IMPLEMENTATION OF MEDIAN DATA SORTING USING MULTIPLEXER LOGIC

D.Shairabee, Sri.T.Vijay Kumar
M.Tech Student, Associate Professor
Department Of ECE
Dr.K.V.S.R. Institute Of Technology, Kurnool

Abstract—A Novel Area efficient and low power multiplexer based Data comparator for median filter in De-noising application is proposed. The proposed method uses multiplexer based implementation of borrow equation in a full subtractor which acts as a basic processing element of a Data Comparator. The proposed work was implemented in Micro wind for three different Models of Mosfet and different technologies. The modifications in the existing borrow equation of a full subtractor using multiplexer only resulted in reduced number of transistors with reduced power.

In digital image processing systems, the acquisition stage may capture impulsive noise along with the image. This physical phenomenon is commonly referred to as "salt-and pepper" noise. The median filter is a nonlinear image processing operation used to remove this impulsive noise from images. This digital filter can be implemented in hardware to speed up the algorithm. However, an SRAM-based FPGA implementation of this filter is then susceptible to configuration memory bit flips induced by single event upsets (SEUs), so a protection technique is needed for critical applications in which the proper filter operation must be ensured. In this paper, a fault-tolerant implementation of the median filter is presented and studied in depth. Our protection technique checks if the median output is within a dynamic range created with the remaining non-median outputs. An output error signal is activated if a corrupted image pixel is detected, then, a partial or complete reconfiguration can be performed to remove the configuration memory error.

I.INTRODUCTION:

Harsh environments, like space, are a challenge for electronic circuits in general and for memories in particular. For example, radiation causes several types of errors that can disrupt the circuit functionality [1]. One common error for SRAM memories is soft errors that change the value of one or more memory cells [2].To avoid corruption in the data stored in the memory, error correction codes (ECCs) are commonly used [3].ECCs add parity check bits to each memory word to detect and correct errors. This requires an encoder to compute those bits when writing to the memory and a decoder to detect and correct errors when reading from the memory. These elements increase the memory area and the power consumption, and can also reduce the access speed. These overheads increase with the error correction capability of the ECC. Traditionally, codes that can correct a single bit error per word have been used. In particular, single error correction–double error detection (SEC–DED) codes that can also detect double errors are commonly used [4]. In recent years, the number of errors that affect more than one memory cell has increased significantly. This is due to the scaling of the memory cells and is projected to grow further [5]. These errors, known as multiple cell upsets (MCUs), pose a challenge for SEC–DED codes. One solution to ensure that the MCU errors can be corrected is to interleave the bits of different logical words so that an MCU affects one bit per word [6]. This is based on the observation that the cells affected by an MCU are physically close [7]. Interleaving, however, has a cost as it complicates the memory design [8]. In some space applications, there is an additional issue as the number of errors is high, and SEC–DED codes may not be sufficient when errors accumulate over time [9]. These issues have led to an increased interest on the use of more advanced ECCs to protect SRAM memories. As MCUs affect cells that are close together, a number of codes that can correct double-adjacent or triple-adjacent errors have been recently proposed [8], [10]– [12]. These codes, in many cases, do not require additional parity check bits and I the rest require only one or two additional bits. The decoding complexity increases but in many cases can still be implemented with limited impact on the memory speed. These codes are useful for applications in which the error rate is low, however, when the error rate is large, codes that can correct errors on multiple independent bits are needed [9]. Research for multibit ECCs has focused on reducing the decoding latency as in many cases, the traditional decoders are serial and require several clock cycles. To some extent this can be done for some traditional ECCs by using a parallel syndrome decoder [13] but the decoder complexity explodes as the error correction capability or the word size increases. Another approach is to use codes that can be decoded with low delay, such as orthogonal Latin squares (OLSs)or difference set (DS) codes [14], [15]. In the case of OLS codes, the

main issue is that they are not optimal in terms of the number of parity check bits and thus require more memory overhead. The DS codes are more competitive in terms of parity check bits but are still not optimal for some word lengths. For example, the *(21, 10)* DS code can correct 2-bit errors while a code with a similar block size and code rate, and the (24,12) extended Golay code [16] can correct 3-bit errors. However, the Golay code requires a more complex decoder that needs several clock cycles [17]. Namba *et al.* [18] have proposed a compromise solution for Bose–Chaudhuri–Hocquenghem codes. The idea is that the most common error patterns are decoded in parallel and the rest serially. In particular, single and double-adjacent errors are corrected in a single clock cycle. This means that the most memory accesses can be completed in a single clock cycle, and only a small percentage of the words in error require a full serial decoding. This can enable the use of traditional ECCs that do not support fast parallel decoding to protect SRAM memories In this brief, the use of the scheme in [18] is considered for the(24,12) Golay code. In more detail, an efficient parallel decoder capable of correcting the single and double-adjacent errors is presented. The decoder exploits the properties of the Golay code to reduce the implementation cost. This results in a decoder that is simpler than a traditional SEC decoder but that can also correct all double-adjacent errors and some triple-adjacent errors. The proposed decoder has been implemented in hardware description language and mapped to a 65-nm technology to show its benefits. The main contribution of this brief is to enable a fast and efficient parallel correction of the single and double-adjacent errors in the (24,12) Golay code.

