On the Computation of Approximation Coefficients in ROM-Based Redundancy for SEU Mitigation on FPGAs

Rui Policarpo Duarte
INESC-ID, IST/U. Lisboa

Mário Véstias
INESC-ID, ISEL/P. Lisboa

Horácio Neto
INESC-ID, IST/U. Lisboa

***Abstract*—Many circuit designs implemented on FPGAs with high-throughput and low-latency requirements don't tolerate spending extra clock cycles waiting for a recovery from an SEU, and hence, often trading-off precision for a sustained, but reliable, throughput. ROM-based RPR makes it possible to meet such performance requirements by detecting and correcting SEUs using a low-latency redundant component producing approximate results from a ROM in parallel with the unit to be protected. Thus far, the contents of such ROM depend solely on a brute-force computation of the approximation values that minimize the objective function for the protected unit. To alleviate such limitations, this work extends the state-of-the-art on this topic by presenting a novel method to formulate and calculate analytically the coefficients of the approximation functions. A comparison with previous work is made to validate the new approach.**

I. Introduction

Reduced Precision Redundancy (RPR) was first presented by [1] as a technique to control errors in voltage scaling designs, and extended to mitigate errors for other applications in [2]–[4]. It performs the same computations as the unit intended to protect using an extra unit connected in parallel, but computing less information. Even though it is not an accurate result, it is *good enough* to determine if the result of the complete/original unit is within a specific threshold. Alternatives to recover, or minimize, Single Event Upsets (SEUs) are: Algorithmic Noise Tolerance (ANT) [5]–[7] and Triple-Modular Redundancy (TMR) [8]. Other research directions such as approximate computing [9]–[12], which are often applied to applications where computations and design parameters are relaxed to minimize the impact of errors.

Figure 1 shows the RPR architecture, comprised of an arithmetic operator (OP), and a smaller version computed in parallel (OP_{RED}) using less bits of data. This architecture imposes a halt on the datapath of, at least, one clock cycle. Recently, [13] proposed a variation of RPR which implements the redundant unit based on a Read-Only Memory (ROM) without the extra clock cycle required for the evaluation step, and therefore it corrects errors without extra clock cycles.

Notwithstanding, the framework proposed for ROM-based RPR, which produces the content for the ROMs, does so by computing a linear approximation for the arithmetic operations to be protected. Insofar, the coefficients for the approximations are retrieved via brute-force algorithm, which doesn't scale well with wordlength increase. In addition to that, in an attempt to save processing time, in [13], the search process for the approximation coefficients stops as soon as a combination of parameters satisfies the objective function, thus not identifying the remaining valid values. This work proves that there are parameters that perform better than the ones provided by previous work. Field-Programmable Gate Arrays (FPGAs) devices with embedded RAM blocks in their structure which are suitable to contain the ROM contents.

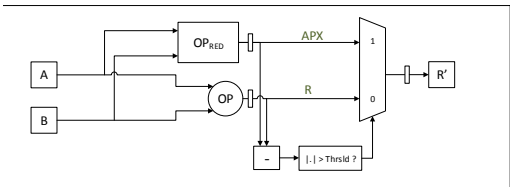

Fig. 1. Simplified block diagram for the typical RPR.

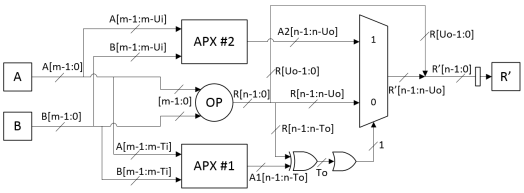

Fig. 2. Simplified block diagram for the ROM-based RPR.

A. ROM-Based Reduced-Precision Redundancy

The ROM-Based RPR variant, which is based on ROMs to hold the *reduced* results, to minimize the latency and hence avoid introducing extra registers in the datapath. Moreover, to further reduce the latency, the detection of errors in the results is determined via a bitwise comparison of both outputs. A bitwise logic XOR followed by a logic OR, diagnoses differences on the Most Significant Bits (MSbs) to be protected from SEUs.

Like previous RPR architectures, this one also encapsulates the original arithmetic operator, so that it is instantiated in the design without changing the interface or the datapath design. Figure 2 presents the block diagram of the ROM-based RPR. Here, the arithmetic operation is computed as intended, and two redundant approximations are computed in parallel. One approximation ROM ($APX\#1$) is used to detect mismatches in the protected MSbs, and the other ($APX\#2$) to replace MSbs at the output if necessary. Both approximations can have any wordlength at their inputs and outputs. The

approximation functions for each ROM must be different, so that a correction can occur. Otherwise, it would act as if only one approximation/ROM was being used. In this RPR variant, the Least Significant Bits (LSbs) at the output are the results of OP, and never replaced with an approximation, as the magnitude of an error on the LSbs is less than any MSbs.

