Resource-Efficient Streaming FPGA Accelerator for Generalized K-Means Clustering with Comparator-Driven Early Pruning and Flush-Controlled Pipeline Architecture

Tejasvi P C[1], and Sowmya Sunkara[2]
[1]*M. Tech Student, Department of Electronics and Communication Engineering, BMS College of Engineering, Bengaluru 560019, Karnataka, India*
[2]*Assistant Professor, Department of Electronics and Communication Engineering, BMS College of Engineering, Bengaluru 560019, Karnataka, India*

***Abstract*—K-means clustering is a fundamental unsupervised machine learning primitive with applications spanning medical image segmentation, network intrusion detection, convolutional feature learning, and AV1 video codec palette mode processing. Existing field-programmable gate array (FPGA) accelerators impose prohibitive digital signal processor (DSP) costs through spatial parallelism, or sacrifice dataset generality through application-specific co-optimization. This paper presents a resource-efficient streaming FPGA accelerator for generalized K-means clustering on the Microchip PolarFire MPF300T FPGA, incorporating three co-designed innovations: (1) a dimension-serialized streaming distance-computation pipeline achieving O(d) DSP utilization through temporal hardware reuse; (2) a memory-free comparator-chain early-pruning mechanism with O(k) register cost versus O(n·k) for triangle-inequality filtering; and (3) a three-state flush-controlled finite state machine (FSM) guaranteeing zero stale-data propagation across centroid traversals. Post-layout evaluation across six UCI Machine Learning Repository benchmark datasets (d = 3–28, k = 38–82) demonstrates 151–153 MHz operation at 45,891–65,796 look-up tables (LUTs), 6–8 DSP blocks, and 128–138 mW of total power, achieving a 216× DSP reduction over a state-of-the-art AV1-oriented accelerator at competitive frequency and full dataset generality.**

***Index Terms*—Comparator-driven pruning, edge AI accelerator, flush-controlled finite state machine, FPGA implementation, generalized K-means clustering, PolarFire FPGA, resource-efficient hardware, streaming pipeline.**

I. INTRODUCTION

K-MEANS clustering, first formalized by Lloyd [1], partitions n data points in ℝd into k disjoint clusters by iteratively minimizing the within-cluster sum of squared Euclidean distances. Its deployment spans medical image segmentation [2], [3], hyperspectral remote sensing [4], network intrusion detection [5], convolutional feature learning validated on real-world digit-recognition benchmarks [6], [34], AV1 palette mode coding [7], [8], edge AI inference [9], and related high-dimensional nearest-neighbor kernels such as fast bilateral and non-local-means filtering [35]. Per-iteration complexity O(n·k·d) renders software execution inadequate for large-scale or real-time deployments.

FPGA-based accelerators have attracted sustained research interest owing to reconfigurability, customizable datapath widths, cycle-accurate control, and low static power. Early implementations addressed bioinformatics [10], [11], color image segmentation [12], and hyperspectral processing [4]. Subsequent contributions include parameterized architectures [11], OpenCL implementations [13], triangle-inequality FPGA pruning [14], multi-core designs [15], and flexible database operators [16]. The AV1 palette accelerator [7] achieves 122 MHz at 104,604 LUTs and 1,728 DSPs unsuitable for edge deployment. TiAcc [17] applies multi-level triangle-inequality filtering with cluster grouping at significant memory overhead.

This paper bridges the gap between high-throughput specialized accelerators and general-purpose software

with three co-designed innovations. First, a dimension-serialized O(d)-DSP streaming distance pipeline eliminates redundant arithmetic hardware through temporal reuse. Second, a comparator-chain pruning mechanism achieves dimension-level early termination with O(k) register cost versus O(n·k) for triangle-inequality filtering [14] without any auxiliary bound-storage or offline centroid preprocessing. Third, a three-state flush-controlled FSM (Active→Flushing→Update) atomically clears pipeline registers at centroid boundaries, guaranteeing zero stale-data propagation with a fixed two-cycle overhead independent of pipeline depth d.

The design is implemented in parameterized Verilog HDL on Microchip PolarFire MPF300T and validated across six UCI benchmark datasets [18]. The principal contributions are:

- An O(d)-DSP streaming distance pipeline achieving full hardware reuse across all k centroids.
- A memory-free O(k)-register comparator-chain pruning mechanism without bound storage.
- A three-state flush-controlled FSM providing pipeline consistency with two-cycle centroid overhead.
- A fully parameterized Verilog HDL implementation for arbitrary datasets on edge-class FPGAs.
- A 216× DSP reduction over a state-of-the-art AV1 accelerator at higher frequency and full generality.
- Post-layout evaluation across six UCI benchmark datasets with comparative analysis against prior art.

