An automated approximation methodology for arithmetic circuits

An Automated Approximation Methodology for Arithmetic Circuits

Sayandip De, Jos Huisken and Henk Corporaal
Department of Electrical Engineering, Eindhoven University of Technology
Email: {sayandip.de, j.a.huisken, h.corporaal}@tue.nl

***Abstract*—Arithmetic circuits like adders and multipliers are key workforces of many error resilient applications. Prior efforts on approximating these arithmetic circuits mainly focused on manual circuit level functional modifications. These manual approaches need high design time and effort. Due to this only a limited no. of approximate design points can be generated from the original circuit leading to a sparsely occupied pareto front. This work proposes an automated approximation methodology for arithmetic circuits. Proposed method approximates the gate level standard cell library and uses these approximate standard cells to modify the netlist of the original circuit. A heuristic design space exploration methodology is proposed to speed-up the design process. We integrate this methodology with traditional ASIC flow and validate our results using adders and multipliers of different bitwidths. We show that our methodology improves on existing state-of-the-art manual as well as automated design techniques by generating non-dominant pareto-fronts. An application case study (sobel edge detection) is shown using approximate arithmetic circuits generated by our methodology. In case of sobel edge detector, we show upto 50% energy improvements for hardly any quality degradation (PSNR $\geq$ 20dB).**

***Index Terms*—Approximate Computing, Logic Synthesis, Design Space Exploration, Low Power Design**

I. Introduction

Computing in the "Dark Silicon" era marks the need to focus on newer computing paradigms which are realistic and economically viable. *Approximate Computing*–which trades-off quality of desired output for energy, area or performance improvements–is a potential candidate to meet these computing challenges. Prior efforts in approximate computing spans across different abstraction levels in the computing spectrum, namely, algorithm, architecture and circuits [1]. The scope of this work is confined to circuit level approximation strategies. These strategies are generally applicable to all domains.

This work focuses on approximation of *arithmetic circuits* like adders and multipliers as they are the key workforces of many error resilient applications. Notable examples are datapaths of image, signal and video processing accelerators, or multiply-accumulate units of artificial neural networks [2]. Approximate arithmetic circuits, such as, approximate adders and multipliers, can be used to replace the exact calculations in these hardware accelerators.

For designing an approximate accelerator which satisfies a global quality constraint (defined by the application, developer or end-user), following steps are needed: (1) an error resilience analysis to determine local quality constraints for each approximate arithmetic submodule, (2) designing approximate arithmetic submodules which satisfy the local quality constraints, thereby, giving different tradeoff points with diverse energy, area or delay, (3) a design space exploration to find the best combination of approximate arithmetic submodules while optimizing for energy, area or delay.

Point (2) is the main focus of this work. A significant amount of effort has been put into the design of approximate adders [3] [4] and multipliers [5]. Prior attempts to design these circuits followed a manual strategy [6] [3] [4]. *Manual design approaches have limited number of design variants, leading to restricted trade-off possibilities. Also, every time global quality constraint changes, manually designing new approximate design versions require high design time. This motivates the move towards an automated design approach for approximate arithmetic circuits, which can reduce design time and generate various approximate versions satisfying a given quality constraint while optimizing for either energy, area or delay.*

In this work, we propose a design automation methodology for synthesis of approximate arithmetic circuits. This work adopts an automated netlist based replacement and modification approach which is compatible with existing logic synthesis tools. This method approximates the gate-level standard cell library and uses these approximate cells to systematically modify the netlist while adhering to the quality constraints. The key contributions of this work are:

- A methodology for automatic synthesis of approximate arithmetic circuits (section III).
- Systematic design and characterization of a gate-level approximate standard cell library (section IV).
- A graph based replacement strategy of standard cells by their approximate versions along with an intelligent design space exploration methodology to speed-up approximate circuit synthesis (section V).
- Analysis and comparison of generated designs with state-of-the-art techniques. Evaluation of generated designs on a real accelerator (sobel edge detector) (section VI).

The rest of the paper is organized as follows: Section II presents an overview of existing approximation techniques for arithmetic circuits. Proposed methodology is explained in section III. Section IV discusses designing of approximate standard cell library. Our design space exploration methodology is explained in section V. Section VI provides the results for our techniques and finally, section VII concludes the paper with a brief summary of our findings.

II. Related Work

This section gives a brief overview of the existing approaches for approximation of adders and multipliers. Prior ef-

forts on functional approximation of arithmetic circuits can be categorized into two types: manual or automated approaches.

