New Protection Techniques Against SEUs for Moving Average Filters in a Radiation Environment

P. Reyes, P. Reviriego, *Member, IEEE*, J. A. Maestro, and O. Ruano

***Abstract*—Single Event Effects (SEEs) caused by radiation are a major concern when working with circuits that need to operate in certain environments, like for example in space applications. In this paper, new techniques for the implementation of moving average filters that provide protection against SEEs are presented, which have a lower circuit complexity and cost than traditional techniques like Triple Modular Redundancy (TMR). The effectiveness of these techniques has been evaluated using a software fault injection platform and the circuits have been synthesized for a commercial library in order to assess their complexity. The main idea behind the presented approach is to exploit the structure of moving average filter implementations to deal with SEEs at a higher level of abstraction.**

***Index Terms*—Digital filters, radiation hardening, redundancy, Single Event Upsets (SEUs).**

I. Introduction

THE effects of radiation on microelectronic circuits have a number of consequences that impact the design of devices that operate in certain environments [1]. One type of these effects are Single Event Effects (SEEs), which cause changes in the values of flips-flops or combinational logic [2]. In order to mitigate the effects of SEEs, a number of techniques can be used at the physical level (device size and structure) [3]. In addition to those techniques, redundancy can be introduced in the design so that it can detect and correct SEEs [4]. Single Event Effects (SEEs) can be classified in Single Event Upsets (SEUs) and Single Event Transients (SETs), depending on whether the change in the digital value occurs in a permanent storage element or in a combinational element. Single Event Transients (SETs) would only result in a persistent error if they propagate to the input of a storage element and they meet the setup and hold constraints there [5].

To deal with SEUs, a common approach is Triple Modular Redundancy (TMR), which triplicates the flip-flops in the design and adds logic to vote in case of conflict. If it is necessary to deal with SETs, Functional Triple Modular Redundancy (FTMR), which also triplicates the combinational logic, can be used [4].

Manuscript received October 4, 2006; revised December 18, 2006. This work was supported by the Spanish Ministry of Education and Science under Grant ESP-2006-04163 through the FEDER program.

P. Reyes, J. A. Maestro, and O. Ruano are with the Universidad Antonio de Nebrija, E-28040 Madrid, Spain (e-mail: preyes@nebrija.es; jmaestro@nebrija.es; oruano@nebrija.es).

P. Reviriego is with the Universidad Carlos III de Madrid, E-28911 Leganés, Spain (e-mail: revirieg@it.uc3m.es).

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

Digital Object Identifier 10.1109/TNS.2007.892118

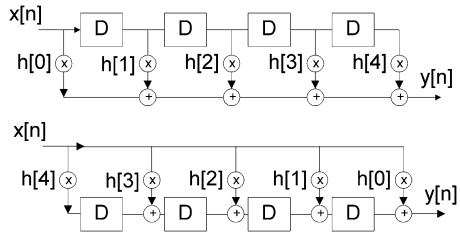

Fig. 1. FIR filter implementation.

One advantage of both TMR and FTMR is that they are general techniques that can be applied to most digital circuits. However, this comes at a high cost in terms of circuit area and power and more so for FTMR.

Another approach to deal with SEEs is to apply circuit specific techniques that exploit the inherent redundancy or fault tolerance of some circuits [6]–[9]. This means that some failures may not have an impact on the circuit functionality or may be detected and corrected using elements of the circuit. For example, in a circuit that implements a communications receiver, an SEU can cause a bit error in the received sequence, but if SEUs occur rarely, the receiver may still meet the Bit Error Rate target without further impacts. In this case, seldom SEUs are not critical and protection would not be an issue. However, if the Error Rate needs to be reduced, SEUs have to be handled accordingly. In the previous example, if it is possible to add a Viterbi Decoder [10] to the receiver, then the error caused by an SEU at the input may be corrected as part of the decoding process. In this way, a protection level has been added to the circuit in a specific way, taking its knowledge (behavior) into account.

Although there is almost no previous work covering their protection [11], Digital Finite Impulse Response (FIR) filters are good candidates for such approach, as they exhibit a regular structure and are frequently used in space applications [12]. A FIR filter performs the following operation [13]:

$$y[n] = \sum_{i=0}^{N-1} x[n-i] \cdot h[i] \tag{1}$$

where x[n] is the input signal, y[n] the output and h[n] the impulse response of the filter whose non-zero values are all in the 0 to $N-1$ interval.

A number of structures have been proposed to implement FIR filters [13]. Two of the most common ones are illustrated in Fig. 1.

Although both implementations are functionally equivalent, they exhibit different behaviors in the presence of SEEs. For example, an SEU in the first structure will cause an error in the

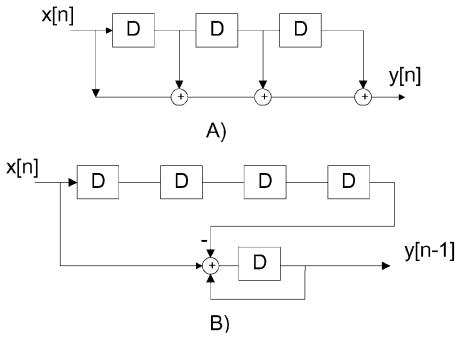

Fig. 2. Examples of moving average filter implementations. (A) FIR (B) IIR.

output that can last for $N-1$ clock cycles at most, whereas in the second one, an SEU will only corrupt $\mathrm{y[n]}$ for one clock cycle. In fact, in the first case, the error in the delay element propagates to the output filtered by a portion of the impulse response depending on the position of the delay element that suffered the SEU.

