



A Kernel-Aware Framework for Energy-Efficient FPGA Edge Detection Using XNOR–popcount and Selective Approximate Arithmetic

Van-Khoa Pham¹

ABSTRACT

Edge-vision systems on resource-constrained platforms require low-latency and energy-efficient front-end processing. Conventional gradient-based edge operators continue to rely on multiply–accumulate (MAC) operations, which can increase logic utilization, switching activity, and power consumption in field-programmable gate array (FPGA) implementations. This study introduces a kernel-aware arithmetic-selection framework, supported by synthesized evidence, for energy-efficient FPGA-based edge detection. The framework combines XNOR–popcount-based arithmetic mapping with selective approximate computation. Its central design principle is to replace MAC operations with exclusive-NOR–population count (XNOR–popcount) when the kernel structure is suitable for binary-friendly arithmetic, and to apply approximation only to the remaining adder-dominated stages when a complete XNOR–popcount mapping is not practical. Under this rule, Prewitt-like operators are mapped to an XNOR–popcount datapath over their active nonzero taps. In contrast, Sobel-like operators are realized using a hybrid datapath that combines binary matching, shift-add processing, and approximate accumulation. The resulting framework shows that multiplier removal is the dominant source of hardware savings, while approximate arithmetic provides a controlled secondary optimization. Overall, the proposed approach establishes a structured design methodology for low-power FPGA edge-detection architectures on embedded platforms.

Article information:

Keywords: Approximate Computing, Edge Detection, Energy-efficient Hardware, Field-programmable Gate Arrays, Kernel-aware Arithmetic, Low-power Design, XNOR–popcount

Article history:

Received: September 19, 2025

Revised: February 5, 2026

Accepted: April 23, 2026

Published: April 30, 2026

(Online)

DOI: 10.37936/ecti-cit.2026202.263898

1. INTRODUCTION

Recent advances in edge intelligence have shifted many vision workloads from cloud servers to embedded platforms. This shift is driven by latency, privacy, bandwidth, and power constraints [1]–[3]. At the same time, the energy cost of computation has become a primary design concern in modern accelerators [4]–[6].

To address these constraints, prior work has explored pruning, quantization, and hardware-aware architectures for efficient inference [7]–[9]. These efforts indicate that algorithmic efficiency and hardware efficiency must be considered jointly. However, low-level vision front ends remain expensive on resource-constrained hardware. Gradient-based operators repeatedly perform local multiplication and accumula-

tion, thereby increasing logic depth, switching activity, and implementation cost. This issue is particularly important on FPGA-based edge devices, where area, power, memory bandwidth, and arithmetic resources are limited.

This study addresses the above challenge by introducing a kernel-aware framework for FPGA-based edge detection. The idea is to employ XNOR–popcount computation when the kernel structure supports a binary-friendly arithmetic mapping, and to use approximate arithmetic only in the remaining multi-bit stages where a complete XNOR–popcount mapping is not practical. In this method, the framework combines structural arithmetic simplification with controlled error-tolerant optimization.

Different edge operators, however, require differ-

¹The author is with the Faculty of Advanced Education, Ho Chi Minh City University of Technology and Engineering, Ho Chi Minh City 700000, Vietnam, Email: khoapv@hcmute.edu.vn

ent arithmetic treatments. Prewitt-like kernels contain active, nonzero coefficients restricted to ± 1 , while zero-valued taps can be masked out or omitted from the XNOR-popcount computation. This structure makes them well-suited to multiplier-free implementation. In contrast, Sobel-like kernels contain weighted ± 2 terms, which make a complete binary mapping less direct. These observations motivate a general design principle: arithmetic optimization should be guided by kernel structure. Multiplier elimination should be prioritized when the active kernel taps support XNOR-popcount reformulation, whereas approximate arithmetic should be applied selectively to residual multi-bit stages that continue to dominate hardware cost.

Accordingly, this work is positioned as a kernel-aware arithmetic-selection framework and evidence synthesis, rather than as a single end-to-end accelerator implemented uniformly across all abstraction levels. It consolidates three complementary evidence streams: circuit-level results demonstrating the efficiency of XNOR-popcount as an arithmetic primitive [10], FPGA module-level results showing that Prewitt filtering can be efficiently mapped to XNOR-popcount computation [11], and system-level results showing that Sobel filtering benefits from selective approximate accumulation [12]. These results are used to derive a unified arithmetic-selection rule for energy-efficient FPGA edge detection.

The contributions of this study are threefold. First, it proposes a kernel-aware framework that combines XNOR-popcount-based multiplier elimination with selective approximate arithmetic. Second, it maps Prewitt-like kernels to an XNOR-popcount datapath over active nonzero taps, while mapping Sobel-like kernels to a hybrid datapath that combines binary matching, shift-add logic, and approximate accumulation. Third, it synthesizes evidence across circuit, module, and system levels to derive a practical design rule for low-power FPGA edge-detection architectures.

The remainder of this paper is organized as follows. Section II reviews related work and motivates the design rule. Section III presents the framework and implementation flow. Section IV discusses the results for the Prewitt and Sobel branches. Section V concludes the paper.

2. RELATED WORK

Edge computing enables low-latency processing by moving computation closer to the data source [1], [2]. Nevertheless, image filtering and feature extraction remain computationally demanding, particularly when local convolution windows are evaluated using multi-bit multiply-accumulate (MAC) structures [3]–[6]. On embedded FPGA platforms, this arithmetic cost is further constrained by limited logic resources, digital signal processing (DSP) slices, and

power budgets. Therefore, arithmetic simplification remains a key requirement for practical edge-vision deployment.

Hardware-aware model design has reduced computational complexity in many inference workloads [7]–[9]. However, these advances do not directly eliminate the arithmetic cost of classical edge operators. Prewitt and Sobel operators remain attractive for embedded vision because they are simple, deterministic, and well-suited to streaming hardware architectures. Their main challenge is therefore not algorithmic relevance, but efficient arithmetic implementation.

