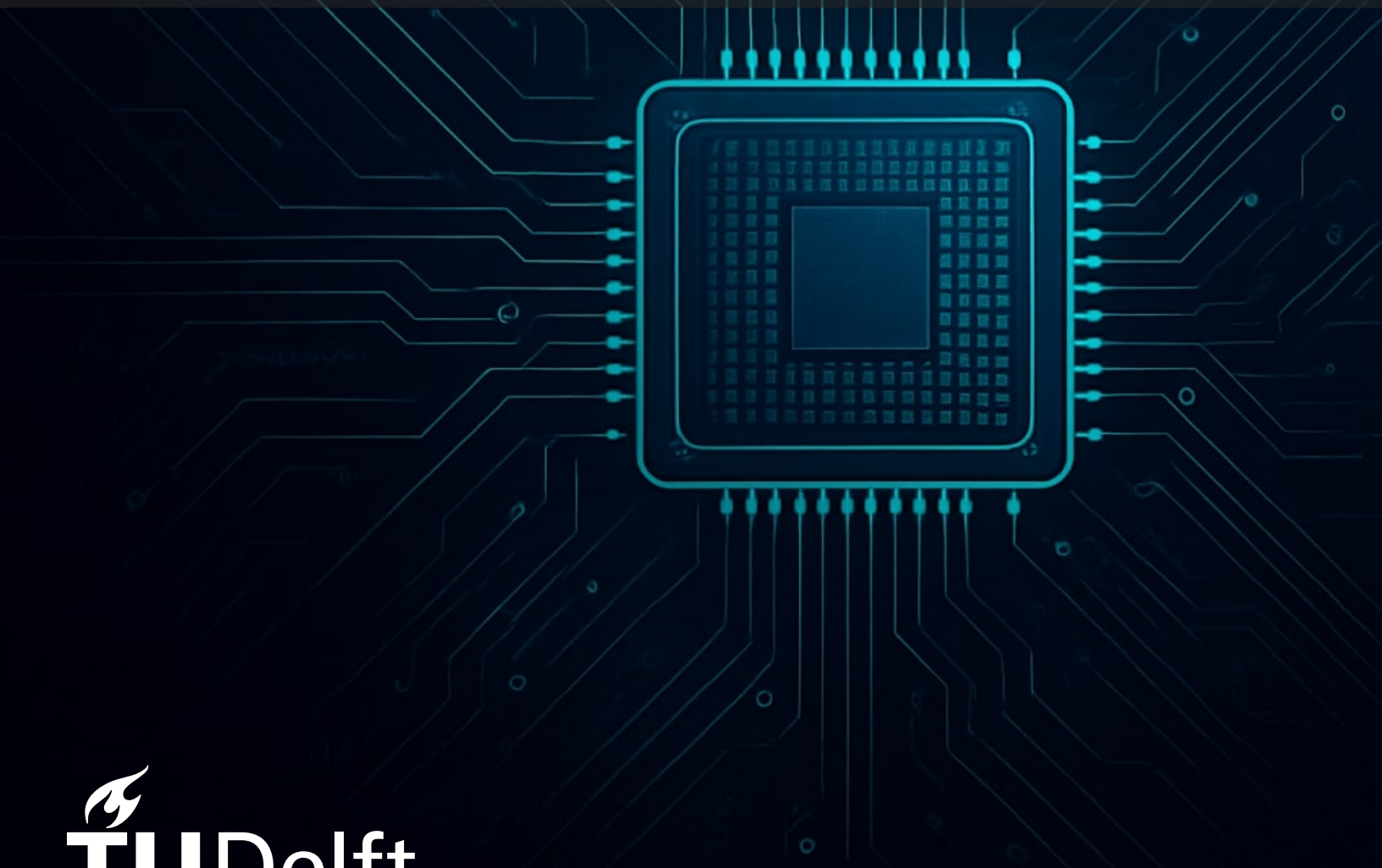
AeDAM: An Event-Driven Architecture Mapping exploration tool

CESE5000 Thesis Project

Naga Subhash Malladi



TU Delft

AeDAM: An Event-Driven Architecture Mapping exploration tool

by

Naga Subhash Malladi

Student number: 6006388

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday June 30, 2025 at 08:30 AM.

Thesis Advisor:	Prof. Dr. Said Hamdioui *Chair Professor of Dependable and Emerging Computer Technologies* *Head of Computer Engineering Laboratory (CE-Lab), TU Delft*
Daily Supervisor:	Dr. Rajendra Bishnoi *Assistant Professor, Computer Engineering (CE) Group, TU Delft*
External Committee Member:	Dr. R.R. Venkatesha Prasad *Associate Professor, Networked Systems (NS) Group, TU Delft*
External Supervisor:	Kanishkan Vadival *Hardware Efficient AI Group, IMEC*
Project Duration:	September 2024 – June 2025
Faculty:	Electrical Engineering, Mathematics and Computer Science Delft University of Technology

Cover: DALL·E (OpenAI) | Style: TU Delft Report Style, modified by Daan Zwaneveld

TU Delft

Preface

This thesis, submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer and Embedded Systems Engineering (CESE) at the Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS), Delft University of Technology, presents the culmination of research conducted in collaboration with IMEC Eindhoven from September 2024 to June 2025. This thesis work was undertaken under the guidance of Dr. Rajendra Bisnoi and Kanishkan Vadivel (IMEC), with Prof. Said Hamdioui serving as the thesis advisor.

The core objective of this research is to address the fundamental limitations of existing design space exploration tools when applied to event-driven accelerator architectures. This is achieved through the development of AeDAM, a specialized design space exploration framework that enables systematic optimization of event-driven neural network accelerators through intelligent mapping generation and analytical cost modeling.

This project represents my first major research endeavor, and it has been an immensely enriching experience. It has allowed me to delve into various critical aspects of computer architecture research, such as accelerator design methodologies, performance modeling and analysis techniques, memory hierarchy optimization strategies, energy-efficient computing paradigms, and systematic evaluation frameworks for domain-specific architectures.

Throughout this process, I have benefited greatly from extensive discussions with Kanishkan Vadivel and Dr. Rajendra Bisnoi, who have provided invaluable insights and guidance on both architectural and implementation issues.

Naga Subhash Malladi
Delft, June 2025

Acknowledgement

I am deeply grateful for the unwavering support and encouragement from my family — my parents, sister, cousin and friends — throughout my master's journey. Their belief in me has been truly invaluable.

I would like to extend my sincerest gratitude to my supervisors, Dr. Rajendra Bisnoi and Kanishkan Vadivel, for their expert guidance, insightful feedback, and continuous encouragement. Their mentorship has been instrumental in shaping this thesis.

I am also thankful to my colleagues at IMEC for their generous support and collaboration.

Special thanks to Prof. Said Hamdioui for his valuable feedback as my thesis advisor, particularly during the green light presentation.

To everyone mentioned, and to all who have supported me in ways big and small — thank you. Your contributions have been essential to the completion of this thesis.

Summary

Design space exploration (DSE) frameworks have become indispensable tools for optimizing neural network accelerator architectures, enabling systematic evaluation of hardware configurations across diverse computational paradigms. However, the proliferation of event-driven and neuromorphic computing architectures has exposed fundamental limitations in existing DSE methodologies, which remain exclusively focused on synchronous, frame-based processing models that cannot adequately capture the unique computational characteristics inherent to event-driven systems.

This thesis presents AeDAM (An Event-Driven Architecture Mapping exploration tool), a specialized design space exploration framework engineered specifically for event-driven accelerator architectures. Unlike conventional DSE tools such as ZigZag that operate within traditional synchronous execution paradigms, AeDAM addresses the critical research gap in modeling event-driven execution patterns that characterize neuromorphic and edge computing applications.

AeDAM's architectural foundation comprises three principal innovations: intelligent event-driven mapping space generation through transformation of traditional Loop Order Memory Access (LOMA) scheduling techniques, a novel analytical cost estimation model specifically extended for asynchronous execution paradigms in dense neural networks, and a unified exploration framework that integrates hardware configuration analysis with specialized event-driven mapping strategies. The framework extends selected components of the established ZigZag memory-centric exploration framework while introducing specialized extensions for event-driven mappings and event-driven computation paradigms.

Comprehensive evaluation demonstrates AeDAM's superior performance characteristics compared to state-of-the-art exploration tools. The framework achieves approximately 2.5× faster exploration times compared to ZigZag for VGGNet neural network, representing a substantial improvement in computational efficiency. AeDAM's optimal mapping configurations deliver significant latency improvements across the VGGNet neural network, achieving performance improvements ranging from 13% to 52% compared to ZigZag-generated solutions. The framework's efficacy is further validated through comprehensive case studies using the SENECA neuromorphic architecture, demonstrating practical applicability for event-driven accelerator design optimization.

These results establish AeDAM as an essential advancement in accelerator design methodology, providing the research community with specialized capabilities for systematic exploration of event-driven architectures that cannot be adequately modeled using existing synchronous design space exploration frameworks. “

Contents

List of Figures

List of Tables

1

Introduction

1.1. Motivation

Deep Neural Networks (DNNs) are pivotal in numerous applications, spanning image recognition, natural language processing, and increasingly, edge-based inference tasks. Despite their versatility, DNNs present significant computational challenges due to intensive memory and processing demands, particularly for deployment on energy-constrained edge devices. To address these workloads efficiently, specialized neural network accelerators employing diverse dataflow strategies—weight-stationary (WS), output-stationary (OS), and input-stationary (IS)—are actively explored in the literature [1].

However, the intensive memory and processing demands of DNNs continue to pose substantial challenges for energy-constrained edge deployments, motivating the exploration of fundamentally different computational paradigms. The emergence of neuromorphic computing has introduced event-driven execution models that draw inspiration from biological neural systems [2], [3]. For applications characterized by irregular data patterns, temporal dynamics, and significant sparsity, event-driven execution offers substantial efficiency gains by triggering computation only when new data is available, significantly reducing redundant memory accesses and computations by skipping ineffectual operations [4], [5]. This approach directly addresses the energy challenges of DNN deployment by eliminating unnecessary computations, making it particularly effective for edge applications requiring real-time processing with stringent power constraints [6].

The realization of efficient event-driven accelerators requires systematic design space exploration to identify optimal mappings and architecture configurations. However, existing design optimization methodologies predominantly focus on synchronous and frame-based execution models that assume regular execution patterns. These conventional approaches are not inherently designed to exploit the advantages of event-driven paradigms, creating a critical gap in the design optimization tools available for next-generation computing architectures.

1.2. Problem Statement

Contemporary design space exploration frameworks have established the foundation for systematic accelerator optimization. Timeloop [7] provides analytical modeling for dense tensor accelerators with comprehensive energy and performance evaluation capabilities. ZigZag [8] offers memory-centric exploration focusing on optimal memory hierarchy utilization through flexible mapping representations. Sparseloop [9] addresses sparse tensor accelerator modeling with statistical characterizations of sparsity patterns. Recent developments include CMDS [10] for cross-layer dataflow optimization, and advanced frameworks like STREAM [11] for multicore architecture explorations.

However, these frameworks share a fundamental limitation: they are exclusively designed for synchronous, frame-based processing models and cannot adequately capture the unique computational characteristics inherent to event-driven systems [12], [13]. This incompatibility creates research gaps that prevent effective optimization of event-driven accelerators.

Current frameworks inadequately handle sparsity exploitation in event-driven contexts, missing opportunities for computational and memory efficiency improvements that are fundamental to event-driven architectures. While existing tools can model static sparsity patterns, they cannot capture the dynamic sparsity characteristics and temporal

irregularities that define event-driven execution, where computational activity varies significantly based on input data patterns and temporal correlation structures.

The vast design space associated with multicore architectures remains insufficiently explored under event-driven conditions, where coordination and synchronization requirements differ significantly from traditional approaches. Event-driven multicore systems require specialized communication protocols and resource allocation strategies that cannot be adequately modeled using existing synchronous design space exploration methodologies.

Furthermore, no unified, systematic exploration framework currently exists that fully captures the complexities and advantages of event-driven execution at both intra-core and inter-core levels. The asynchronous nature of event-driven processing, combined with its dependency on input characteristics and temporal patterns, requires a different modeling approach that existing frameworks cannot provide.

These limitations necessitate the development of specialized design space exploration frameworks specifically engineered for event-driven accelerator optimization, capable of modeling asynchronous execution patterns and dynamic resource utilization characteristics that define event-driven computational paradigms.

1.3. Research Contributions

To address these critical gaps, this thesis introduces An Event-Driven Architecture Mapping (AeDAM), a design space exploration framework developed specifically for event-driven accelerator architectures. The framework leverages and extends the intra-core mapping capabilities of the ZigZag exploration tool [8] to accurately model and evaluate event-driven dataflows. The primary contributions of this work are as follows.

Event-Driven Mapping Space Generation: AeDAM provides mapping space exploration capabilities specific to event-driven architectures through intelligent transformation of traditional mapping methodologies. The framework incorporates advanced heuristic search methods that render the resulting mapping space event-driven in nature and achieve improved energy-delay product optimization.

Event-Driven Analytical Cost Model: AeDAM introduces a novel analytical cost estimation tool specifically designed for event-driven execution paradigms in dense neural network processing. The tool extends word access calculation methods that accurately model the unique data access behavior of event-driven dataflows, enabling precise performance evaluation of event-driven accelerator architectures.

Unified Design Space Exploration Framework: The framework integrates hardware configuration analysis with specialized event-driven mapping strategies, providing comprehensive design space coverage that yields overall energy and latency optimization, optimized mapping schedules, and detailed resource utilization statistics.

By addressing these fundamental gaps, this thesis lays the groundwork for systematically realizing the full potential of event-driven accelerators, establishing the foundation for future research and the design of optimized event-driven accelerators targeting energy-constrained, high-performance edge computing applications [2], [3].

1.4. Structural Outline

This thesis is organized to systematically present the development and evaluation of the AeDAM framework:

Chapter 2: Background Study establishes the theoretical foundation by contrasting frame-based and event-driven execution paradigms, examining existing accelerator architectures, and analyzing current design space exploration methodologies to highlight the unique requirements of event-driven systems.

Chapter 3: Related Works provides a comprehensive analysis of existing design space exploration frameworks, identifying their strengths, limitations, and the specific research gap that motivates the development of AeDAM for event-driven accelerator optimization.

Chapter 4: AeDAM Framework Design presents the architectural foundation and core innovations of the AeDAM framework, including the event-driven mapping transformation methodology and the specialized analytical cost modeling approach that enables systematic exploration of asynchronous execution paradigms.

Chapter 5: AeDAM Implementation details the technical implementation of the framework, including input modeling and configuration management, event-driven mapping space generation algorithms, analytical cost models for energy and performance estimation, and the unified exploration infrastructure that integrates hardware configuration analysis with specialized event-driven mapping strategies.

Chapter 6: Results and Analysis demonstrates AeDAM's capabilities through comprehensive case studies using representative neural network architectures and the SENECA neuromorphic platform [4], validating the framework's effectiveness through systematic comparison with state-of-the-art exploration tools and practical applicability assessment for event-driven accelerator design optimization.

Chapter 7: Conclusion and Future Work summarizes key contributions and evaluation results, demonstrating AeDAM's superior performance characteristics including approximately 2.5× faster exploration times and significant latency improvements ranging from 13% to 52% compared to existing solutions, while outlining future research directions for advancing event-driven accelerator design methodology and extending capabilities to spiking neural networks and multicore architectures.

2

Background Study

2.1. Frame-based Execution

Frame-based execution represents a computational paradigm where processing operations are performed on discrete, complete data blocks rather than individual data elements. In this execution model, a frame represents a complete data block that must be processed as a single unit. The execution process follows a deterministic sequence where an entire frame is loaded from external storage into local memory, computational operations are applied uniformly across the complete data structure, and results are written back as a complete unit. This approach fundamentally differs from element-by-element processing by requiring complete data availability before computation begins.

Frame-based execution exhibits distinctive operational characteristics that provide substantial system advantages. The batch processing nature ensures computational kernels operate on complete datasets, enabling algorithms that require global knowledge of data structures. The temporal locality inherent in frame-based processing provides significant memory system benefits, as entire frames loaded into local memory create predictable access patterns that optimize memory hierarchies and reduce irregular access overhead. Memory access characteristics enable optimal bandwidth utilization through burst transactions that amortize setup costs across large data volumes, while sequential access patterns align with contemporary memory system designs. Computational efficiency emerges through regular frame structures that enable highly optimized kernels applied uniformly across data structures, with batch processing providing optimization opportunities unavailable in streaming models.

Frame-based execution systems demonstrate high performance predictability due to regular frame processing operations, with fixed-size data structures enabling accurate prediction and optimization of processing time and resource requirements. Throughput characteristics scale effectively with available computational resources through efficient utilization of parallel processing elements and memory bandwidth. However, frame-based execution imposes limitations including latency introduction due to complete frame availability requirements before processing begins, making it unsuitable for applications requiring immediate response to individual data elements. The fixed-size nature may result in inefficient resource utilization when processing data that does not align with frame boundaries, particularly for applications with variable data sizes or irregular access patterns.

2.2. Frame-Based Accelerators

Frame-based accelerators represent specialized computing architectures designed to exploit computational and memory access characteristics inherent in frame-structured data processing. These accelerators optimize their entire design around predictable access patterns and computational regularity that characterize frame-based workloads, typically comprising dedicated memory hierarchies for frame buffering, parallel processing elements configured for frame-structured computations, and dataflow mechanisms exploiting temporal and spatial locality. The Eyeriss architecture exemplifies this approach through systematic exploitation of frame-structured neural network computations [1], implementing spatially distributed processing where individual elements operate on frame regions in parallel while maintaining frame-level coordination. Representative implementations demonstrate key architectural principles including exploitation of data reuse through spatial and temporal locality, as seen in systolic array designs where data flows through processing arrays in coordinated manners, passing intermediate results directly

between elements without external memory storage.

Frame-based accelerators demonstrate significant performance and efficiency advantages over general-purpose processors for applications aligning with their design assumptions. Performance benefits stem from elimination of overhead associated with irregular memory access patterns and efficient utilization of parallel processing resources enabled by frame structure regularity. Energy efficiency benefits arise from multiple sources including reduced memory access energy through bulk transfers, minimized control overhead through regular computational patterns, and optimized datapath utilization through frame-structured parallelism. Contemporary frame-based accelerators continue evolving through advances in memory technology, processing element design, and architectural optimization techniques, achieving excellent performance for dense data processing applications such as computer vision and high-throughput machine learning workloads. However, for energy-constrained edge applications characterized by sparse data patterns and stringent power budgets, the requirement to process complete frames regardless of data sparsity can lead to significant energy waste. In such scenarios, event-driven execution paradigms that process only active data elements can provide substantial energy efficiency advantages, particularly for neuromorphic computing and real-time reactive systems where input sparsity is high and immediate response to individual events is critical.

2.3. Event-Driven Execution

Event-driven execution represents a fundamentally different computational paradigm where computation is triggered by individual data element arrival rather than complete data blocks. Each input element, referred to as an event, initiates specific computational operations performed immediately upon occurrence. The defining characteristic is reactive processing where operations occur asynchronously in response to data availability rather than predetermined schedules based on complete dataset assembly. This paradigm eliminates requirements for complete data availability before computation begins, with systems responding to each data element individually and performing all associated computations before proceeding to subsequent events. The computational model emphasizes sparsity exploitation where processing resources are consumed only when events actually occur [4], [5].

Event-driven execution exhibits distinctive operational characteristics differentiating it from batch and frame-based processing models. Reactive processing ensures computational operations commence immediately upon event arrival without requiring coordination with other events or waiting for additional data elements, enabling low latency for individual event processing while adapting dynamically to varying event arrival rates. Memory access patterns typically exhibit irregular and unpredictable characteristics as events arrive in arbitrary orders and trigger access to different memory locations, presenting challenges for memory system optimization but enabling processing of data streams with complex temporal dependencies [14]. The control flow is determined by event sequence and timing rather than predetermined program structures, requiring sophisticated event handling mechanisms while enabling responsive processing of asynchronous data streams.

Event-driven execution enables superior temporal processing capabilities by eliminating inherent latency associated with frame assembly and bulk processing, with individual events receiving immediate processing attention upon arrival. Temporal precision allows accurate preservation of timing relationships between individual events, maintaining precise timing characteristics throughout processing pipelines. However, event-driven execution imposes limitations including irregular memory access patterns that can result in poor cache utilization and increased memory access latency compared to predictable frame-based patterns. Coordination and synchronization requirements introduce complexity in multi-processing environments where events must be processed while maintaining proper ordering and dependency relationships. Efficiency benefits are highly dependent on input data stream sparsity characteristics, with dense data patterns potentially achieving better performance through frame-based processing methods.

