International Journal of Gender, Science and Technology
ISSN: 2040-0748
UGC Care Group I Journal.
Vol-12 Issue-02 July 2023

https://ijgst.com.2023.v12.i2.pp100-106

FAULT TOLERANT PARALLEL FILTERS USING COMMUNICATION APPLICATIONS

Dr. S.V NARESH[1] K.TIRUPATHAIAH[2] G SIVASANKAR[3] CHEJARLA RAGHUNATHA BABU[4]
[1]PROFESSOR, DEPARTMENT OF ECE,
[2] ASST. PROFESSOR, DEPARTMENT OF ECE,
[3]ASST. PROFESSOR, DEPARTMENT OF ECE,
[4] ASST. PROFESSOR, DEPARTMENT OF ECE,
[1,2,3,4] SRI MITTAPALLI COLLEGE OF ENGINEERING

ABSTRACT -

As the complexity of communications 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 lters. 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 liters. In that scheme, each lteristreatedasa bit, and redundant letters 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 liters is addressed in a more general way. In Particular, it is shown that the fact that lter inputs and outputs are not bits but numbers enables more efficient protection. This reduces the protection overhead and makes the number of redundant liters independent of the number of parallel liters. The proposed scheme is rst described and then illustrated with two case studies. Finally, both the effectiveness of protecting against errors and the cost are evaluated for a eld-programmable gate array implementation.

1. INTRODUCTION

Filters are often used in electronic systems to emphasize signals in certain frequency ranges and reject signals in other frequency

ranges. In circuit theory, a filter is an electrical network that alters the amplitude and/or phase characteristics of a signal with respect to frequency. Ideally, a filter will not add new frequencies to the input signal, nor will it change the component frequencies of that signal, but it will change the relative amplitudes of the various frequency components and/or their phase relationships. Today filters are widely used in a number of applications which are based on automotive, medical, and space where reliability of components in digital electronic circuits is critical. Filters of some sort are essential in the operation of most electronic circuits. There are many different bases of classifying filters and these overlap in many different ways; there is no simple hierarchical classification. As the behavioral properties of signal change, the techniques of filtering it will be differ. Being specific with filters, the digital filters have vast applications in digital signal processing. Filtering is also a class of signal processing, the defining feature of filters being the complete or partial suppression of some aspect of the signal. It is therefore in the interest of anyone involved in electronic circuit design to have the ability to develop filter circuits capable of meeting a given set of specifications. In signal processing, a digital filter is a device or process that removes some unwanted component or feature from a signal. Digital filters are used for two general purposes; separation of signals that have been combined, and restoration of signals that have been distorted in some way. Most often, this means removing some

frequencies and not others in order to suppress interfering signals and reduce background noise. Digital filters are a very important part of DSP. In fact, their extraordinary performance is one of the key reasons that DSP has become so popular. As the applications of digital circuits in signal processing reach their peak, possibilities of faults and its detection and correction within digital circuitry may also need to be advanced. However, filters do not exclusively act in the frequency domain; especially in the field of signal processing many other targets for filtering exist.

Correlations can be removed for certain frequency components and not for others without having to act in the frequency

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

domain. It is common in DSP to say that a filter input and output signals are in the time domain. This is because signals are usually created by sampling at regular intervals of time.

PROPOSED SYSTEM:

The proposed scheme is illustrated in the given following picture. The input

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

signals are encoded using a matrix with arbitary 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 above. Therefore, the input signal to the Ith filter is of the form (i=1,2,…., or 6), i.e., vi[n]=ai1·x1[n]+ai2·x2[n]+ai3·x3[n]+ai4·x4 [n]. 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., y1[n],y2[n], y3[n], and y4[n], can be obtained as follows: Where A1235 is a sub matrix of A, including the first, second, third, and fifth rows. This process can be repeated with different sub matrices of A, for

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

example, with A1236, A2345, and A2346. In the error-free case, all the recovered versions of y1[n],y2[n], y3[n], and y4[n] will be the same.