$$\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&0&0&0&1&0&1\\
0&1&0&0&0&0&0&0&0&0&0&0&0&1&1&0&1&1&1&0&0&0&1&1\\
0&0&1&0&0&0&0&0&0&0&0&0&1&0&1&1&0&1&1&1&0&0&0&1\\
0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&1&1&0&1&1&1&0&0&1\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&1&1&0&1&1&1&0&1\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&1&1&0&1&1&1&1\\
0&0&0&0&0&0&1&0&0&0&0&0&1&0&0&0&1&0&1&1&0&1&1&1\\
0&0&0&0&0&0&0&1&0&0&0&0&1&1&0&0&0&1&0&1&1&0&1&1\\
0&0&0&0&0&0&0&0&1&0&0&0&1&1&1&0&0&0&1&0&1&1&0&1\\
0&0&0&0&0&0&0&0&0&1&0&0&0&1&1&1&0&0&0&1&0&1&1&1\\
0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&1&1&0&0&0&1&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&1&1&0
\end{bmatrix}$$

Fig. 1. Parity check matrix of the (24,12) Golay code.

$$\begin{bmatrix}
1&1&0&1&0&0&0&1&0&1&0&1&0&0&0&0&0&0&1&0&0&0&1\\
0&0&1&1&0&1&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&1\\
0&1&0&0&1&1&0&1&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&1\\
0&0&0&1&0&0&1&1&0&1&0&0&0&1&0&1&0&1&0&0&0&0&0&1\\
0&0&0&0&0&1&0&0&1&1&0&1&0&0&0&1&0&1&0&1&0&0&0&1\\
0&0&0&0&0&0&0&1&0&0&1&1&0&1&0&0&0&1&0&1&0&1&0&1\\
0&1&0&0&0&0&0&0&0&1&0&0&1&1&0&1&0&0&0&1&0&1&0&1\\
0&1&0&1&0&0&0&0&0&0&0&1&0&0&1&1&0&1&0&0&0&1&0&1\\
0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&0&1&1&0&1&0&0&0&1\\
0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&0&1&1&0&1&0&1\\
0&1&0&0&0&1&0&1&0&1&0&0&0&0&0&0&0&1&0&0&1&1&0&1\\
0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1&1&0
\end{bmatrix}$$

Fig. 2. Parity check matrix of the (24,12) Golay code with the proposed bit placement.

II.PROPOSED SYSTEM

MEDIAN filtering is a basic filter used to eliminate salt and pepper noise in images and videos. The Median filter uses data comparator as its basic processing element to compute median. Different Median filter architectures were formulated over the years. A need for a suitable VLSI architecture was needed for hardware implementation. Few of the Yester year works are listed below. Sindhu & Vasanth [1] developed a new set of 8 bit Data comparator which could be a viable solution for Rank ordering networks in terms of area, speed and power. Vasanth et al proposed Vasanth sorting with minimum number of comparators for de-noising the image. A 3*3 window is used to arrange data with 25 comparisons in increasing order. For this sorting new parallel architecture is implemented with 8-bit data comparator. Bitwise data comparator to choose high and low values. The advantage of using these comparators is that it operates at high speed and consumes low power [2]. Different styles of full adder comparator is designed and compared in terms of area, speed and power with the help of DSCH and Micro wind tool in 120nm and 70nm technology [3]. Anjuli and Anand used different logic models for 2-bit magnitude. Four logic designs such as CMOS logic style, pseudo NMOS logic style, transmission gate and pass transistor logic (PTL) style were simulated and found that PTL with reduced power, power delay product transistor count [4]. Rangaraju et al proposed an efficient reversible binary comparator using reversible logic with reversible gates like not, Feynman (CNOT) and Peres (PG). The reversible binary comparator involves two stage with former by input circuit and the later with one-bit comparator. Thus a cell with n-bit reversible binary comparator is implemented. PG gate is used in designing binary comparator for computing power consumption delay, Garbage outputs and constant inputs. The main advantage of the work is that it has a less quantum cost and garbage output values [5].

PROPOSED DATA COMPARATOR

An 8 bit Data comparator is a basic processing element in a Median Calculation. The Basic operation in a comparator is subtraction. The Basic processing element in the proposed data comparator is a full subtractor. Fig. 1 gives the basic architecture of a Data Comparator. In this architecture there are two Multiplexers that select a high or low value from the given inputs “a” and “b” based on the carry generated to the selection lines from bitwise comparators.

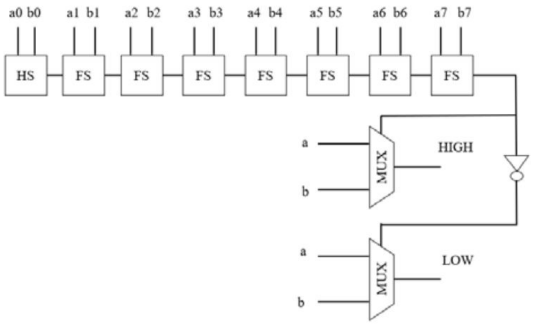


Fig. 1. Basic Architecture of Data Comparator

The Carry generator consists of eight bitwise full subtractor, which propagates carry to select a High and low values from both the input. The initial bit will always have carry to be zero hence a Half subtractor shown in Fig. 2 is considered rather than full subtractor