2.4. Event-Driven Accelerators

Event-driven accelerators represent specialized neuromorphic computing architectures designed to implement and optimize event-driven execution paradigms, embodying principles of asynchronous, sparse computation through hardware mechanisms that respond efficiently to individual events while minimizing energy consumption during inactivity periods. The SENECA neuromorphic architecture exemplifies this approach through hierarchical control systems combining flexible and efficient processing elements [4], implementing dual-controller architectures with RISC-V processors for flexible event preprocessing and custom loop buffer controllers for efficient event execution. Intel's Loihi processor demonstrates scalability through distributed architectures employing asynchronous event-driven communication via address-event representation packets [15]. Advanced implementations such as ISOSceles address inter-layer pipelining challenges in sparse convolutional neural networks through novel dataflow approaches that consume and produce activations in consistent order [16]. Design principles incorporate asynchronous pro-

cessing as core elements that eliminate energy overhead associated with global clock distribution while enabling processing elements to operate according to local event timing requirements.

Event-driven accelerators achieve computational efficiency through various techniques maximizing computational sparsity benefits, including dynamic power gating of inactive processing elements, adaptive voltage and frequency scaling based on event arrival rates, and sophisticated event scheduling algorithms minimizing processing overhead. Memory hierarchy optimization focuses on minimizing energy costs of event-related memory accesses while maintaining flexibility required for diverse event processing patterns through distributed memory systems with local storage for frequently accessed data and sophisticated caching mechanisms exploiting temporal locality in event patterns. Recent advances in event-based neural network optimization have demonstrated comprehensive design space exploration techniques that systematically evaluate architectural modifications and mapping strategies to optimize energy efficiency and latency characteristics for neuromorphic workloads [5].

2.5. Difference b/w the event based and frame based execution

Here are the major differences of the event-based and frame-based executions as shown in the table 2.1

The analysis of event-based and frame-based execution paradigms reveals that each approach offers distinct advantages depending on application characteristics and performance requirements. Event-driven execution demonstrates superior efficiency for sparse data applications through selective processing that eliminates unnecessary computations, resulting in reduced energy consumption and minimal latency for individual events, making it particularly suitable for neuromorphic computing and real-time reactive systems. Conversely, frame-based execution achieves optimal performance for dense data processing scenarios by exploiting predictable memory access patterns and regular computational structures, delivering high throughput and performance predictability essential for computer vision, signal processing, and machine learning applications. The selection between these paradigms should be guided by specific workload characteristics, with event-driven approaches favored for sparse, latency-sensitive applications and frame-based execution preferred for dense, high-throughput computational tasks.

2.6. Process of Exploration

Design Space Exploration in the context of hardware accelerators refers to the systematic investigation of how computational workloads can be optimally mapped and executed across available hardware resources to maximize performance, minimize energy consumption, and achieve efficient utilization of processing elements, memory hierarchies, and interconnection networks [7], [8]. This process addresses the fundamental challenge of determining optimal distribution and scheduling of computational operations across multiple processing elements while respecting memory bandwidth limitations, timing constraints, and energy efficiency requirements. The exploration encompasses two primary parameter domains: workload characteristics including neural network layer dimensions, data precision requirements, and sparsity patterns, and architectural configurations encompassing processing element array geometries, memory hierarchy capacities, bandwidth allocations, and dataflow paradigm selections. The exploration process becomes particularly critical for specialized hardware architectures where the vast design space of possible configurations requires systematic evaluation to identify optimal solutions that deliver superior performance for specific application requirements while maintaining feasibility within implementation constraints.

The core technical mechanisms of exploration center on computational loop mapping and hardware configuration optimization, where complex nested loop structures characteristic of deep learning workloads must be carefully assigned to physical hardware resources. Loop blocking techniques partition large iterations into smaller blocks that align with memory hierarchy capacity constraints, while spatial unrolling distributes loop iterations across multiple processing elements for parallel execution, and temporal scheduling determines the sequence of operations to maximize data reuse opportunities and minimize memory access overhead. Contemporary exploration approaches employ intelligent search algorithms including evolutionary methods, simulated annealing, and reinforcement learning techniques to navigate large design spaces efficiently [18], with multicore accelerator architectures introducing additional complexity requiring sophisticated performance estimation techniques that account for inter-core communication overhead and load balancing considerations [19]. Hardware configuration exploration systematically evaluates critical architectural parameters including processing element array dimensionalities, memory subsystem organizations with buffer capacity allocation and bandwidth provisioning, precision configurations for weights and activations, and network-on-chip topologies that collectively determine system performance, energy efficiency, and area characteristics. Heterogeneous multicore accelerator designs further complicate the exploration process by introducing architectural diversity within individual systems, requiring specialized mapping techniques that exploit unique capabilities of different core types while maintaining overall system coherence [11].

Table 2.1: Comparison of Event-Based and Frame-Based Execution Characteristics

Characteristic	Event-Based Execution	Frame-Based Execution
Processing Model	Individual events processed immediately upon arrival	Complete data blocks processed as atomic units
Data Requirements	Single event with sufficient information for independent processing	Complete frame must be available before processing begins
Latency	Minimal latency for individual events; immediate processing response	Higher latency due to frame assembly and bulk processing requirements
Memory Access Patterns	Irregular and unpredictable; random access based on event characteristics	Regular and predictable; sequential access patterns optimized for bulk transfers
Resource Utilization	On-demand activation; resources consumed only when events occur	Continuous processing; resources maintained active throughout frame processing duration
Sparsity Exploitation	Excellent; processes only active events, avoiding computation on inactive elements	Limited; must process entire data structures regardless of information content
Temporal Processing	Precise timing preservation; maintains exact temporal relationships between events	Quantized timing; temporal information constrained by frame boundaries
Control Flow	Reactive; determined by event arrival sequence and timing	Predetermined; follows deterministic sequence of operations
Parallelization	Complex coordination required; concurrent event processing with independence preservation	Straightforward spatial parallelism; uniform distribution across processing elements
Energy Efficiency	Variable; highly dependent on sparsity characteristics and event arrival patterns	Consistent; optimized for regular computational patterns and bulk operations
Throughput	Variable; adapts to event arrival rates but may suffer from irregular patterns	High and predictable; scales effectively with available computational resources
Cache Performance	Poor cache utilization due to irregular access patterns	Excellent cache performance due to predictable sequential access
Memory Bandwidth	Inefficient; frequent small accesses with unpredictable timing	Efficient; burst transfers that amortize setup costs across large data volumes
Computational Predictability	Low; processing time varies with event characteristics and arrival patterns	High; processing time and resource requirements accurately predictable
Synchronization Requirements	Complex; asynchronous event coordination and dependency management	Simplified; bulk processing enables straightforward coordination mechanisms
Optimal Applications	Sparse data streams, neuromorphic computing, real-time reactive systems	Dense data processing, computer vision, signal processing, machine learning
Primary Advantages	• Low latency for individual events • Excellent sparsity exploitation • Real-time responsiveness • Adaptive resource allocation	• High computational throughput • Predictable performance characteristics • Efficient memory bandwidth utilization • Straightforward parallelization
Primary Limitations	• Irregular memory access patterns • Poor cache utilization • Complex synchronization requirements • Performance dependent on data sparsity	• Frame assembly latency • Inefficient for sparse data • Fixed processing granularity • Poor responsiveness to individual elements
Hardware Implementations	Event-driven accelerators (e.g., SENECA [4], Intel Loihi [15]), neuromorphic architectures	Frame-based accelerators (e.g., Eyeriss [1]), conventional GPU/CPU architectures like Google TPU [17]

The exploration process ultimately enables the systematic evaluation of different dataflow strategies that fundamentally influence data movement patterns and computational scheduling across accelerator architectures. Weight-stationary, input-stationary, output-stationary, and event-driven approaches each present distinct trade-offs for different workload characteristics, with dataflow selection directly impacting memory bandwidth requirements, energy efficiency, processing element utilization, and overall system performance. Event-driven dataflows, exemplified by SENECA, enable efficient exploitation of computational sparsity through selective processing of active data elements [4], while advanced implementations such as ISOSceles demonstrate dataflow optimization through novel approaches that address inter-layer pipelining challenges [16]. Current exploration capabilities for frame-based accelerators are provided through established tools including Timeloop [7], ZigZag [8], ConvFusion [20], Stream [11] and Sparseloop [9] for sparse tensor acceleration, yet comprehensive support for event-driven execution paradigms within current design-space exploration frameworks remains notably lacking, representing a significant research gap that this work addresses through the development of specialized exploration methodologies for event-driven accelerator architectures.

2.6.1. Dataflow Strategies for CNN Accelerators

The selection of appropriate dataflow strategies represents a fundamental design decision that directly influences data movement patterns, computational scheduling, energy efficiency, and overall system performance in CNN accelerators. These strategies determine which data elements remain stationary in processing elements while others flow through the computational array, thereby establishing the foundation for memory access optimization and resource utilization characteristics.

Understanding the dimensional parameters is essential for analyzing these dataflow strategies. The notation IX and IY represents input feature map height and width dimensions, while FX and FY denote filter kernel height and width dimensions. The parameter C indicates the number of input channels, K represents the number of output channels or filters, and OX and OY specify output feature map height and width dimensions. These parameters determine how computational loops are organized and mapped across processing element arrays in different dataflow configurations.

Weight Stationary Dataflow

Stationary Element: Filter weights remain permanently positioned within processing elements [1].

Spatial and Temporal Unrolling: Spatial unrolling operates across K, FX, and FY dimensions, while temporal unrolling addresses IX, IY, and C dimensions.

Primary Data Reuse: This strategy maximizes weight reuse by maintaining filters stationary in processing elements, eliminating repeated weight loading operations.

Memory Bandwidth Requirements: High input and output bandwidth demands characterize this approach, though minimal weight traffic occurs after initial loading phases.

Energy Efficiency: Significant energy reductions result from eliminated repeated weight fetches, making this approach particularly efficient for weight-heavy operations.

Processing Element Utilization: High utilization occurs when filter dimensions align effectively with processing element array geometry.

Buffer Requirements: Large weight buffers are essential, alongside moderate input and output storage allocations.

Control Complexity: Moderate complexity characterizes the control mechanisms due to regular data movement patterns.

Data Movement Pattern: Weights load once during initialization, while inputs and outputs stream continuously through processing elements.

Implementation Examples: Systolic arrays with weight preloading architectures represent typical implementations.

Scalability: Excellent scaling with filter complexity, though weight storage capacity limitations impose constraints.

Primary Advantages: Minimized repeated weight fetches, reduced weight memory bandwidth, efficient depthwise operation support, and predictable access patterns.

Primary Limitations: High input and output bandwidth requirements, substantial weight storage needs, poor utilization with small filters, and limited flexibility for diverse layer configurations.

Optimal Applications: Depthwise separable convolutions, filter-heavy layers, and weight-dominant computational tasks.

Input Stationary Dataflow

Stationary Element: Input feature maps remain stationary within the processing array [7].

Spatial and Temporal Unrolling: Spatial unrolling addresses C, IY, and IX dimensions, while temporal unrolling covers K, FX, and FY dimensions.

Primary Data Reuse: Input feature map reuse maximization occurs through spatial distribution mechanisms across processing elements.

Memory Bandwidth Requirements: High weight and partial sum bandwidth demands emerge, while input activation traffic experiences reduction.

Energy Efficiency: Input activation access energy minimization makes this approach suitable for activation-heavy computational patterns.

Processing Element Utilization: High efficiency occurs when input dimensions align with spatial unrolling capabilities.

Buffer Requirements: Large input buffers are necessary, accompanied by moderate weight and output storage requirements.

Control Complexity: Moderate complexity results from predictable access patterns and established coordination mechanisms.

Data Movement Pattern: Inputs distribute spatially across the array, while weights and outputs flow temporally through processing elements.

Implementation Examples: Spatial architectures with input broadcasting mechanisms exemplify this approach.

Scalability: Effective scaling with input dimensions, though spatial distribution capabilities and coordination complexity impose constraints.

Primary Advantages: Minimized input activation fetches, reduced input memory bandwidth, effective spatial locality exploitation, and efficient input distribution.

Primary Limitations: High weight and partial sum bandwidth demands, large input buffer requirements, complex spatial distribution mechanisms, and limited temporal reuse opportunities.

Optimal Applications: Layers with large input feature maps, activation-heavy computational patterns, and broadcast-amenable operations.

Output Stationary Dataflow

Stationary Element: Partial sums accumulate locally as stationary elements within processing elements [1].

Spatial and Temporal Unrolling: Spatial unrolling operates across K, OX, and OY dimensions, while temporal unrolling addresses C, FX, and FY dimensions.

Primary Data Reuse: Partial sum reuse maximization occurs through local accumulation mechanisms within processing elements.

Memory Bandwidth Requirements: High weight and input bandwidth demands characterize this approach, while partial sum movement remains minimal.

Energy Efficiency: Lowest partial sum movement energy results from local accumulation, making this optimal for accumulation-intensive layers.

Processing Element Utilization: Consistent utilization across diverse layer geometries provides reliable performance characteristics.

Buffer Requirements: Large input and weight buffers are necessary, while minimal partial sum storage suffices due to local accumulation.

Control Complexity: Low complexity emerges from straightforward accumulation flow patterns.

Data Movement Pattern: Inputs and weights flow through processing elements while partial sums remain locally accumulated.

Implementation Examples: The Eyeriss row-stationary implementation represents a prominent example of this approach.

Scalability: Excellent scalability across diverse network architectures provides broad applicability.

Primary Advantages: Minimal partial sum movement, reduced accumulation overhead, general-purpose applicability, and simplified control flow.

Primary Limitations: High weight and input bandwidth requirements, large register file needs, potential memory wall effects, and reduced optimization flexibility.

Optimal Applications: General CNN layers, accumulation-intensive operations, and balanced computational workloads.

Event-Driven Dataflow

Stationary Element: Non-zero activations serve as computational triggers rather than maintaining traditional stationary elements [4].

Spatial and Temporal Unrolling: Spatial unrolling addresses K, FX, and FY dimensions, while temporal unrolling covers IX, IY, and C dimensions.

Primary Data Reuse: Activation sparsity exploitation occurs by processing exclusively non-zero values, eliminating computations on inactive elements.

Memory Bandwidth Requirements: Variable bandwidth utilization based on sparsity characteristics creates irregular access patterns that adapt to data distributions.

Energy Efficiency: Highest potential energy savings with sparse data, though overhead increases with dense data patterns.

Processing Element Utilization: Variable utilization depends significantly on activation sparsity patterns and event arrival characteristics.

Buffer Requirements: Dynamic buffer allocation, event queues, and specialized sparse data structures accommodate asynchronous computational flows.

Control Complexity: High complexity results from irregular execution patterns and sophisticated event handling overhead.

Data Movement Pattern: Event-triggered data movement creates sparse access patterns that differ fundamentally from regular dataflow approaches.

Implementation Examples: The SENECA neuromorphic accelerator represents advanced implementations of event-driven processing [4].

Scalability: Scalability depends on sparsity characteristics and event handling efficiency, with performance varying significantly across different data patterns.

Primary Advantages: Excellent sparsity exploitation, processing only active data elements, significant energy savings potential, and real-time responsiveness capabilities.

Primary Limitations: Irregular execution patterns, complex control overhead, performance dependency on data sparsity, and challenging prediction and optimization requirements.

Optimal Applications: Sparse neural networks, neuromorphic computing systems, and activation-sparse computational scenarios.

The comparative analysis of these dataflow strategies reveals that each approach offers distinct advantages aligned with specific computational characteristics and performance requirements. Weight stationary and input stationary dataflows excel in regular, dense computational scenarios through predictable access patterns and efficient resource utilization, while output stationary approaches provide balanced performance across diverse layer types. Event-driven dataflows demonstrate superior efficiency for sparse computational patterns but introduce complexity

in irregular scenarios. The selection between these strategies should be guided by workload sparsity characteristics, layer dimensions, and performance optimization priorities.

3

Related works

3.1. Timeloop: Systematic Dense Tensor Accelerator Evaluation

Timeloop represents a foundational framework for systematic deep neural network accelerator evaluation, developed by Parashar et al. [7]. The framework employs an analytical approach to model the performance, energy consumption, and area requirements of DNN accelerators through comprehensive design space exploration. Timeloop's primary strength lies in its systematic methodology for evaluating accelerator architectures through detailed modeling of dataflow patterns, memory hierarchies, and computational mappings.

The framework excels in dense tensor modeling, providing accurate energy and performance estimates for traditional DNN workloads. Timeloop integrates seamlessly with Accelergy, an architecture-level energy estimation methodology, enabling comprehensive power analysis across different technology nodes and design configurations. The tool's analytical models have been validated against published accelerator designs, demonstrating accuracy within acceptable error margins for practical design space exploration.

The framework supports various architectural specifications including different dataflows and hardware optimizations for neural network computations. Timeloop has demonstrated effectiveness across multiple case studies and has been adopted as a foundation for extending accelerator design space exploration capabilities. The tool provides comprehensive mapping constraint support and enables systematic evaluation of diverse accelerator configurations for performance optimization.

However, Timeloop's architecture remains fundamentally oriented toward traditional dataflow paradigms and synchronous execution models. The framework lacks capabilities for modeling dynamic, event-driven execution patterns that characterize modern neuromorphic and asynchronous computing systems. While Timeloop can model various mapping strategies and memory hierarchies, it cannot capture the temporal irregularities and asynchronous behaviors that are central to event-driven architectures.

3.2. ZigZag: Memory-Centric Design Space Exploration

ZigZag, developed by Mei et al. [8], introduces a memory-centric approach to DNN accelerator design space exploration that extends traditional mapping techniques with support for uneven spatial and temporal mappings. The framework addresses limitations in existing design space exploration tools by providing flexible mapping representations that capture complex memory access patterns and data reuse opportunities that conventional approaches often miss. ZigZag employs an enhanced memory-centric design space representation based on nested-for-loop formats that decouples operands, memory hierarchy, and mapping scenarios.

The framework implements sophisticated heuristic search strategies including data stationarity maximization and data reuse pruning that navigate the expanded design space efficiently while maintaining computational tractability. ZigZag demonstrates significant improvements over existing frameworks, achieving up to 64% more energy-efficient solutions compared to state-of-the-art approaches. The tool incorporates a loop relevance principle that systematically extracts key information such as memory accesses and required bandwidth from which the hardware cost estimator derives energy and performance values.

ZigZag integrates effectively with ONNX model formats, providing seamless support for modern deep learning workflows and enabling rapid prototyping of accelerator designs. The framework supports both exhaustive and heuristic search methods, reducing the number of mappings to be evaluated by orders of magnitude while maintaining optimality. ZigZag's cost model demonstrates excellent correlation with established frameworks like Timeloop, validating its analytical accuracy for practical design space exploration applications.

Despite its advances in mapping flexibility and search efficiency, ZigZag still operates within traditional synchronous dataflow paradigms and lacks explicit support for sparse tensor acceleration or dynamic, event-driven patterns. The framework cannot model the asynchronous execution characteristics and irregular timing patterns that are fundamental to event-driven accelerator architectures, limiting its applicability to conventional synchronous processing paradigms.

3.3. ConvFusion: Mathematical Modeling for Layer Fusion

ConvFusion, developed by Waeijen et al. [20], models layer fusion in CNNs using mathematical cost models for loop tiling, loop reordering, and memory optimization. The framework addresses the challenge of optimizing fused convolutional layers by providing systematic analysis of data reuse opportunities and memory access patterns across layer boundaries. ConvFusion employs analytical models that predict the impact of different fusion strategies on overall system performance and energy consumption.

The framework demonstrates significant performance improvements, achieving up to 99.75% reduction in external memory accesses in some network configurations [20]. ConvFusion's mathematical models enable precise prediction of memory traffic and computational efficiency for various layer fusion scenarios. The tool systematically evaluates different tiling strategies and loop orderings to identify optimal configurations that minimize data movement while maximizing computational throughput across fused operations.

ConvFusion utilizes a hybrid backend incorporating both Halide and Keras/TensorFlow frameworks, enabling integration with existing deep learning workflows while providing low-level optimization capabilities. The framework has been validated on prominent neural network architectures including ResNet50, VGG16, and InceptionV3, demonstrating consistent performance improvements across diverse computational patterns. The tool provides automated analysis of fusion opportunities and generates optimized implementations for target hardware platforms.

