Plasticine: A Cross-layer Approximation Methodology for Multi-kernel Applications through Minimally Biased, High-throughput, and Energy-efficient SIMD Soft Multiplier-divider

ZAHRA EBRAHIMI, DENNIS KLAR, MOHAMMAD AASIM EKHTIYAR, and AKASH KUMAR, Technische Universität Dresden, Germany

The rapid evolution of error-resilient programs intertwined with their quest for high throughput has motivated the use of Single Instruction, Multiple Data (SIMD) components in Field-Programmable Gate Arrays (FPGAs). Particularly, to exploit the error-resiliency of such applications, *Cross-layer* approximation paradigm has recently gained traction, the ultimate goal of which is to efficiently exploit approximation potentials across layers of abstraction. From circuit- to application-level, valuable studies have proposed various approximation techniques, albeit linked to four drawbacks: First, most of approximate multipliers and dividers operate only in SISD mode. Second, imprecise units are often substituted, merely in a single kernel of a multi-kernel application, with an *end-to-end* analysis in Quality of Results (QoR) and not in the gained performance. Third, state-of-the-art (SoA) strategies neglect the fact that each kernel contributes differently to the end-to-end QoR and performance metrics. Therefore, they lack in adopting a generic methodology for adjusting the approximation knobs to maximize performance gains for a user-defined quality constraint. Finally, multi-level techniques lack in being efficiently supported, from application-, to architecture-, to circuit-level, in a cohesive cross-layer hierarchy.

In this article, we propose *Plasticine*, a cross-layer methodology for multi-kernel applications, which addresses the aforementioned challenges by efficiently utilizing the synergistic effects of a chain of techniques across layers of abstraction. To this end, we propose an application sensitivity analysis and a heuristic that tailor the precision at constituent kernels of the application by finding the most tolerable degree of approximations for each of consecutive kernels, while also satisfying the ultimate user-defined QoR. The chain of approximations is also effectively enabled in a cross-layer hierarchy, from application- to architecture- to circuit-level, through the plasticity of SIMD multiplier-dividers, each supporting dynamic precision variability along with hybrid functionality. The end-to-end evaluations of Plasticine on three multi-kernel applications employed in bio-signal processing, image processing, and moving object tracking for Unmanned Air Vehicles (UAV) demonstrate 41%–64%, 39%–62%, and 70%–86% improvements in area, latency, and Area-Delay-Product (ADP), respectively, over 32-bit fixed precision, with negligible loss in QoR. To springboard future research in reconfigurable and approximate computing communities, our implementations will be available and open-sourced at https://cfaed.tu-dresden.de/pd-downloads.

This research is co-funded by the projects *ReAp: Runtime Reconfigurable Approximate Architecture* (Number 380524764), funded by the German research foundation *Deutsche Forschungsgemeinschaft (DFG)* and *Re-learning: Self-learning and flexible electronics through inherent component reconfiguration* (Number 100382146), funded by the *European Social Fund (ESF)*.
Authors' address: Z. Ebrahimi, D. Klar, M. A. Ekhtiyar, and A. Kumar, Technische Universität Dresden, Dresden, Saxony, Germany; emails: zahra.ebrahimi_mamaghani@tu-dresden.de, dennis.klar@mailbox.tu-dresden.de, mohammad_aasim.ekhtiyar@mailbox.tu-dresden.de, akash.kumar@tu-dresden.de.

1084-4309/2021/11-ART16 $15.00
https://doi.org/10.1145/3486616

CCS Concepts: • **Cross-Layer Approximation → Field-Programmable Gate Arrays;**

Additional Key Words and Phrases: Single instruction multiple data, hybrid multiplier-divider, bio-signal processing, image processing, unmanned air vehicles, Mitchell's algorithm, high-throughput, energy-efficiency

ACM Reference format:
Zahra Ebrahimi, Dennis Klar, Mohammad Aasim Ekhtiyar, and Akash Kumar. 2021. Plasticine: A Cross-layer Approximation Methodology for Multi-kernel Applications through Minimally Biased, High-throughput, and Energy-efficient SIMD Soft Multiplier-divider. *ACM Trans. Des. Autom. Electron. Syst.* 27, 2, Article 16 (November 2021), 33 pages.
https://doi.org/10.1145/3486616

1 INTRODUCTION

The ever-rising quest for real-time processing at the edge is a common requirement in wide spectrum of applications from bio-signal to various cutting-edge image processing programs, e.g., self/object tracking in **Unmanned Aerial Vehicles (UAVs)**. From proliferating use-cases of these edge nodes are drones that are coming at the forefront of diverse domains, including search-and-rescue missions, surveillance and crowd management, agricultural operations, and entertainment. Wearable 24/7 health-monitoring gadgets are also from other widely used nodes, especially when considering 47% of cardiac diseases—the major cause of death globally—occur outside the hospitals [1, 2]. Although **Application-Specific Integrated Circuits (ASICs)** are highly power-efficient platforms for implementation of the aforementioned programs, hardware flexibility is also of paramount concern, which is required to be addressed. For example, the hardware of health gadgets should be flexible to adapt with different patients' physiological traits and the heart's changing activity. Moreover, high-throughput is another requirement, as such *parallelizable* applications are constantly fed with bulk of data. Finally, also by considering the application upgradability that usually outpaces hardware updates, off-the-shelf **Field-Programmable Gate Arrays (FPGAs)** are left as the commercially viable choices that also enjoy high throughput, rapid prototyping, and post-fabrication datapath versatility [3–5].

FPGAs, rewarded by a high degree of parallelism, encompass hard-wired DSP blocks to accelerate multiplication that is the atomic function in image or bio-signal processing workloads. In spite of their advantages, hosting DSP blocks falls short on fulfilling design requirements in a variety of programs: First, their limited number is insufficient for multiplication-intensive or concurrent programs. Second, their fixed locations in FPGAs poses routing overhead and often results in degraded performance of some circuits [6–8]. Finally, they are unable to be efficiently utilized for multiplication with small precision ($<18 \times 18$ bit) [9, 10] and, therefore, cannot render the expected energy gain by precision scaling. Therefore, designers have been forced to also exploit soft **Intellectual Property (IP)** versions of multipliers and dividers, provided by major FPGA vendors such as Xilinx and Intel [11, 12]. Utilizing soft IPs instead of DSPs, for low bit-width operations, has also been recommended by industry and academia [13, 14]. However, the long latency and high resource requirement of LUT-based IPs might still hinder their deployment, e.g., in wearable gadgets and aerial platforms with stringent energy constraints. Moreover, the quest for high throughput is still left unaddressed.

To alleviate resource footprint in the above-mentioned error-resilient programs, various approximation techniques have been emerged at both circuit- (imprecise Add/Mul/Div) and application-level (e.g., precision scaling). However, two main challenges are attributed to **circuit-level** techniques: ① Most of existing approximate multipliers and dividers operate only in SISD mode while also customized for ASIC platforms. In fact, due to the different intrinsic

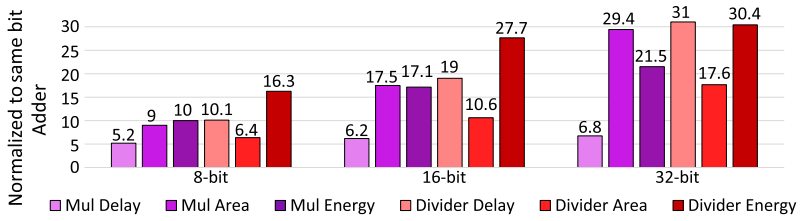


Fig. 1. Comparing area, delay, and energy of 8-, 16-, and 32-bit addition, multiplication, and division operations, when implemented in Virtex-7 FPGA, via Look-up Table (LUT)-based accurate IPs.

architectural specifications, approximation approaches assessed in ASIC platforms have not yielded comparable gains when directly synthesized and ported to FPGAs [7]. ② While most efforts were concentrated on multiplication, studies such as References [15, 16] and our analysis, shown in Figure 1, exhibit the longer latency and higher energy of division compared to the multiplication, which can confine the application speed. Therefore, the approximation of division has recently become more pronounced as, although less frequent, this operation is inevitable in bio-signal processing (heartbeat detection) as well as image processing and vision applications (K-means for unsupervised clustering, JPEG compression, and Harris corner detection). Nevertheless, among the imprecise dividers in the literature, none was customized for either LUT-based or a **Single Instruction, Multiple Data (SIMD)** design.

Literature studies that also consider *application-level* techniques are linked with two drawbacks. First, they have applied approximations mostly, on a single kernel of a multi-kernel application, e.g., in **Discrete Cosine Transform (DCT)** stage of JPEG compression [16, 17]. Their performance gain is also oftentimes appraised and reported for that single kernel and not in an end-to-end implementation of the complete application. Second, kernel precision in most of such approaches is fixed to 16-bits, and the effect of *varying* precision on resource-accuracy tradeoff has not been analyzed. In fact, sticking to fixed-precision strategy ignores the fact that kernels' significance order might differ in the *gained performance* and *lost* ***Quality of Result (QoR)***.[1]

The paradigm of **Cross-Layer** approximation has emerged [19], the ultimate goal of which is not only to exploit various approximation potentials across layers of abstraction, but also to efficiently unlock the synergistic effects of such chain of techniques from application- to architecture- to circuit-layer, in a cohesive cross-layer hierarchy. However, the valuable studies (e.g., Reference [20] and its references) that have tried to enable cross-layer approximation are linked with two main challenges. First, they mostly have leveraged circuit-level inexact Add/Mul/Div in tandem with application-level precision scaling without exploiting SIMD potentials (at architecture-level), which can also elegantly provide support for precision scaling. Second, they are hitherto restricted to neural networks, and studies in other domains have not explored the effect of *changing* precision *along with* utilizing inexact units.

Thus, to harvest architecture-level potentials, the concept of SIMD has been tailored by Intel [21, 22] and Xilinx [23] to provide support for double multiplication within their FPGA DSPs. Such SIMD designs not only harness data-level parallelism capability, but also bridge the gap between circuit- and application-level approximations by supporting Dynamically Reconfigurable Approximation [24] through allowing runtime precision-variability *in a single unit*. However, such SIMD units are customized for ASIC implementation and not analyzed for a cross-layer approach. In short, despite the great efforts of aforementioned studies, they are still faced with two challenges:

[1]Required precision varies not only between programs, but also among kernels of a *single* program, e.g., layers of NN [18].

First, they lack a generic methodology for *multi-kernel applications* to adjust approximation knobs in such a way that can maximize performance gains for a user-defined quality. Second, they have overlooked the SIMD potentials at architecture-level.

To surmount foregoing circuit-level challenges, we have designed the LUT-based hybrid multiplier-divider SIMDive [15], which not only enables the plasticity to switch between two functionalities on-the-fly without the need for reconfiguration, but also can operate in SIMD mode. SIMDive is an integer multiplier-divider, designed based on Mitchell's algorithms [25], which transforms the multiplication (division) operation to addition (subtraction) in the logarithmic representation (Section 3.1 details this algorithm). Transforming the 2D structure of array multiplier and divider to 1D adder and subtractor in the logarithmic representation significantly reduces design complexity, especially bridges the long latency/energy of an accurate divider, nearly to its same-size multiplier. This translation also fits FPGAs and renders substantial gains, as they are already equipped with fast carry chains hardened to accelerate addition. SIMDive is specialized toward accelerating Mul/Div computations with a SIMD approach by probing the presence of Leading One in parallel, in each 4-bit segments of the inputs. Moreover, the implementations of our novel error-reduction schemes—which are also independent from Mul/Div size—are *also* customized for LUT-based platforms. In fact, our light-weight error-reduction schemes use only one LUT for generating one bit in 64 error-reduction terms (see Section 4). Afterwards, the *addition* of these error-coefficients to the original Mitchell's multiplier and divider are performed concurrently and within the same resources, used for the baseline designs [26]. Therefore, the latency overhead in the overall critical path is minimized to only calculation of the error-coefficients, as their addition via ternary adder uses nearly similar resources when using a binary adder (considering the fixed latency of LUT primitives). This is while in Reference [27], MBM [17], and INZeD [16], a non-trivial circuitry is needed for addition of error-reduction terms to the original Mitchell's circuits.

In this article, we propose *Plasticine*, which sets out to enable a cross-layer approximation methodology for multi-kernel applications. Through conducting a novel application-level sensitivity analysis, we highlight four observations as key guidelines for our methodology: (1) It is unnecessary to dedicate the high-precision of 32-bit uniformly to all kernels. (2) Kernels significance order in *lost QoR* can differ from its order in the *gained performance* when approximated by the same techniques. (3) Such multi-kernel applications show higher sensitivity to approximation of addition operation. (4) Low error-bias has also a high impact on the final QoR: Errors with different signs can nullify each other, provided that kernels have aggregation-based structure (i.e., mostly Add/Mul). This last observation also has been hinted in recent *NN-focused works* [24, 28, 29], yet analyzed herein for different domains of applications. Based on such observations, we have proposed the $\frac{\Delta\text{Performance}}{\Delta\text{QoR}}$ as the saliency metric that appropriately reflects intensity of gained performance over a possible accuracy loss when applying a chain of approximations. In fact, we have proposed our cross-layer approximation strategy based on using this deciding metric in a greedy heuristic. Our methodology tailors the precision at successive kernels by finding the most tolerable degree of approximations for each kernel, while also meeting the user-defined accuracy in the ultimate QoR. In short, we make the following novel and key technical research contributions:

- **Minimally biased multiplier & divider with tunable accuracy**. Our SISD designs, with 99.2% accuracy, achieve up to 80% less energy and >7× higher speed than accurate Vivado IPs.
- **First integrated approximate multiplier-divider**. Our hybrid Mul/Div design can switch between two functionalities on-the-fly, without the need of reconfiguration, while still outperforming a single accurate multiplier in terms of area, latency, and energy.

- **FPGA-customized SIMD architectures for proposed Mul/Div**. To adapt for SIMD architectures, we propose an LUT-oriented 4-bit **Leading-One Detector (LOD)**, using only two 6-LUTs. Overall, 16- and 32-bit SIMD units are still smaller than the same-size accurate SISD multiplier.
- **Application-level cross-layer sensitivity analysis**. It determines the *relation* between gained performance and lost QoR, used to adjust the approximation knobs in our proposed methodology.
- **Cross-layer methodology for multi-kernel programs**. Pareto/near-Pareto *mixed-precision* configurations generated by our heuristic not only enable various performance-accuracy tradeoffs, but also render higher gains over uniform-precision counterpart, in different accuracy-levels.
- **End-to-end performance-QoR evaluation on three application domains**. The efficacy of the proposed cross-layer approximation is demonstrated on ubiquitously used applications: Pan-Tompkins heartbeat QRS detection [30] in bio-signal processing, JPEG compression in image processing, and **Harris Corner Detection (HCD)** in UAV/self tracking domain.
- **Open-source model**. To springboard future research for reconfigurable and approximate computing communities, the implementations of our multipliers, dividers, and also FPGA-customized applications will be available and open-sourced at https://cfaed.tu-dresden.de/pd-downloads.

The rest of this article is organized as follows: Section 2 presents a brief survey on the related studies w.r.t. the cross-layer approximation, imprecise multipliers, and dividers and distinguish the contribution of this work from SoAs. Section 3 summarizes a background on Mitchell's multiplication and division algorithms and the structure of multi-kernel applications. We elaborate upon the proposed architectures and approximation methodology in Sections 4 and 5, respectively. Experimental setup and results are detailed in Section 6. Finally, Section 7 draws the conclusion with an outlook to interesting future tracks.

2 RELATED WORK

To enable cross-layer approximation, the utilized techniques are adopted from both circuit- and application-level. Hence, the proper selection of inexact components w.r.t. the target platform architecture is of great importance. In this context, although a substantial amount of effort has been dedicated to ASIC-based imprecise multipliers and dividers (a quantitative evaluation of which can be found in a recent study [31]), the FPGA-specialized designs have also gained traction recently. Herein, we present a compendium of both landscapes and pinpoint SoA approaches in Table 1 and the cross-layer approximation *strategies* in Table 2.

Partial product (PP) approximation: This class of works has applied approximation on: (1) PP generation, e.g., truth table simplification in 2×2 and 4×4 multipliers and used them in a hierarchical design [32]. (2) Accumulation or reduction of PP rows into two, using 3:2 and 4:2 simplified compressors [14, 33–38] in array- or Dadda-based multipliers. (3) Addition of the generated two rows by, e.g., breaking the carry propagation path [7, 39]. The main shortcoming attributed to these works is limited scalability when transported to larger hierarchical, i.e., simplification of Karnaugh map or PP tree should start from scratch, otherwise error may drastically increase as it becomes accumulated in a recursive design approach. Moreover, compressor-based designs are usually heavy on the average relative error index or render moderate performance gains when applied on few least significant PP columns.