A. Proposed Multiplexer based Data Comparator The Proposed Multiplexer based Data Comparator consists of two units 1. Borrow Generation unit and 2. Multiplexer unit to select high and low values. a) Borrow Generation Unit Consider the Borrow equation 1 of a full subtractor as the carry equation is used to propagate carry to the multiplexers of Data Comparator architecture given in Fig. 1. Consider 1 bit full Subtractor, where A and B are inputs, Bin is the borrow in and Bout refers to borrow out. From the truth table of Full Subtractor shown in Table I derive the borrow equation as shown in 1 (1) (2) Alternate form of implementation is made from Table I. Consider the upper half values of the truth table where input " a" will be 0 and the inputs b and bin values changes. Similarly the lower half values of the truth table has "a" as 1. It was observed from the table that if the value of "a" takes 0 then the borrow equation produces 0,1,1,1 for different "b" and "bin" respectively. This can be replaced with an OR gate. Similarly if the value of "a" takes 1 then the borrow equation produces 0,0,0,1 for different "b" and "bin" respectively. This can be replaced with an AND gate. From this logic we can deduce the borrow equation can be deduced in the form of a multiplexer equation as given in 3. (3) The above equation is a multiplexer based logic of borrow equation of a full subtractor. The first part of the equation 3 represents the upper half of the truth table and second half of the truth table represents lower half of truth table.

It indicates that when the input "a" is zero the output will resemble OR gate output of inputs "b" and "bin". Similarly when the input "a" is one the output will resemble AND gate output of inputs "b" and "bin". Hence the entire logic will be implemented in 2X1 Multiplexer only.

TABLE I
TRUTH TABLE OF BORROW OUT EQUATION IN A FULL SUBTRACTOR

A	B	Bin	Bout
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

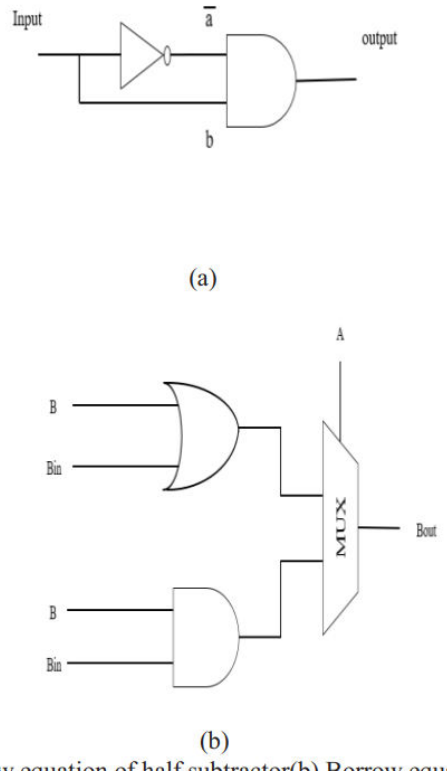


Fig. 2.(a)Borrow equation of half subtractor(b) Borrow equation of full subtractor from equation 3

B. Multiplexer unit to select High and Low values

The output of the Borrow generation unit would be given as input to two multiplexers that gives out the high and low values between the inputs fed to the system. The output of 8- bit full subtractor is given as selection line to custom made multiplexers. The borrow that propagates out of Borrow generation unit is given to upper multiplexer directly and borrow propagated is inverted to give to the selection lines of

the lower multiplexer. Now based on borrow generated the multiplexers compares the given input values and gives a High value and low value as an output. The Implementation is shown in Fig. 3.

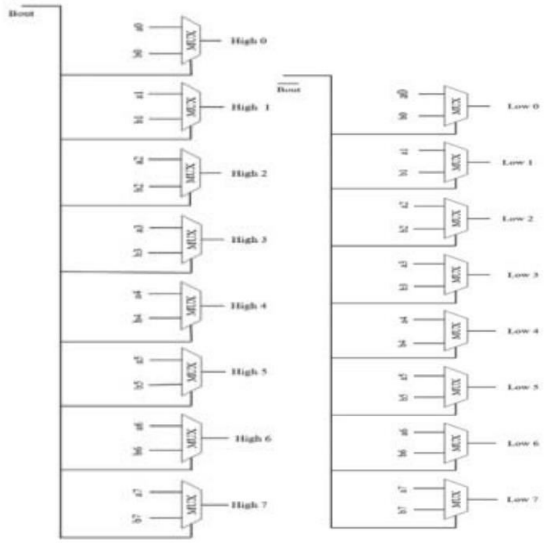


Fig. 3. (a). Multiplier of high value output, (b). Multiplier of low value output.