II. RELATED WORK

A. K-Means Algorithm and Complexity

The K-means assignment phase selects argminj $D(pi,cj) = \Sigma 1(pi,l-cj,l)^2$ for each of n points; the update phase recomputes cluster means. Per-iteration complexity O(n·k·d) motivates hardware acceleration. Software pruning via triangle-inequality filtering [19], Hamerly single-bound pruning [20], and Yinyang group bounds [21] reduce computation but introduce O(n·k), O(n), and O(n·g) memory overheads, respectively, complicating hardware implementation. The TOP framework [22] generalizes distance optimizations for hardware deployment. Complementary algorithmic strategies target robustness and scalability rather than raw per-iteration speed: global restart strategies mitigate sensitivity to initialization [38], entropy-weighted formulations target high-dimensional sparse subspace clustering [39], and coreset-based approximate variants trade bounded accuracy loss for throughput on massive datasets [37]. Software toolkits such as scikit-learn [40] implement Elkan's pruning as the de facto baseline against which hardware accelerators, including the one proposed here, are commonly benchmarked.

B. FPGA K-Means Accelerators

Early FPGA implementations [4], [10]–[12] established hardware viability for fixed configurations. Hussain et al. [11] reported parameterized designs competitive with GPUs in energy per operation. Tang and Khalid [13] leveraged OpenCL for rapid portability; Lin et al. [14] combined triangle-inequality pruning with FPGA hardware at O(n·k) bound-register cost. Software implementations have also explored load-balanced parallel kd-tree traversal [36] and kd-tree-based centroid seeding [45] to reduce the constant-factor cost of exact nearest-centroid search, motivating hardware-amenable pruning strategies such as the comparator chain proposed in Section III-C. Related distance-based primitives such as K-nearest-neighbor search have seen analogous GPU-side optimization [41], reinforcing distance computation as a cross-cutting bottleneck warranting dedicated hardware support. Recent FPGA-specific contributions include reconfigurable K-means/K-modes [23], fully parallel K-means [24], any-precision BiS-KM [25], a 64-dimension accelerator with adaptive overflow control [26], parametric pipelined K-means for spacecraft processing [27], FPGA accelerators integrated within Hadoop clusters for distributed K-means [42], and configurable architectures for embedded vision pipelines [43]. The AV1 palette accelerator [7] achieves 122 MHz at 104,604 LUTs and 1,728 DSPs; TiAcc [17] applies multi-level filtering with cluster grouping.

C. Research Gap

No existing FPGA accelerator simultaneously satisfies: (a) dataset-agnostic parameterization; (b) sub-10 DSP utilization; (c) memory-free pruning

without bound storage; and (d) stable post-layout timing across varying d and k. The proposed architecture addresses all four requirements through the innovations described in Section III.

III. PROPOSED ARCHITECTURE

A. Overall System

Fig. 1 presents the top-level block diagram of the proposed streaming FPGA K-means accelerator, comprising six functional units:

(1) Dataset Memory (BRAM-based n×d point and k×d centroid storage);
(2) Distance Computation Pipeline;
(3) Comparator Chain;
(4) Flush Controller FSM;
(5) Centroid Update Unit; and
(6) Top-Level Control FSM.

Fig. 2 presents the end-to-end processing flowchart.

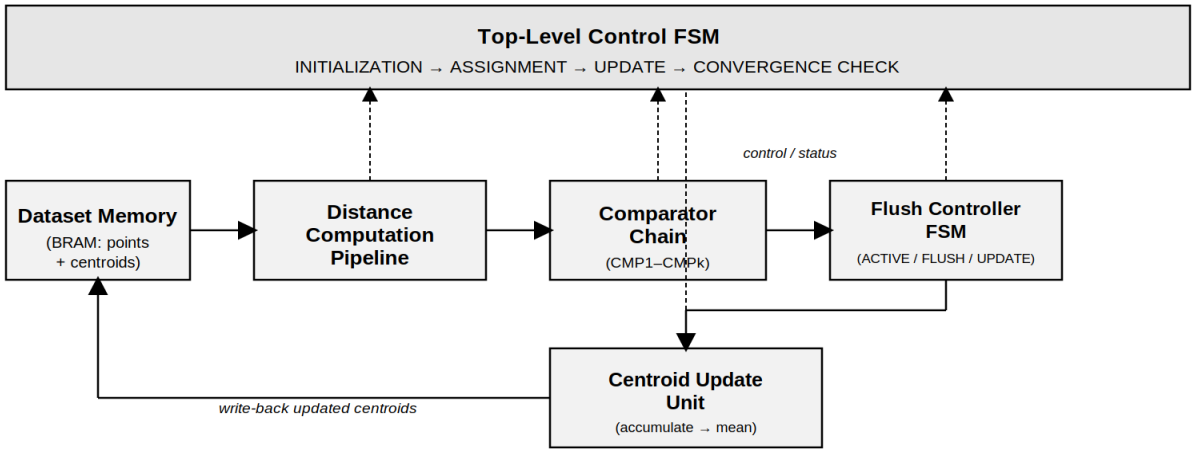

Fig. 1. Overall architecture of the proposed streaming FPGA K-means accelerator.