Most manual approaches introduce modifications to carefully pre-selected parts of the arithmetic circuit. In case of adders, it is possible to approximate the elementary 1-bit adders (IMPACT [7]) or introduce segmented adders (ETA [8]) or modify the carry propagation chain (ACA [9]). Quality configurable adders have been proposed in (GDA [3], GeAr [4]) which modifies the error using several configuration bits. These configuration bits can connect or disconnect some preselected parts of the circuit to tradeoff quality versus power or delay. In case of multipliers, approximation is done in generating partial products, in the partial product tree or the compressors in the partial product tree [10]. A major drawback of these manual approaches is high design time and effort. Only a few approximate versions can be created from the original circuit, leading to a sparsely occupied pareto front.

In contrast to manual approaches, there is limited work in automated algorithmic approaches for designing approximate arithmetic circuits. A depth-first search based design space exploration methodology is proposed in [5] for approximation of multipliers. Various design points, each exhibiting distinct power, area and output quality, are generated by using approximate elementary 1-bit adders and 2x2 multipliers. A multi-objective cartesian genetic programming (CGP)-based approximation process for artihmetic circuits is described in (EvoAprrox8b [11]). Automatic pruning technique have been proposed as a logic level method to select and prune parts of a digital circuit [12] [13].

Our proposed methodology differs from the above techniques in the following aspects: (1) Existing automated techniques try to modify the behavioural RTL description of the design by using simplified elementary adders & multipliers [5] or by using evolutionary algorithms [11]. So, they operate on pre-synthesis stage of ASIC design flow. Our proposed methodology is orthogonal to these approaches. It operates on post-synthesis netlist which makes it more generic. While our primary motivation is to automatically approximate arithmetic circuits, it can be applied to any combinational datapath (not the focus of this paper). (2) Most existing techniques adopt a *bottom-up* approach to incorporating approximate arithmetic modules in complex accelerator datapaths. First, they generate different approximate versions of arithmetic circuits without prior application/accelerator knowledge. A pareto front trading-off quality and power/delay/area is obtained. Then a design space exploration is used to figure out the best combination of the non-dominated points to be used in the approximate accelerator. Contrarily, in *top-down* approach, local quality constraints are obtained by error analysis of the approximate accelerator [point(1), section I]. Our methodology can be used in a *top-down* approach to generate approximate arithmetic modules given a local constraint. This saves design time as generation of densely populated pareto front is expensive.

III. Methodology Overview

In a traditional ASIC design flow, a register transfer level (RTL) code is synthesized to a netlist using a standard cell library (see Fig. 1). We integrate our methodology with this traditional ASIC flow, but instead of using accurate standard cells, proposed methodology uses approximate standard cells for automatic synthesis of arithmetic circuits (adders and multipliers).

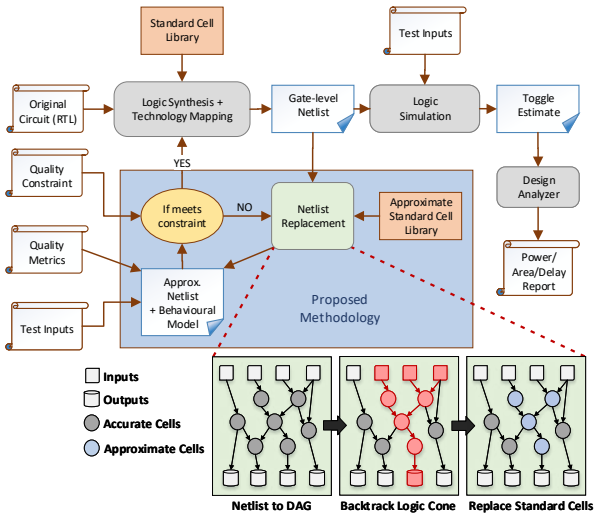

Fig. 1. Proposed methodology integrated with standard design flows.