III.EXTENSION METHOD

DIGITAL image sensors are widely used in space applications, so space radiation can affect the sensor or the image processing system itself. Energetic particles can collide with vulnerable parts in the device (i.e. transistors) leading to, for example, single event upsets (SEUs). Image sensors often acquire undesirable noise along with the captured image frame, so it is important to repair these corrupted pixels to facilitate subsequent image processing operations such as edge detection or object recognition [1]. In order to remove this noise, image filtering and enhancement is usually performed immediately after the image is captured and before any other image processing operation. This approach avoids propagating the input noise, obtaining better resulting images [2]. Different image filtering techniques can be used to remove noise [3], however, they are usually required to perform several mathematical operations on each image pixel to obtain the desired modified version of the input image. This results in a large number of operations per second, which makes field programmable gate arrays (FPGAs) an alternative to the classic implementation of image processing algorithms in microprocessors. In particular, SRAM-based FPGAs provide high performance, high densities, and low cost, while allowing a practically unlimited number of reconfigurations [4]. Moreover, a reprogrammable logic offers the additional benefit of on-the-fly changes, so image processing applications can be evolved to meet more complex requirements. Acquisition noise in digital image processing systems may be composed of impulsive noise such as hot and dead pixels [5]. The term “hot pixel” describes those pixels that are much brighter than surrounding pixels, and the term “dead pixel” refers to those permanently off. When both physical phenomena appear together in the same image they are commonly referred to as “salt-and-pepper” noise, and are mainly caused by charge leakages within the image sensor chip [6]. The first three images in Fig. 1 illustrate the effects of impulsive noise in an image.

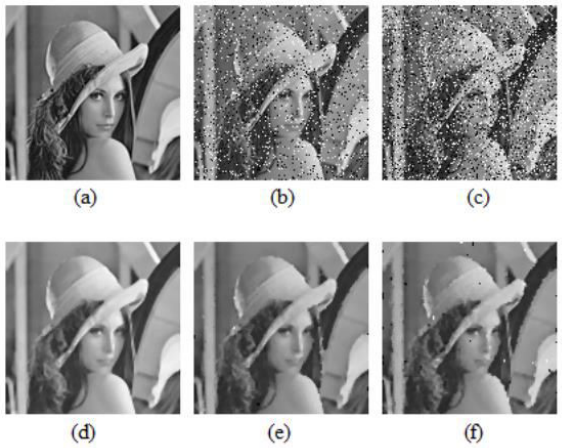


Fig. 1. Effects of impulsive noise in an image and median filter restoration results. (a) Original Lena image, (b) corrupted Lena image with 10% salt and - pepper noise, (c) corrupted Lena image with 20% salt-and-pepper noise, (d) original Lena image filtered, (e) 10% salt-and-pepper noise Lena image restored, and (f) 20% salt-and-pepper noise Lena image restored.

The median filter is a nonlinear image processing operation. In contrast to linear filters, nonlinear filters are preferred against impulsive noise due to their robustness and denoising power [7]. In particular, the median filter can efficiently attenuate the effects of low percentages of this “salt-and-pepper” noise while preserving the edges of the objects existing in the image. The denoising power of the median filter can be observed in Fig. 1 (d), (e) and (f). As mentioned before, SRAM-based FPGAs provide high performance that can be exploited by image processing systems to speed up their algorithms. However, SRAM-based FPGAs are also vulnerable to radiation effects, so an implementation of the median filter in these devices can be susceptible to SEUs in the FPGA configuration memory SRAM cells, changing the median filter

functionalities permanently until the original bitstream is reloaded. An example of a median filter malfunction due to an SEU in the SRAM-based FPGA configuration memory is illustrated in Fig. 2(c).

Fig. 2. (a) Original, (b) median filtered, and (c) bad filtered Lena image [10].

Redundancy-based techniques such as dual modular redundancy (DMR) or reduced precision redundancy (RPR) are typically used to protect digital filters [8, 9]. The main difference between these techniques is that the RPR method combines the main circuit with a low precision replica to detect errors in the most significant bits (MSBs), instead of using a full precision replica as happens in the DMR scheme. This means that least significant bits (LSBs) changes cannot be detected with RPR, but less resource overhead is added. This work is an extension of our paper about a fault tolerant implementation of the median filter presented in [10]. Our technique checks if the median output value is within a dynamic range created each time with the remaining nonmedian outputs. The implemented design is tested with several 8-bit grayscale 128x128 images, and is compared to DMR and RPR schemes. Experimental results show that, although our technique cannot detect as many corrupted pixels as the DMR scheme, it detects enough to prevent 91% of the corrupted images from being erroneously processed by the next image processing operation.

MEDIAN FILTER

The median filter is a nonlinear image processing technique used to remove impulsive noise from images. This spatial filtering operation applies a two-dimensional (2D) window mask to an image region and replaces its original center pixel value with the median value of the pixels contained within the window. The window is then moved to the next image region and the cycle is repeated until the entire image is processed. As a result, the ideal filtered image would have no impulsive noise and perfectly sharp edges. These are intrinsic properties of the median filter that cannot be achieved by traditional linear filtering techniques without resorting to time-consuming data manipulations [11]. However, the median filter is not an ideal filtering operator, and edge preservation worsens as the total percentage of impulsive noise increases.

