the inputs to both modules.

3 Transform-Based Convolution

Direct computation of the circular convolution is a very computationally expensive operation involving N^2 multiplications, where N is the length of the sequences to be convolved. It can be stated as

$$y(n \bmod N) = \sum_{k=0}^{N-1} h(k)x((n-k) \bmod N), \tag{1}$$

where $y(n)$ is the circular convolution of the sequences $h(n)$ and $x(n)$.

The convolution theorem for the Discrete Fourier Transform (DFT) indicates that the circular convolution of two finite sequences can be obtained as the inverse transform of the product of the individual transforms [3]

$$y(n \bmod N) = F^{-1}(F(h(n))F(x(n))), \tag{2}$$

where $F(x(n))$ is the DFT of the sequence $x(n)$. The process of circular convolution using transforms is illustrated in Fig. 1 for a typical filtering application in which one of the sequences is fixed and precomputed (the filter response). For sequences of length N, where $N = 2^n$ and n is an integer, the radix-2 Cooley-Tukey FFT algorithm may be used to reduce the computational complexity of the DFT and Inverse DFT to $N/2 \log 2N$ complex multiplies and $N \log 2N$ complex additions [3]. In the rest of this work N is assumed to be a power of two, such that efficient implementation can be used. This allows the calculation of the circular convolution in $N + 3N \log 2N$ complex operations, assuming that the second sequence is fixed and its DFT is precalculated as is normally the case in filtering applications. For the direct convolution method, there is no benefit in the second sequence being fixed and the number of operations required for the circular convolution is still N^2 multiplications and N^2 additions.

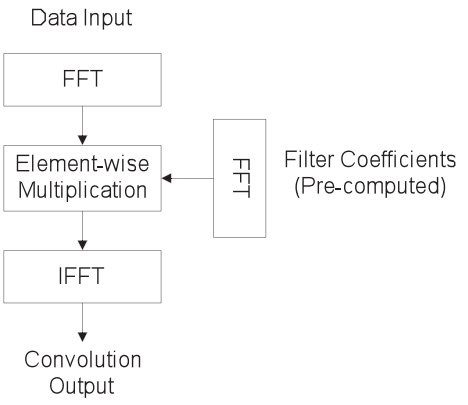

Fig. 1. Circular convolution using transforms.

The FFT consists of n stages of calculation with $N/2$ butterflies in each stage where each butterfly operation is of the form

$$\begin{aligned} y_0 &= x_0 + x_1\omega^k, \\ y_1 &= x_0 - x_1\omega^k. \end{aligned} \tag{3}$$

For efficiency, the butterfly is implemented as a twiddle factor multiplication, $x_1 w^k$, followed by a crossover ($\pm$).

In many practical linear filtering applications, one of the convolution input sequences is very long and the other, which is typically the impulse response of a filter, is comparatively short and fixed. When using transform-based methods, the long sequence is segmented into fixed size blocks prior to processing. The inputs and outputs of the circular convolution must be zero padded and overlapped to allow for the fact that the desired convolution is linear. There are two common methods Overlap Save and Overlap Add. In this work, the Overlap Add method is used [3]. The main parameters for the implementation are the block length L, the filter impulse response length M, and the transform length N. In Overlap Add, the long sequence is broken into blocks of length L. These blocks are padded with $M-1$ zeroes. The value of L is chosen such that the transform length $N = L + M - 1$. The outputs of successive circular convolutions are overlapped by $M-1$ samples and the linear convolution is calculated by adding the samples in the overlap element-wise.

4 Proposed Scheme

Herein, we propose that the convolution is performed by two equivalent modules, operating in parallel on the same input data. The proposed scheme is an evolution of DMR, in that we propose that the blocks of data that the modules operate on are offset from one another by F samples where $0 < F < N$. We refer to this as Offset DMR. The block diagram of the proposed scheme is shown in Fig. 2.

