INVITED: Cross-Layer Approximate Computing: From Logic to Architectures

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

[1]Chair for Embedded Systems, Karlsruhe Institute of Technology (KIT), Germany
[2]Information Technology University (ITU), Lahore, Pakistan
[3]Technische Universität Dresden (TUD), Dresden, Germany
Corresponding Author: muhammad.shafique@kit.edu

ABSTRACT

We present a survey of approximate techniques and discuss concepts for building power-/energy-efficient computing components reaching from approximate accelerators to arithmetic blocks (like adders and multipliers). We provide a systematical understanding of how to generate and explore the design space of approximate components, which enables a wide-range of power/energy, performance, area and output quality tradeoffs, and a high degree of design flexibility to facilitate their design. To enable cross-layer approximate computing, bridging the gap between the logic layer (i.e. arithmetic blocks) and the architecture layer (and even considering the software layers) is crucial. Towards this end, this paper introduces open-source libraries of low-power and high-performance approximate components. The elementary approximate arithmetic blocks (adder and multiplier) are used to develop multi-bit approximate arithmetic blocks and accelerators. An analysis of data-driven resilience and error propagation is discussed. The approximate computing components are a first steps towards a systematic approach to introduce approximate computing paradigms at all levels of abstractions.

Keywords: *Approximate Computing, Logic, Arithmetic, Architecture, Adder, Multiplier, Accelerator, Configurable Accuracy, Cross-Layer, Performance, Low Power, Energy Efficiency, Design Space.*

1 Introduction

Approximate Computing is, though as old as computing itself, a re-emerging paradigm for building highly power-/energy-efficient systems. The basic problems of power density according to the discontinuation of Dennard Scaling force to rethink. As the process technology shrinks and the per-transistor performance/power efficiency is not keeping pace with the well-known power-reduction techniques (like DVFS and power-gating) at various abstraction layers, continuing to support *precise* computing across the stack is most likely not sufficient to solve the rising energy-efficiency challenges.

Approximate Computing (aka InExact Computing) relies on relaxing the bounds of precise/exact computing to provide new opportunities for improving the area, power, and performance efficiency of systems by orders of magnitude at the cost of reduced output quality [1]-[6]. Recently, investigations by Intel [1], IBM [2], Microsoft [3][4], and other research groups [6] have shown that there is a large body of resource-hungry applications that can tolerate approximation errors and a significant portion of their functions/computations still produces outputs that are useful and of acceptable quality for the users[1]. Examples: deep learning networks, big data analytics, recognition and machine learning, web searches, image and video processing, computer vision, artificial intelligence, high performance computing, etc. Their inherent resilience to approximation errors can be attributed to several factors like:

1) *Existence of noise and redundancy in real-world input data*, for instance, camera and other type of sensors;
2) *Perceptual limitations of different application users*, for instance, slight change in pixel value or similar effects may not be perceived by humans due to their psycho-visual limits;
3) *Error attenuation characteristics of processing algorithms*, for instance, motion tracking and data/pattern classification;
4) *Lack of a golden answer*, for instance, in web searches and recommendation systems, CAD tools and meta-heuristics, there exists multiple answers that are equally acceptable for the users.

Moreover, in several cases *either* the best available algorithms cannot produce correct results all the time *or*, it is too expense to produce fully correct or optimal results (e.g., meta-heuristics and probabilistic algorithms). In short, the application world in many cases is already approximate. Similarly, each new technology node faces serious reliability threats [19], which may lead to different types of hardware-level faults. So far, the designers have mostly assumed a notion of exact (numerical or Boolean) equivalence between the specification and implementation of different layers of the hardware and software stacks; see left side of Fig.1.

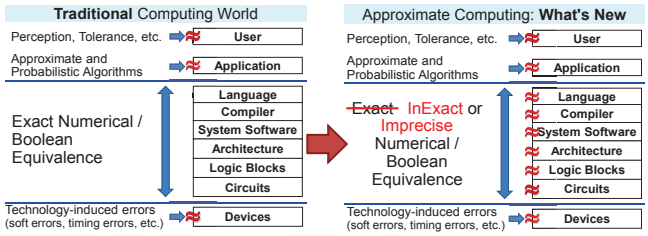

Fig.1: Towards Cross-Layer Approximate Computing from Exact to Imprecise Notion across Different Layers (adapted from [18]).

Approximate computing aims at relaxing the commonly applied notion of exact (numerical or Boolean) equivalence between the specification and implementation at multiple layers of the hardware-software stacks to tradeoff computation accuracy with performance, power, and area; see right side of Fig.1. Unlike early works on probabilistic and stochastic computing that target general purpose processors, recent works on approximate computing are primarily targeting applications-specific systems, so that their inherent resilience can be exploited to the best *without* introducing intolerable errors to the critical hardware/software components. There is a plenty of work done on isolated system layers, like circuit and arithmetic blocks in [7]-[17], and programming language works in [3]-[5].

Required: To enable cross-layer approximate computing from circuit

* This work was conducted when Semeen Rehman was with the Chair for Embedded Systems (CES) at Karlsruhe Institute of Technology (KIT)

DAC '16, June 05-09, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4236-0/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2897937.2905008

[1] A *user* can either be an end-user or a system / sub-system that uses the output of an approximate function to do some further processing.

Table I Approximate Computing Techniques at Different Abstraction Layers of the Hardware and Software Stacks.