Binary neural computation has demonstrated that a substantial portion of the multiplication cost can be removed when operands are represented in binary or bipolar form. XNOR-Net replaced conventional multiplication with bitwise XNOR operations followed by population counting [13]. Binarized neural networks extended this concept by constraining weights and activations to ± 1 [14]. Subsequent studies further developed this direction through efficient inference pipelines, edge-oriented implementations, approximate popcount schemes, and dedicated accelerators [15]–[18]. This body of work is relevant to the present study for two reasons. First, it shows that XNOR-popcount computation can substantially reduce arithmetic complexity without requiring conventional multiplier arrays. Second, it suggests that this principle can be extended beyond neural inference when the target operator has a binary-friendly structure. These observations motivate the use of XNOR-popcount as a general low-cost arithmetic primitive in this work.

Edge detection remains a fundamental stage in many computer vision pipelines because it captures structural boundaries and local intensity variations with relatively low computational complexity. In learning-based vision systems, deeper convolutional representations have been shown to improve edge localization and contour characterization [19]. Studies on effective receptive fields further indicate that multi-layer convolutional networks can extract boundary information across multiple spatial scales [20]. These findings confirm the continuing importance of edge information in both classical and learning-based vision frameworks. However, learning-based edge detectors are often unsuitable for resource-constrained embedded platforms that require deterministic latency, compact hardware realization, and low power consumption. Under these constraints, classical edge operators remain highly relevant due to their algorithmic transparency, predictable resource requirements, and efficient implementation in streaming FPGA architectures.

Approximate computing is a well-established approach for energy-efficient design in workloads that can tolerate bounded numerical error [21]. In image filtering and edge detection, approximate adders

can improve the power–performance–quality trade-off when applied carefully [22]. Comparative studies on Sobel edge detection have shown that moderate approximation can preserve acceptable output quality while reducing logic complexity [23]. Hybrid approximate adders have also been proposed to reduce carry-propagation complexity at the cost of bounded errors in the least significant bits [24]. A key insight from this literature is that approximation should be applied selectively rather than uniformly. In particular, arithmetic inaccuracy should be introduced only in stages where its impact on visual quality is limited, and the corresponding hardware benefit is meaningful. This consideration is especially important for Sobel-based filtering, where the adder tree and gradient-magnitude computation remain major contributors to implementation cost.

FPGA implementations of edge detection are attractive because of their streaming-friendly dataflow, deterministic timing, and deep pipelining capability [25]. Recent studies have reported practical FPGA implementations of Sobel, Prewitt, and related edge operators [26]–[27]. These works confirm that gradient-based filtering maps naturally to line buffers, sliding windows, and regular pipelined

datapaths. However, many existing FPGA edge-detection designs still rely on conventional multi-bit arithmetic. Some works optimize exact arithmetic units, whereas others investigate approximate arithmetic in isolation or focus primarily on deployment without reconsidering the underlying computation model. This separation indicates a methodological gap: arithmetic optimization should first consider whether the kernel structure permits multiplier-free XNOR–popcount mapping, and only then apply approximation to the residual arithmetic stages that continue to dominate cost.

The literature therefore shows progress in efficient edge computation, binary arithmetic, approximate arithmetic, and FPGA deployment, but these directions have generally been studied separately. What remains lacking is a unified framework that first determines whether a kernel can be reformulated into a binary-friendly XNOR–popcount structure and only then applies approximation to the residual arithmetic that continues to dominate cost. Recent studies at three complementary levels support this need. At the circuit level, XNOR–popcount provides the same computational result as MAC with much lower transistor count, power, and delay [10]. At the FPGA

Table 1: Summary of Arithmetic-Level Optimization Strategies for Hardware-Efficient Edge Detection.

Ref.	Design Level	Arithmetic Strategy	Quantitative Evidence	Design Insight
[10]	Circuit-level arithmetic primitive.	Multiplier-based MAC accumulation is replaced by XNOR–popcount computation under binary/bipolar data representation.	Transistor count is reduced from 2376 to 1244, corresponding to a 47.6% reduction. Power consumption decreases from 122 to 38 μ W, corresponding to a 68.9% reduction. Delay is reduced from 581 to 293 ps, corresponding to a 49.6% reduction.	XNOR–popcount is an efficient, low-cost arithmetic primitive when the target computation can be reformulated into binary or bipolar form.
[11]	FPGA-level Prewitt edge detector.	The Prewitt operator is reformulated as an XNOR–popcount datapath by exploiting its active nonzero ± 1 kernel coefficients. At the same time, zero-valued taps are masked out or omitted from the binary computation.	Look-up table (LUT) usage decreases from 379 to 172, corresponding to a 54.6% reduction. Flip-flop usage decreases from 166 to 123, corresponding to a 25.9% reduction. DSP usage is eliminated from 2 to 0. Dynamic power is reduced by approximately 60%, with operation around 250 MHz.	For Prewitt-like kernels whose active nonzero taps are sign-only ± 1 coefficients, an XNOR–popcount architecture with zero-tap masking can eliminate DSP usage and substantially reduce FPGA resource cost.
[12]	System-level Sobel edge detector.	Selective approximate carry-lookahead adders are introduced in the Sobel accumulation path to trade arithmetic precision for energy and throughput efficiency.	The 3-bit approximate design achieves 357.14 frames per second (FPS), 2.8 ms latency, 0.164 W power consumption, 22.43 dB peak signal-to-noise ratio (PSNR), and 0.9532 structural similarity index measure (SSIM).	For mixed-coefficient kernels such as Sobel, selective approximate arithmetic is preferable to full binarization because it improves throughput and energy efficiency with limited hardware overhead while preserving acceptable image quality.

