Abstract

Future technologies, below 90nm, will present transistors so small that they will be heavily influenced by electromagnetic noise and SEU induced errors. This way, together with process variability, design as known today is likely to change. Since several soft errors may occur in a small period of time, the traditional hypothesis that the behavior of a gate is fixed is no longer valid and a different design approach must be taken. The use of inherently robust operators as an alternative to conventional digital arithmetic operators is proposed in this study. The behavior of the proposed operators is analyzed through the simulation of single and multiple random faults injection, and the proposed circuits are robust to severe noise and single event upsets, standing multiple simultaneous faults. The number of tolerated upsets varies according to the number of extra bits appended to the operands, and is limited only by the area restriction.

1. Introduction

As the microelectronics industry moves inexorably towards deep sub-micron technologies, systems designers become increasingly concerned with the reliability of future devices, which will have smaller transistors, that will be more sensible to the effects of electromagnetic noise, neutron and alpha particles. These events will be responsible for the occurrence of transient random faults, even in fully tested and approved circuits.

For this less reliable technology, besides the fact that the probability that large modules will have defects that impair its correct operation can be higher than 50%, it is likely that common gates, such as a simple NAND, will behave as expected only a fraction of the total time and, therefore, will statistically fail.

In order to survive in this new scenario, it is clear that new fault tolerance techniques must be defined, not only for safety critical systems, but to general purpose computing as well. Current fault tolerance techniques are effective for single event upsets (SEUs) and single event transients (SETs). However, they are unlikely to withstand the occurrence of multiple simultaneous faults that is foreseen with those new technologies.

To face this challenge, either completely new materials and manufacturing technologies will have to be developed, or fully innovative circuit design approaches must be taken.

In this paper we present the results of a new approach to cope with this faulty behavior of gates, through the use of bit stream computation as a means to provide tolerance to multiple soft faults in digital circuits for arithmetic intensive applications. Instead of working with a coded word, we work with a redundant representation of the possible output values of a multiplier. By adding extra bits to the binary product, one can achieve intrinsic fault tolerance to transient faults effects and even to a certain amount of manufacturing defects in the circuit. The analysis of the multiplier operation shows that, with the insertion of \( r \) least significant bits in the binary value of the product, the multiplication of two \( n \times n \) multiplier that can withstand up to \( 2^{n-1} \) simultaneous transient faults with no error in the result.

This paper is organized as follows: section 2 describes related work and shows that tolerance to multiple simultaneous transient faults is not being addressed. In section 3, our previous study with stochastic operators is briefly described and commented. In sections 4 and 5, the new operators used in our work are described, together with the reasoning that shows why the multiplication operator is tolerant to multiple simultaneous upsets. In section 6
we discuss the results obtained in the experiments and our plan for future work on this project.

2. Related Work

Recent works stress the industry's growing concerns about the trend towards the increasing incidence of soft errors in circuits designed with future technologies [1, 2, 3]. These errors are not due to poor design nor to manufacturing process defects, but rather they derive from the incidence of external radiation and/or electrical noise, hazards that are known to be harmful to digital and analog circuits behavior and have been subject of study by designers since the early 1980's [4].

What turns soft errors into a major concern nowadays is that the higher frequencies to be reached by future circuits will lead to cycle times shorter than the duration of transient pulses caused by radiation and/or self induced electromagnetic noise. Therefore, those pulses will have a higher probability of affecting the output of combinational circuits, as well as the values stored in memory elements. Besides that, shrinking transistor dimensions and lower operating voltages will make circuits more sensible to neutron and alpha particles, which also induce transient pulses.

Several techniques to maintain circuit reliability even under those critical conditions have been proposed, including hardware implemented parity code and source level code modification [5], time/space redundancy [6], triple modular redundancy (TMR) and double modular redundancy with comparison (DWC) with concurrent error detection (CED) [7].

All those techniques, however, are intended to cope with the occurrence of single soft errors (SEUs/SETs) in a given interval. Since, thanks to the already mentioned technology trends, the probability of occurrence of multiple events will become higher, the single error hypothesis must be changed to a multiple simultaneous errors hypothesis.

