Approximate computing design exploration through data lifetime metrics

Alessandro Savino[1], Michele Portolan [1 2], Regis Leveugle [2], Stefano Di Carlo[1]

[1] Dipartimento di Automatica ed Informatica
Politecnico di Torino, Torino, Italy
{name.surname}@polito.it

[2] Univ. Grenoble Alpes, CNRS, Grenoble INP[1],
TIMA, 38000 Grenoble, France
{name.surname}@univ-grenoble-alpes.fr

***Abstract*— When designing an approximate computing system, the selection of the resources to modify is key. It is important that the error introduced in the system remains reasonable, but the size of the design exploration space can make this extremely difficult. In this paper, we propose to exploit a new metric for this selection: data lifetime. The concept comes from the field of reliability, where it can guide selective hardening: the more often a resource handles "live" data, the more critical it becomes, the more important it will be to protect it. In this paper, we propose to use this same metric in a new way: identify the less critical resources as approximation targets in order to minimize the impact on the global system behavior and therefore decrease the impact of approximation while increasing gains on other criteria.**

Keywords: Approximate Computing, Design Space Exploration, Error metrics

I. INTRODUCTION

In recent years, the new Approximate Computing (AC) paradigm represented a breakthrough for several application domains. In domains where quality of results depend on the human perception (e.g., image processing) or algorithms are inherently resilient to errors (e.g., machine learning) [1][2], approximate results are often hard to distinguish from exact ones. Thereby, designers can efficiently trade-off accuracy of the results with other design parameters such as performance, memory requirements and power consumption.

The main effort of the research community has focused so far on the design of efficient approximate functions and operators [3]. However, all approaches require choosing where, how, and when approximation can be applied. This selection is a complex task since it requires the exploration of a large design space. The set of computations that can be approximated is huge, and it is extremely difficult to predict the effect of the approximation on the precision of the final result [4]. Currently, only few publications addressed the problem of modelling the error propagation in AC applications, predicting the final effect on the outcomes without requiring large simulation campaigns [5] [6]. However, although these approaches solve the problem of estimating the impact of the approximation on the precision of the application, they still do not provide a strategy to select the best subset of resources to approximate.

This paper takes a step forward proposing a technique able to identify a subset of resources that can be approximated while minimizing the impact on the precision of the final output. This significantly reduces the size of the design space to be analyzed.

The proposed approach takes inspiration from the reliability domain. From the reliability standpoint, each hardware/software resource contributes to the reliability level of the system. This contribution is a function of several parameters (architecture, workload, etc.) and is extremely difficult to estimate. In this field, some metrics have been developed to identify critical resources and support efficient design space exploration [7][8][9][10][11]. In this paper we propose to exploit Register Data Lifetime (RDL) [8], i.e., the interval between the time a data is written into a register and the last time it is read, as a candidate metric for design space exploration in an AC context, able to identify those resources that can be approximated with minor impact on system-level precision. The motivation behind this proposal is that a longer life-time is in general related to a larger number of operations using the same value, and data usage is one of the key factors affecting the precision of a program in presence of approximation. This is somehow different from the use of the RDL in the reliability domain, in which the assumption is that more faults may occur during a larger storage time. Nevertheless, even if the mechanism is different the effect is similar and AC can take advantage from available techniques and tools. The proposed technique has been tested on a set of typical benchmarks used to evaluate AC techniques, providing interesting results showing that the proposed approach can be an interesting starting point for the development of a new class of resource approximability metrics.

The paper is organized as follows: first, it discusses related literature in the approximate and reliability domains (Section II), then it introduces in detail the proposed approach (Section III). Section IV provides an experimental proof of concept, while Section V draws conclusions and introduces future perspectives.

II. STATE OF THE ART

A. Approximation strategies

Approximate computing techniques can mainly be grouped in three categories: (i) design of efficient approximate hardware blocks [12], (ii) introduction of errors in the control flow of the application [13] or (iii) trade-off between data size and precision [14].