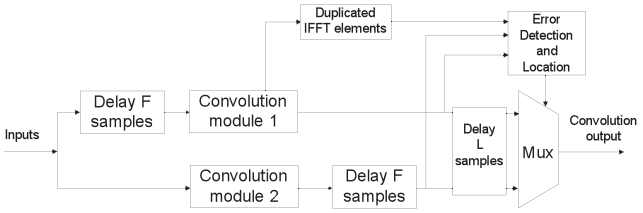

Fig. 2. Block diagram of the Offset DMR approach.

In many environments, the arrival rate of soft errors is so low that the average time between errors is in the range of days [1]. This means that the probability of two soft errors occurring in a short period of time, such as that required to compute a block convolution, is negligible. Therefore, we assume that only one soft error can occur in a short period of time, i.e., consecutive block convolutions. This means that at any one time, a soft error can only corrupt one output block of one convolution module.

In certain cases, the module in error can be determined by examining the pattern of mismatches at the outputs of the two modules. If the mismatches occur within a single output block of module 1 but occur over two output blocks of module 2 then we can conclude that the soft error occurred in module 1. Hence, the system can correct the error by outputting the module 2 results for this block. The condition needed to enable error correction is that the mismatches only correspond to errors in one of the data blocks of one of the modules and correspond to more than one data block of the other module. Depending on where the error occurs and on the value of F, that condition will be met or not.

In some cases, the mismatch pattern is insufficient to determine which block has suffered the error. This can occur when the soft error occurs in the final stages of the inverse transform. Due to the position of the error, very few convolution outputs may be affected and in the extreme case only one. These patterns, which show mismatches corresponding to only one output block of both modules, do not allow for determination of the module in error based solely on outputs of the two modules. Triple Modular Redundancy must be used to protect logic that can give rise to these ambiguous patterns. This can be done by duplicating that logic in one of the convolution modules as shown in Fig. 2. This overhead is small since the patterns can only arise from errors in the final stages of the inverse transform.

To understand the overall approach, it is important to understand how single errors in a convolution propagate to the convolution outputs. Errors can occur in the forward FFT, the multiplication stage or the inverse FFT. The propagation of errors in FFTs has been studied for a long time [19]. The FFT structure determines, for each error location, the number of outputs to which the error propagates and the gain applied to the error as it propagates to each output [20]. In the following, it is assumed that errors at the outputs are such that they cause mismatches between modules that can be detected. The effects of finite precision on the proposed method are discussed at the end of the section.

For example, consider an implementation with the following parameters, $N = 8$, $M = 3$, and $F = 3$ in which a soft error occurs in the second stage of the Inverse FFT (IFFT) of module 1. Fig. 3 shows how the soft error propagates to the outputs of the transform in module 1. These errors show up as mismatches between the outputs of module 1 and 2, as shown in Fig. 4. Since all of the mismatches occur in a single block of module 1, it can be deduced that module 1 is in error and module 2 is correct.

In general, due to the structure of the transform, it can be seen that a soft error in one or more of the inputs to the transform will propagate to all its outputs. This means that any error on the convolution's forward transform, in the multiplication stage or in the twiddle factor multiplication of the first stage of the inverse transform will lead to all of the outputs of the block being

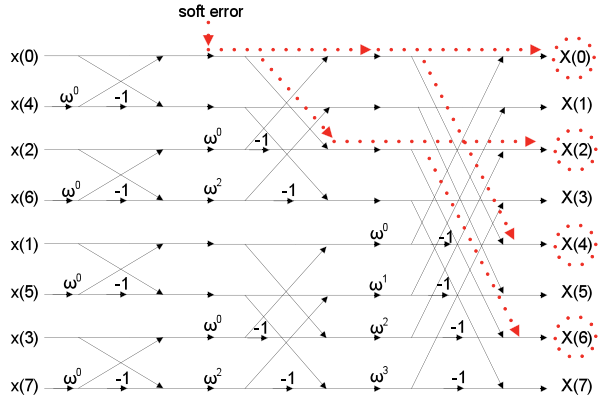

Fig. 3. Eight-point radix-2, FFT showing path of error at the input to stage 2.

mismatched with the corresponding outputs from the other module as the error would propagate to all the outputs in the IFFT stage. If modules 1 and 2 never process the same blocks of data (i.e., $0 < F < N$) then all soft errors in the forward transform, in the multiplication stage, and in the twiddle factor multiplication of the first stage of the inverse transform can be detected and corrected by Offset DMR.