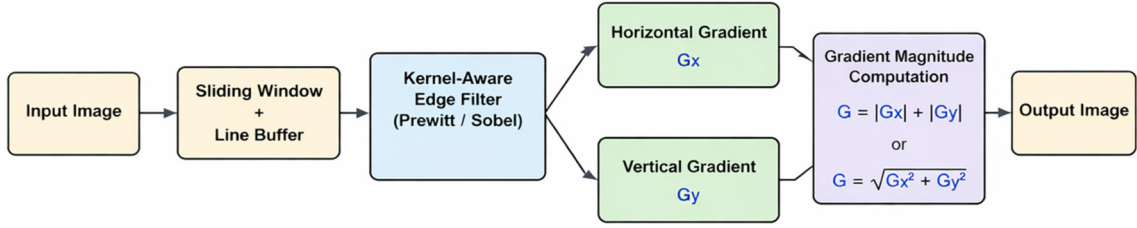


Fig.1: Kernel-aware FPGA edge-detection pipeline. The input image is streamed through a line-buffer/sliding-window stage and processed by a configurable Prewitt/Sobel operator.

module level, this reformulation is especially effective for Prewitt filtering, where the active nonzero coefficients are restricted to ± 1 , and zero-valued taps can be masked out, enabling efficient binary mapping and complete DSP elimination [11]. At the system level, Sobel filtering benefits from selective approximate carry-lookahead addition in the lower-significance accumulation path [12].

This gap motivates the framework proposed in this work. The framework first exploits binary reformulation when the kernel is binary-friendly and then applies approximation only to the residual arithmetic that still dominates hardware cost. This design rule is summarized in Table 1 and developed in the following section.

3. KERNEL-AWARE ARITHMETIC-SELECTION FRAMEWORK AND IMPLEMENTATION

This section defines the kernel-aware arithmetic-selection framework and describes how the corresponding FPGA datapaths are selected for different kernel classes. The quantitative evidence used to motivate and support the design rule is drawn from the circuit-, module-, and system-level studies summarized in Table 1. The central design principle is straightforward: XNOR-popcount arithmetic is employed when binary reformulation is naturally supported by the kernel structure, whereas approximate arithmetic is introduced only when full binary mapping is not practical. This prioritization is important because multiplier elimination typically provides larger hardware savings than localized approximation alone [10]–[12].

Table 1 summarizes the cross-layer evidence drawn from prior studies and the corresponding design rules adopted in this work. At the arithmetic-primitive level, XNOR-popcount computation can reproduce MAC-equivalent operations with reduced complexity, power, and delay [10]. At the FPGA module level, Prewitt filtering maps naturally to this primitive because its active nonzero coefficients are restricted to ± 1 , while zero-valued taps do not contribute to the inner product and can therefore be masked out. At the system level, Sobel filtering requires a different treatment because its weighted ± 2 terms make a com-

plete binary mapping less direct. For this case, a hybrid datapath with selective approximation provides a more suitable trade-off between efficiency and output fidelity [12].

Figure 1 shows the overall streaming FPGA edge-detection pipeline. The input image is first passed through a line-buffer and sliding-window subsystem, which generates a 3×3 neighborhood at each processing cycle. The resulting windowed stream is then fed to a configurable edge-operator core that supports both Prewitt and Sobel operators through a unified interface. The horizontal and vertical gradients, G_x and G_y , are subsequently combined in a gradient-magnitude stage, followed by threshold-based edge classification. This pipeline preserves the regular dataflow required for low-latency FPGA execution and provides a common top-level structure for both operator branches. Although the arithmetic style changes inside the compute core according to kernel structure, buffering, streaming, and output logic remain shared. This common organization also simplifies deployment on Zynq-based systems using Advanced eXtensible Interface (AXI) and direct memory access (DMA) [28]–[30].

Figure 2 details the proposed kernel-aware operator mapping. For Prewitt, the active nonzero ± 1 coefficients allow direct binary reformulation, while zero-valued taps are masked out or omitted from the XNOR-popcount datapath. The 3×3 window is encoded, the active taps are processed by XNOR comparison, popcount, and sign recovery, and explicit multipliers are thereby eliminated [11]. For Sobel, the ± 1 terms remain compatible with binary-style processing, but the ± 2 terms require weighted treatment. The Sobel datapath therefore integrates XNOR-based matching where applicable, shift-add processing for the ± 2 coefficients, and approximate carry-lookahead addition in the residual accumulation stage [12]. This mapping is the main methodological contribution of the study because it shows that arithmetic restructuring should follow operator structure. The framework neither forces a pure binary model onto Sobel nor retains full MAC-based arithmetic for Prewitt. Instead, it assigns each kernel the least costly arithmetic realization that still preserves acceptable output quality and streaming FPGA execution.

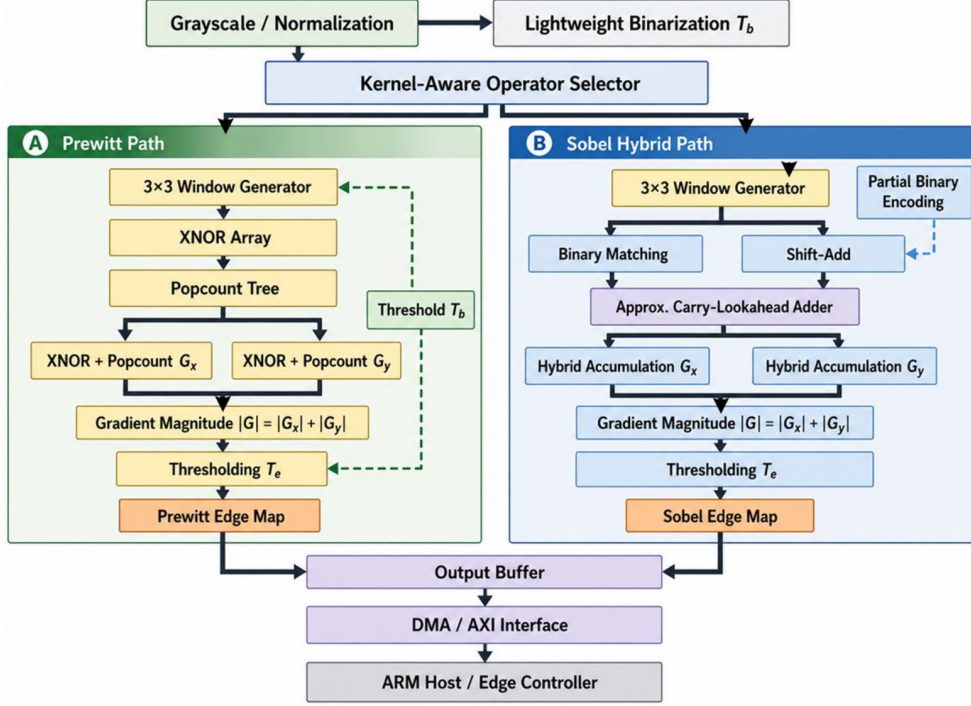


Fig.2: Architecture of the kernel-aware edge detection accelerator.