As it can be seen in this example, just looking at the effects of SEEs on existing filter implementations can yield interesting results. In this paper, one special type of FIR filter is analyzed, the *moving average filter*, which shows some interesting properties for implementation and that is also used in many applications.

A moving average filter performs the following operation [13]:

$$y[n] = \frac{1}{N}\sum_{i=0}^{N-1} x[n-i] \tag{2}$$

which is a particular case of (1). Normally, N is a power of two, so that the division can be implemented with a shift operation. In this case, the filter needs only adders.

A more efficient implementation can be derived by rewriting (2) as follows:

$$y[n] = y[n-1] + \frac{1}{N}(x[n] - x[n-N]). \tag{3}$$

In this case, only two adders are needed irrespective of the value of N. In fact, this implements the FIR filter using an Infinite Impulse Response (IIR) structure. A diagram for both implementations is shown in Fig. 2.

A first look at the effect of SEUs on both structures shows that in the case of the more efficient IIR implementation, SEUs in the delay line or in the accumulator can cause errors in the output that will persist until the filter is reset. This was previously noted in [14] as a drawback of the IIR structure and used to advocate the use of a FIR structure for cascaded moving averaged filters.

The previous discussion clearly shows how the presence of SEEs can influence the choice of the implementation structures for digital filters, and suggests the interest of smart protection techniques.

In the rest of the paper, new techniques to detect and correct SEUs in moving average filter implementations are explored. The paper is structured as follows: first, the proposed protection techniques are presented; then, using a software platform for fault injection, their effectiveness is evaluated in the presence of SEUs; and finally, the techniques are compared in terms of implementation complexity with existing approaches.

II. SEEs Protection Techniques

To come up with optimal SEEs protection techniques, we need to take the requirements of the application in which the filter is used into account. This is part of the bottom line of the presented methodology, generically described as "using the knowledge of the system." In order to assess this proposal, three distinct scenarios will be presented, each of them with different protection requirements. By carefully assessing these requirements, efficient implementations will be generated, which are more convenient than general (non-specific) protection techniques.

A. First Scenario (Low Protection Requirements)

Let us consider a first scenario where the filter is used to detect Ethernet Link Pulses [15] in the presence of noise. Assume that idle periods in between pulses are present, and that the detected pulses drive a state machine that ensures that occasional misdetection causes no problem to the application. In this situation, errors on the output of the filter can be tolerated *as long as they are transient*. In this application, the FIR-like implementation can be directly used, since this kind of errors are not permanent (see previous section). However, in order to use the IIR-like implementation, a protection technique is needed to ensure that the effects of SEUs become transient.

The traditional approach to deal with SEUs would be to apply TMR to all storage elements. If protection against SETs is also desired, the use of FTMR would also imply the replication of the combinational logic, which in this case is relatively small (more so if N is large).

A better approach, however, would be to detect that we are in an idle period and if the accumulator has a value that is greater than the maximum expected value for that case then reset the accumulator and the delay line registers. In this way, an extra counter can be added, which will compute the number of consecutive idle cycles in the system. By idle, we mean cycles in which the system input is below a given threshold, what would indicate the presence of noise in the filter. On the other hand, inputs over the noise threshold would be considered as active data. Therefore, if the counter detects N consecutive idle cycles (being N the number of taps in the circuit), this would mean that anything inside the circuit (stored in the delay line) comes from acquired noise, and therefore it can be discarded (reset).

With this sanity check mechanism, the filter actually is reset whenever N idle cycles are detected, what has an indirect benefit as protection technique for SEUs. No matter when an SEU hits the system, it will only last until the next reset of the circuit, making its effect transient. Notice that the mechanism will only work if frequent N idle cycles can be guaranteed, but since the system knowledge assures that this application can expect this kind of input, the proposed technique will be good enough, with a cost much lower than TMR. This is the core of the first proposed technique, as illustrated in Fig. 3.

Special care must be taken to ensure that SEUs in the added control logic do not cause errors; this can occur, e.g., if an SEU in one of the bits of the counter triggers the reset of the filter

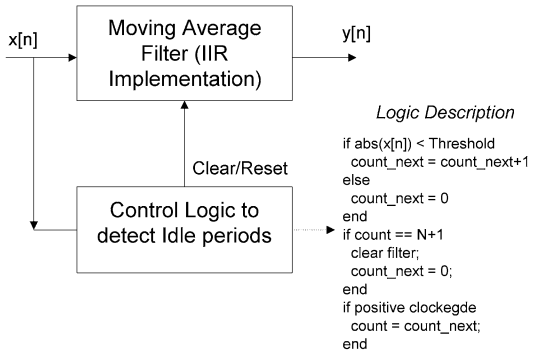

Fig. 3. Illustration of the first protection technique.

