Efficient Coding Schemes for Fault-Tolerant Parallel Filters

Zhen Gao, Pedro Reviriego, Zhan Xu, Xin Su, Jing Wang, and Juan Antonio Maestro

***Abstract*—As the complexity of communications and signal processing systems increases, so does the number of blocks or elements that they have. In many cases, some of those elements operate in parallel, performing the same processing on different signals. A typical example of those elements are digital filters. The increase in complexity also poses reliability challenges and creates the need for fault-tolerant implementations. A scheme based on error correction coding has been recently proposed to protect parallel filters. In that scheme, each filter is treated as a bit, and redundant filters that act as parity check bits are introduced to detect and correct errors. In this brief, the idea of applying coding techniques to protect parallel filters is addressed in a more general way. In particular, it is shown that the fact that filter inputs and outputs are not bits but numbers enables a more efficient protection. This reduces the protection overhead and makes the number of redundant filters independent of the number of parallel filters. The proposed scheme is first described and then illustrated with two case studies. Finally, both the effectiveness in protecting against errors and the cost are evaluated for a field-programmable gate array implementation.**

***Index Terms*—Coding, parallel filters, soft errors.**

I. Introduction

PARALLEL filters are commonly found in modern signal processing and communication systems [1]. In many cases, the filters perform the same processing on different incoming signals as there is a tendency to use multiple-input–multiple-output systems [2].

This parallel operation can be exploited for fault tolerance. In fact, reliability is a major challenge for electronic systems [3]. In particular, soft errors are an important issue, and many techniques have been proposed over the years to mitigate them [4]. Some of these techniques modify the low-level design and implementation of the integrated circuits to prevent soft errors from occurring. Other techniques work at a higher abstraction level by adding redundancy that can detect and correct errors. One classical example is the use of triple modular redundancy (TMR) in which the design is tripled and a majority vote of the outputs are used to correct errors. Another example is the use of error correction codes (ECCs) to protect the bits stored in memory devices [5]. In this case, a number of parity checks are computed and stored in the memory so that errors can be detected and corrected when the data are read. Finally, for applications that have regular structure and properties, those can be exploited to detect and correct errors with a lower cost than TMR. This is the case for many signal processing circuits [6]. In many cases, ECCs or specific protection techniques are combined with TMR to achieve a complete protection. For example, the ECC encoders and decoders may be protected with TMR to ensure that they are not affected by errors. In those cases, TMR is used to protect a small part of the circuit that cannot be protected by the ECC or the specific technique.

The protection of digital filters has been widely studied. For example, fault-tolerant implementations based on the use of residue number systems or arithmetic codes have been proposed [7], [8]. The use of reduced precision replication or word-level protection has been also studied [9], [10]. Another option to perform error correction is to use two different filter implementations in parallel [11]. All those techniques focus on the protection of a single filter.

The protection of parallel filters has only been recently considered. In [12], an initial technique to protect two parallel filters was proposed. This scheme was generalized in [13], where the use of a scheme based on ECCs was presented. In this work, each filter was treated as a bit on an ECC, and additional filters are added to act as parity check bits. This means that, for single error correction, the number of redundant filters needed is the same as the number of bits needed in a traditional single error correction Hamming code [14]. For example, for four parallel filters, three redundant filters are required, whereas for eight filters, four redundant filters are needed. This scheme therefore significantly reduces the implementation cost compared with that of TMR.

This brief studies the protection of parallel filters using more general coding techniques. In particular, a key difference with ECCs is that both filter inputs and outputs are numbers. Therefore, not only a zero or a one can be used for the coding (as done with ECCs). This can be exploited, as shown in the rest of this brief, to provide error correction by adding only two redundant filters regardless of the number of parallel filters. The reduced number of redundant filters does not affect the ability of the scheme to correct errors but reduces the implementation cost. In the rest of this brief, first, the parallel filters and the existing ECC-based protection scheme are described. Then, the proposed coding scheme is presented and illustrated with a few practical case studies. Finally, the case studies are evaluated for a field-programmable gate array (FPGA) implementation and compared with the previously proposed ECC-based technique.