Figure 3 presents the XNOR–popcount primitive used in the binary computation path. The primitive compares binary inputs and encoded kernel values using XNOR logic, counts the number of matches with a popcount tree, and reconstructs the bipolar convolution response using lightweight shift and subtraction logic [10]. Specifically, for a kernel with N binary elements and p XNOR matches, the bipolar inner product can be recovered as $2p-N$. Therefore, the datapath does not require a general-purpose multiplier. This replaces the conventional multiplier-based accumulation stage with simpler logic. Compared with a multiplier-based MAC design, the XNOR–popcount implementation reported in [10] reduces transistor count from 2376 to 1244, power from $122 \mu\text{W}$ to $38 \mu\text{W}$, and delay from 581 ps to 293 ps. These results support the use of XNOR–popcount as the arithmetic foundation of the Prewitt branch and as a partial building block in the hybrid Sobel branch.

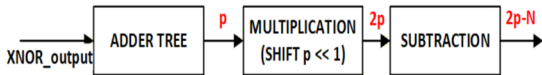


Fig.3: XNOR–popcount primitive and sign-recovery datapath for binary convolution.

Figure 4 highlights the distinction between the input binarization threshold T_b and the edge-decision threshold T_e . Specifically, T_b is applied before binary processing to convert grayscale pixels into a 1-bit representation suitable for XNOR–popcount evaluation. In contrast, T_e is applied after gradient computation to determine whether a pixel is classified as an edge in

the final edge map. These thresholds therefore serve fundamentally different purposes and should not be treated interchangeably. The binarization threshold T_b may be selected using image statistics, such as the intensity histogram, mean intensity, median intensity, or an automatic thresholding criterion. If an automatic thresholding method is used, the method should be specified and cited; otherwise, T_b should be reported as an experimental parameter. This interpretation is consistent with the histogram-guided rationale adopted in the approximate Sobel design, where the input intensity distribution is analyzed to estimate tolerance to arithmetic error [12]. Accordingly, thresholding in the framework is not merely a post-processing operation; it also acts as a mechanism that links input characteristics to arithmetic selection.

Figure 5 illustrates the deployment flow of the proposed accelerator on the PYNQ-Z2 platform. The design is initially specified in C/C++ and synthesized using Vitis high-level synthesis (HLS), through which the high-level description is translated into a hardware intellectual property (IP) core. The generated IP is subsequently imported into the Vivado environment for synthesis, implementation, and bitstream generation, together with AXI-based system integration. At the platform level, the accelerator is deployed on the Zynq device, where the ARM Cortex-A processing system coordinates execution and communicates with the FPGA fabric. The top-level hardware architecture includes an image-control module, an XNOR–popcount/MAC processing block,

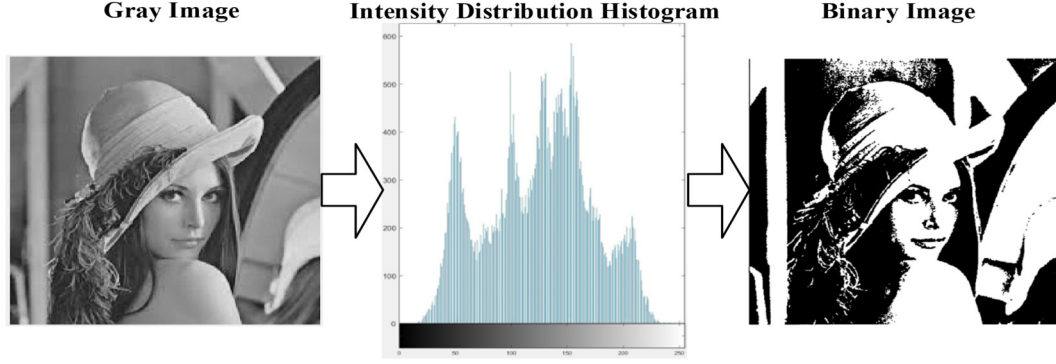


Fig.4: Separation of the binarization threshold T_b and the edge-decision threshold T_e . The input image is first binarized for XNOR-popcount processing, and the resulting gradient magnitude is then thresholded to generate the final edge map.

and an output buffer, interconnected through streaming handshaking signals for input data, validity, readiness, and output transfer. This deployment flow demonstrates that the framework is not confined to an isolated arithmetic module, but can be realized as a complete hardware-software image-processing system on a practical PYNQ-Z2 platform. Moreover, the integration methodology can accommodate different operator instantiations within the proposed kernel-aware framework.

4. RESULTS AND DISCUSSION

This section synthesizes the reported behavior of two representative edge-operator classes under the proposed kernel-aware arithmetic-selection rule. The first is the Prewitt operator, which is amenable to XNOR-popcount reformulation over active nonzero taps. The second is the Sobel operator, which requires selective approximation in the residual accumulation path. This organization reflects the central premise of the study, namely, that arithmetic

simplification should be applied in a kernel-aware manner. Figure 6 compares the edge maps generated by an FPGA implementation based on conventional MAC operations with those produced by an FPGA implementation using XNOR-popcount [11]. The XNOR-popcount implementation preserves the dominant edge structures with only minor deviations from the software reference. For the MNIST and ORL datasets reported in [11], the extracted contours remain clean and mostly one-pixel wide in the digit images, while the main object boundaries and salient facial features are retained in the face images. The differences introduced by the XNOR-popcount datapath are limited mainly to weak textures and noisy edge components, which may appear thinner or be suppressed. Overall, the qualitative results show that XNOR-popcount reformulation can substantially reduce hardware complexity while maintaining the principal edge information in the evaluated images.