Table 1. Summary of SoA Approximation Approaches for ASIC- and FPGA-based Multipliers and Dividers

Approach	Mul/Div	SIMD	ARE[1] up to (%)	Description	Platform	Reported Mul/Div Gain[2]	Cross-Layer[3]/ End-to-end[4]
Partial Product Generation/ Addition/ Accumulation	✓/✗	✗	7.6	Inexact 4:2 compressor in Dadda Mul [33–36, 38]	ASIC	{A, D, P} +	✗/ Partly [35]
			1.7	Asymmetrically utilize inexact compressor in 3/4 of LSB columns [37]		{A, P} ++, D +	✗/✗
			8.4	Simplified Karnaugh map of 4:2 compressor in Booth Mul [56]		A +, E ++	
			Config.	Library of larger multiplier and adders using 2x2 instances [32, 57]		{A, D, P} ++	
			0.3	Cutting the carry propagation path in 4-, 8-bit array multiplier [7, 39]	FPGA	{A, D, P} +	
			8.5	Truth table simplification 3:2/4:2 compressor for Dadda multiplication [14]		{A, P} +	
			Conf.	Library of 4x4 and 8x8 with approximate partial products [53]		{A, P, D} +	
Truncated Mul/Div	✓/✗	✗	10.9	Leading one based: with error compensation [49], with rounding [48, 51]	ASIC	{A, P} ++	✗/✗
	✗/✓		6.7	Leading-one position based 2k+2/k+1 Div plus error reduction circuit [44, 45]		{A, D} +, E ++	
	✓/✗	✓	1.2-4.7	Variable-precision multiplier based on 8-bit truncated instances [44, 45]		{A, E} +	
Multiplicative Dividers	✗/✓	✗	2.9	Piecewise linear approximation and rounding of reciprocal of divisor [46]	ASIC	{D, E} ++	✗/✗
			6.4	Approximating reciprocal by bit manipulation [47], with truncation [47]		{A, D, E} +++	
			16.3	Approximating reciprocal using a table indexed by upper bits of divisor [58]		{A, D} +, E ++	
			4.9	Incremental approximation of reciprocal of divisor using Taylor series [59, 60]		{D, E} +++	Partly / ✗
Mitchell's Multiplication and Division Algorithms [25]	✓/✗	✗	2.9	Enhance Log accuracy: round rather truncation in piecewise approximation [61]	ASIC	{A, P} ++, D +	✗/✗
			> 3.9	Use different approximate adders in Mitchell's multiplier [62]		{A, P} +++	
			2.7	Improving accuracy of Mitchell's Mul with adding one error-correction [17]		{A, P} ++	
			2.7	Adding up to 256 error-coefficient to Mitchell's muliplier [27]		{A, P} +	
	✗/✓		3.0	Add one error-correction (with a similar approach to [17]) [16]		A +, {D, E} ++	
	✓/✗	✗	1-1.6	Adding one to five error-coefficient to Mitchell's multiplier [54]	FPGA	{A, E} ++, T +	✗/✗
	✓/✓	✓	0.8	Adding 64 error-coefficients to Mitchell's multiplier and divider [15]		{A, E, T} ++	
	✓/✓	✓	**Conf.**	**Cross-layer methodology for multi-kernel applications via SIMD Mul/Div**		**{A, E} ++, T +++**	✓/✓

[1] Average of Absolute Relative Error (a.k.a. MRED).
[2] Area/Delay/Energy/Power/Throughput.
[3] Cross-layer: adopt application-level beside circuit (Mul/Div) approximation.
[4] Analyze end-to-end *performance* gain in the entire application.

Table 2. Summary of Cross-layer Approximation Approaches[1] in Multi-layer/Kernel Applications

Techniques	Description	Strategy	Constraint
Quant. (no cross-layer)	Layer-wise quant. via greedy heuristic [63], used, in many SoAs	Greedy	Layers significance only determined by accuracy loss
Inexact Mul, Quant.	Assign Muls w.r.t. layer/weight significance, uniform 8-bit quant. NN [29]	Greedy	Layers significance determined only by accuracy loss
Inexact Mul, Quant.	Layer-wise weight tuning w.r.t. few Mul structure, in 8-bit quant. NN [20]	Genetic Alg.	Significant exploration time
Inexact Mul, Pruning	Neuron/layer pruning followed by weight tuning, uniform quant. NN [64]	Undefined	No detail for *semi*-greedy approach is discussed
Mul, Pruning, Quant.	Replacing Muls in half of neurons, uniform quant. NN [65]	L2 norm	No justification for L2 norm/selection of Mul
Add/Mul, Prec. Scaling	Replacing LSBs' Add/Mul in ECG kernels, fixed to 16-bit precision	-	Approximate kernels aggressively, in appearance order
Mul/Div, Prec. Scaling	Minimally biased Mul/Div, followed by (mixed) precision scaling of kernels	Greedy	Greedy heuristic → no guarantee for an optimal solution

[1] Valuable works like Reference [66] assess voltage scaling instead of inexact units (not in a cross-layer approach), hence, are not in the scope of this paper.

Division with inexact subtractor: This family of dividers [40–43] has replaced the precise subtractors with imprecise counterparts. Generally, such array-based dividers offer high accuracy, however, the resource savings delivered by them are not significant due to maintaining the array structure [44, 45]. Furthermore, they do not gain significantly on delay index (remains notably larger than its peer-sized multiplier).

Multiplicative dividers: In this branch of studies, approximation is applied on the reciprocal of divisor, using, e.g., linear piecewise approximation [46]. In some studies, operands are also truncated/rounded [47]. Moreover, when the truncation of divisor goes beyond a relatively small length, say, four bits, the accuracy of its corresponding reciprocal starts to degrade significantly [46].

Resizing to smaller multiplier/divider: More recent truncation-based approaches; they dynamically determine the position of leading one and then utilize a narrower-width accurate instance of the unit (References [48, 51] for multiplier and References [44, 45, 52] for divider). Albeit they offer notable resource improvements, these truncation techniques suffer from error cases

equal to 100%. In addition, the latency of such dividers still remains significantly higher than a same-sized multiplier.

FPGA-customized multipliers: Truncating carry propagation and LUTs at LSBs of 4×4 multipliers have been customized for FPGAs [7, 39, 53]. Furthermore, an approximate compressor has been recently proposed in Reference [14], geared toward an LUT-oriented implementation. Despite the specialized customization in these schemes, their resource savings have not been significant. This has accentuated the demand for exploring other techniques that can enable higher savings with an acceptable accuracy.

Logarithmic multiplier and dividers: In a more recent trend of studies, studies have adopted Mitchell's algorithms, which translate multiplication (division) into logarithm and addition (subtraction) in the logarithmic representation. Such transformations generally offer higher performance gains, especially in terms of division latency, which is bridged to that of a same-sized multiplier. Such improvements albeit come with the cost of relatively high error (average relative error of 3.8% for multiplication and 4.1% for division). In this context, various schemes have been recently proposed to reduce the error, customized for both ASIC and FPGA. Targeting ASIC, authors in MBM [17] have proposed a single error-reduction term for multiplication, and they employed a similar scheme, later on, for division (INZeD [16]). However, as a single error-reduction term weakly fits all input combinations, it eventuated in many output overflow cases when added to the original designs. To alleviate such problem and also targeting FPGAs, we have recently proposed two FPGA-specialized designs, LeAp [54] and SIMDive [15]. While in LeAp three error-reduction schemes have been presented to reduce ARE of multiplication to ~1%, a more generic approach has been proposed in SIMDive for both multiplication and division that resulted in improving all accuracy metrics (its approach can be further expanded to deliver higher accuracy). SIMDive, REALM multiplier [27], and CADE floating point divider [55] all employ the similar idea of dividing the power-of-two intervals for each operand to 2^F segments (based on F MSBs of fractional parts) and apply a coefficient for each pair of segments to minimize the average relative error in each of sub-regions. SIMDive [15] has enabled further architectural features: first is the plasticity to switch between hybrid functionality of Mul/Div on-the-fly, without the need for reconfiguration. Moreover, its architecture is customized for FPGAs: The LUT-specialized implementation of error-reduction circuit and the addition of it through ternary adder altogether has negligible overhead over Mitchell's baseline (see Table 4). Finally, it enables the ability to operate in SIMD mode, therefore, it is an appropriate candidate to be utilized for dynamic precision scaling techniques and a variety of applications featuring SIMD potentials, e.g, NNs.

Sensitivity analysis strategies for error-resilient applications: In general, the adopted approaches are either application-specific or based on error-injection. For example, ARC, presented initially in Reference [67] and further improved in ASAM [68], distinguishes the error resiliency of applications' innermost loops (considering they consume ~70% of overall time in recognition, data mining, and search applications) by injecting random errors to the internal variables of loops and monitor the fluctuations at the application output. ASAC [69] discovers the amenability of variables to approximation through randomly applying bit-flips on 16 LSBs of the target variable. PAC [70] has upgraded ASAC methodology through assigning a *Degree of Accuracy*, translated to the number of bits that can be approximated as far as the pre-defined quality threshold is satisfied. Targeting multi-kernel applications, most of the researches have focused on the domain of **Neural Networks (NNs)**. Hanif et al. [71] have evaluated the effects of quantization and imprecise Add/Mul in convolutional layers, individually, and observed the effects on the image classification accuracy. This is while works such as References [72–74] and their references identify less critical neurons based on, e.g., Taylor decomposition [75] or partial derivative on the loss function at the output of each neuron. In short, the random error injection mechanisms cannot comprehensively

characterize the error-resiliency. Furthermore, the above-mentioned methods mostly categorize the variables either as approximable or sensitive without showing the end-to-end performance-quality tradeoff of each kernel individually in the multi-kernel application. This can also highlight the priority of kernel for approximation.

QoR-Performance optimization strategies: in brief, mainly three classes of heuristics have been utilized for **Design Space Exploration (DSE)** problems: (1) **Genetic Algorithm (GA)** approaches [76–79] expand the search space by generating a population of configurations in each iteration based on crossover and mutation actions. Besides having high computational complexity, the quality of later configurations in this class of heuristics depends heavily on the initial generations. (2) Simulated annealing [80, 81] randomly samples an action that improves the target metric, while other actions are also accepted based on a predefined probability. Such algorithms are known to be time-consuming due to the large number of generated configurations. (3) Greedy strategies have less complexity and usually better runtime than multi-objective GAs, but they have mostly considered ΔQoR (not the relation between ΔQoR and ΔPerformance) as their saliency metric [29, 63, 66, 110] for their selection strategy.

Precision-tuning strategies: literature studies that target (mixed-)precision tuning can be divided into two categories. Either they adjust the precision of the individual *operations* (intra-kernel granularity) or adjust precision of a group of operations as a kernel (inter-kernel granularity) in a multi-kernel application. As an example for the former, the study of Reference [82] also considers SIMDization in its **Word-Length Optimization (WLO)** algorithm. The priority of blocks for SIMDization is based on their contribution to the overall execution time when evaluated in individual CONV, FIR, and IIR kernels. However, the efforts for enabling efficient mixed-precision configurations of multi-kernel applications has been limited. The recent work of Reference [83] tackles the problem of WLO in cascaded image processing kernels. The adopted search strategy of this study is based on a greedy gradient descent that again targets the quality constraint and later on a local search has been performed to minimize the cost. Therefore, a low-cost methodology is required to adjust approximation knobs of multi-kernel applications in such a way that can maximize performance gains for a user-defined quality. This would be particularly beneficial for programs running on edge devices, as it can efficiently address the on-the-fly accuracy-energy requirements w.r.t. the dynamic changes of, e.g., environment. In this context, the methodology of Plasticine first performs an offline sensitivity analysis followed by the proposed greedy heuristic that considers the $\frac{\Delta ADP}{\Delta QoR}$ as its saliency metric. When evaluated on three multi-kernel case studies from different application domains (see Section 6), the adopted strategy of Plasticine has been able to improve performance-gain over existing greedy approaches and generate the Pareto- or near-Pareto results.

Cross-layer approximation for multi-kernel applications: Studies in this branch usually have applied precision scaling along with imprecise multipliers. It should be noted that orchestrating an *adaptable* precision scaling methodology is an application-level technique [65, 84–87] (such as Plasticine and SoA IBM RAPID chip [88, 89], both of which support precision scaling technique in a multi-precision, SIMDized fashion), and different from non-adaptable truncation of operands (architecture/circuit level as in studies such as Reference [90]. Nevertheless, the focus of cross-layer designs has been almost all targeted for NNs [20, 64, 65, 91]. In their adopted layer-wise approaches, ranking the layers for quantization is determined merely based on their robustness to precision scaling [20, 63]. Such a strategy neglects the significance order of layers on the end-to-end gained performance (when approximated by the same technique), or in general, the relation between performance gain and QoR loss. Besides NN studies, there exists another work, XBioSip [92], that has applied a kernel-wise approximation on a multi-kernel application (Pan-Tompkins

algorithm for QRS complex detection). This cutting-edge work also serves as the pioneer of bio-signal *processing* approximation and has shown remarkable savings by deploying inexact Add/Mul in all kernels. In this work, the precision of kernels is uniform, fixed to 16-bit, and each kernel uses different numbers of 2×2 simplified Muls for its LSB computations. Therefore, beside having a fully customized ASIC implementation, the effect of changing precision is not investigated in this study. Moreover, the approximation strategy of this work is neither cross-layer nor based on a comprehensive sensitivity analysis. In fact, XBioSip has aggressively approximated kernels in their normal appearance order without providing a saliency measure to rank the significance of kernels, as discussed above.[2] The remainder of works have replaced an arithmetic unit with the inexact one, mostly in a single kernel of multi-kernel application, without exploring the effects of *changing* kernel's precision for a cross-layer approach. In short, to the best of our knowledge, no work has proposed a generic cross-layer methodology to appropriately adjust the knobs while also utilizing SIMD potentials to more efficiently support a cohesive three-tiered hierarchy. *Such an adaptable approximation can be served as a viable energy-efficient and high-throughput solution, highly desirable for ubiquitously used image processing and health monitoring edge nodes.*

3 PRELIMINARIES AND BACKGROUND

3.1 Mitchell's Multiplication and Division Algorithm

As shown in Equation (1), Mitchell's algorithms perform imprecise multiplication and division in the logarithmic representation of numbers. Consider the binary representation for N-bit unsigned input A, which can be written as Equation (2), where k reveals the position of the leading one. The rest of the bits (starting from position $k-1$ to 0) are considered as the fractional part and fall in the range of $0 \leq x < 1$.

$$Mitchell'sAlgorithms = \begin{cases} P = A \times B \xrightarrow[\text{Log}]{\text{Approx.}} \widetilde{Log_P} = \widetilde{Log_A} + \widetilde{Log_B} \xrightarrow[\text{Anti-Log}]{\text{Approx.}} \tilde{P} = 2^{\widetilde{Log_P}} \\ D = A \div B \xrightarrow[\text{Log}]{\text{Approx.}} \widetilde{Log_D} = \widetilde{Log_A} - \widetilde{Log_B} \xrightarrow[\text{Anti-Log}]{\text{Approx.}} \tilde{D} = 2^{\widetilde{Log_D}} \end{cases} \tag{1}$$

$$A = 2^k + \sum_{i=0}^{k-1} 2^i b_i = 2^k(1+x) \xrightarrow{e.g.} 43 = 2^5(1+0.01011)_2, 10 = 2^3(1+0.01)_2 \tag{2}$$

In linear mathematics, $Log(1+x)$ is approximated to x for this range of $0 \leq x < 1$[3]; therefore, the approximate logarithm of input A is:

$$Log(A) \simeq k + x \rightarrow Log(43) \simeq (101.01011)_2, Log(10) \simeq (11.01)_2. \tag{3}$$

After applying the same step on the second input to get its approximate Log, the summation (subtraction) of two parts is obtained in Equation (4) (Equation (5)).

$$\widetilde{Log}(\tilde{P}) = (k_1 + k_2) + (x_1 + x_2) \rightarrow K_s = (1000)_2, X_s = (0.10011)_2, \tag{4}$$

$$\widetilde{Log}(\tilde{D}) = (k_1 - k_2) + (x_1 - x_2) \rightarrow K_s = (10)_2, X_s = (0.00011)_2. \tag{5}$$

[2] Although XBioSip tries to tackle this problem (search for higher gain with less QoR loss) by reducing approximately LSBs of LPF by 2 and increase in HPF by 2 (their so-called diagonal search), this strategy still lacks generality for a multi-kernel program.

[3] Throughout this article, Log stands for Log_2.