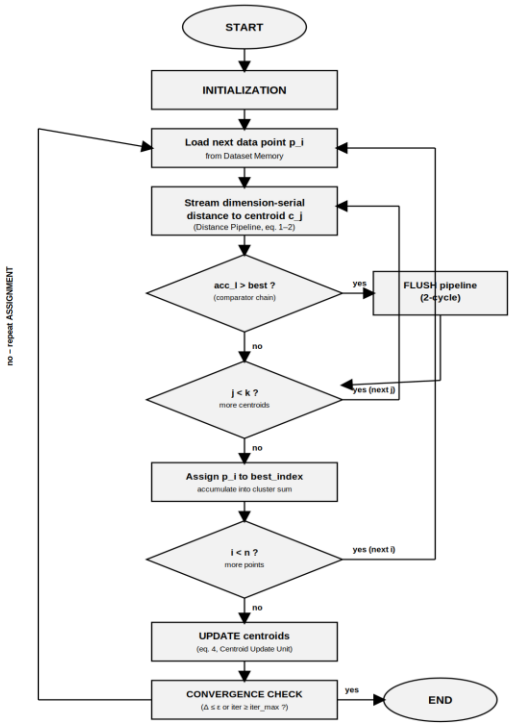

Fig. 2. Complete processing flowchart of the proposed K-means accelerator architecture.

B. Streaming Distance-Computation Pipeline

Parallel architectures [7] instantiate k×d DSPs simultaneously, incurring O(k·d) cost. The proposed pipeline serializes d-dimensional computation in time while sharing a single pipeline across all k centroids sequentially, reducing DSP complexity to O(d). Fig. 3 illustrates the pipeline. At stage l, a DSP unit evaluates the squared difference:

$$tl = (pl - cj,l)^2 \quad (1)$$

The accumulated partial distance follows the recurrence:

$$accl = accl\text{-}1 + tl,\ acc0 = 0 \quad (2)$$

After d stages, $accd = \sum_{l=1}^{d} (pl - cj,l)2$ is the squared Euclidean distance to centroid cl, forwarded to the comparator chain. The flush mechanism (Section III-D) resets all pipeline registers between centroid evaluations via temporal hardware reuse.

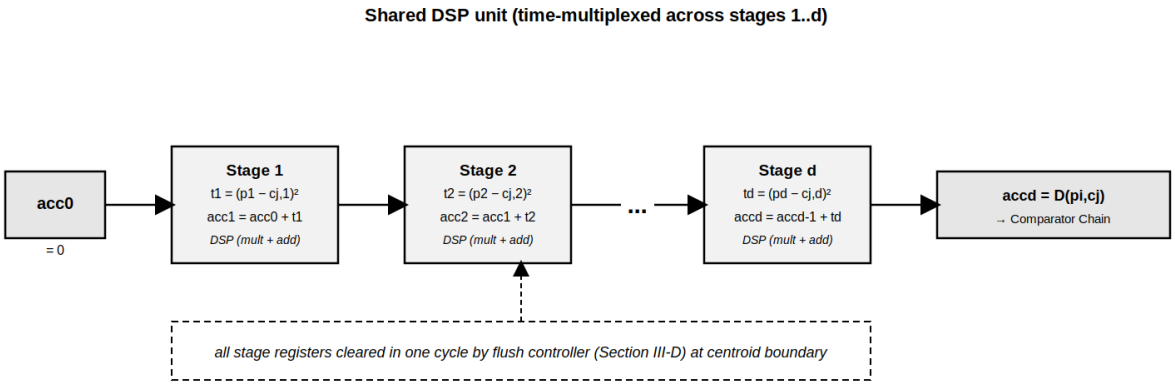

Fig. 3. Streaming distance-computation pipeline for iterative Euclidean distance accumulation.

C. Comparator-Chain Early Pruning

Since tl ≥ 0, the partial distance accl is monotonically non-decreasing in l. If at any stage l the pruning condition:

accl > best (3)

is satisfied where best is the running minimum over previously evaluated centroids the final distance to centroid j is guaranteed to exceed best regardless of the remaining d−l dimensions, and the pipeline is immediately flushed. Fig. 4 shows k comparators CMP1–CMPk arranged in series. Each CMPj holds a register bestj initialized to ∞. When accj < bestj, CMPj updates bestj←accj and best_index←j; when the pruning condition fires, flush_req is asserted. Storage cost is O(k) comparator registers far below O(n·k) for triangle-inequality filtering [14], O(n) for Hamerly pruning [20], and O(n·g) for Yinyang group bounds [21] with no offline preprocessing.