At first, the gate-level netlist of the accurate circuit is converted into a directed acyclic graph (DAG), where the nodes represent standard cell instances and the edges represent wires. Arithmetic circuits follow a notion of bit significance, i.e., each bit has two times higher significance than the previous bit when moving from LSB to MSB. So, each primary output bit is considered as a configurable knob for controlled quality modulation. For a particular primary output bit, the logic cone is obtained by backward dependency search using reverse graph traversal. The cells in this logic cone are replaced with approximate cells from the library having the same type to give an approximate netlist. The above mentioned steps are shown in Fig. 1. *Note: Our methodology can be altered to optimize for energy, area or delay. In the rest of the discussion, we will optimize designs for energy.*

IV. Designing Approximate Standard Cell Library

The first step in our methodology is building an approximate library of standard cells. *Functionally inexact versions of standard cells are designed by altering the truth table of the cells.* Following 3 design rules are considered for building the approximate standard cell library (as shown in Fig. 2):

I. Design Space Reduction: It is not feasible to approximate all cells in the existing standard cell library. Therefore, there is a need for design-space reduction. To achieve this, energy breakdown of design under test is obtained. Only cell types that contribute more than 10% to the total energy consumption of the circuit are selected. For example, for a 8-bit multiplier, only fulladder (FA32) and compressor (CMPE53) cells are selected for approximation after design space reduction (see Fig. 2).

II. Truth Table Optimization: Multiple functionally inexact version of each selected cell type are generated manually

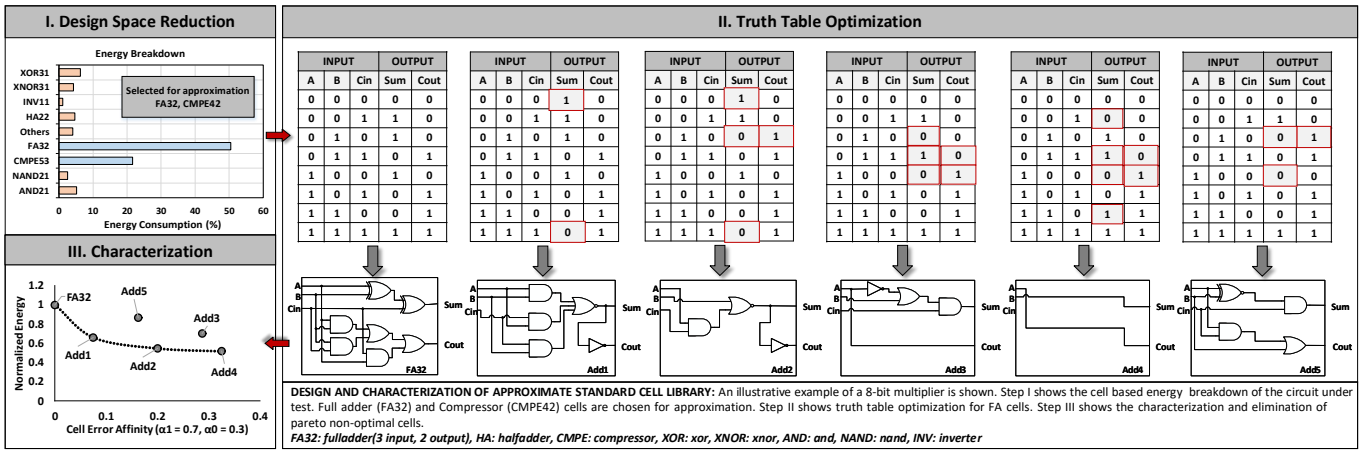

Fig. 2. Steps in Building Approximate Standard Cell Library

by altering the truth table of the cells. Fig. 2 shows an illustrative example where 1-bit fulladder cells are approximated based on designs reported in [7] [2].

III. Characterization: All approximate cell designs generated in step. II are characterized by trading-off accuracy & energy efficiency. Designs that are not pareto-optimal are filtered out (see Fig. 2). For characterization of approximate designs, we formulate a cost function, *Cell Error Affinity* (CEA), which is a weighted sum of error probabilities (E_{prob}) at the output bits of the standard cells. The formulation of E_{prob} for ith output bit is given in equation 1, where M is n-bit accurate output, M' is n-bit approximate output and $input\_count$ is the input bitwidth of the circuit. Equation 2 gives Cell Error Affinity (CEA) which is the weighted sum of E_{prob} for each output bit of the cell. $\alpha_1, \alpha_2, .., \alpha_n$ are the respective weights satisfying $\sum_{i=1}^{n} \alpha_i = 1$.

$$E_{prob}(i) = \sum_{1}^{2^{input\_count}} {}^{1}/_{2^{input\_count}} \; if \; M_i' \neq M_i \tag{1}$$

$$CEA = \sum_{i=1}^{n} \alpha_i \times E_{prob}(i) \tag{2}$$

The weight values are tuned manually by analysing the impact of each erroneous output bit of the standard cell on the global circuit output. In case of multipliers and adders, it is clear that errors in MSB of the standard cells are more critical than those in the LSB. Therefore, pareto-curve shown in Fig. 2 assumes a higher weight for the MSB & lower for the LSB.