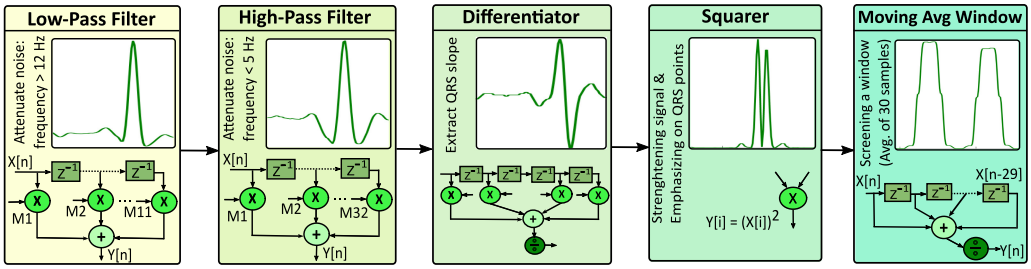


Fig. 2. Kernel structure of Pan-Tompkins QRS detection application.

Finally, by adding the SIMDive coefficient and applying anti-log (a shift operation), binary representation of approximate product (quotient) are derived by Equation (6) (Equation (7)):

$$\tilde{P} = \begin{cases} 2^{k_1+k_2}(1+x_1+x_2), & x_1+x_2 < 1 \\ 2^{k_1+k_2+1}(x_1+x_2), & x_1+x_2 \geq 1 \end{cases} \rightarrow \begin{cases} \tilde{P} = (110011000)_2 = 409 \\ P_{accurate} = 430, \end{cases} \tag{6}$$

$$\tilde{D} = \begin{cases} 2^{k_1-k_2-1}(2+x_1-x_2), & x_1-x_2 < 0 \\ 2^{k_1-k_2}(1+x_1-x_2), & x_1-x_2 \geq 0 \end{cases} \rightarrow \begin{cases} \tilde{D} = (100)_2 = 4 \\ D_{accurate} = 4. \end{cases} \tag{7}$$

3.2 Heartbeat QRS Detection with Pan-Tompkins Algorithm

It has been shown that *processing stage* consumes 70% of total energy in wearable health nodes. Interestingly, both sensing and the processing algorithms in bio-signal analysis exhibit error-resiliency [92, 93], making them potential candidates that can benefit from approximation. The widely used Pan-Tompkins algorithm [30] detects the main heartbeat peak (QRS) in a **Electrocardiogram (ECG)** signal (cardiac arrhythmia results in incorrect ECG wave). Pan-Tompkins application, shown in Figure 2, serves as the main standard of QRS detection in wearable devices. This algorithm determines the number of QRS peaks in the samples, through five stages, being: ECG signals are first passed through an 11-tap low-pass followed by a 32-tap high-pass filter to attenuate high- and low-frequency noises. The filtered stream is fed to a 5-tap digital differentiator to extract the QRS slope information. Squaring is applied to amplify the signal strength and emphasize the higher-frequencies, i.e., QRS points in the complete P-QRS-T complex [92]. Finally, by screening the signal (averaging samples to the width of the moving window), R peaks are detected.

3.3 JPEG Compression

This compression technique is used in many image/video processing programs. JPEG algorithm, shown in Figure 3, is initiated by RGB to YCbCr color conversion, which separates brightness-from color-information of the image (human eye is more sensitive to light than colors). This is followed by applying 8×8 2D-DCT to identify and remove insignificant portions of information in the image, without a significant loss in the visual effect. The main compression takes place via quantization, in which each DCT-converted data is divided to the user-defined compression ratio (a.k.a. quality factor). In this article, we have adopted the normal compression ratio of 20:1, which lies in the range of acceptable compression rates for aerial applications [94]. Afterwards, the quantized coefficients are sorted in a zigzag sequence pattern with a frequency-ascending order (low-frequency components, which are more important to the human visual system will be retained through this re-arrangement and others will be discarded). Finally, Huffman scheme is applied for encoding the image.

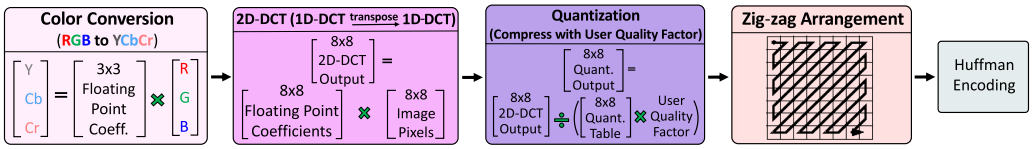


Fig. 3. Kernel structure of JPEG Compression application.

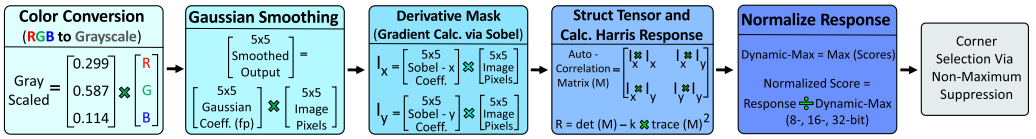


Fig. 4. Kernel structure of Harris Corner Detection application.

3.4 Harris Corner Detection Algorithm

HCD is ubiquitously utilized in tracking algorithms and as a feature extraction algorithm in many vision-based programs such as indoor localization, self-estimation, or object tracking in UAV, due to its robustness in detecting corners in noisy images, captured from on-board camera [95]. This technique locates the corners, i.e., pixels having strong intensity variations with their local neighborhood, in both vertical and horizontal directions. The extracted corners are then joined in consecutive frames, forming a vector in the direction of movement. As illustrated in Figure 4, this algorithm includes six steps: The image is first converted from RGB to grayscale to reduce the load of computations. Then a 3×3 Gaussian filter is applied to remove noise of images taken from on-board camera. It is reported that this pre-processing step remarkably improves the quality of corner detection [96]. Afterwards, the gradient of the image in both horizontal and vertical directions is computed using Sobel operators. These derivatives are then multiplied to each other to construct the so-called auto-correlation matrix. Then the corner strength (a.k.a. Harris response) is calculated, which reveals the intensity of changes to surrounding pixels: Negative response value means the pixel probably belongs to an edge, while small and large positive, empirically chosen, values imply that the pixel belongs to a flat region and corner, respectively. Finally, the corners represented by the highest-value responses are determined via Non-Maximum Suppression, which finds the maxima in windows of 5×5 pixels.

4 PROPOSED SISD AND SIMD MULTIPLIER-DIVIDER ARCHITECTURE WITH HYBRID FUNCTIONALITY

In this section, we elaborate upon our proposed SISD/SIMD multiplier/divider architecture, SIMDive [15]. Afterwards, we present our generic error-reduction approach, which is applicable to both multiplication and division, that are used in this manuscript as circuit-level techniques.

4.1 SIMDive: LUT-Oriented Approximate SISD and SIMD Multiplier/Divider

The overall structure of proposed SIMDive is illustrated in Figure 5(a). Controlling signals, *precision* and *Mul/Div mode,* shown in Figure 5(a), serve to establish diverse sub-word size and functionalities of each module, respectively. We used one-hot encoding preferred by FPGA manufacturers, as proven to be more resource-efficient than binary encoding for reconfigurable fabrics [15, 97]. In SIMDive, leading one detection is calculated separately for each 8-bit on top of 4-bit LODs (implemented by directly configuring LUTs). The structure of 8-LOD is as follows: Detecting the presence of a bit with value "1" is orchestrated in parallel for each 4-bit segment of the input by directly

utilizing one 6-LUT. The LUT is configured to logical OR function, acts as a zero-detection flag, and detects whether all four bits are zero (Flag-LUT). In parallel, a second 6-LUT is configured two 5-LUTs in such a way to reveal the position of leading-one in the 4-bit segment (LOD4-LUT). Finally, based on the resulting bits from the output of these 6-LUTs, the position of leading one in 8-bit LOD is found via a priority logic. In the 8-bit LOD, the position of leading one equals to the concatenation of {*Location index of most significant segment*, *Leading one position in that segment*}, e.g., *leading one position in "00110101" = {{1},{01} = {101}} in binary (5 in decimal).* The first part ({1}) is the output of Flag-LUT that has been applied on upper 4-bit segment, the second part ({01}) is also the result of LOD4-LUTs on the upper segment. A similar method has been also adopted for 16- and 32-bit LODs. For example, in 16-LOD, if the upper half of the operand is zero (obtainable by applying logical-OR function on the outputs of Flag-LUTs on the 4-bit segments of the upper-half (bit 15 downto 8), the 16-bit LOD is equal to lower 8-bit LOD. Else, the position of leading-one is 8+leading-one position in upper-half 8-LOD. Afterwards, addition of integer and fractional parts shown in Figure 5(b) are fulfilled by connecting Virtex-7 slices, each of which can implement a 4-bit addition. As shown in part (c) of this figure, each slice includes four 6-LUTs and its associated fast carry chains, together implement a **Carry Look-Ahead Adder (CLA)**. Extending the 8-bit addition to 16- and 32-bits in our SIMD architecture is easily achieved by connecting the C_{out} from previous adder to the C_{in} of next adder, handled by yellow multiplexers in part (a) of this figure. Division is also performed by altering additions to subtractions (recalling Equation (6) and Equation (7)), using 2's complement modules. SIMDive enables two features: (1) supporting *mixed-precision* and *mixed-functionality*: the proposed SIMD Mul/Div can either operate as a single 32×32 unit or be decomposed into a twin 16×16, one 16×16 and two 8×8, or quad 8×8 units, each of which can act separately as a multiplier or divider. Moreover, in case of sub-word parallel processing, a complete 32-bit unit is not occupied for a division and it can also orchestrate multiplication if needed. (2) The dynamic hybrid-functionality in our design eliminates the need of reconfiguration and is of great interest, especially in multiplication-intensive workloads with fewer division (appealing for variety of application domains). The separate *data-size* signals can also be employed in fine-grained power-gating techniques.

It is worth noting that to avoid overflow in 2N-by-N bit standard division, the condition of $dividend < 2^N \times divisor$ should be satisfied [98], meaning that scaling in the divider lies in the range of 2^0 to 2^{N-1} (similar to Reference [16]). Thus, for the output, only N−1 bits are used from subtractors and N LSBs from $\widetilde{\log}_{dividend}$ is omitted. This approach reduces the logic size for subtractor and barrel shifter, while it does not affect the accuracy. However, supporting SIMD adds on to the complexity of sub-modules. For example, $4 \times 3 = 12$ bit is used for LOD in 32-bit SIMDive or 2- and 4-MUX units are used to select the functionality and sub-word length in intermediate adders.

4.2 Proposed Lightweight Error-reduction Scheme, Specialized for FPGA Architectures

Mitchell's error for 8-bit multiplication and division operations (formulated in Equations (8) and (9)) is plotted as a heat map in Figure 6. Through this figure, the following points can be observed, which are also noted and endorsed by SoAs [16, 17, 27, 55], as detailed below:

- Figures 6(a) and (d) show different error magnitude in each power-of-two interval. This means that adding a single correction term to the output cannot fit for all multiplier/divider sizes [27, 55].
- Figures 6(b) and (e), illustrated based on Equation (8) and Equation (9), prove proportional replication of error in each power-of-two: Irrespective of k_1 and k_2, unique schemes for each

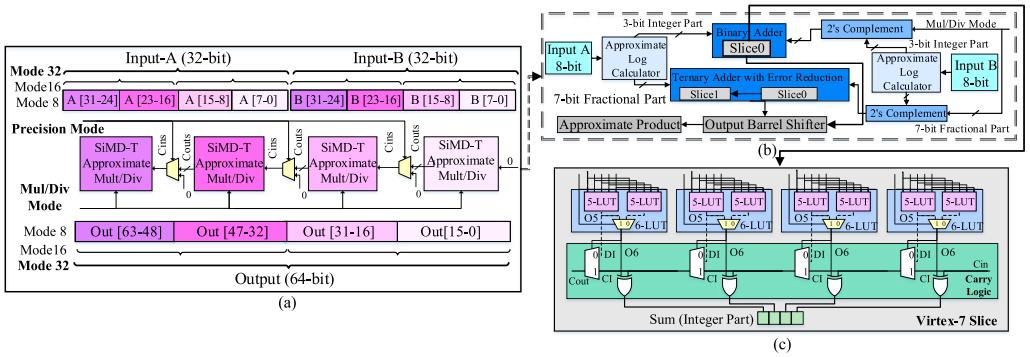


Fig. 5. (a) SIMDive structure, (b) proposed 8-bit multiplier/divider, (c) Virtex-7 slice (used for addition/subtraction of integer/fractional parts).

of Mul/Div may fit all sizes, and they can be added to fractional parts before scaling to save more resources [16, 17]).

- Figures 6(c) and (f) exhibit the non-uniform but symmetrical error distribution: Errors tend to be the same at the beginning and end of each power-of-two interval, encouraging the same reduction approach for all multiplier or divider sizes [16, 17].
- Finally, parts (c) and (f) of the figure also show diverse change in relative error. This means employing a single error-coefficient to the whole interval cannot significantly improve the multiplication error [27]. Moreover, by analyzing the behavior of error in integer division in this article, we further expand the idea behind References [15, 27] (for multiplication) to reduce average and peak of absolute error and also better handle overflow cases, compared with References [16, 17].

$$E_P = P - \tilde{P} = \begin{cases} 2^{k_1+k_2}(x_1x_2), & x_1 + x_2 < 1 \\ 2^{k_1+k_2}(1 - x_1 - x_2 + x_1x_2), & x_1 + x_2 \geq 1, \end{cases} \tag{8}$$

$$E_D = D - \tilde{D} = \begin{cases} 2^{k_1-k_2}\,\frac{(x_1(x_2-1)+x_2-(x_2)^2)}{2(1+x_2)}, & x_1 - x_2 < 0 \\ 2^{k_1-k_2}\,\frac{(x_1x_2-(x_2)^2)}{1+x_2}, & x_1 - x_2 \geq 0. \end{cases} \tag{9}$$

Coalescing the insights from above points incentivizes using multiple error-reduction terms, appropriately opted based on fractional parts. In our error-reduction scheme, we attempted to optimize two factors: (1) *error magnitude* × *error distribution* in each region (can be estimated as the integral of error-magnitude in that region). (2) Partitioning overhead, which depends on the number of MSBs checked in fractional parts. Therefore, as illustrated in Figures 6(c) and (f), the squarish region for all combination of inputs is subdivided to $2^3 \times 2^3 = 64$ sub-regions by merely using 3 MSBs of inputs. Hence, we assign a distinctive coefficient to each of these regions, representing their average error for each sub-interval. For deriving the error-reduction coefficients that minimize the average relative error in each sub-region, we have followed the mathematical approach that is detailed in REALM multiplier [27]. For the sake of reproducible results, the final INIT hexadecimal value for the 16-bit coefficients for both multiplier and divider are also shown in Table 3.

To achieve a lightweight implementation of the proposed error-reduction scheme and also to benefit from the underlying FPGA structure, we have proposed the following approach that efficiently utilizes 6-LUTs: We assign 3 MSBs of each fractional part to inputs of the LUTs, each of which is responsible for calculating one bit of the 64 error-coefficient. In fact, 64 output entries

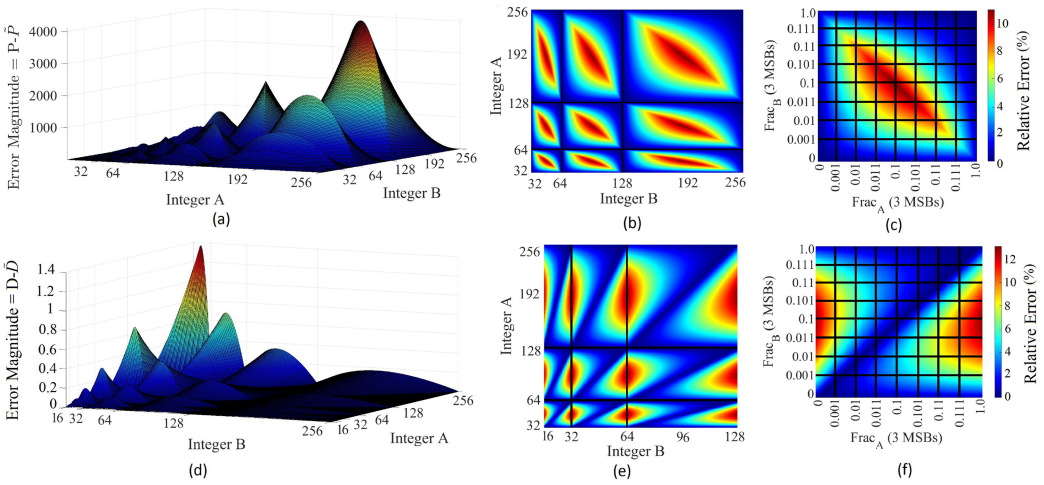


Fig. 6. Mitchell's error: 8 × 8 multiplier (top), 16/8 divider (down). Parts (b) and (e) show error repeats the same behavior in each power-of-two interval (i.e., each multiplier or divider size). Parts (c) and (f) show the proposed error-reduction scheme: Dividing squarish zone of fractional-parts to 64 regions, each has a specific coefficient.