Manuscript received December 31, 2014; accepted February 11, 2015. Date of publication February 16, 2015; date of current version June 25, 2015. This work was supported in part by the 863 Program of China under Grant 2012AA01A502 and in part by the National Natural Science Foundation of China under Grant 61402044. This brief was recommended by Associate Editor Z. Zhang.

Z. Gao is with the School of Electronic Information Engineering, Tianjin University, Tianjin 300072, China (e-mail: zgao@tju.edu.cn).

P. Reviriego and J. A. Maestro are with Universidad Antonio de Nebrija, Madrid 28040, Spain (e-mail: previrie@nebrija.es; jmaestro@nebrija.es).

Z. Xu is with the School of Information and Communication Engineering, Beijing Information Science and Technology University, Beijing 100085, China (email: xuzhan_work@126.com).

X. Su and J. Wang are with the Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: suxin@tsinghua.edu.cn; wangj@tsinghua.edu.cn).

Color versions of one or more of the figures in this brief are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSII.2015.2404219

1549-7747 © 2015 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

The results show that the use of a more general coding scheme reduces the protection overhead while providing a similar error correction capability to that of the ECC scheme.

II. ECC-Based Protection of Parallel Filters

The impulse response $h[n]$ completely defines a discrete-time filter that performs the following operation on the incoming signal $x[n]$:

$$y[n] = \sum_{l=0}^{\infty} x[n-l] \cdot h[l]. \tag{1}$$

The impulse response can be infinite or be nonzero for a finite number of samples. In the first case, the filter is an infinite impulse-response (IIR) filter, and in the second, the filter is a finite impulse-response (FIR) filter. In both cases, the filtering operation is linear such that

$$y_1[n] + y_2[n] = \sum_{l=0}^{\infty} (x_1[n-l] + x_2[n-l]) \cdot h[l]. \tag{2}$$

This property can be exploited in the case of parallel filters that operate on different incoming signals, as shown on Fig. 1. In this case, four filters with the same response process the incoming signals $x_1[n]$, $x_2[n]$, $x_3[n]$, and $x_4[n]$ to produce four outputs $y_1[n]$, $y_2[n]$, $y_3[n]$, and $y_4[n]$. To detect and correct errors, each filter can be viewed as a bit in an ECC, and redundant filters can be added to form parity check bits [13]. This is also illustrated in Fig. 1, where three redundant filters are used to form the parity check bits of a classical single error correction Hamming code [14]. Those correspond to the outputs $z_1[n]$, $z_2[n]$, and $z_3[n]$. Errors can be detected by checking if

$$\begin{aligned} z_1[n] &= y_1[n] + y_2[n] + y_3[n] \\ z_2[n] &= y_1[n] + y_2[n] + y_4[n] \\ z_3[n] &= y_1[n] + y_3[n] + y_4[n]. \end{aligned} \tag{3}$$

When some of those checks fail, an error is detected. The error can be corrected based on which specific checks failed. For example, an error on filter y_1 will cause errors on the checks of z_1, z_2, and z_3. Similarly, errors on the other filters will cause errors on a different group of z_i. Therefore, as with the traditional ECCs, the error can be located. To correct the error, the failing output is reconstructed from the correct outputs. For example, when an error on y_1 is detected, it can be corrected by making

$$y_{c1}[n] = z_1[n] - y_2[n] - y_3[n]. \tag{4}$$

This ECC-based scheme reduces the protection overhead compared with the use of TMR. Table I summarizes the number of redundant filters needed for different parallel filter configurations. It can be observed that the number grows with the logarithm in base two on the number of filters. Therefore, the cost is much smaller than TMR, in which the number of filters is tripled. The cost reductions were confirmed by some case study implementations in [13].

In this ECC-based scheme, the coding of the redundant filters is based on simple additions that replace the XOR binary operations in traditional ECCs. However, since both the inputs and outputs of the filters are sequences of numbers, a more general coding can be used. This type of coding has been explored for linear time-invariant systems (see, for example, [15] and [16]) but not for parallel filters. In those works, the processing of the linear system is modified to incorporate error detection and correction mechanisms. This is different from the approach proposed in this brief, where inputs are encoded but the processing of the filters is not modified. In the following, the use of a coding scheme for parallel filters in which the redundant filters are constructed as linear combinations of the original filters with arbitrary coefficients is explored.