V. Design Space Exploration Methodology

Given an accurate netlist and an approximate standard cell library, there is a need to find out proper combination of approximate cells to obtain different inexact versions of the same circuit adhering to certain accuracy. This is an intractable optimizaton problem leading to design space explosion. So, our methodology employs a heuristic exploration methodology rather than exhaustively trying out all possible combinations, as shown in algorithm 1. The size of design space covered in exhaustive search can be calculated by equation 3, where D_P represents the total number of design points, L represents total no. of cell types, $C_1, C_2, ..C_L$ represents no. of approximate versions of each cell type and $N_1, N_2, ..N_L$ represents no. of instances of each cell type.

$$D_P = C_1^{N_1} \times C_2^{N_2} \times ...C_L^{N_L} \tag{3}$$

To tackle design space explosion problem (as evident from equation 3), an iterative algorithm is proposed that makes the locally optimal choices in successive iterations, with selective back stepping to avoid getting trapped in local optimum, while searching for the global optimum.

The inputs to the algorithm are: RTL description of an arithmetic circuit (Ckt_{orig}), a collection of pre-characterized approximate cells ($Approx_{LIB}$), a quality metric ($Q_{metrics}$), described in section VI and a quality constraint ($Q_{constraint}$) specific to the target application. The algorithm begins by sorting out the different approximate variants of the same cell type in ascending order of their energy cost (lines 4-6). It is followed by two distinct phases of operation. In the first phase, greedy search for a locally optimal solution is performed by only considering the cells with highest approximation (therefore, least energy cost) to get an initial seed (lines 8-18). In the second phase, further fine tuning is done on the initial seed to get a better optimized solution (lines 19-39).

In the initial phase, cell instances in the logic cone on which a particular output bit is dependent, are extracted (section III). This is done by performing reverse graph traversal on directed acyclic graph (DAG). These instances are replaced with least energy consuming approximate cells. This process is iteratively repeated for all output bits, starting with the LSB and then incrementing until the design breaks the quality specifications. This gives the *initial seed*. The iteration index (output bit index) where the design breaks is recorded. In the fine tuning phase, the iterator is rolled back to select a less promising solution (line 26). Cell instances corresponding to the iteration index are replaced with approximate ones chosen from the approximate library list (line 5) sorted in ascending order of energy cost (lines 27-29). This process is repeated for every increment of the iterator until the quality check fails (lines 30-34) which gives a new seed to be used in subsequent iterations. However, in case of no improvements,a previous locally optimal design corresponding to an old seed

is used (lines 35-37). The back step and new seed generation is continued until all the approximate cell versions are iterated, finally resulting in a quality controlled approximate design.

VI. Experimental Results

For the evaluation of the proposed methodology, we implemented approximate versions of various arithmetic circuits as shown in Table I. We also evaluate effectiveness of our generated solutions by systematically using them in a sobel edge-detection application.

Qualtiy Metric Formulation: In this work, three different quality metrics are considered: Maximum Error Distance (MaxED), Mean Error Distance (MED) and Peak Signal to Noise Ratio (PSNR). Equations 4-5 & 6 give the formulation of these quality metrics where M is accurate result, M' is approximate result and S is the sample size.

$$MaxED = max\{|M' - M| \ \forall \ val \ \in T\} \tag{4}$$

$$MED = \frac{1}{|T|}\sum_{n=1}^{|T|}|M' - M| \tag{5}$$

$$MSE = \frac{1}{|S|}\sum_{n=1}^{|S|}|M' - M|^2, PSNR = 20\log_{10}\frac{max\{T\}}{\sqrt{MSE}} \tag{6}$$

$$where\ T = \{val | val \in \mathbb{Z}^+, |val| < 2^{input\_count}\}$$

Experimental Setup: The test circuits were synthesized using Cadence Encounter RTL Compiler at nominal operating conditions, mapped to a 40nm TSMC technology library and functionally verified using Cadence Incisive Enterprise Simulator. A python based behavioural model is generated for each intermediate approximate netlist which is used for functional verification.