When looking into approximate hardware components, several publications focus on adders and multipliers [15]. The basic idea is to introduce an approximation by: (i) reducing the carry propagation chain, (ii) dividing the adder into sub-adders then

[1] Institute of Engineering Univ. Grenoble Alpes

generating carries using different methods, (iii) playing with full-adders' modifications. Similar concepts are applied to multipliers, where the approximation may be introduced by: (i) generating partial products (ii) ignoring some partial products (iii) using approximate counters or compressors in the partial product tree, (iv) composing complex approximate multipliers by means of simple approximate / correct multipliers. For both approximate adders and multipliers, the result of the operation is affected by an error that depends on the input data and can be measured using different metrics such as the Mean Relative Error (MRE) or the Worst-Case Error (WCE), and Error Probability (EP), i.e., the probability that the approximation affects the output. All strategies reduce area, power consumption, and can improve the computation throughput.

When hardware approximation cannot be exploited, it is still possible to approximate by reducing the algorithm execution time through loop perforation. In this case, selected loops can be executed less times (by skipping some counter increment or jumping out of the loop on certain conditions), exploiting the ability of the computation to converge on a near good result in a smaller number of steps. This approach still enables to trade-off power consumption and performance while the hardware area remains constant.

By tuning the data size (e.g., by considering only the original most significant bits or by replacing floating-point data with fixed-point data [16], or leveraging the data precision [14]) the hardware area, the computation time and the power consumption can also be reduced. This strategy is well suited for several classical DSP applications (e.g., Finite Impulse Response - FIR, Infinite Impulse Response - IIR filters, Fast Fourier Transform (FFT), and machine learning).

B. Reliability Estimation

Another source of errors in computations is due in many applications to environmental disturbances, due to e.g., particles, electromagnetic interferences, or voltage variations that may result in erroneous data or so-called "Soft Errors".

A lot of work has been done to obtain estimations of the effect of such errors early in design flow. Replacing Radiation Testing with RT-Level fault injections is a first step [17], but these setups are still too expensive in terms of computational complexity [18], even when hardware acceleration or emulation platforms are used [18], and they require the RTL model to be extremely precise. Several approaches have been proposed to elevate the abstraction of the problem, most of the time relying on statistical modelling of the software [19][20].

Among the different approaches proposed in the literature, the approach proposed in [8], in which Register Data Lifetime is used for reliability estimation is of particular interest for this paper. The overall idea is simple: the longer a data is resident in a register, the higher is the probability of it being corrupted. The application is actually more complex, as there are many subtle points to consider, such as the actual contribution of the data to the final result, or masking effects introduced by the micro-architecture. As systems get bigger, these factors become especially important to avoid taking into account data that will have no effect on the final output [9].

Figure 1 depicts the general principle. Figure 1-a shows a generic RTL description of a circuit composed of a set of storage elements (registers, marked as "R"), connected by a network of combinatorial functions (functions, marked as "F"). Figure 1-b depicts the effect of a Single Event Upset (i.e., single bit-flip) on two different registers. In one case, the fault propagates through the system until it reaches the final output, corrupting it and generating an error. In the second case, the fault is filtered by the combinatorial logic, and has no effect on the final output. This means that the first fault is more critical. Therefore, in case of selective hardening it would be more important to protect the first register rather than the second one in case of an error at this particular clock cycle. Modern systems may be composed of a huge number of registers, so it is not always feasible or cost-effective to harden them all. In the same way an error due to a computation error introduced by approximation will not have the same effect at each cycle and at each location.

A possible solution is to identify the subset of registers that are more critical for the reliability of a given application, so that the designer knows where to focus his efforts. To obtain usable results, the method proposed in [8] developed a detailed model of a target processor microarchitecture, notably taking into account the processor pipeline structure and mechanisms like fast-forwarding. The approach provided consistent results when compared with Fault Injection campaigns and was successfully expanded to generic digital circuits with no assumption on the architecture [9].