However, ConvFusion operates under assumptions of fixed network topologies and predetermined execution schedules, limiting its flexibility for dynamic computational patterns. The framework does not support sparsity exploitation or event-driven computation paradigms, restricting its applicability to dense, synchronous processing scenarios. ConvFusion cannot model the irregular execution patterns and asynchronous behaviors that characterize event-driven accelerator architectures.

3.4. Sparseloop: Comprehensive Sparse Tensor Accelerator Modeling

Sparseloop, developed by Wu et al. [9], models sparse accelerators using a format-agnostic fibertree abstraction that enables systematic analysis of diverse sparsity patterns and compression formats. The framework extends traditional accelerator modeling to handle sparse tensor operations through comprehensive representation of zero-skipping mechanisms and compressed data formats. Sparseloop addresses the growing importance of sparsity exploitation in neural network acceleration by providing accurate modeling of sparse computational patterns.

The framework demonstrates exceptional performance in sparse tensor modeling, offering up to 2000× simulation speedup with 0.1–8% error compared to hardware implementations [9]. Sparseloop incorporates density prediction capabilities and validation mechanisms that ensure accurate representation of sparse data patterns across different neural network layers. The tool systematically models compression formats, dataflows, and sparse access patterns to provide comprehensive analysis of sparse accelerator designs.

Sparseloop builds directly on Timeloop's foundational framework, adding specialized sparse tensor modeling capabilities while maintaining compatibility with existing analytical models [7]. The framework extends Timeloop's mapping representation to handle sparse data structures and provides integrated density prediction for various sparsity patterns. Sparseloop enables systematic exploration of sparse accelerator designs while leveraging established validation methodologies from the dense tensor modeling domain.

The framework's limitations include static sparsity assumptions that cannot adapt to dynamic sparsity patterns during execution. Sparseloop lacks support for event-driven execution paradigms and multicore modeling capabilities, restricting its applicability to single-core sparse accelerators with predetermined sparsity characteristics. The

framework cannot model the temporal irregularities and asynchronous behaviors that emerge in event-driven sparse processing scenarios.

3.5. Stream: Multi-Core Layer Fusion Exploration

Stream, developed at KU Leuven [11], represents the first framework specifically designed for exploring layer-fused, multi-core accelerator architectures. The framework addresses the growing complexity of heterogeneous accelerator designs by providing systematic analysis of inter-core communication patterns and workload distribution strategies. Stream enables comprehensive evaluation of multi-core configurations that leverage layer fusion techniques to optimize overall system performance and energy efficiency.

The framework demonstrates significant performance improvements through systematic multi-core optimization, achieving up to 2.2× Energy-Delay Product improvements in validated hardware implementations [11]. Stream incorporates sophisticated models for inter-core communication overhead and synchronization requirements, enabling accurate prediction of multi-core accelerator performance. The tool systematically evaluates heterogeneous dataflow accelerator mappings with comprehensive latency and energy models that account for the complexities of multi-core coordination.

Stream builds on ZigZag's memory-centric design space exploration foundation while extending capabilities to multi-core scenarios [8]. The framework has been validated on real hardware implementations, demonstrating practical applicability for multi-core accelerator design optimization. Stream provides integrated analysis of both intra-core and inter-core optimization opportunities, enabling comprehensive exploration of heterogeneous accelerator architectures.

However, Stream operates within traditional synchronous execution paradigms and does not support asynchronous execution or event-driven coordination mechanisms. The framework cannot model the irregular timing patterns and dynamic resource allocation requirements that characterize event-driven multi-core systems. Stream's synchronous modeling approach limits its applicability to event-driven accelerator architectures that require asynchronous inter-core communication and coordination.

3.6. Research Gap: Event-Driven Architecture Exploration

The comprehensive analysis of existing design space exploration frameworks reveals a fundamental limitation in current accelerator modeling capabilities. None of the established tools support modeling of event-driven architectures that require representation of asynchronous execution, irregular timing patterns, and dynamic resource allocation mechanisms. These systems demand specialized modeling approaches that can capture the unique characteristics of event-driven computation while maintaining accuracy for performance and energy estimation.

The development of AeDAM addresses this critical research gap by providing specialized support for event-driven accelerator design space exploration. Built by extending the selected components of ZigZag's memory-centric framework, AeDAM extends traditional mapping methodologies to support temporal dynamics and event-driven computation paradigms. This specialized framework enables systematic exploration of event-driven accelerator architectures that cannot be adequately modeled using existing synchronous design space exploration tools.

Table 3.1: Comparison of Frameworks for AI Accelerator Modelling

Framework	Primary Focus	Event-driven Capabilities	Multi-core Support	Sparsity Support	Framework Relationship
Timeloop	Dense tensor modelling	Traditional	Single core	None	Foundation
Zigzag	Memory centric DSE	Traditional	Single core	None	Foundation
Convfusion	Layer fusion optimisation	Traditional	Single core	None	Foundation
Stream	Multi-core exploration	Traditional	Yes	None	Built on Zigzag
Sparseloop	Sparse tensor modelling	Traditional	Single core	Weight and input sparsity	Built on Timeloop
AeDAM	Event driven exploration	Fully Event driven	Single core	None	Utilise parts of Zigzag

4

AeDAM Framework Design

To address the fundamental limitations of existing design space exploration tools in handling event-driven execution paradigms, this work introduces AeDAM—a specialized framework tailored specifically for event-driven accelerator design and optimization. While conventional DSE frameworks excel at synchronous, frame-based processing analysis [7], [8], they fall short in capturing the unique characteristics and advantages of asynchronous, event-based computation that are increasingly important for neuromorphic and edge computing applications [4], [5]. AeDAM bridges this critical gap through innovative transformation of traditional mapping methodologies, [8] into event-driven compatible configurations, coupled with specialized cost modeling techniques that accurately reflect the operational dynamics of event-driven accelerators. The framework's targeted approach enables the identification of superior design points that fully exploit the benefits of event-driven execution paradigms. The three major contributions of AeDAM are

1. **Event-Driven Mapping Space Generation with Intelligent Search Optimization**
   AeDAM provides sophisticated mapping space exploration capabilities specifically engineered for event-driven architectures through intelligent transformation of traditional mapping methodologies. The framework takes well-known LOMA (Loop Order Memory Access) scheduling techniques as a starting point , applying filtration and conversion techniques to generate exclusively event-driven compatible mappings, performing targeted cost estimation for the optimized configurations. AeDAM incorporates advanced heuristic search methods like intelligent grouping of minimum LPF (Loop unrolling Per Factor) factors [21] that render the resulting mapping space event-driven in nature and achieve improved energy-delay product optimization through this mapping transformation method.
2. **Event-Driven Analytical Cost Model for Dense Neural Networks**
   AeDAM introduces a novel analytical cost estimation tool specifically designed for event-driven execution paradigms in dense neural network processing. Building on established analytical modeling foundations [7], [8], the tool extends the word access calculation methods that accurately model the unique data access behavior of event-driven dataflows [4], complemented by static energy and area models for complete cost estimation. AeDAM seamlessly handles the asynchronous behavior inherent in event-driven execution by processing all elements in the dense case, enabling precise performance evaluation of event-driven accelerator architectures through its fine-grained analytical modeling approach that delivers reliable energy and latency metrics for design optimization.
3. **Unified Event-Driven Architecture Design Space Exploration Framework**
   AeDAM offers an end-to-end design space exploration framework that integrates hardware configuration analysis, workload characteristic analysis, and event-driven mapping techniques in a hierarchical way to identify Pareto-optimal architectures across a range of optimization metrics like energy efficiency, latency, and Energy-Delay Product. The framework combines systematic case study methods specifically designed for the evaluation of architectural transformations in event-driven accelerators [5] with detailed analysis of SRAM configurations and Processing Element architectures while leveraging established DSE principles [8] to enable comprehensive exploration of the event-driven design space.

Detailed structure of the internals of the AeDAM framework are as follows

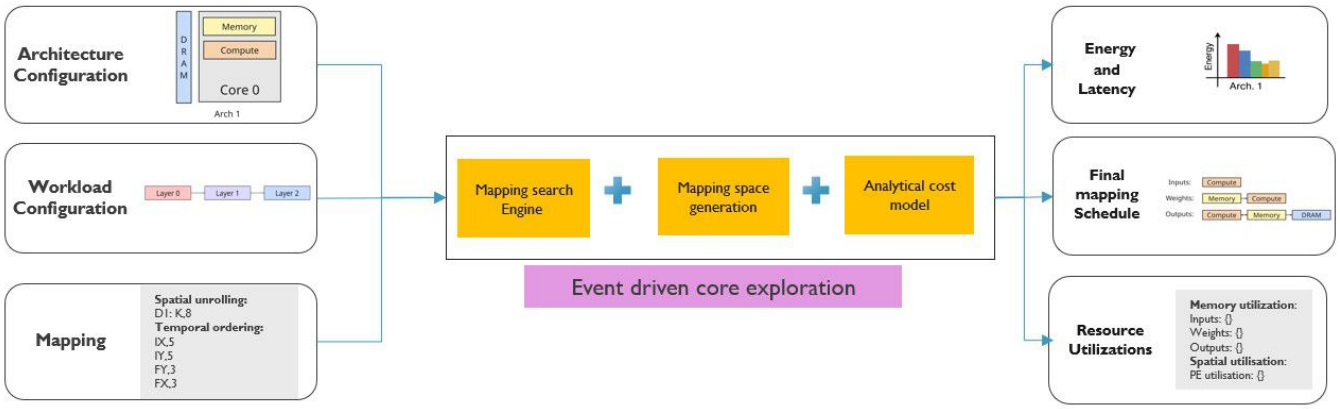


Figure 4.1: Detailed view of AeDAM framework

The AeDAM framework operates through a systematic process that identifies optimal event-driven mappings for the provided architecture configuration. Given hardware configuration specifications, workload characteristics, and mapping constraints, AeDAM generates a comprehensive event-driven mapping space using intelligent transformation algorithms. The specialized analytical cost model then evaluates each mapping configuration for energy, latency, and area metrics. Finally, the framework searches this mapping space to identify optimal solutions based on user-specified criteria (minimum latency, minimum energy, or optimal EDP), producing detailed analytical results and the best mapping configuration for executing the given workload on the target event-driven accelerator architecture.

5

AeDAM Implementation

5.1. Input Modeling and Configuration

The AeDAM framework begins its exploration process through comprehensive input modeling that establishes the foundation for event-driven design space exploration. This section details how AeDAM accepts, processes, and validates the three primary input categories that define the exploration scope and constraints.

5.1.1. Hardware Architecture Configuration

AeDAM requires a detailed specification of the target accelerator architecture to establish the hardware platform for exploration. The input hardware configuration primarily consists of three key components: the memory hierarchy definition, the processing element (PE) configuration, and the architecture constraints.

Memory Hierarchy Definition

The memory hierarchy specification forms the backbone of the hardware configuration, as it describes both the storage structure and the data movement constraints of the architecture. Provided in YAML format, this specification includes multiple memory levels, each characterized by parameters such as capacity, read and write bandwidth, energy cost, access latency, number of ports, and the physical area occupied by each memory block.

The framework supports flexible memory hierarchy configurations in which different operands (e.g., weights, inputs, outputs) can follow distinct memory allocation strategies.

Interconnection topology within the memory hierarchy is automatically derived based on the order of memory instance declarations, with the lowest-level memory components defined first. The memory hierarchy uses *memory operands* as virtual entities, decoupling algorithmic behavior from hardware specifics. These virtual operands are later linked to actual algorithmic operands through the ‘memory_operand_links‘ attribute.

Memory unrolling is controlled using the ‘served_dimensions‘ attribute, which specifies whether a memory level serves individual processing units or groups of units along specific array dimensions.

Processing Element Array Configuration

The PE configuration defines the computational core of the accelerator. It is composed of two main parts: the PE operational unit and the operational array. The operational unit specifies parameters such as energy consumption and area footprint of an individual PE.

The operational array, in turn, is a structured collection of these units and is defined by parameters such as the names and sizes of its dimensions. This array can be configured as either a 1D or 2D structures

Core Integration and Architecture Constraints

Together, the PE array and the memory hierarchy constitute the core of the accelerator, forming the essential computational and storage engine within AeDAM’s modeling framework.

Architecture constraints serve as guiding boundaries during the design space exploration process. These include limitations on total area, power consumption, ratios between different memory levels, bandwidth ceilings, and tech-

nology node parameters. AeDAM validates each input configuration against these constraints to ensure feasibility and practical implementability.

By enforcing these constraints and modeling interactions between compute and memory components in detail, AeDAM ensures that the generated accelerator configurations are not only optimized for performance metrics like energy and latency but also conform to real-world hardware design limitations.

Visual representation of the modelled architecture is as follows:

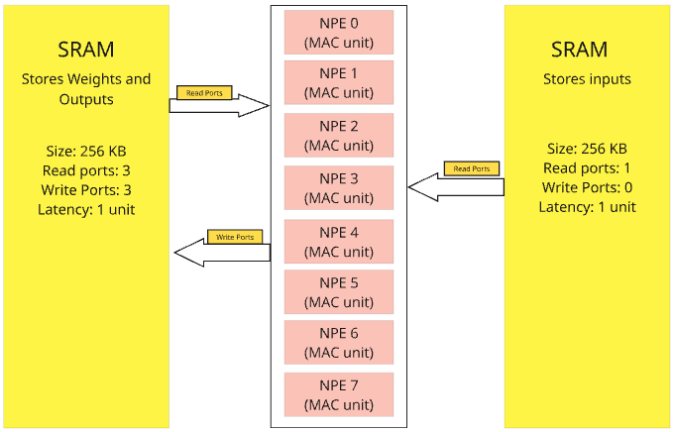


Figure 5.1: Example Architecture model

The architecture is modelled as follows:

Listing 5.1: Example YAML: Eight NPE Accelerator

```yaml
name: eight_NPE_accelerator
operational_array:
  unit_energy: 0.04
  unit_area: 1
  dimensions: [D1]
  sizes: [8]
memories:
  1. sram_2Mb_I2_O:
    size: 2097152
    r_bw: 128
    w_bw: 128
    r_cost: 10.5
    w_cost: 12.8
    area: 3
    r_port: 1
    w_port: 0
    rw_port: 1
    latency: 1
    operands: [I2, O]
    ports:
      - tl: r_port_1
      - fl: rw_port_1
        tl: rw_port_1
    served_dimensions: [D1]

  2. sram_2Mb_I2_O:
    size: 2097152
    r_bw: 128
    w_bw: 128
    r_cost: 10.5
    w_cost: 12.8
    area: 3
    r_port: 1
    w_port: 0
    rw_port: 0
    latency: 1
    operands: [I1]
    ports:
      - tl: r_port_1
    served_dimensions: [D1]
```

5.1.2. Workload Configuration and Modeling

The workload configuration component captures the computational requirements and characteristics of the target neural network layers that will be mapped onto the modelled accelerator architecture.

Neural Network Layer Specification

AeDAM accepts detailed descriptions of neural network layers like convolutional layers and fully connected layers. Each layer specification requires dimensional parameters such as input feature map sizes, filter dimensions, output feature map configurations, stride and padding parameters, and precision requirements. The framework represents these layers using standardized loop notation (`B`, `K`, `C`, `OY`, `OX`, `FY`, `FX`) that facilitates systematic analysis of computational patterns and data dependencies.

Computational Pattern Analysis

The workload modeling component analyzes the computational requirements of each neural network layer to extract relevant metrics for event-driven execution. This includes total multiply-accumulate (MAC) operation counts, data access patterns, temporal dependencies between operations, and opportunities for parallel execution. AeDAM

identifies the computational intensity of each layer, memory access requirements for different operands, and potential bottlenecks that may impact event-driven execution efficiency.

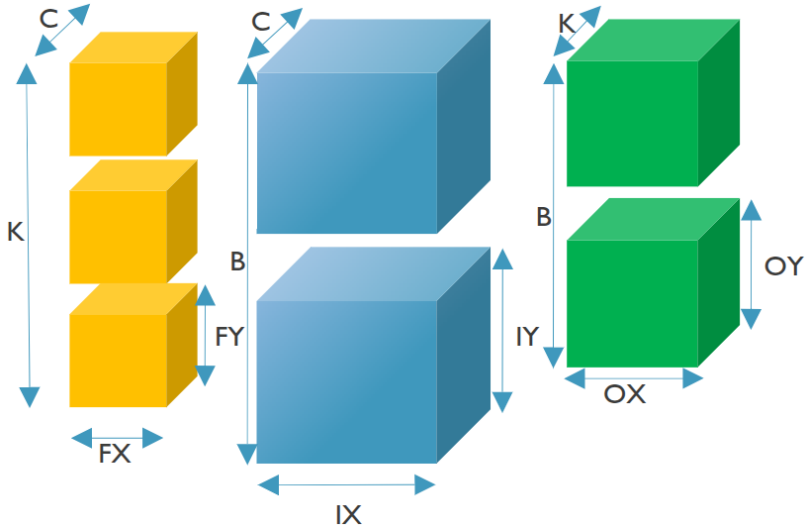


Figure 5.2: Convolution layer model

Listing 5.2: Example YAML: Event-Based Conv Layer Model

```
- id: 0 # Conv2
  name: conv_layer_
  operator_type: Conv
  equation: O[b][k][oy][ox] += W[k][c][fy][fx] * I[b][c][iy][ix]
  dimension_relations: [ox=1*ix+1*fx, oy=1*iy+1*fy]
  loop_dims: [B, K, C, IY, IX, FY, FX]
  loop_sizes: [b, k, c, iy, ix, fy, fx]
  operand_precision:
    W: 16
    I: 16
    O: 16
    O_final: 16
  operand_source:
    I: 0
    W: 0
```

5.1.3. Mapping Constraint Specification

The mapping constraint specification component allows users to define exploration boundaries and optimization preferences that guide the event-driven mapping generation process.

Spatial and Temporal Mapping Constraints

Users can specify constraints on spatial parallelization strategies, including preferred loop unrolling factors for different dimensions and PE array utilization requirements. Temporal mapping constraints define acceptable loop ordering preferences, memory access patterns, and data reuse strategies.

Optimization Criteria Definition

The framework requires specification of optimization objectives that drive the mapping search process. Users can define primary optimization criteria such as minimum energy consumption, minimum latency, optimal Energy-Delay Product (EDP). AeDAM supports weighted combinations of multiple optimization criteria, allowing users to explore trade-offs between different performance metrics according to their specific application requirements.

Listing 5.3: Example YAML: Mapping Constraints

```yaml
- name: default
  spatial_mapping:
    D1:
      - K, k
  temporal_ordering:
    - [FY, fy]
    - [FX, fx]
    - [IY, iy]
    - [IX, ix] # Outermost loop
  memory_operand_links:
    O: O
    W: I2
    I: I1
```

5.2. Event-Driven Mapping Space Generation

Event-driven mapping space generation represents the core innovation of AeDAM, transforming traditional synchronous mapping methodologies into specialized configurations optimized for asynchronous, event-based computation. This process fundamentally differs from conventional dataflow strategies by focusing on exploiting the unique characteristics of event-driven execution paradigms.

5.2.1. Event-Driven Dataflow Paradigm

The event-driven dataflow paradigm represents a fundamental departure from traditional synchronous processing approaches used in conventional CNN accelerators. Unlike weight-stationary (WS), input-stationary (IS), or output-stationary (OS) dataflows that rely on predetermined data movement patterns and regular execution schedules, event-driven dataflows operate on an asynchronous, demand-driven basis where computation is triggered exclusively by the arrival of events.

Event-Triggered Computation Model

In the event-driven paradigm, non-zero activations serve as computational triggers that initiate processing sequences rather than following predetermined execution schedules. This approach fundamentally alters the computational model from a time-driven synchronous system to an event-driven asynchronous system, where processing elements remain inactive until stimulated by incoming events. Each event carries not only data values but also addressing information that determines which computational operations should be executed and where results should be accumulated.

The event-driven model exploits the temporal and spatial sparsity inherent in neural network activations by processing only meaningful data elements while completely bypassing zero-valued activations. This selective processing approach can lead to significant energy savings compared to conventional dataflows that must process complete data structures regardless of their information content. However, this efficiency comes at the cost of increased control complexity and irregular execution patterns that require sophisticated event handling mechanisms.