The concept of "error tolerance" and the relaxation of the 100% correctness requirement for devices and interconnects is proposed in [8] as an alternative to increase yield level for future technologies. According to this proposal, "a circuit is error tolerant with respect to an application if it contains defects that cause internal errors and might cause external errors, and the system that incorporates this circuit produces acceptable results". That work is focused mainly in the concept that, for some applications, a certain degree of error is acceptable and, therefore, parts that would be rejected by manufacturing tests for one application could then be acceptable for others, thereby increasing the yield. That approach would then use different, application dependent, manufacturing tests to select acceptable devices for each application, and no additional provision for tolerance of multiple upsets while in operation is made.

An alternative to tolerate multiple faults has been proposed in [9]. That work aims to detect and recover from single or multiple manufacturing faults and infield errors during the same cycle, using Berger Code prediction for the data path components for small integer data paths (8 or 16 bits, with no more than 16 registers per register file). However, the author themselves comment that beyond this point, data path overhead can reach unacceptable levels. For the control logic, an application-specific error detection scheme is proposed, whose basic concept is to back up only the control logic of a standard processor that is necessary for the instruction subset used by the application. The use of two processors in a master-trailer scheme, where both processors have a built-in self check facilities and use micro rollback with a distance of one cycle, allows the detection and recovery from errors due to transient effects within one cycle (single-event upset). In case the error is not transient, the master processor is considered permanently faulty and the trailer processor takes over, while technicians test and repair the master offline. The main drawbacks of this proposal are the fact that Berger Code, despite detecting multiple-bit faults, is effective only for single and unidirectional faults in combinational circuits. Besides the area overhead incurred in the whole control logic, there is no support for the occurrence of simultaneous faults, which is the main concern of our work.

This paper proposes new ways for designing arithmetic operators that will be tolerant to multiple simultaneous upsets with an area increase that can be adjusted according to the area × tolerance restrictions of the application. In the worst case, when higher tolerance is needed, the area penalty is limited to the data path operators, since other components would be protected using alternative schemes.

3. Previous Studies

In this context, we started our work studying one alternative to build arithmetic stochastic operators that could withstand multiple soft faults, while still producing results within an acceptable margin of error for arithmetic applications, aiming to use such operators in applications that tolerate small error percentages (error tolerant), or in which positive and negative errors cancel one another during cascading calculations, like in digital filters [10].
Despite being very simple circuits, the stochastic operators for addition and multiplication failed to produce results with a reasonable precision when used together in the implementation of a FIR filter. In order to improve the precision, a very high number of samples was required for the multiplier.

Evolving from the stochastic approach, which converts numbers into bit streams, we defined new addition and multiplication operators which, besides being tolerant to multiple soft errors, produce more precise results in all the calculations, if an area overhead is tolerated, as. These operators are described in details in sections 4 and 5.

4. The $\text{CL}^2\text{C}/\text{Adder}$

In stochastic adders, given two bit streams $S_1$ and $S_2$, with associated probabilities $p_1$ and $p_2$, that represent the values to be added, and a third bit stream, $S_3$, used to determine which of the inputs will be transferred to the output, it can be shown [11] that a circuit implemented with a 2:1 multiplexor can be used to generate a bit stream representing the sum, with associated probability $p_s$, so that

$$p_s = 0.5p_1 + 0.5(1-p_3)p_2 \quad (1)$$

The $S_3$ input to the circuit has an associated probability ($p_3$) independent of those of the other two input bit streams and is regarded as a weight [11]. As a consequence, in addition operations, when the function used to generate the random numbers that result in input $S_3$ does not cover all possible combinations, there is an approximation error that decreases as the number of comparisons grows. Since each comparison generates one bit in the stochastic bit stream, and this is done serially, in order to obtain more precise results it is necessary to increase the number of cycles, i.e., spend more time computing the values.

However, when $p_3 = 0.5$ (which can be obtained simply alternating 1s and 0s in the $S_3$ input), expression (1) becomes

$$p_s = 0.5p_1 + 0.5p_2$$

and, therefore,

$$p_s = 0.5(p_1 + p_2),$$

which means that, when $p_3 = 0.5$, a counter, connected to the output of the multiplexor, that is incremented whenever a bit equal to 1 is read, recovers a value $V$ that is half the sum of addends used to generate the $S_1$ and $S_2$ bit streams. Therefore, one must multiply $V$ by 2 to calculate the correct sum, which always leads to an even result. As a consequence, if the correct sum is an even value, the exact sum of the original addends is obtained. However, when the sum is an odd value, the result has an intrinsic error of $\pm 1$.