Results: Fig. 3 shows the accuracy vs energy/area/delay trade-off for a *8-bit signed multiplier* on a pareto frontier. Energy, area & delay scales on the Y axis are normalized with respect to the tool synthesized versions of the accurate design, at relaxed frequency constraint (baseline). The experiments are performed for different target values of MaxED & MED chosen from the sets {250 – 2000} and {5 – 400} respectively. Compared to baseline, designs obtained using our method achieve energy savings of upto 80% for less than 5% degradation in output quality (MaxED $\leq$ 2000)(see Fig. 3, row1,col1). Comparison with literature shows that the designs reported in [5][1] are pareto-domiated by our designs. This is expected as approximate multipliers in [5] were composed of smaller ones and this composition procedure introduces some overhead. It is also observed that our method generates a better densely-spread pareto front compared to bitwidth truncation and Evoapprox [11]. Proposed method also reduces the area-under-the-pareto-front (AUP) for tradeoffs between energy and quality. Average AUP-energy reductions of 11%, 75% & 10% are obtained compared to bitwidth truncation, Rehman et. al [5] and Evoapprox [11] respectively.

[1]Only unsigned multipliers are reported in this work. We make them signed using two's complemented I/Os. Results include the cost of these two's complement modules. For fair comparison, we also report results of the unsigned multipliers without two's complemented I/Os.

Algorithm 1: Optimal Design Search Algorithm

Input : Ckt_{orig} : Exact circuit, $Approx_{LIB}$: List of multiple variants of approximate cell types, Q_{metric} : Qualtiy metric suitable for target application, $Q_{constraint}$: Application specific quality constraint
Output: Ckt_{approx} : Approximate Circuit

1 **Begin**
2 Initialize : $netlist_{orig}$, $netlist_{approx}$ = synthesize (Ckt_{orig})
3 $DAG \leftarrow netlist_{orig}$
4 **for** each different *cell_type* in $Approx_{LIB}$ **do**
5 $list_i$ = sort$_{\text{energy}}$ $(cell\_type, cell\_variants, ascending)$;
6 **end**
7 // **Assuming** N = *Total count of distinct cell_types.*
8 // **Find initial seed**
9 **for** each *outputbit* from *lsb* to *msb* in $netlist_{orig}$ **do**
10 $cell\_names$ = extract_cell_instances $(DAG, outputbit)$;
11 $netlist_{approx}$ = replace $(netlist_{approx}, cell\_names,$
12 $list_1[0], list_2[0], \ .. \ , list_N[0])$;
13 Q_{test} = Q_{metric} $(netlist_{approx})$;
14 **if** Q_{test} *donot meet* $Q_{constraint}$ **then**
15 initial_seed = $previous\_netlist_{approx}$;
16 set **seed** = $initial\_seed$, $outputbit = previous\_outputbit$;
17 **break**
18 **end**
19 // **Fine tuning**
20 **for** each $cell\_list_1$ in $list_1$ **do**
21 **for** each $cell\_list_2$ in $list_2$ **do**
22 . .
23 **for** each $cell\_list_N$ in $list_N$ **do**
24 set **old_seed** = $seed$
25 **for** *outputbit* to *msb* in $seed$ **do**
26 $new\_outputbit$ = backstep $(outputbit)$;
27 $cell\_names$ = extract_cell_instances$(DAG, new_outputbit)$;
28 $seed$ = replace $(seed, cell\_names,$
29 $cell\_list_1, cell\_list_2, \ .. \ , cell\_list_N)$;
30 Q_{test} = Q_{metric} $(seed)$;
31 **if** Q_{test} *donot meet* $Q_{constraint}$ **then**
32 set **seed** = $previous\_seed$, $outputbit = previous\_outputbit$;
33 **break**
34 **end**
35 **if** $seed$ donot improve upon old_seed **then**
36 $seed = old\_seed$;
37 **end**
38 **end**
39 **end**
40 Ckt_{approx} = $seed$;
41 **return** Ckt_{approx};
42 **End**

Fig. 4 shows the accuracy vs energy/area/delay trade-off for *8-bit, 16-bit and 32-bit adders* on a pareto frontier. Energy, area & delay scales on the Y axis are normalized with respect to the tool synthesized RCA adder, at relaxed frequency constraint (baseline). Manual[2]designs approaches like ACA

[2]Some open source adder libraries like ACA, GeAr etc., are optimized for delay, not power. For fair comparison, we consider energy as a metric. We also report AUP for all metrics: energy, area & delay.