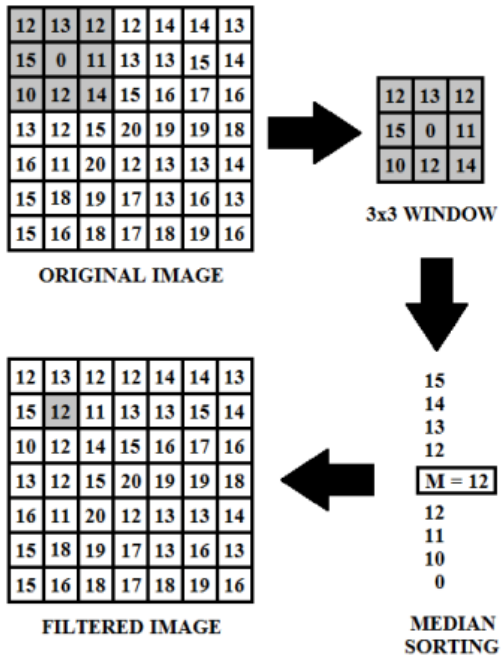


Fig. 3. Median substitution process

In order to process every image pixel, the 3x3 square window should be moved through the image. However, in FPGAs, this can also be achieved using a nine-pixel stream that sequentially passes through the median filter. Each group of nine pixels can be sorted using a structure composed of two-input exchange nodes as the one illustrated in Fig. 4. The exchange node in Fig. 4 performs a two-input sorting using an 8-bit comparator and two 2:1 multiplexers. The two inputs are internally compared and the higher (H) and lower (L) values are obtained. The classic sorting network structure shown in Fig. 5 uses 41 basic nodes. In this figure, each box is an identical exchange node as the one illustrated in Fig. 4. This classic implementation has proven to be far from optimal since improved FPGA implementations have been developed over the years [12]– [15]. Using the Batcher bitonic sorter [16], a 9-input sorting network can be created [17]. This sorting network, composed of 28 basic nodes, significantly reduces the amount of FPGA resources used. However, the architecture proposed by [18] has been followed in this paper since it uses the minimum number of exchange nodes. In this network, the resource usage minimization is achieved because it performs a nine pixel partial sorting, which means that higher and lower output pixel values are unsorted. This is not a

problem since only the median value is needed to replace the center pixel. Fig. 6 illustrates this optimal scheme.

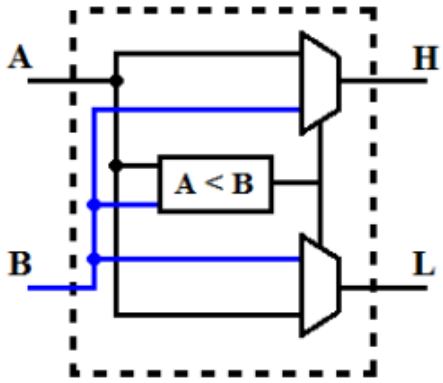


Fig. 4. Internal diagram of one exchange node [10].

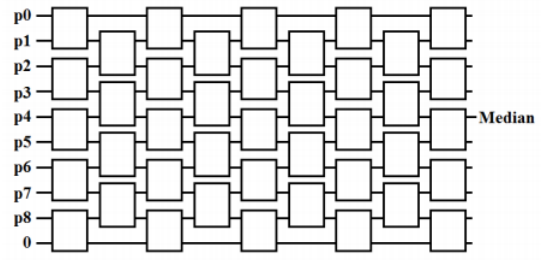


Fig. 5. Classic exchange network scheme for nine input pixels.

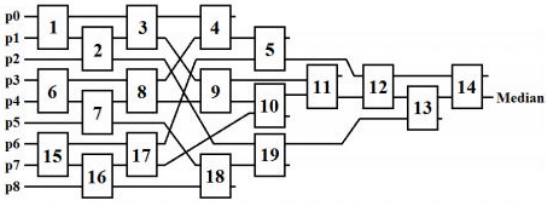


Fig. 6. Minimum exchange network scheme for nine input pixels [10].

PROPOSED TECHNIQUE

Configuration memory errors in SRAM-based FPGAs modify the design functionalities permanently until the original bitstream is reloaded. A redundant scheme can identify and mitigate user register errors. However, in order to remove these configuration errors, it is more practical to detect and then reconfigure partially or completely the faulty design. For this reason, the proposed fault-tolerant technique activates an output error signal if a corrupted image pixel is detected. Then, a partial or complete reconfiguration can be performed to remove the permanent error.

Fig. 7 illustrates the proposed technique. Grey-shaded blocks are added to the original Fig. 6 scheme in order to create a range with the non-median outputs. The range is dynamically determined for each nine pixel group using identical exchange nodes to the one shown in Fig. 4. The upper range (H3 low output) is calculated as the lower value from the four higher input values, and the lower range (L3 high output) is obtained using the higher value from the four lower input pixel values. Once the range is calculated, two 8-bit comparators check if the median output value is within the range, and set their output signals to 0 if the condition is satisfied or 1 otherwise.

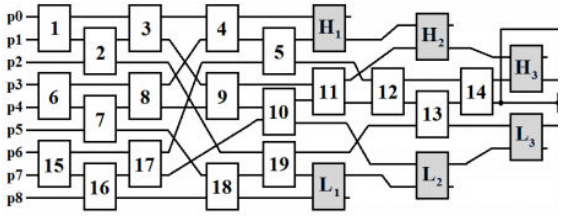


Fig. 7. Proposed technique [10].

For a better understanding of the error detection mechanism a numerical example is presented in Fig. 8, where the median value is delimited by the two closest non-median outputs. As can be observed in this figure, the four highest pixel values (95, 92, 90, and 75) are sorted with the H1, H2, and H3 blocks, obtaining 75 as the upper range. At the same time, the four lowest values (10, 20, 50, and 53) are sorted with L1, L2, and L3, obtaining 53 as the lower range. Thus, if the median is not contained within these values, a malfunction has occurred.