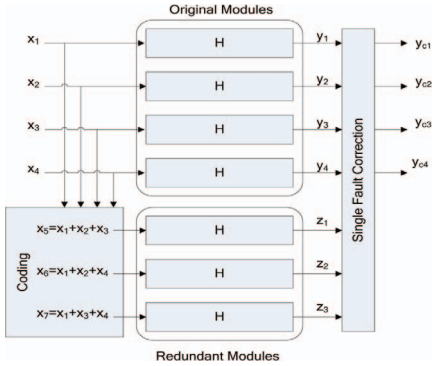

Fig. 1. ECC-based scheme for four filters and a Hamming code (see [13]).

TABLE I
NUMBER OF REDUNDANT FILTERS IN THE ECC-BASED APPROACH

Number of parallel filters	Number of redundant filters
4	3
8	4
16	5
32	6

III. Coding for Fault-Tolerant Parallel Filters

The proposed scheme is illustrated in Fig. 2 for the case of four parallel filters. The input signals are encoded using a matrix with arbitrary coefficients to produce the signals that enter the four original and two redundant filters. In its more general form, this coding matrix A can be formulated as

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \\ a_{51} & a_{52} & a_{53} & a_{54} \\ a_{61} & a_{62} & a_{63} & a_{64} \end{pmatrix}. \tag{5}$$

Therefore, the input signal to the ith filter is of the form ($i = 1, 2, \ldots,$ or 6), i.e.,

$$v_i[n] = a_{i1} \cdot x_1[n] + a_{i2} \cdot x_2[n] + a_{i3} \cdot x_3[n] + a_{i4} \cdot x_4[n]. \tag{6}$$

In a practical implementation, the first four rows of the matrix would be an identity matrix so that the inputs to the original filters are the incoming signals.

With this coding scheme, the outputs of the filters, i.e., $y_1[n]$, $y_2[n]$, $y_3[n]$, and $y_4[n]$, can be obtained as follows:

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}_{1235} = (A_{1235})^{-1} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_5 \end{pmatrix} \tag{7}$$

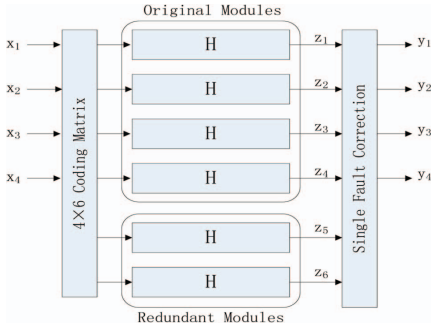

Fig. 2. Proposed coding scheme in general form.

where A_{1235} is a submatrix of A, including the first, second, third, and fifth rows. This process can be repeated with different submatrixes of A, for example, with A_{1236}, A_{2345}, and A_{2346}. In the error-free case, all the recovered versions of $y_1[n]$, $y_2[n]$, $y_3[n]$, and $y_4[n]$ will be the same. When there are differences, an error is detected. For example, suppose that

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}_{1235} \neq \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}_{1236} \quad \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}_{2345} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}_{2346} \tag{8}$$

which means that there is an error among filters {1 2 3 5 6} and that filters {2 3 4 5 6} are correct. Therefore, the faulty filter is filter 1. Then, the error can be corrected by taking the final outputs from a set that does not include filter 1.

The error correction and detection logic can be simplified assuming that there is only a single error. In that case, checking only that, for each recovered set, the sums of the values $y_1[n] + y_2[n] + y_3[n] + y_4[n]$ are equal is enough. In more detail, four checks are needed, each involving five filters and excluding one. For example, if branch 1 is excluded, the error checking would be

$$\begin{cases} s_1^1 = \bar{w}_{2345}(z_2 z_3 z_4 z_5)^T \\ s_1^2 = \bar{w}_{2346}(z_2 z_3 z_4 z_6)^T \\ e_1 = s_1^1 - s_1^2 \end{cases} \tag{9}$$

in which $\bar{w}_{2345} = [1111](A_{2345})^{-1}$, and $\bar{w}_{2346} = [1111](A_{2346})^{-1}$.

