Architectural-Space Exploration of Approximate Multipliers

Semeen Rehman[1], Walaa El-Harouni[2], Muhammad Shafique[2, 3], Akash Kumar[1], Jörg Henkel[2]

[1] Chair for Processor Design, Technische Universität Dresden (TUD), Dresden, Germany

[2] Chair for Embedded Systems, Karlsruhe Institute of Technology (KIT), Germany

[3] Vienna University of Technology (TU Wien), Austria

Corresponding Authors' Email: *semeen.rehman@tu-dresden.de, muhammad.shafique@tuwien.ac.at*

ABSTRACT

This paper presents an architectural-space exploration methodology for designing approximate multipliers. Unlike state-of-the-art, our methodology generates various design points by adapting three key parameters: (1) different types of elementary approximate multiply modules, (2) different types of elementary adder modules for summing the partial products, and (3) selection of bits for approximation in a wide-bit multiplier design. Generation and exploration of such a design space enables a wide-range of multipliers with varying approximation levels, each exhibiting distinct area, power, and output quality, and thereby facilitates approximate computing at higher abstraction levels. We synthesized our designs using Synopsys Design Compiler with a TSMC 45nm technology library and verified using ModelSim gate-level simulations. Power and quality evaluations for various designs are performed using PrimeTime and behavioral models, respectively. The selected designs are then deployed in a JPEG application. For reproducibility and to facilitate further research and development at higher abstraction layers, we have released the RTL and behavioral models of these approximate multipliers and adders as an open-source library at *https://sourceforge.net/projects/lpaclib/*.

Keywords: *Approximate Computing, Multiplier, Adder, Arithmetic, Design Space Exploration, Performance, Area, Power, Configurable Accuracy, Low Power Image Processing, Open Source, Library.*

1 INTRODUCTION

Alleviating the bounds of accuracy and precision in the exact computing offers new avenues for improving the area, power, and performance of on-chip systems at the cost of reduced output quality [1]-[9]. Recent research efforts by IBM [1], Intel [3], Microsoft [4]-[6], and various research groups [2], [7]-[9], [13]-[25] have demonstrated that there is a variety of error resilient applications from different domains that can tolerate approximation errors and still produce useful results. Examples of such applications are: image/video processing, computer vision, RMS (recognition, mining and synthesis), communication and networking, big data analytics, etc. Applications' resilience is typically attributed to different factors, for instance, psycho-visual perception limits, redundancy and noise in real-world data (images, sensors, etc.), statistical nature and error attenuation characteristics of processing algorithms, etc. [1]-[3], [6][7].

Approximate Computing exploits these resilience properties to trade the computation accuracy loss due to the approximate errors (within an acceptable range) for power, area, and performance savings.

ICCAD '16, November 07-10, 2016, Austin, TX, USA
 ISBN 978-1-4503-4466-1/16/11…$15.00
DOI: http://dx.doi.org/10.1145/2966986.2967005

There are several individual works on circuit-/arithmetic level [10]-[22], approximate accelerators [13], application-level approximations [24], programming language support for approximations [4]-[6], [8], and approximate caches [25][26]. However, *for efficiently enabling approximate computing across the complete computing stack* (e.g., at architecture, high level synthesis, and system levels) with high flexibility and configurability, there is a need for a wide-range of power-quality configuration options for approximate arithmetic modules (i.e., multipliers and adders) [2]. *This instantiates the need for architectural space generation and exploration of such approximate modules as targeted in this paper.*

1.1 State-of-the-Art and Open Research Problems

In error resilient (application-specific) systems, two important arithmetic blocks are adders and multipliers, where large-sized multipliers even use adder trees for summing the partial products. Therefore, designing approximate versions of these modules has received significant research interest [10]-[22]. Typically, approximate adders *either* truncate the carry propagation chain for reducing the latency / critical-path [12]-[17], *or* eliminate carry computation and circuit parts to save power consumption [10][11][18]. In general, these adders define a *fixed* scheme for approximation that hampers exploration of quality-power tradeoffs. The Gracefully-Degrading Adder (GDA) [17] and the Generic Accuracy Configurable Adder (GeAR) [12] target reduced latency configurable approximate adders by combining several sub-adder units in overlapping fashion to reduce the length of carry prediction, thus incur a significant area/power overhead. *Therefore, such adders cannot be used for building low-power approximate multipliers*. The works in [10][11] provide four designs of approximate 1-bit full adders (FAs), which will serve as an input to this paper as the elementary adder modules.

In contrast to adders, there is a limited work on approximate multipliers [19]-[22]. The work on Error Tolerant Multiplication (ETM) [21] splits the input operands to improve the delay, power, and area overheads for *certain input combinations*. A power aware 2×2 block approximate multiplier is designed in [19]. A truncated error correction technique is proposed in [20] to selectively correct errors in an approximate multiplier design. The work in [22] only employs bit-width approximation. These works on approximate multiplier explore either a fixed approximate multiplier design or a limited number of selections in terms of bit-width approximations only. However, in general, *these techniques do not jointly explore the selection of approximation type of elementary multiply/add modules and bit-widths* for approximate addition of partial products, thus cannot traverse the full architectural space of quality, area, and power. Moreover, these techniques focus on manual optimizations in the circuit to meet the error criteria such as error magnitude or error rate. However, these works *lack a systematic methodology for designing a wide-range of approximate multipliers* and leave several questions open, for instance, which type of approximate adder, multiplier and bit-width combination to select to fulfill the quality/power/area requirements. This paper aims at addressing these open research questions while targeting area and

power reduction. Improving performance through decreasing delay is not the focus of this work, but performance improvements can be achieved implicitly through circuit simplification of elementary adder and multiplier modules. Therefore, we also provide latency results for the discussed circuits for a comprehensive evaluation.

Before moving to the novel contributions of this paper, we present a motivational case study on approximate multipliers to highlight the limited design space of existing works and the available potential.

1.2 Motivational Case Study: Design Space Coverage of Existing Approximate Multipliers

Fig.1 presents the area, power, and output error plots for an accurate version and different approximate versions of an 8×8 multiplier. The output error cases are recorded by permuting over all possible combinations of 8×8 input operands. We employ the following three well-adopted quality metrics [10][19][22]:

(1) Number of Error Occurrences,
(2) Maximum Error, and
(3) Number of Maximum Error Case Occurrences.

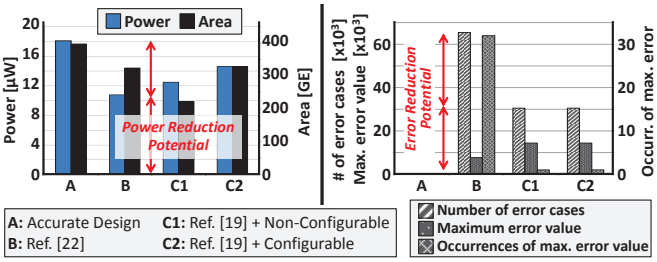

Fig.1: Restricted Design Space of Existing Approximate Multipliers.