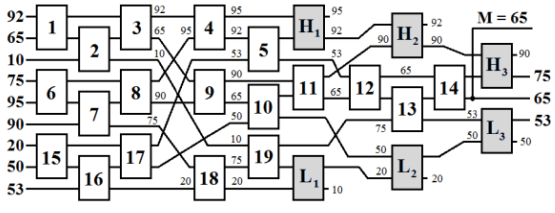


Fig. 8. Numerical example of range values calculation [10].

The median filter is a pure combinational circuit, so SEUs in the SRAM-based FPGA configuration memory can modify the internal routing, changing the behavior of the median sorting network. This may cause a bad median calculation or a non-median input

value modification (i.e. the median value is correctly calculated but other inputs are corrupted during the sortening process). Both errors can be detected by the proposed technique if the median value is out of range, or the calculation of the range itself creates a never-satisfied range condition. Our method does not detect in-range value changes. However, the impact of in-range median value changes is low, as demonstrated in the following section.

IV. MATLAB APPLICATION

The proposed technique has been compared in terms of resource utilization and error detection rate with DMR and RPR schemes. These schemes have been selected since configuration bits errors can only be corrected with a partial or complete FPGA reconfiguration, so it is more appropriate to detect the error and execute the reconfiguration of the device. A DMR system consists of two fully redundant copies of the median filter with a comparator that detects incorrect behaviors. The RPR method uses a replica of the original median filter with reduced precision exchange nodes. The four most significant bits (MSBs) of the 8-bit input pixels are processed and compared to the median output MSBs. Therefore, least significant bits (LSBs) changes cannot be detected, but less resource overhead is added. These two methods and the proposed technique have been implemented in the SRAM-based FPGA part of a Xilinx Zynq7000 All Programmable System on a Chip (SoC) [19]. They have been compared to the original unprotected median filter, obtaining the utilization report presented in Table I.

TABLE I
UTILIZATION REPORT

	Original	RPR	DMR	Proposed
LUTs	286	461	572	385
Overhead	0%	61%	100%	35%

As can be observed in this table, the number of look up tables (LUTs) is, as expected, doubled in the DMR implementation, while the proposed error detection technique only requires 35% of additional logic resources. First, the proper operation of the three studied methods is validated against the standard 8-bit grayscale images shown in Fig. 9. Then, the uncorrupted median filtered (golden) images are stored for subsequent comparisons.

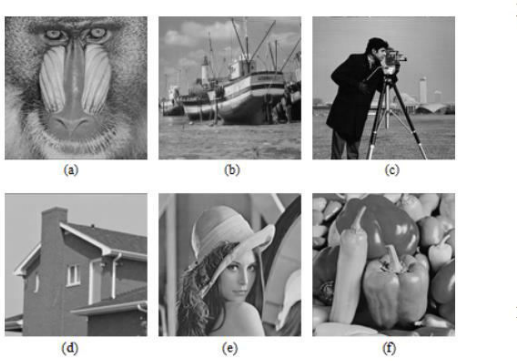


Fig. 9. Standard test images. (a) Baboon, (b) Boat, (c) Cameraman, (d) House, (e) Lena, and (f) Peppers [10].

Once the golden filtered images are obtained, an exhaustive fault injection-correction loop is executed using the Xilinx Soft Error Mitigation (SEM) IP Controller [20]. The SEM IP is an independent circuit that enables read/write operations in the FPGA configuration memory through the internal configuration access port (ICAP). In each iteration, a design under test (DUT) configuration memory bit is flipped, then a test image is processed and the golden comparison results are stored. Finally, the bit flip is corrected and the loop is repeated until all the DUT-related bits are covered. The described fault injection flowchart is illustrated in Fig. 10.

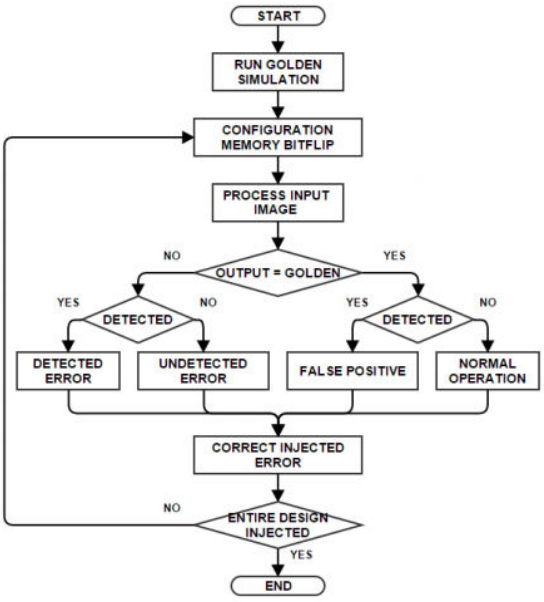


Fig. 10. Fault-injection flowchart.

For a better understanding of the described flowchart, the fault-injection framework is presented in Fig. 11. This figure represents the elements that are loaded in the SRAM-based FPGA. Those modules that are

grouped together inside the grey-shaded "Testbench" box are not affected by the fault injector, this is, only the DUT-related configuration memory bits are flipped.