Clearly, soft errors which give rise to single mismatches can be detected but not corrected by Offset DMR since the module in error cannot be identified by analysis of the mismatch pattern. Herein, we refer to mismatch patterns that cannot be corrected in Offset DMR by pattern matching as ambiguities.

The number of ambiguities depends on N, M, and F. We now perform an analysis to determine the offset, F, which will minimize the number of ambiguities, given N and M.

Depending on N, M, and F, a block output from module 1 will overlap with a number of output blocks from module 2. To simplify the analysis, we need only consider two overlaps: the longest overlap that includes the first sample of the module 1 output block (overlap 1) and the longest overlap that includes the last sample of the module 1 output block (overlap 2) as shown in Fig. 4.

The length of the first overlap, in samples, is

$$o_1 = F + M - 1, \tag{4}$$

beginning at the first sample of the module 1 block.

The length of the second overlap is

$$o_2 = N - F, \tag{5}$$

ending at the last sample of the module 1 block.

Consider the mismatch pattern arising from a soft error at the input to the crossover of stage i of the inverse transform, where $i = 0$ is the first stage and $i = n - 1$ is the final stage. Due to the structure of the inverse transform, the number of possible mismatch patterns is 2^i, the number of mismatches in each pattern is 2^{n-i}, and the separation of mismatches is 2^i samples. In Fig. 3, the error is inserted at stage $i = 1$ resulting in $2^{3-1} = 4$ errors

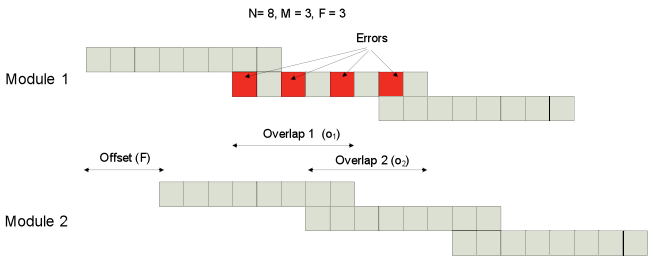

Fig. 4. Offset DMR detection and correction for soft error shown in Fig. 3.

TABLE 1
Mismatch Pattern for $N = 16$, $n = 4$

Stage (i)	Possible patterns	Number of mismatches per pattern	Separation of mismatches
0	1	16	1
1	2	8	2
2	4	4	4
3	8	2	8

separated by 2^1 samples. As an example, Table 1 illustrates the case where $N = 16$. It can be observed that as the error is inserted closer to the input it propagates to more outputs.

It should be noted that not all patterns are equally likely, for example the pattern at stage 0 can be caused by a soft error on any of the nodes of that stage while for patterns in the last stage there are only two nodes on which errors will cause a given pattern.

Ambiguities occur if the first and last mismatches of a given mismatch pattern occur in either one of the overlapping module 2 block outputs.

The number of ambiguities caused by the first overlap due to errors at the inputs to the final stage $(i = n - 1)$ crossovers, i.e., soft errors occurring in the final stage multiplier or penultimate stage $(i = n - 2)$ crossover, is

$$a_1(n-1) = \max\left(o_1 - \frac{N}{2}, 0\right). \tag{6}$$

Similarly, the number of ambiguities caused by the second overlap due to errors at the input to the final stage crossovers is

$$a_2(n-1) = \max\left(o_2 - \frac{N}{2}, 0\right). \tag{7}$$

To combine these it should be taken into account that overlaps may themselves overlap as shown in Fig. 4. This can cause some ambiguities to be included in both a_1 and a_2. Therefore, after adding a_1 and a_2 we need to limit the number of ambiguities to 2^{n-1} to obtain the total number of ambiguities

$$\begin{aligned} a_T(n-1) &= \min\left(2^{n-1}, \max\left(o_1 - \frac{N}{2}, 0\right) + \max\left(o_2 - \frac{N}{2}, 0\right)\right) \\ &= \min\left(2^{n-1}, \max\left(F + M - 1 - \frac{N}{2}, 0\right)\right. \\ &\quad \left. + \max\left(\frac{N}{2} - F, 0\right)\right). \end{aligned} \tag{8}$$