Table 3. Configuration INIT Value (Hexadecimal) for Error-reduction LUTs in Multiplier and Divider

Multiplier				Divider			
Coeff [15]	0000000000000000	Coeff [7]	7AC120E9148D9AD2	Coeff [15]	0000000000000000	Coeff [7]	04AD3A5428BA3D06
Coeff [14]	0000000000000000	Coeff [6]	43012121852990E8	Coeff [14]	0000000000000000	Coeff [6]	D4D18E4EA73F3994
Coeff [13]	0000000000000000	Coeff [5]	D592D1A2C0A5336B	Coeff [13]	0000000000000000	Coeff [5]	F6661C4A32DB3527
Coeff [12]	0000000C18100000	Coeff [4]	94190083448C1259	Coeff [12]	0000018181810000	Coeff [4]	4243DC139C9718A2
Coeff [11]	00061E32266C7800	Coeff [3]	0018123B00013397	Coeff [11]	000386466262E100	Coeff [3]	542DDC1C0C97AC12
Coeff [10]	00396D566C2A2460	Coeff [2]	0102021082046913	Coeff [10]	008C482AD6D0D8E0	Coeff [2]	22B2367E87019850
Coeff [9]	0F4A2401C5B6F69C	Coeff [1]	1551008100820650	Coeff [9]	0E90D2614478D498	Coeff [1]	DA48B05937950D5A
Coeff [8]	361C90D046999F12	Coeff [0]	9500000100010103	Coeff [8]	B375AAD2693CB34C	Coeff [0]	B8427A22AF76B33F

of ith LUT determine ith bit of the 64 coefficients (in binary representation). Therefore, by using only 8, 16, and 32 LUTs, we can efficiently determine 64 error-coefficients for the target 8-, 16-, and 32-bit operation, respectively. Having 64 coefficients appropriately calculated based on the combination of both operands addresses both drawbacks discussed in Section 2, i.e., neglecting magnitude of error due to separately approximating each operand and overflow cases (in fact, the overhead of separately handling such overflow cases is not clearly discussed in MBM [17] and INZeD [16].)

Our approach is highly suited for LUT-oriented platforms as well: LUTs and their associated fast carry chain in Xilinx UNISIM library [99] can be configured to implement a ternary adder. This perfectly suits our error-healing approach, as *we are able to combine the process of adding error-reduction term with fractional parts within the same resources in a single step*. Regardless of adder size, only one more bit at MSB is needed in ternary addition (compared to binary version), since frac1_i+frac2_i+$\text{error\_coefficient}_i$+$C_{in}$ (C_{out} from previous bit) may result in 3 bits, utilization of one more LUT at the end of the chain [26]. Moreover, the delay of FPGA primitives is fixed and, therefore, adding error-reduction term at the same time when fractional parts does not impose additional overhead to the design. This is while in REALM [27], MBM [17], and INZeD [16] a non-trivial circuitry is needed for addition of error-reduction terms to the original Mitchell's circuits.

Shared modules in Hybrid/SIMD designs[4]: Besides the two's complement modules, integration of multiplier and divider into a hybrid marginally affects the complexity of the circuit through including 2-MUX units to select the mode in sub-modules (except LOD/final barrel shifter that are equal in both modes). However, to support simultaneous scalings for sub-word length in SIMD mode, the LOD, error-reduction, and final shifter become more complex. For example, in 32-bit LOD, instead of 5 for SISD mode, $4 \times 3 = 12$ bit is used to also support simultaneous 8-bit leading one detection in SIMD mode (this is similar for integer part adder). However, the complexity of larger adder that is employed for fractional parts is not increased significantly, as the carry out of consequent 8-bit are used as carry-in in the following 8-bit for 16- and 32-bit modes. It is worth noting that there is also a small similarity between SIMDive error-coefficients, as 3 and 4 MSBs are similar in each of multiplier and divider. In short, although the overall overhead of additional circuitry for managing hybrid and SIMD modes is not negligible in 8-bit hybrid design, it gets dwarfed in larger SIMD designs (see Table 5), thanks to transforming the 2D structure of array multiplier (divider) to 1D adder (subtractor) in the logarithmic representation.

SIMDive is highly suitable to be utilized in a cross-layer approximation methodology, the rational behind which is multi-fold: ① Such resource-efficient SIMD architecture bodes well for a cross-layer approach, as it enables a chain of techniques, i.e., precision scaling on top of Mitchell-based Mul/Div (detailed results are presented in Table 4 and Table 5 of Section 6). Especially in case of division, SIMDive reduces the drastically high latency of accurate divider, nearly to latency of same-size multiplier. ② It yields high accuracy, especially negligible error-bias, which as mentioned earlier can play a pivotal role in approximation of consecutive kernels having an aggregation-based structure. ③ The plasticity provided by SIMDive hybrid mode enables on-the-fly switching between two functionalities and without the need for reconfiguration.

Thus, henceforth, we continue our cross-layer methodology with SIMDive (we have further designed 8-bit hybrid SISD Mul/Div and 16/8 SIMD Mul/Div, which are of utilized configurations for the experiments of applications in this manuscript). Nevertheless, other resource-efficient designs having small error index can also be utilized in our cross-layer methodology.

5 PROPOSED KERNEL-WISE SENSITIVITY ANALYSIS AND CROSS-LAYER APPROXIMATION STRATEGY

The main goal is to apply cross-layer approximation on *multi-kernel* applications. The techniques are from different levels of abstraction and applied in a cross-layer hierarchy: precision scaling (application-level) on top of inexact multiplication and division (circuit-level). Achieving this, we first perform a kernel-wise sensitivity analysis to determine error-resiliency of individual kernels to approximation techniques. The result of this analysis is inputted to the proposed heuristic.

5.1 End-to-end Kernel-wise Sensitivity Analysis

This analysis is motivated by the facts that not only kernels contribute differently to each of performance metrics, but also their significance order might differ in the *gained performance* and *lost QoR*. Therefore, the main goal of this analysis is to find ① the sensitivity of final QoR to approximation of each kernel individually, and ② the end-to-end performance improvements, obtained from each technique. Achieving this, the performance-QoR tradeoff is measured by approximating one kernel at a time, when that kernel is subjected to each of n techniques and the rest of kernels are 32-bit accurate. The output of the analysis includes n lists, each list contains a pair of

[4]Please note, considering the default transformation/optimization during synthesis and implementation, the percentage of overhead for each of modules does not remain similar after combining the modules in the final integrated design.

{ΔADP, ΔQoR} for each of kernels, subjected to that technique. This pair thus demonstrates the end-to-end performance-QoR tradeoff for the application when only that kernel is approximated. We have refrained from approximating Add/Sub operations, due to two reasons. The first is the small area and energy of Add/sub versus multiplier and divider (recalling Figure 1). The second is motivated by the experiments conducted in previous studies such as Reference [31]. Such experiments have shown that JPEG compression and other matrix multiplication-based applications show higher accuracy fluctuation to approximation of additions rather than multiplication (we have also evaluated the effect of truncation on addition operations, discussed in Section 6). In fact, as discussed earlier, for the applications having aggregation-based structure, low error-bias plays a pivotal role for the imprecise component. Nevertheless, the analysis of different inexact adders could be further investigated in the future tracks.

5.2 Greedy-based Cross-layer Approximation Heuristic

We propose a generic heuristic approach to generate *near-optimal* mixed-precision configurations—in short amount of time—that can *maximize performance gain* for different user-defined *accuracy thresholds*. As discussed earlier, ΔQoR has been considered as the saliency metric in SoA greedy heuristics for their selection strategies. This neglects the effect of Δ Performance (here, Area-Delay Product-ADP), or in general, relation between these two. Therefore, we propose $\frac{\Delta ADP}{\Delta QoR}$ in our heuristic, as the saliency measure that appropriately reflects the potential end-to-end performance improvements over a possible accuracy degradation, when approximating a kernel.

The inputs of our proposed design generation methodology are the user QoR constraint and the information gathered from the sensitivity analysis, resulting in four list $L_1 - L_4$, are for approximating multiplication, division, or down-scaling the precision of the kernel to either 16- or 8-bit. Each list contains a pair of {ΔADP, ΔQoR} for each kernel, which demonstrates the end-to-end performance-QoR tradeoff of the application, when only that kernel is subjected to the target approximate technique and the rest are intact. As discussed, for performance gain, we have opted ADP improvement. Nevertheless, other metrics such as area or delay can also be considered, based on the designer's objective.

The pseudo-code of our methodology, presented in Algorithm 1, is elaborated as follows: Initially, all application kernels are uniformly set to the highest precision, i.e., 32-bit with accurate operations. In the first step, the *Significance List* is formed by calculating the $\frac{\Delta ADP}{\Delta QoR}$ for each pair of {kernel, technique}. After merging the lists, it is sorted in a descending order, showing the significance order of techniques that resulted in providing higher performance improvement and less fluctuations in QoR, when considered individually. Then, the approximation is applied on kernels with an iterative approach: In each iteration, the heuristic applies approximation that is placed on top of the significance list. It is projected that the primary-applied techniques are replacing original multiplication and division with SIMDive versions, owing to their very low error metrics and high performance gain potentials. Especially in case of division, significant latency improvements are achieved by switching to 32-bit SIMDive divider (46.1 ns in 32-bit accurate to 6.3 in 32-bit approximate, referring to Table 4) than down-scaling the precision with an accurate divider core (46.1 to 21.6 ns). The generated configuration is then appraised on miscellaneous samples to verify whether the end-to-end user-defined accuracy threshold is satisfied.[5] Achieving credible results, the heuristic is evaluated on miscellaneous ECG samples from MIT-BIH [100] and images

[5]Note, error estimation formulation in multi-kernel programs (rather than applying a heuristic) is an open-research problem and not a claimed contribution. Rather, proposed $\frac{\Delta ADP}{\Delta QoR}$ factor enables a good vision to navigate toward *near-optimal* configurations in short amount of time.

ALGORITHM 1: Cross-Layer Approximation Methodology for Multi-Kernel Applications

Input: L1: {End-to-end ADP Saving $_{\text{ApproxMul}}$, QoR-Loss} ∀ 32-bit Kernel
Input: L2: {End-to-end ADP Saving $_{\text{ApproxDiv}}$, QoR-Loss} ∀ 32-bit Kernel
Input: L3: {End-to-end ADP Saving $_{\text{PresScale}}$, QoR-Loss} ∀ Kernel, 16-bit Precision
Input: L4: {End-to-end ADP Saving $_{\text{PresScale}}$, QoR-Loss} ∀ Kernel, 8-bit Precision
Input: User-QoR-Const, Kernel-List
Output: Kernels [Architecture] /* Precision & Mul/Div Structure */

```
Significance-List = Array [];
        /* Compute the effect of imprecise Mul, Div, and precision scaling, on each kernel, individually */
for i in Kernel-List do
    Significance-List ← L1 [i]-ΔADP/ΔQoR(Mul);   /* All multipliers of this 32-bit kernel are approximated */
    Significance-List ← L2 [i]-ΔADP/ΔQoR(Div);   /* All dividers of this 32-bit kernel are approximated */
    Significance-List ← L3 [i]-ΔADP/ΔQoR(16);    /* This kernel is scaled to 16-bit, others are 32-bit */
    Significance-List ← L4 [i]-ΔADP/ΔQoR(8);     /* This kernel is scaled to 8-bit, others are 32-bit */
end
Descending Sort (Significance-List);
while (not timeout) do
    for i in Significance-List do
        Design_Approx = Kernels[Significance-List_i];   /* Approximate in descending order of ΔADP/ΔQoR */
        Output-QoR = Evaluate (Design_Approx);
        if Output-QoR ≥ 0.95 × User-QoR-Const then   /* Also explore temporary configs */
            if Output-QoR ≥ User-QoR-Const then
                Design_temp ← Design_Approx;   /* Update the Potential Configuration */
            end
            i ← i + 1;
            go to 10;
        else
            Break;
        end
    end
end
```

from UAV123 [101], VisDrone [102], and UAVid [103] databases. In our methodology, we slightly increase the freedom to explore *temporary* configurations, having a negligible accuracy difference with the pre-defined threshold (1%–5%). This freedom has enabled the heuristic to provide final QoR-satisfied solutions with also higher resource savings. The reason behind this lies in following interesting observation: Sometimes by combining approximated kernels, the accuracy slightly increases. Our profiling has revealed that it is partly due to the near-zero biased errors of SIMDive units, which are able to neutralize each other in consecutive kernels, thereby *slightly* increasing approximation opportunities, when combined. In this iterative approach, whenever the accuracy of the generated configuration crosses our threshold (with 1%–5% difference with the user-defined threshold), the heuristic backtracks to the prior user-satisfied configuration and follows the search by evaluating the next candidate that has highest priority. It is worth mentioning that the proposed cross-layer methodology for multi-kernel applications is agnostic to both the underlying architecture and the chain of techniques. Our intuition behind adopting the precision scaling is three-fold:

Table 4. Accuracy-resource Tradeoff of SISD Approximate Multipliers and Dividers in the FPGA Implementation

Multiplier								Divider							
8 × 8	Area (LUT)	Latency (ns)	Power (mW)	Energy (uJ)	ARE (%)	PRE[2] (%)	Error Bias (%)	8/4	Area (LUT)	Latency (ns)	Power (mW)	Energy (uJ)	ARE (%)	PRE (%)	Error Bias (%)
Acc-IP NP [15]	72	5.2	15.10	79.06				Acc-IP NP [12]	51	10.74	11.82	128.44			
DRUM-4 [50]	53	4.57	15.15	69.37	5.82	25.35	1.84	AAXD 6/3 [44]	38	6.06	12.21	75.65	2.08	100	1.49
AFM1 [39]	69	5.5	14.68	76.60	0.23	16.52	0.23	AAXD 4/2 [44]	30	4.28	10.06	51.72	3.74	100	2.58
Mitchell [25]	51	4.81	13.21	66.03	3.77	11.11	3.77	Mitchell [25]	36	5.1	11.45	59.3	3.9	13	3.9
MBM [17]	64	5.16	13.36	71.54	2.60	8.59	0.09	INZeD [16]	47	6.12	14.53	81.03	2.93	9.53	0.02
LeAp-5[2] [54]	56	4.93	14.82	73.67	0.99	4.96	0.06	SAADI-EC [59]	103	12.8	17.50	214.50	2.37	8.82	1.92
SIMDive	57	5.08	13.72	74.72	0.82	4.76	0.05	SIMDive	44	5.33	12.24	67.60	0.77	5.2	0.01
16 × 16	Area (LUT)	Latency (ns)	Power (mW)	Energy (uJ)	ARE (%)	PRE (%)	Error Bias (%)	16/8	Area (LUT)	Latency (ns)	Power (mW)	Energy (uJ)	ARE (%)	PRE (%)	Error Bias (%)
Acc-IP NP [15]	287	7.04	45.20	311.90				Acc-IP NP [12]	169	21.67	24.11	537.6			
DRUM-6 [50]	233	5.34	42.59	232.24	1.47	6.31	0.04	AAXD 12/6 [44]	207	18.87	21.26	443.94	0.74	100	0.3
AFM1 [39]	261	7.32	37.98	257.20	1.34	17.80	1.34	AAXD 8/4 [44]	151	12.51	19.53	286.09	2.99	100	0.9
Mitchell [25]	187	4.9	34.15	177.2	3.85	11.11	3.85	Mitchell [25]	122	5.38	21.7	112.5	4.11	13	4.11
MBM [17]	204	6.59	35.92	214.76	2.63	8.83	0.09	INZeD [16]	165	6.28	23.41	145.05	2.93	9.54	0.02
LeAp-5 [54]	227	6.15	36.09	191.71	0.98	4.80	0.06	SAADI-EC [59]	362	25.7	29.24	890.29	2.14	8.82	1.76
SIMDive	216	5.94	34.82	184.24	0.82	4.90	0.05	SIMDive	143	5.56	23.03	123.90	0.78	5.2	0.01
32 × 32	Area (LUT)	Latency (ns)	Power (mW)	Energy (uJ)	ARE (%)	PRE (%)	Error Bias (%)	32/16	Area (LUT)	Latency (ns)	Power (mW)	Energy (uJ)	ARE (%)	PRE (%)	Error Bias (%)
Acc-IP NP [15]	1,037	9.81	114.32	1,065.4				Acc-IP NP [12]	597	46.14	33.08	1,479.9			
DRUM-6 [50]	616	6.35	77.60	356.8	1.53	5.88	0.05	AAXD 12/6 [44]	513	37.2	30.36	1,048.05	0.79	100	0.35
AFM1 [39]	995	10.76	87.94	815.9	2.88	22.40	2.88	AAXD 8/4 [44]	361	24.66	26.71	688.79	3.04	100	1.1
Mitchell [25]	484	5.96	57.1	334.3	3.91	11.11	3.91	Mitchell [25]	349	6.11	33.9	236.1	4.19	13	4.19
MBM [17]	533	7.51	60.14	447.6	2.69	8.74	0.10	INZeD [16]	422	8.15	36.56	305.47	2.96	9.47	0.03
LeAp-5 [54]	547	7.19	60.25	411.5	1.03	4.87	0.05	SAADI-EC [59]	822	51.6	42.61	2,143.33	2.33	9.04	1.85
SIMDive	521	6.88	58.32	362.0	0.91	4.72	0.05	SIMDive	381	6.34	39.94	246.66	0.81	5.16	0.02

[1]Peak Relative Error.