Table 2 summarizes the hardware resource utilization of the Prewitt core under MAC arithmetic and

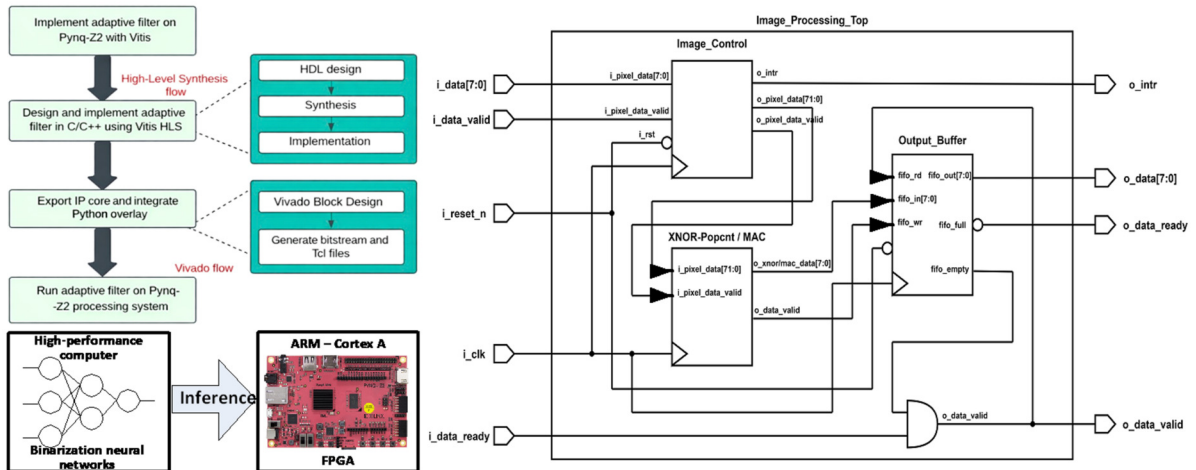


Fig.5: Hardware design flow and top-level FPGA architecture of the accelerator implemented on PYNQ-Z2 using Vitis HLS and Vivado.

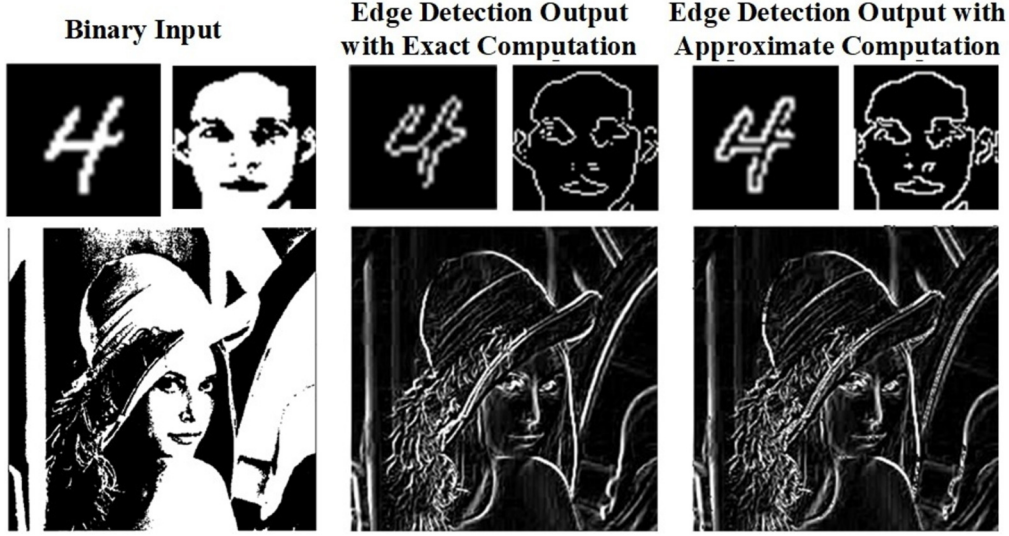


Fig. 6: Qualitative Prewitt edge maps generated by the Exact Computation-based FPGA baseline and the Approximate Computation-based FPGA implementation. The comparison illustrates the effect of binary reformulation on dominant edge preservation under the evaluation setting.

XNOR-popcount arithmetic [11]. The MAC-based method uses 379 LUTs, 166 flip-flops (FFs), and 2 DSP slices, whereas the XNOR-popcount version uses 172 LUTs, 123 FFs, and no DSP slices. This corresponds to approximately 55% fewer LUTs and 26% fewer FFs, together with complete DSP elimination. These savings arise from changing the computation model rather than from minor circuit tuning. Multi-bit multiplication is replaced by XNOR logic, popcount structures, and lightweight sign handling. DSP elimination is particularly important on resource-constrained FPGAs, where dedicated arithmetic blocks are limited. Table 2 therefore indicates that multiplier elimination is the dominant first-order optimization for binary-friendly operators.

Table 2: FPGA resource utilization of Prewitt edge-detection cores implemented with MAC and XNOR-popcount arithmetic [11].

Hardware resources	MAC	XNOR-popcount	Reduction
LUT	379	172	54.6%
Flip-Flops	166	123	25.9%
DSP	2	0	100%

Figure 7 shows the power-latency behavior of the two Prewitt implementations [11]. The dynamic-power plots show that the XNOR-popcount core consistently consumes less power than the MAC-based core, with a reduction of about 60% across the evaluated operating points. The latency results in Fig. 7 were obtained under the fixed-clock evaluation setting, whereas the maximum timing-closure result characterizes the highest achievable operating frequency. Under these conditions, the

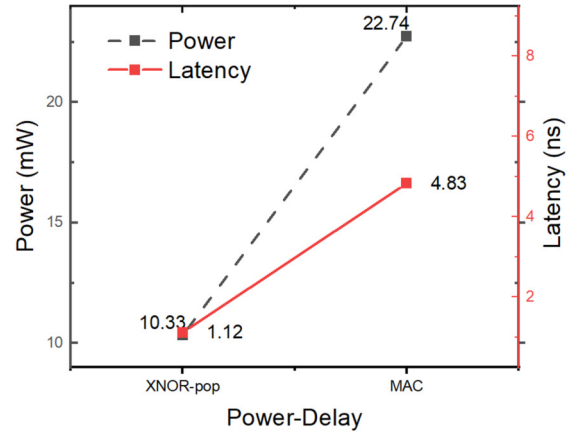


Fig. 7: Implementation results for the Prewitt edge-detection cores under the evaluation setting: dynamic power and per-frame latency measured at 200 MHz. The separately reported timing-closure frequency is approximately 250 MHz [11].