In general, the number of ambiguities due to errors at the inputs to the stage i crossovers is

$$\begin{aligned} a_T(i) &= \min(2^i, \max(o_1 - (N - 2^i), 0) + \max(o_2 - (N - 2^i))) \\ &= \min(2^i, \max(F + M - 1 - N + 2^i, 0) \\ &\quad + \max(o_2 - (2^i - F). \end{aligned} \tag{9}$$

The minimum ambiguity occurs when the overlaps are equal, thus

$$F_{opt} = \frac{N - M + 1}{2}. \tag{10}$$

In this case, the ambiguity arising from errors at the inputs to the crossovers in stage i is

$$a_{opt}(i) = \min(2^i, \max(-N + M - 1 + 2^{i+1}, 0)). \tag{11}$$

Based on this, a soft error occurring at the input to a crossover in stage i will give rise to no ambiguities if

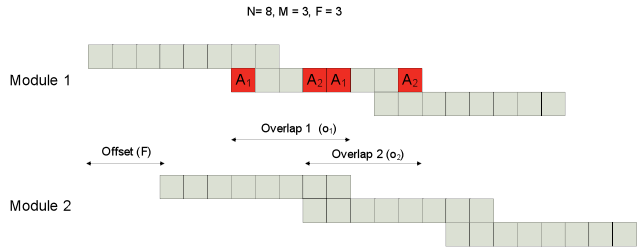

Fig. 5. Example of ambiguities at the inputs of the final stage.

$$M \leq 1 + 2^n - 2^{i+1}. \tag{12}$$

Clearly, for practical cases $(M > 1)$, there will be ambiguities due to errors in the crossovers in the final stage. In addition, there will be $M - 1$ ambiguities due to errors at the input to the crossovers in the final stage $(i = n - 1$ in (11)). However, there can be no ambiguities due to errors prior to the crossover in the penultimate stage $(i = n - 2)$ in (11)) if

$$M \leq 1 + \frac{N}{2}. \tag{13}$$

An example of the $M - 1$ ambiguities at the inputs to the crossovers to the final stage is illustrated in Fig. 5. It can be observed that the first ambiguity A_1 occurs in the first overlap and the second A_2 on the second overlap.

As M becomes larger, then the number of ambiguities becomes greater and the stage at which a soft error can give rise to an ambiguity comes earlier. An ambiguity may be resolved by calculating the value for one of the mismatched outputs a third time. The module in error can then be identified and all mismatches corrected.

TMR must be applied to calculations that, if they suffer an error, would give rise to ambiguous patterns. If we restrict the analysis to the practical case where $M \leq 1 + N/2$, then all ambiguities can be resolved by applying TMR to

- All overlap adders.
- All crossovers in the final stage.
- $M - 1$ twiddle multipliers in the final stage.
- $2(M - 1)$ half crossovers in the penultimate stage.

An example will now be used to explain the need of applying TMR to $2(M - 1)$ half crossovers in the penultimate stage. From (11) we know that there are $M - 1$ points (and patterns) on which errors will not be correctable. Assuming values of $M = 3$ and $F = 3$ as in the previous example, for an eight point transform, one of the patterns that causes an ambiguity is $X(0)$, $X(4)$. In Fig. 6, the elements that need to be triplicated are marked with dotted arrows. It can be seen that in the penultimate stage there are two

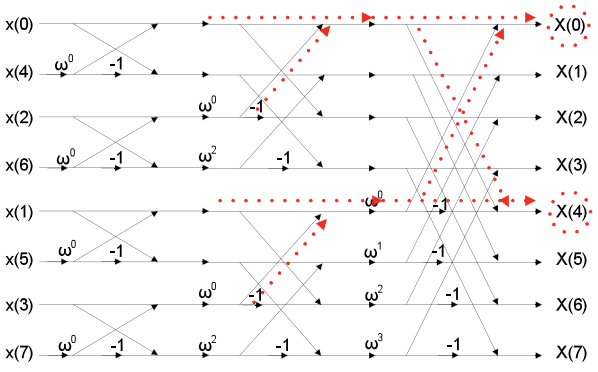

Fig. 6. Example of triplication requirement in the last stages.

half crossovers that need triplication. This occurs for each of the $M-1$ ambiguities giving a total of $2(M-1)$ half crossovers.