TABLE I
BENCHMARK CIRCUITS USED IN EXPERIMENTS

Benchmark Name	Bit Width	Input Dataset	Cell:Cell Type [1]	Approx Cell Types [2]	I:O	Area (GE)	Energy (pJ)	Delay (ps)	Quality Metric
Array multiplier (signed)	8	65536 8-bit integers (dist: uniform, order: random)	155:18	FA32(4), CMPE53(6)	16/16	1242	0.32	2507	MaxED, MED
Ripple Carry adder (RCA)	8	65536 8-bit integers (dist: uniform, order: random)	8:2	FA32(4), HA22(5)	16/9	142	0.03	1558	MaxED, MED
	16	65536 16-bit integers (dist: uniform, order: random)	16:2	FA32(4), HA22(5)	32/17	286	0.12	2646	MaxED, MED
	32	65536 32-bit integers (dist: uniform, order: random)	35:5	FA32(4), HA22(5)	64/33	603	0.42	4000	MaxED, MED
Brent-Kung adder	32	65536 32-bit integers (dist: uniform, order: random)	177:20	FA32(4), HA22(5), XNOR2(3), NOR2(3)	64/33	1412	0.18	1412	MaxED, MED
Kogge-Stone adder	32	65536 32-bit integers (dist: uniform, order: random)	321:25	HA22(5), XOR2(3), AOI21(3), NAND2(3)	64/33	2374	0.21	1108	MaxED, MED

[1] total no. of cells: total no. of different cell instances
[2] cell instances chosen for approximation (no. of approximate variants of each cell type)
[2] FA32: fulladder(3 input, 2 output), HA: halfadder, CMPE: compressor, XOR: xor, XNOR: xnor, NOR: nor, NAND: nand, AOI: and-or-invert

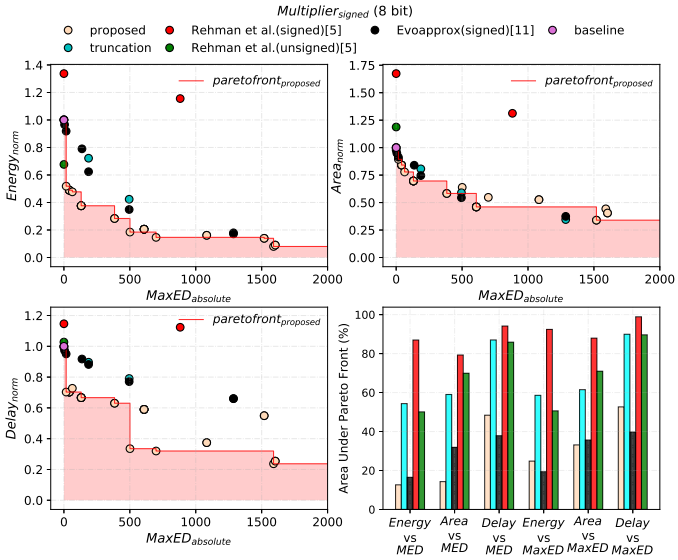

Fig. 3. Pareto fronts showing tradeoffs between MaxED and Energy, Area, Delay for a 8-bit multiplier (signed). The area under the pareto fronts (shaded in red) is plotted for all the plots. Lower area under the curve is better.

[9], ETA [8], GDA [3] and GeAR [4] have limited no. of design points leading to a sparsely populated pareto-front (see Fig. 4). Contrarily, our method provides a dense pareto front which validates the requirement of automated approximation strategy for arithmetic circuits. For 8-bit adders, upto 62% energy savings are obtained compared to baseline for less than 5% quality degradation (MaxED $\leq$ 25). Average AUP-energy reductions of 4%, 42%, 33% & 30% are obtained w.r.t bitwidth truncation, ACA [9], GDA [3] and GeAR [4] respectively. For 8-bit adders, Evoapprox [11] generates a few design points which dominates our results (See Fig. 4 row1, col1). When we analyze some of these cases, we observe that our heuristic misses these design points as it starts by finding an intial seed using the cells least energy cost, followed by fine tuning. Instead, if we start directly with fine tuning, we cover these cases. However, this leads to longer runtimes, which is a trade-off designer needs to consider. For 16-bit and 32-bit adders,