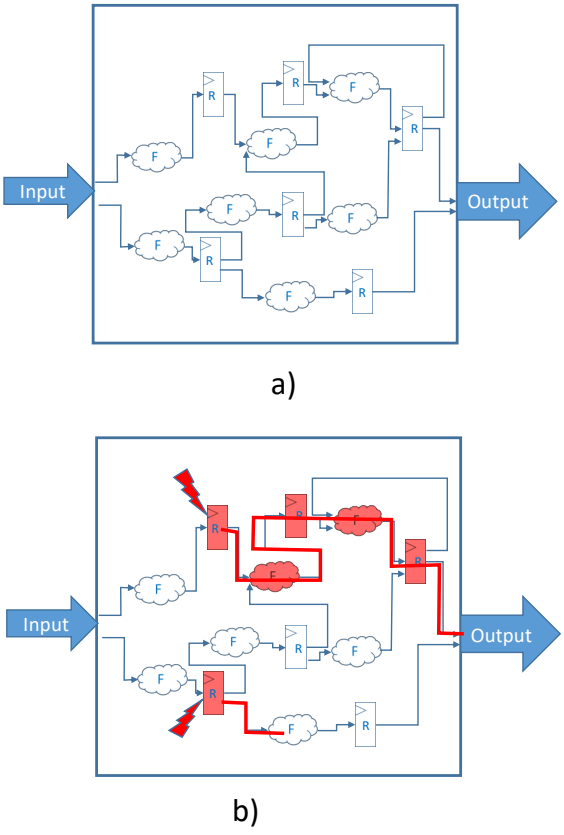

Figure 1 Design Space Exploration for Reliability

III. PROPOSED APPROACH

To present the proposed approach for the identification of candidate resources where to apply approximation, let us start from the same example reported in Figure 1-a. When approximating, one or more of the Functions F_i, considered as "Precise", are replaced by corresponding Approximate Function (AF_i), thus introducing an error e_i at the output of the circuit. Figure 2-b shows an example, where the original Functions are labelled as PF (Precise Function) for better readability. If a different precise function PF_j is replaced by an approximate function AF_j (Figure 2-c) a different approximation error e_j is introduced. The difference comes from the different position and role of the two PF functions in the system, and of course of the impact on the data and control paths.