Similarly, if filters 2, 3, and 4 are excluded, the e_2, e_3, and e_4 error signals can be generated as follows:

$$\begin{cases} s_2^1 = \bar{w}_{1345}(z_1 z_3 z_4 z_5)^T \\ s_2^2 = \bar{w}_{1346}(z_1 z_3 z_4 z_6)^T \\ e_2 = s_2^1 - s_2^2 \end{cases},$$

$$\begin{cases} s_3^1 = \bar{w}_{1245}(z_1 z_2 z_4 z_5)^T \\ s_3^2 = \bar{w}_{1246}(z_1 z_2 z_4 z_6)^T \\ e_3 = s_3^1 - s_3^2 \end{cases} \quad \begin{cases} s_4^1 = \bar{w}_{1235}(z_1 z_2 z_3 z_5)^T \\ s_4^2 = \bar{w}_{1236}(z_1 z_2 z_3 z_6)^T \\ e_4 = s_4^1 - s_4^2. \end{cases} \tag{10}$$

Then, with the error vector $e = [e_1, e_2, e_3, e_4]$, the faulty filter can be located. The process is listed in Table II, in which "#" stands for a nonzero value. Once the faulty filter is identified, the error can be corrected by reconstructing the outputs using the remaining filters.

For the proposed coding scheme to work, the encoding matrix has to fulfill some conditions. Let us consider the calculation of e_1 and define $(A_{2345})^{-1}$ and $(A_{2346})^{-1}$ as

TABLE II
LOCATION OF THE FAULTY FILTER

$e_1 e_2 e_3 e_4$	Filter in error
0 0 0 0	No error
0 # # #	z_1
# 0 # #	z_2
# # 0 #	z_3
# # # 0	z_4
# # # #	z_5
# # # #	z_6

$$(A_{2345})^{-1} = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{pmatrix} \quad (A_{2346})^{-1} = \begin{pmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{pmatrix}. \tag{11}$$

Then

$$\begin{cases} \bar{w}_{2345} = [1111](A_{2345})^{-1} = \left[\sum_{i=1}^{4} b_{i1}, \sum_{i=1}^{4} b_{i2}, \sum_{i=1}^{4} b_{i3}, \sum_{i=1}^{4} b_{i4}\right] \\ \bar{w}_{2346} = [1111](A_{2346})^{-1} = \left[\sum_{i=1}^{4} c_{i1}, \sum_{i=1}^{4} c_{i2}, \sum_{i=1}^{4} c_{i3}, \sum_{i=1}^{4} c_{i4}\right] \end{cases} \tag{12}$$

and finally

$$\begin{cases} s_1^1 = \bar{w}_{2345}(z_2 z_3 z_4 z_5)^T = z_2 \sum_{i=1}^{4} b_{i1} + z_3 \sum_{i=1}^{4} b_{i2} \\ \qquad + z_4 \sum_{i=1}^{4} b_{i3} + z_5 \sum_{i=1}^{4} b_{i4} \\ s_1^2 = \bar{w}_{2346}(z_2 z_3 z_4 z_6)^T = z_2 \sum_{i=1}^{4} c_{i1} + z_3 \sum_{i=1}^{4} c_{i2} \\ \qquad + z_4 \sum_{i=1}^{4} c_{i3} + z_6 \sum_{i=1}^{4} c_{i4} \\ e_1 = z_2 \left(\sum_{i=1}^{4} b_{i1} - \sum_{i=1}^{4} c_{i1}\right) + z_3 \left(\sum_{i=1}^{4} b_{i2} - \sum_{i=1}^{4} c_{i2}\right) \\ \qquad + z_4 \left(\sum_{i=1}^{4} b_{i3} - \sum_{i=1}^{4} c_{i3}\right) + z_5 \sum_{i=1}^{4} b_{i4} - z_6 \sum_{i=1}^{4} c_{i4}. \end{cases} \tag{13}$$