Spatial and Temporal Unrolling Characteristics

Event-driven dataflows typically employ spatial unrolling strategies along K, FX, and FY dimensions, similar to weight-stationary approaches, but with fundamentally different temporal unrolling patterns. The temporal unrolling follows IX, IY, and C dimensions, where the top loops are forced to be input dimensions to perform the event-driven computation.

The spatial unrolling strategy ensures that when an event arrives, the corresponding computation can be efficiently distributed across multiple processing elements. The temporal unrolling pattern enables efficient streaming of events through the processing array while maintaining data locality and minimizing memory access overhead.

5.2.1.1. Event-Driven Mapping Space Transformation

AeDAM generates event-driven mapping spaces through intelligent transformation of traditional LOMA (Loop Order Memory Access) scheduling methodologies. This transformation process converts conventional synchronous loop nest structures into event-driven compatible configurations that maintain computational correctness while exploiting the advantages of asynchronous execution.

LOMA to Event-Driven Conversion Algorithm
The transformation process begins with traditional LOMA scheduling that establishes the fundamental loop ordering and memory access patterns for the target neural network layer. AeDAM then applies specialized filtration and conversion algorithms that restructure these loop nests to accommodate event-driven execution while preserving the essential data dependencies and computational requirements.

The conversion algorithm identifies loop dimensions that can benefit from event-driven execution. The algorithm restructures temporal loops to accommodate event processing while maintaining spatial loops that enable efficient parallel processing of events across the PE array.

Event-Driven Constraint Application
The mapping space generation process applies event-driven specific constraints that ensure generated mappings are compatible with event-based execution paradigms. These constraints include event ordering requirements that maintain computational correctness and memory access pattern restrictions that ensure efficient event processing.

The constraint application process validates each generated mapping against event-driven execution requirements, ensuring that data dependencies are properly maintained despite the asynchronous nature of event processing. This validation includes checking for proper event sequencing, adequate memory capacities for event queues, and compatible memory access patterns that support the event-driven data movement.

Through this comprehensive event-driven mapping space generation process, AeDAM creates specialized mapping configurations that fully exploit the advantages of asynchronous, event-based computation while maintaining compatibility with practical hardware implementations and achieving superior performance compared to conventional synchronous mapping approaches.

Below is the representation of the event driven mapping in which the spatial unrolling is across the output channels and the temporal unrolling is done to access the input event by event

Listing 5.4: Pseudocode for event driven loop ordering

```
for iw in IW:
  for ih in IH:
    for c in C:
      for fw in FW:
        for fh in FH:
          parfor k in K:
            Output[k][ix - fx][iy - fy] = Inputs[c][ix][iy] * Weights[k][fx][fy]
```

5.2.2. Analytical Cost Model

The analytical cost model forms the core of AeDAM's hardware cost estimation, extending ZigZag's established analytical modeling framework with specialized modifications for event-driven execution paradigms. This section details the comprehensive cost estimation process that transforms generated mapping spaces into accurate energy, latency, and area metrics for event-driven accelerator architectures.

The structure of the model is as follows

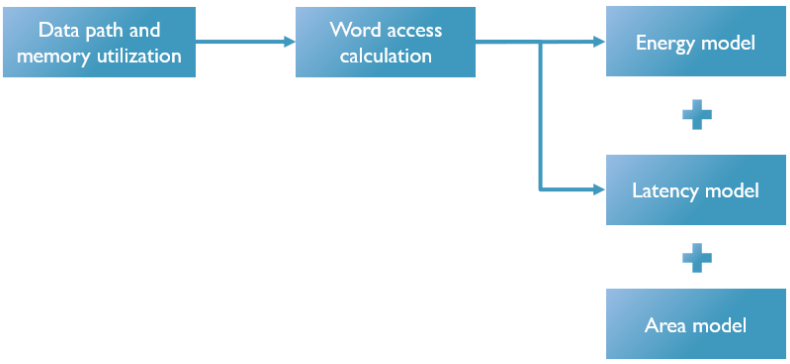


Figure 5.3: Analytical cost model pipeline

5.2.3. Event-Driven Datapath and Utilization Analysis

The datapath and memory utilization analysis establishes the foundation for all subsequent cost calculations by determining how data flows through the memory hierarchy and quantifying resource utilization patterns specific to event-driven execution.

Memory Utilization Calculation For each operand $o \in \{I, W, O\}$ and memory level ℓ, the memory utilization is calculated as:

$$U_{o,\ell} = \frac{D_{o,\ell}}{S_\ell} \times P_{o,\ell}$$

where:

- $D_{o,\ell}$: Data size in bits for operand o at level ℓ
- S_ℓ: Total memory capacity at level ℓ (in bits)
- $P_{o,\ell}$: Precision factor for operand o at level ℓ

Event-Driven Data Movement Patterns Event-driven execution follows a critical path where input events directly trigger computation. The data movement pattern is described as:

Input path: NoC $\rightarrow$ RISC-V controller $\rightarrow$ Task FIFO $\rightarrow$ Loop controller $\rightarrow$ NPE register file
Weight path: SRAM (DMEM) $\rightarrow$ Loop controller $\rightarrow$ NPE register file $\rightarrow$ MAC units
Output path: NPE register file $\rightarrow$ DMEM write $\rightarrow$ Event generator $\rightarrow$ Output FIFO $\rightarrow$ NoC

5.2.3.1. Modified Word Access Calculation Engine

This component adapts ZigZag's memory access modeling to accurately capture event-driven behavior.

General Word Access Formula
For each operand o, memory level ℓ, and data direction in `{rd_out_to_high, wr_in_by_high, rd_out_to_low, wr_in_by_low}`:

$$A_{o,\ell,d} = \left\lceil \frac{a_{o,\ell,d} \times p_{o,\ell,d}}{bw_{\min}(\ell, d)} \right\rceil \times \frac{bw_{\min}(\ell, d)}{bw_{\max}(\ell, d)} \times P_{o,\ell,d} \times U_{o,\ell}$$

where:

- $A_{o,\ell,d}$ = total word accesses for operand o at level ℓ in direction d
- $a_{o,\ell,d}$ = data amount per period
- $p_{o,\ell,d}$ = data precision in bits
- $bw_{\min}(\ell, d)$ = minimum bandwidth at level ℓ for direction d
- $bw_{\max}(\ell, d)$ = maximum bandwidth at level ℓ for direction d
- $P_{o,\ell,d}$ = number of periods for the transfer
- $U_{o,\ell}$ = number of spatial units accessing memory at level ℓ
- $o \in \{I, W, O\}$ = operand type (Input, Weight, Output)
- ℓ = memory level index (0 is closest to the PE)
- d = data movement direction

Event-Driven Output Access Modification
The key innovation in AeDAM's word access calculation lies in its specialized handling of output operands at memory interface boundaries.

SRAM Level (Level 1):

$$A_{O,1,\text{rd\_out\_to\_high}} = \frac{(OX \times OY \times K) \times data\_precision}{bw_{SRAM}}$$

$$A_{O,1,\text{wr\_in\_by\_low}} = 0$$

NoC Level (Level 2):

$$A_{O,2,\text{rd\_out\_to\_low}} = 0$$

$$A_{O,2,\text{wr\_in\_by\_high}} = \frac{(OX \times OY \times K) \times data\_precision}{bw_{NoC}}$$

where:

- OX, OY = output feature map dimensions
- K = number of output channels
- $data\_precision$ = number of bits per output (e.g., 16 bits)
- bw_{SRAM} = SRAM bandwidth (bits per cycle)
- bw_{NoC} = NoC bandwidth (bits per cycle)

Output Dimension Relationships

$$OX = IX + FX, \quad OY = IY + FY$$

$$OW = IW - FW + 1, \quad OH = IH - FH + 1$$

where:

- IX, IY = mapped input tile dimensions
- FX, FY = mapped filter tile dimensions
- IW, IH = full input width and height
- FW, FH = full filter width and height
- OW, OH = full output width and height

Period Count Calculation
The number of transfer periods at each memory level is defined by:

$$P_{o,\ell} = \prod_{t \in \text{outer loops for } (o,\ell)} T_t$$

where:

- $P_{o,\ell}$ = total number of periods for operand o at memory level ℓ
- T_t = trip count (iteration count) of temporal loop t
- "outer loops for (o, ℓ)" = set of temporal loops above memory level ℓ for operand o

5.2.3.2. Energy Model Implementation

The energy model calculates both dynamic and static energy components, with specialized modifications for event-driven execution patterns.

Dynamic Energy Calculation The total dynamic energy comprises MAC energy and memory energy:

$$E_{\text{dynamic}} = E_{\text{MAC}} + E_{\text{memory}}$$

MAC Energy

$$E_{\mathrm{MAC}} = N_{\mathrm{MAC}} \times \epsilon_{\mathrm{MAC}}$$

where:

E_{MAC}	total MAC operation energy in picojoules (pJ),
N_{MAC}	total number of multiply□accumulate operations,
ϵ_{MAC}	energy per MAC operation in pJ.

Memory Energy

$$E_{\mathrm{memory}} = \sum_{o \in \{I,W,O\}} \sum_{\ell=0}^{L-1} \Big[\left(R_{\uparrow}(o,\ell) + R_{\downarrow}(o,\ell)\right) \epsilon_r(\ell) \; + \; \left(W_{\uparrow}(o,\ell) + W_{\downarrow}(o,\ell)\right) \epsilon_w(\ell) \Big]$$

where:

E_{memory}	total memory access energy in pJ,
$o \in \{I, W, O\}$	operand type (Input, Weight, Output),
L	number of memory hierarchy levels,
$R_{\uparrow}(o,\ell), R_{\downarrow}(o,\ell)$	read accesses up/down at level ℓ,
$W_{\uparrow}(o,\ell), W_{\downarrow}(o,\ell)$	write accesses up/down at level ℓ,
$\epsilon_r(\ell), \epsilon_w(\ell)$	read/write energy per access at level ℓ in pJ.

EventDriven Memory Energy

$$E^{\mathrm{event}}_{\mathrm{memory}} = \sum_{o \in \{W,O\}} \sum_{\ell=0}^{L-1} \Big[\left(R_{\uparrow}(o,\ell) + R_{\downarrow}(o,\ell)\right) \epsilon_r(\ell) \; + \; \left(W_{\uparrow}(o,\ell) + W_{\downarrow}(o,\ell)\right) \epsilon_w(\ell) \Big] \; + \; E_{\mathrm{pre}} \; + \; E_{\mathrm{post}}$$

where:

$E_{\mathrm{pre}},\ E_{\mathrm{post}}$	event preprocessing/postprocessing energy per event in pJ.

Static Energy Calculation

$$E_{\mathrm{static}} = P_{\mathrm{static}} \times t_{\mathrm{execution}}$$

Static Power

$$P_{\mathrm{static}} = \sum_{\ell=0}^{L-1} C_\ell \, \rho_\ell \; + \; P_{\mathrm{MAC}} \; + \; P_{\mathrm{control}}$$

where:

C_ℓ	memory capacity in bits at level ℓ,
ρ_ℓ	leakage power density (pW/bit) at level ℓ,
P_{MAC}	static power of the MAC array in pW,
P_{control}	static power of control logic (core + NoC + clock) in pW.

Static Power Components

$$P_{\mathrm{MAC}} = N_{\mathrm{PE}} \times \epsilon_{\mathrm{MAC,static}}, \quad P_{\mathrm{control}} = P_{\mathrm{core}} + P_{\mathrm{noc}} + P_{\mathrm{clock}}$$

where:

N_{PE}	number of processing elements,
$\epsilon_{\mathrm{MAC,static}}$	static power per MAC unit in pW,
$P_{\mathrm{core}}, P_{\mathrm{noc}}, P_{\mathrm{clock}}$	static power of core, NoC router, and clock in pW.

5.2.3.3. Latency Model Implementation

The Latency model calculates computation, data onloading and offloading latency, and overall latency

Computation Latency

$$L_{\text{ideal}} = \left\lceil \frac{N_{\text{MAC}}}{N_{\text{PE}}} \right\rceil \times c, \quad L_{\text{temporal}} = T_{\text{mapping}} \times c + S_{\text{stall/slack}}$$

where:

c	cycles per MAC operation,
T_{mapping}	temporal mapping cycles from schedule,
$S_{\text{stall/slack}}$	combined stall/slack cycles due to memory.

Data Transfer Latency

$$C_{o,\ell}^{(\text{wr})} = \left\lceil \frac{d_{o,\ell}^{(\text{wr})}\, p_{o,\ell}^{(\text{wr})}}{bw_{o,\ell}^{(\text{wr})}} \right\rceil, \quad C_{o,\ell+1}^{(\text{rd})} = \left\lceil \frac{d_{o,\ell+1}^{(\text{rd})}\, p_{o,\ell+1}^{(\text{rd})}}{bw_{o,\ell+1}^{(\text{rd})}} \right\rceil$$

where:

$d_{o,\ell}^{(\text{wr})}, d_{o,\ell+1}^{(\text{rd})}$	data amount written/read in bits,
$p_{o,\ell}^{(\text{wr})}, p_{o,\ell+1}^{(\text{rd})}$	data precision in bits,
$bw_{o,\ell}^{(\text{wr})}, bw_{o,\ell+1}^{(\text{rd})}$	bandwidth in bits/cycle.

Data Onloading and Offloading *Single□Operand Onloading:*

$$D_{\text{on}} = \sum_{i=0}^{L-2} \max\big(C_{\text{wr\_in\_by\_high}}^{(i)},\, C_{\text{rd\_out\_to\_low}}^{(i+1)}\big)$$

TwoOperand Onloading:

$$D_{\text{on}} = \min\big[S_1 + \max(S_2 + H_2 + I_2, H_1 + I_1),\ S_2 + \max(S_1 + H_1 + I_1, H_2 + I_2)\big]$$

Offloading:

$$D_{\text{off}} = \sum_{\ell=0}^{M-2} \max(C_{\ell}^{(\text{rd})}, C_{\ell+1}^{(\text{wr})})$$

where:

S_1, S_2	shared latency components,
H_1, H_2	half□shared latency components,
I_1, I_2	individual latency components,
M	number of memory levels for output.

Total Latency

$$L_0 = L_{\text{temporal}} + S_{\text{stall/slack}}, \quad L_1 = L_0 + D_{\text{on}}, \quad L_2 = L_1 + D_{\text{off}}$$

5.2.3.4. Area Model Implementation

The Area model calculates the total core area by considering the event driven architecture overheads

Memory Area Calculation

$$A_{\text{memory}} = (A_{\text{overhead}} + C_{\text{bits}}\, A_{\text{cell}}\, F_{\text{port}}) \times F_{\text{word}}$$

where:

A_{overhead}	fixed overhead area (μm^2),
C_{bits}	memory capacity in bits,
A_{cell}	area per bit (μm^2/bit),
F_{port}	port factor,
F_{word}	word□width factor.

Port Factor

$$F_{\text{port}} = \begin{cases} 1.0 + 0.27\,\frac{n_{\text{ports}}-1}{3} + 0.05\,n_{\text{write}}, & n_{\text{ports}} \le 4, \\ \min(n_{\text{ports}} \times 0.8, n_{\text{ports}}), & n_{\text{ports}} > 4. \end{cases}$$

where n_{ports} = total ports, n_{write} = write ports.

Technology Scaling

$$F_{\text{tech}} = \left(\frac{t_{\text{node}}}{22}\right)^2$$

where t_{node} in nm.

Event–Driven Overhead

$$A_{\text{event\_overhead}} = A_{\text{noc\_router}} + A_{\text{event\_generator}} + A_{\text{loop\_buffer}} + A_{\text{fifo\_queues}} + A_{\text{control\_logic}}$$

With 22 nm baselines:

$$\begin{aligned} A_{\text{noc\_router}} &= 12100\,F_{\text{tech}}, \\ A_{\text{event\_generator}} &= 9700\,F_{\text{tech}}, \\ A_{\text{loop\_buffer}} &= 10500\,F_{\text{tech}}, \\ A_{\text{fifo\_queues}} &= 3000\,F_{\text{tech}}, \\ A_{\text{control\_logic}} &= 2400\,F_{\text{tech}}. \end{aligned}$$

Total Core Area

$$A_{\text{total}} = A_{\text{memory}} + A_{\text{PE\_array}} + A_{\text{logic\_overhead}} + A_{\text{event\_overhead}}$$

where $A_{\text{logic\_overhead}} = 10900\,F_{\text{tech}}$.

5.3. Mapping Space Search Engine

The mapping space search engine systematically applies the analytical cost model to each mapping in the generated event-driven mapping space and identifies optimal solutions based on specified optimization criteria.

The high level flow of the mapping space search engine is as follows

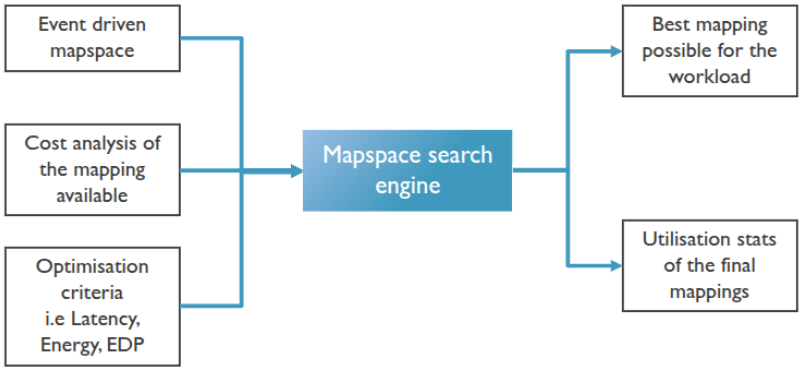


Figure 5.4: Mapspace search engine

5.3.1. Cost Model Application Process

The search engine applies AeDAM's analytical cost model to every mapping candidate in the event-driven mapping space through a standardized evaluation sequence.

For each mapping, the engine executes the complete cost estimation pipeline: memory utilization analysis, word access computation with event-driven modifications, energy calculation (including both dynamic and static components), latency analysis (comprising computation and data transfer), and area assessment. The engine coordinates these calculations to ensure proper dependency handling and applies event-driven specific modifications, such as

the modified output operand access patterns where outputs bypass intermediate memory levels and write directly to the NoC.

The cost model application maintains consistency across all mappings while handling event-driven parameters correctly, ensuring that each mapping receives accurate performance estimates that reflect the unique characteristics of asynchronous, event-based computation.

5.3.2. Multi-Criteria Optimization and Comparison

The search engine evaluates each mapping against three optimization criteria and systematically identifies the best performing solutions.

Optimization Criteria Evaluation

The engine maintains running records of optimal mappings for each criterion:

- **Energy Optimization:** Identifies mappings that minimize total energy (including dynamic MAC and memory energy plus static leakage energy).
- **Latency Optimization:** Finds mappings that achieve minimum execution latency, including computation cycles, memory stalls, and data transfer overhead.
- **Energy-Delay Product (EDP):** Discovers mappings that balance energy and latency for optimal Energy-Delay Product.

Comparison and Selection Process

For each evaluated mapping, the engine compares performance metrics against current best solutions and updates the optimal mappings when superior candidates are discovered. The comparison process validates mapping feasibility, ensures event-driven compliance, and maintains performance rankings across the entire mapping space.

The engine tracks performance improvements and trade-offs, providing insights into how different optimization objectives affect mapping selection and overall accelerator performance.

5.3.3. Results Organization and Output Generation

The search engine produces comprehensive results that enable informed design decisions for event-driven accelerator architectures.

Optimal Mapping Documentation

For each optimization criterion, the engine generates complete mapping specifications, including:

- Spatial unrolling patterns
- Temporal loop ordering
- Memory allocation strategies
- Detailed performance metrics

These results include an energy breakdown across memory hierarchy levels, latency component analysis, resource utilization statistics, and event-driven specific metrics such as sparsity exploitation benefits and memory access pattern efficiency.

The final output contains actionable recommendations for accelerator design, implementation guidance for optimal mappings, and insights into the effectiveness of event-driven execution paradigms for the target neural network workloads.

This streamlined approach ensures that AeDAM's mapping space search engine efficiently processes the generated event-driven mapping space, accurately applies cost models, and provides clear optimization results that guide effective accelerator design decisions.

5.4. Comprehensive Output Generation and Analysis

AeDAM generates comprehensive output reports that follow established design space exploration conventions while incorporating event-driven specific insights and analysis capabilities.