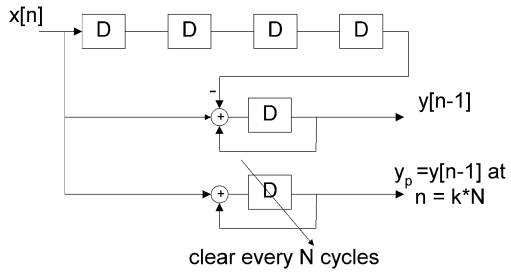

Fig. 4. Second proposed technique.

during the detection of a pulse. With a careful selection of the threshold value for the counter we can ensure that this situation will never happen. If N is a power of two, then the threshold to reset can be set to $N+1$ samples, rather than to N. In this way, a single SEU in only one bit will never cause the mentioned reset, since the value of the counter would be zero during pulse detection, and in order to get to $N+1$ (reset condition), two bits flipped to 1 would be needed (what can be caused by two simultaneous SEUs or other phenomena like single heavy ions with large diffusion track radii).

B. Second Scenario (Average Protection Requirements)

As a second scenario, let us take an application that can tolerate occasional errors on the output of the filter (like in the first scenario) but that has no guarantee of idle periods that can be used to reset the accumulator and delay line registers.

In this situation, the computation of the output can be done in parallel by another structure added to the filter, for the sake of comparison. Obviously, if this added structure is a replica of the filter itself, we would be doubling the complexity of the system. To avoid this situation, this parallel structure will be implemented with a *decimated* filter (as illustrated in Fig. 4), which has a structure simpler than a regular filter, with the drawback that it only computes the right output one out of N cycles.

In this way, the output of both structures would be compared at each $\mathrm{y}[\mathrm{n}^* N]$, what is one out of N times. After the comparison, several cases can arise:

- Both structures have the same output. This means the system is error-free.
- The structures have different outputs. This means that an SEU has hit the original filter or the secondary (decimated) one. To determine where the error is, the decimated filter is reset, and after N cycles the comparison is made again. After this, it may happen that:
  - The error is gone. That means the SEU was in the decimated filter.
  - The error is still present. That means the SEU is in the main filter which needs to be reset, in order to eliminate it.

Regardless of which of the previous situations happens, the SEU will always be transient, what satisfies the application requirement, but in this case, no idle periods are needed. Therefore, the protection level has been increased with a minimum complexity increment thanks to applying the knowledge of the system.

It is worth noting that although we have focused in SEUs, the techniques proposed so far would work as effectively for SETs, as their effects are, in the worst case, equivalent to an SEU (with the exception of a SET in the incoming signal for the second technique). This compares with the triplication of the combinational logic required in FTMR to deal with SETs.

C. Third Scenario (High Protection Requirements)

As a third and final scenario, let us assume that we want to provide protection such that SEUs do not cause any errors in the output of the filter. One alternative to using TMR in all registers is to take advantage of the fact that the registers for the FIR implementation in the upper part of Fig. 1 and for the IIR like implementation are connected in such a way that their values do not suffer changes as they move across the delay line.

This can be used to compute a two-dimensional parity as follows:

- For each input value, compute a 'vertical' parity bit Pv.
- For each bit position on the input value, compute a 'horizontal' parity bit Ph across all the bits that have that position on the registers of the delay line.

Pv is only updated at the input of the delay line, while Ph is updated every clock cycle with the bit entering the delay line and the one leaving it. These two sets, Pv and Ph, form the accumulated parity of the circuit, which is constantly being updated.

Dynamically, each time a new value reaches the circuit, both the horizontal and vertical parity are re-checked and compared with the accumulated values. Then, several situations can arise:

- The actual and accumulated values are the same. There is no problem with the system and its behavior can be taken as correct.
- There is a discrepancy between a bit of the accumulated and actual Ph *and* between a bit of the accumulated and actual Pv. If both differences happen, that means an SEU has affected a register in the delay line. The bit affected by the SEU is the crossing point of the discrepant Ph and Pv. In this way, since it has been identified, it can be corrected instantly, and therefore, the system behavior remains correct.
- There is a discrepancy between a bit of the accumulated and actual Ph *or* between a bit of the accumulated and actual Pv. If only one of the parity registers shows the discrepancy, it would mean an SEU has affected the discrepant parity register itself. It is important to remember that all the

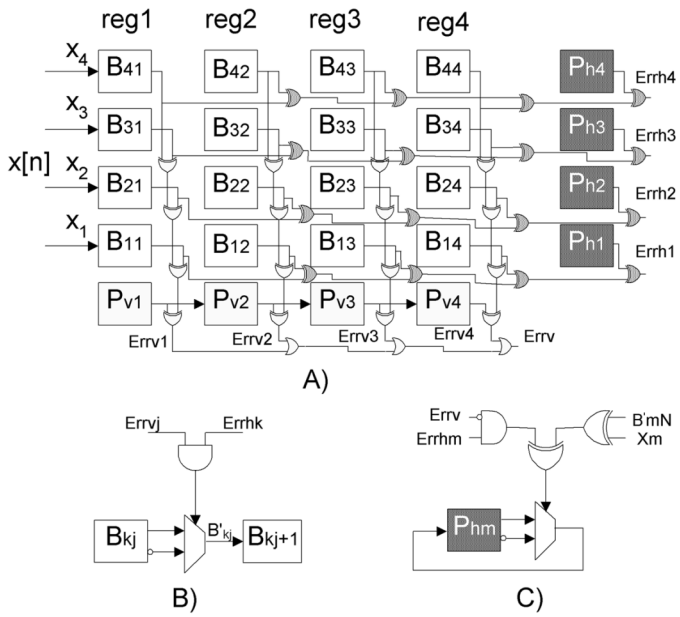

Fig. 5. Third proposed technique.