[2]Initial version of LeAp is a sequential design. For a fair comparison with other designs, in terms of the resource cost for its error-reduction scheme, it is implemented combinational-based in this manuscript.

Table 5. SIMDive Hybrid and SIMD Design Parameters (8-bit Hybrid and 16-8 SIMD Are Designed for This Article)

	Area (LUT)	Latency (ns)	Power (mW)	Energy (uJ)	ARE (%)	PRE (%)	Error Bias (%)		Area (LUT)	Latency (ns)	Power (mW)	Energy (uJ)	ARE (%)	PRE (%)	Error Bias (%)
8-bit Hybrid Mul/Div (SISD)	87	5.57	25.69	144.9	0.82	5.2	0.05	16-bit Hybrid Mul/Div (SISD)	261	5.84	57.33	235.9	0.82	5.2	0.05
SIMD Mul/Div (16/8 bit)	283	6.13	79.2	502.41				SIMD Mul/Div (32/16/8 bit)	1059	7.92	97.81	778.1			

First, it not only reduces the amount of computations, but also the latency of the application can be reduced, due to the shortened propagation delay of operations in the critical path. Second, precision scaling also positively contributes to reducing memory footprint and its associated data movement energy. Third, errors arising from neglecting LSBs will not result in noticeable decline in the final accuracy (detailed results for this on three studied applications are discussed in the next section).

6 RESULTS AND DISCUSSION

In this section, first, we present the implementation results of SIMDive SISD and SIMD configurations and compare them against SoA multipliers and dividers. Afterwards, the results of sensitivity analysis in three applications are presented, which provided inputs of the proposed approximation methodology. Finally the results of proposed cross-layer strategy versus baselines is assessed.

6.1 Evaluation and Characterization of SIMDive against SoA Multipliers and Dividers

We have evaluated SIMDive against five accurate and approximate cutting-edge multipliers and dividers. All the designs are developed in Verilog HDL, synthesized, and implemented in Xilinx Vivado 2019.2 on Virtex-7 FPGA. To ensure scalability of multipliers and dividers, they are implemented and compared for comprehensive precision of 8-, 16-, and 32-bit. Area and latency are collected from Vivado reports. Power and energy dissipation are scrutinized through simulations in **Xilinx Power Estimator (XPE)** over 100M inputs, uniformly distributed in a random order over whole input interval. Accuracy metrics: The behavioral structure of multipliers and dividers are developed in C++ to calculate their accuracy metrics. For 8- and 16-bit designs exhaustive test is applied. For 32-bit error characterization, 2^{32}, ~4.3B input pairs, uniformly distributed random over the whole 32-bit interval have been evaluated in Monte Carlo simulations.[6]

The circuit-level results, summarized in Tables 4 and 5, demonstrate that SoA logarithmic designs yield a better performance-accuracy tradeoff than other counterparts. This is due to transforming the 2D array-based structure of Mul/Div to 1D Add/Sub through Mitchell's algorithms. Specifically, although the gains from SIMDive designs is highly pronounced in 32- and 16-bit, the performance metrics of both SIMDive multiplier and divider are still superior than accurate IPs in 8-bit precision. In terms of accuracy, SIMDive designs render higher accuracy than other cutting-edge counterparts, including MBM w.r.t. both latency and energy. Particularly, in case of division, SIMDive reduces the drastically high latency of accurate divider, nearly to latency of same-size multiplier. Furthermore, the peak relative error in SIMDive divider is successfully bounded to 5.2%. Contrarily, there are a lot of cases with error near or equal to 100% in the truncated-based AAXD divider (that also increases with truncating more bits). Such high error cases can result in false positive peaks in heartbeat and corner detection. Also, as discussed, the unbiased errors of SIMDive can cancel out each other and minimize error accumulation in the aggregation-based (mostly Add/Mul) structure of bio-signal and image-processing kernels. Finally, recalling Section 4, boosting precision in our approach comes with a negligible cost, compared to the original Mitchell's design: One more LUT means adding one more bit to the error-coefficient, thereby reducing the original biased-error of Mitchell's multiplication/division. The effect of such LSB-truncation approach has already been discussed in detail, in REALM/MBM/INZeD studies. In terms of performance, the plasticity provided by SIMDive hybrid mode enables on-the-fly switching between two functionalities and without the need of reconfiguration. It is worth noting that the improvement obtained by our logarithmic designs are also reflected in SISD to SIMD transformation: For example, referring to Section 4, in SIMDive the detection of leading one is performed in parallel in each 4-bit segment of inputs, which is inherently suited for parallelization in a SIMD design. Interestingly, this transformation comes with a tolerable cost, as both SISD hybrid and SIMD modes (16- and 32-bit) are still smaller and faster than one accurate multiplier [15].

6.2 Application-level Sensitivity Analysis on Three Multi-kernel Applications

Figure 7 illustrates the inputs for sensitivity analysis and its generated outputs as the inputs for cross-layer design generation methodology. The conducted analysis is detailed in the following: Hardware implementation of applications: As illustrated in Figure 7, the source-code of applications is synthesized with Xilinx Vivado. Pan-Tompkins algorithm is adopted from Reference [92]. For JPEG and HCD, we have developed both in C++[7] and synthesized them using Vivado

[6]Simulated on Rack Server with Intel Xeon E5-2667 CPU @ 3.20 GHz and 512 GB RAM.

[7]The base of JPEG Compression is adopted from AxBench [104] and further optimized for a resource-efficient implementation on FPGA, e.g., by transforming 2D-DCT computations to the butterfly-based 1D-DCT approach [104, 105].

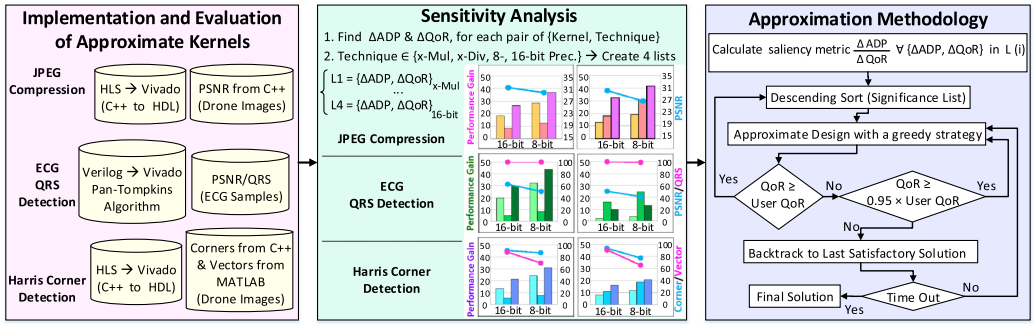


Fig. 7. Overall three-step flow for the proposed sensitivity analysis and cross-layer approximation methodology.

High-Level Synthesis (HLS) and disabled DSPs.[8] To efficiently reflect the optimizations of SIMDive in final HDL design, we have adopted a three-step approach to bridge this resource gap. In the first step, we have coded individual functions for each size of accurate multiplications and divisions in the application. This was for both accurate multiplier and divider functions (SISD) and also approximate SISD/SIMD designs (for SIMD configurations, respective operands are packed in the function based on Table 7). This approach has facilitated replacement of such SISD/SIMD modules with optimized SIMDive versions later on in step 3. Afterwards, using HLS inline pragma, we have forced the compiler to generate an independent HDL file for each of these functions. In the third step, the HDL description of the respective functions implementation that realizes the target SISD or SIMD operations (based on Table 7 of the article) are replaced by HDL-optimized versions of SIMDive modules. HLS has two main advantages: First, it enables the flexibility to apply user-defined directives (when needed) more easily in a system-level implementation and shortens the process of generating variety of customized configurations. Second, it facilitates the high-level behavioral evaluation of approximate designs.

Performance analysis of applications: The end-to-end performance gains shown in Figure 8 are measured after adjusting the approximation knobs in one kernel of the baseline accurate C++ code: The precision of data/operations in the kernel is reduced from 32-bit to the target one and accurate multipliers/dividers are substituted with SIMDive versions. For exact performance analysis, the HDL design of all accurate and approximate versions, generated by HLS, have been further passed to the downstream implementation phase, placed and routed on Virtex-7.

QoR analysis of applications: The evaluated application-level quality metrics are: QRS and **Peak Signal to Noise Ratio (PSNR)** for Pan-Tompkins, PSNR for JPEG, and percentage of correct vectors for HCD. In fact, for generating the motion/terrain vectors, similar to Reference [106], the extracted corners from HCD algorithm have been passed through MATLAB simulations.[9]

Benchmarks: The reports in Figure 8 are obtained by conducting the ECG analysis on real-world MIT-BIH database [100] in MATLAB. For JPEG and HCD, we have gauged their end-to-end accuracy alterations on 100 miscellaneous images from different aerial imagery datasets [101–103].

Cross-layer sensitivity analysis: Figures 8(a), (b), and (c) summarize the contribution of each kernel in the end-to-end LUT count, latency (Σ delay of consecutive kernels) of the application, and

[8]Using LUTs-based soft multipliers for low bit-widths operations has also been recommended by, e.g., Xilinx [13].
[9]In Reference [106], only DCT *adders* are approximated (DCT is used as a pre-processing compression step in studied UAV programs).

ADP,[10] when implemented in accurate mode. Recalling Section 3, *zigzag* and *Huffman* kernels in JPEG implement a re-arrangement and encoding scheme. To remain inline with industrial standards, we refrained from applying approximations in this process. In addition, the *corner selection via non-maximum suppression* also remained accurate in HCD, due to its moderately low resource footprint and the fact that it mainly comprises comparison operations. As can be seen, although most of existing works have focused on DCT kernel in JPEG (because of its multiplicative-structure and high area), quantization kernel contributes more significantly to the total application latency.

Targeting a high accuracy, computations can be carried out in floating point or 32-bit integer, in both bio-signal and image processing applications [107] Note, filters' coefficients and the intermediate results of kernels are considered in 32-bit integer format for baseline designs. The bit-width of the intermediate results can also be increased in consecutive kernels, even when input image is in 8-bit gray-scale. However, bounding the precision to 16- or 8-bit has also reported to be satisfactory in SoA studies [16, 17]. The intermediate results of each kernel are scaled before being used in the next kernel. Targeting an adaptable cross-layer approximation, in our analysis, we inspect the effects of kernels precision-scaling and imprecise multiplication and division on QoR-performance tradeoff, described as follows:

As the first step, we have deployed our SIMDive multiplier and divider, having 99.2% accuracy, in all 32-bit kernels. Afterwards, we have applied precision scaling one-kernel-at-a-time, while others are remained at 32-bit. The obtained *end-to-end* performance improvements and their respective QoR for each of these configurations have been reported in Figure 8. This figure, therefore, exhibits the sensitivity of kernels, individually, to approximations from different layers of abstraction (precision scaling atop of imprecise multiplication and division). The key observations of our analysis for the three applications are pin-pointed in the following:

Insight 1- Deploying SIMDive marginally affects the accuracy: We have observed replacing accurate multiplier and divider with SIMDive—having 99.2% accuracy—in all stages has marginally affected the final QoR, even when precision of kernels are tuned at 16-bit. Fluctuations in the visual quality of JPEG compressed image and the number of detected corners and correct vectors in HCD were also negligible. Part (b) in each of Figure 9 and Figure 10 (also Figure 14 and Figure 15 in Appendix) exhibit visual examples of such negligible changes by deployment of SIMDive (more examples can be found in the Appendix). The PSNR of 32-bit SIMDive-based ECG is also 45.8 (the tradeoff between QoR and ADP improvement in all configurations are shown in Figure 11). This negligible accuracy loss is due to two facts: First, the small error metric of SIMDive, i.e., ARE and PRE. In fact, the analysis of HCD application has shown that more false positive or false negative vectors cases will occur, provided that the approximate unit has a high peak error (e.g., 100% for AAXD dividers, shown in Table 4). The same also holds true for heartbeat (QRS) peaks, especially considering that the 100% error of division occurs at the last ECG stage. Second, our profiling has revealed that the near-zero biased errors of our multiplier and divider have been able to cancel out each other in consecutive kernels and prevent a drastic error accumulation in the aggregation-based (mostly Add/Mul) structure of kernels. Based on this observation, we have been able to deploy SIMDive in all kernels and thus the size of design space has been reduced to only changing the precision of kernels. In fact, our analysis exhibits that even 4-coefficient based SIMDive results in 100% detection in heartbeat or vectors generated by Harris corners. Nonetheless, we opted to increase the accuracy of Mul/Div (comes with the negligible cost of few more LUTs) and create opportunity for our adaptable precision-scaling.

Insight 2- Applications show more sensitivity to approximation of addition: We have observed that truncating the output of Add/Sub in kernels (when more than few bits), significantly

[10]Comprehensive energy calculation for multi-kernel programs on large ECG/image dataset is targeted for follow-up track.

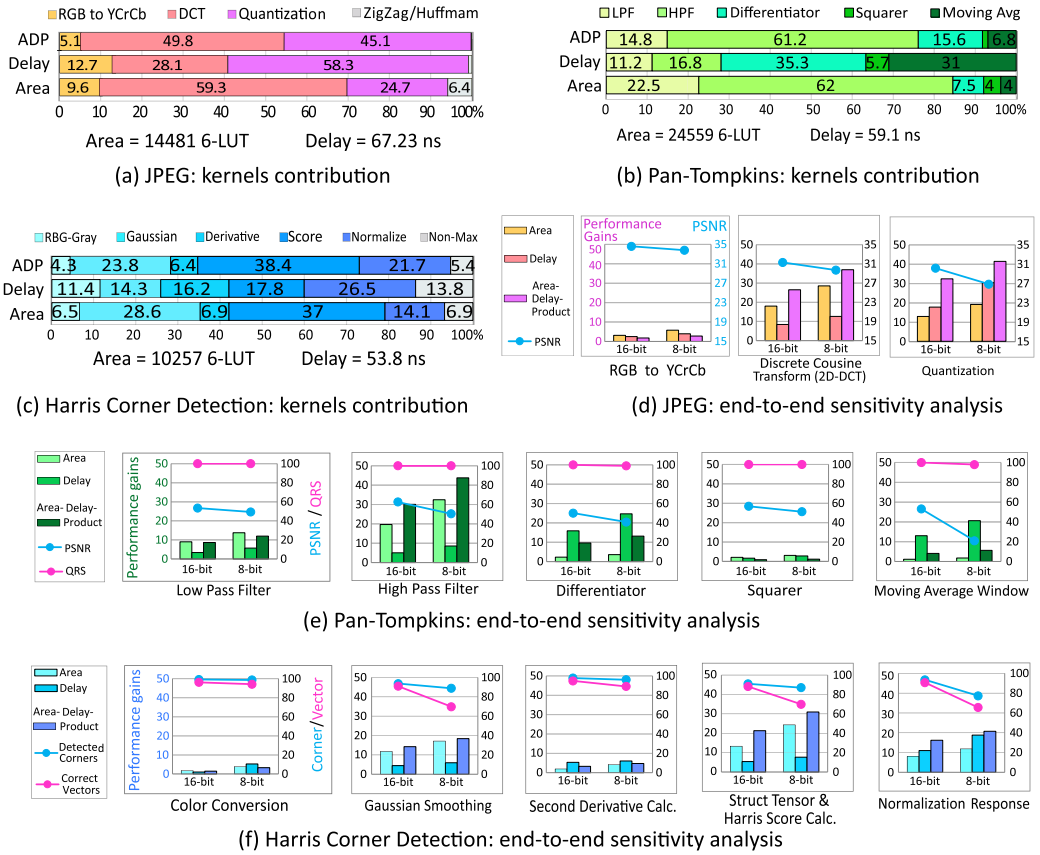
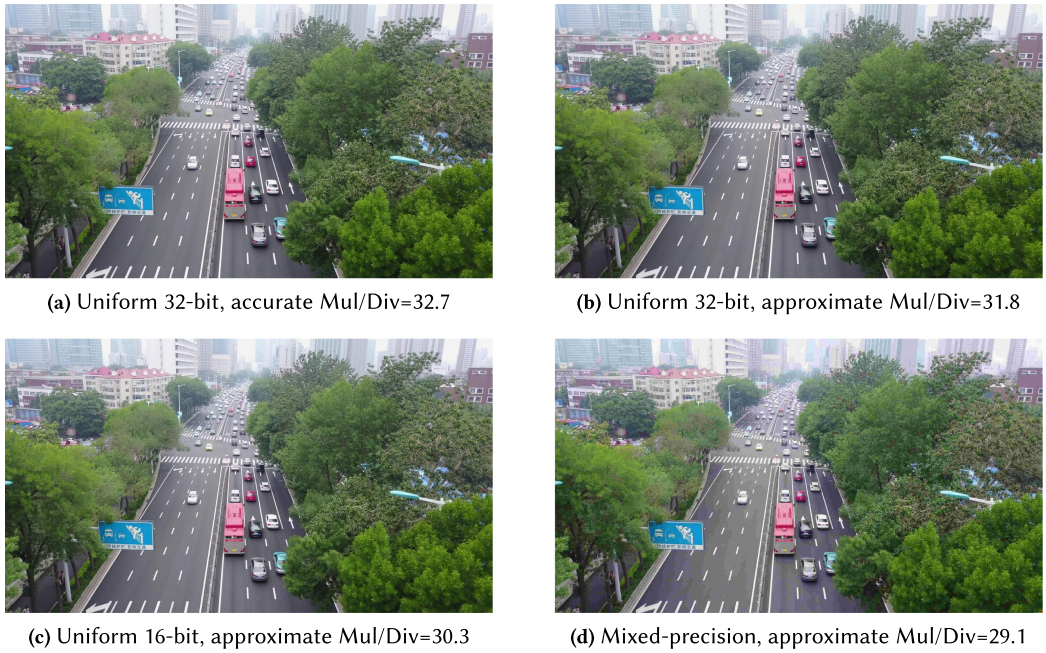


Fig. 8. Sensitivity analysis to cross-layer approximation in three applications. (a), (b), (c): contribution of each kernel in performance metrics of accurate implementation. (d), (e), (f): the tradeoff between end-to-end performance gains and QoR degradation after scaling down the precision of *one* kernel having SIMDive Mul/Div, while the rest are 32-bit accurate (32-bit integer with accurate operations). Area is in terms of LUT and delay in terms of ns.

affects the accuracy (so addition is kept accurate in the depicted figure). This happened especially in *differentiator* or *moving average window* stages of ECG application and *corner response calculation* stage of HCD. As discussed earlier, such observation are also endorsed by, e.g., Reference [31] for matrix multiplication-based applications.

