# Stochastic Modeling to Accelerate Approximate Operators Simulation

Justine Bonnot, Karol Desnos, Daniel Menard

# Stochastic Modeling to Accelerate Approximate Operators Simulation

Justine Bonnot[1], Karol Desnos[1], Daniel Menard[1]
[1]UBL, INSA Rennes, IETR CNRS UMR 6164, Rennes, France
email: {jbonnot, kdesnos, dmenard}@insa-rennes.fr

*Abstract*—**Approximate operators have been developed to overcome the performance limitations of the original accurate arithmetic operators. They trade off the output quality of the operator and its energy consumption, area or delay. To benefit from this trade-off, the logic structure of the original accurate operator is modified. When integrating approximate operators in a complex system, numerous simulations of the application are required to ensure the fulfillment of the application requirements, despite the induced approximations. Because the hardware implementation of approximate operators is not always available in early phases of application prototyping, long software simulation of their complex bit-level structure has to be used. This paper proposes a fast simulator for approximate operators built from the output values of the original accurate operator. The error due to the approximation is modeled by a stochastic process whose features are learned from the errors of the approximate operator. The proposed simulator is compared to the bit-accurate logic-level simulation and to a simulator of approximate operators built on the input values of the operator. Experiments on 10-bit operators show that the proposed method is up to 63× faster than a bit-accurate logic level simulation.**

## I. Introduction

The current challenge, when designing algorithms for embedded platforms, is to embed always more intricate algorithms on over-constrained platforms, whether it be constraints on the energy consumption or on the memory footprint.

To overcome this paradox, approximate computing techniques have been proposed, taking advantage of the inherent error-resilience of a majority of algorithms to embed. Indeed, algorithms in the image, video processing or data mining fields, or recursive algorithms are error-resilient by nature, as presented in [4] for video coding algorithms. The output quality of these algorithms is traded-off with their implementation cost using approximations, such that real-time and power consumption constraints of embedded systems are met. Among the numerous approximation techniques proposed, inexact arithmetic operators (approximate operators) are under consideration in this paper.

To overcome the current limitations in terms of power consumption and/or area of the accurate arithmetic operators, their boolean function can be modified. This functional approximation induces errors to reduce the logic complexity and/or the length of critical paths as presented in [5]. Approximate operators, as the Almost Correct Adder (ACA) presented in [8] or the Approximate Array Multiplier (AAM) in [7], are under consideration in this paper. To use an approximate operator in an application, a design space exploration is processed to find the best approximation configuration as well as to ensure that the application quality of service is still met despite the induced approximations. To explore the whole design space to fix the different parameters of the approximate operators and find which operations should be approximated, a fast simulation of the approximate operators is needed. However, the simulation of approximate operators is intricate since their hardware implementation is more complex than their accurate implementation. To study the impact of the induced approximations on the application, techniques as interval or affine arithmetic [3] cannot be used. Indeed, they do not monitor the impact of the approximation on the application quality metric. Moreover, the frequency of error occurrence is not taken into account.

Consequently, to study the impact of approximate operators in an application, the simulation is required. To process the design space exploration of an application, the approximate operator is simulated within the application described with a C code. Then, the different approximation perspectives are simulated with a large set of input data to ensure the quality of service of the application. Currently, simulating approximate operators is complex since the operation cannot be implemented directly with a single native instruction of the host computer. To simulate them, three alternatives can be considered. The simulation can be done in Hardware Description Language (HDL) at the logic level as presented in [6]. Hardware in the loop approach can be considered by implementing the operator in an Field Programmable Gate Array (FPGA). Nevertheless, these two alternatives require the implementation of a complex interface with the C code or the computer. The simplest solution is to simulate approximate operators with an equivalent C code reproducing the internal logic structure of the operator. This software simulation is very long since, to be bit-accurate, the simulation must be considered at the logic-level (BALL simulation). Despite being prohibitively long, numerous simulations, with regards to the set of inputs and the different approximation configurations, are necessary. Without being able to test the impact of such operators in an application, their use in real embedded systems is compromised. Thus, a fast simulation of these operators is strongly needed to quickly evaluate their impact on the application quality.

In this paper, we propose a new approach to evaluate quickly the application quality metric from simulation. To accelerate the simulation compared to a BALL one, the error $\widehat{e}$ due to approximate operators is modeled by a pseudo-random variable (p.r.v.) $\tilde{e}$. The error $\widehat{e}$ depends on the values handled by the operators. Thus, the output set is decomposed in subspaces and a p.r.v. $\tilde{e}_i$ is associated to each subspace. The characteristics of each p.r.v. $\tilde{e}_i$ are stored in a table $\mathcal{T}_{err}$.