Layers	Techniques	Related Work	Short Description Regarding the Technique	Motivation	Case Study	Dependencies on Other layers
Software	Selective Approximation	[38]	Adaptively skips different function executions for the prediction parts along with the data/operation decimation depending upon the video properties	Improved Thermal Profile	HEVC video encoder	no
		[20][21]	Automatically identifies error-resilient parts within a code which can be skipped (*Code Perforation*) while keeping the error within predefined range	Improved performance	Recognition, Mining and Synthesis (RMS)	no
	Timing Relaxation	[22][23]	Relaxing synchronization in parallel programs and by exploiting the properties of serial or iterative convergence algorithms for increasing parallelism by eliminating dependencies between computations.		Recognition and Mining (RM)	no
	Domain specific approximation	[25][26]	Utilizes domain specific knowledge for proposing approximate models, in some cases scalable models, with improved performance		Machine Learning, applications, etc.	-
	Functional Approximation	[24]	Complex approximatable segments of codes are replaced with trained Approximate Neural Networks (ANN) through parrot transformation which are executed using NPU augmented processors		fft, inversek2j, jmeint, jpeg, kmeans, sobel	yes, architectural and H/W
Archi-tectural	Approximate Cache	[39]	Leveraging the video properties for data approximations through shutting down the error correction in MLC-STTRAM based Cache architectures	power efficient	HEVC video encoder	Yes, application software
	Approximation in Data storage	[27][28]	Hybrid memories with *power efficient unreliable memory* arrays allows aggressive voltage over-scaling for improving power efficiency	Power/memory efficient	video processing/ vision applications	yes, hardware
	Selective Approximation	[4][29]	Executes some chosen instructions or code segments in approximate mode, i.e. on approximate hardware	Improved performance	fft, sor, mc, smm, lu, zxing, jmeint, imagefill, raytracer, RMS	yes, hardware
	Domain specific approximation	[30]-[31]	Utilizes domain specific knowledge for designing application specific accelerators	Power efficient, Improved performance	RMS and vision applications etc.	-
	Functional Approximation	[7]-[9] [11][13][14] [32][33]	Truncation of critical paths in circuits to increase performance at the cost of accuracy		DSP, Vision/Image Processing, RMS applications etc.	no
HW/ Circuit	Timing Relaxation	[34][35]	Deliberate reduction of voltage for improving power efficiency of circuits i.e. voltage over-scaling	power efficient	RMS and vision applications etc.	no
	Functional Approximation	[12]	Reduction of hardware complexity of circuits using approximate equivalent models with less number of transistors			no

to all the way up to the application layer, there is a need to bridge the gap between the logic, architecture and software layers. The key is to provide a wide-range of power/performance/area vs. quality configurations for approximate components.

1.1 Contributions of This Paper

This paper introduces concepts for building (application-specific) approximate computing architectures. This paper presents:

1) A Survey of Approximate Computing Techniques covering different layers of the hardware and software stacks. (Section 2).

2) Approximate Adders and Characterization (Section 4): We implemented two different design approaches: (I) Multiple *low-power 1-bit approximate full-adders* [11][12] that serve as basis for building more complex multi-bit approximate adders; and (II) A *Generic Accuracy Configurable (GeAr)* adder model [14] that provides a generic model of building multi-bit approximate adders by carry approximation and prediction. It thereby enables a wide range of accuracy configurability and variable approximation modes. We show that GeAr's error probability model allows convenient comparison of different configurations and hence enables the designer to wisely select an appropriate configuration.

3) Approximate Multipliers (Section 5): An approximate multi-bit multiplier is composed of 2x2 approximate multiplier blocks and different multi-bit approximate adders for partial product summation.

4) Multi-Accelerator Approximate Computing Architecture (Section 6): A methodology to develop accelerators with variable approximation modes, using multi-bit approximate adders and multipliers, is presented. It provides a foundation for realizing approximate computing architectures with a sea of accelerators.

5) Open Source Libraries of Low-Power and High-Performance Approximate Arithmetic Components: The synthesizable VHDL files and behavioral models in C/MATLAB are provided as open-source libraries at: https://sourceforge.net/projects/approxadderlib/ and https://sourceforge.net/projects/lpaclib/.

This paper is part of DAC 2016 Special Session *"Cross-Layer Approximate Computing: Challenges and Solutions"*. Other papers in this special session are "Cross-Layer Approximations for Neuromorphic Computing: From Devices to Circuits and Systems [40], "Approximate Computing with Partially Unreliable Dynamic Random Access Memory: Approximate DRAM" [41], and "Programming Uncertain Things" [42].

2 A Survey of Approximation Techniques

An overview of different state-of-the-art techniques at different hardware/software layers is presented in Table I. We broadly classify these schemes into following five categories (Table III).

Table II Classification of Types of Approximations.

Selective Approximation	Analysis of software code or instructions to suggest a certain accuracy mode for a part of code
Timing Relaxation	Relaxing of synchronization, timing and handshaking constraints to reduce control overhead
Functional Approximation	An approximate alternative of an algorithm that improves area/power performance
Domain Specific Approximation	Leveraging the domain specific knowledge for approximations in applications and their algorithms
Data/Information Approximations	Use of unreliable memories, load value approximation, data truncation, data decimation, etc.

