UNIVERSITY OF CALIFORNIA,
IRVINE

# Hybrid Bit-parallel and -serial Processing for Flexible Precision AI Accelerator

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Electrical and Computer Engineering

by

Yuhua (Benji) Huang

Thesis Committee:
Assistant Professor Hyoukjun Kwon, Chair
Assistant Professor Sitao Huang
Assistant Professor Salma Elmalaki

2025

# DEDICATION

First and foremost, I would like to express my deepest gratitude to my parents, for their endless love, unwavering support, and for believing in me every step of the way. Their encouragement and sacrifices have made my pursuit of a master's degree possible.
I am also profoundly grateful to my advisor, Prof. Jun, for his outstanding guidance, insightful feedback, and constant patience. His expertise and mentorship have been invaluable to both my academic growth and the success of this work.
Finally, I would like to thank all of my friends who provided help and motivation throughout this journey.

Benji Huang
May/25/2025

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# ACKNOWLEDGMENTS

Portions of Chapter 3 of this dissertation are adapted from our earlier work: "FlexiBit: Fully Flexible Precision Bit-parallel Accelerator Architecture for Arbitrary Mixed Precision AI" by Faraz Tahmasebi, Yian Wang, Benji Y.H. Huang, Hyoukjun Kwon, arXiv:2411.18065 (2024), used with permission. I served as second author on that paper and gratefully acknowledge my co-author's contributions.

# ABSTRACT OF THE THESIS

Hybrid Bit-parallel and -serial Processing for Flexible Precision AI Accelerator

By

Yuhua (Benji) Huang

Master of Science in Electrical and Computer Engineering

University of California, Irvine, 2025

Assistant Professor Hyoukjun Kwon, Chair

Targeting the next generation of AI accelerators, FlexiBit has been proposed as a fully flexible-precision, bit-parallel architecture that efficiently supports both floating-point and integer arithmetic in arbitrary precisions and formats. By enabling true bit-parallel execution for any bitwidth—rather than relying on temporal bit-serial techniques—FlexiBit eliminates compute-unit underutilization and delivers substantial performance-and-area gains.

Building on the FlexiBit foundation, this thesis presents a Hybrid Bit-parallel and Bit-serial Processing Architecture that delivers dynamic precision and performance scalability under tight area and energy constraints. This thesis design a dual-mode processing element that can operate in a wide, low-latency parallel mode or a narrow, energy-efficient serial mode, and rapidly switch between them at runtime. Across precisions from 1 to 64 bits, we systematically measure area, latency, and energy to characterize the full design space. Furthermore, we introduce a word-sliced scheme—partitioning an N-bit operand into K slices of P bits—to interpolate between pure parallel and pure serial extremes. Our results demonstrate that hybrid configurations can achieve near-parallel throughput with area and energy costs approaching those of purely serial designs, offering a practical, adaptable accelerator solution for AI workloads with varying accuracy and efficiency requirements.

# Chapter 1

# Introduction

The rapid evolution of artificial intelligence (AI), particularly in deep learning and large language models (LLMs), has intensified the demand for specialized hardware accelerators capable of delivering high performance with energy efficiency. Traditional fixed-precision architectures often fall short in balancing computational throughput, energy consumption, and adaptability to varying precision requirements inherent in modern AI workloads[5].

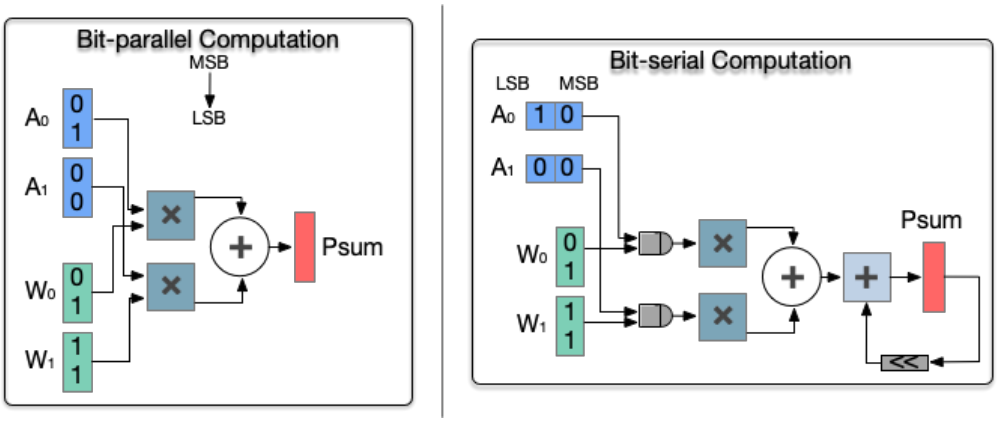



Figure 1.1: BitParallel Computation: Fixed numerical representation length for both A and B and execution time is constant independent of A's representation; Bit-serial computation for A: Execution time depends on A's length[10]

Bit-parallel architectures, as depicted in Figure 2.1, have long been the cornerstone of high-

performance AI accelerators. By processing entire words or vectors simultaneously, these architectures exploit data-level parallelism for substantial speedups. However, their rigidity in supporting standard precision formats (e.g., FP32, FP16, INT8) limits efficiency when addressing diverse and dynamic precision requirements of contemporary AI models [20].

Recent research highlights significant benefits from employing lower and non-standard precisions, such as FP6 and FP5. These formats maintain model accuracy while reducing computational overhead. Nevertheless, the absence of hardware support for arbitrary precisions necessitates frequent redesigns, incurring high costs and limited flexibility [19].

Addressing these constraints, our prior work, FlexiBit, introduced a fully flexible precision bit-parallel accelerator supporting arbitrary mixed-precision formats. Unlike previous bit-serial designs, such as Stripes, which offer flexibility at the expense of performance due to bit-wise temporal processing [10], FlexiBit enables efficient bit-parallel processing across diverse precisions. This capability achieves significant performance advantages, notably 1.66× higher performance per area on GPT-3 at FP6 precision compared to traditional Tensor Core-like architectures, 1.62× higher than BitFusion, and a substantial 3.9× improvement in performance per area compared to state-of-the-art bit-parallel architectures [19].

Moreover, energy-delay product (EDP) comparisons reveal FlexiBit's superior balance of performance and efficiency, achieving a 4.3× lower EDP compared to bit-serial accelerators such as Cambricon-P[7] and 2.5× lower EDP than in-memory bit-serial designs like DRISA[13]. Such findings underline the significant advantages of FlexiBit's architecture for emerging precision and format requirements.

Parallelly, hybrid processing approaches, combining bit-parallel and bit-serial methodologies, have emerged as promising techniques to further optimize energy efficiency and flexibility. This dissertation expands upon this hybrid approach, proposing and evaluating an integrated bit-parallel and bit-serial processing architecture. By strategically leveraging bit-serial pro-

cessing in areas with minimal latency penalties, particularly at low precisions (e.g., 1 to 5 bits), our hybrid architecture maximizes area and power savings while maintaining competitive performance metrics.

Specifically, this hybrid methodology offers negligible overhead at ultra-low precisions, such as 1-bit precision, thus providing almost penalty-free flexibility. The hybrid approach not only extends FlexiBit's foundational benefits but also offers a scalable solution adaptable to various AI workloads, dynamically balancing performance, energy efficiency, and area utilization.

In conclusion, this dissertation thoroughly investigates and demonstrates that a hybrid bit-parallel/bit-serial architecture effectively addresses the increasingly diverse precision requirements of modern AI, delivering versatile and efficient solutions essential for future AI accelerators.

# Chapter 2

# Background and Motivation

## 2.1 Mixed-Precision in modern AI Workloads

The central challenge arises from deciding which parts of a computation can tolerate lower precision without significant loss in final result quality[6]. Exhaustively exploring every possible precision assignment is infeasible—the search space grows exponentially with network depth or algorithmic steps. Furthermore, determining an effective quantization or precision strategy often requires balancing conflicting objectives (accuracy vs. latency vs. energy), each of which depends on both algorithmic characteristics and target hardware features[21][2]

## 2.2 Challenges of Mixed-Precision

### Precision Selection Complexity

Choosing optimal bit-widths per layer or operation involves an enormous combinatorial design space. For deep neural networks, a brute-force search over every layer's precision is

computationally prohibitive [6]. Reinforcement-learning or second-order methods can guide this process but add design complexity and require estimating Hessian spectra or hardware feedback [21].

### Numerical Stability and Accuracy

Lower-precision formats introduce increased quantization and rounding errors. Algorithms sensitive to error propagation—such as iterative solvers or normalization steps—may suffer convergence issues or degraded final accuracy if precision is too low. Mitigation strategies (e.g., loss scaling in mixed-precision training) add overhead and complexity to the software stack [16].

### Hardware Heterogeneity

Different hardware targets (GPUs, TPUs, ASICs) support varying mixed-precision capabilities. For example, NVIDIA Tensor Cores deliver high throughput in mixed FP16/FP32 operations but require careful programming to avoid precision-induced errors [15][17]. Automated frameworks must account for architecture-specific latency and power profiles, further expanding the optimization space.

### Control and Runtime Overheads

Dynamic mixed-precision schemes that adjust precision on-the-fly incur control logic and potential stalls. Static approaches minimize runtime overhead but may be overly conservative, forgoing potential speed or energy gains [8]. Compiler-generated control signals can reduce overhead but require sophisticated compilation pipelines.

## 2.3 Importance of Mixed-Precision

### Throughput and Energy Efficiency

Reducing operand bit-widths directly decreases memory bandwidth and arithmetic complexity, often yielding 2×-4× speedups on modern accelerators with proportional energy savings. Mixed-precision training has demonstrated near-2× memory reduction and significant performance boosts without sacrificing model accuracy.

### Scalability for Large Models

State-of-the-art language and vision models (e.g., GPT, ResNet) contain billions of parameters. Uniform high precision incurs prohibitive power and latency costs. Mixed-precision quantization (e.g., HAWQ) enables scaling these models to production deployment by compressing less-sensitive layers more aggressively.

### Flexible Hardware Utilization

Support for multiple precisions allows a single accelerator design to target a broad workload spectrum—training, inference, edge, or datacenter—maximizing utilization of scarce silicon resources [15]. Automated quantization frameworks (e.g., HAQ) exploit this flexibility to tailor bit-widths for specific hardware constraints, achieving 1.4-1.95× latency reduction and nearly 2× energy savings[11].

**Algorithmic Adaptivity**

Mixed-precision arithmetic underpins adaptive numerical methods where initial coarse computations are refined at higher precision, accelerating convergence for scientific simulations and optimization routines. This adaptivity is increasingly vital for real-time applications in robotics, finance, and scientific computing.

The diverse requirements of AI workloads necessitate support for multiple precision formats. Flexible precision allows hardware to adapt to the specific needs of different layers within a neural network, optimizing performance and energy efficiency. For example, early layers in a network might tolerate lower precision, while later layers require higher precision to maintain accuracy.

Implementing flexible precision in hardware accelerators poses challenges, including increased design complexity and potential under-utilization of compute resources. However, the benefits in terms of reduced memory bandwidth, lower power consumption, and improved throughput make it a critical feature in modern AI accelerators.

## 2.4 Bit-parallel and Bit-serial Architectures

Bit-parallel processing architectures are designed to handle entire words or vectors of data in a single clock cycle. This approach leverages wide datapaths and parallel compute units, enabling high throughput and efficient utilization[18] of hardware resources when operating on standard, fixed-precision formats such as FP32, FP16, or INT8. The main advantage of bit-parallel architectures lies in their ability to exploit data-level parallelism, making them well-suited for workloads with uniform precision requirements.

On the other hand, bit-serial processing architectures operate on data one bit at a time,

sequentially processing each bit position across multiple cycles[4][9]. While this results in lower raw performance compared to bit-parallel designs, bit-serial architectures offer significant flexibility. They can efficiently support arbitrary and mixed-precision formats without the need for specialized hardware for each precision. This adaptability is particularly valuable for AI workloads that benefit from dynamic precision scaling to balance accuracy and efficiency[22][12].

Hybrid architectures aim to combine the strengths of both approaches. By integrating bit-parallel and bit-serial processing elements, a hybrid architecture can deliver high throughput for standard precisions while retaining the flexibility to efficiently handle non-standard or mixed-precision formats. This enables AI accelerators to dynamically adapt to the precision requirements of different layers or operations within a model, optimizing both performance and energy efficiency.

## 2.5 Floating-Point Formats and Flexible Precision in AI Accelerators

Floating-point (FP) arithmetic is fundamental in AI computations due to its ability to represent a wide dynamic range of values efficiently. The IEEE 754 standard[1] [14] [3]defines several FP formats, as in Figure 2.1:

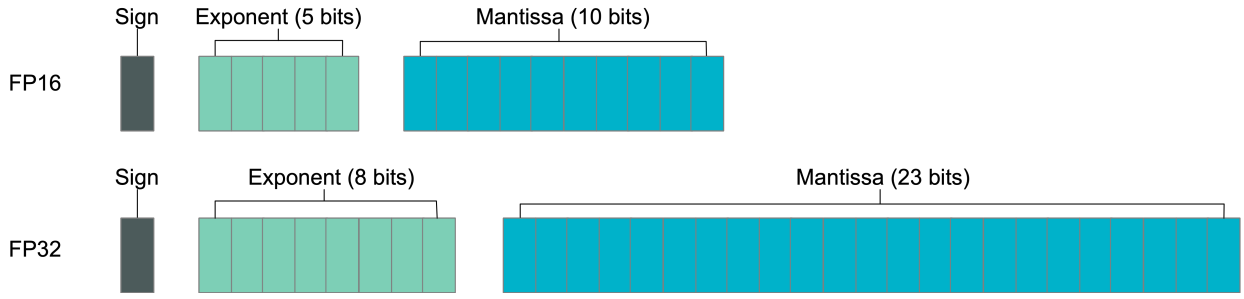



Figure 2.1: FP Formats example.[1]

These formats balance the trade-off between precision, dynamic range, and computational efficiency. For instance, BF16 maintains the dynamic range of FP32, making it suitable for training deep neural networks, while FP8 is often used in inference scenarios where reduced precision is acceptable[3].

## 2.6 Previous Work: Flexibit Architecture

The original FlexiBit design was a bit-parallel floating-point accelerator tailored specifically for efficient AI inference workloads, capable of dynamically supporting a wide range of numeric formats and precisions. FlexiBit's primary architectural innovation lies in its ability to process floating-point operations at arbitrary precision, significantly optimizing computation for varying model requirements and data precision needs encountered across diverse AI workloads.