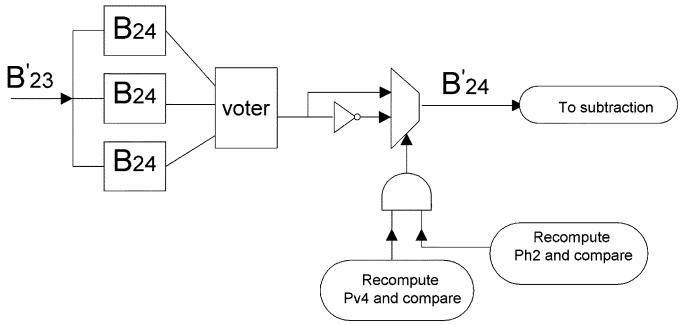

Fig. 6. Proposed implementation for the last delay line register in technique 3.

extra structure added for protection can also be affected by SEUs.

This is the outline of the third proposed technique, as illustrated in Fig. 5.

Note that if an SEU strikes on $\mathrm{Ph_x}$ the error can be permanent and therefore it needs to be corrected. One alternative is to use TMR on $\mathrm{Ph_x}$. However, it seems that a better option is to generate a signal $\mathrm{Err_v} = \mathrm{Err_{v1}}$ or $\mathrm{Err_{v2}}$ or ... *or* $\mathrm{Err_{vn}}$ that indicates if there is an error on the vertical parity bits. Then if $(\mathrm{Err_v} == 0)$ and $(\mathrm{Err_{hi}} == 1)$ we know that there has been an SEU on $\mathrm{Ph_i}$ and we can do $\mathrm{Ph_i} = \mathrm{not}(\mathrm{Ph_i})$. This is illustrated in Fig. 5(C).

As already mentioned, at each clock cycle Phs and Pvs are recomputed and compared with the stored values. If there are no differences $\mathrm{B_{m,n}}$ would be updated at each clock cycle with $\mathrm{B_{m,n-1}}$. If both $\mathrm{Pv_m}$ and $\mathrm{Ph_{n-1}}$ show differences with the stored values, then $\mathrm{B_{m,n}}$ would be updated with $\mathrm{B_{m,n-1}}$ negated. This correction logic is illustrated in Fig. 5(B).

The overhead of this approach is $N + \mathrm{M}$ flip-flops plus the logic to compute the parities and detect the errors. In principle, it would work for both the FIR and IIR implementations.

D. Revisiting the Third Scenario: Delay Issues

There is, however, one potential problem with the last technique that can be explained by analyzing Fig. 5(B). Basically, if an SEU occurs in the middle of a clock cycle, it will not be corrected until the new value of $\mathrm{B_{kj}}$ propagates through the parity checking logic and sets $\mathrm{Err_{hk}}$ and $\mathrm{Err_{vj}}$ to 1 (so that the inverted output of the flip-flop is selected). So, potentially, if the SEU occurs near the end of the clock cycle the wrong value may be stored in the next element of the delay line before the correction can take place.

To avoid this problem, buffers could be used to delay the inputs to the multiplexer so that they have approximately the same delay that the signal that controls that multiplexer. However, this may be complex to do and it would also complicate synthesis and place and route that would require manual intervention. This would make the design technology dependent and therefore harder to reuse.

A better alternative can be found by noting that for the IIR implementation, this problem is only an issue if it occurs in the last element (register) of the delay line. If it happens on previous elements, it would be corrected during the next clock cycle, and it would never cause a difference in the filter output (since only the last element of the delay line is used for subtraction). This observation could be used to solve the problem by adding TMR to this last register only, such that SEUs in this register would not propagate to the multiplexer. In this situation, for this last register, we only need to correct SEUs that occurred in the previous delay line register and that have been propagated to this because they occurred at the very end of the clock cycle (other SEUs will be eliminated by TMR). The overall strategy is shown in Fig. 6. The additional cost in terms of area is quite low, as TMR is only added to one register.

By adding this protection, we ensure that SEUs will not cause any error on the filter output, and therefore the protection requirements stated for this scenario are met.

Note that this can only be done for the IIR structure. However, for the FIR one, we can have errors in the filter output caused by SEUs that occur in any delay line register (all are connected through adders to the filter output) at a time, such that the wrong value propagates to the output before it can be corrected. The amount of time during which an SEU can cause this error is the difference between the delay of the correction logic and the data input to the multiplexer. If this is small compared to the clock cycle, then most of the SEUs will not cause errors in the filter output and this technique may still be useful. If that is not the case, buffers could be used as discussed before.

A similar problem to the one just discussed can occur when correcting $\mathrm{Ph_k}$ if an SEU has changed $\mathrm{B_{kj}}$, and $\mathrm{Err_{hk}}$ becomes 1 before $\mathrm{Err_v}$ does. In this case, $\mathrm{Ph_k}$ would be inverted, and if that happens at the end of a clock cycle, the inverted $\mathrm{Ph_k}$ could be stored. However, in this situation, the error in $\mathrm{Ph_k}$ will not cause errors on the filter output, and it will be corrected in the next clock cycle. Finally, the two previous problems could also theoretically occur simultaneously. In that case, $\mathrm{Ph_k}$ would be inverted and $\mathrm{B_{kj}}$ would not be corrected creating an error, in practice. Nevertheless, this is unlikely, since $\mathrm{Err_{hk}}$ should be 1 for the first problem to occur, and in that case $\mathrm{Err_{vj}}$ needs to

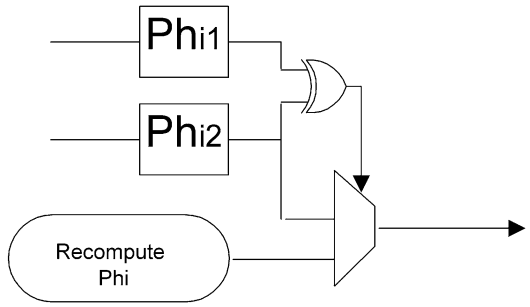

Fig. 7. Method for $\mathrm{Ph_i}$ protection in technique 3B.