The 8×8 multiplier design is based on recursive multiplier construction, where 2×2 elementary multiply modules are employed to generate partial products, and an adder tree is used to add the partial products and obtain the product result for 8×8 multiply operation. The accurate version *A* contains both 2×2 multiply and adder tree with accurate multiply/add hardware modules. The multiplier *B* is based on the design of [22], where 2×2 multiplier is accurate, but adders for *x-LSBs* (Least Significant bits; in this case x=4) related partial products are approximate while all other adders are accurate. *C1* and *C2* are based on the non-configurable and configurable designs of 2×2 approximate multiplier of [19], while the adder tree is accurate for summing the partial products. *B* is better in terms of power, while *C*s are better in terms of number of error cases. Fig.1 illustrates that the sparse design space of state-of-the-art approximate multipliers is primarily due to their constraints on one or more design parameters, and their fixed architectures. The design space will completely change, if only the type of the approximate adder is changed. However, such a selection will influence the final quality, power, and area results. Fig.1 shows that there is a huge potential for filling the missing points in the architectural design space of approximate multipliers.

1.3 What is Required and What are the Associated Research Challenges?

Given several approximate designs for elementary adders and multipliers modules, there is a need for a methodology for generation and exploration of a huge number of approximate multiplier design alternatives with varying tradeoff points for area, power, and accuracy. However, exploration of the architectural design space becomes more challenging for large-sized multiplier, where decisions on the following customizable parameters need to be taken.

(1) Employing accurate, approximate, or configurable 2×2 multiplication units;
(2) Employing accurate or approximate adders, when computing the sum of partial products in recursive multiplier designs;
(3) How many LSBs need to be approximated in the adder tree?

Size of Design Space: Let $N_{approxAdd}$ is the number of different types of elementary approximate adders, $N_{approxLSB}$ is the number of options for LSBs approximation, $N_{approxMul}$ is the number of different types of elementary approximate multipliers and *NMM* is the number of elementary multiply modules. The total number of design-space points D_{sp} can be calculated using the following equation:

$$D_{sp} = (N_{approxAdd}) * (N_{approxLSB}) * (N_{approxMul})^{NMM} \quad (1)$$

An Example Calculation for the Design Space Size: Let us consider a 4×4 multiplier with the elementary multiply module (used for the partial product generation) of size 2×2. Given one accurate and four approximate designs of 2×2 multiply, we have 5 different types of elementary multiply modules. Similarly, let us also consider that we have 4 different types of elementary add modules (1 accurate, and 3 approximate designs) and 3 possibilities for LSB-approximations (0-, 2-, or 4-bits). According to the Eq. 1, we will have 4 x 3 x 5^4 = 7500 possible configurations for building a 4×4 multiplier. Similarly, for an 8×8 multiplier, the design space includes 1.831×10^{12} possible points. Note that in this case, the assumption was that the selection for the adder type and the approximate LSBs will apply to the whole adder tree. Therefore, configuring each adder separately will further expand the design space.

1.4 Our Novel Contributions

In this paper, we present a novel methodology for generation and exploration of the architectural design-space of approximate multipliers, and thereby enabling various approximate versions as tradeoff points with diverse power, output quality, and area properties. Different multiplier designs are synthesized using Synopsys Design Compiler for a 45nm TSMC technology library and functionally verified using ModelSim gate-level simulations. The power is estimated using PrimeTime. Selected multiplier designs are applied to a JPEG application with different combinations of approximate and accurate DCT / IDCT.

Configuration coverage for state-of-the-art approximate multipliers: our methodology covers numerous combinations of different design parameters and thereby enables covering the configurations of state-of-the-art approximate multipliers like [19][22]. As a result, we can even evaluate the efficacy of those designs w.r.t. the new design points in terms of power, area, and output quality.

1.5 Open-Source Library

The RTL and behavioral models of these approximate multipliers and adders are released as part of the *open-source library* at *https://sourceforge.net/projects/lpaclib/* [23]. This will not only facilitate reproducing and comparing the results, but will also enable further research and development at higher abstraction layers. For instance, (i) architects can use this library for building approximate hardware accelerators manually or using high-level synthesis tools; or (ii) different design points can be exploited by variable resilience applications/functions to meet specific system requirements.

2 EXPLORATION OF APPROXIMATE MULTIPLIERS

2.1 Methodology Overview

Fig.2 presents an overview of our methodology for generation and exploration of the architectural design-space of approximate multipliers. It consists of the following four key steps (discussed in

detail in the following sections).

1) *Library Building (Section 2.2):* First, we build a library of elementary approximate adder and multiplier modules. Besides re-using the existing designs, we developed another 2×2 approximate multiplier in both configurable[1] and non-configurable designs to expand the design space [2].
2) *Characterization (Section 2.3):* Afterwards, we perform area, power, and quality characterization of different elementary approximate modules and filter out non-Pareto points.
3) *Building Large-Sized Modules (Section 2.4):* Various approximate multipliers are generated using combinations of (i) elementary approximate multiplier modules, (ii) elementary approximate adder modules, and (iii) number of approximated LSBs.
4) *Selection of Design Points (Section 2.5):* An exploration heuristic is employed that selects a subset of power-efficient design points considering the quality constraints.

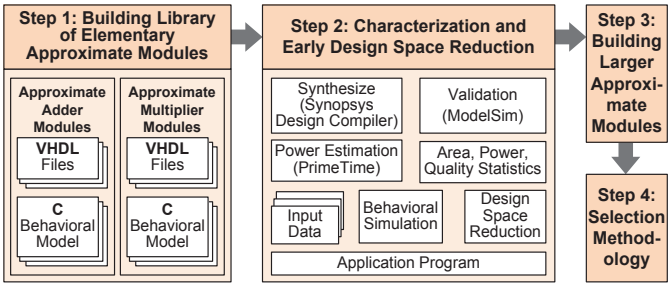

Fig.2: Methodology Overview.