Compared to our previous approach (FnF$_i$) presented in [1], in the proposed Fast and Fuzzy simulator (FnF$_o$), the output set is decomposed instead of the input set. Consequently, the size of the table storing the statistical parameters $\tilde{e}_i$ is dramatically decreased in the case of an addition or subtraction operator. The p.r.v. $\tilde{e}$ combines a Bernouilli distributed p.r.v. to control the error frequency and a uniform p.r.v. to control the error amplitude. Moreover, the effect of the number of subspaces on the result of the simulation is analyzed.

For a simple approximate adder on 16 bits, a simulation time saving up to $5.5\times$ is demonstrated. For a more sophisticated approximate mutiplier on 10 bits, a simulation time saving up to $63\times$ is demonstrated while keeping a good quality on the simulation output.

The paper is organized as follows: Section II details the proposed FnF$_o$ simulation. The efficiency of the FnF$_o$ simulation in terms of time savings and quality as well as a comparison with the FnF$_i$ simulation, are exposed in Section III. Finally, Section IV concludes the article.

## II. Proposed technique

### A. Modelization of the approximate operator error by a stochastic process

In this part , the proposed method to accelerate the quality evaluation process is presented. Let $\widehat{\diamond}$ be an approximate operator whose input operands $x \in I_x = [\underline{x}; \overline{x}]$ and $y \in I_y = [\underline{y}; \overline{y}]$ are coded on $N_x$ and $N_y$ bits respectively. $\underline{x}$ and $\overline{x}$ represent the minimum and maximum value of x (same for $y$). The accurate original operator is $\diamond$. The output of the accurate operator $\diamond$ is $z = x \diamond y$ and is coded on $N_z$ bits. The set of all the possible output values for the accurate operator $\diamond$ is $\mathcal{O}$. For an addition or subtraction, the output set $\mathcal{O}$ is composed of $2^{\max(N_x,N_y)+1}$ values and for a multiplier $\mathcal{O}$ is composed of $2^{N_x+N_y}$ values.

Let $\widehat{e}(x, y)$ be the error at the output of the approximate operator whose inputs are $x$ and $y$. $\widehat{e}(x, y)$ is expressed as:

$$\widehat{e}(x, y) = x\widehat{\diamond}y - x \diamond y \tag{1}$$

In case of fixed-point arithmetic, the error generated by the finite word-length has been widely studied to derive mathematical models as presented in [2]. Indeed, the fixed-point error can be considered as a uniform random variable. The frequency of error occurrence is equal to 1 since the error is always present but the error amplitude is small compared to the amplitude of the original signal. Contrary to fixed-point errors, the error $\widehat{e}$ due to approximate operators can have a high amplitude, but does not always occur. Consequently, when using approximate operators, the frequency of error occurrence must be low so as not to degrade too much the output quality of the application. Thus, defining a mathematical model for approximate operators is still an issue.

The straightforward approach to evaluate the degradation on the application quality is to simulate the application with the approximations on a set of representative data. To capture the effects of approximate operators on the application quality, the internal logic structure of the operator must be simulated. Even if this simulation is carried-out with C language, the logical level simulation slows down drastically the application simulation process. A table storing the error $\widehat{e}$ and addressed by $x$ and $y$ can be considered to reproduce the exact behavior of the approximate operator. Nevertheless, this table is composed of $2^{N_x+N_y}$ elements. The amount of memory to store this table is prohibitively big even for small values of $N_x$ and $N_y$.

In the proposed approach, the error due to the approximation is modeled by a stochastic process whose features are determined with an operator characterization phase. The proposed method aims to replace the approximate operator $\widehat{\diamond}$ by the accurate version of the operator $\diamond$ plus an error $\tilde{e}$ with the same statistical characteristics as the error $\widehat{e}$ generated by the approximate operator, as:

$$x\widehat{\diamond}y \Leftrightarrow x \diamond y + \tilde{e} \tag{2}$$

### B. Pseudo-random variables $\tilde{e}_i$ to model the error $\widehat{e}$

Considering a single error $\tilde{e}$ to model the error of the entire set $I_x \times I_y$ leads to a coarse model of the approximate operator error $\widehat{e}$. To model the approximate operator error more finely, the proposed method decomposes the output set $\mathcal{O}$ in $2^{N_z-F}$ subspaces $\mathcal{O}_i$. The number F is called the fuzziness degree and impacts the accuracy of the modelization and the size of the subspaces. For each subspace $\mathcal{O}_i$, a p.r.v. $\tilde{e}_i$ is used to model the error within the subspace $\mathcal{O}_i$. Each subspace $\mathcal{O}_i$ contains the output values $z = x \diamond y$ sharing the same $N_z$-F Most Significant Bits (MSB). These output values are modeled by the same p.r.v. $\tilde{e}_i$. The bigger F, the bigger the subspaces $\mathcal{O}_i$ and consequently the more information are summarized within a single p.r.v. $\tilde{e}_i$.

The statistical characteristics of the p.r.v. $\tilde{e}_i$ are stored in a table $\mathcal{T}_{err}$ and the $N_z$-F MSB of the variable $z$ are used to address $\mathcal{T}_{err}$. Rather than indexing this table with the inputs, as in the simulator FnF$_i$, the proposed approach uses the output values to index the table $\mathcal{T}_{err}$.

Numerous input combinations do not generate an error at the output of the approximate operator $\widehat{\diamond}$. Let $f_i$ be the frequency of error occurrence in the subspace $\mathcal{O}_i$. The error $\widehat{e}$ is equal to 0 with a probability of $1 - f_i$. To model the error committed in the subspace $\mathcal{O}_i$, the p.r.v. $\tilde{e}_i$ is generated with the Equation 3.

$$\tilde{e}_i = p_i.\,(a_i.u_i + b_i) \tag{3}$$

In Equation 3, $u_i$ represents a uniform random variable and $p_i$ a random boolean variable whose distribution follows a Bernoulli law. The random variable $p_i$ is equal to 1 with a probability $f_i$ and to 0 with a probability $1 - f_i$. The random variable $p_i$ is obtained from the random variable $u_i$ uniformly distributed in the interval $[0, 1]$ and presented in Equation 4. The variables $(a_i, b_i)$ are the coefficient of the affine form used to compute an error value with the right amplitude.

$$p_i = \begin{cases} 1 & \text{if } u_i > f_i \\ 0 & \text{else.} \end{cases} \tag{4}$$

During the approximate operator error characterization phase to build the proposed simulator, for each subspace $\mathcal{O}_i$, the characteristics of the error $\widehat{e}_i$ generated by the approximate operator are extracted. For each $\mathcal{O}_i$ in $\mathcal{O}$, the error values

$\widehat{e}_i$ are computed for the input combinations $(x, y)$ such that $z = x \diamond y \in \mathcal{O}_i$. Then, from these error values in $\mathcal{O}_i$, the error amplitude represented by $(a_i, b_i)$ and the threshold $f_i$ to generate an error with the same frequency of error occurrence are computed and stored in the table $\mathcal{T}_{err}$.

### C. Algorithm to simulate $x \widehat{\diamond} y$

The algorithm to simulate $x \widehat{\diamond} y$ is built with two precomputed tables, $\mathcal{T}_{idx}$ and $\mathcal{T}_{err}$. $\mathcal{T}_{idx}$ is used to know if an error occurs in $\mathcal{O}_i$, and is of size $2^{N_z - F}$. The table $\mathcal{T}_{err}$ stores the statistical characteristics $(a_i, b_i, f_i)$ of the different errors $\tilde{e}_i$.

The computation of $x \widehat{\diamond} y$ is detailed in Figure 1. Firstly, the exact value $z = x \diamond y$ is computed. Then, the $N_z$-F MSB of $z$ are extracted, leading to the value $z_0$, that addresses the table $T_{idx}$. The value $z_0$ indicates in which subspace $\mathcal{O}_i$ the output value $z$ belongs to.