This classification reveals that most of the approximation schemes may be applied at multiple layers for achieving the desired goals, i.e., improved performance, power efficiency or area reduction. The table also exposes the potential for improvement. For instance, *Timing Relaxation* may be explored at the architectural layer by relaxing synchronization between various finite state machines working in tandem. The table also highlights inter-layer dependencies and application areas of each technique.

Recent advances in Approximate Computing have been highlighted through a dedicated Special Issue on the topic in the IEEE Design and Test Magazine [43]. A wide range of techniques like Approximate Register File for GPUs [44], Approximate Load Value Prediction [45], Error Prediction for approximate accelerators [46], and RRAM-Based Analog Approximate Computing [47], have been explored.

3 Tool Flow and Experimental Setup

Fig.2 presents the integrated tool flow used for development and

evaluation of our approximate designs (adders, multipliers, and accelerators) presented in this paper. Depending upon the type, a given approximate design is implemented using structural and RTL description (in VHDL or Verilog) which is then synthesized using Synopsis Design Compiler (DC, for the ASIC design flow) or using XILINX ISE (for the FPGA design flow). The generated netlist are simulated on ModelSim for functional verification, and to obtain VCD (Value Change Dump) and SAIF (Switching Activity Interchange Format) files for power estimation. In the ASIC design flow, the PrimeTime tool is used for power estimation. For output quality evaluation in application scenarios, an equivalent behavior model (in C or MATLAB) is developed. Selected implementations are also used for error probability analysis. Different designs are compared for area, performance/power, and output quality.

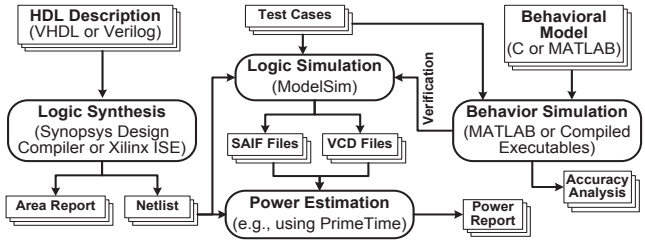

Fig.2: Experimental Setup.

4 Approximate Adders

Approximate adders provide inexact results within certain error bounds through two complementary methods: (1) approximating the logic of 1-bit full adder (FA) using circuit simplification (see Section 4.1); and (2) approximating the logic of carry propagation through breaking the chain of carry (see Section 4.2). In the latter case, error probability can be reduced through carry prediction using the overlapping sub-adders or carry-lookAhead like logic. Furthermore, in case of configurable designs, error correction can be obtained by employing an adaptive error detection and correction unit. In Section 4.2, we will discuss our *GeAr* adder model [14] that provides a generalized model of building multi-bit approximate adders while also accounting for reducing carry propagation error through overlapping sub-adders and a light-weight configurable error correction. Depending upon the resilience of an application, it may happen that an erroneous output of approximate adder does not lead to an error in the output of a given application function. Such a resilience depends upon the error masking properties of subsequent operations in the data flow, and the control flow probabilities of the application program. This error masking analysis can be applied to design approximate accelerators.

4.1 1-Bit Approximate Full Adders

Table III Truth Tables and Characterization for Different Approximate 1-Bit Full Adders based on the IMPACT design of [11][12].

Inputs			AccuFA		$ApxFA_1$		$ApxFA_2$		$ApxFA_3$		$ApxFA_4$		$ApxFA_5$	
A	*B*	C_{in}	*Sum*	C_{out}	*Sum*	C_{out}	*Sum*	C_{out}	*Sum*	C_{out}	*Sum*	C_{out}	*Sum*	C_{out}
0	0	0	0	0	0	0	**1**	**0**	**1**	**0**	0	0	0	0
0	0	1	1	0	1	0	1	0	1	0	1	0	**0**	**0**
0	1	0	1	0	**0**	**1**	1	0	**0**	**1**	**0**	**0**	1	0
0	1	1	0	1	0	1	0	1	0	1	**1**	**0**	**1**	**0**
1	0	0	1	0	**0**	**0**	1	0	1	0	**0**	**1**	**0**	**1**
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
1	1	0	0	1	0	1	0	1	0	1	0	1	**1**	**1**
1	1	1	1	1	1	1	**0**	**1**	**0**	**1**	1	1	1	1
Area [GE]			4.41		4.23		1.94		1.59		1.76		0	
Power [nW]			1130		771		294		198		416		0	
#Error Cases			0		2		2		3		3		4	

We have implemented an accurate 1-bit full adder (*AccuFA*) and several approximate versions (*ApxFA*) based on the state-of-the-art IMPACT adder designs from [11][12]. The truth tables of these adders are provided in Table III, which can also be used to develop equivalent gate-level descriptions. Error Cases are shown in bold red. These designs mainly rely on logic simplification, for instance, $ApxFA_3$ inverts the C_{out} bit to compute the *Sum*. $ApxFA_5$ represents the most aggressive approximation by assigning inputs to the outputs, i.e., $C_{out} = A$; and $Sum = B$. The area, power, and quality results of these adders, following the experimental methodology of Section 3, are presented in last three rows of Table III.

4.2 GeAR: Generic-Accuracy Configurable Adders

Careful exploration of design space of power/performance/area and output quality demands support for accuracy configurability and error modeling support across the complete hardware/software stack. As an example, an instruction set architecture that provides various accuracy bounds on a certain "add" instruction shall demand an equivalent approximate adder that can be configured to match the accuracy requirements. Furthermore, without proper error modeling support the compiler may not be able to correctly simulate the performance on the approximate adder to be used. Towards this end, we developed the *GeAr* adder [14], which is an open-source *accuracy configurable* and highly parameterizable adder model. We also developed corresponding error models and correction circuitry. The error models allow evaluating a particular configuration and hence can be linked to a compiler that can approximate instructions.