As in Figure 2.2, FlexiBit consists of multiple dedicated arithmetic and logic units, each designed for parallel operation on full-width operands. The key computational blocks include the **FBRT** mantissa multiplier, **Exponent Adder**, **concatenation tree**, **normalization unit**, and **accumulator**. The mantissa multiplier uses a novel Flexible Bit Reduction Tree (FBRT) architecture, enabling efficient and parallel multiplication of mantissa operands with arbitrary lengths without underutilizing the computational resources. Meanwhile, the exponent adder and normalization units ensure correct floating-point arithmetic operations across variable numeric formats.

Another significant innovation is FlexiBit's sophisticated yet efficient control logic. By employing compiler-generated control signals that configure arithmetic precision at compile-time, FlexiBit significantly reduces runtime overhead, thereby ensuring high throughput and low latency operation. The design also adeptly handles implicit leading bits in floating-

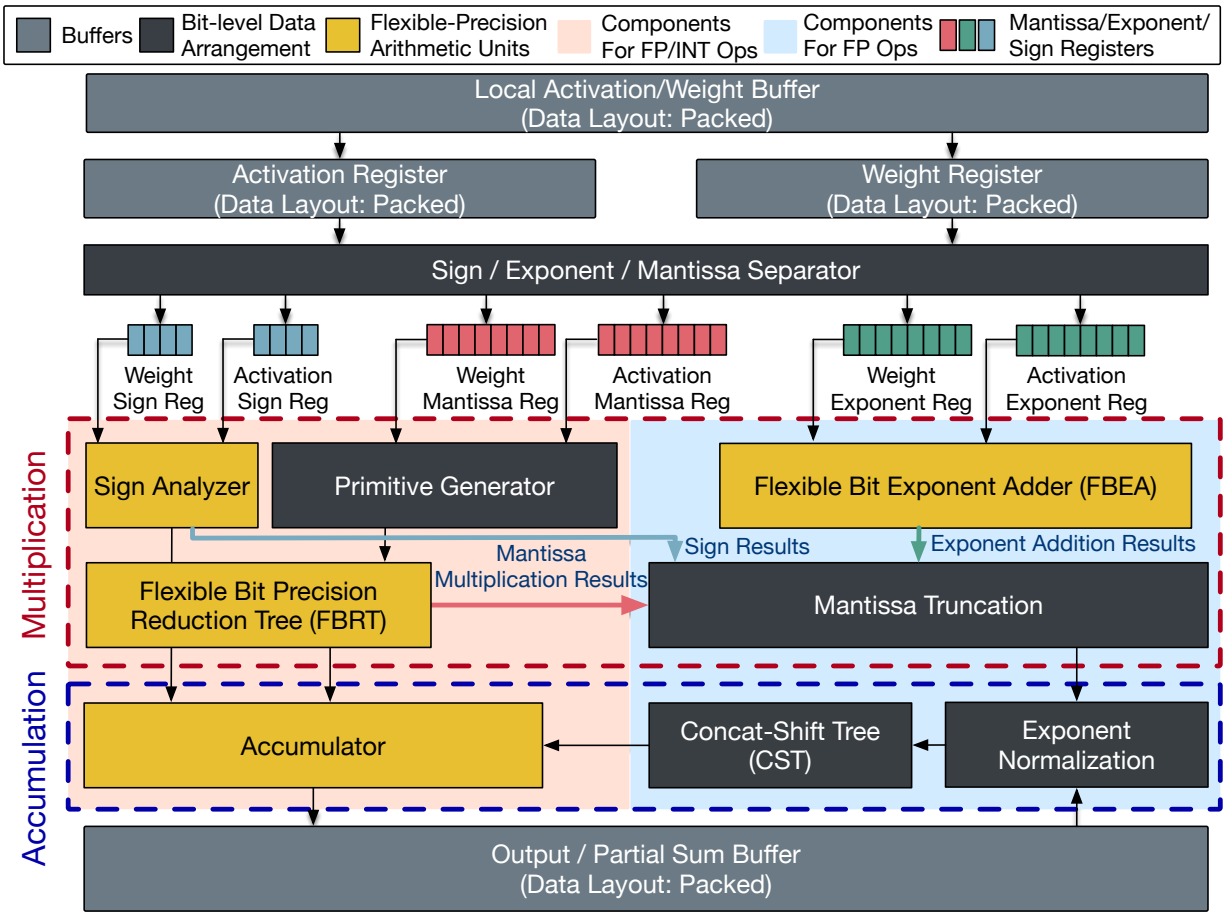



Figure 2.2: The micro architecture of the original Flexibit PE.[19]

point representations through clever post-processing strategies, minimizing area overhead and maximizing computational efficiency.

## 2.7 Motivation for Bit-Serial Version

To better improve on-chip area resource utilization and optimize accelerator performance across diverse operational contexts, this thesis adapts the Flexibit architecture to operate in a bit-serial manner. Unlike traditional bit-parallel processing, which handles full-width operands simultaneously, bit-serial processing sequentially manages operands bit-by-bit across multiple clock cycles. This adaptation significantly reduces area overhead, making

it particularly suitable for resource-constrained environments.

Additionally, the thesis explores the broader accelerator design space to systematically identify effective computational patterns, focusing on the trade-offs between area, power consumption, and performance. By thoroughly evaluating various configurations, the Flexibit architecture is optimized through hybrid combinations of parallel and serial processing tailored for different precision requirements. This methodology ensures the accelerator's adaptability to specific AI workloads, balancing computational efficiency and resource utilization to deliver optimal performance metrics under varying precision demands.

**Goals of Bit-Serial Adaptation**

- Substantially reduce cell area.
- Enable flexible, runtime-adaptive precision.
- Trade-off among on-chip area, power, and performance, find optimal configurations for different precision inputs.

# Chapter 3

# Hybrid Bit-Parallel and Bit-Serial Architecture

## 3.1 Design Methodology: From Bit-Parallel to Hybrid Bit-Serial

To enhance flexibility and area efficiency, we systematically adapted the FlexiBit Processing Elements (PEs) from a purely bit-parallel architecture to a hybrid architecture supporting both bit-parallel and bit-serial modes. Each arithmetic and data processing block within a PE is encapsulated within a dual-mode wrapper, enabling dynamic or static selection between:

- **Bit-parallel mode**: Retains the original parallel combinational logic.
- **Bit-serial mode**: Implements serialized operations, sequentially processing bits across multiple clock cycles.

### 3.1.1 Arithmetic and Data Processing Blocks Targeted for Hybridization

We applied this dual-mode hybridization approach to the following critical functional blocks:

- Mantissa Multiplier
- Exponent Adder
- Normalization Unit
- Concatenation Shift Tree
- Accumulator
- Primitive Generator
- Input Organizer
- Sign Analyzer

### 3.1.2 Dual-Mode Wrapper Implementation

Each targeted arithmetic unit employs a dual-mode wrapper, constructed using SystemVerilog's `generate` constructs controlled by a parameterized flag `PARALLEL`. This parameter selects the operational mode at compile-time:

```
module DualModeExample (
parameter bit PARALLEL = 1
)(
input  logic clk, reset, start,
```

```
output logic done,
input  logic \[N-1:0] data\_in,
output logic \[N-1:0] data\_out
);

generate
if (PARALLEL) begin
// Original parallel combinational logic
assign data\_out = parallel\_logic(data\_in);
assign done = 1'b1; // Single-cycle completion
end else begin
// Bit-serial sequential logic
always\_ff @(posedge clk or posedge reset) begin
if (reset)
/\* initialization logic */;
else
/* bit-serial FSM operations \*/;
end
end
endgenerate
endmodule
```

### 3.1.3 Bit-Serial Finite State Machine (FSM) Implementation

For bit-serial mode, each arithmetic unit contains a dedicated Finite State Machine (FSM) to sequentially control operand processing. Typical FSM states include:

- `IDLE`: Await a `start` pulse.
- `LOAD`: Latch operands into internal registers.
- `PROCESS`: Execute serialized computations bit-by-bit.
- `DONE`: Output the results and signal completion via the `done` pulse.

### 3.1.4 Transition from Wide Data Paths to Narrow Bit-Serial Pipelines

To achieve efficient serialization, we replaced wide combinational data paths with narrow bit-level pipelines:

- Multipliers are implemented as shift-and-add serial multipliers.
- Adders adopt bit-serial ripple-carry configurations.
- Concatenation shift trees serialize their shifting operations, precisely indexing bits sequentially.

## 3.2 Detailed Hybrid Bit-Serial Architecture Overview

### 3.2.1 Architecture Hierarchy

Figure 3.1 illustrates the current hybrid bit-parallel and bit-serial PE architecture, highlighting its modular structure and interconnections among functional units.

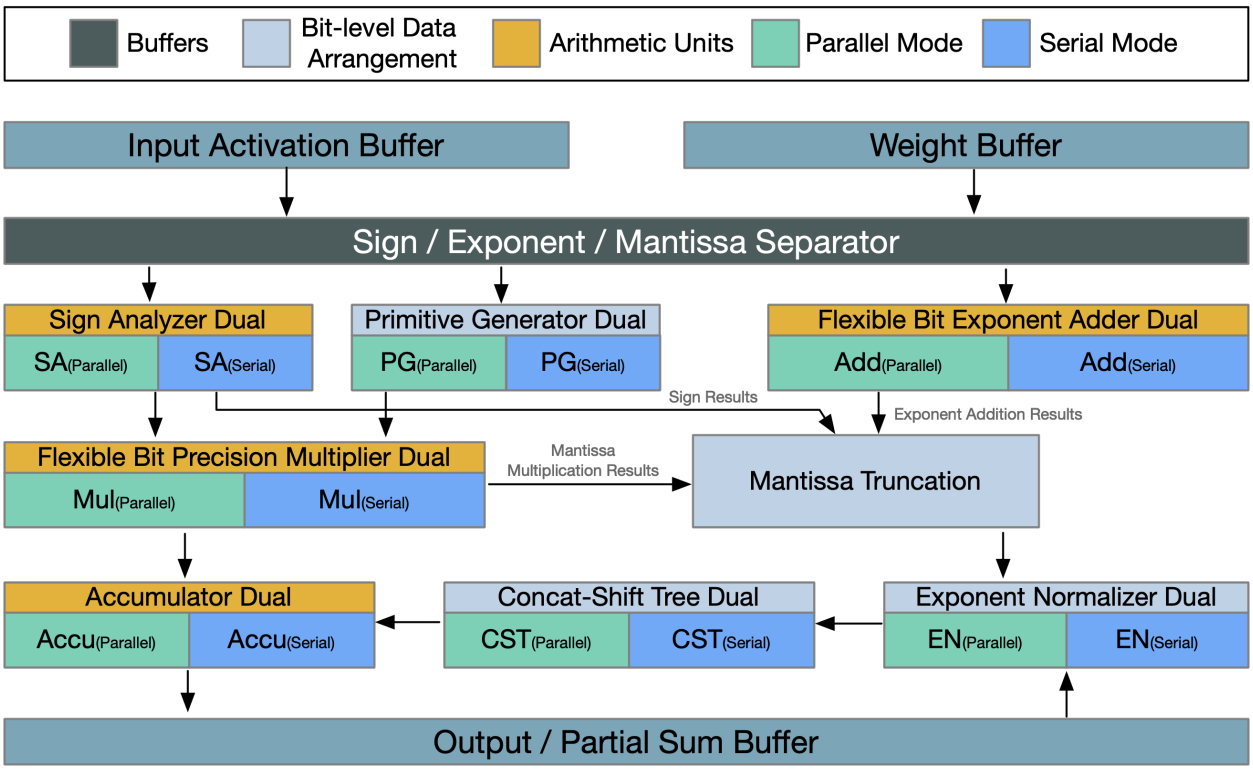



Figure 3.1: Hybrid Bit-Parallel and Bit-Serial PE Architecture

### 3.2.2 Core Functional Units in Bit-Serial Mode

#### Mantissa Multiplier (Shift-and-Add Multiplier)

- Multiplicand and multiplier operands shifted sequentially.
- Conditional addition performed each cycle to accumulate the result.
- Computation spans $N$ cycles, proportional to operand width.

#### Exponent Adder (Ripple-Carry Adder)

- Sequential bit-wise addition from Least Significant Bit (LSB) to Most Significant Bit (MSB).

- Carries propagated across clock cycles, resulting in latency proportional to operand width.

**Exponent Normalizer**

- Sequential bit-wise shifting to normalize exponent values according to floating-point standards.

**Concatenation Shift Tree (Serialized)**

- Serialized concatenation and shifting using an FSM to precisely index and shift bits.
- Boundary bits handled via neighbor connections.

**Accumulator (Ripple-Carry Accumulator)**

- Accumulation performed sequentially at the bit level.
- Precision checkpoints included to reset carry propagation.

## 3.3 Latency Measurement Methodology

Latency measurements across precision levels and operating modes follow a systematic approach:

1. Trigger each block's `start` input for precisely one clock cycle.
2. Count cycles until the `done` signal activates.
3. Log the measured latency for each precision and configuration.

# Chapter 4

# Bit-parallel vs -serial Design Space Exploration - Area & Power & Latency

## 4.1 Experiment Setup

In this chapter, we will present the results of our experiments using the proposed design space exploration framework. The experiments are conducted on a 15nm library, and we will analyze the impact of different configurations on area, power, and latency.

In our hybrid bit-parallel/bit-serial accelerator, each processing element (PE) comprises eight functional submodules as in Table 4.1.

Each submodule can be independently configured to operate in bit-parallel or bit-serial mode via an 8-bit configuration vector—logic "1" selects bit-parallel, "0" selects bit-serial—yielding $2^8 = 256$ unique PE configurations. For example, the binary string `'11111111'` denotes a

fully bit-parallel setup, whereas `'00000000'` denotes a fully bit-serial setup and '10100000' gives us a configuration of bit-parallel Multiplier and Accumulator while the other module stays bit-serial. During simulation, we sweep all 256 configurations, applying each configuration across the array of PEs. This systematic exploration enables precise characterization of area, power, and performance trade-offs for different precision requirements.