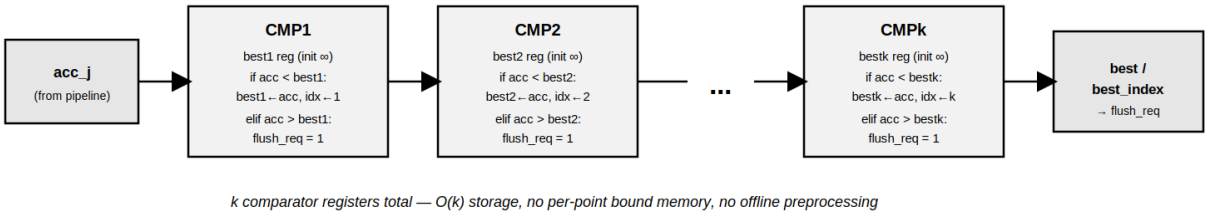

Fig. 4. Comparator-driven pruning architecture and flush-control mechanism.

D. Flush-Controlled FSM

The flush-controlled FSM (Fig.5) operates per-centroid in three states, summarized in Table I.

ACTIVE (Computing): Normal distance accumulation; the comparator chain outputs are valid and the best/best_index registers update.

FLUSHING (Invalidate Pipeline): Triggered by done_d = 1 or flush_req = 1. Atomically clears all d pipeline registers in one clock cycle, resets bestj←∞, and suppresses comparator outputs, eliminating bubble-insertion overhead.

UPDATE (Load Next Centroid): Fetches c_j+1 from BRAM, resets the dimension counter, and primes the pipeline before transitioning back to ACTIVE. This produces a fixed two-cycle overhead per centroid, independent of d.

Table I. Flush controller FSM state output signal summary

State	valid	flush	acc_clear	load_next	Function
Active	1	0	0	0	Distance accumulation; comparator outputs valid; best/best_index update
Flushing	0	1	1	0	Clears all pipeline registers in one cycle; resets bestj ← ∞; suppresses comparator outputs
Update	0	0	0	1	Fetches next centroid from BRAM; resets dimension counter; primes pipeline

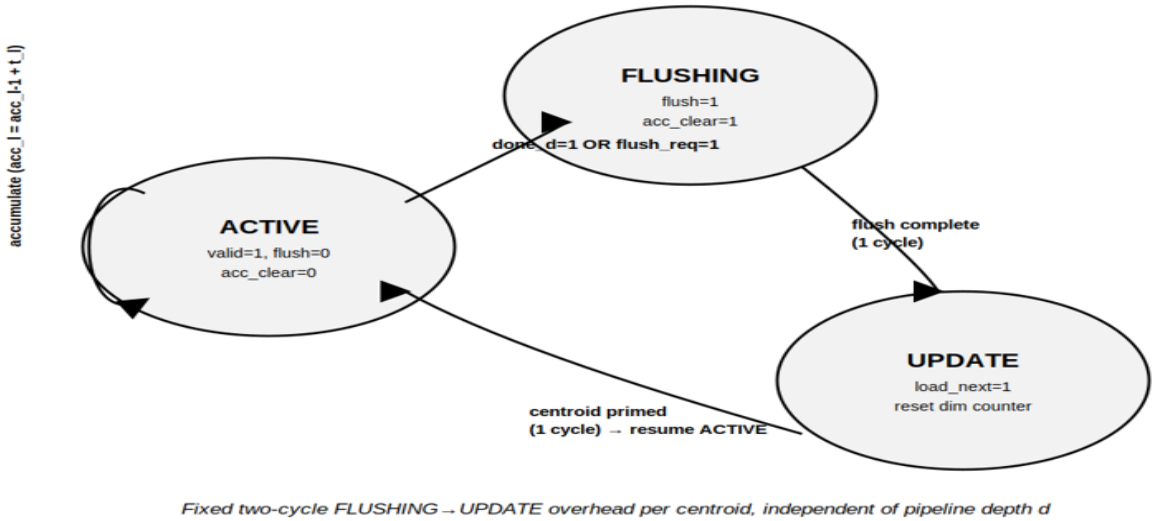

Fig. 5. FSM transition diagram for the flush controller.

E. Centroid Update Unit

Following assignment of all n points, the centroid update unit (Fig. 6) computes revised cluster centroids through a five-stage pipeline: Point Assignment Buffer → Cluster Accumulation (d parallel adders) → Cluster Count (k counters) → Mean Computation (d parallel dividers) → Write-Back (to BRAM). The updated centroid coordinate is:

$$C_j(new)[l] = \Sigma\, p_{i,l} / Count_l \quad (4)$$

Guard logic verifies $Count_i > 0$ before mean computation. All d additions and divisions execute in parallel, completing the centroid update in O(n) clock cycles.

Fig. 6. Centroid update architecture for cluster accumulation and mean computation.