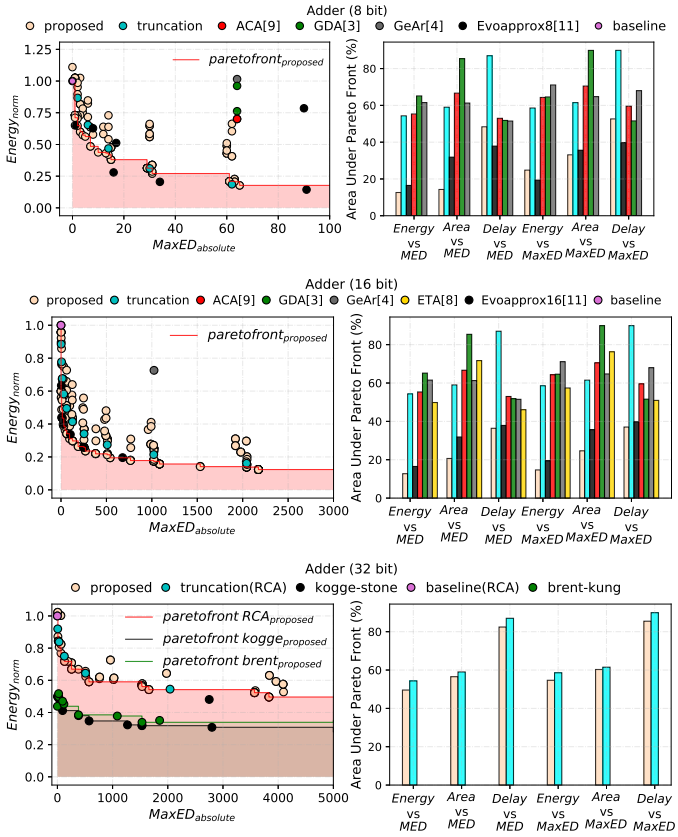

Fig. 4. Pareto fronts showing tradeoffs between MaxED and Energy for 8-bit, 16-bit and 32-bit adders. The area under the pareto fronts (shaded in red) is plotted for all the plots. Lower area under the curve is better.

our heuristic methodology out performs existing approaches in the literature, including, bitwidth truncation & Evoapprox [11]. Fig. 4 (row3, col1) shows the energy improvements obtained for high speed adder designs like brent-kung & kogge-stone adders. This validates the applicability of our methodology for both high speed (brent-kung & kogge-stone) as well as low power (RCA) adders.

Heuristic Optimality Analysis: Fig. 5 shows a comparative analysis between exhaustive search and heuristic search

(Section V). The experiment took 7.5 hours for exhaustive search & 70 seconds per optimal design point for heuristic search on a Intel i7 (3.4 GHz) processor and 32GB RAM. Total runtime is mostly dominated by the ASIC design flow. A speedup of 10× is obtained over exhaustive search while giving near optimal solutions (Fig. 5).

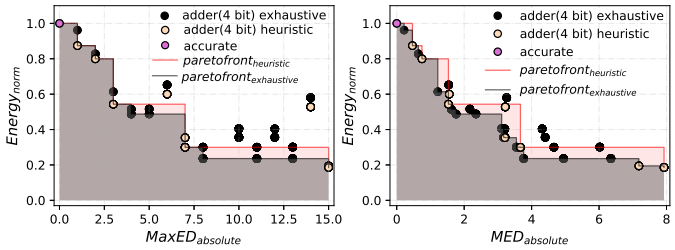

Fig. 5. Heuristic versus exhaustive search (4-bit Adder)