The overall flow is as follows in figure 5.5

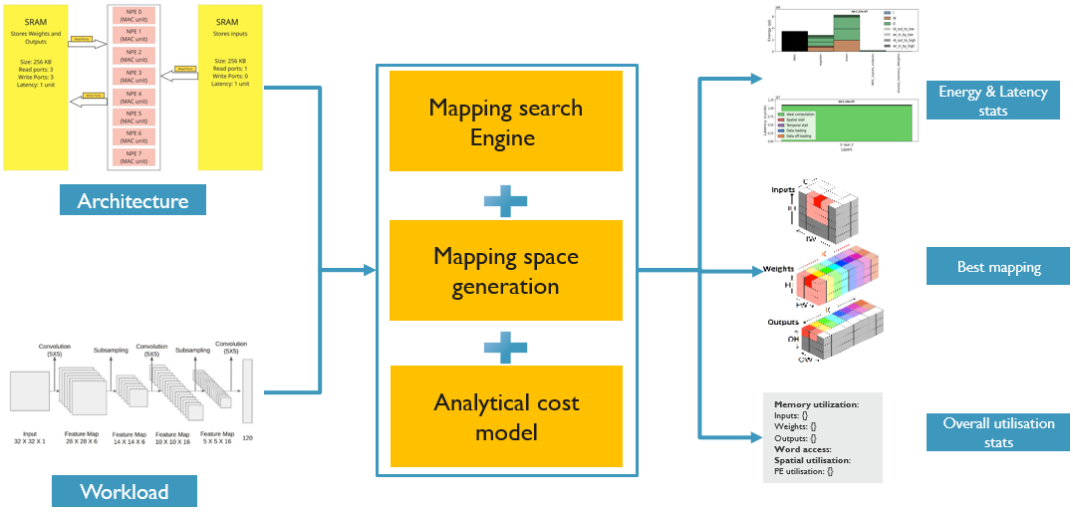


Figure 5.5: Overall block diagram

5.4.1. Performance Visualization and Analysis

Energy and Latency Results

AeDAM produces detailed energy and latency analysis graphs that visualize the performance characteristics of optimal mappings across the explored design space. The energy breakdown includes a comprehensive analysis of dynamic energy consumption (from MAC operations and memory hierarchy accesses) and static energy components (such as leakage power), with specific attention to energy savings achieved through event-driven execution patterns.

The latency analysis provides a detailed timing breakdown, including computation cycles, data onloading and offloading overheads, and memory stall components. For event-driven architectures, the latency results highlight the benefits of reduced data movement and elimination of frame buffering overhead compared to conventional synchronous processing approaches.

Resource Utilization Statistics

The framework generates comprehensive resource utilization reports detailing memory utilization across all hierarchy levels, processing element (PE) array utilization efficiency, spatial utilization metrics, and memory bandwidth usage patterns. These statistics offer insights into how effectively the optimal mappings exploit available hardware resources and identify potential architectural bottlenecks.

5.4.2. Optimal Mapping Configuration Output

Final Mapping Schedule Generation

AeDAM produces complete mapping specifications for each optimization criterion. These include:

- **Spatial unrolling configurations:** defining how loop dimensions are distributed across the PE array,
- **Temporal loop ordering:** determining the sequence of computational operations,
- **Memory allocation strategies:** specifying operand placement across the memory hierarchy,

The mapping schedules also include loop nest specifications, memory access patterns, data movement timelines, and resource allocation details, enabling direct implementation in event-driven accelerator architectures.

Performance Metrics Summary

The output includes comprehensive performance summaries presenting:

- Energy consumption breakdowns,
- Execution latency analysis,
- Area utilization metrics,

- Energy-Delay Product (EDP) calculations.

Comparative analyses highlight how event-driven optimal mappings perform relative to conventional approaches, and quantify the benefits achieved through specialized event-driven design space exploration.

These outputs provide designers with actionable insights for accelerator architecture development, clear guidance for optimal mapping implementation, and a deep understanding of the performance trade-offs involved in event-driven neural network accelerator design. By combining detailed technical specifications with high-level performance analysis, AeDAM supports informed decision-making throughout the accelerator design process—from initial architecture conception to final implementation optimization.

6

Results and Analysis

This section is divided into three major parts:

1. **Validation of the Tool**
   The event-driven core modeling block developed in AeDAM is validated by manually implementing the analytical formulas for each functionality. This ensures that the tool correctly produces expected behavior.
2. **Comparison with Existing Frameworks**
   The AeDAM framework leverages selected components from the established ZigZag framework while introducing specialized extensions for event-driven execution paradigms. A comprehensive evaluation compares baseline ZigZag capabilities, intermediate modifications incorporating event-driven support, and the complete AeDAM implementation. This systematic analysis demonstrates AeDAM's enhanced modeling capabilities for event-driven accelerator architectures.
3. **Case Study Analysis**
   AeDAM enables systematic exploration of critical architectural parameters within event-driven accelerator systems, encompassing SRAM configurations, processing element architectures, and interconnect specifications, including Network-on-Chip topology, DRAM bandwidth allocation, and buffer hierarchy sizing. The framework's capabilities are demonstrated through a comprehensive analysis of the SENECA architecture, providing both empirical validation of AeDAM's modeling accuracy and identification of optimal architectural configurations for representative workload scenarios.

6.1. Validation of the tool

The following section validates the Datapath, wordaccess, latency and energy calculations block of the AeDAM tool flow.

The validation experiments employ an 8-NPE SENECA architecture as the target hardware platform for event-driven design space exploration. This architecture represents a neuromorphic accelerator configuration with eight neural processing elements organized in a linear array structure, specifically designed to handle sparse, event-driven computational patterns. The SENECA architecture implements a dual-controller design that combines flexible preprocessing capabilities with efficient event execution, making it suitable for evaluating the performance characteristics of event-driven workloads under various mapping strategies (see Figure 6.1).

The experimental validation utilizes the VGG neural network as the target workload to assess the accuracy and effectiveness of the design space exploration framework. VGG is a widely-used convolutional neural network architecture characterized by its systematic use of small convolutional filters and deep layer structures. This network provides a representative workload for evaluating accelerator performance across different computational patterns, from early convolutional layers with high spatial dimensions to deeper layers with increased channel complexity (see Figure 6.2).

The first layer of the VGG neural network serves as the validation benchmark for the framework, with detailed layer specifications presented in Table 6.1

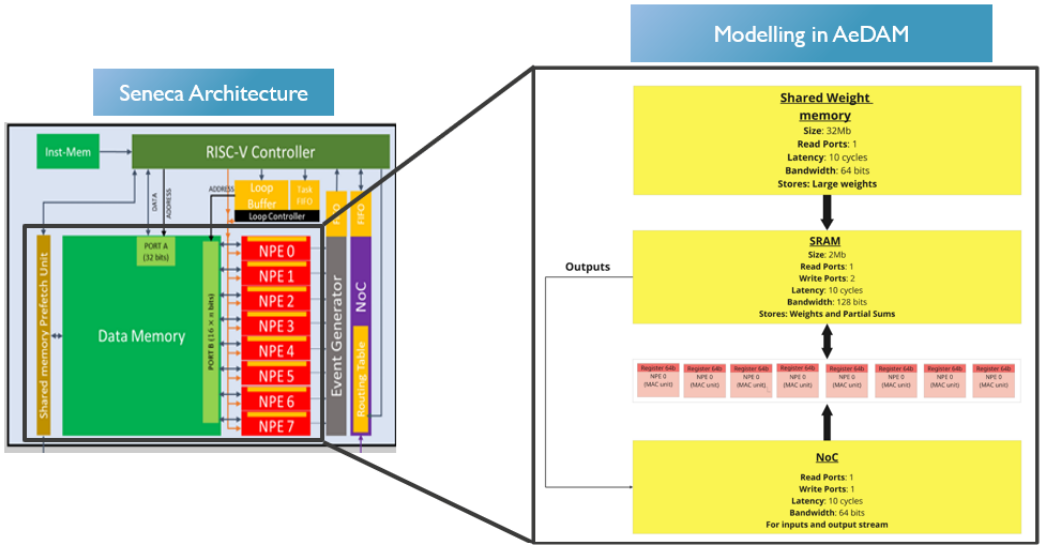


Figure 6.1: 8-NPE SENECA Architecture Configuration

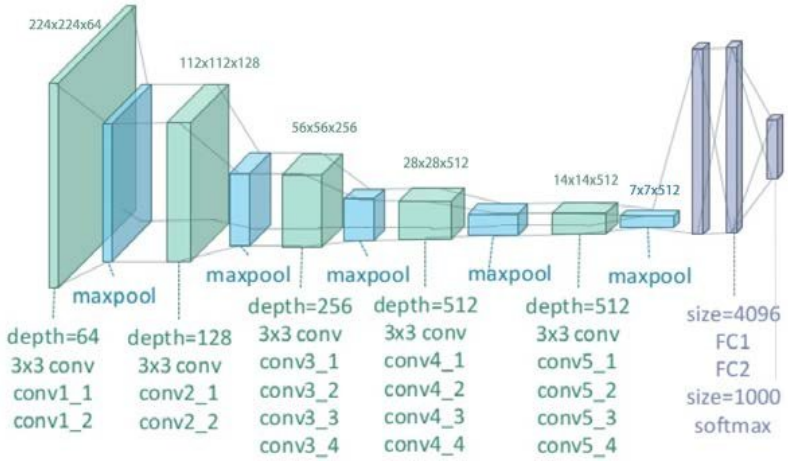


Figure 6.2: VGG Neural Network Structure

Layer Configuration	
Inputs (B,C,IX,IY)	(1,3,224,224)
Weights (K,C,FX,FY)	(64,3,3,3)
Outputs (B,K,OX,OY)	(1,64,224,224)
Total MACs	86704128

Table 6.1: Layer Configuration

Datapath validation The datapath validation ensures that the event-driven flow of the inputs, weights, partial sums, and outputs in AeDAM aligns with the operational behavior of the target architecture.

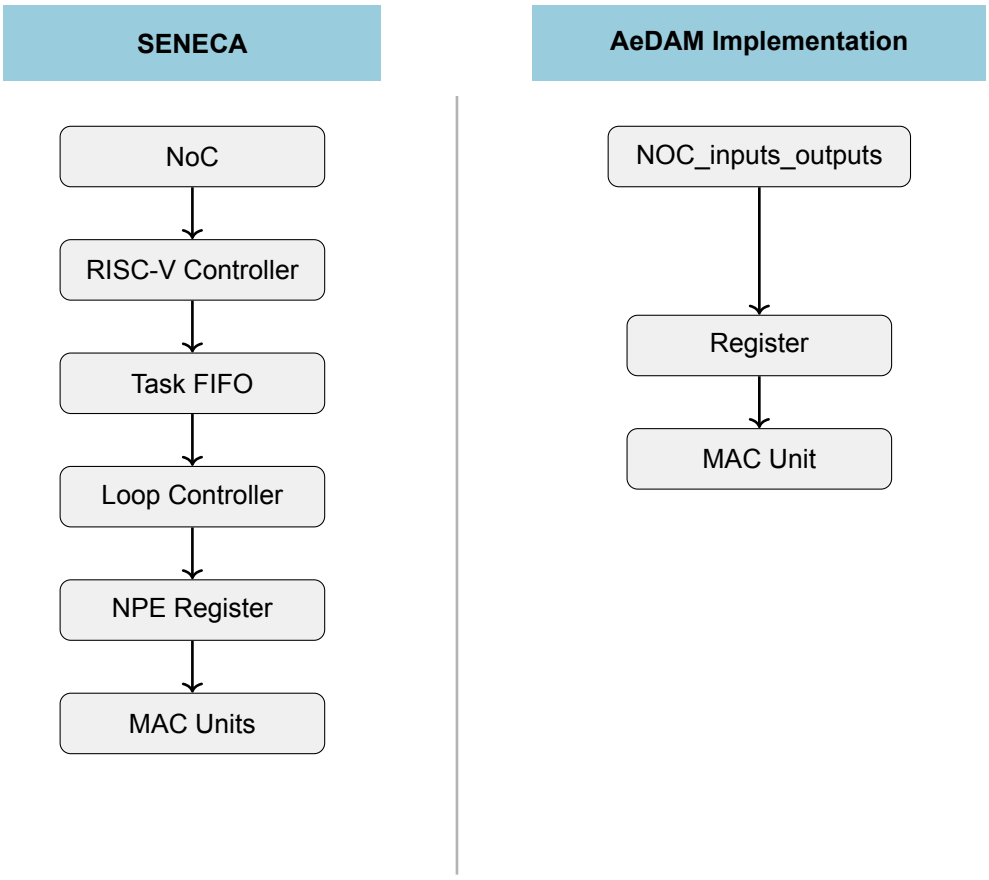
These paths are systematically compared to the actual datapath in SENECA's hardware, verifying that every event-triggered data movement, accumulation, and storage operation follows the expected behavior.

This validation confirms that AeDAM accurately captures the dynamic behavior of the event-driven core—through precise event triggers, their effects on the datapath, and the propagation through the SENECA pipeline.

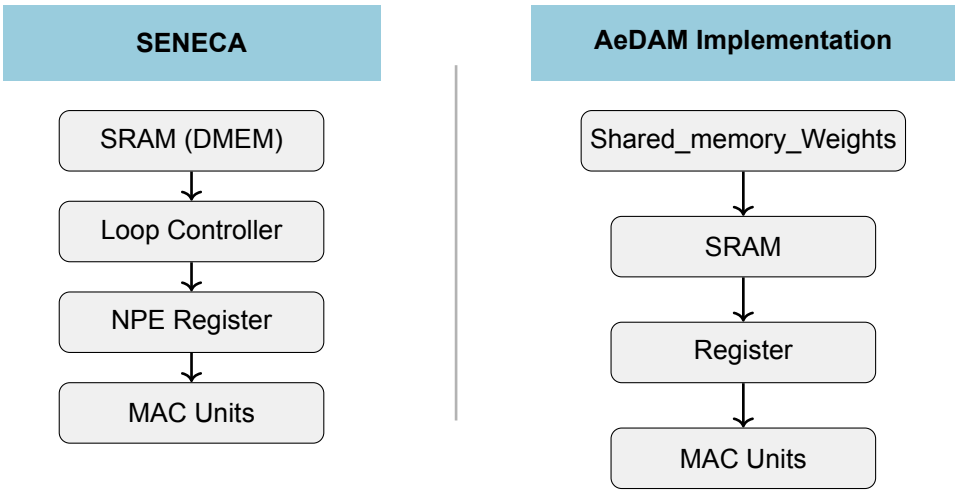
Here are the datapath comparisons of the SENECA and the AeDAM-modeled SENECA core:

1. **Input Path**

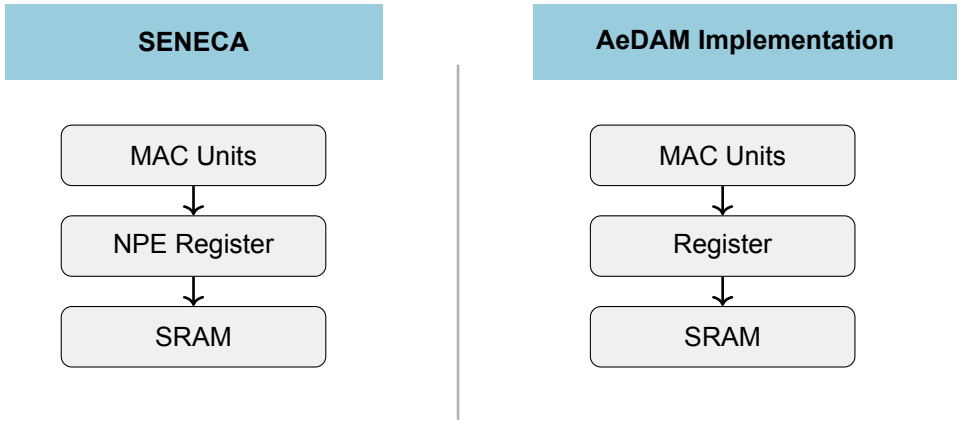
In the SENECA architecture, the input data flows through multiple stages: starting from the NoC, it passes through the RISC-V controller, Task FIFO, and Loop Controller before reaching the NPE Register and finally the MAC units. This sequence ensures proper scheduling and control of data movement in the event-driven pipeline. In AeDAM, this flow is abstracted to a simplified path where the input directly moves from the `NOC_inputs_outputs` module to the Register and then to the MAC unit. This captures the core behavior while removing intermediate control elements for modeling simplicity.



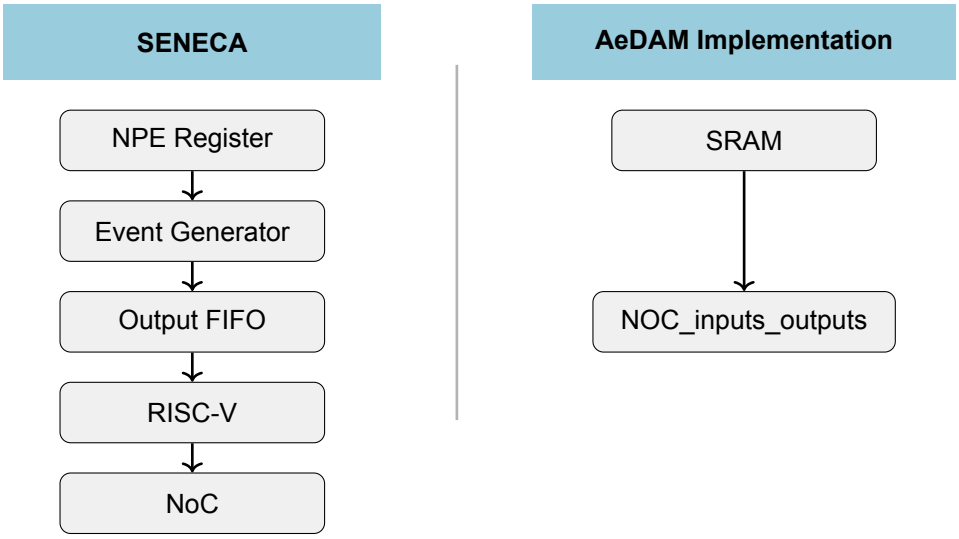
2. **Weight Path**
   SENECA fetches weights from SRAM (DMEM), passes them through the Loop Controller, and stores them in the NPE Register before computation by the MAC units. This structured flow ensures correct indexing and reuse of weights. AeDAM models this using a `Shared_memory_Weights` block followed by an internal SRAM buffer, Register, and then the MAC unit. Although simplified, this mirrors the data path faithfully, maintaining the same computational flow.



3. **Partial Sum Path**
   In both SENECA and AeDAM, the partial sums follow a similar and consistent path. The MAC units compute the partial sums, which are then temporarily stored in the Register and finally written back to SRAM. This confirms that AeDAM captures the essential accumulation behavior required for iterative computations in neural networks.



4. **Output Path**
   SENECA handles output generation through a multi-stage flow: from the NPE Register to the Event Generator, then to the Output FIFO, followed by the RISC-V controller, and finally transmitted via the NoC. This ensures outputs are event-tagged and managed appropriately for downstream processing. AeDAM simplifies this by directly moving the final output from SRAM to the `NOC_inputs_outputs` module, effectively modeling the output dispatch while abstracting the internal control mechanisms.



Wordaccess validation The validation of word access counts is essential for accurate latency and energy calculations in event-driven architectures, as memory access patterns directly determine both timing and energy consumption. The word access behavior must match the datapath execution sequence, where input activations are streamed one by one, triggering weight retrievals from memory, followed by partial sum computation, and finally output generation. This validation process includes five key access types: input reads representing sequential streaming of events with direct writes to processing units; weight reads involving retrieval of weight elements from the memory hierarchy; weight writes distributing elements to computation units; partial sum writes capturing intermediate results; and output writes representing final results with unidirectional flow. To validate this approach, the 8-NPE SENECA architecture was used with VGGNet's first convolutional layer as the test case, employing the AeDAM framework to model event-driven access patterns with constraints specific to neuromorphic systems.