Table 4.1: Processing Element Submodules and Their Functions

| **Module** | **Abbrev.** | **Function Description** |
|---|---|---|
| Multiplier | Mul | Executes bit-parallel mantissa multiplication via the Flexible Bit Reduction Tree (FBRT), aggregating cross-product primitive bits in a fat-tree structure for arbitrary mantissa lengths. |
| Adder | Add | Performs exponent addition with flexible bit-width support using a bit-serial ripple-carry adder, enabling variable-precision format operations. |
| Accumulator | Accu | Temporally accumulates partial products over multiple cycles to support both bit-parallel and bit-serial dataflows. |
| Concatenation-Shift Tree | CST | Serializes the concatenation and shifting of mantissa and exponent segments using additional lateral links for bit-level reconfigurability. |
| Exponent Normalizer | EN | Normalizes exponent results by sequentially shifting and adjusting bit positions to enforce the correct floating-point format. |
| Input Organizer | IOrg | Sorts and aligns activation and weight primitives by bit index, preparing the bit streams for FBRT and downstream arithmetic units. |
| Primitive Generator | PG | Generates the set of cross-product AND primitives (including implicit leading ones) required by FBRT for flexible-precision multiplication. |
| Sign Analyzer | SA | Extracts and propagates the sign bit through the PE pipeline, ensuring correct sign handling in both bit-parallel and bit-serial modes. |

## 4.2 Measurement Methodology

1. **Measurement Methodology**

   We synthesized a single Processing Element (PE) for all 256 hybrid bit-parallel/bit-serial configurations and for operand widths $w = 1$ to 16. To visualize this three-dimensional dataset compactly, each 8-bit configuration string is split into two 4-bit nibbles:

   - **X-axis:** bits for {`Mul`, `Add`, `Accu`, `CST`}
   - **Y-axis:** bits for {`EN`, `IOrg`, `PG`, `SA`}

   This yields a $16 \times 16$ heatmap for each width, where color encodes total cell area in ($\mu$m$^2$) and total cell power in (mW). Figure 4.1 shows these heatmaps for $w = 1, \ldots, 16$, and Figure 4.2 shows the corresponding power heatmaps.

## 4.3 Area and Power Result Analysis

### 4.3.1 Area and Power Result Analysis when bit width=1

At precision $w = 1$ as in Figure 4.3, we observe the following key patterns in the total PE area and power:

1. **Multiplier mode has zero impact.** Toggling the multiplier between bit-parallel and bit-serial yields no change in total area or power. For $w = 1$, the multiplier's network is trivial, so pipelining it does not save—or cost—any silicon or dynamic energy.
2. **Per-module area and power savings are nearly constant.** Let $A_0 \approx 156.7457\,\mu\text{m}^2$ be the area and $P_0$ the power with all modules in parallel mode. If we denote by $\delta A_i$

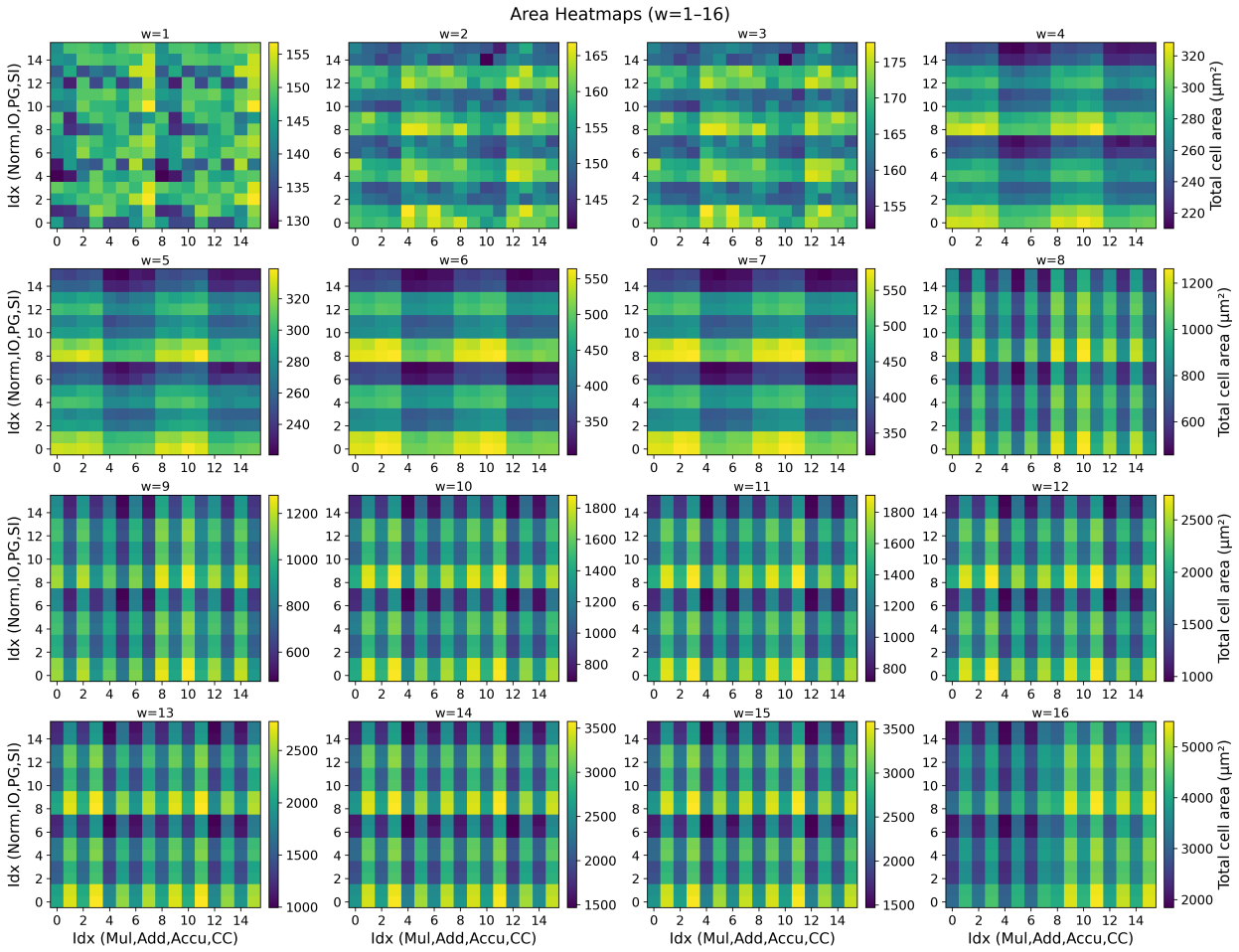



Figure 4.1: Area results heatmap for all width, where the binary form of axis tiks represent the configuration of each module. The area unit is in square micrometers.

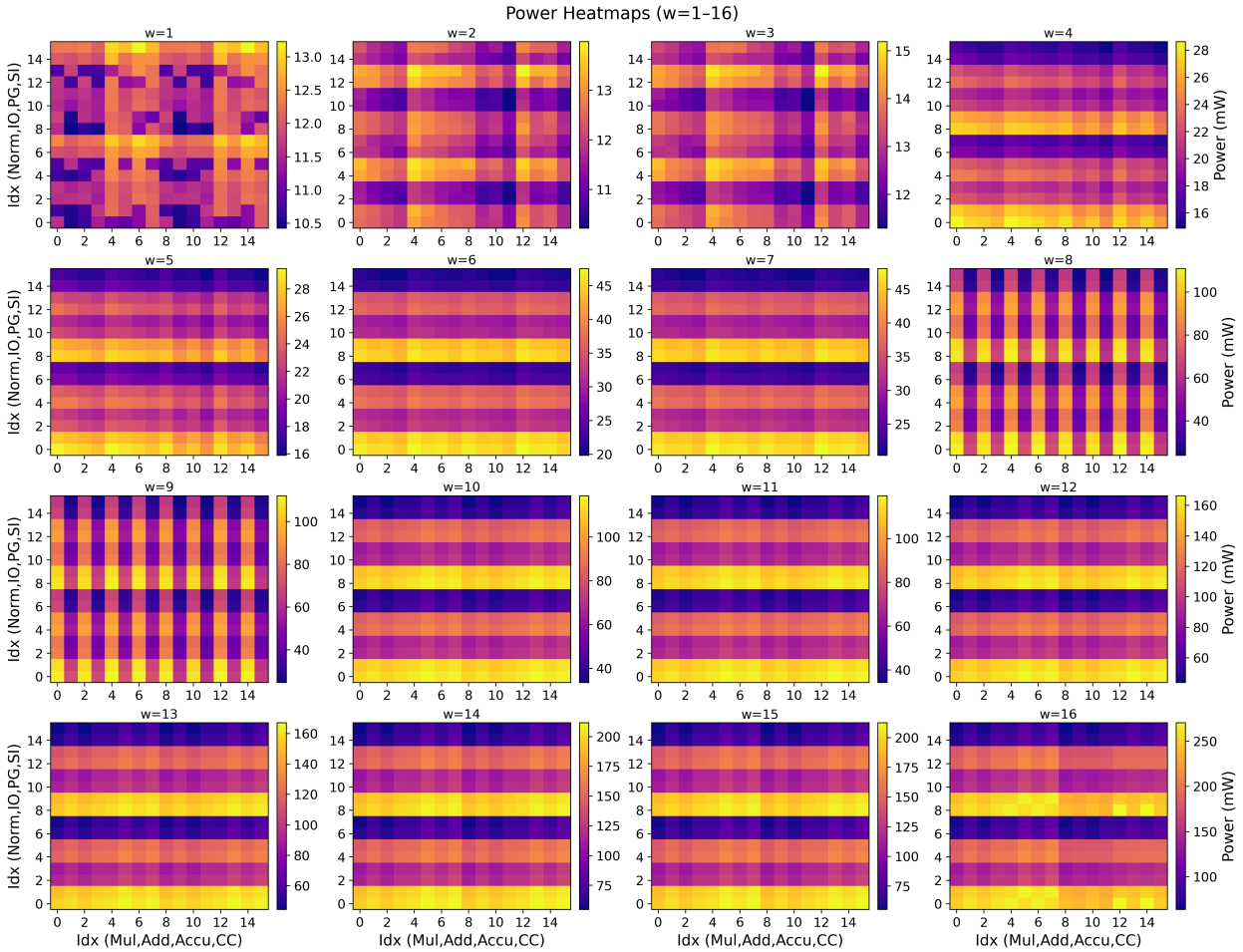
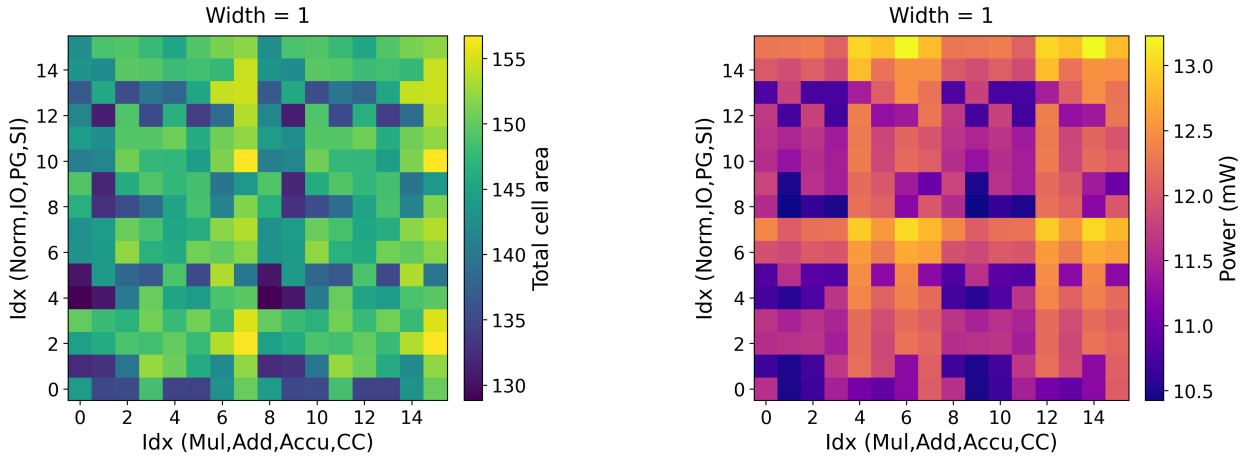



Figure 4.2: Impact to power when toggling serial/parallel mode of each PE -module




(a) Area result for bit width = 1

(b) Power result for bit width = 1

Figure 4.3: Comparison of area and power heatmaps at bit width = 1.