2.2 Building the Library of Approximate Modules

Building 2x2 Approximate Multipliers: The constitutional building blocks of the multiplication circuits are the 2-bit×2-bit multipliers to generate the partial products and the adder trees for summing the partial products. First, we implemented the accurate 2×2 multiplier module (see *AccuMul* in Fig.3(a)), and the state-of-the-art 2×2 approximate multiplier module from [19] (see $ApproxMul_1$ in Fig.3(b)), which supersedes existing low-power approximate multiplier designs. It reduces the output to 3-bits by approximating the 3×3 case to give a product of a value=7, instead of 9. This leads to the maximum error magnitude of 2. The work in [19] also proposed an accuracy-configurable version that adds *2* to the approximate result when signaled for accurate computation. In order to *expand the design space* and to obtain a design with a *reduced amount of maximum error* (that may be required by certain applications), we developed four new designs of 2×2 approximate multipliers; see $ApproxMul_2$ – $ApproxMul_5$ in Fig.3(c-f). The characterization results for different multiplier modules are discussed in Table I.

Out of our four designs, we finally selected the design of $ApproxMul_2$ (Fig.3(c)) for building the large-sized multiplier modules, because it lies on the Pareto-optimal curve as shown in Table I. This design reduces the maximum error magnitude compared to state-of-the-art design [19]. Unlike the design of $ApproxMul_1$ in [19], our $ApproxMul_2$ does not reduce the output to 3-bits. Rather, *we use the concept of equating similar output bits.* The truth table in Fig.3(c) shows that the MSB and the LSB are equal except in the highlighted cases. If we remove the logic for LSB evaluation and then connect the MSB to the LSB, we get the following three approximation cases, all of which have an absolute error magnitude of 1: (I) `01x01 = 0000`; (II) `01x11 = 0010`; and (III) `11x01 = 0010`. This has the advantage that, if accurate computation is signaled, the error correction of a product requires only *inverting* the LSB. Therefore, our configurable design exhibits an area-/power-efficient error correction logic. On the contrary, the state-of-the-art $ApproxMul_1$ [19] requires an extra *adder logic* to correct the result that undermines the savings of approximations. Note: the error cases in both types of multipliers are known at design time (e.g., as shown in Fig.3(c)) and the results for configurable versions in Table I consider both error detection and correction.

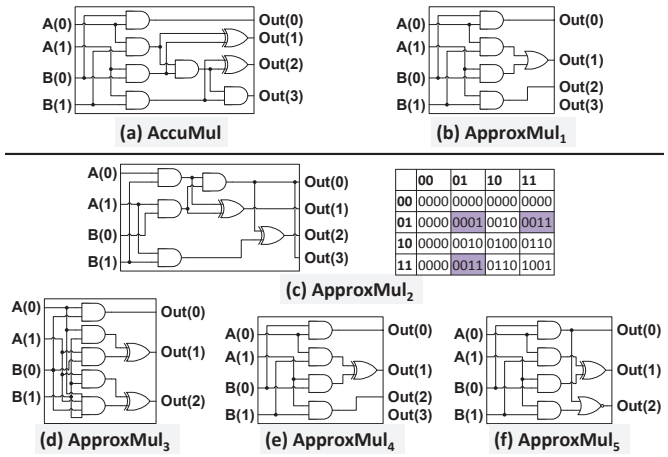

Fig.3: 2x2 Multiplier Designs (a) Accurate; (b) State-of-the-Art Approximate Multiplier of [19]; (c-f) Our Approximate Multiplier Designs with Truth Table of the Selected Pareto-Optimal Design.

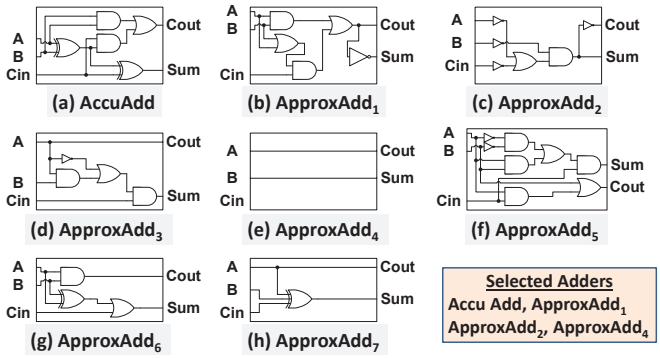

Fig.4: 1-bit Adder Designs (a) Accurate; (b-f) State-of-the-Art Approximate Adders of [10][11]; (g-h) Additionally Tested Approximate Adder Designs.

Implementing Approximate 1-bit Full-Adders: First, we implemented (1) the accurate 1-bit full-adder (see *AccuAdd* in Fig.4(a)), and (2) five state-of-the-art approximate 1-bit full-adder modules from [11] (see $ApproxAdd_1$ – $ApproxAdd_5$ in Fig.4(b-f)), which supersede existing low-power approximate adder designs. These designs exhibit transistor-level approximations, and we implemented the gate-level circuits according to their reported truth-tables. $ApproxAdd_5$ simplifies the logic of *Sum* evaluation and exhibit two error cases. $ApproxAdd_1$ computes the *Sum* bit by simply inverting the C_{out} bit. $ApproxAdd_2$ and $ApproxAdd_3$ are obtained by having different combinations of the above two types of approximations. An aggressive approximation is done in $ApproxAdd_4$ by simply rewiring, i.e., C_{out} = A, and *Sum* = B. In order to *expand the design space*, we additionally designed and tested two new designs of 1-bit approximate adders (see

[1] Note: a configurable version indicates an accuracy-configurability, i.e., the approximate multiplier has an additional error correction unit, which can be activated and deactivated through an enable signal. This is not to be confused with hardware re-configurability of FPGAs.

ApproxAdd$_6$ and *ApproxAdd*$_7$ in Fig.4(g-h)). The characterization results for different adder modules are discussed in Table II. It shows that designs like *ApproxAdd*$_3$, *ApproxAdd*$_5$, *ApproxAdd*$_6$, and *ApproxAdd*$_7$ do not lie on the Pareto-optimal curve, and therefore filtered out in the early design-space reduction step. Other variable latency adders like [12]-[17] can also be used in the adder tree for partial product summation, but due to carry chain breaking, they may lead to a higher error magnitude, which may not be desirable. Our methodology is orthogonal to the use of low-power or high-performance variable latency adders for partial product summation.

2.3 Characterization and Early Design Space Reduction

Area, power, and accuracy for all library components are examined separately using the tool flow in Fig.8 (see tool-flow in Section 3.1). The output accuracy is quantified as (1) number of error cases, (2) maximum error magnitude, and (3) occurrences of the maximum error, compared to the accurate adder. For output quality testing, all possible combinations of inputs were applied to the elementary adder/multiplier modules. Table I and Table II illustrate the area, power, latency, and output quality results of approximate and accurate multiplier modules. Based on the characterization results we performed the *design-space reduction*.

Table I Area, Latency, Power, and Quality Characterization of Elementary Approximate 2x2 Multiplier Modules.