XNOR-popcount datapath achieves lower per-frame latency because of its shallower arithmetic path, and timing closure is reported at approximately 250 MHz, about five times the MAC baseline [11]. This efficiency gain is achieved with only a limited reduction in fidelity. In the evaluation, dissimilarity remains below 0.08 relative to exact arithmetic, while normalized accuracy remains around 0.93 [11]. Taken together, these results indicate that XNOR-popcount reformulation can deliver substantial hardware savings while maintaining practically useful output quality in the evaluated Prewitt cases.

Figure 8 examines the effect of approximation width on the Sobel accelerator. At the system level, resource utilization remains relatively stable

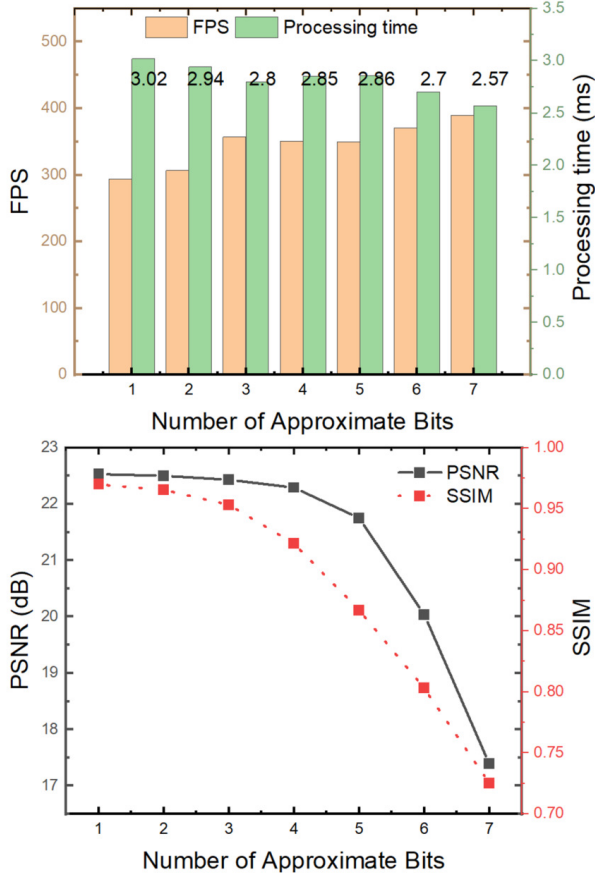


Fig.8: Effect of approximation width on the Sobel accelerator in terms of throughput, processing time, PSNR, and SSIM [12].

across the evaluated adder configurations. LUT usage varies only within a narrow range, the FF count remains largely stable except for a synthesis-related increase in the 4-bit configuration, and DSP and block random-access memory (BRAM) usage remain unchanged. At the kernel level, LUT usage increases only moderately before stabilizing, while BRAM consumption remains constant. These results suggest that approximate arithmetic has a limited impact on the overall resource profile of the system.

Compared with the exact baseline, corresponding to 0-bit approximation, increasing the approximation width up to 7 bits improves throughput from 220.75 FPS to 357.14 FPS and reduces latency from 4.53 ms to 2.8 ms [12]. However, image quality decreases progressively as the approximation width increases beyond moderate levels. The 3-bit approximate design provides a balanced operating point, achieving a PSNR of 22.43 dB and an SSIM of 0.9532, while still reaching 357.14 FPS and 2.8 ms latency. Therefore, the 3-bit configuration represents the best reported trade-off in this design space, with 357.14 FPS, 2.8 ms latency, 0.164 W power consumption, 22.43 dB PSNR, and 0.9532 SSIM.

Figure 9 compares qualitative Sobel outputs gen-

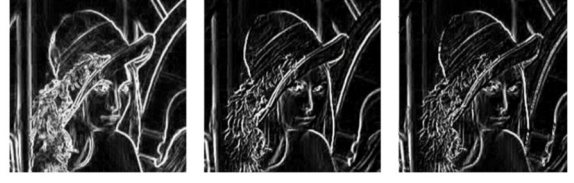


Fig.9: Qualitative Sobel outputs produced by the CPU, exact FPGA, and approximate FPGA implementations [12].

erated by the central processing unit (CPU), exact FPGA, and approximate FPGA implementations. In the comparison, the exact FPGA output provides the closest match to the selected reference output. In contrast, the approximate FPGA output retains the dominant contours and salient structural features with limited artifacts around weak edges. The CPU result should be interpreted only under the specific implementation, precision, and threshold settings used in the experiment; it should not be taken as evidence that CPU-based execution is inherently less accurate. Under these settings, the approximate FPGA design preserves the main edge information while improving throughput and energy efficiency. This comparison indicates that selective approximate arithmetic can preserve the functionally important content of the edge map. Although the approximate Sobel design does not reproduce every weak detail exactly, it retains the dominant geometric information required for practical embedded vision.

Table 3: System-Level Comparison of CPU, Exact FPGA, and Approximate FPGA Sobel Implementations [12].