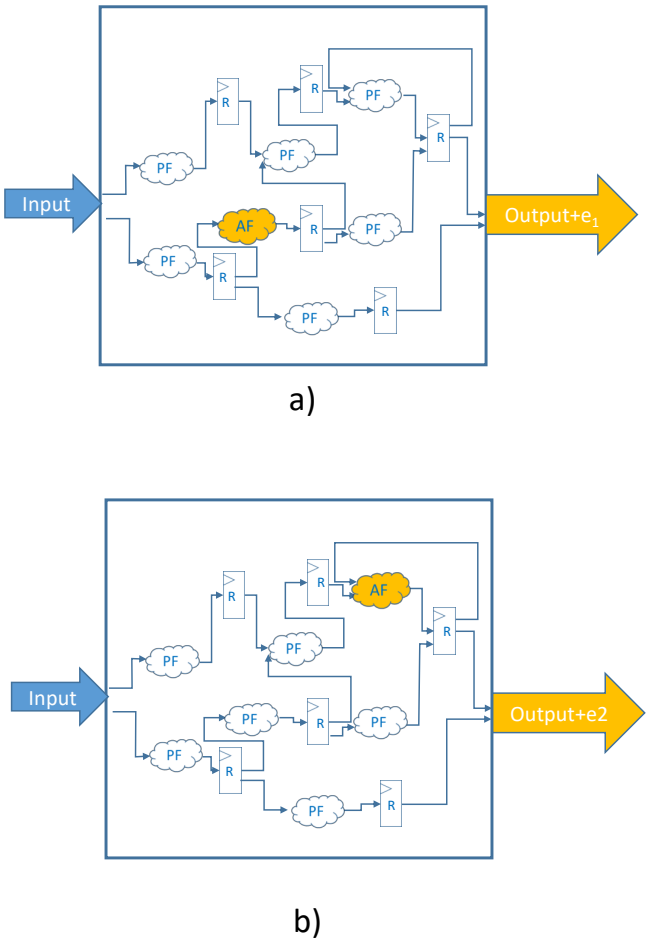

Figure 2 Design Space Exploration for Approximation

This mechanism is extremely similar to the fault propagation mechanism depicted in Figure 1. The basic assumption of the proposed approach is that both the precision of the application in presence of approximation and the reliability of a system in presence of errors (e.g., soft errors) are similar mechanisms depending on the data path and data dependency. Therefore, we would like to exploit a metric such as the Register Data Lifetime (RDL), that has proven to be a valuable instrument to identify critical resources in the reliability domain, to select candidate resources for approximation able to minimize the error at the output of the computation. The main difference when moving from the reliability to the AC domain is that, for reliability, the relationship between RDL and criticality is direct: the more time a data is "alive", the higher is the probability that it can be corrupted by a transient effect. In AC, the focus is rather on the usage of the data, i.e., a high RDL might not be directly related to a significant impact on the precision. However, due to data locality, very often data remain resident for long time when they require to be used multiple times. Therefore, RDL can be exploited as an indirect metric also for the analysis of critical resources in AC applications. However, since RDL is not a direct metric, different metrics strongly related to the data usage must be evaluated. In this paper, the following three different metrics have been considered:

1. the RDL,the ratio between the number of reads and the number of writes *(#R/#W)*.
2. the average period between two reads over the execution time (*AVG #Reads*).
3. the average period between two writes over the execution time (*AVG #Writes*).

To assess our hypothesis, i.e., RDL and other data usage related metrics can be used to identify candidate functions for approximation, the simulation procedure summarized by Algorithm 1 is used. The algorithm considers a target system under investigation (SUI) and computes the above-mentioned metrics for all storage resources.

```
1. Metrics = {RDL, #R/#W,
2.            AVG. #Reads, AVG. #Writes}
3. AC_Policies = {R, W, R&W}
4. SUI = the benchmark to evaluate
5. resources {the set of target resources}
6. foreach m in metric
7.   statistics[m] = compute_metric (SUI,
8.                                   m, resources)
9.   Q = compute_quartiles(statistics[m])
10.  foreach q in Q
11.     foreach p in Policies
12.        SUI_aprx = apply_AF(SUI, q)
13.        collect_data(SUI, SUI_aprx)
14.     endfor
15.  endfor
16. endfor
```

Algorithm 1: Evaluation methodology

The SUI is analyzed for each metric (lines 6-16). A metric can be computed by profiling a golden execution, collecting statistics on the usage of each resource (lines 7 and 8). Resources are then ranked based on the target metric and grouped into four bins (Q vector output of line 9). Figure 3 reports an example of the RDL metric computed for 28 resources. Resources are sorted for increasing RDL and grouped in four quartiles. The assumption here is that overall, resources belonging to the same quartile have similar impact on the precision of the system when approximated.

To prove this, the approximation of all resources in each quartile by means of an approximate operator is evaluated and compared with the result of the reference application (line 10-15).

The approximate version (SUI_aprx) is created from the original version by replacing each PF connected to the resources in the quartile with an AF based on three different approximation strategies: if the PF writes in a target resource (W), or if it reads from a target resource (R), or all the times it writes to or reads from a target resource (R&W).

Collecting data for the comparison between the approximate SUI (SUI_aprx) and the golden reference (SUI) requires simulating the application several times with different input data. For each simulation, input data is randomly generated to assure the results will not be dependent on the testbench.