This adder, compared with the stochastic one, supplies more precise results with a limited number of samples, provided that all the possible values with a given bit width are supplied to the comparator, in any order - not necessarily random. This condition can be fulfilled with the use of an $n$-bit counter dimensioned to generate all the possible integer values in the interval $[0, 2^n-1]$. Therefore, a counter has been used to generate the values to be compared with the addends in our design.

Such an adder is far simpler than the conventional adder used in digital computers. This could be used in multimedia applications to increase the number of adders, since there is enough parallelism in this kind of application to be explored by massively parallel circuits. Besides that, the complexity of the adder circuit is independent of the magnitude of the values to be added.

However, when the input to the adder is already a bit stream in which the numbers of 1s represent the value of the operand, the addition operation can be implemented in a very straightforward way, by simply selecting bits from the input streams alternately, instead of using the multiplexor. Also, in order to multiply the resulting bit stream by two, it is enough to duplicate each bit, once again without using any special circuit for that besides connections.

We now have a very simple scheme to add bit streams, whose only drawback is an error by $+1$ or $-1$ when the sum is and odd value. That solution is useful in the implementation of certain kinds of applications, where the outputs of multipliers that generate products in the form of bit streams must be added.

Moreover, despite being not completely error free, the behavior of this adder in face of multiple faults in the bit stream is quite tolerant, mainly for applications that support small deviations in the results of series of additions. In order to show that, we have simulated the injection of multiple simultaneous faults in the bit streams.

For bit streams with 1023 bits, up to eight simultaneous bit flips have been simulated, i.e., the complement of the correct bit value has been generated for 8 of 1023 bits, randomly selected within the bit stream. Table 1 shows the average error percentage in additions, using conventional operations (Convent.), and compares it with the use of the adder without faults (0 faults) and with injected simultaneous faults affecting different gates of the circuit (Robust adder, with 0, 2, 4 and 8 faults).

For this experiment, the same pair of values has been added 1000 times using each adder (except for the conventional one). For the comparisons, values
have been generated by a software function, simulating one counter.

### Table 1. Errors in addition

<table>
<thead>
<tr>
<th></th>
<th>Robust Adder</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Convent.</td>
</tr>
<tr>
<td></td>
<td>0.0000 %</td>
</tr>
</tbody>
</table>

As expected, for the experiments using the adder, the results have a small error, below 0.3%. However, the injection of faults (forcing 2, 4 or 8 bit flips during the calculation of each sum) resulted in no significant changes in the error percentage, confirming our initial assumptions.

5. The CL$^2$C/Multiplier

Using the basic idea of representing the product by a stream of bits in which the number of 1s is related to the value of the product, we have designed a non stochastic device that generates exact products and, when prepared to be fault tolerant to multiple simultaneous upsets, gives no errors in case an overhead in the bit stream representation is accepted.

The basic principle behind the design of the multiplication operator is to generate a bit stream with exactly the same quantity of bits equal to 1 as the product being calculated. In order to simplify the explanation of our solution, let us suppose that the multiplication factors, named F1 and F2, are only 3-bit wide, and that their bits are numbered as 0, 1 and 2, from the least significant to the most significant one. In this case, the multiplication is carried out according to the algorithm shown in Figure 1, where each bit of the product ($P_i$) is obtained by the addition of the partial products $F_{j} \times F_{k}$ that are aligned above it in the figure.

Each partial product is calculated using a single AND gate. The bits that form each partial product have different weights, depending on their position: the bit resulting from $F_{20} \cdot F_{10}$ has weight 1 ($2^0$), those from $F_{20} \cdot F_{11}$ and $F_{21} \cdot F_{10}$ have weight 2 ($2^1$), and so on, until the result from $F_{22} \cdot F_{12}$, which has weight 16 ($2^4 = 2^{2n-2}$, where $n$ is the number of bits of each factor).

**Figure 1. Multiplication algorithm**

Now, instead of adding the partial products, as it is usually done, we generate a stream of bits, containing as many bits for each partial product as the weight of that specific partial product. This means that the partial product with weight 1 generates only 1 bit (with value 0 or 1) in the final bit stream that will represent the product, each of the partial products with weight 2 generates 2 bits (total of 4 bits) in the stream, and so on, until the partial product with weight 16, which generates 16 bits in the bit stream, totaling $1 + 4 + 12 + 16 + 16 = 49$ bits in the final stream.