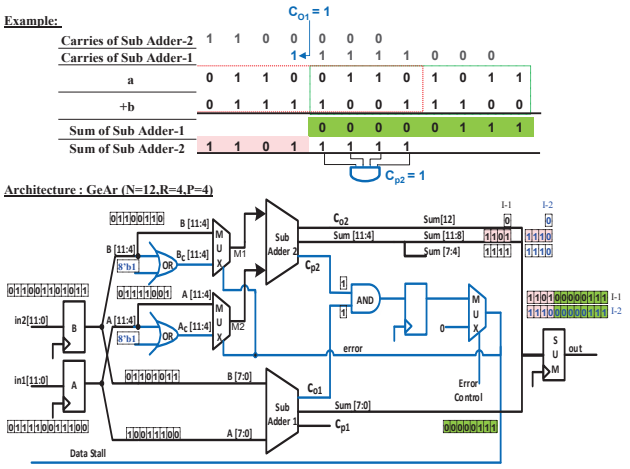

Fig.3: GeAr adder architecture and illustration [14].

In order to add two *N* bit operands, *GeAr* adder makes use of *k L-bit* sub-adders in parallel to perform the approximate addition, where *L*<=*N*, as shown in Fig.3. Let *R* be the number of resultant bits contributing to the final sum and *P* be the number of previous bits used for carry prediction in each sub-adder. Each sub-adder produces *R-bits* result except the first sub-adder which produces *L-bits* result where *L=R+P*. For a given *N*, *R*, and *P* the number of required sub-adders *k is given by* $k=((N-L)/R)+1$. Fig.3 provides an illustrative example for the addition of two 12-bit numbers using a GeAr configuration with N=12, R=4, P=4. GeAr adder provides a reduced delay as compared to an *N-bit* accurate adder since the carry propagation is now limited to *L* bits only. An error is generated when the P bits of any sub-adder are in propagation mode (C_{p2}=1) and a carry was generated by previous sub-adder (C_{o1}=1). Thus, an optional *iterative error detection and recovery circuitry* (highlighted in blue in Fig.3) was incorporated. Whenever, an error is detected the LSB of sub-adder inputs are forced to 1 in the subsequent iteration to correct the error. Various architectural configurations of the GeAr adder are implemented in Verilog and synthesized using Xilinx ISE and implemented for Xilinx Virtex 6 XC6VLX75T FPGA. Functional

verification was performed using MATLAB.

GeAr Error Model: Assuming, the occurrence of *1* or a *0* for each input bit as an equally likely event, the propagate probability ($\rho[Pr]$) and generate probability ($\rho[Gr]$) becomes *0.5* and *0.25*, respectively. Each sub-adder except the first sub-adder will generate R error generating events, Z_m for $m=1,2,\ldots,R$. Hence for k sub-adders there will be $R\times(k-1)$ error generating events $Z_{m+(s\times R)}$ for $s=1,2,\ldots,(k-2)$. Considering all the error generating cases, the probability of error associated with the adder was generalized to be.

$$\rho[Error]=\rho\left[\bigcup_{i=1}^{R\times(k-1)}Z_i\right]=\sum_{j=1}^{R\times(k-1)}\rho[Z_j]-\sum_{j<k}\rho[Z_j\cap Z_k]$$
$$+\ldots+(-1)^{R\times(k-1)+1}\rho\left[Z_1\cap\ldots\cap Z_{R\times(k-1)}\right]$$

Using this model, the error probability for any *GeAr* architectural configuration can be calculated. This allows fast evaluation of various adder configurations without exhaustive simulations.

GeAr design space: Different combinations of *R and P* for an N-bit GeAr adder results in approximate adder designs with different area/performance/accuracy tradeoff. Furthermore, various configurations of GeAr adder model directly translate to state-of-the-art approximate adders (for instance, ACA-I [7], ACA-II [9], ETAII [8] and GDA [13]. GeAr error model thus enables fast exploration of design space of approximate adders when working at a higher abstract layer of the system stack. As an illustration, Table IV provides the percentage accuracy computed using the given error model for all the possible R/P possibilities for an 11-bit GrAr adder. The table also provides the area requirement (in terms of Virtex 6 XC6VLX75T LUTs) for each configuration. For the constraint of maximum accuracy percentage, GeAr (R = 1, P = 9) can be selected.