When there are differences, an error is detected Which means that there is an error among filters {12356} 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 y1[n]+ y2[n]+y3[n]+y4[n]are equal is enough. In more detail, four checks are needed, each involving five filters and excluding one. For example if one branch is excluded, the error checking would be in

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

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

which W2345= [1111](A2345)-1, and w2346= [1111](A2346)-1. Similarly, if filters 2,3,4 are excluded, tele2, e3, and e4 error signals can be generated as follows.

The error correction and detection logic can be simplified assuming that there is only a single error. In the values are equal to the that case checking only a single error for each recovered set, the sums of the values each involving five filters and excluding one. For example, if the batches can be generated as signals as follows. Then, with the error vector

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

Then

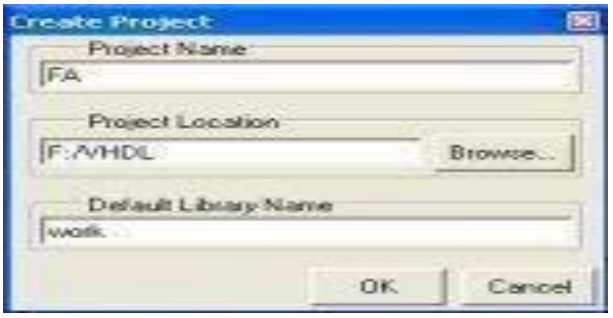
$$\begin{cases} \bar{w}_{2345}=[1111](A_{2345})^{-1}=\left[\sum_{i=1}^{4} b_{i1}, \sum_{i=1}^{4} b_{i2}, \sum_{i=1}^{4} b_{i3}, \sum_{i=1}^{4} b_{i4}\right] \\ \bar{w}_{2346}=[1111](A_{2346})^{-1}=\left[\sum_{i=1}^{4} c_{i1}, \sum_{i=1}^{4} c_{i2}, \sum_{i=1}^{4} c_{i3}, \sum_{i=1}^{4} c_{i4}\right]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

and finally

$$\begin{cases} s_1^1=\bar{w}_{2345}(z_2z_3z_4z_5)^T=z_2\sum_{i=1}^{4} b_{i1}+z_3\sum_{i=1}^{4} b_{i2} \\ \qquad +z_4\sum_{i=1}^{4} b_{i3}+z_5\sum_{i=1}^{4} b_{i4} \\ s_1^2=\bar{w}_{2346}(z_2z_3z_4z_6)^T=z_2\sum_{i=1}^{4} c_{i1}+z_3\sum_{i=1}^{4} c_{i2} \\ \qquad +z_4\sum_{i=1}^{4} c_{i3}+z_6\sum_{i=1}^{4} c_{i4} \\ e_1=z_2\left(\sum_{i=1}^{4} b_{i1}-\sum_{i=1}^{4} c_{i1}\right)+z_3\left(\sum_{i=1}^{4} b_{i2}-\sum_{i=1}^{4} c_{i2}\right) \\ \quad +z_4\left(\sum_{i=1}^{4} b_{i3}-\sum_{i=1}^{4} c_{i3}\right)+z_5\sum_{i=1}^{4} b_{i4}-z_6\sum_{i=1}^{4} c_{i4}. \end{cases}$$

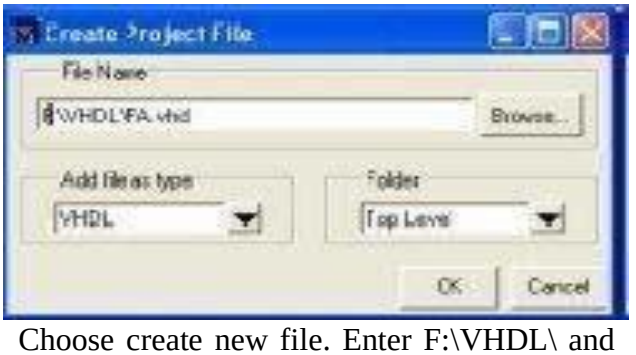
(12)