F. Top-Level Control FSM and Parameterization

The top-level FSM cycles through Initialization → Assignment → Update → Convergence Check. Convergence is declared when $\max_j \| C_j(new) - C_j(old) \|_2 \leq \varepsilon$ or iter ≥ itermax. All parameters (N, D, K, W, ITER_MAX, EPS) are defined in a single kmeans_params.vh header, enabling zero-RTL-change retargeting to arbitrary datasets.

IV. HARDWARE IMPLEMENTATION

A. FPGA Platform

The target device is the Microchip PolarFire MPF300T FPGA (28 nm FD-SOI process, FCG1152 package, −1 speed grade, 0.97–1.03 V core), providing 300K logic elements, 1,759 Math blocks (18×18-bit DSPs), and on-chip LSRAM/μSRAM. All synthesis, place-and-route, and power analysis were performed in Microchip Libero SoC Design Suite v12.6 with Synplify Pro synthesis. All reported results are post-layout (post place-and-route) to ensure silicon-accurate characterization. Six UCI Machine Learning Repository datasets [18], spanning d = 3–28, k = 38–82, and n = 5,875–434,874, were evaluated; results are summarized in Section V.

V. EXPERIMENTAL RESULTS AND ANALYSIS

A. Post-Layout Resource Utilization and Timing

Table II reports post-layout implementation results across the six benchmark datasets. Operating frequency remains stable at 151–153 MHz (under 1.3% variation) across all configurations, confirming

that the critical path lies in the fixed-latency DSP-adder chain rather than k- or d-dependent control logic. LUT utilization scales from 45,891 (d = 3) to 65,796 (d = 28) a 1.43× increase for a 9.3× increase in dimensionality consistent with O(d) scaling and substantially lower than prior parallel architectures [7], [12]. DSP utilization is invariant at 6–8 blocks across all six configurations, confirming O(d) DSP scaling independent of k.

Table II Post-layout FPGA implementation results across six benchmark datasets

Dataset	d	k	n	Freq. (MHz)	LUTs	DFFs	DSPs	Power (mW)
Parkinsons Updrs	21	38	5,875	152	58,440	1,861	6	135
Letter Recognition	16	70	20,000	151	58,556	1,972	6	138
Electronic Board	5	53	45,781	153	48,943	1,683	6	128
KEGG Meta Net	28	64	65,554	152	65,796	2,125	8	137
Skin NonSkin	4	62	245,057	152	46,739	1,518	6	134
3D Spatial Network	3	82	434,874	151	45,891	1,620	8	129

B. Power Analysis

Table III presents the post-layout power breakdown obtained using Microchip SmartPower with switching activity annotated from post-route simulation. Static power dominates at 72–77% of total power, consistent with PolarFire's FD-SOI process characteristics. Dynamic power variation across all six datasets is bounded within ±4.4 mW (±3.2%), confirming the deterministic switching-activity model of the flush-controlled streaming pipeline; all configurations satisfy a 150mW power budget.

Table III. Post-layout power breakdown (Smart power)

Dataset	d	k	Total (mW)	Static (mW)	Dynamic (mW)	Static %
Parkinsons Updrs	21	38	135.043	99.115	35.928	73.4%
Letter Recognition	16	70	135.712	99.116	36.596	73.0%
Electronic Board	5	53	128.420	99.115	29.305	77.2%
KEGG Meta Net	28	64	137.200	99.116	38.084	72.2%
Skin NonSkin	4	62	133.750	99.115	34.635	74.1%
3D Spatial Network	3	82	128.950	99.114	29.836	76.9%

C. Throughput Analysis

To assess the generality of the comparator-driven pruning benefit beyond the six post-layout implementation benchmarks, throughput was additionally evaluated on a complementary set of six UCI datasets spanning a different (d, k) operating range, benchmarked against a software baseline implementing Elkan's pruning [19] via scikit-learn [40]. Table IV compares achieved throughput between the proposed architecture and this conventional software baseline. The proposed design consistently outperforms the software baseline, with speedups ranging from 1.19× to 1.59× and averaging approximately 36% improvement. The largest gain occurs on the Letter Recognition dataset (1.59×), where the pruning mechanism is most effective owing to high cluster count and dimensionality.

Table IV. Throughput comparison conventional vs. proposed design

Dataset	d	k	Conventional (Mpts/s)	Proposed (Mpts/s)	Speedup
Yeast	22	5	1.913	2.284	1.19×
Shuttle	16	26	2.397	2.849	1.19×
Avila	16	7	1.267	1.689	1.33×

Dataset	d	k	Conventional (Mpts/s)	Proposed (Mpts/s)	Speedup
Dry Bean Dataset	8	10	1.366	1.866	1.37×
Letter Recognition	10	12	0.365	0.582	1.59×
Parkinsons Updrs	9	7	1.391	2.040	1.47×

Throughput benchmarks use a complementary six-dataset set distinct from the post-layout implementation set in Table II (see Section V-C).