Insight 3- Kernels contribute differently to each of performance metrics: Figures 8(d), (e), and (f) also reveal that kernels exhibit different behavior w.r.t. the gains from the same approximation: For example, when targeting LUT saving, approximation of HPF (second kernel) is more beneficial in ECG, while when detection latency improvement is the goal, differentiator should be approximated (because of its division, acting as speed bottleneck operation). Interestingly, as can be seen in Figure 8, although DCT has higher ADP than quantization in the original accurate configuration, by applying same precision scaling on both kernels, more energy saving is achieved via approximation of quantization. This is because, beside LUT, latency of quantization also significantly reduces (due to its divider). This further corroborates our strategy, that the knobs should be adjusted based on the *gained* improvement, not based on their primary order, in the non-approximated version.



(a) Uniform 32-bit, accurate Mul/Div=32.7

(b) Uniform 32-bit, approximate Mul/Div=31.8

(c) Uniform 16-bit, approximate Mul/Div=30.3

(d) Mixed-precision, approximate Mul/Div=29.1

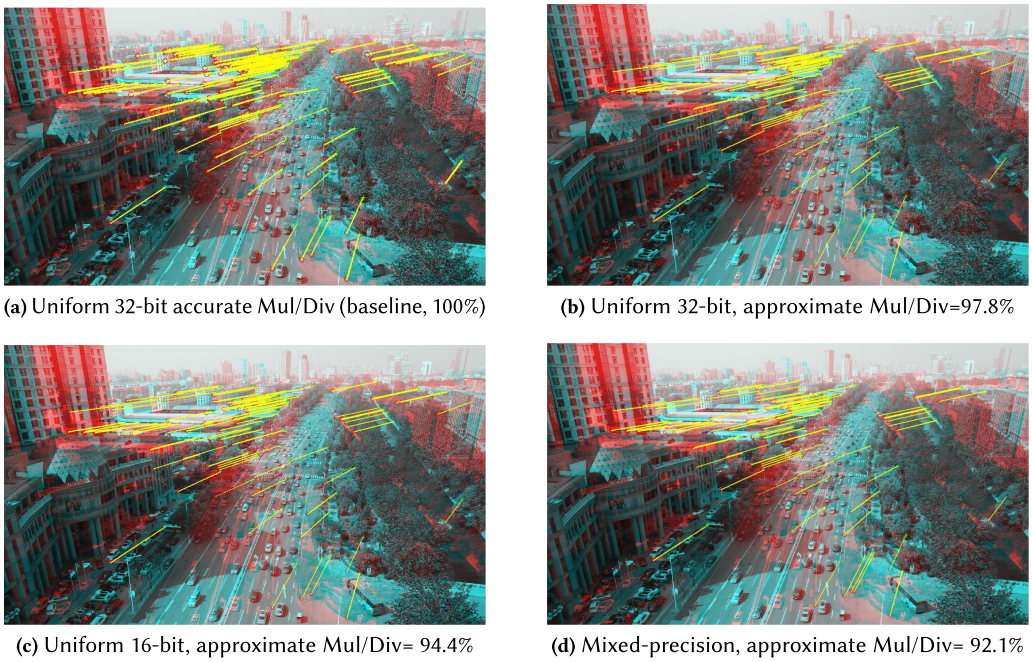
Fig. 9. JPEG compression for aerial images: The PSNR fluctuations are still tolerable by applying precision scaling and Mul/Div approximation.

Insight 4- Kernels significance order in lost QoR can differ from its order in the gained performance: While applying the same approximation results in a specific order for gained performance, the same/inverse order will not necessarily appear in QoR fluctuations. For example, referring to Figure 8(e), among Pan-Tompkins kernels, *HPF* enables the highest LUT saving and affects the ultimate PSNR the least. However, as shown in part (e), *Harris score* kernel not only enables the highest LUT saving, but also changes the final accuracy more than other down-scaled kernels. Hence, Insight-3 and Insight-4 endorse that: First, the relation between ΔPerformance and ΔQoR should be considered. Second, applying the most aggressive approximation on early stages (which is the adopted strategy in XBioSiP [92]) not only neglects order of stages w.r.t. their QoR sensitivity, but also disregards their order in the *obtained* performance improvement.

Conclusion: in short, aforementioned insights corroborate our strategy that considers the *relation* between the *gained performance* and *QoR loss*. Moreover, provided that the multi-kernel application has aggregation-based structure, adopting imprecise operations with minimally biased errors in all kernels can create the opportunity for applying a runtime adaptable precision scaling (efficiently supported herein, thorough on-the-fly configurable SIMDive units). The outputs of this sensitivity analysis, i.e., end-to-end performance gain and QoR loss, when each kernel is approximated by different techniques (imprecise multiplication, division and scaled down to 16- or 8-bit) are given in four lists to the cross-layer approximation methodology, discussed in Section 5.

6.3 Evaluating Mixed-precision Configurations and Proposed Heuristic on Multi-kernel Applications

Herein, the overall efficacy of the proposed cross-layer approximation methodology is assessed. Figure 11 details the obtained ADP improvement for different kernel configurations and



(a) Uniform 32-bit accurate Mul/Div (baseline, 100%)

(b) Uniform 32-bit, approximate Mul/Div=97.8%

(c) Uniform 16-bit, approximate Mul/Div= 94.4%

(d) Mixed-precision, approximate Mul/Div= 92.1%

Fig. 10. Tracking via Harris Corner Detection: The number of detected corners and vectors marginally changes with approximation. (Changes in Harris score range also enable detection of new vectors in the threshold-based selection of tracking algorithm in MATLAB. Average of false positive vectors is less than 4%.)

Table 6. Simulation Setup & Exhaustive Runtime for Three Applications (Kernels Mul/Div Replaced by SIMDive)

Pan-Tompkins (ECG)		JPEG Compression		Harris Corner Detection	
# Configurations	**Exhaustive Time**	**# Configurations**	**Exhaustive Time**	**# Configurations**	**Exhaustive Time**
Five kernels (32-, 16, or 8-bit) $3 \times 3 \times 3 \times 3 \times 3$ = 243 configs	11.7 hours in MATLAB (20k real-world ECG samples from MIT-BIH [100])	$3 \times 3 \times 3$ = 27 configs	5.2 hours in HLS (100 HD drone images [101–103])	$3 \times 3 \times 3 \times 3 \times 3 \times$ = 243 configs	62.6 hours: HLS (corners) + MATLAB (vectors) (100 drone images [101–103])

distinguishes the mixed-precision configurations found by the heuristic. For packing SIMD operations inside precision reduced kernels, dependency of multiplication and/or division operations has been considered (referring to Figures 2, 3, and 4). Empirical observations from sensitivity analysis, such as adopting the approximate multiplier and divider in all kernels and maintaining accurate structure of Add operation, has pruned the dominated points and efficiently reduced the size of design space that is evaluated in our case studies: As detailed in Table 6, simulating 243 possible ECG scenarios in MATLAB (each of five kernels can be adjusted to 8-, 16-, or 32-bit) takes a runtime of 11.7 hours when analyzing 20K real-world ECG samples of MIT-BIH database, sampled at the frequency of 200 Hz. Runtimes of 27 scenarios in JPEG and 243 scenarios in HCD (HLS generated corners processed in MATLAB-based code to produce terrain vectors) also take 5.2 and 62.6 hours over 100 image samples, respectively. The following inferences are highlighted on the functioning and efficacy of our cross-layer strategy:

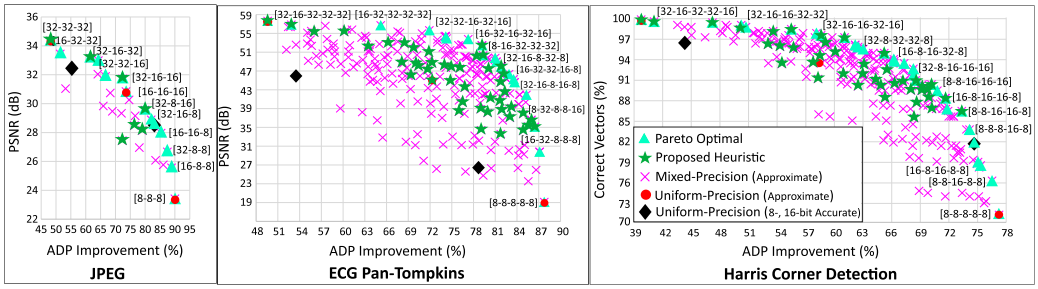


Fig. 11. *Approximate, mixed-precision* configs generate accuracy-performance tradeoffs with similar/higher gains than *accurate-uniform*. The heuristic also finds many of Pareto/near-Pareto points. Configs found in 20% of exhaustive time are distinguished by green stars (QoR thresholds: JPEG 28 dB, ECG 35 dB, HCD 90%).

- It can be deduced from Figure 11 that not only new resource-accuracy tradeoff levels are generated by *mixed-precision* configurations, but also they can enable higher savings than the *uniform* counterparts in similar accuracy-level. Note that there is not only one unique, but a set of configurations with permissible resource-accuracy tradeoffs.
- Although the proposed design generation methodology is a heuristic and may not always find all Pareto-optimal solutions, for the three case studies it has been able to generate most of Pareto/near-Pareto points for applications within the first ~10%–30% of their execution time (the points generated in first 20% of exhaustive time are shown in green stars in Figure 11). This asserts the effectiveness of the heuristic and its notable saving in exploration time, especially when brute-force exploration is not time-wise tractable for a larger design space. Note, Figure 11 shows how the heuristic proceeds toward the Pareto points (when the user-QoR threshold is 70%).
- $\frac{\Delta ADP}{\Delta QoR}$ is customizable w.r.t. the design constraints and objectives. For example, on the similar basis of a greedy heuristic, the work in the domain of NNs [29, 63] mostly targets memory traffic reduction and applies *layer-wise* quantization. Herein, we opted ADP, as in an FPGA platform, smaller ADP can better reflect the resource savings. Nevertheless, other performance metrics can also be explored based on designer's goal.
- This flexibility provided by our adaptable approximation strategy also can be exploited in a runtime energy-management technique: The bitstream of arbitrary heuristic-generated configurations or Pareto-optimal configurations (provided that exhaustive search is feasible) can also be stored in the FPGA SPI/BPI flash memories [108] to be loaded at runtime for various accuracy-performance tradeoffs.

6.4 End-to-end Performance Gains of Heuristic-generated Configurations on Multi-kernel Applications

In this section, the end-to-end QoR-performance tradeoff for five approaches are evaluated on Pan-Tompkins, JPEG compression, and HCD applications. Approximation approaches comprise the two proposed: uniform 16-bit and mixed-precision SIMDive based configurations, found by the proposed $\frac{\Delta ADP}{\Delta QoR}$ based heuristic, compared against two baselines, i.e., uniform 32- and 16-bit accurate counterparts and mixed-precision SIMDive-based configurations, found by SoA ΔQoR-based heuristic (this heuristic is adopted in References [29, 63]). In this article, we have adhered to PSNR of 35 and 28 dB for ECG and JPEG compression and 90% correct vectors for

Table 7. Kernels' SISD/SIMD Structure in Three Applications, w.r.t. the Final Approximate Configuration

Pan Tompkins (ECG)		JPEG Compression		Harris Corner Detection	
Kernel Name	**Configuration**	**Kernel Name**	**Configuration**	**Kernel Name**	**Configuration**
Low-Pass Filter	SIMD 8–16 bit	RGB to YCbCr	SIMD 8–16 bit	RGB to Grayscale	SIMD 8–16 bit
High-Pass Filter	SIMD 8–16 bit	2D-DCT	SIMD 8–16 bit	Gaussian Smoothing	SIMD 8–16 bit
Differentiator	SISD 16 bit	Quantization	SISD 16 bit	Derivative Sobel	SIMD 8–16 bit
Squarer	SIMD 8–16 bit			Tensor & Score Response	SISD 16 bit
Moving Avg Filter	SISD 16 bit			Normalization	SISD 16 bit

HCD application (which is reported as an acceptable confidence level for moving object tracking [106]). In fact, for the Pan-Tompkins case study, although the work of XBioSip [92] has allowed PSNR of 19.2 dB (which also satisfies QRS threshold of 100%), herein, we maintain a high PSNR value, i.e., 35 dB that is also expected to deliver 100% detection accuracy [92]. For these quality thresholds, the following configurations are found by the proposed heuristic: [8-8-16-8-16] for five-kernel ECG Pan-Tompkins, [8-8-16] for three-kernel JPEG Compression, and [8-8-8-16-16] for five-kernel HCD. Accordingly, the adopted SISD/SIMD structure of SIMDive for the proposed kernel configurations is detailed in Table 7.[11] The end-to-end performance gain from such heuristic-generated configurations are evaluated against the baselines in Figure 12. The results detailed in this figure demonstrate that not only the mixed-precision configurations enable higher improvement over uniform 16-bit precision counterpart, but also the proposed $\frac{\Delta ADP}{\Delta QoR}$-based heuristic outperforms the ΔQoR based approach [29, 63] in ECG and HCD applications.

Finally, the two greedy strategies are appraised on various QoR thresholds in all applications, the results of which are presented in Figure 13. It should be noted that the functionality and thereby, the quality of final configuration found by heuristic approaches depends on different factors, including kernel structure, exploration time, accuracy-performance tradeoff of the approximate components, and so on. Nevertheless, as detailed in Figure 13, we have observed that the proposed heuristic based on $\frac{\Delta ADP}{\Delta QoR}$ saliency metric can find solutions with higher performance improvement when the accuracy threshold is moderate to relatively high. On the contrary, when the ultimate quality threshold is relaxed, ΔQoR-based heuristic takes more steps with smaller quality changes (which also increased the heuristic exploration time) and gradually approaches to a solution with slightly higher gains, up to 8%. It should be highlighted that to fairly assess the quality of solution found by ΔQoR-based heuristic against the ones found by our proposed approach (shown in Figure 13), we also enabled the same 1%–5% freedom for both approaches and started the heuristic after efficiently pruning the design space by the proposed sensitivity analysis. Comparing the search complexity of various heuristics such as Reference [66], for which the original size of design space is different, is planned for a future track of this extension manuscript.

Discussion: it should be highlighted that although more complicated object tracking or heart arrhythmia programs utilize machine-learning techniques for feature extraction, continuously outsourcing the complete video stream (for the former) or patient bio-signal data (for the latter) to the insecure/untrustworthy network or process/store it on the third-party cloud can pose performance bottleneck for a real-time processing and deplete the battery in a short interval. Therefore, enabling extraction of some features at the edge is highly desired.