e= [e1, e2, e3, e4], the faulty filter can be located. The process is listed in table 2, in which # stands for the values, the filter is identified. The outputs can be corrected by using the remaining pass code matrix to fulfill some conditions. Let us consider the calculation of 1 and 2 define (A2345)-1 and (A2346)-1 as follows. Therefore, the error check will ensure error detection when the sums of the columns in each of the matrixes A-12345 and A-12346 are different and nonzero. Furthermore, we can define the error detection matrix as c= [c1, c2, c3, c4] ^{T}. on the left side of the error will ensure that detection when the sums of the columns in each of the matrix as given the following transpose values. On the left side of the interface, under the project tab is the frame listing of the flies that pertain to the opened project. The library frame lists the entities of the project to the right is the model sim shell frame. It is an extension of MS-DOS, so both model sim and MS-DOS commands can be executed. Once model sim has been started, create a new project:



File>NEW>PROJECT….

Name the project FA, set the project location to F:/VHDL and click OK. A new window should appear to add new files to the project.



Choose create new file. Enter F:\VHDL\ and click ok. Then close the “add new files”, window. Additional files can be added latter

by choosing from the menu: project > add file to the project.

EDITING SOURCE FILES:

Double click the FA. VHDL source found under the work space window's project lab. This will open up an empty text editor configured to highlight VHDL syntax. Copy the source found at the end of this document. After watching the code for the entity and its architecture save and close the source file. Hence the last files are forbidded to look upon the selected files with out any data .

COMPILING PROJECTS:

select the file from the project files list frame and right click on it. Select compile to just compile. This file or compile to adjust the files in the current project. If there are errors within the current project. If there are errors with in the code or the project, a red failure message will be displayed. Double click these red errors for more. Otherwise, if all is well then no red warning or error messages will be displayed. A new window will be displayed listing the design entity's signals and their initial value in these items. In waveforms and listing are ordered in which they are declared in the code. To display the waveform, select the signals for the waveform to display(hold CTRL and click to select multiple signals) and from the signal list window menu select. In this case, it is possible use to verilog write a test bench to verify the functionalty of the design using files on the host computer. A verilog model is translated into the gate and wires that are mapped onto a programmable logic device such as CPLD or FPGA based then it is the actual hardware being configured, rather than the VERILOG was to write a test bench to verify the host computer. It is possible to use VERILOG to write a test bench using files on the host computer ti

define stimuli and to interact with the user and to compare results with those expected. FPGA stands for field programmable gate array which write a test bench to verify the functionality of the design using files on theresults to be expected. FPGA can be configured by the end user to implement specific circuitry. Speed is up to 100MHZ but at present speed is GHZ. Main applications are DSP, FPGA based computers, logic emultion, ASIC and ASSP. FPGA can be programmed mainly on SRAM(static random acess memory). It is volatile and the main avantage of using SRAM programming technology is reconfigurability. Issues in FPGA technology are complexity of logic element, clock support and interconnections routing.

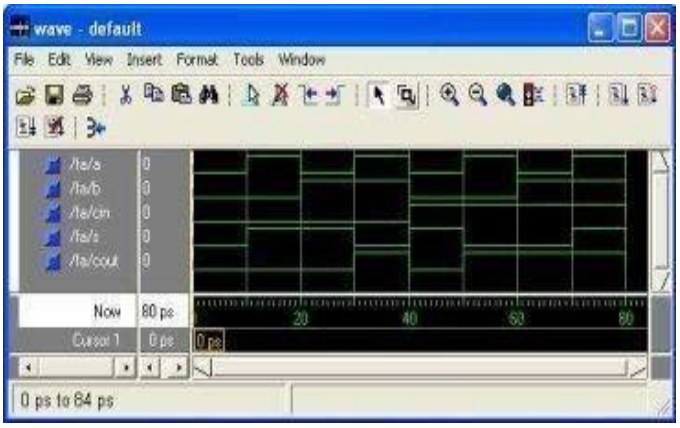
SIMULATING WITH MODELSIM:

To simulate, first the entity design has to be loaded into the simulator. Do this by selecting from the menu: **simulate > simulate** a new window will appear listing all the entities that are in the work library. Select FA entity for simulation and click **OK.** Oftentimes it will be necessary to create entities with multiple architectures. In this case the architecture has to be specified for the simulation. Expand the tree for the entity and select the architecture to be simulated and then click ok. Creating test files for the simulator after the design is loaded, clear up any previous data and restart the timer by typing in the prompt **VIEW> SIGNALS**.