Table IV Accuracy/Area tradeoff for all possible N=11 GeAr adder

Accuracy									
P \ R	1	2	3	4	5	6	7	8	9
1	0	2.8	4.9	17	31	38	40	63	63
2	9.1	18	33	47	59	64	77	89	-
3	31	49	60	75	78	85	92	-	-
4	65	75	83	87	92	97	-	-	-
5	83	89	93	95	98	-	-	-	-
6	94	96	98	99	-	-	-	-	-
7	98	99	100	-	-	-	-	-	-
8	99	100	-	-	-	-	-	-	-
9	100	-	-	-	-	-	-	-	-

Area (LUTs) requirement									
P \ R	1	2	3	4	5	6	7	8	9
1	12	11	18	17	12	14	16	18	20
2	12	18	15	18	14	16	18	20	-
3	20	16	18	14	16	18	20	-	-
4	25	24	21	16	18	20	-	-	-
5	36	21	16	18	20	-	-	-	-
6	35	24	18	20	-	-	-	-	-
7	32	18	20	-	-	-	-	-	-
8	27	20	-	-	-	-	-	-	-
9	20	-	-	-	-	-	-	-	-

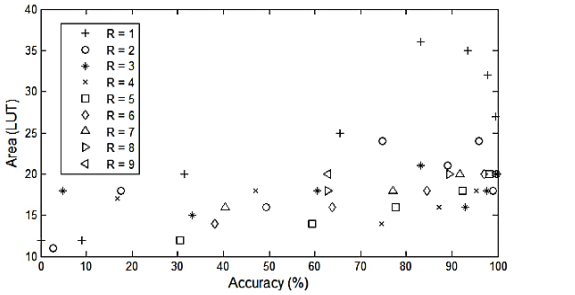

Fig.4: Area/Accuracy design space for an 11 bit GeAr adder

Fig.4 provides a pictorial view of the Accuracy/Area design space of the GeAr adder. R = 1 is represented by a red color plus '+' symbol, R = 2 by green circle 'o' symbol and similarly all the remaining R bits are represented. The legend in the figure shows all the representations for R bits. The combined metrics enables the user to select GeAr adder that meets multiple constraints. As an example, to find a low-area adder configuration with at least 90% accuracy, GeAr adder configuration of R = 3 (the star symbol) can be selected. Since there are multiple configurations for R = 3, Table IV can be consulted to identify the required configuration as R = 3 and P = 5.

5 Approximate Multipliers

Efficient multiplier designs (like Wallace Tree) incorporate small-sized multipliers along with an adder tree to sum the partial products. Therefore, to realize an approximate multi-bit multiplier, first an approximate 2x2 multiplier is required. The state-of-the-art 2x2 approximate multiplier ($ApxMul_{SoA}$) in [15] eliminates the 4th bit in the output as a mean of approximation; see truth table in Fig.5. In this case, only the product of 3 by 3 is incorrect, i.e. 7 instead of 9. This leads to a maximum error value of 2. In case the constraint on the maximum error value is 1, such a design cannot be used. Therefore, we developed a novel 2x2 approximate multiplier ($ApxMul_{Our}$) that simply connects the most significant bit (MSB) to the least significant bit (LSB); see truth table in Fig.5. This leads to an error value of 1, but in more cases. In short, depending upon the bound on the maximum error value or number of error cases, either $ApxMul_{SoA}$ or $ApxMul_{Our}$ can be deployed in approximate accelerators. Table I presents the truth tables and characterization of accurate multiplier (*AccMul*) and (non-)configurable versions of $ApxMul_{SoA}$ and $ApxMul_{Our}$. It is important to note that our configurable version $CfgMul_{Our}$ incurs less area and power due to a simple correction via an inverter in contrast to an extra addition for the $CfgMul_{SoA}$.

$ApxMul_{SoA}$

	00	01	10	11
00	000	000	000	000
01	000	001	010	011
10	000	010	100	110
11	000	011	110	***111***

Correction: *Adder*

$ApxMul_{Our}$

	00	01	10	11
00	0000	0000	0000	0000
01	0000	***0000***	0010	***0010***
10	0000	0010	0100	0110
11	0000	***0010***	0110	1001

Correction: *Inverter*

LEGEND	
AccMul	Accurate Multiplier
$ApxMul_{SoA}$	State-of-the-Art (SoA) Approximate Multiplier
$CfgMul_{SoA}$	SoA Configurable Multiplier
$ApxMul_{Our}$	Our Approximate Multiplier
$CfgMul_{Our}$	Our Configurable Multiplier

	AccMul	$ApxMul_{SoA}$	$CfgMul_{SoA}$	$ApxMul_{Our}$	$CfgMul_{Our}$
Area [GE]	6.880	3.704	7.232	4.939	6.350
Power [nW]	542.9	363	525	262	379
No. of Error Cases	0	1	-	3	-
Max. Error Value	0	2	-	1	-

Fig.5: Comparing 2x2 Accurate and Approximate Multipliers.

Several multi-bit approximate multipliers are then built by using different types of 1-bit approximate full adders, different types of approximate 2x2 multipliers, and different numbers of LSBs to be approximated in multi-bit approximate adders used for partial product summation. Fig.6 illustrates area and power results for accurate and different multi-bit approximate multipliers.

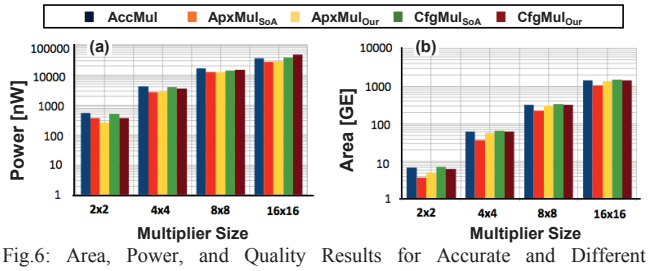

Fig.6: Area, Power, and Quality Results for Accurate and Different Approximate Multipliers of 2x2, 4x4, 8x8, and 16x16 bit-widths.