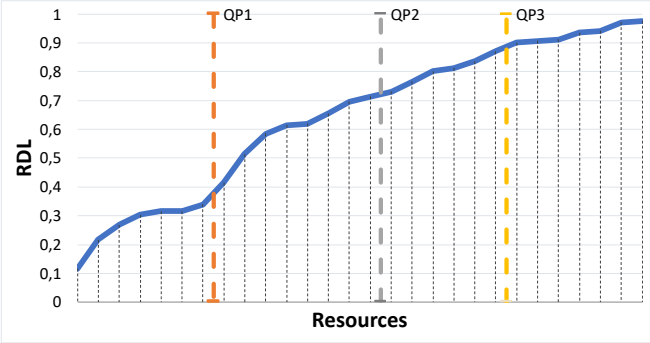

Figure 3 Ordered lifetime with Quartiles Points

By comparing the output of the two versions of the SUI it is possible to compute the error introduced by approximating a given resource and by analyzing the average errors for resources in each quartile we can validate our hypothesis that there should be a clear correlation between quartiles and impact on the precision.

IV. Experimental Results

In order to perform experiments on a set of benchmarks, we developed an experimental proof of concept of the analysis described in Algorithm 1. For simplicity, the target SUI is emulated at the software level. In reference to Figure 2, software variables take the role of storage elements, while arithmetic functions take the role of Precise Functions (PF), for which we also have an Approximate Function (AF) alternative. In particular sums and multiplications are the target functions of our analysis since they have been extensively studied in the AC domain. We considered the software implementation of precise and approximate 8-bit adders and 32-and 8-bits multipliers provided by [15]. For this first preliminary investigation, we selected one approximate adder (with MRE=1.65% WCE=128, and EP=1.6%) and one approximate multiplier (MRE=1.99%, WCE=820, and EP=86.5%). Multiplier results are provided over 16 bits.

Two benchmarks often used in the AC domain were analyzed:

1. Matrix Multiplication. it is a very common computational module used in several real applications including image processing and neural networks employing both multipliers and adders. To keep the analysis time low a 3x3 matrix multiplication was analyzed.
2. Finite Impulse Response (FIR) filter. It is a filter whose impulse response is of finite duration. It is a very common DSP application. Four different 32th-order filters were analyzed: (i) a low-pass filter (LPF), (ii) a high-pass filter (HPF), (iii) a band-pass filter (BPF), and (iv) an arbitrary-pass filter (APF). For the FIR the randomness was applied in the form of white noise generation.

These test cases represent opposite system-level requirements: while Matrix Multiplication computes a precise result and is therefore extremely sensitive to approximation, the FIR algorithms present several shifts and rounding steps, so they are intrinsically more resilient to errors.

The experimental setup follows Algorithm 1. In order to collect data (line 13 of the alg. 1) we implemented these steps:

1. We gerated random values for all inputs. For the FIR, a white noise generation apporach has been followed.
2. Inputs are provided to both the approximate version (SUI_aprx) and the precise version (SUI) and the outputs collected.
3. For each output value, we compute the error (e_i) as the diffence between the precise output and the approximated one. The error is uses to track the frequencies (f_i) of all possibile errors.

To reduce data dependency, we repeat these steps several times, stopping when the frequency distribution function was stable, i.e., the weighted average of the error did not change more than 0.01%. The weighted average is computed using the following equation:

$$E = \frac{\sum_{\forall error}(e_i * f_i)}{\sum_{\forall error} f_i}$$

Where E represents the average error affecting a single output of the computation, e_i is the error (e.g., 0, 1, 2 etc.) and f_i is the frequency at which e_i is detected. Eventually, value E is normalized: the order of magnitude of the error may significantly change from application to application but what is interesting here is the trend. Matrix multiplication implies a sequence of multiply operations, and their results accumulated several times for each index of the output matrix. Thus, once an error is introduced by a multiplication, the accumulation is not able to reduce the error in any case, but just to add another error if an approximate version is used. Similarly, the FIR general behavior is also based on multiplications (between the specific coefficients shaping the filter type and the input samples) and accumulations but, due to some right shift operations over the final value of each output, errors in the lower bits of the results can be discarded.

Figure 4, Figure 5, Figure 6, Figure 7 show the normalized projection of E, for each metric, of all considered applications grouped by policy.

A. Result Feedbacks

Figure 4 shows that the RDL metric with (R) policy is strongly related to the approximated quartile, i.e., resources in Q1 have a lower impact on the accuracy of the system w.r.t. resources in Q4. This is consistent also considering very different applications (e.g., Matrix multiplication and FIR). When RDL is used

in combination with other approximation policies its accuracy is reduced.