| Module | width=1 | | width=2 | | width=3 | | width=4 | |
|---|---|---|---|---|---|---|---|---|
| | $\overline{\Delta A}$ | $\max \Delta A$ | $\overline{\Delta A}$ | $\max \Delta A$ | $\overline{\Delta A}$ | $\max \Delta A$ | $\overline{\Delta A}$ | $\max \Delta A$ |
| Mul | 0.000 | 0.000 | 2.389 | 7.127 | 2.377 | 7.127 | 3.110 | 10.764 |
| Add | 6.172 | 18.776 | 3.673 | 10.322 | 3.734 | 10.322 | 23.300 | 31.359 |
| Accu | 6.860 | 19.661 | 2.994 | 10.125 | 2.981 | 10.125 | 4.489 | 14.156 |
| CST | 5.104 | 17.302 | 2.452 | 8.454 | 2.474 | 8.454 | 2.644 | 8.651 |
| EN | 4.328 | 16.859 | 2.245 | 9.535 | 2.281 | 9.535 | 3.202 | 8.946 |
| IOrg | 5.285 | 15.974 | 2.160 | 8.946 | 2.194 | 8.946 | 28.393 | 35.734 |
| PG | 7.104 | 17.744 | 9.387 | 18.334 | 9.439 | 18.334 | 41.708 | 50.430 |
| SA | 4.610 | 15.925 | 2.463 | 9.732 | 2.479 | 9.732 | 9.827 | 18.039 |

| Module | width=5 | | width=6 | | width=7 | | width=8 | |
|---|---|---|---|---|---|---|---|---|
| | $\overline{\Delta A}$ | $\max \Delta A$ | $\overline{\Delta A}$ | $\max \Delta A$ | $\overline{\Delta A}$ | $\max \Delta A$ | $\overline{\Delta A}$ | $\max \Delta A$ |
| Mul | 3.164 | 10.764 | 0.018 | 1.180 | 0.000 | 0.000 | 87.500 | 97.223 |
| Add | 23.247 | 31.359 | 48.691 | 56.623 | 48.657 | 56.623 | 72.040 | 83.362 |
| Accu | 4.540 | 15.286 | 9.357 | 15.581 | 9.297 | 15.581 | 29.165 | 40.157 |
| CST | 2.674 | 9.535 | 2.794 | 8.847 | 2.818 | 8.847 | 299.027 | 307.151 |
| EN | 3.201 | 8.946 | 2.971 | 10.371 | 2.994 | 9.191 | 3.883 | 13.369 |
| IOrg | 28.324 | 35.734 | 68.417 | 77.119 | 68.348 | 77.119 | 121.641 | 130.843 |
| PG | 41.636 | 50.430 | 120.362 | 127.894 | 120.392 | 127.893 | 172.818 | 184.566 |
| SA | 9.896 | 17.596 | 9.916 | 17.498 | 9.890 | 17.498 | 17.944 | 29.983 |

| Module | width=9 | | width=10 | | width=11 | | width=12 | |
|---|---|---|---|---|---|---|---|---|
| | $\overline{\Delta A}$ | $\max \Delta A$ | $\overline{\Delta A}$ | $\max \Delta A$ | $\overline{\Delta A}$ | $\max \Delta A$ | $\overline{\Delta A}$ | $\max \Delta A$ |
| Mul | 87.251 | 96.879 | 0.029 | 1.327 | 0.018 | 1.180 | 0.018 | 1.180 |
| Add | 72.169 | 86.213 | 138.389 | 149.422 | 138.394 | 149.422 | 193.458 | 211.845 |
| Accu | 29.163 | 43.008 | 47.767 | 57.262 | 47.722 | 57.262 | 75.943 | 95.502 |
| CST | 298.878 | 307.151 | 409.452 | 423.690 | 409.671 | 428.900 | 646.678 | 670.384 |
| EN | 3.830 | 13.369 | 6.566 | 15.385 | 6.505 | 15.385 | 13.375 | 35.144 |
| IOrg | 121.593 | 130.843 | 221.456 | 237.847 | 221.680 | 238.338 | 317.491 | 338.903 |
| PG | 172.858 | 184.566 | 353.269 | 370.704 | 353.245 | 370.704 | 498.202 | 507.544 |
| SA | 17.801 | 32.834 | 18.790 | 24.330 | 18.943 | 24.330 | 38.322 | 57.754 |

| Module | width=13 | | width=14 | | width=15 | | width=16 | |
|---|---|---|---|---|---|---|---|---|
| | $\overline{\Delta A}$ | $\max \Delta A$ | $\overline{\Delta A}$ | $\max \Delta A$ | $\overline{\Delta A}$ | $\max \Delta A$ | $\overline{\Delta A}$ | $\max \Delta A$ |
| Mul | 0.037 | 1.180 | 0.065 | 1.180 | 0.037 | 1.180 | 970.211 | 991.101 |
| Add | 193.239 | 211.845 | 162.950 | 186.434 | 162.111 | 186.532 | 131.416 | 156.107 |
| Accu | 76.349 | 95.502 | 108.857 | 143.770 | 108.976 | 141.165 | 152.858 | 170.459 |
| CST | 646.793 | 670.384 | 886.383 | 909.558 | 885.401 | 912.015 | 1208.204 | 1236.959 |
| EN | 13.344 | 35.144 | 21.520 | 50.184 | 21.808 | 50.282 | 40.915 | 58.884 |
| IOrg | 317.093 | 342.000 | 350.534 | 361.759 | 351.349 | 362.791 | 452.086 | 487.588 |
| PG | 498.367 | 507.544 | 557.728 | 569.967 | 558.635 | 574.194 | 652.765 | 679.723 |
| SA | 38.450 | 54.657 | 38.688 | 77.709 | 39.523 | 92.357 | 50.167 | 70.287 |

Table 4.2: Absolute and maximum area deltas for each bit mode across widths 1-16.

| Module | width=1 | | width=2 | | width=3 | | width=4 | |
|---|---|---|---|---|---|---|---|---|
| | $\overline{\Delta P}$ | $\max \Delta P$ | $\overline{\Delta P}$ | $\max \Delta P$ | $\overline{\Delta P}$ | $\max \Delta P$ | $\overline{\Delta P}$ | $\max \Delta P$ |
| Mul | 0.000 | 0.000 | 0.415 | 1.011 | 0.421 | 0.921 | 0.865 | 2.193 |
| Add | 0.751 | 1.744 | 0.565 | 1.220 | 0.557 | 1.137 | 0.178 | 0.738 |
| Accu | 0.335 | 1.181 | 0.496 | 0.879 | 0.495 | 0.747 | 0.865 | 1.400 |
| CST | 0.417 | 1.307 | 0.449 | 0.949 | 0.465 | 0.978 | 0.851 | 2.027 |
| EN | 0.293 | 1.132 | 0.090 | 0.503 | 0.086 | 0.442 | 0.843 | 1.345 |
| IOrg | 0.616 | 1.322 | 0.607 | 1.422 | 0.605 | 1.303 | 3.857 | 4.485 |
| PG | 0.701 | 1.610 | 1.144 | 1.449 | 1.151 | 1.498 | 5.355 | 5.998 |
| SA | 0.361 | 1.120 | 0.186 | 0.939 | 0.180 | 0.840 | 1.131 | 1.626 |

| Module | width=5 | | width=6 | | width=7 | | width=8 | |
|---|---|---|---|---|---|---|---|---|
| | $\overline{\Delta P}$ | $\max \Delta P$ | $\overline{\Delta P}$ | $\max \Delta P$ | $\overline{\Delta P}$ | $\max \Delta P$ | $\overline{\Delta P}$ | $\max \Delta P$ |
| Mul | 0.839 | 2.086 | 0.001 | 0.059 | 0.000 | 0.000 | 2.362 | 6.352 |
| Add | 0.186 | 0.662 | 0.716 | 1.246 | 0.720 | 1.249 | 2.065 | 2.691 |
| Accu | 0.843 | 1.389 | 1.035 | 1.471 | 1.037 | 1.469 | 0.875 | 1.917 |
| CST | 0.864 | 2.043 | 0.770 | 1.429 | 0.772 | 1.429 | 39.911 | 45.285 |
| EN | 0.818 | 1.209 | 1.029 | 1.466 | 1.025 | 1.462 | 1.415 | 1.990 |
| IOrg | 3.857 | 4.491 | 8.743 | 9.409 | 8.739 | 9.399 | 15.175 | 16.813 |
| PG | 5.320 | 5.963 | 14.094 | 15.135 | 14.097 | 15.127 | 23.352 | 25.455 |
| SA | 1.126 | 1.670 | 1.194 | 1.781 | 1.192 | 1.780 | 2.751 | 3.444 |

| Module | width=9 | | width=10 | | width=11 | | width=12 | |
|---|---|---|---|---|---|---|---|---|
| | $\overline{\Delta P}$ | $\max \Delta P$ | $\overline{\Delta P}$ | $\max \Delta P$ | $\overline{\Delta P}$ | $\max \Delta P$ | $\overline{\Delta P}$ | $\max \Delta P$ |
| Mul | 2.299 | 6.291 | 0.001 | 0.097 | 0.001 | 0.059 | 0.001 | 0.056 |
| Add | 2.096 | 2.743 | 4.056 | 4.519 | 4.050 | 4.517 | 6.449 | 7.041 |
| Accu | 0.947 | 2.060 | 0.659 | 1.303 | 0.657 | 1.302 | 0.519 | 1.689 |
| CST | 39.975 | 45.336 | 3.842 | 5.757 | 3.853 | 5.754 | 6.500 | 9.208 |
| EN | 1.398 | 1.963 | 1.617 | 2.086 | 1.617 | 2.091 | 1.929 | 2.512 |
| IOrg | 15.207 | 16.808 | 26.511 | 28.201 | 26.506 | 28.195 | 38.142 | 40.586 |
| PG | 23.419 | 25.666 | 45.462 | 47.692 | 45.456 | 47.696 | 63.936 | 66.783 |
| SA | 2.773 | 3.506 | 2.805 | 3.078 | 2.799 | 3.078 | 4.607 | 5.157 |

| Module | width=13 | | width=14 | | width=15 | | width=16 | |
|---|---|---|---|---|---|---|---|---|
| | $\overline{\Delta P}$ | $\max \Delta P$ | $\overline{\Delta P}$ | $\max \Delta P$ | $\overline{\Delta P}$ | $\max \Delta P$ | $\overline{\Delta P}$ | $\max \Delta P$ |
| Mul | 0.002 | 0.078 | 0.004 | 0.079 | 0.002 | 0.069 | 9.303 | 25.019 |
| Add | 6.464 | 7.052 | 10.173 | 11.660 | 10.140 | 11.621 | 13.167 | 30.217 |
| Accu | 0.536 | 1.679 | 0.692 | 1.984 | 0.661 | 1.920 | 1.420 | 3.027 |
| CST | 6.499 | 9.220 | 9.136 | 12.722 | 9.153 | 12.461 | 9.983 | 23.655 |
| EN | 1.939 | 2.525 | 2.057 | 3.558 | 2.022 | 2.754 | 1.765 | 2.783 |
| IOrg | 38.142 | 40.635 | 47.826 | 50.898 | 47.835 | 50.858 | 62.236 | 76.896 |
| PG | 63.947 | 66.787 | 82.663 | 86.351 | 82.592 | 86.235 | 105.242 | 121.082 |
| SA | 4.604 | 5.273 | 4.592 | 5.929 | 4.641 | 5.722 | 6.679 | 23.334 |

Table 4.3: Mean and maximum $\Delta$ for each bit-mode across widths 1-16.

and $\delta P_i$ the average area and power savings when module $i$ is switched to serial mode, then

$$A \approx A_0 - \sum_i \delta A_i \, \mathbf{1}_{\{i \text{ serial}\}}, \qquad P \approx P_0 - \sum_i \delta P_i \, \mathbf{1}_{\{i \text{ serial}\}}.$$

This linear superposition explains why hybrid configurations follow a nearly straight-line Pareto frontier in both area vs. latency and power vs. latency.

3. **Module savings summary.** Table 4.4 lists the mean and maximum area and power savings observed by flipping each module's mode bit at $w = 1$:

| **Module** | $\overline{\Delta A}$ (µm$^2$) | $\max \Delta A$ (µm$^2$) | $\overline{\Delta P}$ (mW) | $\max \Delta P$ (mW) |
|---|---|---|---|---|
| Multiplier (Mul) | 0.000 | 0.000 | 0.000 | 0.000 |
| Adder (Add) | 6.550 | 18.780 | 0.751 | 1.744 |
| Accumulator (Accu) | 7.310 | 19.660 | 0.335 | 1.181 |
| Concatenation-Shift Tree (CST) | 5.120 | 17.300 | 0.417 | 1.307 |
| Exponent Normalizer (EN) | 4.500 | 16.860 | 0.293 | 1.132 |
| Input Organizer (IOrg) | 5.290 | 15.970 | 0.616 | 1.322 |
| Primitive Generator (PG) | 7.840 | 17.740 | 0.701 | 1.610 |
| Sign Analyzer (SA) | 4.440 | 15.930 | 0.361 | 1.120 |

Table 4.4: Average and maximum area ($\Delta A$) and power ($\Delta P$) savings when switching each module to serial mode at $w = 1$.

These results confirm that, at $w = 1$, the multiplier is "free" to toggle without area or power impact, while every other block contributes predictable savings in both metrics. This clean, almost linear budget in area and energy simplifies the design of hybrid word-sliced configurations for arbitrary precision.

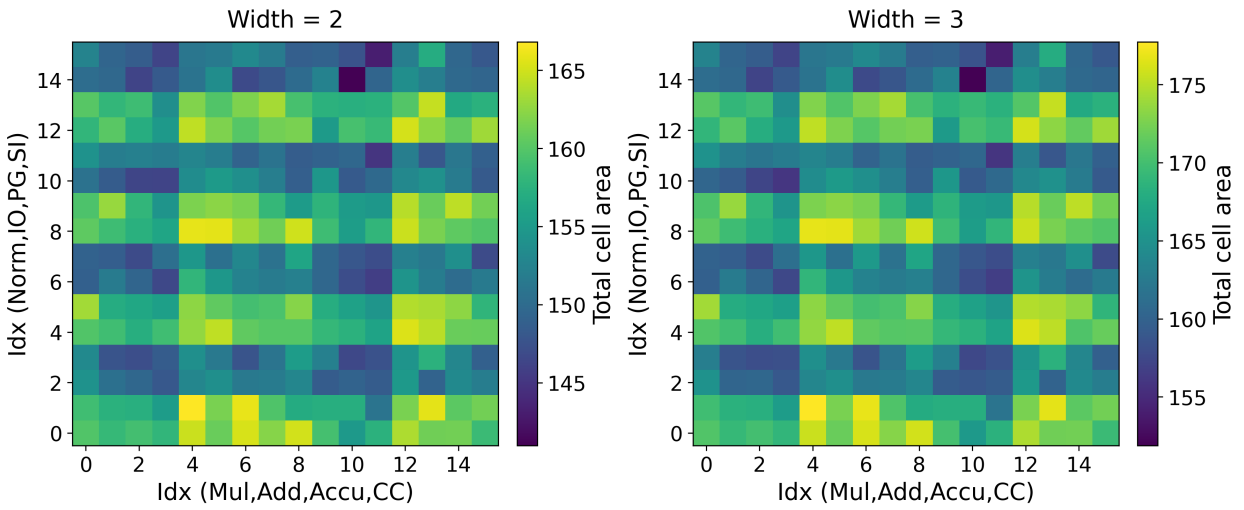



Figure 4.4: Comparison of area distributions across all mode combinations for $w = 2$ and $w = 3$. The two heatmaps share the same structural shape but are offset by a constant per-bit area overhead.