6 Multi-Accelerator Approximate Computing Architectures

In this section we provide our approach on systematic development (application-specific) of approximate computing architectures that consist of a number of accelerators with variable approximation levels. Fig.7 presents the flow of generating approximate accelerators from the basic approximate logic blocks (i.e. elementary or multi-bit approximate adder, subtractor, multiplier, divider, etc.) along with the characterization and design space exploration steps. The library of approximate logic block (see Sections 4 and 5) is characterized for power, area, performance, and output quality (in terms of maximum error value or total number of error cases). Based on this characterization a set of pareto-optimal points is selected in the design space exploration process, which is then used to develop multi-bit approximate logic blocks (adders, multipliers, etc.). For this, a statistical error analysis is also important to adopt appropriate

basic approximate logic blocks. This helps in fast quality analysis of individual blocks without extensive numerical simulations and quick exploration. These multi-bit approximate logic blocks are then used for generating approximate accelerators. These accelerators can either be generated manually (as done in this paper) or using specialized high-level synthesis (HLS) techniques/tools for approximate computing, which is an interesting research problem.

In order to have low-quality degradation but high power savings, it is important to analyze the error masking and propagation behavior in the accelerator data path. It may happen that some logical operations may mask the erroneous output of approximate adders/multipliers. Performing such a statistical error analysis and leveraging it to automatically generate efficient approximate accelerators is an interesting open research problem for the community towards cross-layer approximate computing.

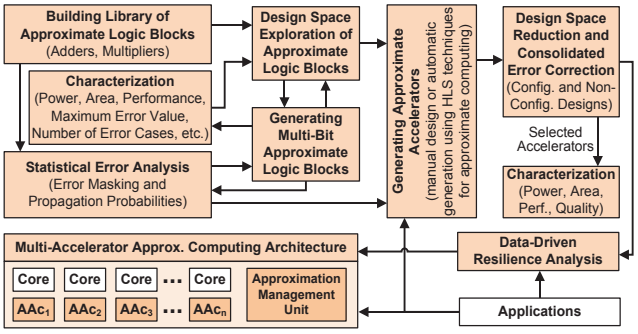

Fig.7: Methodology Flow for Creating Approximate Accelerators and Integration into a Multi-Accelerator Approximate Computing Architecture.

In case of adaptive systems, where an accelerator is required to operate sometimes in approximate mode and sometimes in accurate mode, or need to adaptively change the level of approximation, usage of configurable adder/multiplier blocks is required. A configuration word can then set the control bits of different approximate logic blocks in the accelerator data path. Furthermore, in order to reduce the area/power overhead, statistical error analysis of these approximate logic blocks can be exploited to design consolidated error correction units; see Section 6.1. A wide-range of diverse approximate accelerators in a multi-accelerator approximate computing architecture enables a high degree of flexibility and adaptivity. For a set of concurrently executing applications, an appropriate set of accelerators and their appropriate approximation modes are selected by the approximation management unit, such that, the performance and quality constraints of those applications are met and the overall power is minimized. Developing such an approximation management unit is also an open research problem. Such a management unit needs to account for the variable resilience properties of different applications, and even different functions within an application. Furthermore, data-driven resilience also needs to be accounted for run-time approximation control in such architectures as motivated in Section 6.2

Case Study on High Efficiency Video Coding: Fig.8 presents the output value results for different approximate versions of the SAD (Sum of Absolute Differences) accelerator used in the motion estimation function of a typical video codec (like H.264 or HEVC). It is important to note that the whole error surface for the approximate case is shifted and roughly follows the same trend. This means that the "global minima" (i.e. the best candidate block with the minimum SAD value) remains the same. As a result, the output of the motion estimation process (i.e. the best candidate as the motion vector) is unchanged despite the fact that different types of approximate adders and subtractors are employed in implementing different approximate variants of the SAD accelerator. Usage of a particular type of approximate adder/subtractor is one point. The number of LSBs to be approximate however is yet another point. Fig.9 shows the impact of approximate LSBs for different approximated SAD variants (each with a particular type of approximate adder) when employed in the latest High Efficiency Video Codec (HEVC). The impact of approximations on the output quality is plotted in the form of increase in the output bit-rate (compressed bits produced per second) compared to the accurate adder. Basically, HEVC compresses the current block using the information of the best predictor, obtained via SAD. We noticed that approximating 6-bits of the adders in the SAD accelerator results in a large increase in the bit-rate, which is typically unacceptable by the end-user. However, approximating 2-bits and 4-bits result in a marginal bit-rate increase. Furthermore, we noticed that approximating 4-bits always resulted in an overall lower power consumption compared to approximating the 2-bits, for all types of approximate adders (see Section 4). Therefore, in this case study, $ApxSAD_2$ or $ApxSAD_3$ with 4-bit approximations would be the most reasonable solution providing a good tradeoff between power reduction and output quality degradation.

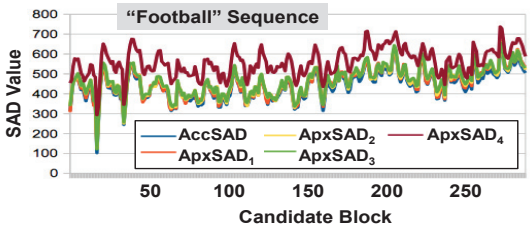

Fig.8: Different Variants of Approximate Accelerator for Sum of Absolution Differences (SAD) block used in the Motion Estimation process.