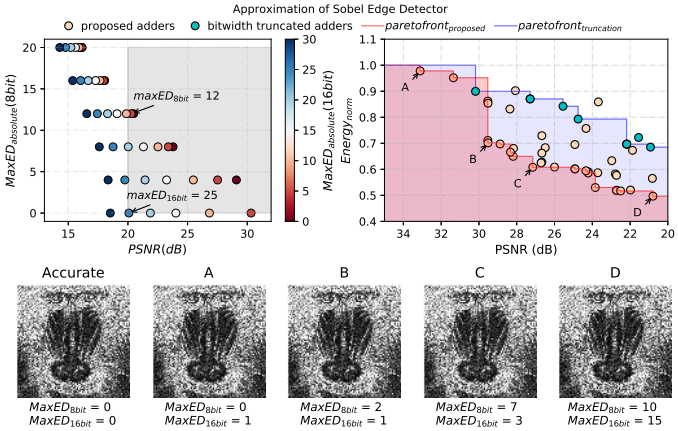

Fig. 6. Approximation of sobel edge detector

Application Case Study: To test the effectiveness of our method, a case study on sobel edge detection is performed. Considered accelerator architecture consists of six 8-bit adds, two 16-bit adds and one shift, each for vertical as well as horizontal edge detection. We follow the *top-down* approach (see section I, II) for approximate accelerator design. Firstly, an error resilience analysis [2] is performed on a python-based behavioural model of sobel edge detector by emulating use of approximate adders. A design space exploration is performed using emulated 8-bit and 16-bit approximate adders with different error bounds and quality impact on final sobel output is analysed. As shown in Fig. 6 (row1, col1), each point gives a particular combination of emulated 8-bit and 16-bit approximate adders with certain error bounds. Y-axis gives the error bounds on the emulated 8-bit adders while the colorbar gives the error bounds on the emulated 16-bit adders. For each combination, the final application PSNR is given on X-axis. So, for a given global quality constraint of $PSNR \geq 20dB$, local quality constraints are obtained for 8-bit (MaxED $\leq$ 12) and 16-bit (MaxED $\leq$ 25) adders. Then, using our methodology, 8-bit & 16-bit approximate adders are generated to meet these local constraints. Finally, a design space exploration is performed using the non-dominated pareto points, to obtain the best trade-off between energy and PSNR (Fig. 6, row1, col2). For hardly any quality degradation (Fig. 6, row2, col5), upto 50% energy improvements are obtained (point D, Fig. 6, row1, col2). Fig. 6, row1, col1, also shows that our designs significantly outperforms bitwidth truncated designs. Detailed analysis shows that our designs perform better as truncated designs make unidirectional errors (either + or -). However, proposed designs make bidirectional (+/-) errors at the output which leads to error compensation.

VII. Conclusion and Future Work

This work proposes an automated approximation methodology for arithmetic circuits. Proposed method approximates the gate level standard cell library and uses these approximate standard cells to modify the netlist of the original circuit. A heuristic design space exploration methodology is proposed to speed-up the design process. We integrate proposed methodology with traditional ASIC flow and validate our results using adders and multipliers of different bitwidths. Upto 80% improvements in energy efficiency is obtained for less that 5% degradation in quality. A comparative analysis with state-of-the-art techniques as well as bitwidth truncation is performed. We show that our methodology improves on existing techniques by generating dense and non-dominant pareto-fronts. For a real application(sobel edge detection), approximate arithmetic circuits generated by our methodology achieve upto 50% energy improvements for hardly any quality degradation (PSNR $\geq$ 20dB). Future iterations aim to extend support for sequential circuits as well as automated inexact standard cell generation.

Acknowledgment

This research has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under grant agreement no 674875 (oCPS).

References

[1] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, Mar. 2016.
[2] S. De et al., "Designing energy efficient approximate multipliers for neural acceleration," in *DSD*, 2018.
[3] R. Ye et al., "On reconfiguration-oriented approximate adder design and its application," in *ICCAD*, 2013.
[4] M. Shafique et al., "A low latency generic accuracy configurable adder," in *DAC*, 2015.
[5] S. Rehman et al., "Architectural-space exploration of approximate multipliers," in *ICCAD*, 2016.
[6] R. Zendegani et al., "RoBA Multiplier: A Rounding-Based Approximate Multiplier for High-Speed yet Energy-Efficient Digital Signal Processing," *IEEE TVLSI*, 2017.
[7] V. Gupta et al., "IMPACT: IMPrecise adders for low-power approximate computing," *ISLPED*, 2011.
[8] N. Zhu et al., "Enhanced low-power high-speed adder for error-tolerant application," in *ISOCC*, Nov 2010.
[9] A. K. Verma at al., "Variable latency speculative addition: A new paradigm for arithmetic circuit design," in *DATE*, 2008.
[10] H. Jiang et al., "A comparative evaluation of approximate multipliers," in *NANOARCH*, 2016.
[11] V. Mrazek et al., "Evoapproxsb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *DATE*, 2017.
[12] J. Broc et al., "A fast pruning technique for low-power inexact circuit design," in *LASCAS*, 2015.
[13] J. Schlachter et al., "Design and Applications of Approximate Circuits by Gate-Level Pruning," *IEEE TVLSI*, 2017.