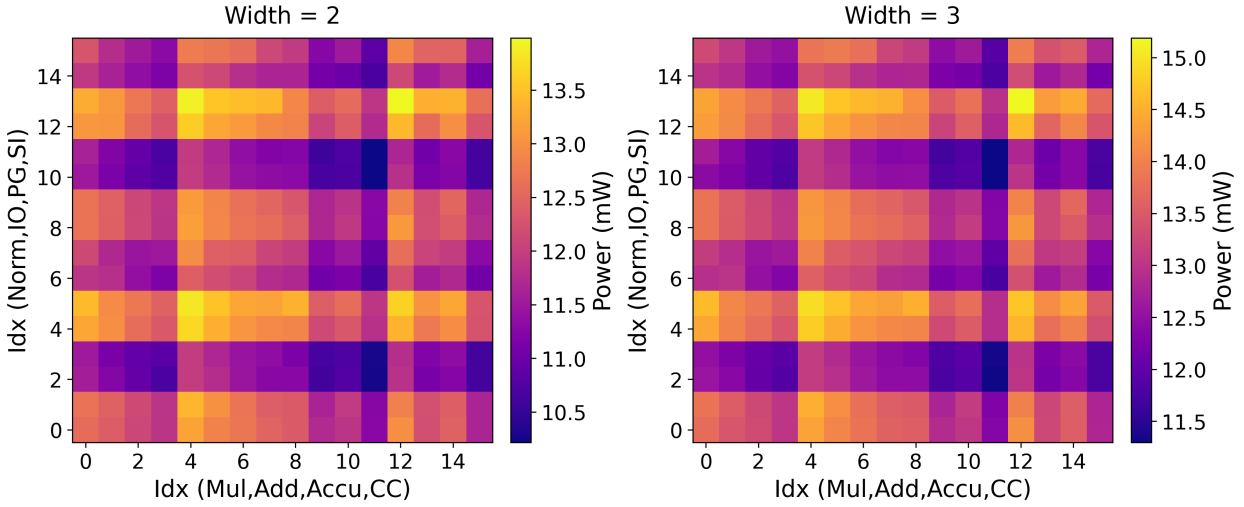



Figure 4.5: Comparison of power distributions across all mode combinations for $w = 2$ and $w = 3$. Like the area result, the two heatmaps share the same structural shape.

### 4.3.2 Area & Power Result Analysis when bit width=2, 3

Comparison of both area and power savings at $w = 2$ and $w = 3$ shows that, as with area, the power-differential heatmaps are structurally identical between the two widths and differ only by a uniform per-bit shift. The multiplier now contributes a small but consistent saving in both metrics ($\delta A_{\text{Mul}} \approx 2.39$, $\delta P_{\text{Mul}} \approx 0.42\,\text{mW}$), while the seven other blocks exhibit almost the same mean and maximum $\Delta A$ and $\Delta P$ at both widths (within a few percent). This confirms that each block $i$ has invariant per-bit area $\delta A_i$ and power $\delta P_i$ coupons, so the total area and power obey

$$A(w, \{b_i\}) \approx A_0(w) - \sum_i b_i\, \delta A_i, \quad P(w, \{b_i\}) \approx P_0(w) - \sum_i b_i\, \delta P_i.$$

| **Module** | $w = 2$ Area | | $w = 2$ Power | | $w = 3$ Area | | $w = 3$ Power | |
|---|---|---|---|---|---|---|---|---|
| | mean ΔA ($\mu m^2$) | max ΔA ($\mu m^2$) | mean ΔP (mW) | max ΔP (mW) | mean ΔA ($\mu m^2$) | max ΔA ($\mu m^2$) | mean ΔP (mW) | max ΔP (mW) |
| Mul | 2.389 | 7.127 | 0.415 | 1.011 | 2.377 | 7.127 | 0.421 | 0.921 |
| Add | 3.673 | 10.322 | 0.565 | 1.220 | 3.734 | 10.322 | 0.557 | 1.137 |
| Accu | 2.994 | 10.125 | 0.496 | 0.879 | 2.981 | 10.125 | 0.495 | 0.747 |
| CST | 2.452 | 8.454 | 0.449 | 0.949 | 2.474 | 8.454 | 0.465 | 0.978 |
| EN | 2.245 | 9.535 | 0.090 | 0.503 | 2.281 | 9.535 | 0.086 | 0.442 |
| IOrg | 2.160 | 8.946 | 0.607 | 1.422 | 2.194 | 8.946 | 0.605 | 1.303 |
| PG | 9.387 | 18.334 | 1.144 | 1.449 | 9.439 | 18.334 | 1.151 | 1.498 |
| SA | 2.463 | 9.732 | 0.186 | 0.939 | 2.479 | 9.732 | 0.180 | 0.840 |

Table 4.5: Mean and maximum area ($\Delta A$) and power ($\Delta P$) savings per module at $w = 2$ and $w = 3$.

The close correspondence between the area and power savings underscores the additive and separable nature of each module's contribution: toggling any block from parallel to serial yields nearly the same reduction in both metrics regardless of whether $w = 2$ or $w = 3$, with the only changing factor being the uniform baseline shift between the two precisions.

### 4.3.3 Area & Power Result Analysis when bit width=4 to 7

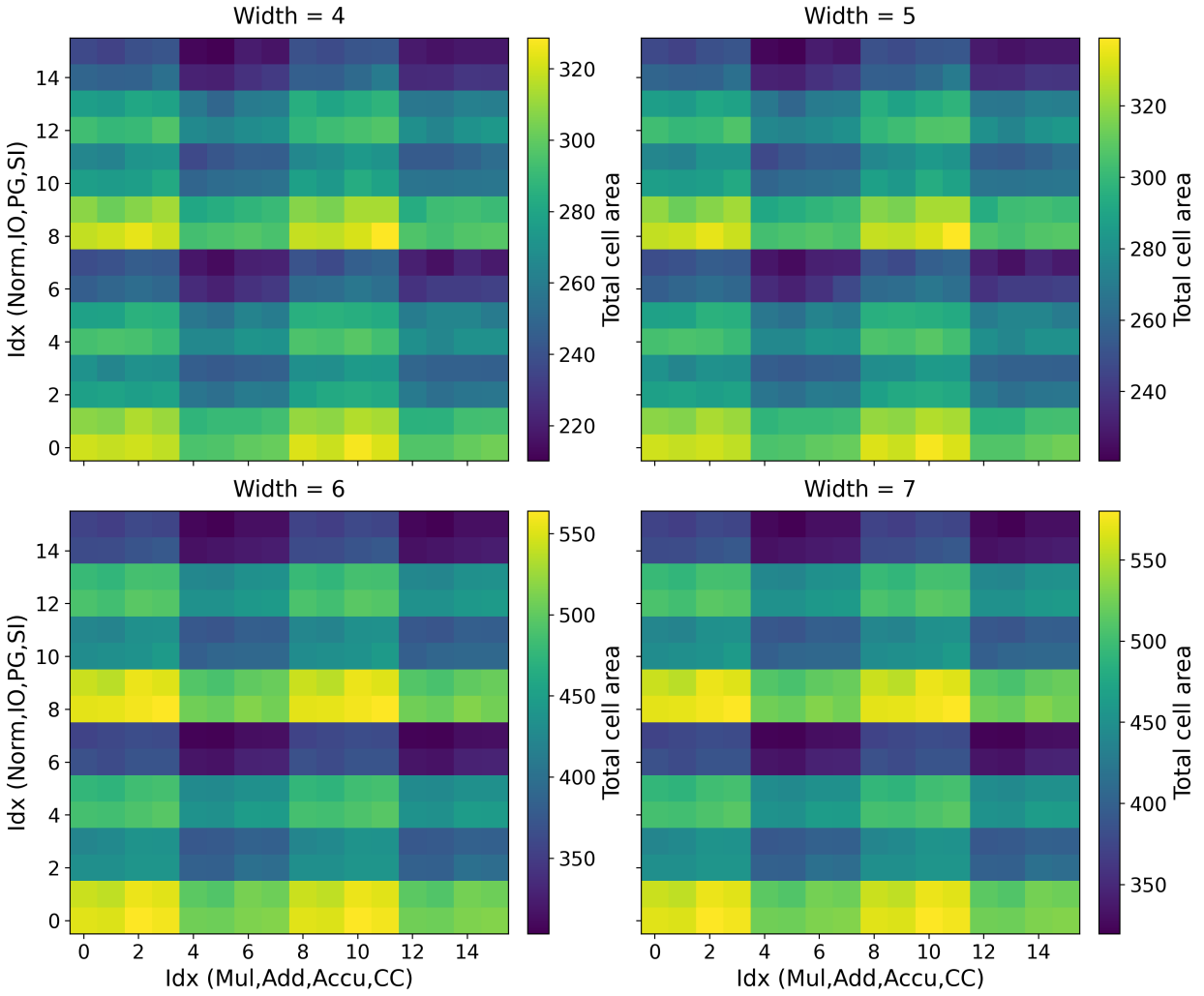



Figure 4.6: Comparison of area distributions across all mode combinations for $w = 4$ to $w = 7$. The heatmaps share the same structural shape.

As shown in Table 4.2 and Table 4.3, the area and power savings for each module at $w$ = 4-7 clearly split into two regimes. For $w = 4, 5$, the "heavy" modules—Primitive Generator and Input Organizer—offer moderate savings ($\delta A_{\mathrm{PG}} \approx 41.7\ \mu\mathrm{m}^2$, $\delta P_{\mathrm{PG}} \approx 5.35$ mW; $\delta A_{\mathrm{IOrg}} \approx 28.4\ \mu\mathrm{m}^2$, $\delta P_{\mathrm{IOrg}} \approx 3.86$ mW), while the multiplier still contributes a small nonzero benefit ($\delta A_{\mathrm{Mul}} \approx 3.1\ \mu\mathrm{m}^2$, $\delta P_{\mathrm{Mul}} \approx 0.85$ mW). At $w = 6, 7$, both area and power deltas for these same modules jump to roughly double their earlier values ($\delta A_{\mathrm{PG}} \approx 120\ \mu\mathrm{m}^2$, $\delta P_{\mathrm{PG}} \approx 14.1$ mW; $\delta A_{\mathrm{IOrg}} \approx 68\ \mu\mathrm{m}^2$, $\delta P_{\mathrm{IOrg}} \approx 8.74$ mW), whereas the multiplier's savings collapse to near

zero in both metrics. The smaller blocks (Adder, Accumulator, CST, EN, SA) exhibit only minor variation across these two plateaus.

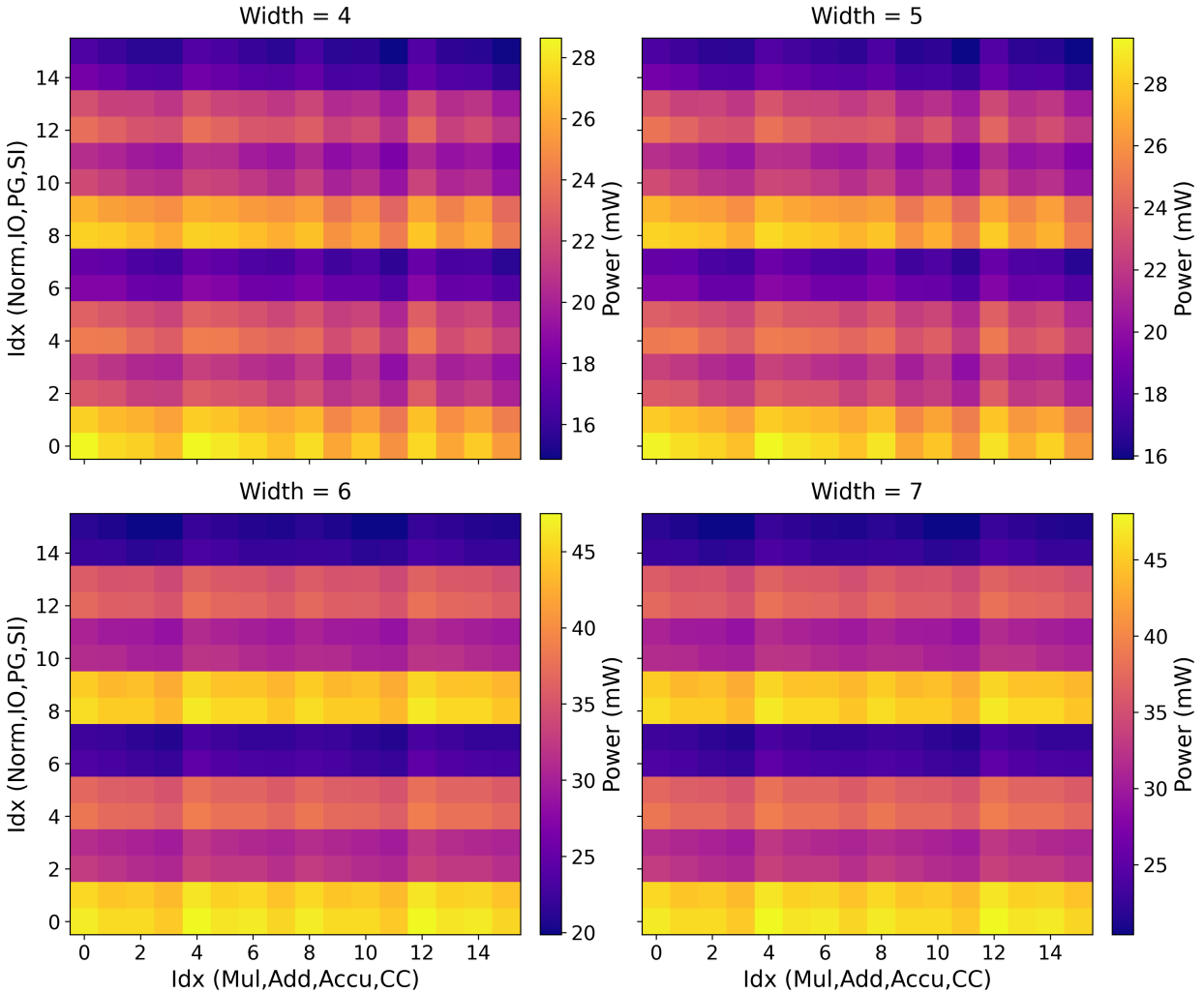



Figure 4.7: Comparison of area distributions across all mode combinations for $w = 4$ to $w = 7$. The heatmaps share the same structural shape.

Architecturally, these two discrete jumps at $w = 4$ and again at $w = 6$ correspond to the insertion of additional pipeline registers or the splitting of carry chains in the bit-parallel reduction trees to satisfy cycle-time constraints. Each register stage adds a nearly constant "coupon" of area and power cost in the parallel instantiation, so toggling to the serial version redeems exactly that coupon. Once the multiplier tree itself is pipelined (or mapped into DSP macros) by $w \geq 6$, its parallel and serial implementations are identical in depth and

resource usage, hence its $\delta A$ and $\delta P$ vanish.