[11] The reported gains are w.r.t. the *computations*, without considering the overhead of read+pack/write+unpack. Addressing further architectural details is left for future work.

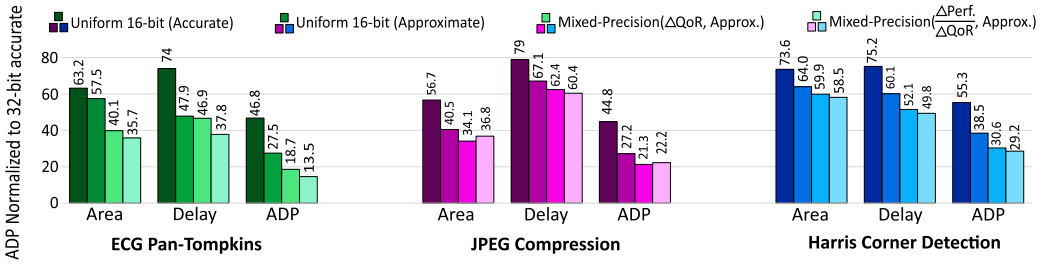


Fig. 12. End-to-end performance gains of different strategies, compared to 32-bit accurate (uniform) configuration: overall, mixed-precision configurations, generated by the proposed $\frac{\Delta ADP}{\Delta QoR}$-based heuristic achieves higher gain than the ΔQoR-based heuristics [29, 63].

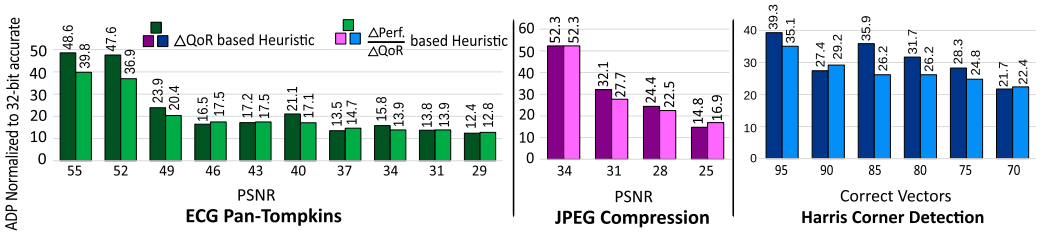


Fig. 13. Comparing $\frac{\Delta ADP}{\Delta QoR}$-based versus ΔQoR-based heuristic in various QoR levels: The former approach has generated configurations with higher ADP improvement when the accuracy threshold is high.

7 FUTURE WORKS AND CONCLUSION

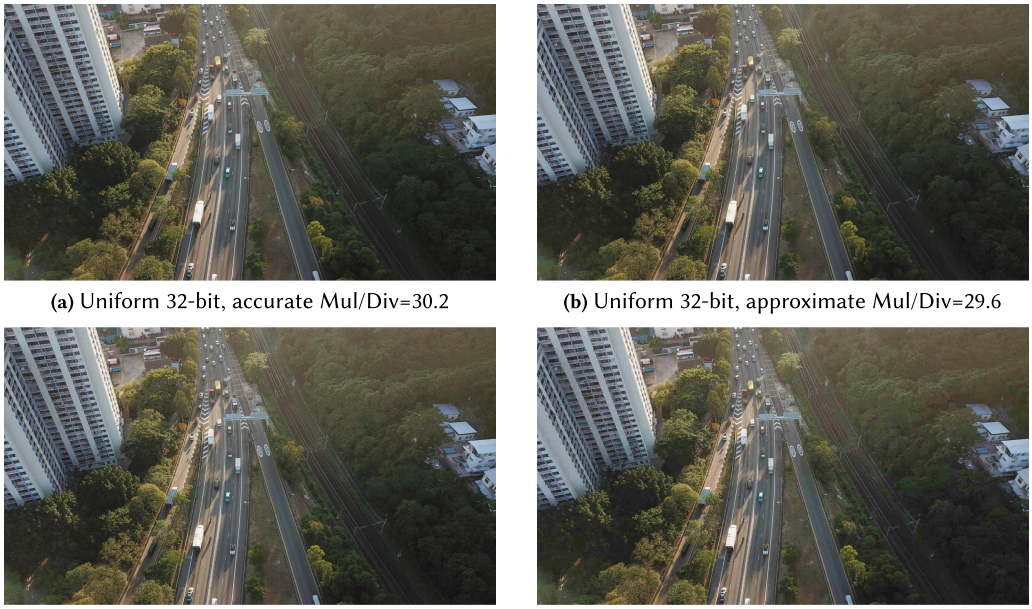
In this article, we proposed *Plasticine*, a cross-layer approximation methodology for multi-kernel application, which efficiently utilizes the synergistic effects of a chain of techniques across layers of abstraction. The chain of approximations is also effectively enabled in a three-tiered cross-layer hierarchy through the plasticity of SIMD multiplier-dividers, each of which can support precision variability along with hybrid functionality. To this end, we first proposed an end-to-end sensitivity analysis that finds the most tolerable degree of approximations for each kernel. Such data were used afterwards in our proposed heuristic, which tailors the precision at successive kernels in such a way to maximize performance gain, while also satisfying the user-defined accuracy in the ultimate QoR. Our heuristic-generated Pareto or near-Pareto mixed-precision configurations—evaluated on three application domains—not only have enabled various performance-accuracy tradeoffs, but also provided higher savings versus uniform-precision counterparts in different accuracy-levels.

For future track, we intend to further expand the applicability of SIMDive and Plasticine cross-layer methodology for layer-wise approximation of NNs. Especially, the SIMD mode of SIMDive unit are interesting candidates for implementation of convolution layers (required to be explored in detail). In addition, we intend to design an approximate SIMD **Arithmetic Logic Unit (ALU)** and assess its applicability in data-path of soft processors such as RISC-V, in which a significant part of execution unit is dedicated to **floating point (FP)** units [109]. Particularly, in FP multiplier, more than 90% of total area and power is consumed by mantissa multiplier [17], where the position of leading-one is fixed at MSB. Therefore, a lightweight FP version of our logarithmic multiplier, in which the leading one detection unit is omitted, can result in notable resource savings.

Furthermore, it should be noted that the selection of optimal precision for each operation in a large application is a non-trivial challenge for approximate computing community. This open

research question can be divided into intra- and inter-kernel granularity. Specifically, optimally adjusting the approximation knobs within each kernel requires a detailed error propagation behavior analysis in a probabilistic manner [66] (as the propagation of error depends on the kernel's control and data-flow pattern and errors of consecutive operations can be amplified or dampened through the downstream computations, depending on the kernel structure). Therefore, an interesting track could be generalizing Plasticine methodology to also enable the efficient heterogeneity of operaation precision in an intra-kernel granularity.

APPENDIX



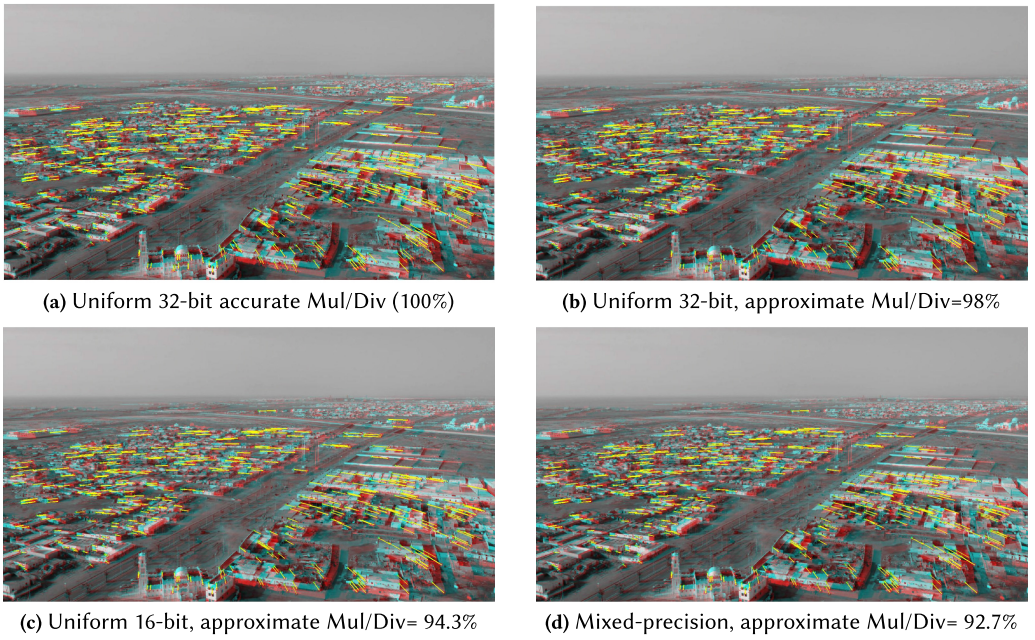
(a) Uniform 32-bit, accurate Mul/Div=30.2

(b) Uniform 32-bit, approximate Mul/Div=29.6

(c) Uniform 16-bit, approximate Mul/Div=28.3

(d) Mixed-precision, approximate Mul/Div=27.4

Fig. 14. PSNR in JPEG application: Visual quality of image is gracefully maintained after approximations.



(a) Uniform 32-bit accurate Mul/Div (100%)

(b) Uniform 32-bit, approximate Mul/Div=98%

(c) Uniform 16-bit, approximate Mul/Div= 94.3%

(d) Mixed-precision, approximate Mul/Div= 92.7%

Fig. 15. Corners and vector detected via Harris Corner Detection application: As can be seen even with approximation, the direction of the movement has been found by many vectors.

REFERENCES

[1] World Health Organisation. 2018. Cardiovascular diseases (CVDs). Retrieved from https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds).

[2] P. Kostic. 2017. Heart Disease and Early Heart Attack Care. Retrieved from https://www.bnl.gov/hr/occmed/hpp/linkable_files/pdf/EarlyHeartAttackSymptoms.pdf.

[3] Y. Yang, C. Wang, L. Gong, and X. Zhou. 2019. FPNet: Customized convolutional neural network for FPGA platforms. In *IEEE International Conference on Field-Programmable Technology (ICFPT)*

[4] X. Gu, Y. Zhu, Sh. Zhou, Ch. Wang, M. Qiu, and G. Wang. 2016. A real-time FPGA-based accelerator for ECG analysis and diagnosis using association-rule mining. *ACM Trans. Embed. Comput. Syst.* 15, 2 (2016).

[5] H. K. Chatterjee, M. Mitra, and R. Gupta. 2015. Real–time detection of electrocardiogram wave features using template matching and implementation in FPGA. *Int. J. Biomed. Eng. Technol.* 17, 3 (2015).

[6] S. Ullah, S. Rehman, M. Shafique, and A. Kumar. 2021. High-performance accurate and approximate multipliers for FPGA-based hardware accelerators. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* (2021).

[7] S. Ullah, S. Rehman, B. S. Prabakaran, F. Kriebel, M. A. Hanif, M. Shafique, and A. Kumar. 2018. Area-optimized low-latency approximate multipliers for FPGA-based hardware accelerators. In *IEEE/ACM Design Automation Conference (DAC)*.

[8] I. Kuon and J. Rose. 2007. Measuring the gap between FPGAs and ASICs. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 26, 2 (2007).

[9] A. Boutros, S. Yazdanshenas, and V. Betz. 2018. Embracing diversity: Enhanced DSP blocks for low-precision deep learning on FPGAs. In *IEEE International Conference on Field Programmable Logic and Applications (FPL)*.

[10] S. Lee, D. Kim, D. Nguyen, and J. Lee. 2019. Double MAC on a DSP: Boosting the performance of convolutional neural networks on FPGAs. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 38, 5 (2019).

[11] Xilinx. 2015. LogiCORE IP multiplier v12.0. Retrieved from https://www.xilinx.com/support/documentation/ip_documentation/mult_gen/v12_0/pg108-mult-gen.pdf.

[12] Xilinx. 2016. LogiCORE IP Divider v5.1. Retrieved from https://www.xilinx.com/support/documentation/ip_documentation/div_gen/v5_1/pg151-div-gen.pdf.

[13] Xilinx. 2018. 7 Series DSP48E1 Slice. Retrieved from https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf.

[14] N. Van Toan and J. Lee. 2020. FPGA-based multi-level approximate multipliers for high-performance error-resilient applications. *IEEE Access* 8 (2020).
[15] Z. Ebrahimi, S. Ullah, and A. Kumar. 2020. SIMDive: Approximate SIMD Soft multiplier-divider for FPGAs with tunable accuracy. In *ACM Great Lakes Symposium on VLSI (GLSVLSI)*.
[16] H. Saadat, H. Javaid, and S. Parameswaran. 2019. Approximate integer and floating-point dividers with near-zero error bias. In *IEEE/ACM Design Automation Conference (DAC)*.
[17] H. Saadat, H. Bokhari, and S. Parameswaran. 2018. Minimally biased multipliers for approximate integer and floating-point multiplication. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 37, 11 (2018).
[18] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos. 2016. Proteus: Exploiting numerical precision variability in deep neural networks. In *International Conference on Supercomputing (ICS)*.
[19] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel. 2016. Cross-layer approximate computing: From logic to architectures. In *IEEE/ACM Design Automation Conference (DAC)*.
[20] V. Mrazek, Z. Vasicek, L. Sekanina, M. A. Hanif, and M. Shafique. 2019. ALWANN: Automatic layer-wise approximation of deep neural network accelerators without retraining. In *IEEE/ACM International Conference on Computer-aided Design (ICCAD)*.
[21] M. Langhammer, S. Gribok, and G. Baeckler. 2020. High density pipelined 8bit multiplier systolic arrays for FPGA. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*.
[22] M. Langhammer, B. Pasca, G. Baeckler, and S. Gribok. 2019. Extracting INT8 multipliers from INT18 multipliers. In *IEEE International Conference on Field Programmable Logic and Applications (FPL)*.
[23] Y. Fu, E. Wu, A. Sirasao, S. Attia, K. Khan, and R. Wittig. 2017. Deep learning with INT8 optimization on Xilinx devices. *White Paper* (2017). https://www.xilinx.com/support/documentation/white_papers/wp486-deep-learning-int8.pdf.
[24] G. Zervakis, H. Saadat, H. Amrouch, A. Gerstlauer, S. Parameswaran, and J. Henkel. 2021. Approximate computing for ML: State-of-the-art, challenges and visions. In *Asia & South Pacific Design Automation Conference (ASP-DAC)*.
[25] J. N. Mitchell. 1962. Computer multiplication and division using binary logarithms. *IRE Trans. Electron. Comput.* 11, 4 (Aug. 1962).
[26] A. R. Baranwal, S. Ullah, S. S. Sahoo, and A. Kumar. 2020. ReLAccS: A multi-level approach to accelerator design for reinforcement learning on FPGA-based systems. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 40, 9 (2020).
[27] H. Saadat, H. Javaid, A. Ignjatovic, and S. Parameswaran. 2020. REALM: Reduced-error approximate log-based integer multiplier. In *Design, Automation & Test in Europe Conference (DATE)*.
[28] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han. 2020. Improving the accuracy and hardware efficiency of neural networks using approximate multipliers. *IEEE Trans. Very Large Scale Integ. Syst.* 28, 2 (2020).
[29] Z. G. Tasoulas, G. Zervakis, I. Anagnostopoulos, H. Amrouch, and J. Henkel. 2020. Weight-oriented approximation for energy-efficient neural network inference accelerators. *IEEE Trans. Circ. Syst. I (TCAS-I): Reg. Papers* 67, 12 (2020).
[30] J. Pan and W. J. Tompkins. 1985. A real-time QRS detection algorithm. *IEEE Trans. Biomed. Eng.* 32, 3 (1985).
[31] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han. 2020. Approximate arithmetic circuits: A survey, characterization, and recent applications. *Proc. IEEE* 108, 12 (2020).
[32] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, J. Henkel, and J. Henkel. 2016. Architectural-space exploration of approximate multipliers. In *IEEE/ACM International Conference on Computer-aided Design (ICCAD)*.
[33] A. G. M. Strollo, E. Napoli, D. De Caro, N. Petra, and G. D. Meo. 2020. Comparison and extension of approximate 4-2 compressors for low-power approximate multipliers. *IEEE Trans. Circ. Syst. I (TCAS-I): Reg. Papers* 67, 9 (2020).
[34] O. Akbari, M. Kamal, A. Afzali-Kusha, and M. Pedram. 2017. Dual-quality 4:2 Compressors for utilizing in dynamic accuracy configurable multipliers. *IEEE Trans. Very Large Scale Integ. Syst.* 25, 4 (2017).
[35] S. Venkatachalam and S. Ko. 2017. Design of power and area efficient approximate multipliers. *IEEE Trans. Very Large Scale Integ. Syst.* 25, 5 (2017).
[36] D. Esposito, A. G. M. Strollo, E. Napoli, D. De Caro, and N. Petra. 2018. Approximate multipliers based on new approximate compressors. *IEEE Trans. Circ. Syst. I (TCAS-I): Reg. Papers* 65, 12 (2018).
[37] M. Wang, Y. Luo, M. An, Y. Qiu, M. Zheng, Z. Wang, and H. Pan. 2020. An optimized compression strategy for compressor-based approximate multiplier. In *IEEE International Symposium on Circuits and Systems (ISCAS)*.
[38] P. J. Edavoor, S. Raveendran, and A. D. Rahulkar. 2020. Approximate multiplier design using novel dual-stage 4:2 compressors. *IEEE Access* 8 (2020).
[39] Y. Guo, H. Sun, and S. Kimura. 2020. Small-area and low-power FPGA-based multipliers using approximate elementary modules. In *Asia & South Pacific Design Automation Conference (ASP-DAC)*.
[40] E. Adams, S. Venkatachalam, and S. Ko. 2020. Approximate restoring dividers using inexact cells and estimation from partial remainders. *IEEE Trans. Comput.* 69, 4 (2020).
[41] S. Venkatachalam, E. Adams, and S. Ko. 2019. Design of approximate restoring dividers. In *IEEE International Symposium on Circuits and Systems (ISCAS)*.