Therefore, the error check will ensure error detection when the sums of the columns in each of the matrixes A_{2345}^{-1} and A_{2346}^{-1} are different and nonzero. Furthermore, we can define the error detection matrix as $C = [\bar{c}_1, \bar{c}_2, \bar{c}_3, \bar{c}_4]^T$, in which

$$\begin{cases} \bar{c}_1 = \left[0, \left(\sum_{i=1}^{4} b_{i1} - \sum_{i=1}^{4} c_{i1}\right), \left(\sum_{i=1}^{4} b_{i2} - \sum_{i=1}^{4} c_{i2}\right), \right. \\ \qquad \left. \left(\sum_{i=1}^{4} b_{i3} - \sum_{i=1}^{4} c_{i3}\right), \sum_{i=1}^{4} b_{i4}, -\sum_{i=1}^{4} c_{i4}\right] \\ \bar{c}_2 = \left[\left(\sum_{i=1}^{4} b_{i1} - \sum_{i=1}^{4} c_{i1}\right), 0, \left(\sum_{i=1}^{4} b_{i2} - \sum_{i=1}^{4} c_{i2}\right), \right. \\ \qquad \left. \left(\sum_{i=1}^{4} b_{i3} - \sum_{i=1}^{4} c_{i3}\right), \sum_{i=1}^{4} b_{i4}, -\sum_{i=1}^{4} c_{i4}\right] \\ \bar{c}_3 = \left[\left(\sum_{i=1}^{4} b_{i1} - \sum_{i=1}^{4} c_{i1}\right), \left(\sum_{i=1}^{4} b_{i2} - \sum_{i=1}^{4} c_{i2}\right), 0, \right. \\ \qquad \left. \left(\sum_{i=1}^{4} b_{i3} - \sum_{i=1}^{4} c_{i3}\right), \sum_{i=1}^{4} b_{i4}, -\sum_{i=1}^{4} c_{i4}\right] \\ \bar{c}_4 = \left[\left(\sum_{i=1}^{4} b_{i1} - \sum_{i=1}^{4} c_{i1}\right), \left(\sum_{i=1}^{4} b_{i2} - \sum_{i=1}^{4} c_{i2}\right), \right. \\ \qquad \left. \left(\sum_{i=1}^{4} b_{i3} - \sum_{i=1}^{4} c_{i3}\right), 0, \sum_{i=1}^{4} b_{i4}, -\sum_{i=1}^{4} c_{i4}\right]. \end{cases}$$

Then, the error detection vector can be expressed as $\bar{e} = C\bar{z}^T$, in which $\bar{e} = [e_1, e_2, e_3, e_4]^T$, and $\bar{z} = [z_1, z_2, z_3, z_4, z_5, z_6]$. Since the coding matrix A is known, matrix C can be determined in advance and stored for error detection. The complexity of the error detection and correction relies on the design of A. For a practical implementation, the first four filter outputs $(z_1[n], z_2[n], z_3[n], z_4[n])$ should be equal to the final filters outputs $(y_1[n], y_2[n], y_3[n], y_4[n])$ in order to avoid additional computations to reconstruct those outputs. This greatly simplifies the implementation. In order to illustrate the use of the proposed coding scheme in a practical application, the next section presents two case studies that are then evaluated both in terms of implementation cost and protection effectiveness.

IV. Practical Implementations

To illustrate the use of the proposed scheme, two case studies that consider the protection of four and eight parallel filters are presented here. In both cases, a coding matrix that preserves the inputs to the original filters is used. This is illustrated in Fig. 3 for the case of four parallel filters. The corresponding A matrix is the identity matrix on the first four rows, and only the last two rows have generic coefficients. The matrix is

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ a_{51} & a_{52} & a_{53} & a_{54} \\ a_{61} & a_{62} & a_{63} & a_{64} \end{pmatrix}. \tag{14}$$

To simplify the implementation, those rows should have values that minimize the complexity of multiplications and the increase in the dynamic range in the redundant filters. To that end, the following values can be selected for the last two rows: $[a_{51}\, a_{52}\, a_{53}\, a_{54}] = [1\,1\,1\,1]$ and $[a_{61}\, a_{62}\, a_{63}\, a_{64}] = [1\,2\,3\,4]$.