Metric	FPGA		CPU
	Approximate computation	Exact computation	
PSNR (dB)	22.43	40.00	11.3
SSIM	0.9532	1.0000	0.397
FPS	357.14	220.75	0.0717
Latency (ms)	2.8	4.53	13928.65
FPS/W	2177.6	1206.3	0.026
Power (W)	0.164	0.183	2.72

Table 3 compares the Sobel implementations at the system level [12]. The 3-bit approximate FPGA achieves 357.14 FPS, 2.8 ms latency, 0.164 W power, and 2177.6 frames per second per watt (FPS/W). The exact FPGA achieves higher image quality, with PSNR of 40 dB and SSIM of 1.0, but lower performance at 220.75 FPS and 4.53 ms latency, together with higher power consumption. The CPU baseline exhibits lower image quality, lower throughput, substantially higher latency, and lower energy efficiency than the two FPGA implementations. These results indicate that the approximate FPGA occupies the most favorable efficiency-performance region among the three evaluated platforms. The hardware over-

head is also limited: the approximate version uses 6309 LUTs and 8328 FFs, compared with 6291 LUTs and 8302 FFs for the exact design, with no increase in DSP, look-up table random-access memory (LUTRAM), or BRAM usage [12]. The study reports that dynamic power is about 10% lower than that of the exact FPGA. This suggests that selective approximation is a practical optimization for Sobel-based embedded vision when the approximation width is chosen carefully.

Table 4: FPGA resource utilization comparison for exact and approximate adders [12].

Resource	Approximate adder	Exact adder	Available
LUTs	6309	6291	53200
FFs	8328	8302	106400
DSP	9	9	220
LUTRAM	683	683	17400
BRAM	3	3	140

Table 4 compares the system-level FPGA resource utilization of the exact and approximate Sobel accelerators. The approximate design introduces only small overheads in LUT and FF usage, approximately 0.29% and 0.31%, respectively, compared with the exact FPGA implementation, while DSP, LUTRAM, and BRAM consumption remain unchanged. Thus, the main benefit of approximation in the Sobel branch is not area reduction at the system level, but improved throughput and energy efficiency with limited additional hardware cost.

Finally, Table 5 compares the framework with representative FPGA-based edge-detection designs from the literature [31]. A key distinction becomes clear. Designs that remain in conventional fixed-point or multiplier-based arithmetic can still obtain worthwhile gains through approximation, but their arithmetic structure remains fundamentally multi-bit and

MAC-oriented. In contrast, the Prewitt branch of the present framework eliminates multipliers through binary reformulation, producing larger reductions in LUT count, DSP usage, dynamic power, and timing cost. The Sobel branch addresses a different case: because full binary reformulation is less natural, it applies approximation selectively in the accumulation path and thereby improves throughput and energy efficiency with only modest hardware overhead. The framework should therefore not be interpreted as a single architecture optimized for all kernels. Rather, it is a methodology for assigning arithmetic style according to kernel properties. Within the scope of the previous studies, this methodology combines area reduction, DSP elimination, lower power, improved timing, and acceptable output quality within one coherent design approach.

5. CONCLUSION

This study presented a kernel-aware arithmetic-selection framework for energy-efficient FPGA edge detection. The design rule is to prioritize XNOR–popcount-based multiplier elimination when the kernel supports binary or bipolar reformulation, and to apply approximate arithmetic only to residual multi-bit stages when full binary mapping is not natural. Prewitt-like kernels whose active nonzero taps are restricted to ± 1 benefit primarily from XNOR–popcount reformulation with zero-tap masking, which reduces LUT and FF usage, eliminates DSP use, and preserves the dominant edge structures observed in the qualitative comparisons. In contrast, Sobel-like mixed-coefficient kernels benefit from a hybrid datapath that combines shift-add processing with selective approximate accumulation. In the Sobel implementation, the 3-bit approximate config-

Table 5: Comparison with representative arithmetic optimizations relevant to FPGA edge-detection pipelines.

Study	Branch/operator	Arithmetic style	Hardware overhead	Power/energy effect	Frequency/throughput	Main insight
[31]	Multiplier design	Approximate fixed-point multiplier.	22% fewer LUTs than exact Booth/Dadda equivalents.	Up to 80% energy saving versus exact multipliers.	250 MHz on Artix-7; 1.8× speedup.	Remains multi-bit multiplier-oriented.
This study	Prewitt branch	XNOR–popcount reformulation.	54.6% LUT reduction, 25.9% FF reduction, DSPs eliminated.	≈60% lower dynamic power than MAC.	≈250 MHz on Zynq-7000; ≈5× speedup.	Multiplier elimination is most effective for ± 1 kernels.
	Sobel branch	Hybrid shift-add and selective approximate accumulation.	≈0.29% LUT and ≈0.31% FF overhead versus exact FPGA; DSP/LUTRAM/BRAM unchanged.	≈10% lower dynamic power than exact FPGA.	357.14 FPS, 2.8 ms latency for 3-bit approximation.	Selective approximation is beneficial for mixed-coefficient kernels where full binary reformulation is not natural.

uration achieves 357.14 FPS, 2.8 ms latency, 0.164 W power, 22.43 dB PSNR, and 0.9532 SSIM. These results support the use of kernel-aware arithmetic selection as a practical methodology for matching datapath structure to kernel properties in low-power FPGA edge-detection designs.

ACKNOWLEDGEMENT

This study was financially supported by Ho Chi Minh City University of Technology and Engineering (HCM-UTE), Vietnam.

AUTHOR CONTRIBUTIONS

Conceptualization, methodology, formal analysis, V.-K.P.; investigation, V.-K.P.; writing—original draft preparation, V.-K.P.; writing—review and editing, V.-K.P.