	Area [GE]	Latency [ns]	Power [nW]	Number of Error Cases	Max Error Magnitude	Occ. of Max Error
AccuMul	6.880	0.10	543	-	-	-
ApproxMul$_1$	3.704	0.06	363	1	2	1
ApproxMul$_2$	4.939	0.10	262	3	1	3
ApproxMul$_{1C}$	7.232	0.17	525	-	-	-
ApproxMul$_{2C}$	6.350	0.13	379	-	-	-
ApproxMul$_3$	5.645	0.10	464	1	8	1
ApproxMul$_4$	5.292	0.10	422	1	4	1
ApproxMul$_5$	5.645	0.10	467	1	8	1

Table I shows that only *AccuMul*, *ApproxMul*$_1$ [19], and *ApproxMul*$_2$ lie on the Pareto-optimal curve, and all other designs can be filtered out to reduce the design space. The reason of selection of *ApproxMul*$_1$ is the reduced number of error cases, and *ApproxMul*$_2$ is selected due to the reduced maximum error value. These three 2×2 multiplier modules are used in the next step of building large-sized multipliers. Table II shows that, out of 7 design options, only *AccuAdd*, *ApproxAdd*$_1$, *ApproxAdd*$_2$, and *ApproxAdd*$_4$ lie on the Pareto-optimal curve, and all other designs can be filtered out to reduce the design space. Note, in order to speedup the design space exploration process and to avoid inefficient design points of bigger multi-bit multiplier modules, only Pareto-optimal points are beneficial. Non-Pareto-optimal points will most likely be pruned in the design space exploration process.

Table II Area, Latency, Power, and Quality Characterization of Elementary Approximate 1-Bit Full-Adder Modules.

	Area [GE]	Latency [ns]	Power [nW]	Number of Error Cases	Max Error Magnitude	Occ. of Max Error
AccuAdd	4.41	0.12	1130	0	0	0
ApproxAdd$_1$	1.94	0.07	294	2	1	2
ApproxAdd$_2$	1.59	0.05	198	3	1	3
ApproxAdd$_3$	1.76	0.06	416	3	1	3
ApproxAdd$_4$	0	0.00	0	4	1	4
ApproxAdd$_5$	4.23	0.10	771	2	1	2
ApproxAdd$_6$	3.18	0.10	409	2	1	2
ApproxAdd$_7$	3.18	0.13	709	2	2	2

The key considerations during the selection are:

1) *ApproxAdd*$_4$ offers the best area and power results, but the lowest accuracy in terms of number of error cases. *ApproxAdd*$_5$ has the worst area/power results among all approximate designs, so it is ignored.
2) *ApproxAdd*$_2$ is selected as it offers good power and area, and reasonable accuracy (maximum value of *1*, and *3* error cases).
3) *ApproxAdd*$_1$ and *ApproxAdd*$_3$ have very similar area results, but *ApproxAdd*$_1$ offers reduced power consumption and better accuracy, and is therefore selected. Accordingly, *ApproxAdd*$_3$ is pruned.

2.4 Building Large-Sized Adder and Multiplier Modules

Larger components are built using the elementary 1-bit full-adders and 2×2-bits multipliers. Following the design principles of recursive multipliers, in order to obtain large-size approximate multipliers, we need multi-bit adders to sum up the partial products.

Building a Multi-Bit Approximate Adder Chain: Building a multi-bit adder chain is illustrated in Fig.5. FA_{AA} denotes a 1-bit accurate full-adder while FA_{XA} refers to an approximate 1-bit full-adder. An *N-bit* adder can be composed of K approximate adders and $N - K$ accurate adders, such that $K \leq N$. Fig.5 presents an example using K=2.

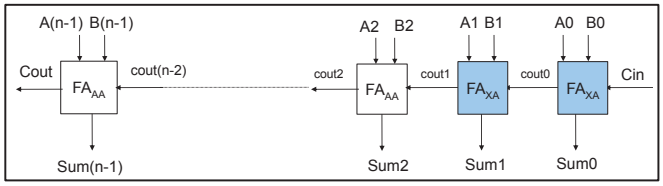

Fig.5: Multi-Bit Adder Chain using Accurate or Approximate 1-Bit Adders.

Without loss of generality, in this paper, we consider a ripple carry adder design where carry chain is not broken. However, other techniques to break the carry chain to explore another dimension of approximation, like the works of [12]-[17], can also be employed and is orthogonal to the contributions of this paper. Note that, to avoid high error magnitudes, we consider that LSBs are the first candidate for approximations, while MSBs should be accurate. Therefore, we consider K-consecutive approximate adders in our multi-bit adder design, such that K can also be seen as the number of LSBs added with 1-bit approximate adders. Also, for the ease of discussion and evaluation, we consider only one particular type of elementary approximate adder module (*ApproxAdd*$_1$ or *ApproxAdd*$_2$, but not both) in one design. However, our methodology is not restricted to this assumption. Our experience is that, such homogeneity is important for early reduction of the design space size, and shortening the development and testing time.

Table III Area, Latency, Power, and Quality Characterization of Approximate 8-Bit Adders.

	Area [GE]	Latency [ns]	Power [nW]	Number of Error Cases	Max Error Magnitude	Occ. of Max Error
AccuAdd	48.157	1.03	3250	0	0	0
ApproxAdd$_1$*, 4 LSBs*	38.279	0.93	2040	89600	15	512
ApproxAdd$_1$*, 8 LSBs*	28.400	0.89	1680	117950	255	2
ApproxAdd$_2$*, 4 LSBs*	36.868	0.72	1730	108288	15	768
ApproxAdd$_2$*, 8 LSBs*	25.578	0.49	1350	126891	255	3
ApproxAdd$_4$*, 4 LSBs*	29.988	0.52	926	122880	8	8192
ApproxAdd$_4$*, 8 LSBs*	12.348	0.06	480	130560	128	512

Table III presents area, power, latency, and accuracy results for 8-bit adder designs with different amounts of LSBs approximated using

ApproxAdd$_1$, *ApproxAdd*$_2$, and *ApproxAdd*$_4$ as elementary 1-bit full-adders. For the number of LSBs to approximate, we decided to go for a step size of 4 bits; (i) 0 LSBs (fully accurate), (ii) 4 LSBs (half the bit-width), and (iii) 8 LSBs (fully approximate). The total number of tested cases is 131,072 cases; (256*256) combinations *2 for C_{in} = 0 or 1.