When considering the remaining metrics (Figure 5, Figure 6, Figure 7), it is clear that none of them perform well as RDL. This is due to the fact that the approximation depends on data (and the interrelationships among data) but most importantly because each metric depicts a different aspect of the data usage. In this sense, it is an interesting observation that all the other metrics work better when associated with a write policy.

V. Conclusions and Perspectives

In this paper, we showed how Register Data Lifetime can be used as a Design Space Exploration metric in the AC domain. Our experimental results show an interesting correlation between the RDL and the impact of approximation on the final results of a computation. This corroborates our hypothesis and points out several development directions. Overall, the fact that RDL performs better on reading policies while other metrics work better with writing policies suggests that a combination of these metrics could be analyzed in the future to better guide the selection of resources to approximate and to develop AC dedicated metrics.

References

[1] V. K. Chippa, S. T. Chakradhar, K. Roy and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, 2013, pp. 1-9., doi: 10.1145/2463209.2488873

[2] G. S. Rodrigues, F. L. Kastensmidt, V. Pouget and A. Bosio, "Performances VS Reliability: how to exploit Approximate Computing for Safety-Critical applications," 2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS), Platja d'Aro, 2018, pp. 291-294., doi: 10.1109/IOLTS.2018.8474122

[3] Sparsh Mittal. 2016. A Survey of Techniques for Approximate Computing. ACM Comput. Surv. 48, 4, Article 62 (March 2016), 33 pages. DOI: https://doi.org/10.1145/2893356

[4] M. Barbareschi, F. Iannucci and A. Mazzeo, "A Pruning Technique for B&B Based Design Exploration of Approximate Computing Variants," 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Pittsburgh, PA, 2016, pp. 707-712., doi: 10.1109/ISVLSI.2016.110

[5] Vallero, A. et al., Early Component-Based System Reliability Analysis for Approximate Computing Systems, in Proceedings Of The 2nd Workshop On Approximate Computing (WAPCO), pp.1-4, DOI: 10.13140/RG.2.1.3883.5604

[6] M. Traiola, A. Savino, M. Barbareschi, S. D. Carlo and A. Bosio, "Predicting the Impact of Functional Approximation: from Component- to Application-Level," 2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS), Platja d'Aro, 2018, pp. 61-64., doi: 10.1109/IOLTS.2018.8474072

[7] Vallero. A et al., SyRA: Early System Reliability Analysis for Cross-layer Soft Errors Resilience in Memory Arrays of Microprocessor Systems, in IEEE Transactions on Computers, 2018.

[8] K. Chibani et al., "Fast accurate evaluation of register lifetime and criticality in a pipelined microprocessor", 22nd IFIP/IEEE Int. Conf. on Very Large Scale Integration (VLSI-SoC), 2014, pp. 260-265

[9] K Chibani, M Portolan, R Leveugle, **"**Application-aware soft error sensitivity evaluation without fault injections-Application to Leon3", European Conference on Radiation and its Effects on Components and Systems (RADECS'16), 2016

[10] A. Vallero et al., "Cross-layer system reliability assessment framework for hardware faults," 2016 IEEE International Test Conference (ITC), Fort Worth, TX, 2016, pp. 1-10., doi: 10.1109/TEST.2016.7805863

[11] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto and L. Taghaferri, "Data criticality estimation in software applications," International Test Conference, 2003. Proceedings. ITC 2003., Charlotte, NC, USA, 2003, pp. 802-810.