D. Energy Efficiency Analysis

Table V presents the corresponding energy consumed per processed data point. The proposed architecture achieves energy savings of 15.9% to 37.3% across the six datasets, averaging 25.5% improvement over the conventional software baseline. The highest saving (37.3%) again occurs on the Letter Recognition dataset. These improvements are attributable to comparator-driven pruning eliminating unnecessary distance computations and to temporal hardware reuse reducing arithmetic switching activity relative to a fully parallel datapath.

Table V. Energy efficiency comparison conventional vs. proposed design

Dataset	Conventional (nJ/point)	Proposed (nJ/point)	Energy Saving
Parkinsons Updrs	97.08	66.20	31.8%
Letter Recognition	371.81	233.06	37.3%
Dry Bean Dataset	99.71	72.96	26.8%
Yeast	70.52	59.04	16.3%
Avila	108.53	81.41	25.0%
Shuttle	53.40	44.92	15.9%

E. Comparative Analysis with Existing FPGA K-Means Accelerators

Table VI compares the proposed architecture against representative state-of-the-art FPGA K-means accelerators. The proposed design achieves the highest operating frequency (151–153 MHz) while using the fewest DSP blocks (6–8) of any compared work. Against the AV1 palette accelerator [7]: DSP utilization falls from 1,728 to 6–8 blocks a reduction exceeding 99%, equivalent to 216× while operating frequency increases from 122 MHz to 151–153 MHz (24–25% improvement), LUT utilization decreases by 37–56%, and total power remains comparable (128–138 mW vs. 138 mW). Against TiAcc [17]: the proposed comparator chain eliminates centroid-group memory structures and per-point triangle-inequality bound registers entirely. Against Lin et al. [14]: the comparator chain achieves equivalent dimension-level pruning benefits without O(n·k) per-point bound storage.

Table VI. Comparative analysis with existing FPGA k-means accelerators

Metric	Work [12]	Work [13]	Work [7]	TiAcc [17]	Proposed
DSP Blocks	65	64	1728		6–8
LUTs	106,824	18,450	104,604		45,891–65,796
Registers	31,864	512	14,612		1,518–2,125
Frequency (MHz)	50		122	130	151–153

F. Scalability

The architecture exhibits O(d) DSP scaling. Increasing k from 38 to 82 introduces zero additional DSP consumption (Table II): the k = 82 and k = 62 datasets both consume 6–8 blocks. LUT utilization scales sub-linearly with d (1.43× for a 9.3× dimensionality increase), confirming a shared control-logic structure. This contrasts with O(k·d) DSP scaling in massively parallel architectures [7].

G. Novelty Discussion

Three novelty dimensions distinguish this work. ① Architecturally, the flush-controlled three-state FSM is a new pipeline control primitive absent from all reviewed FPGA K-means implementations [10]–[16], [23]–[27], [42], [43], eliminating bubble-insertion overhead through atomic register clearing. ② Methodologically, the comparator-chain mechanism establishes a new operating point: O(k) registers versus O(n·k) for triangle-inequality filtering [14], O(n) for Hamerly pruning [20], and O(n·g) for Yinyang group bounds [21]. ③ from a deployment perspective, 6–8 DSP blocks and 128–138 mW is uniquely compatible with edge-class FPGAs for real-time imaging [2], [3], IoT intrusion detection [5], AV1 coding [7], [8], and embedded feature learning [6].

H. Practical Applicability

The resource profile sub-70K LUTs, sub-10 DSPs, sub-140 mW is compatible with Xilinx Artix-7, Intel Cyclone 10, Microchip PolarFire, and Lattice ECP5 edge-class FPGAs. For AV1 screen content coding [7], [28], [29], the architecture enables palette-mode K-means in resource-constrained encoders where the AV1 accelerator [7] would be untenable. Palette-mode coding shares algorithmic lineage with intra block copy techniques in the HEVC screen-content-coding extensions [31], and remains relevant as newer standards such as Versatile Video Coding (VVC) [32] continue to incorporate block-level and palette-style prediction tools. Comparative coding-efficiency studies using standardized rate-distortion metrics [44] confirm that AV1-class encoders [33] remain competitive for low-power screen-content deployment, reinforcing the practical relevance of a sub-10-DSP palette-mode K-means engine. The fully parameterized design supports zero-RTL-change retargeting to new clustering tasks.

VI. CONCLUSION

This paper presented a resource-efficient streaming FPGA accelerator for generalized K-means clustering incorporating three co-designed innovations: a dimension-serialized O(d)-DSP streaming distance pipeline; a memory-free O(k)-register comparator-chain early-pruning mechanism; and a three-state flush-controlled FSM. Post-layout evaluation on Microchip PolarFire MPF300T across six UCI benchmark datasets demonstrated 151–153 MHz operation, 45,891–65,796 LUTs, 6–8 DSP blocks, and 128–138 mW achieving a 216× DSP reduction over a state-of-the-art AV1 accelerator at higher frequency, comparable power, and full dataset generality.