Building a Large-Sized Approximate Multiplier: A large-sized multiplier can be recursively partitioned into multiple small-sized multiplication operations, such that, each of these operations can be processed in parallel in the same cycle. Fig.6 illustrates the process of recursive partitioning and partial product addition. Let us assume a number *X* is multiplied with another number *Y*, each is *2W-bit* wide, such that $W \in \mathbb{N}$. *X* and Y can be written as (X_H,X_L) and (Y_H,Y_L), respectively, where *L=H=W*-bits. Fig.6 shows how a *2Wx2W* multiplication is broken into multiple *WxW* multiplications, and the partial products at appropriate bit-positions are added using multi-bit adders. X_LxY_L, X_HxY_L, X_LxY_H, and X_HxY_H are the partial products and require four *WxW* multipliers. There can be different design options, for instance, X_LxY_L can carried out using approximate multipliers, while all other partial products are obtained using accurate multipliers. Or, X_LxY_L, X_HxY_L, and X_LxY_H are carried out using approximate multipliers, while X_HxY_H is obtained using an accurate multiplier. Similarly, for different partial products, different types of elementary 2x2 multipliers can be employed.

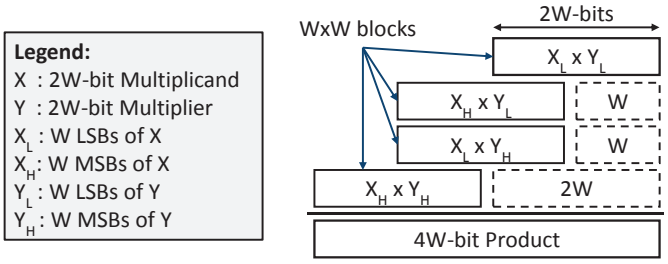

Fig.6: Building larger multipliers.

An Example: If W=4-bits, then the operation corresponds to 8x8 multiplier, and partial products are generated using 4x4 multipliers, which in turn can be broken down into multiple 2x2 multiplications.

Requirement of Elementary Multiplier Modules: The following equation calculates the number of elementary multiplier blocks for building a larger multiplier of 2^Lx2^L size. For instance, a 16x16 multiplier consists of 64 2x2 blocks. This lead to the questions: how many elementary approximate multiplier modules to use and what should be their distribution in a large-sized approximate multiplier architecture? In case the approximate adders are also considered, the challenge will be to first find out if it will be better to use an accurate multiplier and an approximate adder-tree or vice-versa, or a mixture.

$$2^L * 2^L \text{N. blocks} = 4^{L-1} \quad (2)$$

As seen from Table I, the approximate version from [19] results in more savings than the proposed approximation. It also offers better accuracy as only one out of the 16 possible outputs is inaccurate, while 3 out of 16 are inaccurate in the proposed approximation. However, the accuracy configurable version proposed out-performs its counter-part suggested in [19]. We then build larger multipliers (4x4, 8x8, and 16x16) out of the available 2x2 blocks while keeping the adder tree accurate. In doing so, we applied the same choice of multiplier type to all constituent 2x2 blocks. However, any other type of multiplier can also be used, but it is avoided in the implementations to simplify the exploration and development effort. However, our methodology and framework are orthogonal to this decision and can cover both cases. Table IV and Table V present the results for 4x4 and 8x8 multipliers.

The 4x4 multiplier based on the approximation from [19] results in more savings than our proposed approximation, while our accuracy configurable version offer better results in comparison. As for accuracy, *ApproxMul*$_1$ results in less number of error cases, on the other hand, the maximum error value for *ApproxMul*$_2$ is half compared to that of *ApproxMul*$_1$, but occurs more frequently (see Table IV). An appropriate approximate version can be selected based on the quality constraints, i.e., maximum error value or number of error occurrences. Number of test cases are 256 and 65536 for 4x4 and 8x8 multipliers, respectively.

Note, in the current work, we do not consider reuse of 2x2 multiplier modules within one large-sized multiplier, which will lead to a multi-cycle design. The reason is to achieve faster implementations. However, if a multi-cycle design is desired, reusing of 2x2 multiplier modules can also be considered. In this case, a new decision about which 2x2 multiplier module to reuse, accurate or approximate one, would be required for different bits, for which design space exploration algorithm needs to be extended accordingly.

Table IV Area, Latency, Power, and Quality Characterization of Approximate 4x4 Multiplier Modules.

	Area [GE]	Latency [ns]	Power [nW]	Number of Error Cases	Max Error Value	Occ. of Max Error
AccuMul	62.45	0.72	4110	-	-	-
ApproxMul$_1$	37.04	0.51	2670	49	50	1
ApproxMul$_2$	56.80	0.75	2830	119	25	7
ApproxMul$_{1C}$	63.86	0.81	4000	-	-	-
ApproxMul$_{2C}$	61.39	0.79	3630	-	-	-

Table V Area, Latency, Power, and Quality Characterization of Approximate 8x8 Multiplier Modules.

	Area [GE]	Latency [ns]	Power [nW]	Number of Error Cases	Max Error Value	Occ. of Max Error
AccuMul	324.22	1.75	16500	-	-	-
ApproxMul$_1$	222.62	1.60	12500	30625	14450	1
ApproxMul$_2$	302.70	1.81	12600	53375	7225	31
ApproxMul$_{1C}$	329.87	1.83	14600	-	-	-
ApproxMul$_{2C}$	321.05	1.83	15300	-	-	-

2.5 Selection methodology

Each design point in the architectural space has different characterizing properties like area, power, and accuracy/quality. State-of-the-art works (like [10]) typically employ the metric *QAP* (the product of quality/output error, area, and power) as a cost function to evaluate different design options, where a smaller value of *QAP* denotes a better design. However, our analysis showed that this metric lacks distinctiveness. For instance, let us assume that we want to compare the accurate adder *AccuAdd* with the *ApproxAdd*$_4$ (see Table II). Using the number of error cases as a metric, it is clear that *ApproxAdd*$_4$ has the lowest accuracy, however, if we calculate the *QAP=4*0*0*, the result is *0*. The *QAP* for the *AccuAdd* is *QAP=0*4.41*1130*, which will also result in 0. For this reason, the cost function *QAP* cannot be used for evaluation of distinctive design points in the architectural space of approximate multipliers.

In our methodology, we consider the quality constraint as an input from the system designer. Therefore, our methodology determines the cost of a design point as a function of only the area and power. Since the product of power and area may again lead to multiple points with a cost of *0*, we formulate the cost function as a *weighted sum of power and area* (i.e., WAP); see Eq. 2. w_A and w_P denote the weights

for area and power factors. A_X and P_X denote the area and power consumption of a given design point X.

$$WAP_X = w_A * A_X + w_P * P_X \quad (3)$$

The designer can specify which of the two (i.e., area or power) is of more importance for the design under consideration. Of course, area has an impact on the power consumption. However, our design space shows that, even two design points with similar area values could have very different power consumption, which is primarily due to more switching activity. Fig.7 shows the recursive partitioning tree for a large-sized multiplier until it reaches the leaf node, which corresponds to the elementary 2×2 multiply module. The figure shows the multiplier tree of a 16×16 multiplier as an example. The right-most-branch always represents the block of highest significance. As we move to the left, the significance decreases until we reach the least significance at the left-most-branch. In the above tree, if level 0 is the root, the most significant block is the right-most 2×2 block at the highest level (level 3). For this reason, a *depth-first* traversal of the multiplier tree (starting on the right) is used when exploring the possibilities of approximating the 2×2 modules.