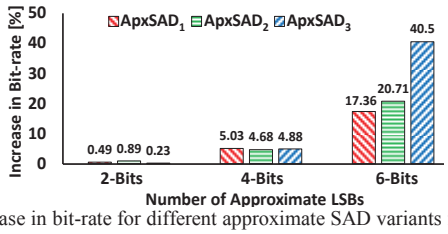

Fig.9: Increase in bit-rate for different approximate SAD variants for different number of approximate LSBs.

6.1 Consolidated Error Correction

Most state-of-the-art accuracy configurable approximate adders utilize an integrated Error Detection and Correction (EDC) circuitry. Consequently, the accumulated area overhead due to the EDC (integrated within individual adders) is significant. In our recent work [37], we proposed a low-cost Consolidated Error Correction (CEC) unit that essentially corrects the accumulated error at the accelerator output. It was observed that the magnitude of error in most of the approximate adders could only have certain specific values. Thus, for a cascade of adders the error recovery shall only comprise addition of a certain offset value to compensate for the imprecision. The scheme can take further benefit from cross-layer analysis to save valuable area/energy being utilized by the individual error correction units. The functionality of the proposed design was verified by applying it to our open source GeAr adder.

6.2 Data-Dependent Resilience Aspects

Most state of the art approximate arithmetic units [7]-[9], [11], [13], [14], [32], and [33] assume uniform distribution for the input data. We show that this may not always be the case and hence there is a need to explore data dependent resilience of various applications. We applied two variants (accurate and approximate) of low pass filter on a random set of input images. We compared the output quality in terms of SSIM (Structure Similarity Index Measure) [36], a psycho-visual measure of error, to assess the degradation. It was observed (Fig.10) that for the same adder and kernel, the achieved accuracy varied across the images. Thus, an input image or a video may have a

varying degree of psycho-visual resilience to a particular approximate circuit depending upon the content. Hence, detailed investigation of data-driven resilience and its exploitation towards configurable approximation control is an important research problem to be solved for cross-layer approximate computing.

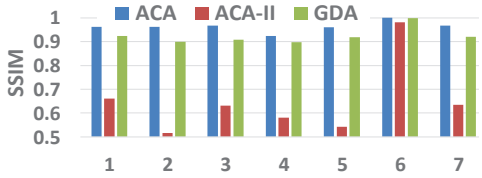

Fig.10: SSIM value for 7 images after low pass filtering on approximate hardware

7 Conclusion

In order to cope with the power density and energy efficiency problems, and in order to transgress the boundaries of commonly-used power management techniques, approximate computing may be a solution when applied across all layers of hardware and software. Key challenges: transferring the output quality constraints to different hardware/software layers, leveraging the data-driven resilience for adaptive approximation control, and bridging the gap between hardware and software to enable hardware/software coordinated approximates while minimizing the quality loss or achieving a much higher power savings.

We are providing open-source libraries of approximate components at: https://sourceforge.net/projects/approxadderlib/ and https://sourceforge.net/projects/lpaclib/. We believe that these open-sourcing efforts are necessary to expedite the research and development simultaneously at multiple system layers, and to facilitate reproducible results and research.

8 REFERENCES

[1] 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.

[2] R. Nair, "Big data needs approximate computing: technical perspective", ACM Communications, vol. 58, no. 1, pp. 104, 2015.

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

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

[5] S. Misailovic, M. Carbin, S. Achour, Z. Qi, M. C. Rinard, "Chisel: reliability- and accuracy-aware optimization of approximate computational kernels", ACM International Conference on Object Oriented Programming Systems Languages & Applications (OOPSLA), 309-328, 2014.

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

[7] A. K. Verma, P. Brisk, P. Ienne, "Variable Latency Speculative Addition: A New Paradigm for Arithmetic Circuit Design", IEEE/ACM Design, Automation and Test in Europe (DATE), pp. 1250 - 1255, 2008.

[8] N. Zhu, W.-L. Goh, K.-S. Yeo, "An enhanced low-power high-speed Adder for Error-Tolerant application", International Symposium on Integrated Circuits (ISIC), pp. 69 - 72, 2009.

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

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

[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 (TCAD), vol. 32, no. 1, pp. 124-137, 2013.

[12] 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.

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

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

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

[16] M. B. Sullivan, E. E. Swartzlander, "Truncated error correction for flexible approximate multiplication", Asilomar Conference on Signals, Systems and Computers (ASILOMAR), pp. 355–359, 2012.

[17] K. Bhardwaj, P. S. Mane, J. Henkel, "Power- and Area-Efficient Approximate Wallace Tree Multiplier for Error-Resilience Systems", International Symposium on Quality Electronic Design (ISQED) , pp. 263-269, 2014.

[18] A. Raghunathan and K. Roy, "Approximate Computing Across the Stack: Architecture and Systems", Approximate Computing Workshop (**AC**), 2015.

[19] J. Henkel, L. Bauer, N. Dutt, P. Gupta, S. Nassif, M. Shafique, M. Tahoori, N. Wehn, "Reliable On-Chip Systems in the Nano-Era: Lessons Learnt and Future Trends", ACM/IEEE Design Automation Conference (DAC), 2013.

[20] H. Hoffmann, S. Misailovic, S. Sidiroglou, A. Agarwal, M. Rinard, "Using code perforation to improve performance, reduce energy consumption, respond to failures", MIT Technical Report: MIT-CSAIL-TR 2009-042, 2009.