The results demonstrate precise alignment between manual calculations and AeDAM framework predictions across all memory access categories, confirming the accuracy of the automated modeling approach. Weight reads and

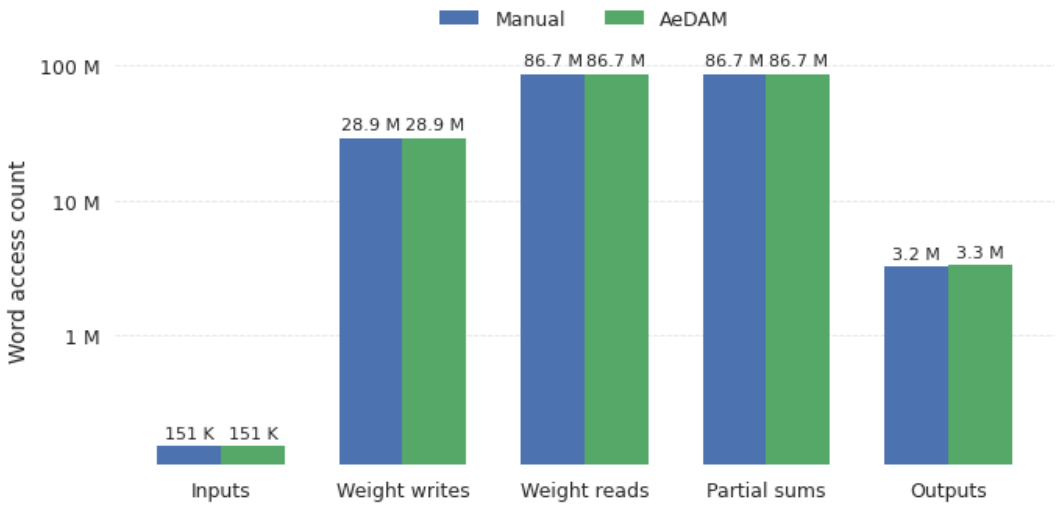


Figure 6.3: Word access count validation

partial sums represent the dominant memory operations at 86.7 million accesses each, reflecting the computational intensity of convolutional operations where each weight parameter must be accessed multiple times and partial results accumulated throughout the processing sequence. Weight writes account for 28.9 million operations as filter parameters are distributed across processing elements, while input reads remain minimal at 151,000 operations due to, event-driven processing paradigm that activates only when non-zero inputs occur. Output operations total approximately 3.2 million accesses, corresponding to final feature map generation. The near-perfect correspondence between manual and automated predictions validates the framework's capability to accurately model complex memory access patterns in event-driven architectures, establishing confidence for subsequent energy and performance analyses.

Latency validation The latency validation demonstrates perfect correspondence between manual calculations and AeDAM framework predictions, with both approaches yielding identical cycle counts of 10,838,016 total cycles for processing the first convolutional layer. The analysis reveals that true computational cycles dominate the execution profile, accounting for the overwhelming majority of processing time, while data movement overhead remains minimal. Data loading operations require only 11 cycles, reflecting the efficient event-driven data streaming approach where input activations are processed as they arrive. Stall cycles total 150,526, representing brief periods where processing elements await data availability, while data offloading operations consume 10,857 cycles for result storage. The negligible data movement overhead compared to computational cycles indicates efficient memory hierarchy utilization and demonstrates that the event-driven architecture successfully minimizes traditional memory wall effects. This validation confirms the framework's precision in modeling temporal behavior for neuromorphic accelerators and establishes confidence in latency predictions for design space exploration across different architectural configurations.

Energy validation The energy validation demonstrates excellent agreement between manual calculations and AeDAM framework predictions across all operational categories, confirming the accuracy of energy modelling for event-driven architectures. The analysis reveals that partial-sum operations and multiply–accumulate computations constitute the dominant energy consumers, with partial sums accounting for approximately 5.8×10^5 energy units and MAC operations consuming 3.5×10^5 units, reflecting the energy-intensive nature of accumulation and arithmetic processing in convolutional neural networks. Weight-read operations from the memory hierarchy contribute a moderate overhead at roughly 2.2×10^5 units, while weight writes to SRAM require slightly less energy for parameter distribution. Input operations through the network-on-chip interface show minimal consumption at 1.5–1.7×10^3 units, consistent with the sparse, event-driven paradigm that activates only on non-zero inputs. The close correspondence between manual and automated predictions across all categories validates the framework's capability to model energy consumption accurately in neuromorphic accelerators, providing a reliable foundation for energy-aware design-space exploration and optimisation strategies.

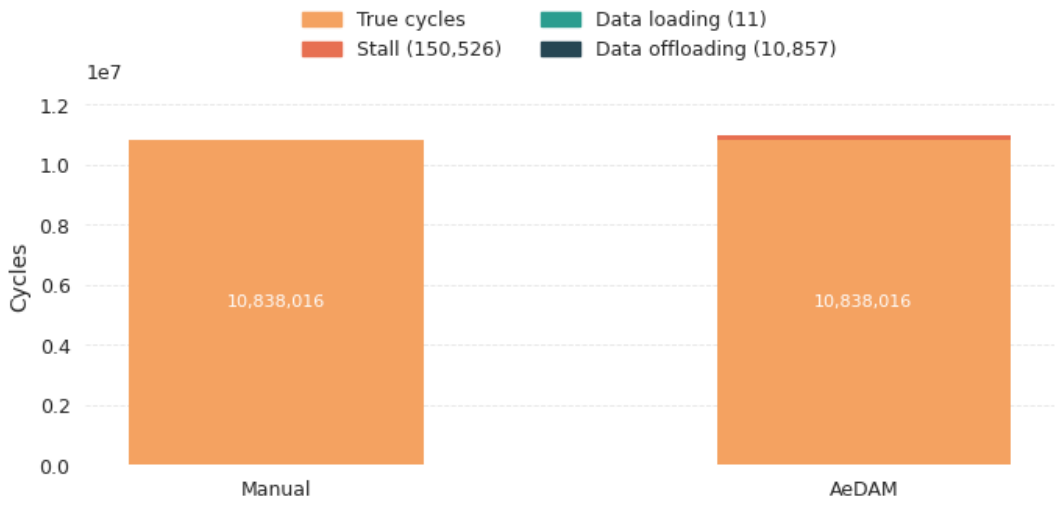


Figure 6.4: Latency validation

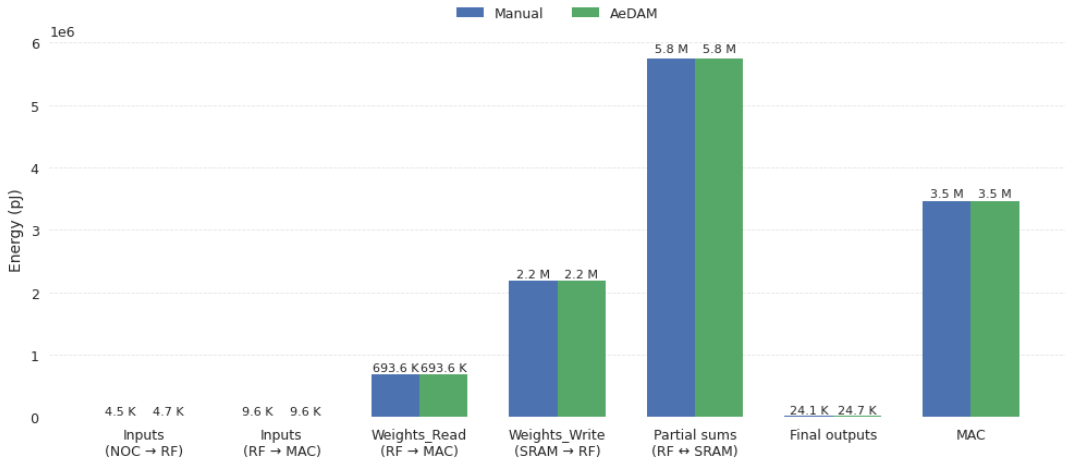


Figure 6.5: Energy validation

6.2. Comparison with Existing Frameworks

This section presents a comparative analysis between the state-of-the-art ZigZag framework and the AeDAM framework across multiple evaluation criteria. The comparison encompasses mapping space generation capabilities, exploration speeds, and optimization performance for specific layers of the VGGNet architecture. This analysis provides insights into the relative strengths and limitations of each framework when applied to their respective accelerator paradigms.

Framework Overview and Architectural Focus ZigZag represents a rapid deep neural network accelerator joint architecture and mapping design-space exploration framework that targets frame-based processing architectures [8]. The framework implements comprehensive design-space exploration capabilities for conventional accelerator architectures that process complete data frames as atomic computational units. ZigZag employs sophisticated mapping search engines and analytical cost estimation models to identify optimal configurations across diverse hardware architectures and workload characteristics.

AeDAM constitutes a specialized mapping and design-space exploration framework specifically developed for event-driven accelerator architectures [4]. The fundamental distinction between these frameworks lies in their target architectural paradigms: ZigZag focuses on frame-based accelerators that process complete data blocks, while AeDAM addresses the unique requirements of event-driven architectures that respond to individual activation events.

Exploration Methodology Differences The architectural paradigm difference necessitates fundamentally different exploration approaches and optimization strategies. Frame-based accelerators benefit from predictable memory access patterns and regular computational structures that enable systematic optimization of spatial and temporal mapping strategies. In contrast, event-driven architectures require specialized modeling techniques to capture the irregular, sparse activation patterns and asynchronous processing characteristics that define neuromorphic computing paradigms.

Leveraging Existing Components AeDAM extends and utilizes components from existing design-space exploration frameworks while incorporating novel modeling capabilities specifically tailored to event-driven processing requirements. This development approach leverages proven analytical modeling techniques from ZigZag, while introducing specialized features for sparse data handling, irregular memory access patterns, and event-driven control-flow management that distinguish event-driven accelerators from their frame-based counterparts.

The complete flow of AeDAM and Zigzag are as follows

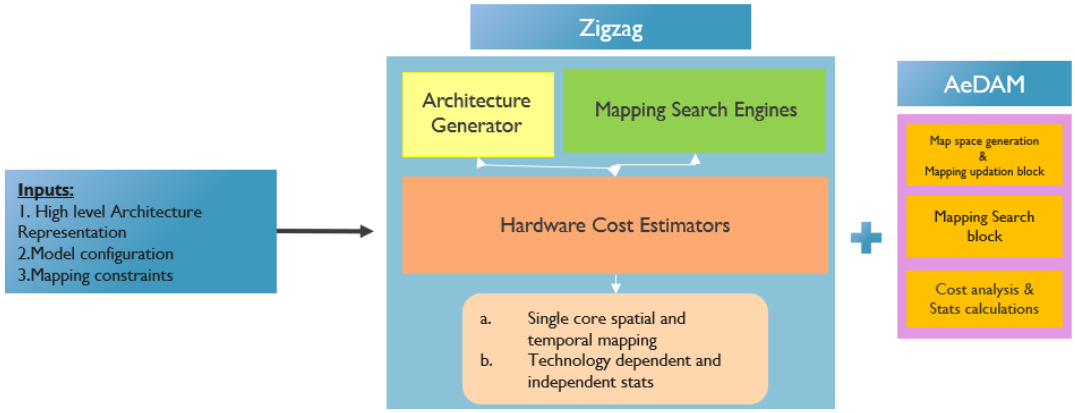


Figure 6.6: Zigzag and AeDAM high level block diagram

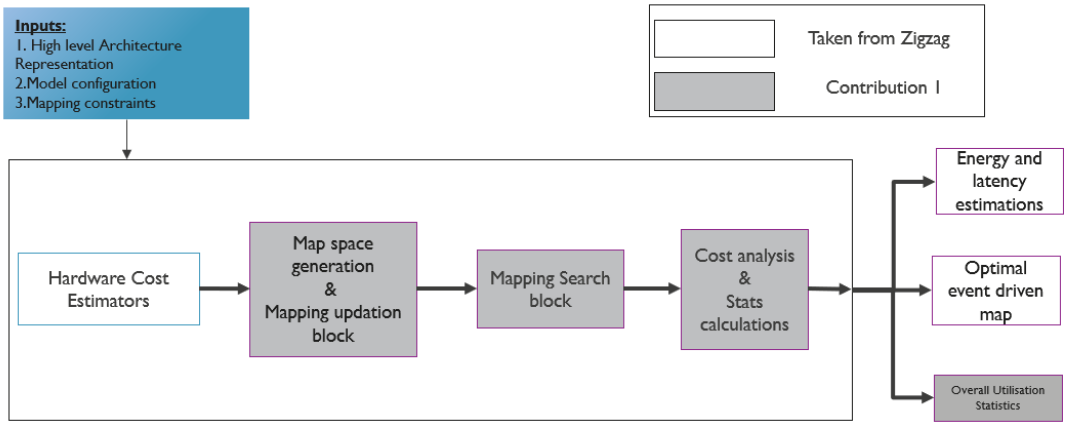


Figure 6.7: Components adapted from ZigZag and integrated into the AeDAM framework

Event-Driven Mapping Space Generated by AeDAM The comparison between exhaustive search and AeDAM's heuristic approach demonstrates a significant reduction in mapping-space complexity while maintaining exploration effectiveness. The results show substantial pruning across all unrolling dimensions. For the K-dimension, the number of possible unrollings drops from 3.38×10^9 (exhaustive) to 6.99×10^7 (AeDAM), i.e. about a 98% reduction. Both the F_X and F_Y dimensions exhibit similar behaviour, decreasing from 9.98×10^9 exhaustive possibilities to 1.15×10^8 heuristic mappings each as shown in Figure 6.8. This dramatic reduction stems from AeDAM's LOMA-based heuristic search strategy, which uses Loop-Partitioning-Function lumping to eliminate redundant and sub-optimal configurations. The framework intelligently prunes the design space by discarding mapping patterns unlikely to yield competitive solutions, thereby focusing computational resources on promising regions while preserving exploration quality for event-driven accelerator optimisation.

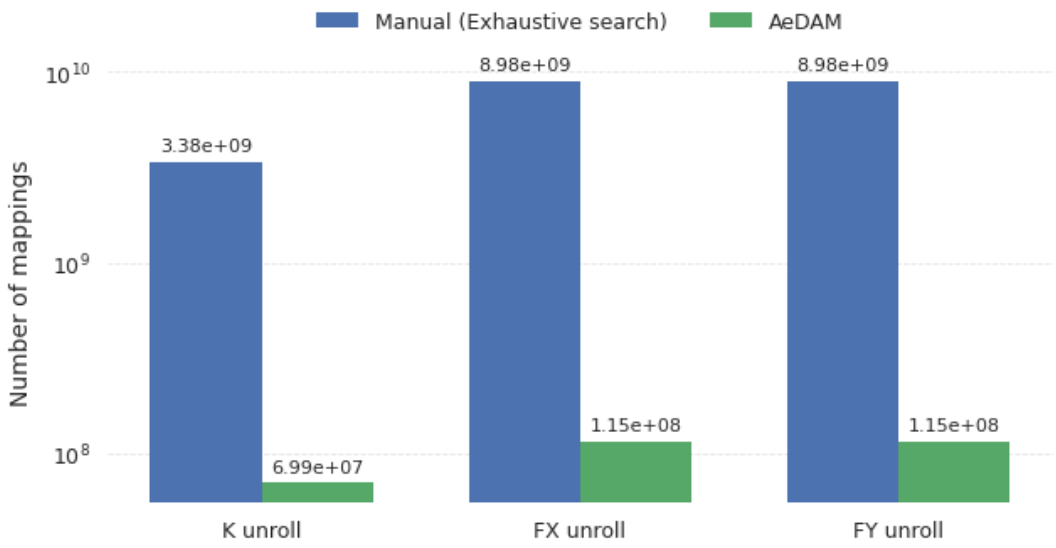


Figure 6.8: Mapping Space Comparison: Exhaustive Search vs. AeDAM Heuristic

Mapping Space Comparison: ZigZag vs. AeDAM The comparison between ZigZag and AeDAM mapping space generation reveals distinct exploration characteristics driven by their respective architectural paradigms as shown in Figure 6.9. ZigZag consistently generates approximately 720-725 mappings for convolutional layers, significantly exceeding AeDAM's 36 mappings per layer. This substantial difference stems from ZigZag's comprehensive exploration of multiple dataflow paradigms, like weight-stationary, input-stationary, and output-stationary approaches, each requiring extensive mapping permutations to optimize data movement and computational scheduling. AeDAM's reduced mapping space reflects its specialized focus on event-driven architectures, where the exploration concentrates on event-driven dataflow rather than exhaustive dataflow combinations, resulting in a more focused but architecturally appropriate exploration scope. The convergence observed in fully connected layers demonstrates how architectural constraints influence mapping space generation. Both frameworks show reduced exploration spaces for these layers, with ZigZag decreasing from 120 mappings in FC1 to minimal options in FC3, while AeDAM maintains consistent exploration patterns aligned with event-driven processing requirements. This behavior indicates that dense computational patterns in fully connected layers provide fewer optimization opportunities regardless of the underlying accelerator paradigm.

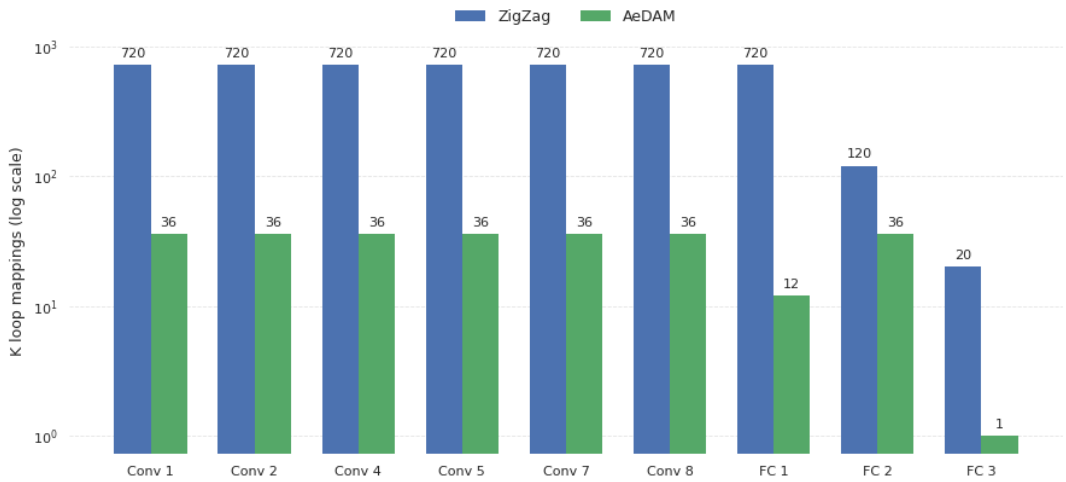


Figure 6.9: Mapping Space Comparison: ZigZag vs. AeDAM across Network Layers

Exploration Time Analysis The exploration time comparison demonstrates AeDAM's superior computational efficiency in identifying optimal mapping configurations compared to ZigZag across different network architectures. For VGGNet optimization, AeDAM completes design space exploration in 100 seconds compared to ZigZag's 254 seconds, representing approximately 60% reduction in exploration time. This performance advantage extends to ResNet-8 evaluation, where AeDAM requires 120 seconds versus ZigZag's 281 seconds, maintaining consistent efficiency improvements of approximately 57% as shown in the Figure 6.10

The substantial time savings achieved by AeDAM result from its focused exploration strategy that targets event-driven mapping configurations rather than exhaustively evaluating multiple dataflow paradigms. While ZigZag must systematically explore weight-stationary, input-stationary, and output-stationary mapping alternatives across extensive search spaces.