### 4.3.4 Area & Power Result Analysis when bit width=8 to 15 & 16

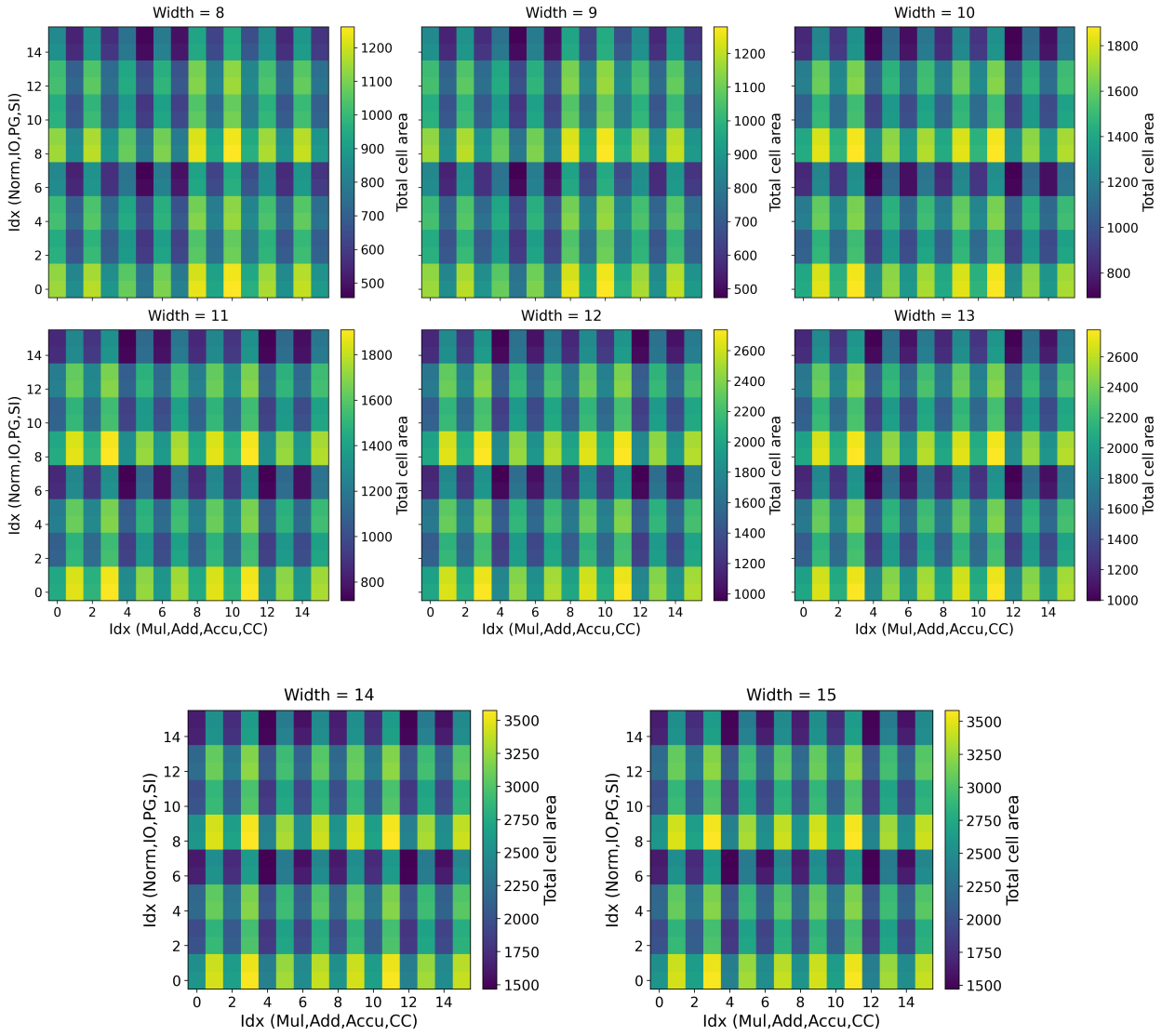



Figure 4.8: Comparison of area distributions across all mode combinations for wider precision settings. Top: combined $w = 8$-13. Bottom: individual maps for $w = 14$ and $w = 15$.

For the area and power results of widths $8 \leq w \leq 15$, both the area and power savings again break into four repeatable sub-bands—$\{8, 9\}, \{10, 11\}, \{12, 13\}, \{14, 15\}$—each corre-

sponding to a fixed reduction-tree depth of $\lceil \log_2 w \rceil = 3$. Within each pair the per-module coupons $\delta A_i$ and $\delta P_i$ remain essentially unchanged (see Table 4.2 and Table 4.3), producing heatmaps with identical shapes that are simply shifted in brightness by the uniform baseline cost of each new pipeline stage introduced at $w = 10$, $w = 12$, and $w = 14$. In Figure 4.8 and Figure 4.9, The "heavy" blocks—Concat-Shift Tree, Primitive Generator, and Input Organizer—continue to dominate both area and power savings, while the multiplier's toggle cost collapses to near zero once its FBRT network is fully pipelined at $w \geq 10$.

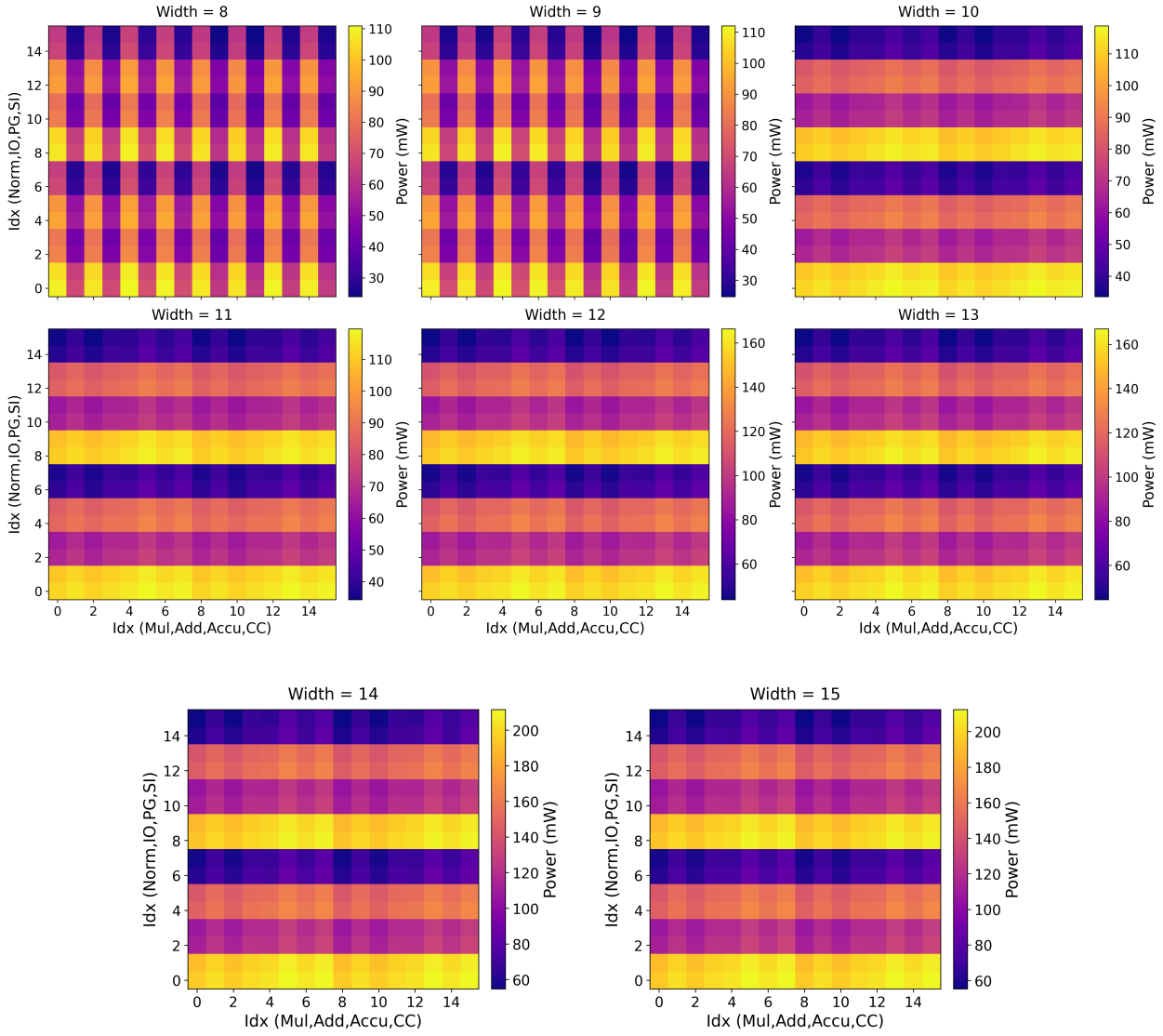
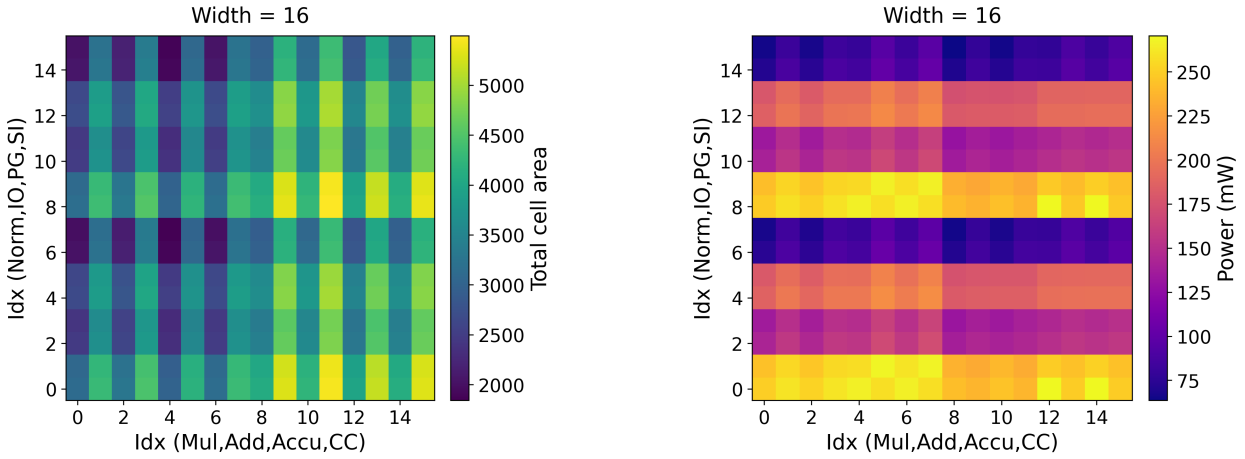



Figure 4.9: Comparison of Power distributions across all mode combinations for wider precision settings. Top: combined $w = 8$-13. Bottom: individual maps for $w = 14$ and $w = 15$.

At $w = 16$, a fifth register insertion (or carry-chain split) raises the reduction-tree depth to 4, triggering a pronounced jump in both metrics (see Figure 4.10a). Now every module's $\delta A_i$ and $\delta P_i$ roughly double from the previous plateau: the multiplier re-emerges with an area coupon of $\approx 970\,\mu\text{m}^2$ and a power coupon of $\approx 9.3\,\text{mW}$, while the Primitive Generator surges to $\approx 653\,\mu\text{m}^2$ and $\approx 105\,\text{mW}$, and the Concat-Shift Tree to $\approx 1208\,\mu\text{m}^2$ and $\approx 39.9\,\text{mW}$. This dramatic new threshold confirms that each extra bit of tree depth imposes a nearly constant "pipeline-stage cost" in both silicon and energy, reinforcing the linear superposition models

$$A(w) \approx A_0(w) - \sum_i b_i\,\delta A_i, \quad P(w) \approx P_0(w) - \sum_i b_i\,\delta P_i$$

for hybrid bit-parallel/serial configurations.




(a) Area result for $w = 16$.

(b) Power result for $w = 16$.

Figure 4.10: Comparison of area and power heatmaps at bit-width $w = 16$.

## 4.4 Analysis of Par/Ser Mode Impact on Area & Power Across Bit Widths

Our analysis begins by computing, for each bit-width $w$ and for each of the eight mode bits $\{Mul, Add, Accu, CST, EN, IOrg, PG, SA\}$, the change in total cell area (or dynamic power) resulting from flipping that bit alone. Concretely, for each configuration vector $\mathbf{b} \in \{0,1\}^8$ we record the "base" area (or power) $A(\mathbf{b})$, then flip one coordinate $i$ to obtain $\mathbf{b}'$ and record $\left|A(\mathbf{b}) - A(\mathbf{b}')\right|$. By aggregating these per-flip deltas we extract the minimum, mean, and maximum $\Delta$-area (or $\Delta$-power) for each mode bit at each $w$. In the resulting Figure 4.11 and Figure 4.12, the *savings points*—bit widths at which the $\Delta$-cost substantially *decreases*—highlight the transition from one parallel-block size to the next (namely at $w = 2, 4, 8, 16, \ldots$) for each module.