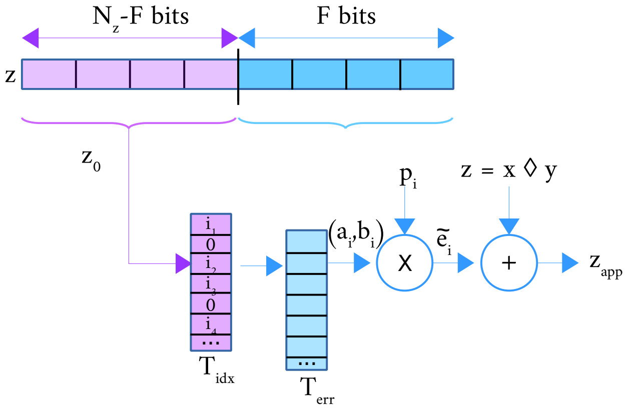

Fig. 1. Simulation of $x \widehat{\diamond} y$

The value $\mathcal{T}_{idx}[z_0]$ indicates if $x \widehat{\diamond} y$ may generate an error. If $\mathcal{T}_{idx}[z_0]$ is equal to zero, no error is generated and the accurate version of the arithmetic operator already computed $z$ is directly used, thus avoiding any supplementary processing. Otherwise, $\mathcal{T}_{idx}[z_0]$ gives the index $k$ to address the second table $\mathcal{T}_{err}$ and allows to retrieve the parameters $a_i$, $b_i$ and $f_i$ of the p.r.v. $\tilde{e}_i$. The error $\tilde{e}_i$ is finally generated from the Equation 3.

To generate the uniform random variable $u_i$ used to compute the error $\tilde{e}_i$, the LSB of the accurate output value $z$ are considered. As described in [9], the LSB of $z$ can be considered as a uniform random variable. Widrow derived that the LSB of a signal can be considered as a white random additive noise non-correlated with the input signal. The LSB of $z$ are then `xor`ed with a constant $K$ to scramble it. The obtained result is the uniform random variable $u_i$ .

The C code developed to implement the proposed approach has been optimized to waste the least cycles possible when simulating operands that do not generate any error.

## III. Experimental Results

The proposed approach allows a fast simulation of approximate operators to process the design space exploration of an application with approximate operators. To demonstrate the savings on the simulation time, the BALL simulation time is compared to the FnF$_o$ and FnF$_i$ simulation time for two operators and different input operands word-lengths. Then, the impact of the fuzziness degree $F$ on the simulation output quality is presented. Finally, the overhead in terms of computation time and memory footprint due to the approximate operator characterization phase is quantified for FnF$_o$ and FnF$_i$.

### A. Simulation time savings with the FnF simulation

To demonstrate the simulation time savings offered by the use of the FnF$_o$ simulator, the simulation time of FnF$_o$ is compared with the BALL and the FnF$_i$ simulation time. The results have been obtained on a processor Intel i7-6700 with 32Go of memory for two operators, the ACA and the AAM.

The simulation time for different input operand bit-width for the operator ACA is presented in Figure 2. The internal logic structure of the ACA is simple to reproduce with a C code since the approximation simply consists in cutting the carry-chain length of an addition. The simulation time gains are moderate due to the simplicity of the logic structure of this adder and are up to $5.5\times$ compared to the BALL simulation. The FnF$_o$ simulator is faster than the FnF$_i$ since the tables to store are much smaller, and less operations are needed when computing a simulation with no errors.

The FnF$_i$ simulator was particularly useful for the design space exploration of an algorithm with more complex operators, as for instance the AAM. Indeed, for the simulation of a 16-bit AAM, the FnF$_i$ was $44\times$ faster than the BALL simulation. Neverheless, the behavior of the FnF$_o$ with a multiplier is more complex, as presented in Figure 3. With the AAM, the FnF$_o$ simulation time gains are increasing up to $63\times$ on a 10-bit AAM, to then decrease up to 37 on a 16-bit AAM. The FnF$_i$ offers higher gains for multipliers whose input bit-width is greater or equal to 11, but in both cases, the simulation time savings on a 16-bit multiplier are considerable for the design space exploration of an application.