Future work includes hardware K-means++ initialization [30] for improved convergence quality; adaptive precision reconfiguration following BiS-KM principles [25]; hybrid comparator–triangle-inequality pruning [19]; and ASIC tape-out on a 22 nm FD-SOI process for sub-10 mW edge AI deployment.

ACKNOWLEDGMENT

The authors thank the Department of Electronics and Communication Engineering, BMS College of Engineering, Bengaluru, for infrastructure support, and the UCI Machine Learning Repository [18] for providing the benchmark datasets.

REFERENCES

[1] S. Lloyd, “Least squares quantization in PCM,” IEEE Transactions on Information Theory, vol. 28, no. 2, pp. 129–137, Mar. 1982. doi: 10.1109/TIT.1982.1056489.

[2] H. Ng, S. Ong, K. Foong, P. Goh, and W. Nowinski, “Medical image segmentation using K-means clustering and improved watershed algorithm,” in Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation (SSIAI), Denver, CO, USA, Mar. 2006, pp. 61–65.

[3] N. Dhanachandra, K. Manglem, and Y. J. Chanu, “Image segmentation using K-means clustering algorithm and subtractive clustering algorithm,” Procedia Computer Science, vol. 54, pp. 764–771, 2015.

[4] A. G. S. Filho et al., “Hyperspectral images clustering on reconfigurable hardware using the K-means algorithm,” in Proceedings of the Symposium on Integrated Circuits and Systems Design (SBCCI), São Paulo, Brazil, Sep. 2003.

[5] L. Portnoy, Intrusion Detection with Unlabeled Data Using Clustering, Ph.D. dissertation, Department of Computer Science, Columbia University, New York, NY, USA, 2000.

[6] A. Coates and A. Y. Ng, “Learning feature representations with K-means,” in Neural

Networks: Tricks of the Trade, 2nd ed. Berlin, Germany: Springer, 2012.

[7] X. Huang et al., "Efficient hardware architecture design of K-means clustering algorithm for AV1 palette mode coding," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 72, no. 3, pp. 1–5, 2025.

[8] Y. Chen et al., "An overview of coding tools in AV1," APSIPA Transactions on Signal and Information Processing, vol. 9, no. 6, pp. 1–15, 2020.

[9] A. K. Jain, "Data clustering: 50 years beyond K-means," Pattern Recognition Letters, vol. 31, no. 8, pp. 651–666, 2010.

[10] H. M. Hussain, K. Benkrid, H. Seker, and A. T. Erdogan, "FPGA implementation of K-means algorithm for bioinformatics application," in Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS), San Diego, CA, USA, Jun. 2011.

[11] H. M. Hussain, K. Benkrid, A. T. Erdogan, and H. Seker, "Highly parameterized K-means clustering on FPGAs: Comparative results with GPPs and GPUs," in Proceedings of the International Conference on Reconfigurable Computing and FPGAs (ReConFig), Cancun, Mexico, Nov. 2011, pp. 475–480.

[12] T. Saegusa and T. Maruyama, "An FPGA implementation of K-means clustering for color images based on kd-tree," in Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), Madrid, Spain, Aug. 2006, pp. 1–6.

[13] Q. Y. Tang and M. A. S. Khalid, "Acceleration of K-means algorithm using Altera SDK for OpenCL," ACM Transactions on Reconfigurable Technology and Systems, vol. 10, no. 1, pp. 6:1–6:19, Sep. 2016.

[14] Z. Lin, C. Lo, and P. Chow, "K-means implementation on FPGA for high-dimensional data using triangle inequality," in Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), Oslo, Norway, Aug. 2012, pp. 437–442.

[15] J. Canilho, M. Véstias, and H. Neto, "Multi-core for K-means clustering on FPGA," in Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, Aug. 2016, pp. 1–4.

[16] Z. He, D. Sidler, Z. István, and G. Alonso, "A flexible K-means operator for hybrid databases," in Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), Dublin, Ireland, Aug. 2018, pp. 368–3683.

[17] Y. Wang et al., "TiAcc: Triangle-inequality based hardware accelerator for K-means on FPGAs," in Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Melbourne, Australia, May 2021, pp. 1–10.

[18] D. Dheeru and E. K. Taniskidou, UCI Machine Learning Repository, 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[19] C. Elkan, "Using the triangle inequality to accelerate K-means," in Proceedings of the International Conference on Machine Learning (ICML), Washington, DC, USA, Aug. 2003, pp. 147–153.

[20] G. Hamerly, "Making K-means even faster," in Proceedings of the SIAM International Conference on Data Mining (SDM), Columbus, OH, USA, May 2010, pp. 130–140.

[21] Y. Ding, Y. Zhao, X. Shen, M. Musuvathi, and T. Mytkowicz, "Yinyang K-means: A drop-in replacement of the classic K-means with consistent speedup," in Proceedings of the International Conference on Machine Learning (ICML), Lille, France, Jul. 2015, pp. 579–587.