This produces the following check matrix:

$$C = \begin{bmatrix} \bar{c}_1 \\ \bar{c}_2 \\ \bar{c}_3 \\ \bar{c}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 2 & 3 & 1 & -1 \\ -\frac{1}{2} & 0 & \frac{1}{2} & 1 & 1 & -\frac{1}{2} \\ -\frac{2}{3} & -\frac{1}{3} & 0 & \frac{1}{3} & 1 & -\frac{1}{3} \\ -\frac{3}{4} & -\frac{1}{2} & -\frac{1}{4} & 0 & 1 & -\frac{1}{4} \end{bmatrix} \tag{15}$$

and error detection vector

$$\bar{e} = C\bar{z}^T = \begin{bmatrix} z_5 - z_6 + z_2 + 2z_3 + 3z_4 \\ \frac{1}{2}(2z_5 - z_6 - z_1 + z_3 + 2z_4) \\ \frac{1}{3}(3z_5 - z_6 - 2z_1 - z_2 + z_4) \\ \frac{1}{4}(4z_5 - z_6 - 3z_1 - 2z_2 - z_3) \end{bmatrix}. \tag{16}$$

By defining

$$\begin{cases} p_1 = z_5 - (z_1 + z_2 + z_3 + z_4) \\ p_2 = z_6 - (z_1 + 2z_2 + 3z_3 + 4z_4) \end{cases} \tag{17}$$

the check vector can be expressed as

$$\bar{e} = C\bar{z}^T = \begin{bmatrix} (p_1 - p_2) \\ \frac{1}{2}(2p_1 - p_2) \\ \frac{1}{3}(3p_1 - p_2) \\ \frac{1}{4}(4p_1 - p_2) \end{bmatrix}. \tag{18}$$

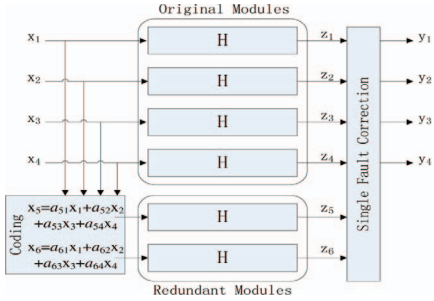

Fig. 3. Practical coding scheme to protect four parallel filters.

This simplified checking can be also derived by noting that p_1 and p_2 simply check if the output values of the redundant filters (z_5 and z_6) match the value reconstructed using the outputs of the original filters (z_1, z_2, z_3, and z_4). Therefore, in the absence of errors, both p_1 and p_2 will be zero. From the coding matrix, for nonzero p_1 and p_2, it becomes clear that an error on the first filter will make $p_1 = p_2$ as both a_{51} and a_{61} are one. An error on the second filter will make $2^* p_1 = p_2$ as $a_{52} = 1$ and $a_{62} = 2$ and so on. Therefore, the vector given by (18) can be used to identify the filter in error using the mapping shown in Table II. For four nonzero values, the faulty filter between the fifth and sixth filters can be identified by checking whether p_1 or p_2 is nonzero. In fact, to check against Table II, the following simplified vector with lower complexity can be used:

$$\bar{e}_{\text{sim}} = \begin{bmatrix} p_1 - p_2 \\ 2p_1 - p_2 \\ 3p_1 - p_2 \\ 4p_1 - p_2 \end{bmatrix}. \tag{19}$$