The error detection, location, and correction logic can be implemented serially as follows: a counter is associated with each module such that the first output sample of a block is numbered $N-1$ and the last is numbered 0. When there are no errors, the counter is reset to $N-1$ at the beginning of the next block, which occurs at value $M-2$. When a mismatch is detected, the counter value is stored in a register (start sample number) and the counter is let run to zero before initializing it to $N-1$ again. When the last mismatch within N samples of the first is encountered, the counter value is stored in another register (end sample number). N samples after the first mismatch, the end sample number is subtracted from the start sample number to obtain the error span. An error span is obtained for each module. If the error spans have different signs, the positive error span is associated with the module in error. If both spans are positive then the pattern is ambiguous. In this case, the outputs of module 1 and the triple protection logic are compared. If they are the same then module 2 is in error. If they are different then module 1 is in error. If both spans are negative then more than one soft error has occurred and the method fails.

In a fully parallel hardware implementation without pipelining, the arrival times of the errors at the outputs of the module will not be exactly equal. In some cases, this may mean that, for a short period of time, the actual error pattern differs from that predicted by this analysis. If this period coincides with the point at which the result is evaluated the error may not be corrected. The frequency of this occurrence depends on the ratio of the duration of the soft error to the timing skew between output errors.

The problem can be alleviated by careful design of the circuit to reduce skew. Alternatively, the pattern recognition technique may be designed so as to be robust to a small number of missing mismatches. However, most implementations will use pipelining since the complete convolution, or even one of the transforms, can rarely be completed in one clock cycle. In pipelined hardware implementations, once a bit reversal is sampled by a flip-flop, its effect is propagated to the system output. If the pipelining is placed at the input to all crossovers then the pattern detection technique described herein will work correctly in all cases.

In practical systems, the FFT and IFFT use fixed-point arithmetic, leading to round-off error at the module outputs. Based on an analysis of the module or simulation, the round-off error can be bounded to be less than $2*\eta$ [16]. Since the modules operate on different data blocks, the round-off error will be different in corresponding module outputs. Thus a mismatch is only deemed to have occurred in cases where the difference between corresponding outputs is greater than η. For a more complete discussion of the effects of finite precision on the FFT and its implications for ABFT techniques the reader is referred, for example, to [19] and the references therein.

5 Results

In this section, the area of the proposed Offset DMR system is compared with that of TMR, which also provides error detection and correction. TMR has been chosen as it is the traditional benchmark for comparison and also because for most of the previously proposed techniques no detailed cost analysis is provided. Another important reason is that both offset DMR and TMR are straightforward to implement, since both are based on replication of the unprotected design. Alternative approaches, like the ones based on RNS, cyclic codes or concurrent error detection on the transforms require significant changes to the unprotected design and therefore involve a larger design effort than TMR or Offset DMR. As a reference, the figures provided in [9] for the protection overhead lie between 65 and 195 percent compared to values close to 100 percent typically achieved by offset DMR. This indicates that Offset DMR can be competitive in terms of cost.

TABLE 2
Area Savings (Percent) for Proposed Method for Various Bit Widths and FFT Lengths

Bits, b	Transform Lengths, N			
	16	64	256	1024
8	28.99	30.26	30.96	31.40
16	29.51	30.61	31.22	31.61
32	29.76	30.79	31.35	31.72

We consider the most practical case where $M = 1 + N/2$. An FFT of length N has $\log 2N$ stages, each consisting of $N/2$ butterflies. A convolution involves the forward FFT followed by the multiplication stage followed by the inverse FFT. To implement the proposed Offset DMR system, this needs to be duplicated with some additional calculations in one of the convolution modules for the final and penultimate stages. To ensure 100 percent fault coverage all overlap adders in the final stage, i.e., $(M-1)$ additions need to be duplicated in one of the convolution modules. In the worst case, this will involve $N/2$ additions. All crossovers in the final stage also need to be duplicated which involves $N/2$ butterflies with two additions in each giving an extra N additions. Duplicating $M-1$ twiddle factors in the final stage involves an extra $N/2$ multiplications and duplicating $2(M-1)$ half crossovers in the penultimate stage requires an extra N additions.