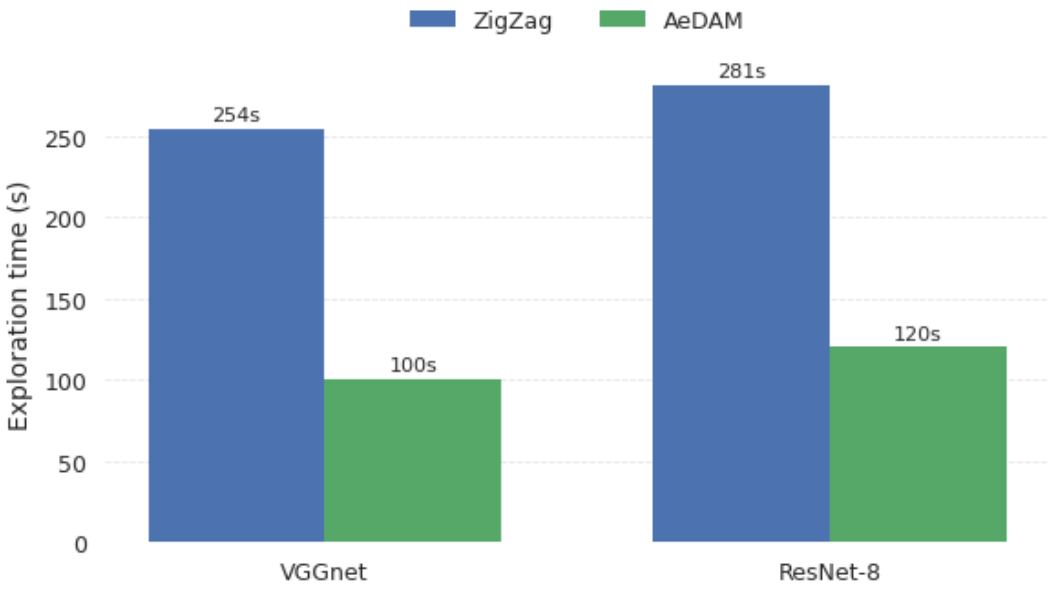


Figure 6.10: Exploration Time Comparison: ZigZag vs. AeDAM

Latency comparision The latency analysis reveals substantial performance advantages for AeDAM's optimal mapping configurations compared to ZigZag's best solutions across different VGGNet layers. For Layer 1, AeDAM achieves 10,939,416 total cycles compared to ZigZag's 12,543,578 cycles, representing approximately 13% latency reduction. The performance advantage becomes more pronounced in Layer 7, where AeDAM requires 116,018,340 cycles versus ZigZag's 241,180,411 cycles, demonstrating a remarkable 52% improvement in execution time as shown in the Figure 6.11 The latency breakdown analysis indicates that true computational cycles dominate the execution profile for both frameworks, with minimal overhead from data movement operations. AeDAM's superior performance stems from its event-driven mapping optimizations. The framework's ability to eliminate redundant operations through selective processing of active events results in more efficient execution sequences compared to ZigZag's frame-based approach that must process complete data structures regardless of sparsity characteristics. The increasing performance gap between deeper layers suggests that AeDAM's optimization strategies become more effective as network complexity increases, providing substantial benefits for computationally intensive layers where efficient resource utilization directly impacts overall system performance.

Energy comparision The energy consumption analysis reveals distinct optimization characteristics between AeDAM and ZigZag mapping strategies, with all measurements expressed in picojoules (pJ). For Layer 1, AeDAM exhibits higher weight read energy consumption at approximately 1950811 pJ compared to ZigZag's 77 pJ, reflecting the event-driven architecture's dynamic weight retrieval requirements for event-driven processing. Conversely, AeDAM achieves superior efficiency in output write operations, consuming 24733 pJ versus ZigZag's 145797 pJ. Layer 7 demonstrates amplified energy distribution patterns, where AeDAM's weight read energy reaches 41617981 pJ compared to ZigZag's 378224 pJ, while maintaining output write consumption at 6458 pJ versus ZigZag's 6683 pJ. This energy profile reflects fundamental architectural differences between event-driven and frame-based processing

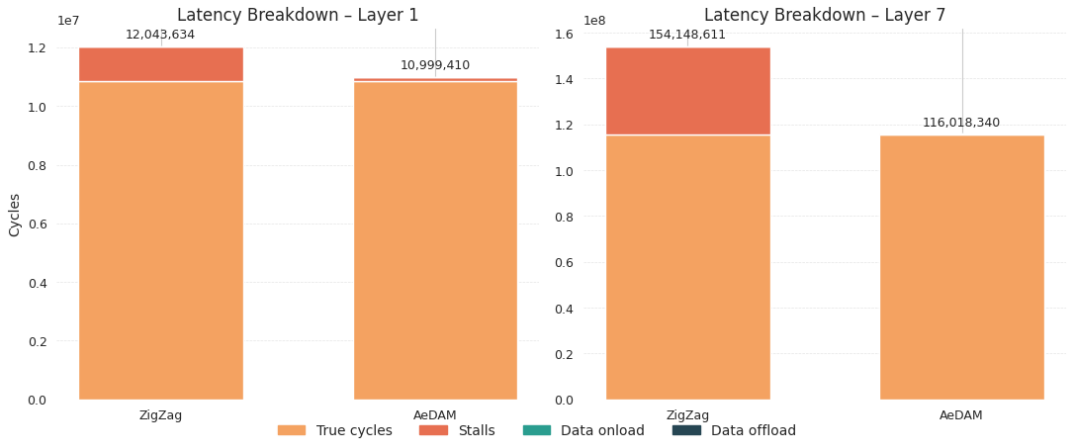


Figure 6.11: Latency Comparison: ZigZag vs. AeDAM for VGGNet Layers 1 and 7

approaches. AeDAM's increased weight access energy results from selective processing that retrieves filter parameters only when corresponding activations are active, avoiding unnecessary computations on zero-valued inputs, but as the input is dense in nature due to which all the weights are supposed to be accessed The energy trade-offs demonstrate that while AeDAM incurs overhead for dynamic weight management, it achieves significant savings in output processing operations. This optimization strategy becomes advantageous for sparse neural network workloads where energy savings from eliminated computations exceed the overhead associated with selective weight access patterns, resulting in overall system efficiency improvements for event-driven accelerator architectures.

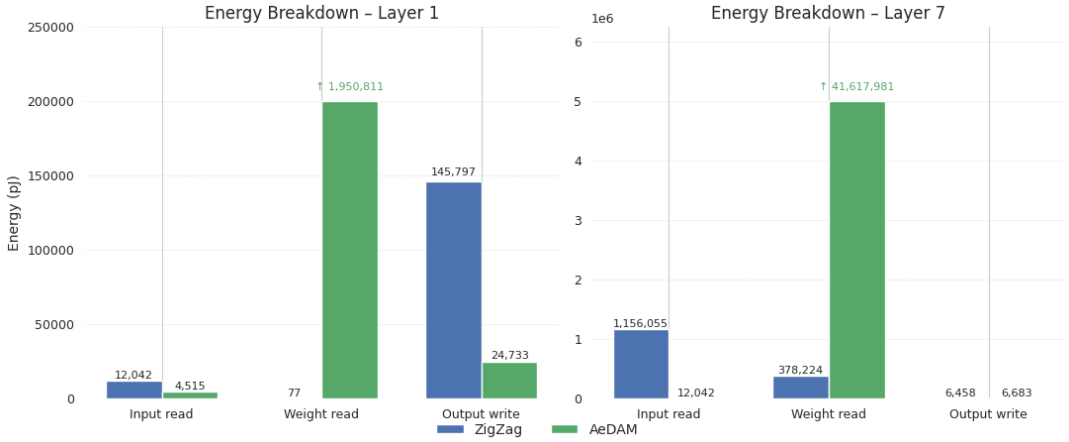


Figure 6.12: Energy Comparison: ZigZag vs. AeDAM for VGGNet Layers 1 and 7

6.3. Case Studies

This section demonstrates the AeDAM framework's application for design space exploration of the SENECA neuromorphic architecture. The investigation systematically explores SRAM memory configurations and processing element arrangements to identify optimal hardware designs for executing VGGNet neural network workloads.

6.3.1. Exploring the best SRAM configurations

The exploration focuses on two primary parameters: SRAM capacity and memory bandwidth. Additional memory characteristics including access latency and read-write energy costs remain constant, as these derive from the underlying memory technology specifications. The study evaluates 40 distinct accelerator architectures with varying SRAM configurations to establish comprehensive energy-latency-area relationships for VGGNet execution.

The results reveal a clear Pareto-optimal frontier where latency reductions require corresponding energy increases. Beyond a critical inflection point, additional energy expenditure yields diminishing latency improvements, indicating

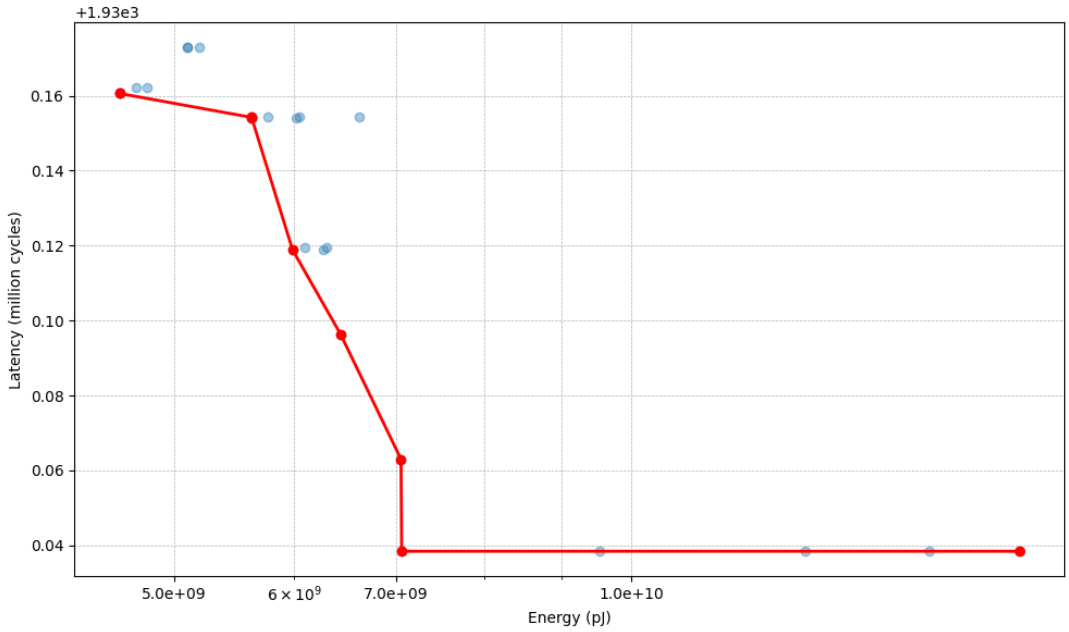


Figure 6.13: Energy vs. Latency Trade-off for SRAM Configuration Exploration

performance saturation effects where memory constraints limit further optimization benefits.

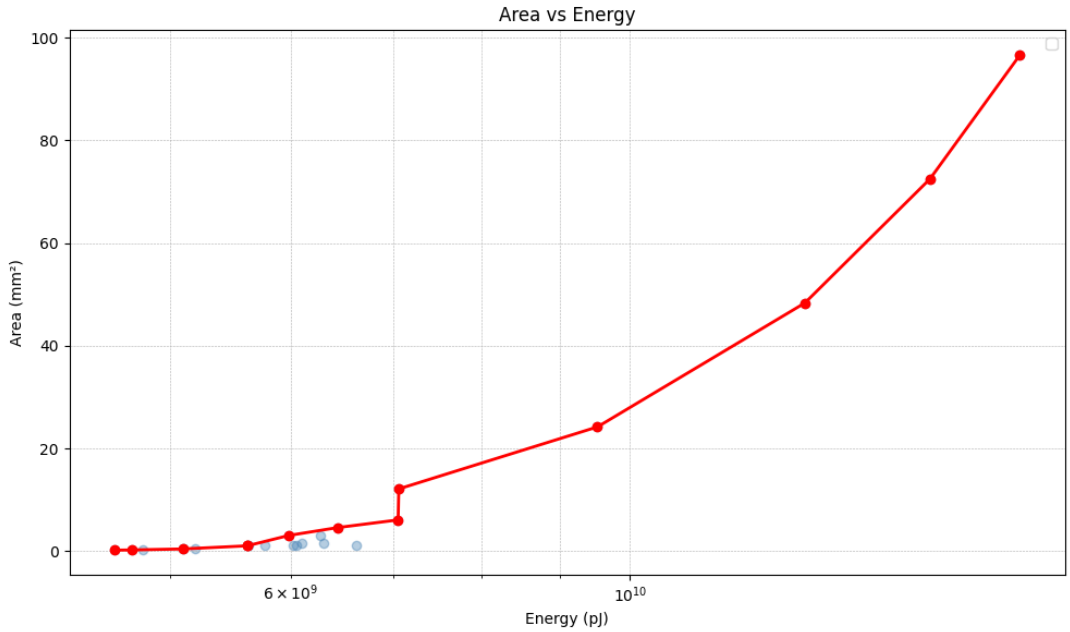


Figure 6.14: Energy vs. Area Trade-off for SRAM Configuration Exploration

The energy-area relationship demonstrates implementation cost implications of different SRAM configurations, enabling assessment of area overhead associated with energy optimization strategies and guiding silicon resource allocation decisions.

The SRAM configuration exploration reveals three distinct operational regimes characterized by fundamentally different energy-latency-area relationships. This analysis provides critical insights for memory hierarchy optimization in event-driven accelerator architectures.

Regime 1: Resource-Constrained Region ($\leq 5.5 \times 10^9$ pJ)

In this low-energy domain, SRAM configurations exhibit minimal area footprint (< 5 mm^2) but suffer from se-

vere latency penalties (≈ 0.16 million cycles). This represents configurations with small, bandwidth-limited memory hierarchies that necessitate frequent external memory accesses, creating a memory-bound bottleneck that dominates execution time despite energy efficiency.

Regime 2: Efficiency Sweet Spot (5.5–6.5×10^9 pJ)
This critical transition zone represents the optimal design space where modest energy increases ($\leq 40\%$) yield dramatic latency reductions ($> 75\%$) with acceptable area overhead (5–15 mm^2). The steep latency improvement suggests this region captures the transition from memory-bound to compute-bound operation, where increased SRAM capacity and bandwidth eliminate the primary performance bottleneck.

Regime 3: Performance-Saturated Region ($> 7 \times 10^9$ pJ)
Beyond the knee point, the relationship fundamentally shifts—latency saturates at ≈ 0.04 million cycles regardless of energy consumption increasing by 70%, while area continues growing superlinearly (reaching > 95 mm^2). This indicates that performance is no longer memory-limited; additional SRAM resources provide diminishing returns as other architectural bottlenecks become dominant.

Design Implications The analysis reveals that optimal SRAM configuration should target the efficiency sweet spot around 5.5×10^9 pJ, achieving near-optimal latency while maintaining reasonable area costs. This characterizes a fundamental constraint in accelerator design where memory hierarchy optimization must balance competing demands of energy efficiency, area utilization, and performance requirements.

6.3.2. Exploring the best PE configurations

The PE configuration study systematically evaluates processing element architectures ranging from single-dimension arrays to multi-dimension configurations across three memory hierarchy levels: 786,432 bits, 2,097,152 bits, and 4,194,304 bits. The evaluated configurations include:

Single-dimension arrays: 8, 16, 32, 64, 128, 256 processing elements
Multi-dimension arrays: 8×3, 8×16, 64×64, 32×128, 16×64, 12×12, 16×16 configurations

Architectural Paradigm Characterization

The comprehensive analysis reveals fundamentally different optimization behaviors between single-dimension and multi-dimension processing element arrays, challenging conventional accelerator design assumptions.

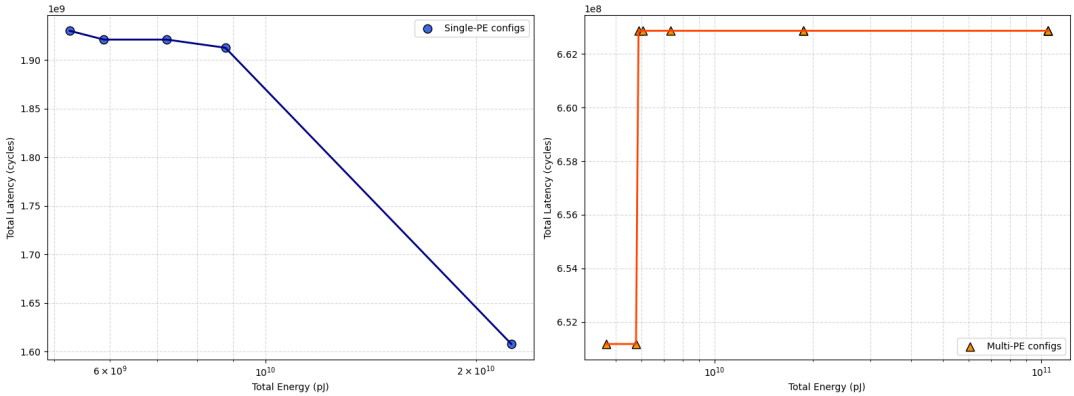


Figure 6.15: Energy-Latency Trade-off for Single-Dimension Arrays (768Kb Memory)

Single-Dimension Arrays: Threshold-Driven Efficiency Paradigm Single-dimension arrays exhibit bistable behavior with distinct operating regimes. The energy-latency relationship demonstrates a flat plateau at approximately 1.92×10^9 cycles followed by a sharp 16% latency reduction at an energy threshold of 1×10^{10} pJ. This threshold behavior indicates two distinct operating modes: an efficiency mode characterized by low energy consumption and high latency, and a performance mode offering high energy consumption with reduced latency.

The energy-area relationship reveals exponential scaling with a critical inflection point around 6×10^9 pJ where area explodes from less than 1 mm^2 to greater than 25 mm^2. This efficiency cliff represents a fundamental design boundary beyond which area costs become prohibitive for marginal performance improvements.

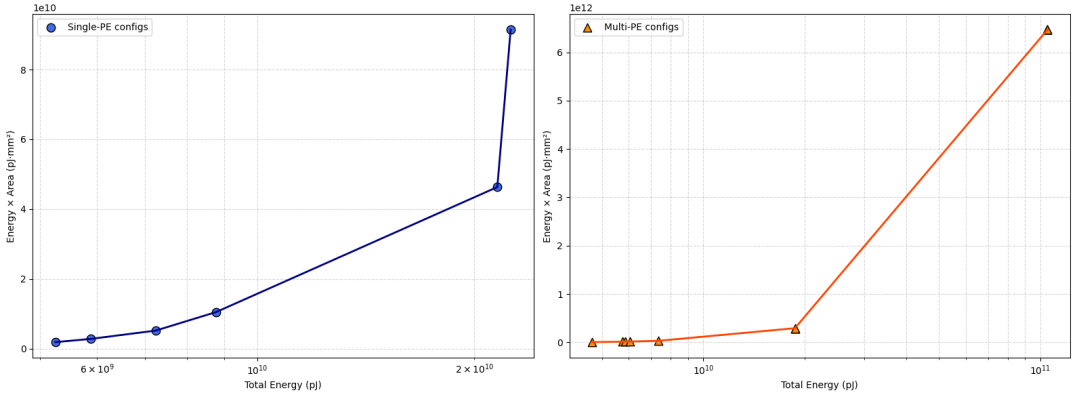


Figure 6.16: Energy-Area Trade-off for Single-Dimension Arrays (768Kb Memory)

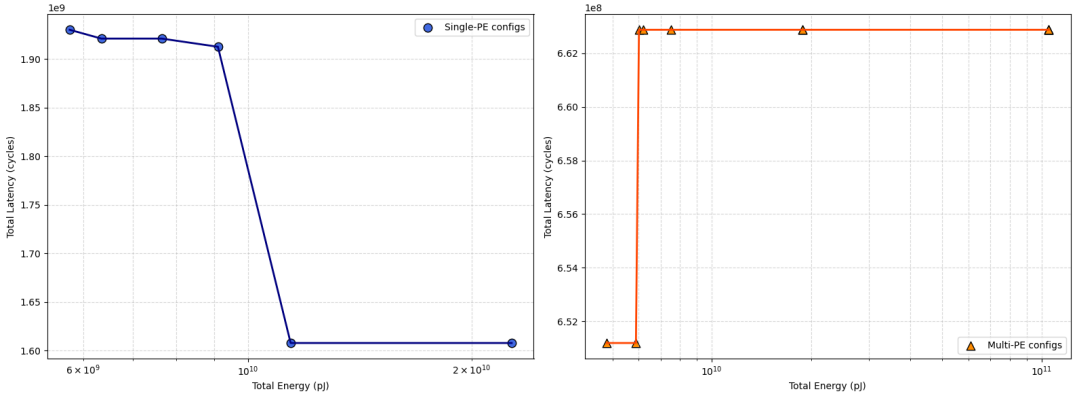


Figure 6.17: Energy-Latency Trade-off for Multi-Dimension Arrays (2Mb Memory)

Multi-Dimension Arrays: Performance-Saturated Paradigm Multi-dimension arrays demonstrate step-function plateau behavior with latency locked at approximately 6.6×10^8 cycles across more than two orders of magnitude in energy consumption. This performance saturation indicates that spatial parallelism effectively eliminates computational bottlenecks, but additional energy investment yields area overhead without latency benefits.