In essence, the first ROM tries to approximate correctly the maximum number of MSbs, and the second ROM *compensates* any incorrectness in the MSBs (due to lack of precision in the first ROM or an SEU). In practice, a truncated approximation always shows a negative bias, therefore it misses most the comparisons for the MSbs. However, an approximation with a positive bias is *closer* to the correct values of the MSbs. In a positively biased approximation, in some occasions it produces excess errors, which a *not so biased* approximation, as the truncated, may hold correct the correct value for the MSbs. The magnitude of a deviation between the correct value and an approximation is dictated by the number of significant bits in the representation, as expected.

II. Mathematical Formulation

Assuming an input argument of a generic arithmetic function, coded in binary using N bits, it is expressed as the sum of all bits weighted by their corresponding power of 2: $A = \sum_{i=1}^{N} a_i.2^{-i}, A \in [0,1[$. The values of the MSbs of A are represented as the contribution of the Ti left-most bits, and are expressed as: $A_{MSb} = \sum_{i=1}^{Ti} Ai.2^{-i}$.

For the same variable A, its MSbs are also expressed as the integer part of the division of A by 2^{Ti}, and the truncation operation. The floor function is used to isolate the MSbs, resulting in: $A_{MSb} = \left\lfloor \frac{A}{2^{Ti}} \right\rfloor = trunc(A, 2^{Ti})$. Similarly, the LSbs are the reminder of the modulo operator, or the difference between the variable and its MSbs: $A_{LSb} = mod(A, 2^{Ti})$ and $A = 2^{Ti}.A_{MSb} + A_{LSb}$ provides: $A_{LSb} = A - A_{MSb}.2^{Ti} = A - 2^{Ti}.\left\lfloor \frac{A}{2^{Ti}} \right\rfloor$. For simplicity, the magnitude of the reminder for a truncated variable in its Ti MSbs is defined as: $R_{Ti} = 2^{-Ti-1}$.

The floor operator applied to an integer variable x, representing a binary value, its integer part is defined as: $x_{int} + \varepsilon$, $\lfloor x \rfloor = x_{int}$ and $x = x_{int} + \varepsilon, \varepsilon \in [0, 2^{-ulp}[$. This formulation is also applicable to operations: $\lfloor x + y \rfloor - \lfloor x \rfloor + \lfloor y \rfloor \leq 2^{-Ti}$. ulp is the least bit of information in the decimal representation.

Adopting the previous formulation to the approximation of the MSbs for a generic function $\star$, being $A_{Apx} = ((A_{MSb} + M)$ and $B_{Apx} = ((B_{MSb} + M)$. The approximation $(A \star B)_{MSb} = (A_{Apx} \star B_{Apx} + L)_{MSB}$ is described by: $\left\lfloor \frac{A \star B}{2^{To}} \right\rfloor = \left\lfloor \frac{A_{MSB} \star B_{MSB} + L}{2^{To}} \right\rfloor$.

III. Parameter Estimation

A. Addition

To compute the approximation for the addition operation, it consists of replacing the $\star$ operator with $+$:$(A + B)_{MSb} = ((A_{MSb} + M) + (B_{MSb} + M) + L)_{MSb}$. For simplicity, M is ignored since its contribution is compensated by L. The main objective is finding a relation between the approximation wordlengths and L which minimizes the errors in the approximation's MSbs. The MSbs of $A + B$ are given by:$(A+B)_{MSb} = \left\lfloor \frac{A+B}{2^{To}} \right\rfloor$ The MSbs of the expected result are now related to the MSbs of the approximation result with truncated inputs and parameter L: $\left\lfloor \frac{A+B}{2^{To}} \right\rfloor = \left\lfloor \frac{A_{MSB}+B_{MSB}+L}{2^{To}} \right\rfloor$.

Solving the expression it gives: $\frac{A+B}{2^{To}} - R_{To} = \frac{\frac{A+B}{2^{Ti}} - 2R_{Ti} + L}{2^{To}} - R_{To}$. The $2^{Ti/o}$ over a variable means that only the Ti/o most significant bits are considered: $\frac{A+B}{2^{To}} = \frac{A+B}{2^{Ti+To}} - \frac{2R_{Ti}}{2^{To}} + \frac{L}{2^{To}}$. As the interest is on compensating the LSbs, the MSbs truncation is discarded: $A + B = \frac{A+B-1}{2^{Ti}} + L$.

On the left side of the equation there's the result of the operation with the complete operands. On the right there are the truncated operands being added with L. To have the same information of the MSbs on both sides, L has to *fill in* the truncated bits and the constant value being subtracted, towards the least significant bit in its representation. This is the equivalent, to fill in the LSbs with 1 and then add 1, to promote the carry propagation over the MSbs of truncated operands. The LSb of the input is 2^{-N}, therefore the value to be incremented is obtained via $L = 2^{N-Ti}$.