Each butterfly has one complex multiplication and two complex additions. A complex multiplication can be implemented most efficiently using three integer multiplications and five integer additions [21]. A complex addition consists of two integer additions. One butterfly, therefore, consists of three integer multiplications and nine integer additions.

We now compare the gate areas of the Offset DMR and TMR circuits. Since the circuits have a similar structure, the postlayout area overhead of the internal interconnect is assumed to be the same. Let one integer addition consist of b Full Adders (FA) where b is the bit width. Therefore, one complex addition involves $2b$ FAs. One b-bit integer multiplication can be implemented using $(b-1)$ Half Adders (HA) and $(b-1)(b-3)$ FAs [22]. Since one HA is approximately 3/7 FA, the area of one integer multiplication in terms of b is $b^2 - 25b/7 + 18/7$ which is henceforth referred to as X in order to simplify the expressions. Therefore, the area of a complex multiplication in terms of b-bit integer additions is $3X + 5b$ and similarly, the area of a butterfly is $3X + 9b$.

Equations (14) and (15) provide the number of integer additions required for Offset DMR (including the necessary triple protection logic) and full TMR, respectively, while (16) shows the percentage area saving achieved using Offset DMR rather than TMR

$$\text{Offset DMR area} = 2N\log_2 N(3X+9b) + \frac{15N}{2}X + \frac{35N}{2}b, \tag{14}$$

$$\text{TMR area} = 3N\log_2 N(3X+9b) + 9NX + 15Nb, \tag{15}$$

$$\%\text{Saving} = \frac{N\log_2 N(3X+9b) + \frac{3N}{2}X - \frac{5N}{2}b}{3N\log_2 N(3X+9b) + 9NX + 15Nb} \times 100. \tag{16}$$

The percentage savings for bit widths (b) 8, 16, and 32 with various transform lengths are shown in Table 2.

The formulas above exclude the delay lines and detection and correction logic for both Offset DMR and TMR. If implemented serially, the area cost of the detection and correction logic is small: four subtractors, one absolute value operation, two decrementers, four registers, one comparator, one mux, and some control logic.

In both TMR and Offset DMR, the detection and correction logic is simple compared to the convolution operations. Hence, their area cost is assumed to be negligible compared with that of the overall convolution.

Transform-based convolution has an intrinsic delay of L samples. The proposed technique has an additional intrinsic delay of F samples due to the offset and L samples for mismatch pattern comparisons as part of error detection and location, as shown in Fig. 2.

6 Conclusions

A novel, low overhead soft error detection and correction technique for parallel hardware implementation of transform-based convolution has been introduced. It has been found that area savings of approximately 30 percent compared to TMR can be achieved. An advantage of the proposed scheme is that it does not require the use of sophisticated arithmetic such as RNS or complex codings of the inputs as in many of the concurrent error detection approaches to protect transforms. This simplicity facilitates its application in real designs.

References

[1] R. Baumann, "Soft Errors in Advanced Computer Systems," *IEEE Design and Test of Computers,* vol. 22, no. 3, pp. 258-266, May/June 2005.

[2] M. Nicolaidis, "Design for Soft Error Mitigation," *IEEE Trans. Device and Materials Reliability,* vol. 5, no. 3, pp. 405-418, Sept. 2005.

[3] J.K. Proakis and D.G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Application,* third ed. Prentice Hall, 1996.

[4] B. Shim and N.R. Shanbhag, "Energy-Effcient Soft Error-Tolerant Digital Signal Processing," *IEEE Trans. Very Large Scale Integration Systems,* vol. 14, no. 4, pp. 336-348, Apr. 2006.