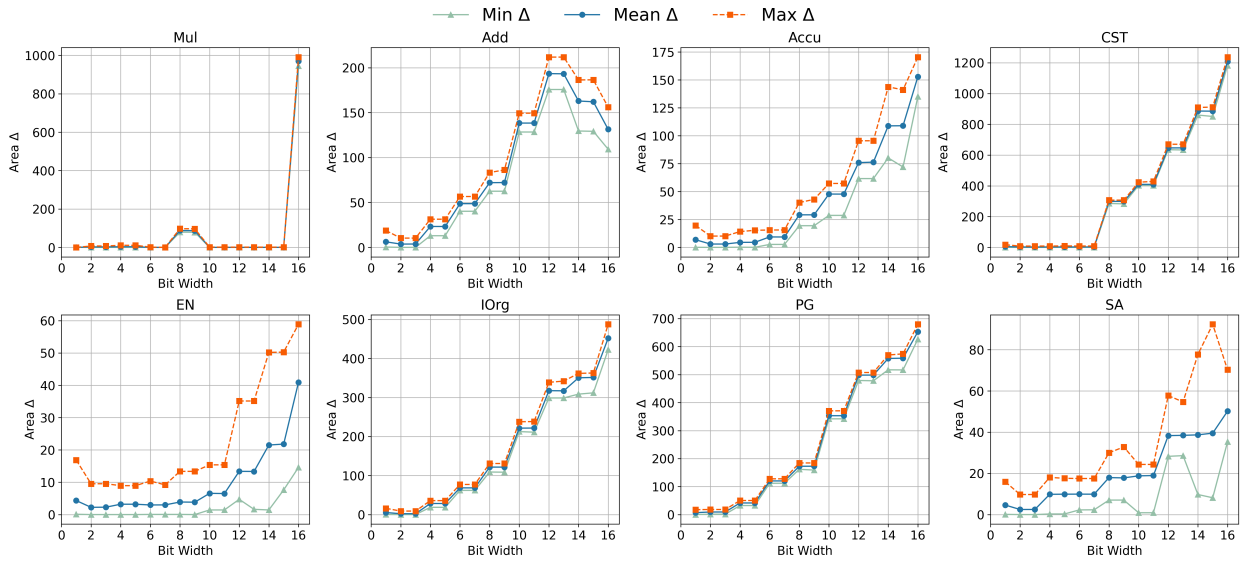



Figure 4.11: Min, mean, and max changes in cell area ($\Delta$-area) for each mode bit configuration as a function of data bit-width. Each subplot shows how flipping a single mode bit affects the design's total cell area—green triangles for the smallest observed $\Delta$-area, blue circles for the average $\Delta$-area, and orange squares for the largest $\Delta$-area—revealing constant behavior within each parallel-block band (e.g. widths 8-15) and jumps at the 2, 4, 8, and 16 boundary crossings.

**Deciphering the Area Curves.** In Figure 4.11, the area-$\Delta$ curves exhibit piecewise-constant behavior within each band of widths $[2^k, 2^{k+1} - 1]$. When $w$ crosses a power-of-two boundary, the bit-parallel tree depth increments by one, causing a sudden jump in per-bit area impact for modules whose flips still toggle the entire parallel block. For instance, in the **Adder** subplot, the mean and max $\Delta$ remain roughly constant for $w = 8 \ldots 13$, then *drop* at $w = 14, 15$ because the highest bits now live in the smaller serial remainder rather than the full 8-bit parallel adder. Similarly, the **Multiplier** shows modest area up to $w = 8$, a bump at $w = 9$ (two 8×8 partial-product engines), and a dramatic spike at $w = 16$ when a single 16×16 multiplier block is used. Modules like the **Accumulator** and **Sign Analyzer** grow only modestly beyond $w = 8$ because their design falls back entirely to the small serial path once past the first parallel chunk.

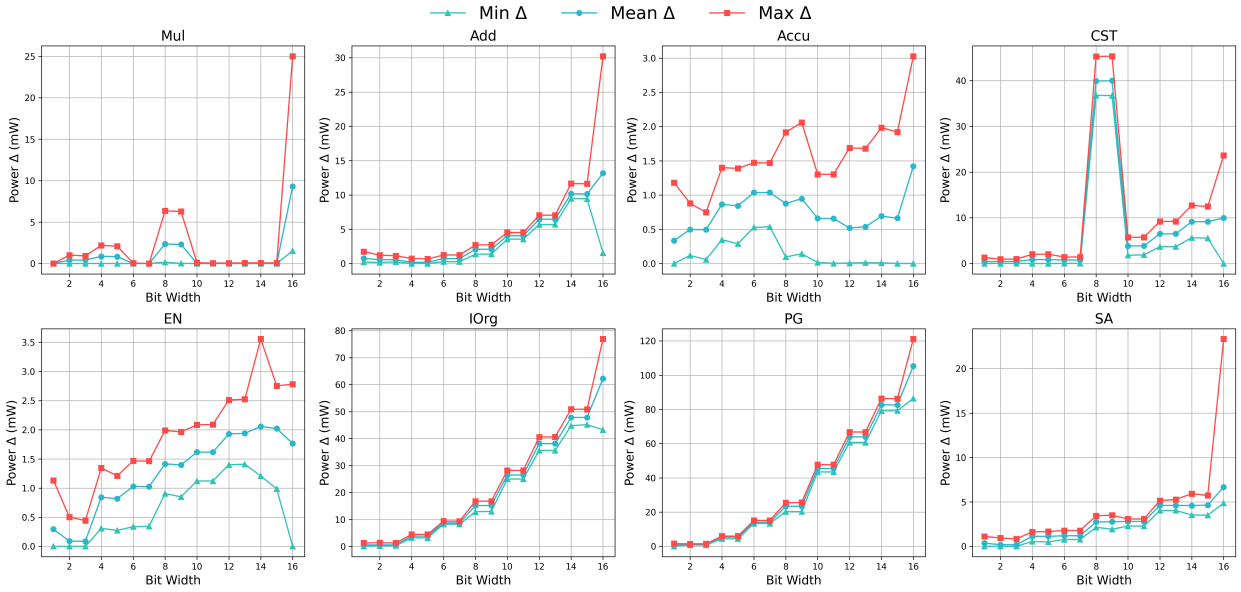



Figure 4.12: Min, mean, and max changes in dynamic power ($\Delta$-power in mW) for each mode bit configuration across bit-widths. Each of the eight subplots corresponds to one mode bit, with green triangles marking the minimum $\Delta$-power, blue circles the average, and red squares the maximum. Band-boundary effects at widths 2, 4, 8, and 16 produce pronounced spikes or drops, reflecting the transition between parallel- and serial-processing blocks in the hybrid architecture.

**Deciphering the Power Curves.** In Figure 4.12Power-$\Delta$ curves mirror the area-$\Delta$ patterns but emphasize switching activity. The **Adder's** minimum $\Delta$ collapses at $w = 16$, since there now exist bit-positions whose flips only toggle the tiny serial carry chain, drawing negligible dynamic power. The **Accumulator** minimum remains flat for $w \geq 10$, as each additional bit exclusively uses the unchanging serial adder and register. The **Concat-Shift Tree** spikes at $w = 9$ because some flips begin to cascade through two 8-bit barrel shifters. And the **Exponent Normalizer** minimum dips whenever the highest bits shift out of the current parallel band into a purely serial path (notably at $w = 9$ and $w = 16$).

**Summary of Key Savings Points.** Across both area and power analyses, the *savings points* all align with the $\{2, 4, 8, 16, \dots\}$ band boundaries where the parallel-block depth increments:

- **Adder** (*add*): cost drops at $w = 14, 15$, since high bits exit the 8-bit parallel tree.
- **Multiplier** (*mul*): bump at $w = 9$ (two 8×8 engines) and large spike at $w = 16$ (single 16×16 engine).
- **Accumulator** (*accu*) & **Sign Analyzer** (*si*): grow slowly past $w = 8$; no further parallel savings beyond the first band.
- **Normalizer** (*norm*) & **Primitive Generator** (*pg*): savings when $w$ crosses 8 and 16.
- **Input Organizer** (*io*) & **Carry-Select Tree** (*cc*): similar band-boundary drops at $w = 9$ and $w = 16$.

Modules whose $\Delta$ grows nearly linearly (Accumulator, Sign Analyzer) should be sized in the remainder band (e.g. $w = 9 \dots 15$) to avoid further parallel-block growth and maximize area/power savings.

## 4.5 Latency Results Analysis

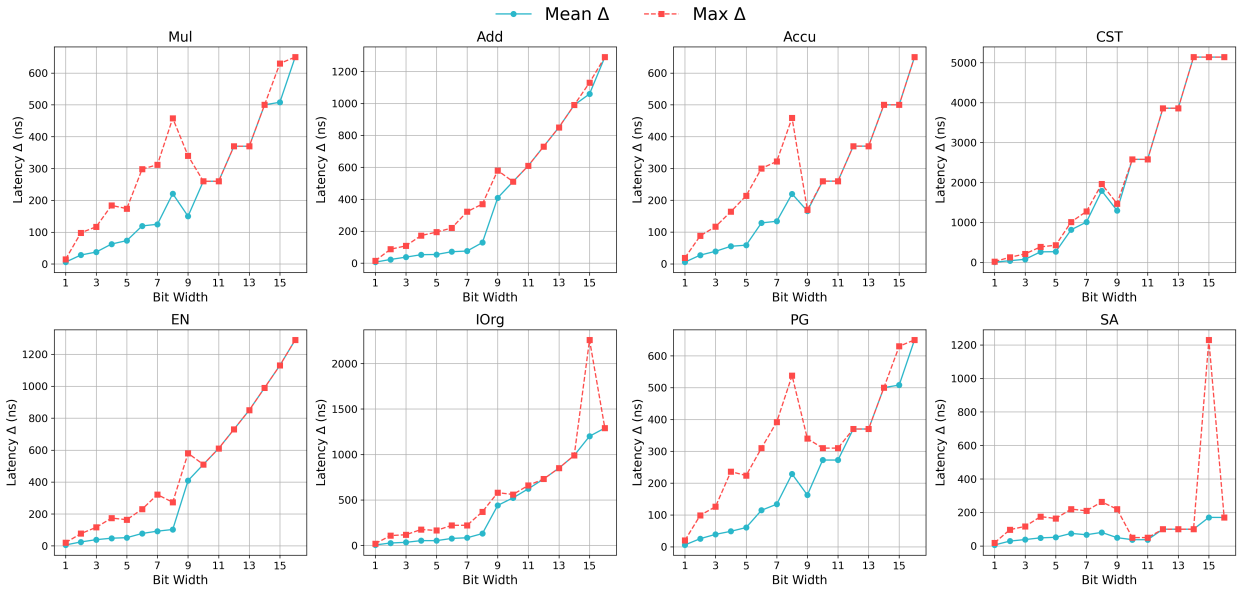



Figure 4.13: Mean, and max changes in

**Expected Behavior of Sub-module Parallelism vs. Serialism** When a sub-module is configured in fully parallel mode (i.e. `_PARALLEL=1`), all of its bit-level operations (for example, primitive multiplications or concatenation shifts) execute concurrently, yielding a nearly constant one-cycle latency regardless of bit-width. In contrast, in serial mode (`_PARALLEL=0`) each primitive element is processed sequentially, so its latency grows linearly with the number of primitives (and since the number of primitives often scales as $\propto w^2$ for an $w$-bit mantissa, the overall delay can rise very steeply as $w$ increases).

**Latency Sweep Methodology and Observations** We built a parameterized SystemVerilog testbench that instantiates `PE_Hybrid` with varying `REGISTER_WIDTH` and sweeps all combinations of the eight sub-module parallel flags. A Python driver auto-generates a small wrapper for each configuration, compiles and simulates it under VCS, and logs the cycle count from the `read_data` pulse to `done_r`. Figure 4.13 plots both the mean and maximum

latency differences between serial and parallel operation for each module versus bit width. As expected, modules with large primitive counts—such as *MantissaMultDual* (Mul) and the concatenation tree (CST)—show almost flat curves in parallel mode but accelerate rapidly once run serially beyond $w \approx 8$. Modules with fewer primitives (e.g. Add, Accu) exhibit more gradual growth, while truly parallel configurations remain constant across all widths.

**Summary and Low-Precision Highlight** Overall, the full-PE latency is dominated by any remaining serial stages, whose loop counts scale with bit width, whereas fully parallel stages impose negligible delay. Crucially, at very low precision ($w = 1$) the latency gap between serial and parallel is minimal—often only a handful of cycles—because the number of primitives is itself small. This implies that supporting ultra-low precision (e.g. binary operation) incurs almost no performance penalty even in serial mode, making it especially attractive for highly quantized inference tasks where hardware simplicity and minimal area are paramount.

## 4.6 Conclusion of Area, Power and Latency Analysis

In our notation, for each dual-mode block $i$ we define

$$\delta_i(w) \;=\; \big[\text{metric}_i\big]_{\text{par}}(w) \;-\; \big[\text{metric}_i\big]_{\text{ser}}(w),$$

and we report both the mean $\overline{\delta_i}$ and the maximum $\max \delta_i$ over the configuration space at each bit-width $w$. The difference between $\max \delta_i$ and $\overline{\delta_i}$ thus corresponds exactly to the largest possible saving achievable by toggling block $i$ from parallel to serial mode (i.e. the "best-case" configuration benefit).

Our area-distribution analysis reveals that the PE's eight dual-mode blocks give rise to four

distinct precision bands—$w = 1$, $2-3$, $4-7$, and $8-15$—which correspond exactly to the tree depths $\lceil \log_2 w \rceil = 0, 1, 2, 3$ in the bit-parallel instantiations. Within each band, the relative pattern of area savings when toggling any block from parallel to serial mode remains invariant, since each block contributes a fixed per-bit area difference $\delta_i$. As a result, the heatmaps for any two widths in the same band are pixel-identical up to a uniform vertical shift.

At the boundaries $w = 2, 4, 8$, synthesis must insert additional pipeline registers (or split carry chains) to meet cycle-time constraints, producing discrete jumps in the $\delta_i$ curves. Beyond $w \geq 8$, the FBRT multiplier itself is fully pipelined (or mapped into DSP macros), so its parallel-vs-serial area difference vanishes. Meanwhile, the overall area and the spread across configurations grow sharply with $w$, yet remain separable and additive: each module's toggle produces a vertical or horizontal stripe in the heatmap, with no significant cross-term interactions.

These findings have clear design implications. For high-precision kernels ($w \geq 8$), the largest slopes—found in the Multiplier, Concat-Shift Tree, and Input Organizer—should be operated in serial mode to minimize area, while lighter blocks can remain parallel. Conversely, for ultra low-latency, low-bitwidth operations ($w \leq 4$), an all-parallel configuration yields modest area overhead, enabling maximal throughput without a prohibitive footprint.

We performed an analogous sweep of end-to-end PE latency by toggling each block's `_PARALLEL` flag across all $2^8$ combinations and bit widths, logging the cycle count from the `read_data` pulse to the PE's `done_r` signal. Figure 4.13 plots the mean and maximum $\delta_i$ (parallel minus serial) versus $w$ for each block. As expected, fully parallel blocks exhibit nearly constant latency, while serial blocks incur a per-primitive loop cost that grows roughly as $\mathcal{O}(w^2)$. Importantly, at very low precision ($w = 1$), the number of primitives—and hence the serial loop length—is minimal, so the latency difference between parallel and serial modes is also minimal, often only a few cycles. This suggests that binary or ternary inference can tolerate

serial implementations without significant performance penalty, making them attractive for area- and power-constrained designs.