This provides a simple implementation as only three multiplications are needed and two of them are by powers of two and only require a shift. Another three multiplications are needed to compute p_2. Thus, in total, the scheme requires only six multiplications. This shows that the error location logic can be efficiently implemented. Finally, when an error is detected, it can be corrected by recomputing the affected filter output using z_5 and the remaining original filter outputs. For example, for an error in filter 1, correction is implemented as

$$z_1^{\text{corrected}} = z_5 - (z_2 + z_3 + z_4). \tag{20}$$

The second case study is similar, but eight filters have to be protected. The coding matrix used is shown in (21). The structure is the same as in the first case study and so is the number of redundant filters; as in the proposed scheme, it does not depend on the number of filters. This is a clear advantage over the previous ECC scheme in which the number of redundant filters grows as the number of filters to protect increases. Following the same procedure for the calculation of the detection matrix C, the form of the check vector to detect and locate errors [see (21)] is similar to that of the first case study, but in this case, seven multiplications are needed to compute the check vector and another seven to compute z_{10}. It can be seen that the same matrix structure and correction procedure can be applied to protect any given number of parallel filters. The scheme can detect and correct all errors that affect a single filter. This is

the same error correction capability as that of the previous ECC scheme in [13]. Equation (21) is as follows:

$$A=\begin{bmatrix}1&0&0&0&0&0&0&0\\0&1&0&0&0&0&0&0\\0&0&1&0&0&0&0&0\\0&0&0&1&0&0&0&0\\0&0&0&0&1&0&0&0\\0&0&0&0&0&1&0&0\\0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&1\\1&1&1&1&1&1&1&1\\1&2&3&4&5&6&7&8\end{bmatrix} \quad \bar{e}_{\text{sim}}=\begin{bmatrix}p_1-p_2\\2p_1-p_2\\3p_1-p_2\\4p_1-p_2\\5p_1-p_2\\6p_1-p_2\\7p_1-p_2\\8p_1-p_2\end{bmatrix} \tag{21}$$

in which

$$\begin{cases}p_1=z_9-(z_1+z_2+z_3+z_4+z_5+z_6+z_7+z_8)\\p_2=z_{10}-(z_1+2z_2+3z_3+4z_4+5z_5+6z_6+7z_7+8z_8).\end{cases} \tag{22}$$

To summarize, the computational complexity needed to implement error detection and correction in the proposed method is mainly due to the two redundant filter modules. The other operations needed to compute the check vector and correct the errors are simpler, although slightly more complex than those in the ECC-based scheme presented in [13].

V. Evaluation

The case studies presented in the previous section have been implemented in HDL and mapped to a Xilinx Virtex 4 FPGA XC4VLX80. The added logic for coding and error correction is protected with TMR to ensure that errors do not affect the corrected filters outputs. The parallel filters are FIR filters with 16 coefficients. The input data and coefficients are quantized with 8 bits. The filter output is quantized with 18 bits. To avoid saturation, the inputs for the first and second redundant filters are expanded to 10 and 12 bits for the first case study and to 11 and 14 bits for the second. In addition to the proposed scheme, the ECC-based scheme presented in [13] has been also implemented to assess the cost benefits of the new technique.

The results in terms of resource usage are summarized on Tables III and IV. It can be observed that, for both case studies, the proposed scheme reduces the implementation cost for all resource types compared with the ECC-based scheme. That scheme needs three and four redundant filters in each case as opposed to only two in the new scheme. This explains the observed reductions. Ideally, if the error detection part is not considered, the savings should be approximately 1/7 (or 14.3%) in the first case study and 2/12 (or 16.6%) in the second. However, the actual saving in slices and lookup tables is a little bit lower. This is because the error detection/correction logic is more complex in the new scheme than in [13] and the complexity of the redundant filters is slightly higher than that in the ECC scheme due to the wider inputs.

Fault injection experiments have been conducted to assess the effectiveness of the scheme to correct errors. In the experiments, 10 000 single-event upsets have been randomly inserted in the coefficients and inputs of each filter, respectively; and a tolerance level of 1 is used for the checks in (19). Results show that all faults that introduce errors out of the range $[-1\ 1]$ can be detected and corrected. This confirms the effectiveness of the scheme to correct single errors, which is similar to the protection provided by the previous ECC scheme.