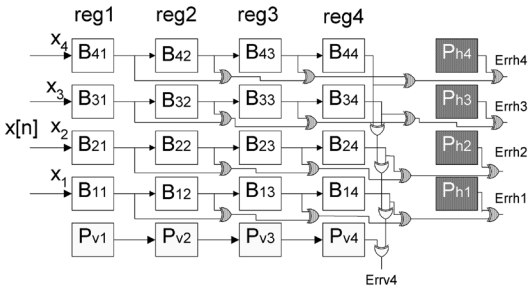

Fig. 8. Implementation detail of technique 3B.

be 0 for the second problem to occur. In most cases, the delay to generate $\mathrm{Err_{hk}}$ would be similar or larger than the one to generate $\mathrm{Err_{vj}}$ (assuming that we have more taps than bits in each register).

E. Protection Technique Optimization

The technique explained in scenario 3 would be referred to as *technique 3* in the rest of the document, as some optimizations are going to be further discussed. We start by noting that in the case of the IIR-like implementation, the scheme can be simplified, as the values of the registers are only used at the end of the delay line. This allows correcting only the errors in the last register of the delay line, so that the additional logic is greatly reduced (basically the logic on Fig. 5(B) would now be used in the last register rather than in all of them). The price paid for this is that now SEUs that occur a few clock cycles apart are more likely to have the effect of multiple upsets producing errors at the output. For an environment in which that is unlikely, this technique 3A would be a suitable option.

Once the correction on all but the last register in the delay line has been removed, the re-computation of the vertical errors on the previous registers is kept, for the sole purpose of Ph protection. An alternative would be to duplicate the registers that store Ph_i so that if there is an SEU in one of them, we can recover by doing: $if\ \mathrm{Ph_i}1! = Ph_i2$, then, use the re-computed version of Ph_i for the next clock cycle as shown in Fig. 7. This allows removing all the logic for vertical error re-computation and provides an additional reduction of complexity as illustrated in Fig. 8. This optimization will be referred to as technique 3B. It has the additional advantage that it does not exhibit the problem on the correction of $\mathrm{Ph_k}$ previously discussed.

As stated before for both optimizations, 3A and 3B, if two SEUs occur N or less clock cycles apart, it may not be possible to correct them. In fact, there are situations in which that is also true for technique 3. These situations involve an SEU hitting one Pv followed by another SEU prior to the erroneous Pv leaving the delay line. This could be avoided with a small area increment by protecting Pv in a similar way to what has been proposed for Ph in Fig. 7. This option, however, is not fully explored in this paper due to lack of space.

One of these situations is as follows: an SEU hits one Pv and then, later, another one hits one of the bits of the register that contains the erroneous Pv. In that case, the vertical parity checking would give the correct result and the erroneous bit will not be corrected. This is however an unlikely event when compared with all the possible situations that would cause an error in techniques 3A and 3B. For example, if an SEU hits $\mathrm{B_{41}}$ and two clock cycles later another SEU hits $\mathrm{B_{33}}$, we would have errors on $\mathrm{B_{33}}$ and $\mathrm{B_{43}}$. This would set $\mathrm{Err_{h3}}$ and $\mathrm{Err_{h4}}$ but not $\mathrm{Err_{v3}}$, and therefore no correction would be made. Another example is when an SEU hits $\mathrm{B_{41}}$ and two clock cycles later another one hits $\mathrm{B_{22}}$. In that case, $\mathrm{Err_{h4}}$, $\mathrm{Err_{h2}}$, $\mathrm{Err_{v3}}$ and $\mathrm{Err_{v2}}$ would be set to 1, so that when the first erroneous value reaches $\mathrm{B_{44}}$ at the next cycle, both $\mathrm{B_{44}}$ and $\mathrm{B_{24}}$ will be inverted introducing an error in $\mathrm{B_{24}}$. In general, it can be seen that any two SEUs on the register bits will create errors on the filter output for techniques 3A and 3B.

The previous example also shows that the original technique 3 is unable to protect against Multiple Bits Upsets (MBUs). Although we have not fully investigated the effects of MBUs in this paper, it can be demonstrated that none of the proposed techniques nor TMR can ensure that there will be no errors in the filter output, although TMR will be more robust. In the case of the proposed technique 3 for the IIR, the MBUs can create permanent errors that could be dealt with by adding technique 1 or 2 to make those errors transient.

Finally, note that for the IIR like implementation, it may be possible to use a memory block to store the registers, since only one read and one write operation per clock cycle is needed. Although not explored in this paper, using a memory with built-in SEU protection [16] may be then an alternative to protect the filter.

Another observation is that the proposed scheme for the FIR implementation of the moving average filter, with the limitations previously outlined, can be in fact used for any FIR filter that is implemented with that structure.