Note that, when all the generated partial products are equal to 1, this means that we are multiplying $7_{10} \times 7_{10}$ (or $1112_{2} \times 1112_{2}$), whose product is $49_{10}$ and, therefore, all bits of the output stream should be equal to 1. Accordingly, when we multiply 0 x 0, all the 49 bits of the stream will be equal to 0.

The gate network required to implement this algorithm would then have 6 inputs (one for each bit of the factors) and 49 outputs (one for each bit of the bit stream).

We could use only one AND gate for each partial product (total of 9 gates in our example), in order to reduce the area required by the circuit, and connect the output of each gate to as many outputs of the circuit as necessary to achieve the total amount of bits in the stream. However, our main goal is to achieve robustness against soft errors, and we therefore trade area for fault tolerance, using one AND gate to generate each output bit, which means 49 gates, many of them connected to the same partial product bits and, therefore, generating the same value, as explained above. This redundancy is one of the key aspects of the multiplier design, but it can be reduced to different intermediate levels, depending on the area $\times$ tolerance requirements of a project. As an example, we could connect up to two outputs of the multiplier to the same AND gate, thereby reducing the area of the device, but also the robustness, as explained in the next paragraph.

In the above example, this would reduce the total of AND gates from 49 to 25 ($1+2+6+8+8$), and for wider factors the area reduction becomes even more significant.

The second feature that increases the robustness of the multiplier operator is the addition of redundant bits in the product. This could be done by multiplying each factor by 2, using a 1-bit shift left, before generating the product bit stream, which leads to a bit stream 4 times longer than the necessary to represent the original product. It must be noted that this shift operation for the input operands is virtual, since it can be implemented only through the adequate connections of the input bits to the circuit components, without adding any device to the circuit. As an example, in order to multiply two 7-bit positive integers, each of them is virtually multiplied by 2 by a shift left, resulting in 8 bits for each factor.

However, the product bit stream would then have 65,025 significant bits (to support products up to 255 x 255), instead of 16,129 bits (to support products up to}
127 x 127), which means an increase in the number of outputs of the multiplier. Again, the area penalty can be adjusted according to the area x tolerance requirements of each application.

Since the concept of using bit streams to represent integer values requires a conversion into a binary value, by counting the number of bits equal to 1 at the end of the operations, the converted product would then be 4 times larger than the exact result. In order to adjust the product, this count must be divided by four, using a 2-bit shift right. This operation can be easily implemented through the adequate connections of the counter bits 2 to 15 to the outputs 0 to 13 of the conversion circuit. However, since bit streams are the key to fault tolerance, in applications where this is possible, their conversion to binary codes should be postponed as much as possible in the calculation.

5.1. Why are bit streams more tolerant to faults than binary codes?

For the purpose of the following discussion, let us call "positive flips" the change of bits from 0 to 1 and "negative flips" the change from 1 to 0, and, as an example, let us imagine the multiplication of two 7-bit integers, with values a and b, respectively.

If there was no concern about bit flips, the multiplication circuit could then multiply the two 7-bit values and generate a bit stream with a total of 16,129 bits, with $a \times b$ bits equal to 1 and 16,129 - $a \times b$ bits equal to 0.

Even though no overhead was added to prevent soft faults, the bit stream approach already gives protection against an even number of multiple faults in which the number of positive and negative flips is the same. In those cases, no matter how many flips occur, the results will always be correct, since each positive flip compensates one negative flip, keeping the total number of ones unchanged. This is already an advantage when compared with conventional parity schemes.

However, if the number of positive and negative bit flips is different, then the count of bits equal to 1 would change, leading to an error in the interpretation of the product value. This unit error may be neglectable for high product values, but may become important for small product values (e.g., an error by 1 in a product equal to 1 represents 100%).

In order to avoid this error, it is necessary to add some redundancy to the bit stream, multiplying each factor by two before the multiplication, which results in two 8-bit numbers with values $2a$ and $2b$, respectively, according to the above described algorithm. Those values would then be processed by the multiplication circuit, that would generate a bit stream with 65,025 bits, with $2a \times 2b$, or $4ab$ bits equal to 1 and 65,025 - $4ab$ bits equal to 0. This means that our product bit stream now has 4 times the number of ones that are required to represent the exact product. Consequently, at the end of the calculations the count of ones should be divided by four in order to recover the correct result.