[12] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, J. Henkel and J. Henkel, "Architectural-space exploration of approximate multipliers," 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, 2016, pp. 1-8., doi: 10.1145/2966986.2967005

[13] H. Omar, M. Ahmad and O. Khan, "GraphTuner: An Input Dependence Aware Loop Perforation Scheme for Efficient Execution of Approximated Graph Algorithms," 2017 IEEE International Conference on Computer Design (ICCD), Boston, MA, 2017, pp. 201-208., doi: 10.1109/ICCD.2017.38

[14] B. Barrois and O. Sentieys, "Customizing fixed-point and floating-point arithmetic — A case study in K-means clustering," 2017 IEEE International Workshop on Signal Processing Systems (SiPS), Lorient, 2017, pp. 1-6., doi: 10.1109/SiPS.2017.8109980

[15] V. Mrazek, R. Hrbacek, Z. Vasicek and L. Sekanina, "EvoApproxSb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, Lausanne, 2017, pp. 258-261., doi: 10.23919/DATE.2017.7926993

[16] D. Menard and O. Sentieys, "A methodology for evaluating the precision of fixed-point systems," 2002 IEEE International Conference on Acoustics, Speech, and Signal Processing, Orlando, FL, 2002, pp. III-3152-III-3155., doi: 10.1109/ICASSP.2002.5745318

[17] E. Jenn et al, "Fault injection into VHDL models: the MEFISTO tool", 24th Symposium on Fault-Tolerant Computing (FTCS), 1994, pp. 66-75

[18] R. Leveugle, "Towards modeling for dependability of complex integrated circuits", 5th IEEE Int. On-Line Testing workshop, Rhodes, Greece, July 5-7, 1999, pp. 194-198

[19] S. S. Mukherjee et al., "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor", 36th IEEE Int. Symposium on Microarchitecture (MICRO-36), 2003, pp. 29-40

[20] A. Savino, S. Di Carlo, G. Politano, A. Benso, A. Bosio, G. Di Natale, "Statistical reliability estimation of microprocessor-based systems", IEEE Trans. on Computer, vol. 61, no. 11, Nov. 2012, pp. 1521-1534