[42] L. Chen, J. Han, W. Liu, P. Montuschi, and F. Lombardi. 2018. Design, evaluation and application of approximate high-radix dividers. *IEEE Trans. Multi-scale Comput. Syst.* 4, 3 (2018).
[43] L. Chen, J. Han, W. Liu, and F. Lombardi. 2016. On the design of approximate restoring dividers for error-tolerant applications. *IEEE Trans. Comput.* 65, 8 (2016).
[44] H. Jiang, L. Liu, F. Lombardi, and J. Han. 2019. Low-power unsigned divider and square root circuit designs using adaptive approximation. *IEEE Trans. Comput.* 68, 11 (2019).
[45] H. Jiang, L. Liu, F. Lombardi, and J. Han. 2018. Adaptive approximation in arithmetic circuits: A low-power unsigned divider design. In *Design, Automation & Test in Europe Conference (DATE)*.
[46] M. Vaeztourshizi, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. 2018. An energy-efficient, yet highly-accurate, approximate non-iterative divider. In *International Symposium on Low Power Electronics and Design (ISLPED)*.
[47] S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram, and Z. Navabi. 2017. TruncApp: A truncation-based approximate divider for energy efficient DSP applications. In *Design, Automation & Test in Europe Conference (DATE)*.
[48] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram. 2019. TOSAM: An energy-efficient truncation- and rounding-based scalable approximate multiplier. *IEEE Trans. Very Large Scale Integ. Syst.* 27, 5 (2019).
[49] F. Frustaci, S. Perri, P. Corsonello, and M. Alioto. 2020. Approximate multipliers with dynamic truncation for energy reduction via graceful quality degradation. *IEEE Trans. Circ. Syst. II (TCAS-II): Expr. Briefs* 67, 12 (2020).
[50] S. Hashemi, R. I. Bahar, and S. Reda. 2015. DRUM: A dynamic range unbiased multiplier for approximate applications. In *IEEE/ACM International Conference on Computer-aided Design (ICCAD)*.
[51] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram. 2017. RoBA Multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing. *IEEE Trans. Very Large Scale Integ. Syst.* 25, 2 (2017).
[52] S. Hashemi, R. I. Bahar, and S. Reda. 2016. A low-power dynamic divider for approximate applications. In *IEEE/ACM Design Automation Conference (DAC)*.
[53] S. Ullah, S. S. Murthy, and A. Kumar. 2018. SMApproxLib: Library of FPGA-based approximate multipliers. In *IEEE/ACM Design Automation Conference (DAC)*.
[54] Z. Ebrahimi, S. Ullah, and A. Kumar. 2020. LeAp: Leading-one detection-based softcore approximate multipliers with tunable accuracy. In *Asia & South Pacific Design Automation Conference (ASP-DAC)*.
[55] M. Imani, R. Garcia, A. Huang, and T. Rosing. 2019. CADE: Configurable approximate divider for energy efficiency. In *Design, Automation, & Test in Europe Conference (DATE)*.
[56] M. S. Ansari, H. Jiang, B. F. Cockburn, and J. Han. 2018. Low-power approximate multipliers using encoded partial products and approximate compressors. *IEEE J. Emerg. Select. Topics Circ. Syst.* 8, 3 (2018).
[57] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina. 2017. EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *Design, Automation & Test in Europe Conference (DATE)*.
[58] R. Zendegani, M. Kamal, A. Fayyazi, A. Afzali-Kusha, S. Safari, and M. Pedram. 2016. SEERAD: A high speed yet energy-efficient rounding-based approximate divider. In *Design, Automation & Test in Europe Conference (DATE)*.
[59] J. Melchert, S. Behroozi, J. Li, and Y. Kim. 2019. SAADI-EC: A quality-configurable approximate divider for energy efficiency. *IEEE Trans. Very Large Scale Integ. Syst.* 27, 11 (2019).
[60] S. Behroozi, J. Li, J. Melchert, and Y. Kim. 2019. SAADI: A scalable accuracy approximate divider for dynamic energy-quality scaling. In *Asia & South Pacific Design Automation Conference (ASP-DAC)*.
[61] M. S. Ansari, B. F. Cockburn, and J. Han. 2019. A hardware-efficient logarithmic multiplier with improved accuracy. In *Design, Automation & Test in Europe Conference (DATE)*.
[62] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi. 2018. Design and evaluation of approximate logarithmic multipliers for low power error-tolerant applications. *IEEE Trans. Circ. Syst. I (TCAS-I): Reg. Papers* 65, 9 (2018).
[63] P. Judd, J. Albericio, T. Hetherington, T. Aamodt, N. E. Jerger, R. Urtasun, and A. Moshovos. 2015. Reduced-precision strategies for bounded memory in deep neural nets. *arXiv preprint* (2015).
[64] S. Sarwar, G. Srinivasan, B. Han, P. Wijesinghe, A. Jaiswal, P. Panda, A. Raghunathan, and K. Roy. 2018. Energy efficient neural computing: A study of cross-layer approximations. *IEEE J. Emerg. Select. Topics Circ. Syst* 8, 4 (Dec. 2018).
[65] M. A. Hanif, A. Marchisio, T. Arif, R. Hafiz, S. Rehman, and M. Shafique. 2018. X-DNNs: Systematic cross-layer approximations for energy-efficient deep neural networks. *J. Low Power Electron.* 14, 4 (2018).
[66] S. Lee, L. K. John, and A. Gerstlauer. 2017. High-level synthesis of approximate hardware under joint precision and voltage scaling. In *Design, Automation, & Test in Europe Conference (DATE)*.
[67] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan. 2013. Analysis and characterization of inherent application resilience for approximate computing. In *ACM/IEEE Design Automation Conference (DAC)*.

[68] G. A. Gillani and A. B. J. Kokkeler. 2017. Improving error resilience analysis methodology of iterative workloads for approximate computing. In *ACM Computing Frontiers Conference (CF)*.
[69] P. Roy, R. Ray, C. Wang, and W. F. Wong. 2014. ASAC: Automatic sensitivity analysis for approximate computing. *ACM Conference on Languages, Compilers and Tools for Embedded Systems (LCTEC)* 49, 5 (2014).
[70] P. Roy, J. Wang, and W. F. Wong. 2015. PAC: Program analysis for approximation-aware compilation. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*.
[71] M. A. Hanif, R. Hafiz, and M. Shafique. 2018. Error resilience analysis for systematically employing approximate computing in convolutional neural networks. In *Design, Automation, & Test in Europe Conference (DATE)*.
[72] C. Schorn, A. Guntoro, and G. Ascheid. 2018. Accurate neuron resilience prediction for a flexible reliability management in neural network accelerators. In *Design, Automation, & Test in Europe Conference (DATE)*.
[73] G. Montavon, W. Samek, and K. Müller. 2018. Methods for interpreting and understanding deep neural networks. *Dig. Sig. Process.* 73 (2018).
[74] C. Wu, W. Shan, and J. Xu. 2019. Dynamic adaptation of approximate bit-width for CNNs based on quantitative error resilience. In *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*
[75] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K. R. Müller. 2017. Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recog.* 65 (2017).
[76] V. Mrazek, L. Sekanina, and Z. Vasicek. 2020. Libraries of approximate circuits: Automated design and application in CNN Accelerators. *IEEE J. Emerg. Select. Topics Circ. Syst* 10, 4 (2020).
[77] L. Sekanina, Z. Vasicek, and V. Mrazek. 2019. *Automated Search-based Functional Approximation for Digital Circuits.* Springer International Publishing.
[78] K. Nepal, S. Hashemi, H. Tann, R. I. Bahar, and S. Reda. 2019. Automated high-level generation of low-power approximate computing circuits. *IEEE Trans. Emerg. Topics Comput.* 7, 1 (2019).
[79] J. Huang, J. Lach, and G. Robins. 2012. A methodology for energy-quality trade-off using imprecise hardware. In *IEEE/ACM Design Automation Conference (DAC)*.
[80] S. Lee, D. Lee, K. Han, E. Shriver, L. K. John, and A. Gerstlauer. 2016. Statistical quality modeling of approximate hardware. In *International Symposium on Quality Electronic Design (ISQED)*.
[81] L. Witschen, M. Awais, H. Gh, T. Wiersema, and M. Platzner. 2019. CIRCA: Towards a modular and extensible framework for approximate circuit generation. *Microelectron. Reliab.* 99 (2019).
[82] A. H. El Moussawi and S. Derrien. 2017. Superword level parallelism aware word length optimization. In *Design, Automation, & Test in Europe Conference (DATE)*.
[83] V. P. Ha, T. Yuki, and O. Sentieys. 2020. Towards generic and scalable word-length optimization. In *Design, Automation, & Test in Europe Conference (DATE)*.
[84] W. Liu, F. Lombardi, and M. Shulte. 2020. A retrospective and prospective view of approximate computing. *Proc. IEEE* 108, 3 (2020).
[85] C. De la Parra, A. Guntoro, and A. Kumar. 2020. ProxSim: GPU-based simulation framework for cross-layer approximate DNN optimization. In *Design, Automation & Test in Europe Conference (DATE)*.
[86] C. De la Parra, A. Guntoro, and A. Kumar. 2020. Improving approximate neural networks for perception tasks through specialized optimization. *Fut. Gen. Comput. Syst.* 113 (2020).
[87] A. Bosio, D. Menard, and O. Sentieys. 2019. A comprehensive analysis of approximate computing techniques: From component-to application-level. In *Design, Automation & Test in Europe Conference (DATE)*.
[88] S. Venkataramani, X. Sun, N. Wang, C. Chen, J. Choi, M. Kang, A. Agarwal, J. Oh, S. Jain, T. Babinsky, N. Cao, T. Fox, B. Fleischer, G. Gristede, M. Guillorn, H. Haynie, H. Inoue, K. Ishizaki, M. Klaiber, S. Lo, G. Maier, S.a Mueller, M. Scheuermann, E. Ogawa, M. Schaal et al. 2020. Efficient AI system design with cross-layer approximate computing. *Proc. IEEE* 108, 12 (2020).
[89] S. Venkataramani, V. Srinivasan, W. Wang, S. Sen, J. Zhang, A. Agrawal, M. Kar, S. Jain, A. Mannari, Hoang Tran et al. 2021. RaPiD: AI accelerator for ultra-low precision training and inference. *IEEE International Symposium on Circuits and Systems (ISCAS)*.
[90] P. Yin, C. Wang, W. Liu, and F. Lombardi. 2018. Design of dynamic range approximate logarithmic multipliers. In *ACM Great Lakes Symposium on VLSI (GLSVLSI)*.
[91] Y. Fan, Xiaoxi Wu, Jiying Dong, and Zhi Qi. 2019. AxDNN: Towards cross-layer design of approximate DNNs. In *Asia & South Pacific Design Automation Conference (ASP-DAC)*.
[92] B. Prabakaran, S. Rehman, and M. Shafique. 2019. XBioSiP: A methodology for approximate bio-signal processing at the edge. In *IEEE/ACM Design Automation Conference (DAC)*.
[93] S. Basu, L. Duch, R. Braojos, G. Ansaloni, L. Pozzi, and D. Atienza. 2017. An inexact ultra-low power bio-signal processing architecture with lightweight error recovery. *ACM Trans. Embed. Comput. Syst.* 16, 5s (2017).
[94] K. Gao, S. Yao, H. AliAkbarpour, S. Agarwal, G. Seetharaman, and K. Palaniappan. 2019. Sensitivity of multiview 3D point cloud reconstruction to compression quality and image feature detectability. In *IEEE Applied Imagery and Pattern Recognition Workshop (AIPR)*.

[95] C. Harris and M. Stephens. 1988. A combined corner and edge detector. In *Alvey Vision Conference (AVC)*, Vol. 15.
[96] J. Sánchez, Nelson Monzón, and Agustín Salgado. 2018. An analysis and implementation of the Harris corner detector. *Image Process. On Line* 8 (2018).
[97] P. Ashenden. 2010. *The Designer's Guide to VHDL*. Morgan Kaufmann.
[98] B. Parhami. 2010. *Computer Arithmetic: Algorithms and Hardware Designs*. Vol. 20. Oxford University Press.
[99] Xilinx. 2013. Xilinx 7 Series FPGA Programmable Guide for HDL Designs. Retrieved from https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/7series_hdl.pdf.
[100] A. L. Goldberger et al. 2000. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation* 101, 23 (2000).
[101] M. Mueller, N. Smith, and B. Ghanem. 2016. A benchmark and simulator for UAV tracking. In *European Conference on Computer Vision (ECCV)*.
[102] H. Fan, Longyin Wen, Dawei Du, Pengfei Zhu, Qinghua Hu, Haibin Ling, Mubarak Shah, Biao Wang, Bin Dong, Di Yuan, Dong Wang, Dongjie Zhou, Haoyang Sun, Hossein Ghanei-Yakhdan, Huchuan Lu, Javad Khaghani, Jinghao Zhou, Keyang Wang, Lei Pang, Lei Zhang, Li Cheng, Liting Lin, Lu Ding, Nana Fan, Peng Wang, Penghao Zhang, Ruiyan Ma, Seyed Mojtaba Marvasti-Zadeh, Shohreh Kasaei, Shuhao Chen, Simiao Lai, Tianyang Xu, Wentao He, Xiaojun Wu, Xin Hou, Xuefeng Zhu, Yanjie Gao, Yanyun Zhao, Yong Wang, Yong Xu, Yubo Sun, Yuting Yang, Yuxuan Li, Zezhou Wang, Zhenwei He, Zhenyu He, Zhipeng Luo, Zhongjian Huang, Zhongzhou Zhang, Zikai Zhang, and Zitong Yi. 2020. VisDrone-SOT2020: The vision meets drone single object tracking challenge results. In *Workshops in European Conference on Computer Vision (ECCV)*.
[103] Y. Lyu, G. Vosselman, G. Xi, A. Yilmaz, and M. Ying Yang. 2020. UAVid: A semantic segmentation dataset for UAV imagery. *ISPRS J. Photogram. Rem. Sens.* 165 (2020).
[104] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran. 2017. AxBench: A multiplatform benchmark suite for approximate computing. *IEEE Des. Test* 34, 2 (2017).
[105] M. Jridi, Y. Ouerhani, and A. Alfalou. 2013. Low complexity DCT engine for image and video compression. In *Real-time Image and Video Processing (RTIVP)*, Vol. 8656.
[106] T. Nomani, M. Mohsin, Z. Pervaiz, and M. Shafique. 2020. xUAVs: Towards efficient approximate computing for UAVs—Low power approximate adders with single LUT Delay for FPGA-based aerial imaging optimization. *IEEE Access* 8 (2020).
[107] SmartCardia. 2018. 32-bit ARM Cortex-M3 inside SmartCardia-INYU. https://smartcardia.com/.
[108] Xilinx. 2018. 7 Series FPGAs Configuration User Guide (UG470). https://www.xilinx.com/support/documentation/user_guides/ug470_7Series_Config.pdf.
[109] S. Tamimi, Z. Ebrahimi, B. Khaleghi, and H. Asadi. 2019. An efficient SRAM-based reconfigurable architecture for embedded processors. *IEEE Trans. Comput.-aided Des. Integ. Circ. Syst.* 38, 3 (2019).
[110] Z. Ebrahimi and A. Kumar. 2021. BioCare: An energy-efficient CGRA for bio-signal processing at the edge. In *International Symposium on Circuits and Systems (ISCAS)*.

Received March 2021; revised August 2021; accepted August 2021