References

- [1] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30-39, 2017.
- [2] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," in *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655-1674, Aug. 2019.
- [3] V. Sze, Y. -H. Chen, T. -J. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295-2329, Dec. 2017.
- [4] M. Horowitz, "1.1 Computing's Energy Problem (and What We Can Do About It)," *Proceedings of the IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10-14, 2014.
- [5] N. P. Jouppi *et al.*, "In-Datcenter Performance Analysis of a Tensor Processing Unit," *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1-12, 2017.
- [6] Y.-H. Chen, T. Krishna, J. S. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127-138, 2017.
- [7] S. Han, H. Mao and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks With Pruning, Trained Quantization and Huffman Coding," *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.
- [8] A. Gholami *et al.*, "SqueezeNext: Hardware-Aware Neural Network Design," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Salt Lake City, UT, USA, pp. 1719-1728, 2018.
- [9] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv Preprint arXiv:1704.04861*, 2017.
- [10] V.-K. Pham, L. Le and T.-K. Tran Thi, "XNOR-Popcount, an Alternative Solution to the Accumulation Multiplication Method for Approximate Computations, to Improve Latency and Power Efficiency," *Journal of Technical Education Science*, vol. 20, no. 1, pp. 12-20, 2025.
- [11] V.-K. Pham and L. Le, "An Energy-Efficient Hardware Module for Edge Detection Using XNOR-Popcount in Resource-Constrained Devices," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 41, no. 1, pp. 73-82, 2026.
- [12] B.-L. Pham, V.-K. Pham, M.-P. Doan Thi and T.-T. Nguyen, "Real-Time, Energy-Efficient Sobel Edge Detection on FPGA Using Approximate Carry-Lookahead Adders," *Proceedings of the 28th International Conference on Mechatronics Technology (ICMT)*, pp. 357-362, 2025.
- [13] M. Rastegari, V. Ordonez, J. Redmon and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," *Proceedings of the European Conference on Computer Vision (ECCV)*, Lecture Notes in Computer Science, vol. 9908, pp. 525-542, 2016.
- [14] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks With Weights and Activations Constrained to +1 or -1," *arXiv Preprint arXiv:1602.02830*, 2016.
- [15] S. Zhu, L. H. K. Duong and W. Liu, "XOR-Net: An Efficient Computation Pipeline for Binary Neural Network Inference on Edge Devices," *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 124-131, 2020.
- [16] M. Rastegari, V. Ordonez, J. Redmon and A. Farhadi, "Enabling AI at the Edge With XNOR-Networks," *Communications of the ACM*, vol. 63, no. 12, pp. 83-90, 2020.
- [17] S. Rasoulinezhad, S. Fox, H. Zhou, L. Wang, D. Boland and P. H. W. Leong, "MajorityNets: BNNs Utilising Approximate Popcount for Improved Efficiency," *Proceedings of the International Conference on Field-Programmable Technology (ICFPT)*, pp. 339-342, 2019.
- [18] F. Conti, P. D. Schiavone and L. Benini, "XNOR Neural Engine: A Hardware Accelerator IP for 21.6-fJ/op Binary Neural Network Inference," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2940-2951, Nov. 2018.
- [19] Y. Liu *et al.*, "Richer Convolutional Features for Edge Detection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 8, pp. 1939-1946, 1 Aug. 2019.
- [20] W. Luo, Y. Li, R. Urtasun and R. Zemel,

- “Understanding the Effective Receptive Field in Deep Convolutional Neural Networks,” *Advances in Neural Information Processing Systems*, pp. 4905–4913, 2016.
- [21] J. Han and M. Orshansky, “Approximate Computing: An Emerging Paradigm for Energy-Efficient Design,” *Proceedings of the 18th IEEE European Test Symposium (ETS)*, pp. 1–6, 2013.
- [22] L. B. Soares, M. M. A. da Rosa, C. M. Diniz, E. A. C. da Costa and S. Bampi, “Exploring Power-Performance-Quality tradeoff of Approximate Adders for Energy Efficient Sobel Filtering,” *Proceedings of the IEEE 9th Latin American Symposium on Circuits and Systems (LAS-CAS)*, pp. 1–4, 2018.
- [23] Y. Chung and Y. Kim, “Comparison of Approximate Computing With Sobel Edge Detection,” *IEEE Transactions on Smart Processing and Computing*, vol. 10, no. 4, pp. 355–361, 2021.
- [24] A. B. Kahng and S. Kang, “Accuracy-Configurable Adder for Approximate Arithmetic Designs,” *Proceedings of the 49th Annual Design Automation Conference (DAC)*, pp. 820–825, 2012.
- [25] J. Schiel and A. Bainbridge-Smith, “Efficient Edge Detection on Low-Cost FPGAs,” *arXiv Preprint arXiv:1512.00504*, 2015.
- [26] N. Shylashree, M. A. Naik, A. S. Mamatha and V. Sridhar, “Design and Implementation of Image Edge Detection Algorithm on FPGA,” *International Journal of Circuits, Systems and Signal Processing*, vol. 16, pp. 628–636, 2022.
- [27] N. Nausheen, A. Seal, P. Khanna and S. Halder, “A FPGA-Based Implementation of Sobel Edge Detection,” *Microprocessors and Microsystems*, vol. 56, pp. 84–91, 2018.
- [28] AMD Xilinx, *Vitis High-Level Synthesis User Guide (UG1399)*, Version 2023.1, 2023.
- [29] AMD Xilinx, *Vitis Vision Library User Guide*, Version 2023.1, 2023.
- [30] AMD, *Zynq 7000 SoC Technical Reference Manual (UG585)*, Version 1.15, 2026.
- [31] S. Perri, F. Spagnolo, F. Frustaci and P. Corsonello, “Designing Energy-Efficient Approximate Multipliers,” *Journal of Low Power Electronics and Applications*, vol. 12, no. 4, Art. 49, 2022.



Van-Khoa Pham received the B.S. and M.S. degrees in Computer Technology and Electronics Engineering from the Ho Chi Minh City University of Technology and Education (HCMUTE), Ho Chi Minh City, Vietnam, in 2010 and 2014, respectively, and the Ph.D. degree in electronics engineering from Kookmin University, Seoul, South Korea, in 2019. In 2010, he joined the Integrated Circuit Design Research and Education Center (ICDREC), where he contributed to the development of the VN8-01 MCU, the first commercially designed and fabricated microcontroller in Vietnam. From 2011 to 2021, he was a faculty member with the Faculty of Electrical and Electronics Engineering, HCMUTE. He is currently a Senior Lecturer with the Department of Computer and Communication Engineering, HCMUTE. He also serves as the Head of the Computer Engineering Technology Program and the Electronics and Communications Engineering Technology Program, Faculty of Advanced Education, Ho Chi Minh City University of Technology and Engineering (HCM-UTE).

His research interests include low-power VLSI design, memory design, Internet of Things applications, and AI-based solutions. His email address is khoapv@hcmute.edu.vn. ORCID: 0000-0002-6129-5856.