TABLE III
Resource Comparison for Four Parallel FIR Filters

	Unprotected	ECC based [13]	Proposed	Saving
Slices	2944	6611	5792	12.4%
Flip-flops	1328	3196	2408	24.7%
LUTs	5692	12401	11171	9.9%

TABLE IV
Resource Comparison for Eight Parallel FIR Filters

	Unprotected	ECC based [13]	Proposed	Saving
Slices	5888	11589	9872	14.8%
Flip-flops	2656	5286	3994	24.4%
LUTs	11384	21872	19136	12.5%

VI. Conclusion

A new method to implement fault-tolerant parallel filters has been presented in this brief. The proposed scheme exploits the linearity of filters to implement an error correction mechanism. In particular, two redundant filters whose inputs are linear combinations of the original filter inputs are used to detect and locate the errors. The coding of those linear combinations was formulated as a general problem to then show how it can efficiently be implemented. The practical implementation was illustrated with two case studies that were evaluated for an FPGA implementation and compared with a previously proposed technique. That technique relies on the use of ECCs such that each filter is treated as a bit in the ECC. The results show that the proposed scheme outperforms the ECC technique (lower costs achieving similar fault-tolerant capability). Therefore, the proposed technique can be useful to implement fault-tolerant parallel filters. Future work will consider applying the scheme to parallel filters that have the same input signal but different impulse responses.

References

[1] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*, Englewood Cliffs, N.J., USA: Prentice Hall, 1993.

[2] A. Sibille, C. Oestges and A. Zanella, *MIMO: From Theory to Implementation*, New York, NY, USA: Academic, 2010.

[3] N. Kanekawa, E. H. Ibe, T. Suga and Y. Uematsu, *Dependability in Electronic Systems: Mitigation of Hardware Failures, Soft Errors, and Electro-Magnetic Disturbances*, New York, NY, USA: Springer Verlag, 2010.

[4] M. Nicolaidis, "Design for soft error mitigation," *IEEE Trans. Device Mater. Rel.*, vol. 5, no. 3, pp. 405–418, Sep. 2005.

[5] C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," *IBM J. Res. Develop.*, vol. 28, no. 2, pp. 124–134, Mar. 1984.

[6] A. Reddy and P. Banarjee "Algorithm-based fault detection for signal processing applications," *IEEE Trans. Comput.*, vol. 39, no. 10, pp. 1304–1308, Oct. 1990.

[7] S. Pontarelli, G. C. Cardarilli, M. Re, and A. Salsano, "Totally fault tolerant RNS based FIR filters," in *Proc. IEEE IOLTS*, 2008, pp. 192–194.

[8] Z. Gao, W. Yang, X. Chen, M. Zhao and J. Wang, "Fault missing rate analysis of the arithmetic residue codes based fault-tolerant FIR filter design," in *Proc. IEEE IOLTS*, 2012, pp. 130–133.

[9] B. Shim and N. Shanbhag, "Energy-efficient soft error-tolerant digital signal processing," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 14, no. 4, pp. 336–348, Apr. 2006.

[10] Y.-H. Huang, "High-efficiency soft-error-tolerant digital signal processing using fine-grain subword-detection processing," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 18, no 2, pp. 291–304, Feb. 2010.

[11] P. Reviriego, C. J. Bleakley, and J. A. Maestro, "Structural DMR: A technique for implementation of soft-error-tolerant FIR filters," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 58, no. 8, pp. 512–516, Aug. 2011.

[12] P. Reviriego, S. Pontarelli, C. Bleakley and J. A. Maestro, "Area efficient concurrent error detection and correction for parallel filters," *IET Electron. Lett.*, vol. 48, no 20, pp. 1258–1260, Sep. 2012.

[13] Z. Gao *et al.*, "Fault tolerant parallel filters based on error correction codes," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 23, no. 2, pp. 384–387, Feb. 2015.

[14] R. W. Hamming, "Error correcting and error detecting codes," *Bell Sys. Tech. J.*, vol. 29, pp. 147–160, Apr. 1950.

[15] A. Chatterjee, and M. A. d'Abreu, "The design of fault-tolerant linear digital state variable systems: Theory and techniques," *IEEE Trans. Comput.*, vol. 42, no. 7, pp. 794–808, Jul. 1993.

[16] C. N. Hadjicostis, "Coding Approaches to Fault Tolerance in Dynamic Systems," Ph.D. dissertation, MIT Press, Cambridge, MA, USA, 1999.