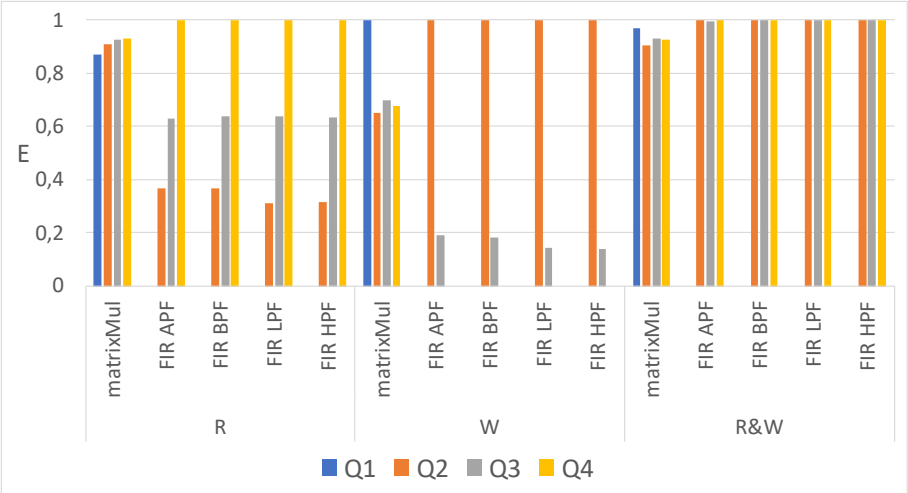

Figure 4: RDL normilized weighted error (E) on a single application output

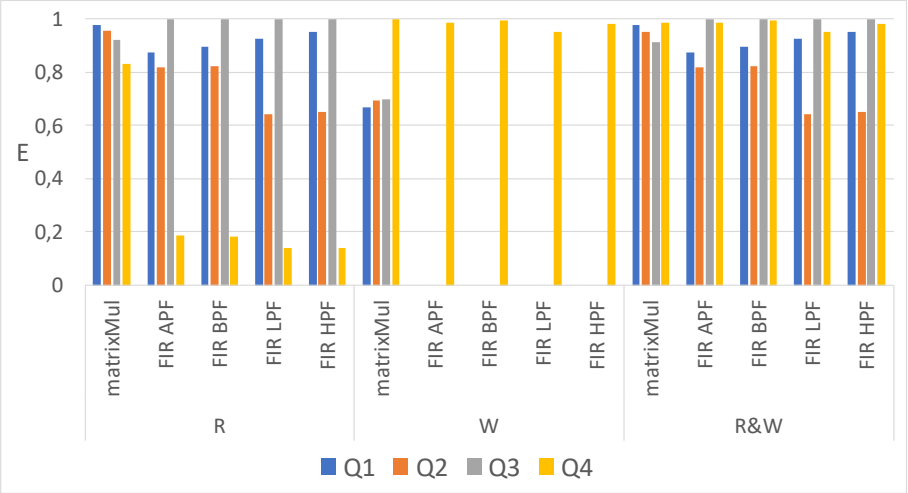

Figure 5: #R/#W normalized weighted error (E) on a single application output

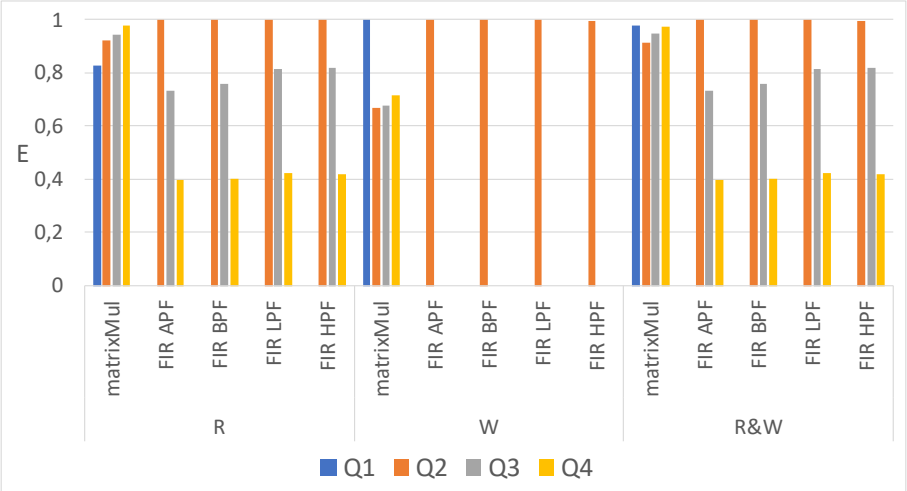

Figure 6: AVG Reads normilized weighted error (E) on a single application output

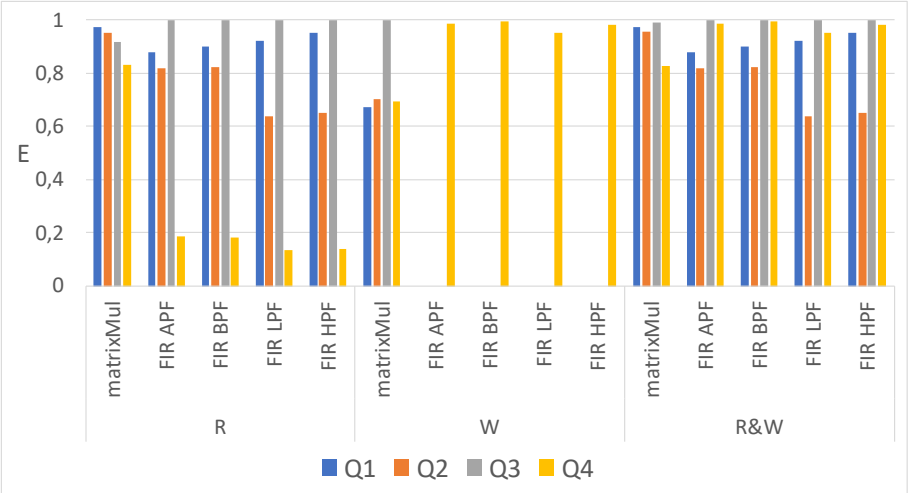

Figure 7: AVG Writes normilized weighted error (E) on a single application output