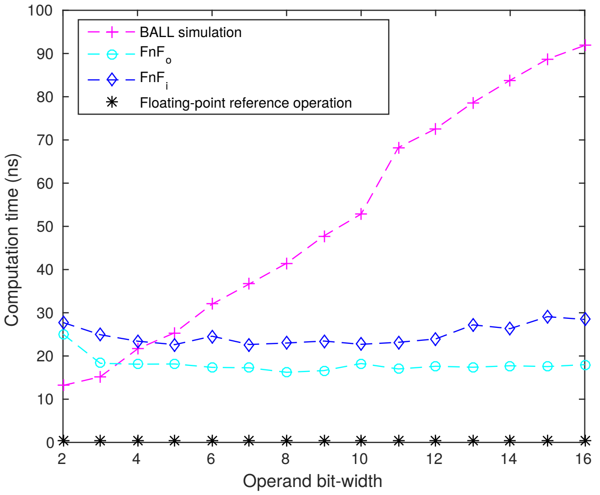

Fig. 2. Simulation time for the BALL and FnF simulation of the ACA

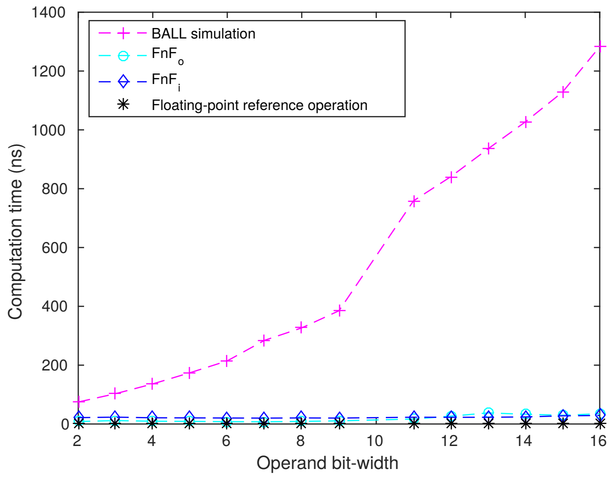

Fig. 3. Simulation time for the BALL and FnF simulation of the AAM

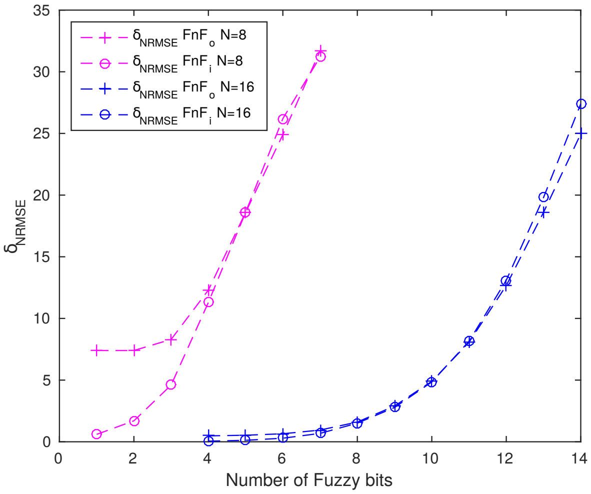

Fig. 4. $\delta_{NRMSE}$ between the BALL simulation, the $\text{FnF}_o$ and $\text{FnF}_i$ simulation of the ACA (carry chain-length cut at half)

### B. Impact of $F$ on the quality of the simulation

The lower $F$ is, the more accurate the simulation is, and the bigger the tables to store are. $F$ fosters a trade-off between the memory space available for the simulator and the accuracy needed to study the impact of the approximate operator on the application. The simulation time depends on the number of errors generated by the original approximate operator $\widehat{\diamond}$ and on $F$ that impacts the size of the tables and consequently the cache misses. The lower $F$, the slower the FnF simulation.

To study the impact of the fuzziness degree on the simulation output quality, the relative error of normalized rooted mean squared error (NRMSE) between the approximation (computed with the BALL simulation) and the $\text{FnF}_o$ and $\text{FnF}_i$ simulations are presented in Figure 4 for two operators, the ACA on 8 bits with a carry chain-length cut at 4 and the ACA on 16 bits with a carry chain-length cut at 8. The relative error between both NRMSE is called $\delta_{NRMSE}$ and is expressed in percent. For the ACA on 8 bits, $\delta_{NRMSE}$ stays under $10\%$ if $F$ is lower or equal to half of the input bit-width. On the 16-bit ACA, the margin is bigger. Indeed, $\delta_{NRMSE}$ stays under $10\%$ until F is equal to 12. The supplementary error due to the proposed model is acceptable for a number of Fuzzy bits $F$ between 50 to 75 % of the total operator word-length. As shown in the next section, this leads to small tables for our approach.