**Summary.** By quantifying $\delta_i(w)$ across both area and latency metrics, we identify exactly which blocks deliver the greatest "bang for buck" when switched to serial mode, and under what precision regimes. For $w \geq 8$, serializing the heavy arithmetic units yields the largest area savings with manageable latency overhead, whereas for $w \leq 4$, full parallelism is both area-efficient and latency-optimal. Thus, our multi-dimensional analysis offers a clear, quantitative guide to configuring the PE for any target precision and performance envelope.

## 4.7 EDP & Area Result Analysis

### 4.7.1 Analysis of Hybrid PE Metrics

As established in the original FlexiBit evaluation, FlexiBit's fully flexible PE achieves substantially better EDP than both bit-serial and in-memory designs—4.3× lower EDP than Cambricon-P and 2.5× lower EDP than DRISA in the Cloud-B configuration [19]. To understand how our hybrid PE modes trade off area, latency, and energy across operand widths, we normalize each hybrid-PE configuration's EDP, delay, power, and area against the baseline FlexiBit PE at the same width.

Figure 4.14 reveals a clear "turning point" at $w \approx 8$ bits. For low precisions ($w < 8$):

- **EDP uniformly improves.** Even the worst-case (max) hybrid EDP is below the baseline FlexiBit EDP for all $w = 2 \ldots 5$. The average EDP reduction exceeds 50% in this regime, demonstrating that bit-serial execution of the most area-hungry submodules consistently yields energy–delay benefits.

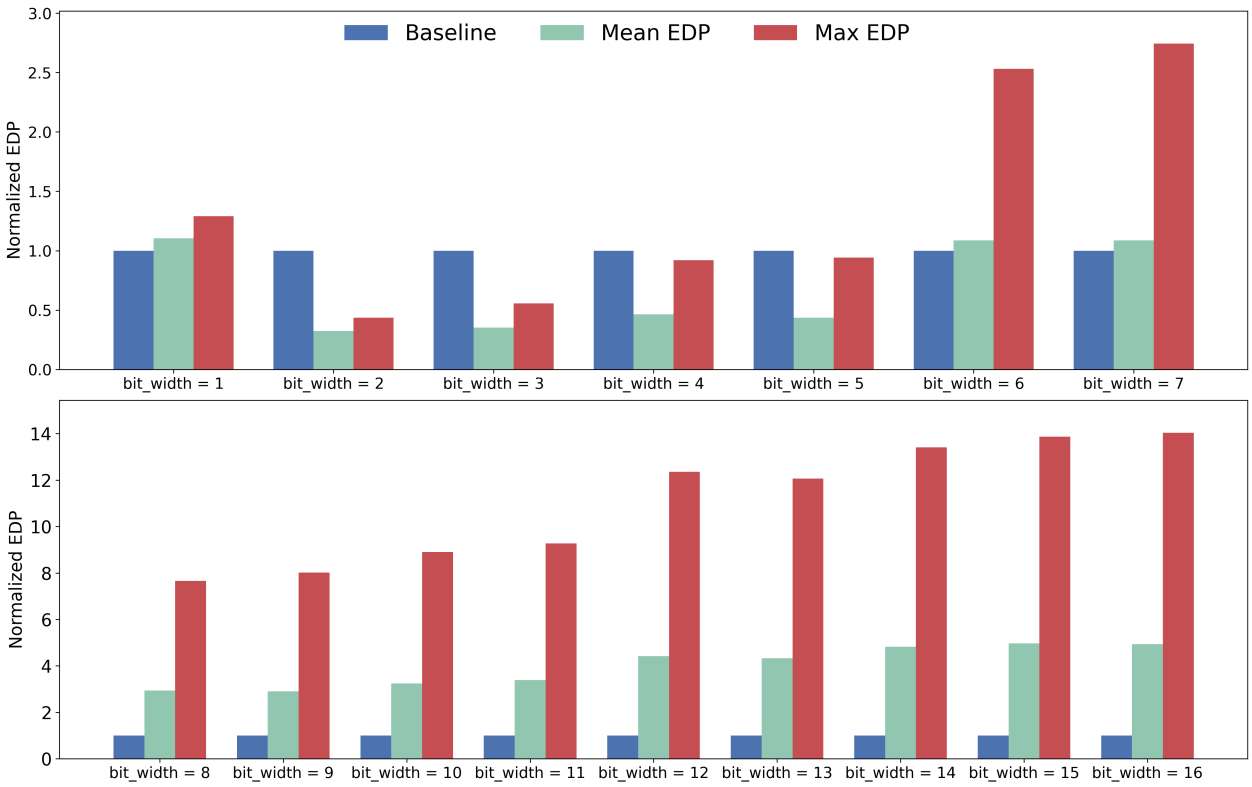



Figure 4.14: Hybrid PE EDP normalized to the baseline FlexiBit PE at each operand width. Shaded bars show the full min–max range across all hybrid configurations at each $w$.

- **Negligible overhead at 1 bit.** At $w = 1$, the maximum hybrid EDP is within 5% of the baseline (normalized max $\approx 1.05$) and the minimum falls below baseline ($\approx 0.93$). This near-zero penalty confirms that bit-serial logic incurs minimal delay or energy overhead when true bit-parallelism is already trivial.

- **Area savings align with EDP gains.** As shown in Figure 4.15, hybrid configurations at $w < 8$ reduce cell area by up to 30% versus the baseline, directly contributing to the EDP improvements.

Beyond $w \geq 8$, latency overheads from serializing the largest modules begin to outweigh power reductions for some configurations, causing the max-EDP bar to climb above unity. However, the *best-case* hybrid still outperforms the baseline up to $w \approx 12$, suggesting selective, compiler-guided mode selection can extend benefits into mid-precision kernels.

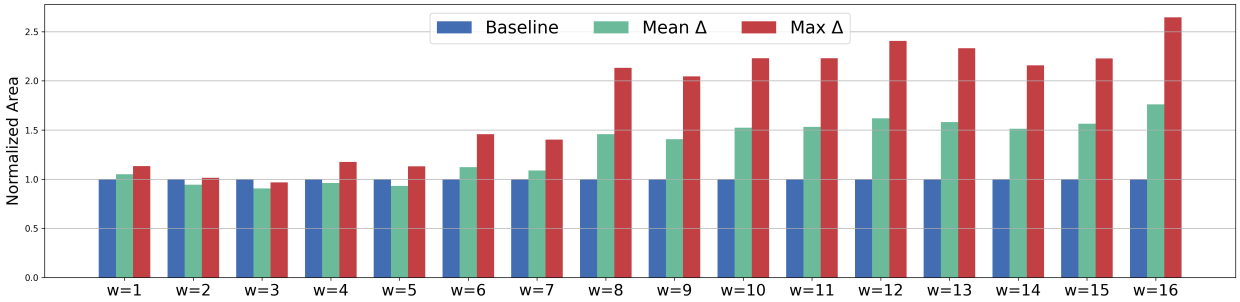



Figure 4.15: Hybrid PE total cell area normalized to the baseline FlexiBit PE at each operand width.

### 4.7.2 Summary

Our hybrid bit-parallel/bit-serial PE design captures the flexibility of FlexiBit while unlocking further EDP and area savings at low precisions. The key findings are:

- **Optimal regime is $w < 8$.** In this range, *all* hybrid configurations improve upon the baseline EDP, with no trade-offs needed—enabling plug-and-play low-precision acceleration.
- **Trivial overhead at $w = 1$.** Operating entirely in bit-serial mode for 1-bit data incurs almost no penalty, making the hybrid PE ideal for binary-neural and other ultra-low-precision workloads.
- **Mode-selective extension to mid-precision.** For $8 \leq w \leq 12$, careful selection of which modules to serialize sustains EDP gains while controlling latency impact.

Taken together, these results validate the hybrid architecture as a practical means to seamlessly adapt to the full spectrum of precisions encountered in modern AI inference, delivering best-of-both-worlds performance, energy efficiency, and area utilization.

# Chapter 5

# Conclusion and Future Work

This thesis presented a comprehensive design-space exploration of a hybrid bit-parallel and bit-serial processing architecture aimed at improving performance, area efficiency, and energy-delay product (EDP) for artificial intelligence accelerators. Through systematic experimental analyses conducted across multiple operand precisions (1 to 16 bits), we rigorously evaluated the trade-offs in area, power, latency, and EDP for various hybrid configurations.

Our detailed evaluation identified critical precision thresholds and architectural turning points, specifically highlighting significant benefits at low operand precisions. Notably, at very low precisions (less than 8 bits), hybrid architectures consistently outperformed fully bit-parallel designs in both area and energy efficiency, with negligible latency penalties. Particularly striking is the negligible overhead observed at the lowest precision setting (1-bit), where hybrid designs exhibited nearly identical performance and superior area-power efficiency compared to fully parallel counterparts. This result demonstrates the compelling potential for hybrid designs in ultra-low-precision inference scenarios, such as binary and ternary neural network computations.

In addition, the exploration of module-level parallel-serial toggling revealed that individ-

ual modules exhibit predictable, additive contributions to overall area, power, and latency. Modules such as the Multiplier, Concatenation-Shift Tree (CST), and Primitive Generator (PG) dominate resource savings when operated serially at mid to high precisions, whereas at lower precisions, these benefits become universally favorable. This insight allows strategic, compiler-guided selection of module operation modes to tailor architectures optimally for specific precision requirements.

Moreover, the EDP analysis reinforced these findings, establishing clear operational regimes. At operand widths below 8 bits, hybrid architectures demonstrated uniformly improved EDP, with savings surpassing 50% compared to fully bit-parallel designs. Conversely, at precisions beyond this point, latency penalties from serialization gradually offset power savings, suggesting targeted serialization as a practical approach for mid-precision ranges.

This thesis opens several promising avenues for future exploration:

- **Dynamic Runtime Adaptation**: Extending the static compile-time configuration strategy presented herein to dynamic runtime adaptation, enabling real-time precision adjustments based on workload characteristics and energy budgets.
- **Expanded Precision Support**: Investigating hybrid architectures tailored for emerging numeric formats, including mixed floating-point and integer formats beyond FP6 and FP5, and assessing their performance and efficiency in practical AI inference scenarios.
- **Integrated System-Level Evaluation**: Conducting holistic system-level evaluations incorporating memory hierarchy interactions, dataflow optimization, and communication overhead to fully assess the benefits and costs associated with hybrid processing elements in large-scale accelerator systems.
- **Machine Learning-Driven Configuration Optimization**: Employing machine

learning and reinforcement-learning methods to automate and refine the selection of optimal bit-parallel/bit-serial configurations across various precisions, thereby enhancing compiler and design-space exploration efficiency.

- **Fabrication and Hardware Validation**: Pursuing practical hardware implementations, including fabrication and empirical validation, to verify simulation-derived insights and refine the design under real-world operational conditions.

In conclusion, this thesis demonstrates that hybrid bit-parallel and bit-serial architectures provide an effective and scalable solution to address the diverse precision and efficiency requirements inherent in modern AI workloads. By strategically exploiting both architectural paradigms, future designs can achieve unparalleled flexibility, performance, and energy efficiency, positioning them as vital building blocks for next-generation AI accelerators.

# Bibliography


[1] Ieee standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84, 2019.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[3] N. Burgess, J. Milanovic, N. Stephens, K. Monachopoulos, and D. Mansell. Bfloat16 processing for neural networks. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, pages 88–91, 2019.

[4] Y. Chen, A. F. AbouElhamayed, X. Dai, Y. Wang, M. Andronic, G. A. Constantinides, and M. S. Abdelfattah. Bitmod: Bit-serial mixture-of-datatype llm acceleration. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 1082–1097, 2025.

[5] 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*, 52(1):127–138, 2017.

[6] Z. Dong, Z. Yao, A. Gholami, M. Mahoney, and K. Keutzer. HAWQ: Hessian AWare Quantization of Neural Networks With Mixed-Precision . In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 293–302, Los Alamitos, CA, USA, Nov. 2019. IEEE Computer Society.

[7] Y. Hao, Y. Zhao, C. Liu, Z. Du, S. Cheng, X. Li, X. Hu, Q. Guo, Z. Xu, and T. Chen. Cambricon-p: A bitflow architecture for arbitrary precision computing. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 57–72, 2022.

[8] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, T. Chen, G. Hu, S. Shi, and X. Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes, 2018.

[9] C. Jo and K. Lee. Bit-serial multiplier based neural processing element with approximate adder tree. In *2020 International SoC Design Conference (ISOCC)*, pages 286–287, 2020.

[10] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, and A. Moshovos. Stripes: Bit-serial deep neural network computing. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, 2016.

[11] H. Kwon, A. Samajdar, and T. Krishna. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 461–475, 2018.

[12] S. Li and P. Gupta. Bit-serial weight pools: Compression and arbitrary precision execution of neural networks on resource constrained processors. In D. Marculescu, Y. Chi, and C. Wu, editors, *Proceedings of Machine Learning and Systems*, volume 4, pages 238–250, 2022.

[13] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie. Drisa: A dram-based reconfigurable in-situ accelerator. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 288–301, 2017.

[14] S.-y. Liu, Z. Liu, X. Huang, P. Dong, and K.-T. Cheng. LLM-FP4: 4-bit floating-point quantized transformers. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 592–605, Singapore, Dec. 2023. Association for Computational Linguistics.

[15] S. Markidis, S. W. D. Chien, E. Laure, I. B. Peng, and J. S. Vetter. Nvidia tensor core programmability, performance amp; precision. In *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 522–531. IEEE, May 2018.

[16] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu. Mixed precision training, 2018.

[17] NVIDIA. Nvidia tensor cores: High-performance matrix operations for deep learning. Technical report, NVIDIA, 2017.

[18] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 304–315, 2019.

[19] F. Tahmasebi, Y. Wang, B. Y. H. Huang, and H. Kwon. Flexibit: Fully flexible precision bit-parallel accelerator architecture for arbitrary mixed precision ai, 2024.

[20] S. e. Venkataramani. Rapid: Ai accelerator for ultra-low precision training and inference. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 153–166, 2021.

[21] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. HAQ: Hardware-Aware Automated Quantization With Mixed Precision . In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8604–8612, Los Alamitos, CA, USA, June 2019. IEEE Computer Society.

[22] Z. Yang, J. Zhuang, J. Yin, C. Yu, A. K. Jones, and P. Zhou. Aim: Accelerating arbitrary-precision integer multiplication on heterogeneous reconfigurable computing platform versal acap. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9, 2023.