A new window will be displayed listing the

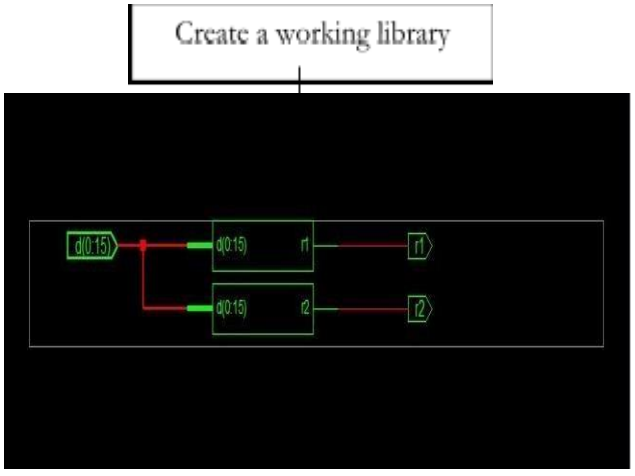
design entity's signals and their initial value. Items in waveform and listing are ordered in the same order in which they are declared in the code. To display the waveform, select the signals for the waveform to display (hold ctrl and click to select multiple signals) and from the ssignal list window menu script.

Add > wave> selected signals



BASIC SIMULATION FLOW:

The following diagrams shows the basic steps for simulating a design in MODELSIM. Hence the library files are modified to the simulation flow.



RESULTS & SCHEMATICS:

The follwing schematic explains the functional inputs of the circuit given by the elegancy values that has been recoverd from the project library with all the delegant details.

The technological values of the screen would be negligible with the new values to be implemented fault tolerant parallel filters has been presented in this brief. The proposed

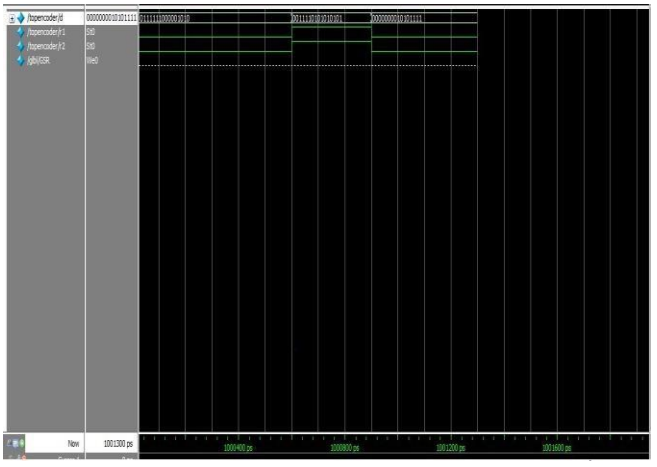
Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	6	4656	0%
Number of 4 input LUTs	10	9312	0%
Number of bonded IOBs	18	232	7%

scheme exploits the brief values of the methods. In particular two redundant filters whose inputs are linear used to detect and locate the errors. The coding of those linear combinations was formulated as a general problem to them how it can efficiently understands the data.

DEALY:

```
Data Path: d<12> to r1
                                   Gate     Net
Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
----------------------------------------  ------------
 IBUF:I->O             2   1.106   0.532  d_12_IBUF (d_12_IBUF)
 LUT4:I0->O            1   0.612   0.509  r11/x4/Mxor_c_xo<2>1 (r11/c<3>)
 LUT4:I0->O            1   0.612   0.509  r11/x5/Mxor_c_xo<0> (r11/x5/Mxor_c_xo<0>)
 LUT4:I0->O            1   0.612   0.357  r11/x5/Mxor_c_xo<3> (r1_OBUF)
 OBUF:I->O                 3.169          r1_OBUF (r1)
----------------------------------------
Total                      8.018ns (6.111ns logic, 1.907ns route)
                                   (76.2% logic, 23.8% route)
```

SIMULATION:



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 ElectroMagnetic 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.

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.