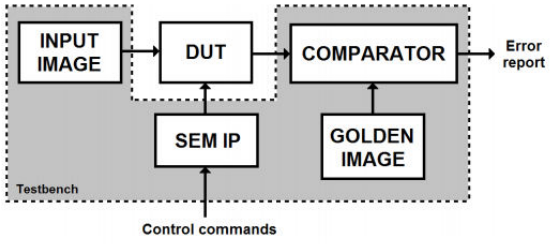
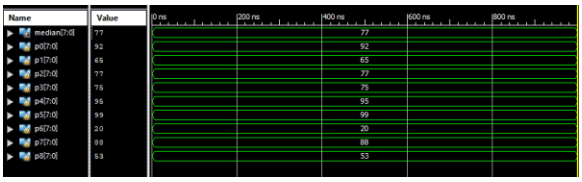


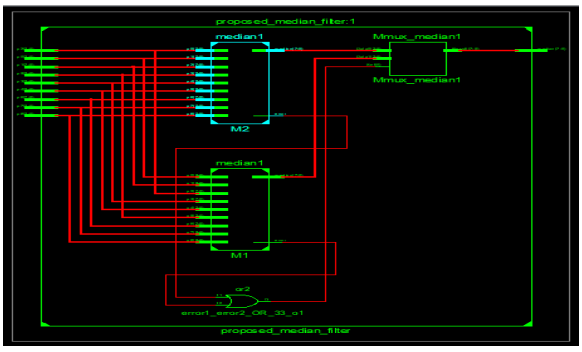
Fig. 11. Fault-injection framework [10].

The exhaustive fault injection-correction loop is performed for each error detection technique with all the standard test images, obtaining an image-level and a pixel-level report for each image and technique. An image-level report includes the number of corrupted and uncorrupted images that the technique has detected, while a pixel-level report contains the total number of corrupted and uncorrupted pixels that the technique has detected. So, with these reports, an estimation of the number of corrupted pixels per image can be calculated, as will be presented later in the next section.

V.SIMULATION RESULT

PRPOSED RESULT





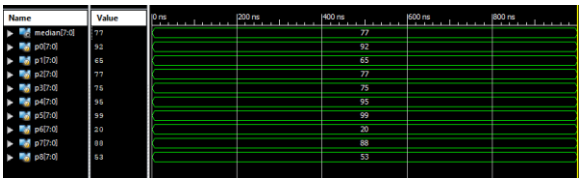
DESIGN SUMMARY:

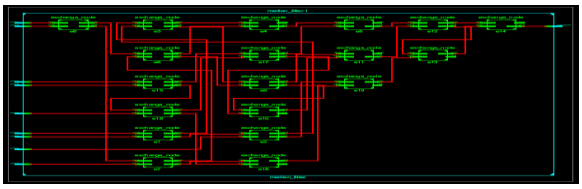
Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	855	63400	1%
Number of fully used LUT-FF pairs	0	855	0%
Number of bonded IOBs	80	210	38%

TIME SUMMARY:

```
LUT6:I0->O            14    0.097   0.571  M1/e13/a[7]_b[7]_LessThan_1_o24
LUT6:I3->O             4    0.097   0.707  M1/e13/Mmux_h61 (M1/h13<5>)
LUT6:I0->O             1    0.097   0.693  M1/e14/a[7]_b[7]_LessThan_1_o22
LUT6:I0->O            15    0.097   0.576  M1/e14/a[7]_b[7]_LessThan_1_o24
LUT6:I3->O             3    0.097   0.703  M1/e14/Mmux_h61 (M1/h14<5>)
LUT6:I0->O             1    0.097   0.693  M1/eH3/a[7]_b[7]_LessThan_1_o22
LUT6:I0->O             5    0.097   0.314  M1/eH3/a[7]_b[7]_LessThan_1_o24
LUT6:I5->O             3    0.097   0.703  M1/eH3/a[7]_b[7]_LessThan_1_o25
LUT6:I0->O             1    0.097   0.511  M1/median[7]_lH3[7]_LessThan_2_o
LUT6:I3->O             1    0.097   0.511  M1/median[7]_lH3[7]_LessThan_2_o
LUT5:I2->O             1    0.097   0.556  M1/median[7]_lH3[7]_LessThan_2_o
LUT5:I1->O             1    0.097   0.693  error1_error2_OR_33_o9 (error1_e
LUT6:I0->O             8    0.097   0.725  error1_error2_OR_33_o10 (error1_
LUT6:I0->O             1    0.097   0.279  Mmux_median81 (median_7_OBUF)
OBUF:I->O                   0.000          median_7_OBUF (median<7>)
----------------------------------------
Total                      26.919ns (3.493ns logic, 23.426ns route)
                                    (13.0% logic, 87.0% route)
```

EXTENSION RESULT:





DESIGN SUMMARY:

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	316	63400	0%
Number of fully used LUT-FF pairs	0	316	0%
Number of bonded IOBs	80	210	38%