B. Multiplication

The procedure for multiplication is the same as for the sum. As a matter of a fact, it is the same for any other arithmetic expression. The MSbs of the expected result with the ones from the approximation are related by the expression: $(A \times B)_{MSB} = (A_{Apx} \times B_{Apx} + L)_{MSB}$. Solving for the left side: $\left\lfloor \frac{A \times B}{2^{To}} \right\rfloor = \frac{A \times B}{2^{To}} - \varepsilon = \frac{A \times B}{2^{To}} - R_{To}$. The truncated operands A_{Apx} and B_{Apx} are simplified: $A_{Apx} = A_{MSB} + M = \left\lfloor \frac{A}{2^{Ti}} \right\rfloor + M = \frac{A}{2^{Ti}} - R_{Ti} + M$. Replacing both sides of the equation results in: $\frac{A \times B}{2^{To}} - R_{To} = \left\lfloor \frac{A_{Apx} \times B_{Apx} + L}{2^{To}} \right\rfloor - R_{To}$. Which simplifies to: $\frac{A \times B}{2^{To}} = \left\lfloor (\frac{A}{2^{Ti}} - R_{Ti} + M) \times (\frac{B}{2^{Ti}} - R_{Ti} + M) + L \right\rfloor$. Once the coefficient M relates with the input operand, it is possible to determine its value: $\frac{A \times B}{2^{To}} = \frac{(\frac{A}{2^{Ti}} - R_{Ti} + M) \times (\frac{B}{2^{Ti}} - R_{Ti} + M) + L}{2^{To}}$. Similarly to the addition operation, the information missing for each truncated argument corresponds to the value immediately below the first MSb, or in other way, all the LSbs. So, finally: $M = 2^{N-Ti} - 1$.

IV. Evaluation

Tests were conducted by generating several combinations of input and output wordlengths for the sum and multiplication operators. It was observable that the values produced by the previous framework were within the set of values from the mathematical expression. Moreover, for the same wordlengths, the proposed formulation offered less MSb corrections to produce the same correct results.

V. Conclusion

This work proposes a novel methodology to obtain optimal parameters for the generation of the approximations for ROM-based RPR. Besides improvement in performance of the framework, the new method produces better approximations.

ACKNOWLEDGMENTS

This work was supported by national funds through Fundao para a Cincia e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

REFERENCES

[1] B. Shim and N. Shanbhag, "Reduced precision redundancy for low-power digital filtering," in *Signals, Systems and Computers, 2001. Conference Record of the Thirty-Fifth Asilomar Conference on*, vol. 1, 2001, pp. 148–152 vol.1.

[2] B. Shim, S. Sridhara, and N. Shanbhag, "Reliable low-power digital signal processing via reduced precision redundancy," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, no. 5, pp. 497 –510, may 2004.

[3] G. Gaubatz and B. Sunar, "Robust finite field arithmetic for fault-tolerant public-key cryptography," in *Fault Diagnosis and Tolerance in Cryptography*, ser. Lecture Notes in Computer Science, L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, Eds. Springer Berlin / Heidelberg, 2006, vol. 4236, pp. 196–210, 10.1007/11889700_18. [Online]. Available: http://dx.doi.org/10.1007/11889700_18

[4] B. Pratt, M. Fuller, and M. Wirthlin, "Reduced-precision redundancy on FPGAs," *Int. J. Reconfig. Comput.*, vol. 2011, pp. 3:3–3:3, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1155/2011/897189

[5] R. Hegde and N. R. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *Proceedings of the 1999 international symposium on Low power electronics and design*, ser. ISLPED '99. New York, NY, USA: ACM, 1999, pp. 30–35. [Online]. Available: http://doi.acm.org/10.1145/313817.313834

[6] E. P. Kim and N. R. Shanbhag, "Statistical analysis of algorithmic noise tolerance," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 2731–2735.

[7] D. Seo and L. R. Varshney, "Information-theoretic limits of algorithmic noise tolerance," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, Oct 2016, pp. 1–4.

[8] F. L. Kastensmidt, L. Sterpone, L. Carro, and M. S. Reorda, "On the optimal design of triple modular redundancy logic for sram-based fpgas," in *Design, Automation and Test in Europe*, March 2005, pp. 1290–1295 Vol. 2.

[9] A. Agrawal, J. Choi, K. Gopalakrishnan, S. Gupta, R. Nair, J. Oh, D. A. Prener, S. Shukla, V. Srinivasan, and Z. Sura, "Approximate computing: Challenges and opportunities," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, Oct 2016, pp. 1–8.

[10] P. K. Krause and I. Polian, "Adaptive voltage over-scaling for resilient applications," in *Proc. Design, Automation & Test in Europe Conf. & Exhibition (DATE)*, 2011, pp. 1–6.

[11] D. Roberts, T. Austin, D. Blauww, T. Mudge, and K. Flautner, "Error analysis for the support of robust voltage scaling," in *Proc. Sixth Int. Symp. Quality of Electronic Design ISQED 2005*, 2005, pp. 65–70.

[12] M. A. Breuer, "Adaptive computers," *Information and Control*, vol. 11, no. 4, pp. 402–422, 1967.

[13] R. P. Duarte and C. S. Bouganis, "Zero-latency datapath error correction framework for over-clocking dsp applications on fpgas," in *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14)*, Dec 2014, pp. 1–7.