Algorithm 1: Design Selection using DFS (Depth-First Search)

INPUT: (i) QMetric: the quality criteria, for example, the highest error magnitude; (ii) QConstraint: the condition to satisfy for the given metric (for example, QMetric < 5); (iii) W_A: area weight; (iv) W_P: power weight.
OUTPUT: (i) the selected configuration of 2-bit multipliers, adders, and LSBs; (ii) the associated quality metric
BEGIN

1. $WAP_{Add} = EvalWAPforAdders(W_A, W_P)$; //Evaluate WAP for all considered 1-bit FAs
2. $AdderList = AscendingSort(WAP_{Add})$;
3. $WAP_{Mult} = EvalWAPforMultipliers(W_A, W_P)$; *//Evaluate WAP for all considered 2x2 multipliers*
4. $MultiplierList = AscendingSort(WAP_{Mult})$;
5. $LSBsValues = DescendingSort(LSBValues)$;

6. **while** $(travesring\ multiplierTree\ depthFirst)$ **do**
7. **for** $all\ mult \in MultiplierList$ **do**
8. **for** $all\ add \in AdderList$ **do**
9. **for** $all\ lsb \in LSBValues$ **do**
10. $QCurrent = ApplyErrorModel(QMetric, mult, add, lsb)$;

11. **if** $(QCurrent\ meets\ QConstraint)$ **then**
12. **return** $current\ configuration$
13. **endif**
14. **end for**
15. **end for**
16. **end for**
17. **end while**

END

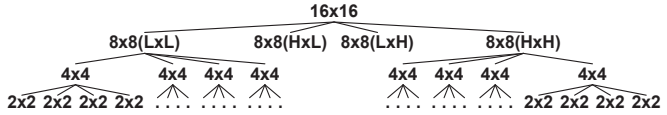

Fig.7: Recursive Partitioning for Building Large-Sized Multipliers.

For each 2×2 module combination selected while traversing the tree, we first need to select the elementary 2×2 approximate multiplier. This is done by evaluating all alternatives in the *MultiplierList* (see Algo.1) which is already sorted ascendingly, meaning that the *WAP-wise* best alternative is always at the top. For each multiplier alternative, all adder types are then tested for different LSB-combinations. The *AdderList* is also sorted ascendingly and having the *WAP-wise* best approximation at the top. On the contrary, the higher the LSB, the higher the approximation level (i.e., *WAP*-wise best). That is why those are sorted in a descending order. In Summary, the algorithm starts using the highest approximation available and moves toward a more accurate solution. The first configuration that satisfies the quality criteria is selected as the solution. Each combination is evaluated using an appropriate error model, which computes the quality of this specific combination (*QCurrent*). This *QCurrent* is then checked against *QConstraint* (the designer requirement given as input).

Note, the algorithm stops at the first solution that satisfies the constraints. It can alternatively, be left to run to provide an ordering of all possible points of the design-space.

Complexity Analysis: It is well-known that a DFS has a variable complexity according to the data structure used for the tree representation. A common approach is to use an adjacency list; in this case the complexity of DFS is O(V+E); where V and E are the number of vertices and edges respectively. Since the multiplier structure is fixed (see figure), a 64x64 multiplier would have 337 vertices and 336 edges. Assuming we have D adders, M multipliers, and B LSBs combinations, then the complexity would be O(D*M*B). For our library, this exploration process takes only a few minutes on an Intel Core i7 PC.

It is important to note that the focus is not to find an optimal multiplier, rather to find the best possible configuration given a certain quality metric and an instance of the library of blocks. It would be obvious that changing the metric or the library of components may/will change the selection.

Generalization: The above exploration is made for recursive multipliers that use adders for partial product summation. In case a fast multiplier does not need an adder tree, the decision will be based on the selection of elementary multiplier modules and the number of bits for approximation. Depending upon a particular design of a fast multiplier, selection methodology may need to be customized to achieve the best possible results.

3 RESULTS AND DISCUSSION

3.1 Experimental Setup and Tool Flow

Fig.8 presents our integrated tool flow and evaluation setup for designing and validating the approximate computing modules. Both structural description (VHDL) and behavioral description (C) were developed for each approximate design. *Synopsys Design Compiler* (DC) synthesizes the given VHDL design, and generates a netlist file and reports about area, latency, and power.

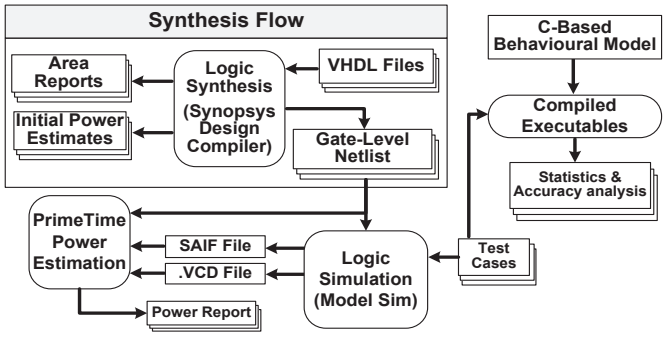

Fig.8: Tool Flow and Experimental Setup.

We ran the Synopsys Design Compiler using *WCCOM* (Worst-Case Commercial) operating conditions. *Wire_load_model* was set to segmented and the area optimization option was enabled. The