TIME SUMMARY:

```
LUT6:I3->O             3    0.097   0.703  e10/Mmux_h61 (h10<5>)
LUT6:I0->O             1    0.097   0.295  e11/a[7]_b[7]_LessThan_1_o22 (e
LUT6:I5->O             7    0.097   0.539  e11/a[7]_b[7]_LessThan_1_o24 (e
LUT6:I3->O             4    0.097   0.707  e11/Mmux_l61 (l11<5>)
LUT6:I0->O             1    0.097   0.295  e12/a[7]_b[7]_LessThan_1_o22 (e
LUT6:I5->O            14    0.097   0.571  e12/a[7]_b[7]_LessThan_1_o24 (e
LUT6:I3->O             3    0.097   0.693  e12/Mmux_h161 (l12<5>)
LUT6:I1->O             1    0.097   0.295  e13/a[7]_b[7]_LessThan_1_o22 (e
LUT6:I5->O             7    0.097   0.539  e13/a[7]_b[7]_LessThan_1_o24 (e
LUT6:I3->O             3    0.097   0.703  e13/Mmux_h61 (h13<5>)
LUT6:I0->O             1    0.097   0.295  e14/a[7]_b[7]_LessThan_1_o22 (e
LUT6:I5->O             7    0.097   0.539  e14/a[7]_b[7]_LessThan_1_o24 (e
LUT6:I3->O             1    0.097   0.279  e14/Mmux_171 (median_6_OBUF)
OBUF:I->O                   0.000          median_6_OBUF (median<6>)
----------------------------------------
Total                      16.887ns (2.620ns logic, 14.267ns route)
                                    (15.5% logic, 84.5% route)
```

VI.CONCLUSION

This paper is an extension of our previous work presented in [10] about a fault-tolerant implementation of the median filter. Our proposed technique checks if the median output value is within a dynamic range created each time with the remaining non-median outputs. The proposed

technique has been compared in terms of resource utilization and error detection rate with DMR and RPR schemes. The design has been implemented in a Xilinx SRAM-based FPGA and several fault-injection campaigns have been performed. Image-level experimental results show that, adding 35% of FPGA resources to the original design, 91% of the corrupted images can be detected. This lower overhead means that a lower number of false positive detections are performed, so less reprogramming time is wasted and, consequently, the time that the median filter is not operational due to the reconfiguration is minimized. Pixel-level results show that images with homogeneous pixel-value regions are less susceptible to be corrupted when processed by a configuration memory damaged median filter. More homogeneous pixel-value regions imply more median filter inputs with equal values, so the probability that a median filter malfunction alters the median value is decreased. The test images have also been analyzed to prove that they cover different input scenarios. These experiments have revealed that the RPR method is dependent on the input images, while the detection rate of our proposed technique is not affected by the input images. In conclusion, although our technique cannot detect as many corrupted pixels as the DMR scheme, it detects enough to prevent 91% of the corrupted images from being erroneously processed by the next image processing module. Moreover, the remaining 9% of undetected images present a lower MSE value than the DMR, which means that the image damage propagation to successive image processing operations is minimized.

REFERENCES

[1] R. D. Schrimpf and D. M. Fleetwood, *Radiation Effects and Soft Errors in Integrated Circuits and Electronic Devices*. Singapore:World Scientific, 2004.

[2] R. C. Baumann, "Soft errors in advanced computer systems," *IEEE Des. Test. Comput.*, vol. 22, no. 3, pp. 258–266, May/Jun. 2005.

[3] 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.

[4] M. Y. Hsiao, "A class of optimal minimum odd-weight-column SEC-DED codes," *IBM J. Res. Develop.*, vol. 14, no. 4, pp. 301–395, Jul. 1970.

[5] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule," *IEEE Trans. Electron Devices*, vol. 57, no. 7, pp. 1527–1538, Jul. 2010.

[6] P. Reviriego, J. A. Maestro, S. Baeg, S. Wen, and R. Wong, "Protection of memories suffering MCUs through the selection of the optimal interleaving distance," *IEEE Trans. Nucl. Sci.*, vol. 57, no. 4, pp. 2124–2128, Aug. 2010.

[7] S. Satoh, Y. Tosaka, and S. A. Wender, "Geometric effect of multiple-bit soft errors induced by cosmic ray neutrons on DRAM's," *IEEE Electron Device Lett.*, vol. 21, no. 6, pp. 310–312, Jun. 2000.

[8] A. Neale and M. Sachdev, "A new SEC-DED error correction code subclass for adjacent MBU tolerance in embedded memory," *IEEETrans. Device Mater. Rel.*, vol. 13, no. 1, pp. 223–230, Mar. 2013.

[9] M. A. Bajura *et al.*, "Models and algorithmic limits for an ECC-based approach to hardening sub-100-nm SRAMs," *IEEE Trans. Nucl. Sci.*, vol. 54, no. 4, pp. 935–945, Aug. 2007.

[10] A. Dutta and N. A. Touba, "Multiple bit upset tolerant memory using a selective cycle avoidance based SEC-DED-DAEC code," in *Proc. IEEEVLSI Test Symp.*, May 2007, pp. 349–354.

[11] Z. Ming, X. L. Yi, and L. H. Wei, "New SEC-DED-DAEC codes for multiple bit upsets mitigation in memory," in *Proc. IEEE/IFIP 20th Int.Conf. VLSI Syst.-Chip*, Oct. 2011, pp. 254–259.

[12] L.-J. Saiz-Adalid, P. Reviriego, P. Gil, S. Pontarelli, and J. A. Maestro, "MCU tolerance in SRAMs through low-redundancy triple adjacent error correction," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, to be published.