### C. Approximate operator characterization phase

Compared to the BALL simulation, a pre-processing phase is required to build the tables $\mathcal{T}_{err}$ and $\mathcal{T}_{idx}$ for each operator. The approximate operator error $\widehat{e}_i$ is characterized and the statistical characteristics of the p.r.v. $\tilde{e}_i$ are computed and stored in the tables $\mathcal{T}_{idx}$ and $\mathcal{T}_{err}$. In Table I, the memory footprint to store the tables $\mathcal{T}_{err}$ and $\mathcal{T}_{idx}$ for an average value of $F$ i.e. $F = \lfloor \frac{N_z}{2} \rfloor$ is provided, as well as the time to build the tables. The characterization step is done once off-line for each new operator and the tables used in this paper are available online to avoid this step. The simulator $\text{FnF}_o$ is longer to build by construction because consecutive output values of an operator are not necessarily belonging to the same subspace, which induces a bad locality in terms of memory accesses. The results are presented in the case of an exhaustive test of all the value $x$ and $y$. Nevertheless, for operators with a high operand word-length, an exhaustive test is not mandatory. Accurate estimation of the statistical parameters $a_i, b_i$ and $f_i$ can be obtained with a limited number of operand values reducing dramatically the time required to build the tables. The main advantage of the proposed approach compared to the $\text{FnF}_i$ is the significant reduction of the size of the tables for the adder. For the ACA on 16 bits, the memory footprint is divided by 166 with the proposed approach compared to tables obtained with the $\text{FnF}_i$ simulator.

| $\widehat{\diamond}$ | N | t ($\text{FnF}_o$) | t ($\text{FnF}_i$) | Mem ($\text{FnF}_o$) | Mem ($\text{FnF}_i$) |
|---|---|---|---|---|---|
| **AAM** | 6 | $<$ 1ms | $<$ 1ms | 764B | 1KiB |
| | 7 | $<$ 1ms | 0.2s | 3KiB | 3KiB |
| | 8 | $<$ 1ms | 0.3s | 45KiB | 3KiB |
| | 10 | 0.4s | 3.5s | 12KiB | 12KiB |
| | 16 | 4h20' | 137.5s | 768KiB | 768KiB |
| **ACA** | 6 | $<$ 1ms | $<$ 1ms | 192B | 1.3KiB |
| | 7,8 | $<$ 1ms | $<$ 1ms | 384B | 4KiB |
| | 10 | 37s | 2.98s | 768B | 16KiB |
| | 16 | 4h15' | 62.5s | 6 KiB | 1MiB |

TABLE I. TIME AND MEMORY OVERHEAD TO CONSTRUCT THE FAST AND FUZZY SIMULATOR

## IV. CONCLUSION

This paper presents a fast simulator for the design space exploration of an application with approximate operators. Built on the output values of the original accurate operator, the proposed $\text{FnF}_o$ simulator is always faster than the $\text{FnF}_i$ simulator for the approximate adder ACA, and for the approximate multiplier AAM up to an input bit-width equal to 10. The size of the tables on which the simulator is based is significantly lower than with the $\text{FnF}_i$ for the adder. This paper proposes a comparison to help the approximate algorithm designer to choose the best simulator depending on the considered operator. As a future work, a method to characterize high word-length operators with Monte-Carlo simulations will be developed to avoid to exhaustively test all the operand values.

# References

[1] J. Bonnot, K. Desnos, and D. Menard. Fast and fuzzy: a simulator for inexact arithmetic operators. https://mycore.core-cloud.net/index.php/s/j8lBnw4bq8Z1IrL, 2017. submitted to DATE 2018.

[2] T. Hilaire, D. Menard, and O. Sentieys. Bit Accurate Roundoff Noise Analysis of Fixed-point Linear Controllers. pages 607–612, Sept. 2008.

[3] J. Huang, J. Lach, and G. Robins. Analytic error modeling for imprecise arithmetic circuits. *Proc. SELSE*, 2011.

[4] A. K. Katsaggelos, F. Ishtiaq, L. P. Kondi, M.-C. Hong, M. Banham, and J. Brailean. Error resilience and concealment in video coding. In *Signal Processing Conference (EUSIPCO 1998), 9th European*, pages 1–8. IEEE, 1998.

[5] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel. A low latency generic accuracy configurable adder. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.

[6] C. Valderrama, F. Naçabal, P. Paulin, and A. Jerraya. Automatic vhdl-c interface generation for distributed cosimulation: Application to large design examples. *Design Automation for Embedded Systems*, 3(2), 1998.

[7] L.-D. Van, S.-S. Wang, and W.-S. Feng. Design of the lower error fixed-width multiplier and its application. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47(10), 2000.

[8] A. K. Verma, P. Brisk, and P. Ienne. Variable latency speculative addition: A new paradigm for arithmetic circuit design. In *Proceedings of the conference on Design, automation and test in Europe*. ACM, 2008.

[9] B. Widrow and I. Kollár. Quantization noise. *Cambridge University Press*, 2, 2008.