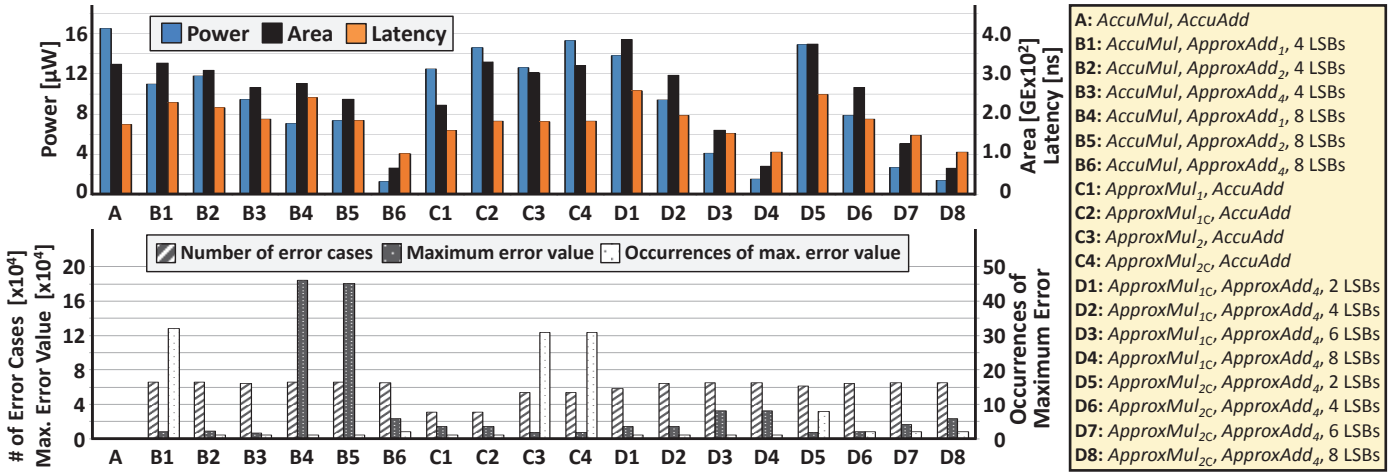

Fig.9: Evaluating the Area, Power, and Quality Architectural-Design Space of Approximate Multipliers; also showing the Legend of Multiplier Designs.

Table VI Comparing the Image Quality and Compression Efficiency when Applying Selected Approximate Multipliers to the JPEG Application. X1 [Accurate DCT and IDCT], X2 [Only DCT Approximated], X3 [Only IDCT Approximated], X4 [Approximated DCT and IDCT].

	X1	X2							X3							X4						
		B3	B6	C1	C3	D6	D7	D8	B3	B6	C1	C3	D6	D7	D8	B3	B6	C1	C3	D6	D7	D8
PSNR [dB]	31.42	20.5	20.43	31.05	22.72	19.79	20.15	20.49	22.32	21.26	31.15	27.31	22.34	21.12	21.4	17.38	21.68	30.92	20.26	16.28	21.37	21.68
Compressed file size [Bytes]	9,810	12,229	5,485	9,788	11,407	12,630	6,964	4,762	9,810							12,229	5,485	9,788	11,407	12,630	6,964	4,762

	X1	X2							X3							X4						
		B3	B6	C1	C3	D6	D7	D8	B3	B6	C1	C3	D6	D7	D8	B3	B6	C1	C3	D6	D7	D8
PSNR [dB]	30.56	21.97	21.47	30.18	23.94	21.2	21.08	21.52	23.22	21.99	30.44	26.39	22.74	21.53	21.97	18.61	22.29	30.12	26.37	17.47	22	22.15
Compressed file size [Bytes]	27013	33,343	13,927	26,689	29,635	34,462	17,802	11,806	27,013							33,343	13,927	26,689	29,635	34,462	17,802	11,806

generated netlist is then used as input to the gate-level simulation tool (*ModelSim*). Once the correct behavior of the hardware is tested and confirmed, *VCD* (Value Change Dump) and *SAIF* (Switching Activity Interchange Format) files are generated and fed into the *Primetime* tool to generate more accurate power reports. In order to evaluate the quality of each approximate design, we developed an equivalent behavioral model in C and performed extensive test to permute over all possible combinations of input operands. Afterwards, number of error cases, maximum error magnitude, and occurrences of maximum error cases are recorded for each design. Since studying the evaluation of weights is not the focus of this paper, and we consider it as a given input from the system designer, we use $w_A=0.5$ and $w_P=0.5$, i.e., treating both area and power equally important.

3.2 Evaluating the Design Space Points

Fig.9 illustrates a subset of the architectural space of an 8×8 approximate multiplier design along with the power, area, latency, and quality characterizations of different designs. There are 19 design-points using variants of multiplier type, adder type, and number of approximated LSBs. Some of these points also cover state-of-the-art approximate multipliers, for instance the ones that use $ApproxMul_1$. Under different constraint values, our methodology outputs 8 design points (*A*, *B3*, *B6*, *C1*, *C3*, *D6*, *D7*, and *D8*) that we then applied to a JPEG application for further quality assessment. Important points on the selection are discussed below.

1. The first selected design is the fully-accurate one, which also serves as the baseline.
2. Two out of the six "accurate multiplier with approximate adder" combinations are chosen, i.e., *B3* and *B6*. As shown, *B4* and *B5* have the highest maximum error in the *B*–category, so they are both excluded. On the other hand, *B1* and *B2* have the highest power and area attributes, they are also filtered-out. In the *B*–category, *B6* has the lowest area and power while having a high number of error cases with low occurrences. In contrast, *B3* has the least maximum error value and least occurrences but has high area/power characteristics. Therefore, *B3* and *B6* were favored since they offer interesting trade-offs between power and quality.
3. In the C–category (i.e., approximate multiplier with accurate adder), C1 and C3 are the approximate multipliers while C2 and C4 are their corresponding accuracy-configurable versions. *C1* has lower area than *C3* but higher maximum error value. We chose to include the approximate versions.
4. *D8* has the least power and area among the *D*–category (i.e., approximate multiplier with approximate adder). *D4* also warrants low area and power but has higher inaccuracies, so *D8* was preferred. *D3* is similar to *D4* regarding quality but has higher area and power, for this reason, it is not selected. *D5* and *D6* have the lowest maximum error value but *D6* exhibits better area and power characteristics. *D7* is an excellent trade-off in terms of area, power, and accuracy attributes.

3.3 Quality Evaluation of Selected Approximate Multipliers when Employed in a JPEG Application

Table VI illustrates the image quality and compression efficiency comparison when applying different design points to the DCT and IDCT functions of a JPEG application in different accurate/approximate combinations. When evaluating the efficiency of the selected approximation variants in terms of the size of the compressed file, we notice that for both images, when approximate DCT is employed, the selected variants always result in the same order of [*D8*, *B6*, *D7*, *C1*, *C3*, *B3*, *D6*]. Here, *D8* has the highest

reduction in compressed file size of more than 50% for both test images. *C1* at the borderline, where the compressed file size is nearly equal to the accurate version. This shows that combining several approximations, such as *D8*, can result in power and area savings (see Fig.9) and increased performance when compared to approximating either the adder or the multiplier.