[5] Y.-H. Huang, "High-Efficiency Soft-Error-Tolerant Digital Signal Processing Using Fine-Grain Subword-Detection Processing," *IEEE Trans. Very Large Scale Integration Systems Systems,* vol. 18, no 2, pp. 291-304, Feb. 2010.

[6] A. Reddy and P. Banarjee, "Algorithm-Based Fault Detection for Signal Processing Applications," *IEEE Trans. Computers,* vol. 39, no. 10, pp. 1304-1308, Oct. 1990.

[7] J. Jou and J.A. Abraham, "Fault-Tolerant FFT Networks," *IEEE Trans. Computers,* vol. 37, no. 5, pp. 548-561, May 1988.

[8] G.R. Redinbo, "System Level Reliability in Convolution Computations," *IEEE Trans. Acoustics, Speech and Signal Processing,* vol. 37, no. 8, pp. 1241-1252, Aug. 1989.

[9] P.E. Beckmann and B.R. Musicus, "Fast Fault-Tolerant Digital Convolution Using a Polynomial Residue Number System," *IEEE Trans. Signal Processing,* vol. 41, no. 7, pp. 2300-2313, July 1993.

[10] P. Reviriego, J.A. Maestro, A. O'Donnell, and C. Bleakley, "Soft Error Detection and Correction for FFT Based Convolution Using Different Block Lengths," *Proc. IEEE Int'l On-Line Testing Symp.,* pp. 138-143, June 2009.

[11] S. Sundaram and C.N. Hadjicostis, "Fault-Tolerant Convolution via Chinese Remainder Codes Constructed from Non-Coprime Moduli," *IEEE Trans. Signal Processing,* vol. 56, no. 9, pp. 4244-4254, Sept. 2008.

[12] A.B. O'Donnell and C.J. Bleakley, "Area Efficient Fault Tolerant Convolution Using RRNS with NTTS and WSCA," *Electronics Letters,* vol. 44, no. 10, pp. 648-649, May 2008.

[13] Y.M. Hsu, V. Piuri, and E.E. Swartzlander, "Efficient Time Redundancy for Error Correcting Inner-Product Units and Convolvers," *Proc. IEEE Int'l Workshop Defect and Fault Tolerance in Very Large Scale Integration Systems Systems Systems,* pp. 198-206, 1995.

[14] L. Dadda and V. Piuri, "Bit-Modular Defect/Fault-Tolerant Convolvers," *Proc. IEEE Int'l Workshop Defect and Fault Tolerance in Very Large Scale Integration Systems Systems Systems,* pp. 90-98, 1995.

[15] J.M. Tahir, S.S. Dlay, R.N.G. Naguib, and R.R. Hinton, "Self-Checking Architectures for Fast Hartley Transform," *Proc. European Design and Test Conf.,* pp. 363-371, 1995.

[16] G.O. Choong, Y. Hee, and V.K. Raj, "An Efficient Algorithm-Based Concurrent Error Detection for FFT Networks," *IEEE Trans. Computers,* vol. 44, no. 9, pp. 1157-1162, Sept. 1995.

[17] S. Wang and N.K. Jha, "Algorithm-Based Fault Tolerance for FFT Networks," *IEEE Trans. Computers,* vol. 43, no. 7, pp. 849-854, July 1994.

[18] J.J. Metzner, S. Chin, and A. Ray, "Control of Communication Networks for Multiple Mobile Platforms," *Proc. Am. Control Conf.,* vol. 1, pp. 242-247, June 2003.

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

[20] G.R. Redinbo, "Concurrent Error Detection in Fast Unitary Transform Algorithms," *Proc. Int'l Conf. Dependable Systems and Networks,* pp, 37-46, 2001.

[21] R.E. Blahut, *Fast Algorithms for Digital Signal Processing.* Addison-Wesley Publishing Company, 1985.

[22] P.F. Stelling, C.U. Martel, V.G. Oklobdzij, and R. Ravi, "Optimal Circuits for Parallel Multipliers," *IEEE Trans. Computers,* vol. 47, no. 3, pp. 273-285, Mar. 1998.