III. Evaluation of the SEU Protection Techniques

In the previous section, some protection techniques have been discussed that can be used for moving average filters in certain applications. In this section, their effectiveness for actual implementations will be evaluated. The proposed techniques have been implemented in VHDL and then the circuits have been synthesized for a commercial ASIC library (see Section IV). This will also allow assessing the efficiency of the proposed techniques in terms of circuit complexity and compare it with the traditional approaches in the next section. Then, a simulation environment will be used to insert SEUs in the circuit and evaluate their effects.

A block diagram of the environment is shown in Fig. 9. This environment uses Matlab to generate the reference signals for the input and output of the filter that are stored in files. These

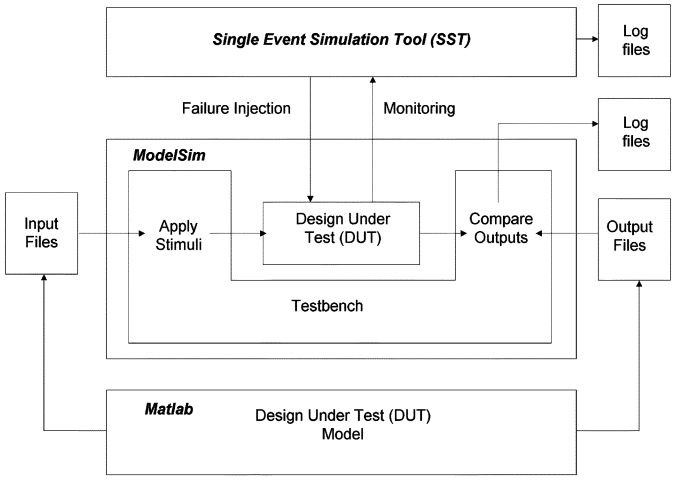

Fig. 9. Simulation environment block diagram.

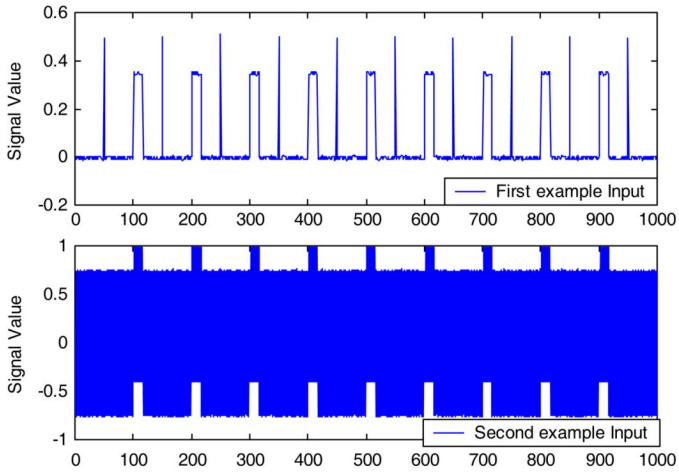

Fig. 10. Input signals for examples 1 (top) and 2 (bottom).

files are then used to drive the circuit implementation using a commercial VHDL simulator (Modelsim), whose output is compared with the reference output in order to detect any discrepancies. SEEs are introduced using the Single Event Upset Simulation Tool (SST) developed at the European Space Agency [17], which has been updated so that it works with the latest releases of Modelsim and extended in functionality for the purpose of these experiments [18].

This environment is flexible in the way that different signals are easily generated in Matlab to exercise the filter under different conditions. The SST is also convenient to define when and where to insert SEEs in a design. A case study of the tool working with FIR and IIR filters can be found in [19].

To better illustrate these features, the first protection technique described in the previous section has been evaluated, assuming an 8-bit input signal, $\mathrm{x[n]}$, with range $-1, 1$ and N set to 16. The threshold to reset the accumulator is three LSBs. We start by generating a sequence (see top of Fig. 10) of pulses (instants multiple of 100) plus impulsive noise (instants 50, 150, 250, etc.) in Matlab, and then an SEU is introduced in the accumulator on cycle 20, in order to check the discrepancies at the output. The results (see Fig. 11) show that there is no permanent error and also demonstrate how the circuit recovers from such errors. We have also used the SST to insert SEUs only during the reception of pulses and only in the protection logic, to check that pulse detection is not affected by SEUs in this case.

For the second example, a signal that contains the pulses plus a high frequency noise has been generated (see bottom of Fig. 10), and then the SST has been used to insert an SEU in cycle 10. In this case, technique 1 would not work, as the incoming signal continuously exceeds the threshold. As it can be seen in Fig. 12, technique 2 removes the error in approximately $2^{*}N$ samples. Besides, the SST has been employed to check that SEUs in the decimated filter used for correction do not cause errors in the output of the filter.