When evaluating those variants for PSNR values, observations are, to a certain extent, contradicting. When sorting the approximate variants in order of decreasing PSNR, we get a different order for *X2*, *X3*, and *X4*. And also for the same *X*, the order differs with different images. For example, *B3* and *B6* are two counter cases, *B3* having higher power and area while having lower error. The PSNR values are almost equal, while there is a huge difference observed in the compressed file size to the extent that *B3* is worse than the accurate case with an increase of 23% in the compressed file size. Also, in contrast to the design-space results, when using PSNR as an indication *D8* always performs better than *D7* and *D7* performing in turn better than *D6*. In summary, the results show that simply using the PSNR metric is not sufficient, because amounts of bits also matter. Therefore, a designer needs to account for both PSNR and bit rate for making a power-efficient decision.

4 CONCLUSION

We presented a methodology to generate and explore the architectural space of approximate multiplier using variants of approximate/accurate elementary multiply/add modules and number of approximated LSBs. We performed ASIC-synthesis and Primetime power estimation for various designs using a 45nm technology, and evaluated for area, power, latency, and output quality (in terms of different error metrics). A subset of selected design points is applied to a real-world application of JPEG in different combinations and evaluated for output quality. We provide the complete library of approximate multipliers and adders (including both RTL and behavioral models) online at *https://sourceforge.net/projects/lpaclib/*. This open-source library enables reproducing and comparing the results and will enable further research and development on applying approximate computing at higher abstraction layers of HW/SW stacks.

This work is the first step towards the development of open-source elementary approximate modules, and a *generic* selection methodology for building large-sized multi-bit approximate arithmetic module using smaller elementary approximate modules, and selecting efficient configurations.

REFERENCES

[1] R. Nair, *"Big data needs approximate computing: technical perspective"*, ACM Communications, 58(1): 104, 2015.

[2] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, J. Henkel, *"Cross-Layer Approximate Computing: From Logic to Architectures"*, ACM/EDAC/IEEE 53rd Design Automation Conference (DAC), 2016.

[3] A. K. Mishra, R. Barik, S. Paul, *"iACT: A Software-Hardware Framework for Understanding the Scope of Approximate Computing"*, Workshop on Approximate Computing Across the System Stack (WACAS), 2014.

[4] J. Bornholt, T. Mytkowicz, K. S. McKinley, *"Uncertain<T>: Abstractions for Uncertain Hardware and Software"*, IEEE Micro 35(3): 132-143, 2015.

[5] J. Bornholt, T. Mytkowicz, K. S. McKinley, *"Uncertain: a first-order type for uncertain data"*, ASPLOS, pp. 51-66, 2014.

[6] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, *"Architecture support for disciplined approximate programming"*, International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2012.

[7] V. Chippa, S. Chakradhar, K. Roy, and A. Raghunathan, *"Analysis and characterization of inherent application resilience for approximate computing"*, Design Automation Conference (DAC), 2013.

[8] S. Misailovic, M. Carbin, S. Achour, Z. Qi, M. C. Rinard, *"Chisel: reliability- and accuracy-aware optimization of approximate computational kernels"*, OOPSLA, 309-328, 2014.

[9] G. Pekhimenko, D. Koutra, K. Qian, *"Approximate computing: Application analysis and hardware design"*, Online available: www.cs.cmu.edu/~gpekhime/Projects/15740/paper.pdf.

[10] V. Gupta, D. Mohapatra, S.P. Park, A. Raghunathan, *"IMPACT: IMPrecise adders for low-power approximate computing"*, International Symposium on Low Power Electronics and Design (ISLPED), pp. 409 – 414, 2011.

[11] V. Gupta, D. Mohapatra, A. Raghunathan, K. Roy, *"Low-Power Digital Signal Processing Using Approximate Adders"*, IEEE Transaction on CAD of Integrated Circuits and Systems 32(1): 124-137, 2013.

[12] M. Shafique, W. Ahmad, R. Hafiz, J. Henkel, *"A Low Latency Generic Accuracy Configurable Adder"*, IEEE/ACM Design Automation Conference (DAC), 2015.

[13] S. Mazahir, O. Hasan, R. Hafiz, M. Shafique, J. Henkel, *"An Area-Efficient Consolidated Configurable Error Correction for Approximate Hardware Accelerators"*, ACM/EDAC/IEEE 53rd Design Automation Conference (DAC), 2016.

[14] A. K. Verma, P. Brisk, P. Ienne, *"Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design"*. Design, Automation and Test in Europe (DATE), 2008.

[15] N. Zhu, W.-L. Goh, K.-S. Yeo, *"An enhanced low-power high-speed Adder for Error-Tolerant application"*, 12th International Symposium on Integrated Circuits (ISIC), 2009.

[16] A. B. Kahng, S. Kang, *"Accuracy-configurable adder for approximate arithmetic designs"*, IEEE/ACM Design Automation Conference (DAC), pp.820-825, 2012.

[17] R. Ye, T. Wang, F. Yuan, R. Kumar, Q. Xu, *"On reconfiguration-oriented approximate adder design and its application"*, International Conference on Computer-Aided Design (ICCAD), pp.48-54, 2013.

[18] J. Miao, K. He, A. Gerstlauer, M. Orshansky, *"Modeling and synthesis of quality-energy optimal approximate adders"*, International Conference on Computer Aided Design (ICCAD), pp. 728-735, 2012.

[19] P. Kulkarni, P. Gupta, M. Ercegovac, *"Trading Accuracy for Power with an Underdesigned Multiplier Architecture"*, 24th International Conference on VLSI Design (VLSI Design), pp. 346 – 351, 2011.

[20] M. B. Sullivan, E. E. Swartzlander, "Truncated error correction for flexible approximate multiplication", ASILOMAR, pp. 355–359, 2012.

[21] K. Y. Kyaw, W.-L. Goh, K.-S. Yeo, *"Low-power high-speed multiplier for error-tolerant application"*, IEEE International Conference of Electron Devices and Solid-State Circuits (EDSSC), 2010.

[22] K. Bhardwaj, P. S. Mane, J. Henkel, *"Power- and Area-Efficient Approximate Wallace Tree Multiplier for Error-Resilience Systems"*, ISQED, 2014.

[23] Open-Source Library of Low-Power Approximate Computing Modules: https://sourceforge.net/projects/lpaclib/.

[24] D. Palomino, M. Shafique, A. Susin, J. Henkel, *"Thermal Optimization using Adaptive Approximate Computing for Video Coding"*, IEEE/ACM 19th Design, Automation and Test in Europe Conference (DATE), 2016.

[25] F. Sampaio, M. Shafique, B. Zatt, S. Bampi, J. Henkel, *"Approximation-Aware Multi-Level Cells STT-RAM Cache Architecture"*, IEEE International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES), 2015.

[26] J. San Miguel, J. Albericio, A. Moshovos, N. E. Jerger, "Doppelgänger: A Cache for Approximate Computing", IEEE 48th International Symposium on Microarchitecture (MICRO), 2015.