Now, let us suppose that, besides any quantity of positive and negative flips pairs, a balance of up to three positive flips remains. This results in a total of up to $4ab + 3$ ones and 65,025 - $4ab - 3$ zeroes in the bit stream. Despite the three faults, when the number of ones is counted and divided by four, at the end of the calculation, the recovered result will still be $(4ab + 3)/4$, which is equal to $ab$ (the exact product) with a remainder of 3 (that is discarded). Similar reasoning can be used to show that, if the balance of flips is negative, from 1 to 4, this will result in an error by 1 in the final product, since $(4ab - 4)/4$ is equal to $ab - 1$. As already mentioned, this unit error can become important, depending on the product value.

Let us now suppose that the exact product can be represented by a binary number $\mathbf{x}...\mathbf{x}$, where each $\mathbf{x}$ can be a 0 or a 1. The value of this number is the count of bits equal to 1 in the corresponding product bit stream. When the product is multiplied by four, that binary number becomes $\mathbf{x}...\mathbf{00}$. Therefore, when up to three bits equal to 1 are added to the stream, only the two least significant digits of $\mathbf{x}...\mathbf{00}$ are affected. However, if only one bit equal to 1 flips to 0, there is a borrow from the 00 part that subtracts 1 from the $\mathbf{x}...\mathbf{x}$ part of the number.

This reasoning can be generalized to any case in which $r$ redundant bits are appended to the product value ($\mathbf{x}...\mathbf{x}$), thereby multiplying it by $2^r$. It can be shown that no matter how many redundant bits are appended, the bit stream will always be tolerant only to positive balances of up to $2^r$-1 flips and that a negative balance of 1 will be enough to produce a wrong product.

To solve this problem, one can add an "excess" of $2^{r-1}$ to the value. In the previous example, this would change the value of $\mathbf{x}...\mathbf{00}$ to $\mathbf{x}...\mathbf{10}$, and now the bit stream would be tolerant to a maximum positive balance of 1, but would tolerate negative balances up to 2, without any influence in the final product. Generalizing again, appending $r$ redundant bits to $\mathbf{x}...\mathbf{x}$ and adding $2^{r-1}$ to the result, we always get a value represented by $\mathbf{x}...\mathbf{1}$ followed by $r-1$ zeros, and the corresponding bit stream will be tolerant to balances of up to $2^{r-1}$-1 positive flips or up to $2^r$ negative flips.
Table 2 shows the balances of positive or negative flips that are tolerated by the bit streams, according to the number of redundant bits used. Note that these numbers do not depend on the width of the input operands (factors), but only on the number of redundant bits appended to the product.

<table>
<thead>
<tr>
<th>number of redundant bits</th>
<th>balance of positive flips</th>
<th>balance of negative flips</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>$r$</td>
<td>$2^{r-1} - 1$</td>
<td>$2^{r-1}$</td>
</tr>
</tbody>
</table>

It is clear that the error, as a percentage of the correct product, decreases for higher product values and increases for smaller product values. Therefore, if the application is too critical and additional area overhead is still acceptable, multiplying the factors by $2^r$ before generating the bit stream produces exact results when balances of up to $2^{r-1} - 1$ positive flips or $2^{r-1}$ negative flips occur.

6. Conclusions and Future Work

This paper has introduced a new approach to deal with soft errors, whose incidence will certainly grow as the VLSI technology migrates towards deep sub-micron dimensions, which consists on the use of bit stream computation operators, instead of conventional digital ones.

The experiments conducted with simulation of adders and multipliers have confirmed the initial assumptions and encourage further investigation towards the design of general purpose operators which will be tolerant to multiple faults.

Therefore, the next steps in this research will be the exploration of the design space for new implementations of those operators, stressing parallelism in order to reduce the time required to perform the operations. Our immediate goal is the implementation of an application circuit using standard cells and full custom designs, in order to compare the real area requirements and their speed with those of the same circuit using conventional fault tolerance schemes, such as TMR. In the long term, we aim to produce some chips using the proposed operators and test their behavior when submitted to radiation.

7. References