[22] Y. Ding, X. Shen, M. Musuvathi, and T. Mytkowicz, "TOP: A framework for enabling algorithmic optimizations for distance-related problems," 2015.

[23] L. A. Maciel, M. A. Souza, and H. C. de Freitas, "Reconfigurable FPGA-based K-means/K-modes architecture for network intrusion detection," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 67, no. 8, pp. 1459–1463, Aug. 2020.

[24] L. A. Dias, J. C. Ferreira, and M. A. C. Fernandes, "Parallel implementation of K-means algorithm on FPGA," IEEE Access, vol. 8, pp. 41071–41084, 2020.

[25] Z. He, Z. Wang, and G. Alonso, "BiS-KM: Enabling any-precision K-means on FPGAs," in Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), Seaside, CA, USA, Feb. 2020, pp. 233–243.

[26] L. Du, Y. Du, and M.-C. F. Chang, “A reconfigurable 64-dimension K-means clustering accelerator with adaptive overflow control,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 67, no. 4, pp. 760–764, Apr. 2020.

[27] B. Morcillo, D. Báscones, C. González, J. M. Mendías, and D. Mozos, “Parametric pipelined K-means implementation for hyperspectral processing on spacecraft embedded FPGA,” IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 17, pp. 15927–15941, May 2024.

[28] J. Han et al., “A technical overview of AV1,” Proceedings of the IEEE, vol. 109, no. 9, pp. 1435–1462, Sep. 2021.

[29] W. Pu et al., “Palette mode coding in HEVC screen content coding extension,” IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 6, no. 4, pp. 420–432, Dec. 2016.

[30] D. Arthur and S. Vassilvitskii, “K-means++: The advantages of careful seeding,” in Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), New Orleans, LA, USA, Jan. 2007, pp. 1027–1035.

[31] X. Xu et al., "Intra block copy in HEVC screen content coding extensions," IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 6, no. 4, pp. 409–419, Dec. 2016.

[32] B. Bross et al., "Overview of the versatile video coding (VVC) standard and its applications," IEEE Transactions on Circuits and Systems for Video Technology, vol. 31, no. 10, pp. 3736–3764, Oct. 2021.

[33] D. Grois, T. Nguyen, and D. Marpe, "Coding efficiency comparison of AV1/VP9, H.265/MPEG-HEVC, and H.264/MPEG-AVC encoders," in Proc. Picture Coding Symposium (PCS), Nuremberg, Germany, Dec. 2016, pp. 1–5.

[34] Y. Netzer et al., "Reading digits in natural images with unsupervised feature learning," in Proc. NIPS Workshop, Granada, Spain, Dec. 2011.

[35] P. Nair and K. N. Chaudhury, "Fast high-dimensional bilateral and nonlocal means filtering," IEEE Transactions on Image Processing, vol. 28, no. 3, pp. 1470–1481, Mar. 2019.

[36] G. Di Fatta and D. Pettinger, "Dynamic load balancing in parallel kd-tree K-means," in Proc. International Conference on Computer and Information Technology (ICCIT), Bradford, U.K., Jun. 2010.

[37] M. Shindler, A. Wong, and A. W. Meyerson, "Fast and accurate K-means for large datasets," in Proc. Advances in Neural Information Processing Systems (NIPS), 2011, pp. 2375–2383.

[38] A. Likas, N. Vlassis, and J. J. Verbeek, "The global K-means clustering algorithm," Pattern Recognition, vol. 36, no. 2, pp. 451–461, 2003.

[39] L. Jing, M. K. Ng, and J. Z. Huang, "An entropy weighting K-means algorithm for subspace clustering of high-dimensional sparse data," IEEE Transactions on Knowledge and Data Engineering, vol. 19, pp. 1026–1041, 2007.

[40] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[41] G. Chen, Y. Ding, and X. Shen, "Sweet KNN: An efficient KNN on GPU," in Proc. IEEE International Conference on Data Engineering (ICDE), San Diego, CA, USA, Apr. 2017, pp. 621–632.

[42] C. Chung and Y. Wang, "Hadoop cluster with FPGA-based hardware accelerators for K-means," in Proc. IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), Taipei, Taiwan, Jun. 2017, pp. 143–144.

[43] J. S. S. Kutty, F. Boussaid, and A. Amira, "A high speed configurable FPGA architecture for K-mean clustering," in Proc. IEEE International Symposium on Circuits and Systems (ISCAS), Beijing, China, May 2013.

[44] G. Bjontegaard, "Calculation of average PSNR differences between RD curves," ITU-T, Geneva, Switzerland, Document VCEG-M33, Apr. 2001.

[45] P. Sirait and A. M. Arymurthy, "Cluster centres determination based on kd tree in K-means clustering," in Proc. DFMA, 2010.