For the third example, some SEUs have been introduced in the registers of the delay line, in order to check that they are corrected. Then, other SEUs have been applied to the parity bits, in order to verify that they do not trigger erroneous corrections. Both VHDL simulations (introducing delays in the parity

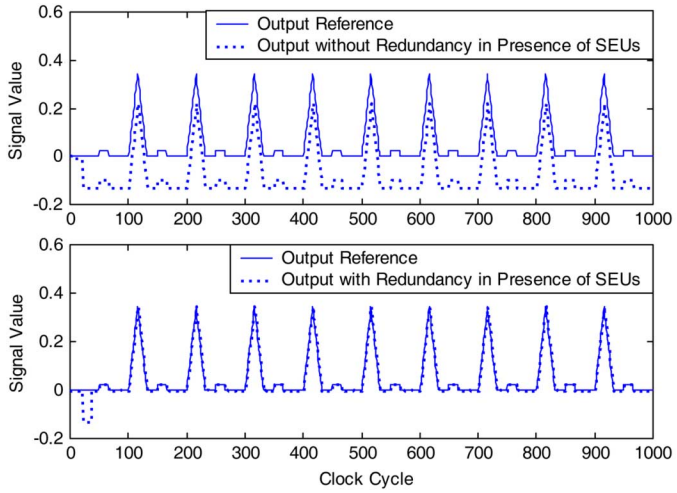

Fig. 11. Output of an unprotected IIR (top) and of an IIR protected with technique 1 (bottom).

checking and delay back-annotated simulations on the synthesized netlist) have been run, in order to check that the SEU propagation described in the previous section causes errors in the filter output when TMR is not present in the last delay register element. Reciprocally, it has been proved that this does not happen once TMR is added. With the final implementations of the different variants of the technique, we have verified that the output of the filter does not show any discrepancy with the reference output when inserting random SEUs that are at least N clock cycles apart. We have also checked that SEUs that are closer than N clock cycles apart can create errors on the filter output, as described before.

Although we have focused on analyzing the effects of SEUs affecting some particular bits or registers, the fault insertion campaigns applied to all three examples are far more extensive, including in all cases thousands of random SEUs.

IV. Comparison with TMR

In this section, the proposed techniques are compared with the traditional ones in terms of complexity. We have focused on

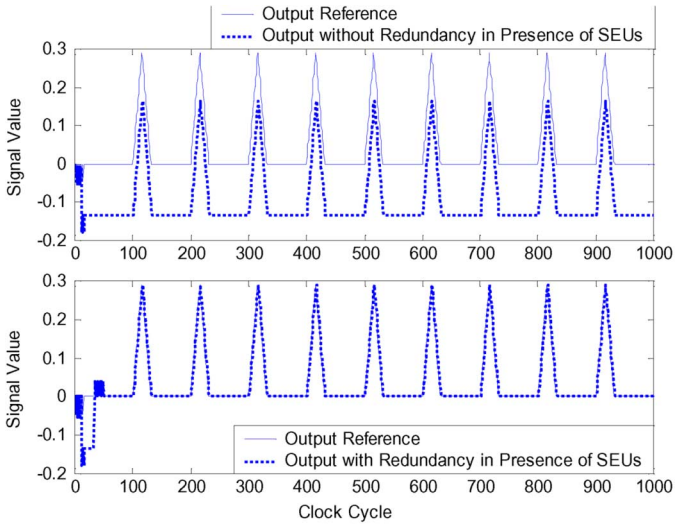

Fig. 12. Output of an unprotected IIR (top) and of an IIR protected with technique 2 (bottom).

TABLE I
NUMBER OF GATES FOR THE FIR LIKE IMPLEMENTATIONS

	N = 8	N = 16	N = 32
FIR	713	1591	3788
FIR with TMR	1686	3538	7500

TABLE II
NUMBER OF GATES FOR THE IIR LIKE IMPLEMENTATIONS

	N = 8	N = 16	N = 32
IIR	517	866	1550
IIR with Tech. 1	690	1141	2024
IIR with Tech. 2	835	1300	2186
IIR with TMR	1633	2962	5591
IIR with Tech. 3	1452	2332	4029
IIR with Tech. 3A	1263	1915	3194
IIR with Tech. 3B	1173	1701	2727

the number of equivalent gates, as an estimate of circuit complexity. The results (see Tables I and II) have been generated for a TSMC 0.25um library and assuming a 50-Mhz clock and an 8-bit datapath. For the FIR case, the structure showed at the top of Fig. 1 has been used, as it is the one that results in a lower gate count.

The first thing to note is that the IIR implementation is more efficient than the FIR one, as expected, and that for large values of N, the difference can be more than double. This explains why the IIR implementation is normally used in the absence of SEUs. It should also be remarked that although not shown in the tables, the IIR implementation is also faster, and therefore, for the same technology, the filter can operate at higher clock frequencies.

In the presence of SEUs, the unprotected IIR implementation will suffer permanent errors, while the FIR one would only show temporary ones. In this case, what needs to be compared is the unprotected FIR versus the IIR protected with techniques 1 and 2 that mitigate the effects of SEUs, so that they become temporary. It can be seen that for large values of N, the IIR with techniques 1 and 2 is still significantly smaller that the FIR. For $N = 16$, the reduction is 28% and 18% respectively, while for $N = 32$, the reduction is over 40% in both cases. This is so because for these techniques, the amount of redundancy introduced is almost independent of the value of N. The IIR protected with these techniques still has the speed advantage versus the FIR that has been mentioned before.

If we now focus on implementations that ensure that isolated SEUs do not cause errors on the filter output, we can compare the IIR protected with technique 3 (in its various forms) versus the IIR protected with TMR (note that the FIR with TMR is not considered, as it would be equivalent in this case to the IIR with TMR but with larger number of gates). The original technique 3 results in a reduction of 10% to 30% in the number of gates. The variants 3A and 3B provide larger savings of 20% to 40% in the first case, and 30% to 50% in the second. These results show that technique 3 may be a good choice in this case. Whether to use the original technique 3 or one of its variants will depend on the probability of having two SEUs occurring a few (N or less than N) clock cycles apart. If that is negligible, then 3A or 3B should be preferred in order to minimize the implementation cost. If it is not, then the original technique 3 should be used to provide better protection against those events.