[21] S. Sidiroglou, S. Misailovic, H. Hoffmann, M. Rinard, "Managing Performance vs. Accuracy Trade-offs With Loop Perforation", ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE), pp.124-134, 2011.

[22] Sasa Misailovic, Deokhwan Kim, Martin Rinard, "Parallelizing sequential programs with statistical accuracy tests", MIT Technical Report: MIT-CSAIL-TR-2010-038, 2010.

[23] J. Mengte, A. Raghunathan, S. Chakradhar, S. Byna, "Exploiting the forgiving nature of applications for scalable parallel execution", IEEE International Symposium on Parallel & Distributed Processing (IPDPS), pp. 1-12 2010.

[24] H. Esmaeilzadeh, A. Sampson, L. Ceze, D. Burger, "Neural acceleration for general-purpose approximate programs", International. Symposium on Microarchitecture (MICRO), pp. 449–460, 2012.

[25] S. Venkataramani, A. Ranjan, K. Roy, A. Raghunathan, "AxNN: energy-efficient neuromorphic systems using approximate computing", International symposium on Low power electronics and design (ISLPED), pp. 27–32, 2014.

[26] S. Venkataramani, A. Raghunathan, J. Liu, M. Shoaib, "Scalable-effort classifiers for energy-efficient machine learning", ACM/IEEE Design Automation Conference (DAC), 2015.

[27] I. J. Chang, J. Chang, D. Mohapatra, K. Roy, "A Priority-Based 6T/8T Hybrid SRAM Architecture for Aggressive Voltage Scaling in Video Applications", IEEE Transactions on Circuits and Systems for Video Technology (TCSVT), vol. 21, no. 2, pp. 101-112, 2011.

[28] G. Karakonstantis, D. Mohapatra, K. Roy, "Logic and memory design based on unequal error protection for voltage-scalable, robust and adaptive dsp systems", Journal of Signal Processing Systems (JSPS), vol. 68, no. 3, pp. 415–431, 2012.

[29] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, A. Raghunathan, "Quality programmable vector processors for approximate computing", IEEE/ACM International Symposium on Microarchitecture (MICRO), 2013.

[30] V. K. Chippa, D. Mohapatra, K. Roy, S. T. Chakradhar, A. Raghunathan, "Scalable Effort Hardware Design", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 22, no. 9, pp. 2004-2016, 2014.

[31] V. K. Chippa, S. Venkataramani, K. Roy, and A. Raghunathan.StoRM: A stochastic recognition and mining processor. In Proc. ISLPED, pages 39–44, 2014.

[32] C. Liu, J. Han, F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery", Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014.

[33] F. Farshchi, M. S. Abrishami, S. M. Fakhraie, "New approximate multiplier for low power digital signal processing", Computer Architecture and Digital Systems (CADS), pp. 25-30, 2013.

[34] D. Mohapatra, V. K. Chippa, A. Raghunathan, K. Roy, "Design of voltage-scalable meta-functions for approximate computing", Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011.

[35] S. G. Ramasubramanian, S. Venkataramani, A. Parandhaman, A. Raghunathan, "Relax-and-retime: A methodology for energy-efficient recovery based design", ACM/IEEE Design Automation Conference (DAC), 2013.

[36] Z. Wang, A. Bovik, H. Sheikh, E. P. Simoncelli, "The SSIM index for image quality assessment", IEEE Transaction on Image Processing, vol, 13, no.4, 2004.

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

[38] D. Palomino, M. Shafique, A. Susin, J. Henkel, "Thermal Optimization using Adaptive Approximate Computing for Video Coding", Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016.

[39] 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.

[40] S. Sarwar, G. Srinivasan, S. Venkataramani, A. Sengupta, A. Raghunathan, K. Roy, "Cross-Layer Approximations for Neuromorphic Computing: From Devices to Circuits and Systems", ACM/IEEE Design Automation Conference (DAC), 2016.

[41] D. M. Mathew, C. Weis, N. Wehn, "Approximate Computing with Partially Unreliable Dynamic Random Access Memory: Approximate DRAM", ACM/IEEE Design Automation Conference (DAC), 2016.

[42] T. Mytkowicz, "Programming Uncertain Things", ACM/IEEE Design Automation Conference (DAC), (Presentation Only) 2016.

[43] J. Henkel, "Approximate Computing: Solving Computing's Inefficiency Problem?", IEEE Design and Test, 2016.

[44] D. Jeong, Y. H. Oh, J. W. Lee and Y. Park, "An eDRAM-Based Approximate Register File for GPUs," in IEEE Design & Test, vol. 33, no. 1, pp. 23-31, Feb. 2016.

[45] A. Yazdanbakhsh, B. Thwaites, H. Esmaeilzadeh, G. Pekhimenko, O. Mutlu and T. C. Mowry, "Mitigating the Memory Bottleneck With Approximate Load Value Prediction," in IEEE Design & Test, vol. 33, no. 1, pp. 32-42, Feb. 2016.

[46] D. S. Khudia, B. Zamirai, M. Samadi and S. Mahlke, "Quality Control for Approximate Accelerators by Error Prediction," in IEEE Design & Test, vol. 33, no. 1, pp. 43-50, Feb. 2016.

[47] B. Li, P. Gu, Y. Wang and H. Yang, "Exploring the Precision Limitation for RRAM-Based Analog Approximate Computing," in IEEE Design & Test, vol. 33, no. 1, pp. 51-58, Feb. 2016.