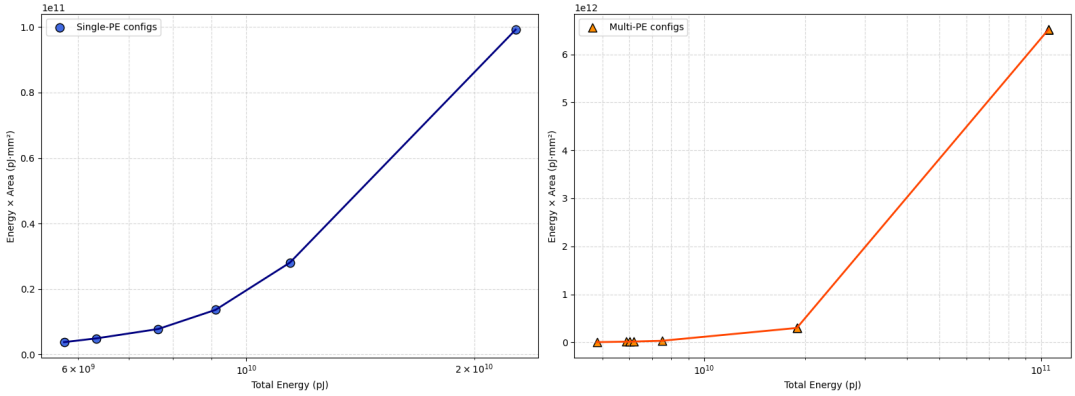


Figure 6.18: Energy-Area Trade-off for Multi-Dimension Arrays (2Mb Memory)

The energy-area relationship exhibits linear scaling with predictable area growth (approximately 0.6 $mm^2/10^{10}$ pJ coefficient), enabling predictable resource allocation for performance-oriented designs.

Critical Discovery: Bifurcated Optimization Landscape

The comprehensive analysis reveals that the design space is fundamentally bifurcated rather than continuous, requiring paradigm-specific optimization strategies:

Energy Efficiency Optimization (Single-Dimension Focus)

- Optimal operating point: $\sim 6 \times 10^9$ pJ, $\sim 1.92 \times 10^9$ cycles, < 1 mm^2
- Efficiency cliff: Beyond 6×10^9 pJ, area costs explode (25× increase) for modest 16% latency improvement
- Design implication: Strict energy budgets favor single-dimension arrays operated in efficiency mode

Performance Optimization (Multi-Dimension Focus)

- Optimal operating point: $\sim 5 \times 10^9$ pJ, $\sim 6.6 \times 10^8$ cycles, < 3 mm^2
- Performance plateau: 2.9× latency advantage maintained across wide energy range
- Design implication: Performance-critical applications should target minimal viable multi-dimension configuration

Memory Capacity Invariance Analysis

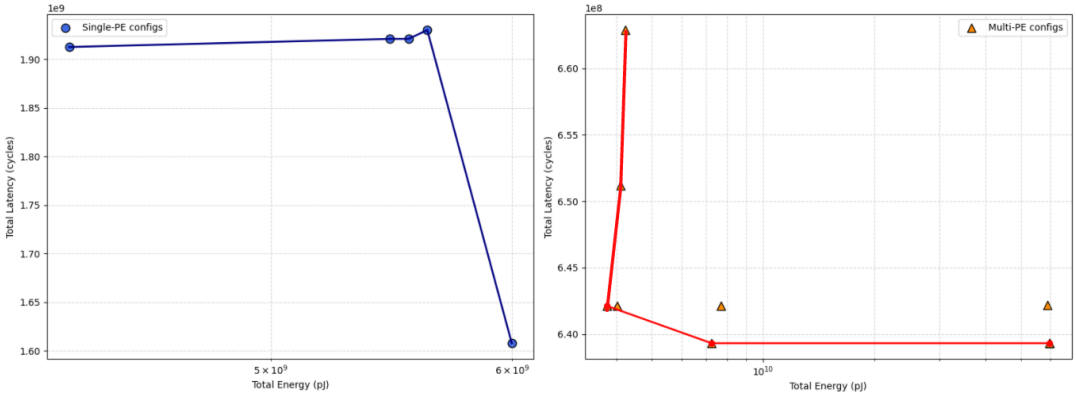


Figure 6.19: Energy-Latency Trade-off for High-Capacity Configuration (4Mb Memory)

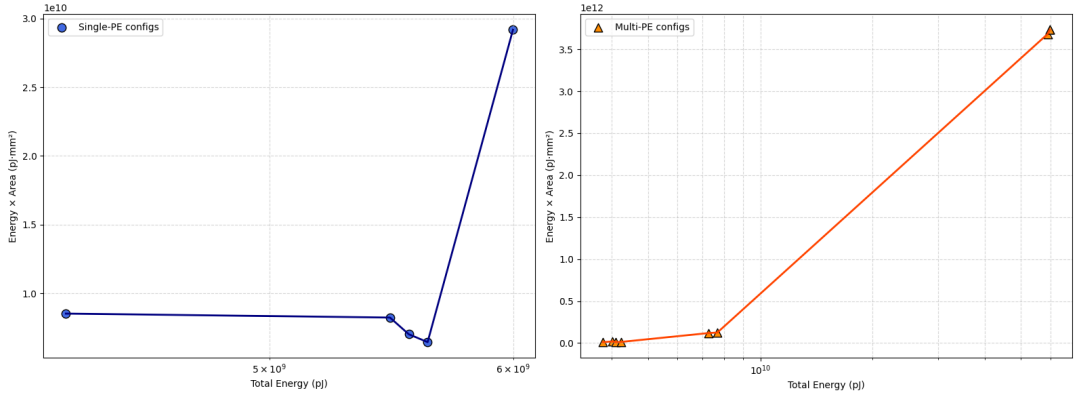


Figure 6.20: Energy-Area Trade-off for High-Capacity Configuration (4Mb Memory)

Architectural behavior patterns remain invariant across memory capacities, demonstrating predictable scaling coefficients:

768KB → 2048KB → 4096KB

- Energy floor improvement: $4.68 \times 10^9 \rightarrow 4.81 \times 10^9 \rightarrow 3.78 \times 10^9$ pJ (19% total improvement)
- Area floor scaling: $0.35 \rightarrow 0.66 \rightarrow 1.15$ mm^2 (3.3× linear scaling)
- Latency characteristics: Identical behavioral patterns with $< 1\%$ variance

Design Space Navigation Framework

The analysis reveals three critical thresholds governing design space navigation:

1. **Single-Dimension Efficiency Cliff ($\sim 6 \times 10^9$ pJ):** Below threshold provides linear energy-area scaling with stable latency; above threshold causes exponential area explosion for modest latency improvement.
2. **Multi-Dimension Entry Point ($\sim 5 \times 10^9$ pJ):** Minimum energy required for viable multi-dimension operation, representing architectural paradigm transition boundary.
3. **Performance Saturation Boundary ($\sim 2 \times 10^{10}$ pJ):** Beyond this point, multi-dimension arrays show diminishing latency returns while area costs continue linear growth.

Conclusion: Threshold-Driven Architecture Selection
The comprehensive PE configuration analysis fundamentally reframes accelerator design from continuous optimization to discrete paradigm selection. The discovery of threshold-driven behaviors, performance plateaus suggests that optimal accelerator design requires architectural regime identification rather than parameter tuning.

For VGGNet specifically, the data conclusively demonstrates that multi-dimensional arrays operating at minimal viable energy ($\sim 5 \times 10^9$ pJ) represent the optimal balance of performance, energy efficiency, and area utilization.

The invariance of behavioral patterns across memory capacities enables predictive design scaling, while the bifurcated optimization landscape suggests that future accelerator architectures should be designed for specific operating regimes rather than attempting to optimize across paradigms.

6.3.3. Optimal configuration for the VGGnet model

Metrics	**Baseline**	**Optimal**	**Changes (Optimal vs Baseline)**
Configuration	SRAM: 2,097,152 Bandwidth: 128 Ports: 4 Latency: 2 $E_{\text{bit\_pj}}$: 0.18	SRAM: 4,194,304 Bandwidth: 128 Ports: 4 Latency: 2 $E_{\text{bit\_pj}}$: 0.225	SRAM capacity increased by 2× Energy per bit increased by 1.25×
Latency (cycles)	1,930,172,842	651,109,076	2.96× reduction in latency
Energy (pJ)	5,750,098,134	4,203,655,191	1.37× reduction in total energy
Area (mm^2)	0.6577	1.3860	2.11× increase in area
E $\times$ D^2	2.14×10^{28}	1.78×10^{27}	12.0× reduction in E×D^2
E $\times$ A	3,781,880,057	5,826,224,444	1.54× increase in E×A

Table 6.2: Comparison between Baseline and Optimal PE configurations.

The optimal design achieves a 2.96× latency reduction and 1.37× energy improvement through strategic memory-compute co-optimization. The decision to increase SRAM capacity from 256KB to 512KB represents a well-positioned Pareto front selection that maximizes performance gains while controlling area overhead.
Technical Justification: The 512KB SRAM configuration operates at the efficiency inflection point where memory bandwidth becomes the primary performance determinant rather than capacity constraints. This positioning enables the multi-dimensional PE array to achieve spatial parallelism effectively, resulting in the dramatic 12× improvement in energy-delay product (E×D²). The controlled 2.11× area increase avoids the exponential scaling that occurs beyond this capacity threshold while delivering the critical latency performance required for real-time VGGnet inference.
Design Space Validation: The optimization validates a multi-objective approach that prioritizes performance improvement over pure energy minimization. The configuration successfully exploits the workload-specific computational patterns of convolutional neural networks, where spatial dataflow architectures combined with appropriately sized memory hierarchies deliver superior energy-performance characteristics.

7

Conclusion and Future work

7.1. Conclusion

This thesis successfully addressed the fundamental inefficiency of existing mapping tools designed for synchronous accelerators when applied to event-driven architectures by developing AeDAM, a specialized design-space exploration framework for event-driven architectures. Key contributions and achievements include:

- **Exploration Performance:** Demonstrated significant performance improvements over the state-of-the-art exploration tool, achieving approximately 2.5× faster exploration times compared to ZigZag for VGGNet (100 s versus 254 s) and similar improvements for ResNet-8 architectures.
- **Mapping Quality:** Validated AeDAM's effectiveness through superior exploration results, with optimal mappings achieving 13% better latency performance for Layer 1 and 52% improvement for Layer 7 of VGGNet compared to ZigZag-generated solutions.
- **Energy Optimization:** Delivered substantial improvements in output energy consumption, with optimized configurations achieving up to 5× reduction in output energy overhead through specialized event-driven optimization strategies.
- **Comprehensive DSE Capabilities:** Encompassed SRAM configuration optimization revealing three distinct operational regimes, processing-element arrangement analysis demonstrating threshold-driven optimization landscapes, and memory-technology evaluation enabling systematic architectural parameter exploration.
- **Practical Validation:** Confirmed framework applicability through the SENECA architecture case study, with the optimal configuration achieving 2.96× latency reduction and 1.37× energy improvement while maintaining a 12× improvement in energy-delay product.

These results establish AeDAM as an essential tool for event-driven accelerator design optimization, providing the research community with specialized capabilities for event driven architecture explorations

7.2. Future Work

Future research directions focus on extending AeDAM's capabilities to address emerging requirements in neuromorphic computing and broader event-driven architectural paradigms:

- **Multicore Architecture Integration:** Develop event-driven dataflow exploration for multicore architectures to enable scalable event driven system design across multiple processing cores while maintaining the efficiency benefits of sparse, asynchronous computation.
- **Sparsity input integrations:** Implement comprehensive weight and input sparsity models to enhance the framework's optimization capabilities by enabling more sophisticated exploitation of computational sparsity patterns characteristic of modern neural network workloads.
- **Adaptive Sparsity Systems:** Develop adaptive sparsity models capable of responding to dynamic input sparsity patterns to provide real-time optimization capabilities essential for deployment scenarios with varying computational loads and input characteristics.

- **Spiking Neural Network Support:** Extend framework support to Spiking Neural Networks to enable comprehensive design-space exploration for biologically-inspired computing systems where temporal dynamics and event-driven processing are fundamental architectural requirements rather than optimization strategies.

These enhancements will strengthen AeDAM's capabilities for event-driven accelerator design space exploration, contributing to improved modeling tools for energy-efficient computing applications in edge devices and neuromorphic systems.

References

[1] 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*, Jan. 2017. DOI: 10.1109/JSSC.2016.2616357. [Online]. Available: https://doi.org/10.1109/JSSC.2016.2616357.

[2] A. Mehonic, D. Ielmini, K. Roy, O. Mutlu, S. Kvatinsky, T. Serrano-Gotarredona, B. Linares-Barranco, S. Spiga, S. Savel'ev, A. G. Balanov, N. Chawla, G. Desoli, G. Malavena, C. M. Compagnoni, Z. Wang, J. J. Yang, S. G. Sarwat, A. Sebastian, T. Mikolajick, and R. Waser, "Roadmap to neuromorphic computing with emerging technologies," *APL Materials*, vol. 12, no. 10, Oct. 2024. DOI: 10.1063/5.0179424. [Online]. Available: https://doi.org/10.1063/5.0179424.

[3] D. Kudithipudi, C. D. Schuman, C. M. Vineyard, T. Pandit, C. Merkel, R. Kubendran, J. B. Aimone, G. Orchard, C. Mayr, R. Benosman, J. Hays, C. Young, C. Bartolozzi, A. Majumdar, S. G. Cardwell, M. Payvand, S. Buckley, S. Kulkarni, H. A. Gonzalez, and S. Furber, "Neuromorphic computing at scale," *Nature*, vol. 637, no. 8047, pp. 801–812, Jan. 2025. DOI: 10.1038/s41586-024-08253-8. [Online]. Available: https://doi.org/10.1038/s41586-024-08253-8.

[4] G. Tang, K. Vadivel, Y. Xu, R. Bilgic, K. Shidqi, P. Detterer, S. Traferro, M. Konijnenburg, M. Sifalakis, G.-J. van Schaik, and A. Yousefzadeh, "Seneca: Building a fully digital neuromorphic processor, design trade-offs and challenges," *Frontiers in Neuroscience*, Jun. 23, 2023. DOI: 10.3389/fnins.2023.1187252. [Online]. Available: https://doi.org/10.3389/fnins.2023.1187252.

[5] Y. Xu, K. Shidqi, G.-J. van Schaik, R. Bilgic, A. Dobrita, S. Wang, R. Meijer, P. Nembhani, C. Arjmand, P. Martinello, A. Gebregiorgis, S. Hamdioui, P. Detterer, S. Traferro, M. Konijnenburg, K. Vadivel, M. Sifalakis, G. Tang, and A. Yousefzadeh, "Optimizing event-based neural networks on digital neuromorphic architecture: A comprehensive design space exploration," *Frontiers in Neuroscience*, Mar. 28, 2024. DOI: 10.3389/fnins.2024.1335422. [Online]. Available: https://doi.org/10.3389/fnins.2024.1335422.

[6] M. Isik, K. Tiwari, M. B. Eryilmaz, and I. C. Dikmen, "Accelerating sensor fusion in neuromorphic computing: A case study on loihi-2," *arXiv*, Aug. 28, 2024. eprint: 2408.16096. [Online]. Available: https://arxiv.org/abs/2408.16096.

[7] A. Parashar, P. Raina, Y. N. Wu, P.-A. Tsai, V. Sze, J. S. Emer, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, and S. W. Keckler, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 24, 2019. DOI: 10.1109/ISPASS.2019.00042. [Online]. Available: https://doi.org/10.1109/ISPASS.2019.00042.

[8] L. Mei, P. Houshmand, V. Jain, S. Giraldo, and M. Verhelst, "Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators," *IEEE Transactions on Computers*, Aug. 1, 2021. DOI: 10.1109/TC.2021.3059962. [Online]. Available: https://doi.org/10.1109/TC.2021.3059962.

[9] Y. N. Wu, P.-A. Tsai, A. Parashar, V. Sze, and J. S. Emer, "Sparseloop: An analytical approach to sparse tensor accelerator modeling," in *Proceedings of the 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2022. DOI: 10.1109/MICRO56248.2022.00096. [Online]. Available: https://doi.org/10.1109/MICRO56248.2022.00096.

[10] M. Shi, S. Colleman, C. VanDeMieroop, A. Joseph, M. Meijer, W. Dehaene, and M. Verhelst, "Cmds: Cross-layer dataflow optimization for dnn accelerators exploiting multi-bank memories," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, Apr. 5, 2023. DOI: 10.1109/ISQED57927.2023.10129330. [Online]. Available: https://doi.org/10.1109/ISQED57927.2023.10129330.

[11] A. Symons, L. Mei, S. Colleman, P. Houshmand, S. Karl, and M. Verhelst, "Towards heterogeneous multi-core accelerators exploiting fine-grained scheduling of layer-fused deep neural networks," *arXiv (Cornell University)*, Dec. 20, 2022. DOI: 10.48550/arXiv.2212.10612. [Online]. Available: https://doi.org/10.48550/arXiv.2212.10612.

[12] J. E. Pedersen, S. Abreu, M. Jobst, G. Lenz, V. Fra, F. C. Bauer, D. R. Muir, P. Zhou, B. Vogginger, K. Heckel, G. Urgese, S. Shankar, T. C. Stewart, S. Sheik, and J. K. Eshraghian, "Neuromorphic intermediate representation: A unified instruction set for interoperable brain-inspired computing," *Nature Communications*, vol. 15, p. 8122, Sep. 2024. DOI: 10.1038/s41467-024-52259-9. [Online]. Available: https://doi.org/10.1038/s41467-024-52259-9.

[13] A. S. Cassidy, J. V. Arthur, F. Akopyan, A. Andreopoulos, K. Appuswamy, P. Datta, M. V. Debole, S. K. Esser, C. Ortega Otero, J. Sawada, B. Taba, A. Amir, D. Bablani, P. J. Carlson, M. D. Flickner, R. Gandhasri, G. J. Garreau, M. Ito, J. L. Klamo, J. A. Kusnitz, N. J. McClatchey, J. L. McKinstry, Y. Nakamura, T. Nayak, B. Risk, K. Schleupen, B. Shaw, J. Sivagnaname, D. F. Smith, I. Terrizzano, T. Ueda, and D. S. Modha, "11.4 ibm northpole: An architecture for neural network inference with a 12 nm chip," *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, Feb. 18, 2024. DOI: 10.1109/ISSCC49657.2024.10454451. [Online]. Available: https://doi.org/10.1109/ISSCC49657.2024.10454451.

[14] M. Yu, T. Xiang, V. P. K. Miriyala, and T. E. Carlson, "Multiply-and-fire: An event-driven sparse neural network accelerator," *ACM Transactions on Architecture and Code Optimization*, 2023. DOI: 10.1145/3630255. [Online]. Available: https://doi.org/10.1145/3630255.

[15] I. Labs. "Taking neuromorphic computing to the next level with loihi 2," Intel. (2021), [Online]. Available: https://download.intel.com/newsroom/2021/new-technologies/neuromorphic-computing-loihi-2-brief.pdf.

[16] Y. Yang, J. S. Emer, and D. Sánchez, "Isosceles: Accelerating sparse cnns through inter-layer pipelining," in *Proceedings of the 29th International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 1, 2023. DOI: 10.1109/HPCA56546.2023.10071080. [Online]. Available: https://doi.org/10.1109/HPCA56546.2023.10071080.

[17] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," *arXiv (Cornell University)*, Apr. 17, 2017. DOI: 10.48550/arxiv.1704.04760. [Online]. Available: https://arxiv.org/abs/1704.04760.

[18] V. J. B. Jung, A. Symons, L. Mei, M. Verhelst, and L. Benini, "Salsa: Simulated annealing based loop-ordering scheduler for dnn accelerators," in *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Jun. 6, 2023. DOI: 10.1109/AICAS57966.2023.10168625. [Online]. Available: https://doi.org/10.1109/AICAS57966.2023.10168625.

[19] S. Kim, Y. Seo, S. Park, and C. S. Park, "Optimization of multi-core accelerator performance based on accurate performance estimation," *IEEE Access*, Feb. 2022. DOI: 10.1109/ACCESS.2022.3151876. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3151876.

[20] L. Waeijen, S. Sioutas, M. Peemen, M. Lindwer, and H. Corporaal, "Convfusion: A model for layer fusion in convolutional neural networks," *IEEE Access*, Dec. 10, 2021. DOI: 10.1109/ACCESS.2021.3134930. [Online]. Available: https://doi.org/10.1109/ACCESS.2021.3134930.

[21] A. Symons, L. Mei, and M. Verhelst, "Loma: Fast auto-scheduling on dnn accelerators through loop-order-based memory allocation," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Jun. 6, 2021. DOI: 10.1109/AICAS51828.2021.9458493. [Online]. Available: https://doi.org/10.1109/AICAS51828.2021.9458493.