In summary, the proposed techniques 1 and 2 enable the use of the IIR structure when transient errors are acceptable at the filter output, while technique 3 provides effective protection when all the errors are to be avoided at the filter output. In both cases, significant savings in terms of area are obtained when compared with traditional techniques.

V. CONCLUSIONS AND FUTURE WORK

In this paper, new techniques to protect moving average filter implementations from the effects of SEUs have been presented. These techniques exploit both application and system knowledge in order to provide a more intelligent protection that results in a lower circuit complexity compared to TMR, as it has been shown in the paper. The proposed techniques have also been tested using a simulation environment that enables a flexible testing of the effects of SEUs on signal processing circuits.

In previous sections, it has been mentioned that some of the proposed techniques can be applied to general FIR filters. Evaluating their effectiveness and complexity is a natural extension of the work presented in this paper. FPGA-based filter implementations could also be studied to assess if similar techniques are applicable.

More generally, the same broad idea of using system knowledge to derive protection techniques can also be applied to other types of filters like generic IIR filters, adaptive filters, filter banks, etc.

Another topic for future work would be to evaluate the proposed techniques using more complex fault models [20], [21]. This requires major changes to the SST that are planned for future releases of the tool.

ACKNOWLEDGMENT

The authors would like to thank D. Gonzalez-Gutierrez for his help with the ESA SST tool and for his comments and suggestions, and E. Nogales for her contribution to some of the experiments that are reported in this paper.

REFERENCES

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

[2] P. E. Dodd and L. L. Massengill, "Basic mechanisms and modeling of single-event upset in digital microelectronics," *IEEE Trans. Nucl. Sci.*, vol. 50, no. 3, pp. 583–602, Jun. 2003.

[3] M. P. Baze, S. P. Buchner, and D. McMorrow, "A digital CMOS technique for SEU hardening," *IEEE Trans. Nucl. Sci.*, vol. 47, no. 6, pp. 2603–2608, Dec. 2000.

[4] S. Hanbic, FTMR: Functional Triple Modular Redundancy ESA Rep. FPGA-003-001, Dec. 2002.

[5] P. E. Dodd, M. R. Shaneyfelt, J. A. Felix, and J. R. Schwank, "Production and propagation of single-event transients in high-speed digital logic ICs," *IEEE Trans. Nucl. Sci.*, vol. 51, no. 6, pp. 3278–3284, Dec. 2004.

[6] B. Shim, N. R. Shanbhag, and S. Lee, "Energy-efficient soft error-tolerant digital signal processing," in *Signals, Syst. Comput., 2003. Conf. Rec. 37th Asilomar*, Nov. 2003, pp. 1493–1497.

[7] 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.

[8] S. Wang and N. K. Jha, "Algorithm-based fault tolerance for FFT networks," *IEEE Trans. Comput.*, vol. 43, no. 7, pp. 849–854, Jul. 1994.

[9] S. Lu, J. Shih, and S. Huang, "Design-for-testability and fault-tolerant techniques for FFT processors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 6, pp. 732–741, Jun. 2005.

[10] B. Sklar, *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[11] R. Hentschke, F. Marques, F. Lima, L. Carro, A. Susin, and R. Reis, "Analyzing area and performance penalty of protecting different digital modules with Hamming code and triple modular redundancy," in *Proc. 15th Symp. Integr. Circuits Syst. Design*, 2002, pp. 95–100.

[12] C. Lambert-Nebout and G. Moury, "On-board digital signal processing for IASI mission," in *Proc. 6th Int. Workshop Digital Signal Process. Tech. Space Applicat.*, Noordwijk, The Netherlands, Sep. 1998.

[13] A. V. Oppenheim and R. W. Schafer, *Discrete Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1999.

[14] H. Sato, "Moving average filter," U.S. patent 6,304,133, Oct. 16, 2001.

[15] *IEEE 802.3i Ethernet over Unshielded Twisted Pairs Standard (10BaseT)*, IEEE 802.3i, 1990.

[16] "Using EDAC RAM for radtolerant RTAX-S FPGAs and axcelerator FPGAs," Actel Application Note.

[17] D. Gonzalez-Gutierrez, "Single event upset simulation tool functional description," ESA Rep. TEC-EDM/DCC-SST2, Jul. 2004.

[18] *Computer Architecture and Technology Group, Universidad Antonio de Nebrija*, [Online]. Available: http://www.nebrija.es/~jmaestro/esa

[19] "A simulation platform for the study of soft errors on signal processing circuits through software fault injection," TR-2006Sep-001, Sep. 2006 [Online]. Available: http://www.nebrija.es/~jmaestro/TR-2006Sep-001.pdf

[20] A. M. Chugg, M. J. Moutrie, and R. Jones, "Broadening of the variance of the number of upsets in a read-cycle by MBUs," *IEEE Trans. Nucl. Sci.*, vol. 51, no. 6, pp. 3701–3707, Dec. 2004.

[21] D. Radaelli, H. Puchner, S. Wong, and S. Daniel, "Investigation of multi-bit upsets in a 150 nm technology SRAM device," *IEEE Trans. Nucl. Sci.*, vol. 52, no. 6, pp. 2433–2437, Dec. 2005.