Low-Latency Resampling Architectures for Particle Filters on FPGA

XINGYU LIU
MASTER´S THESIS
DEPARTMENT OF ELECTRICAL AND INFORMATION TECHNOLOGY
FACULTY OF ENGINEERING | LTH | LUND UNIVERSITY

Low-Latency Resampling Architectures for Particle Filters on FPGA

Xingyu Liu

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University

Supervisor: Dumitra Iancu and Per Andersson

Examiner: Pietro Andreani

June 1, 2026

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

As wireless communication systems evolve toward 6G and joint communication and sensing paradigms, emerging applications impose stringent low-latency constraints on target tracking systems. Within the hardware implementation of particle filters, the traditional systematic resampling algorithm suffers from global data dependencies and sequential memory accesses, creating a fundamental performance bottleneck.

To address this latency issue, this report designs, implements, and evaluates two distinct resampling architectures on an FPGA platform.

The first design is an optimized sequential architecture that utilizes pre-fetching logic to effectively mask the inherent read latency of the on-chip memory.

To completely break the sequential bottleneck, the second design introduces a parallel architecture based on the Metropolis-Hastings algorithm. By employing an array of independent processing elements and a decentralized memory topology, combining private working memories with dual-port shared read-only memories, this architecture eliminates global data dependency and avoids the routing congestion and stall penalties associated with traditional crossbar switches.

Hardware results at a 100 MHz clock frequency demonstrate that the optimized sequential architecture achieves an execution time of 81.98 μs. Furthermore, the parallel architecture utilizing 16 PEs when dynamically downscaled to 4 iterations reduces the execution latency to 10.49 μs delivering a 15.6 times speedup compared to the standard baseline.

While the parallel approach consumes more Block RAM and digital signal processing resources and exhibits a bounded statistical degradation, this trade-off provides a mathematically sound and scalable hardware solution to meet the real time demands of next generation applications.

Acknowledgments

This Master thesis would not exist without the help of my supervisors. I want to express my deep thanks to Dumitra Iancu and Per Andersson. They gave me a lot of patient guidance during the whole project. Their strict attitude towards research and full support for my exploration helped me finish this work. Finally, a big thank you to my family and friends for standing by my side all the time.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Project Background

With the continuous evolution of wireless communication paradigms toward 6G, systems are increasingly gravitating toward Joint Communication and Sensing (JCAS) [1]. Distributed MIMO (D-MIMO) systems leverage enhanced spatial resolution provided by spatially separated access points to achieve high precision positioning. In these non-linear and non-Gaussian environments, Particle Filters (PFs) are extensively employed to track mobile targets, as they offer superior estimation accuracy compared to traditional linear filters [2]. However, emerging 6G use cases, such as collaborative robotics and Extended Reality (XR), impose low latency constraints, typically requiring a response time within 1 to 5 milliseconds [3]. Within the recursive execution of the particle filtering algorithm, the resampling step has consistently been identified as the primary performance bottleneck [1, 4]. This is primarily because traditional resampling algorithms inherently rely on global data dependencies, which restricts their scalability and parallelization in hardware [4].

1.2 Problem Statement and Motivation

The fundamental purpose of resampling is to mitigate particle degeneracy, a phenomenon where the filter weight concentrates on a negligible subset of particles [2, 5]. Most conventional algorithms, such as systematic resampling (SR), rely on the computation of a cumulative sum of normalized weights. This approach introduces a global data dependency, as every step in the accumulation must await the result of the preceding calculation, thereby severely limiting the system's ability to execute in a truly parallel or streaming fashion [4, 6]. Furthermore, it is worth mentioning that the on-demand memory accesses required for SR inevitably consume non-negligible clock cycles, which inherently introduces additional processing latency into the overall hardware system [1]. In hardware implementations such as FPGAs, these issues show as significant pipeline stalls, which drastically increase the execution time of the resampling stage. Research has shown that the latency incurred during resampling, defined as the cycles where no output is produced, is a critical barrier to meeting the sub-5ms requirements of 6G applications [1, 4] .

Therefore, there is a strong practical motivation to explore alternative hardware architectures capable of minimizing these delays.

1.3 Thesis Objectives

This thesis aims to design, implement, and evaluate two distinct hardware architectures for particle filter resampling on an FPGA platform to address the latency issue. The first is an Optimized Sequential Architecture which introduces advanced pre-fetching logic with synchronous FIFOs. By fetching weight values from memory in parallel with the arithmetic execution, this architecture aims to mask memory read latencies and reduce the overall cycle count. The second is a Parallel Architecture utilizing the Metropolis-Hastings (MH) algorithm. Unlike traditional methods, the MH approach eliminates the need for global collective operations, thereby enabling a highly parallelized array of processing elements (PEs) [7] . A core objective of this research is the quantitative assessment of the tradeoff between hardware resource utilization (such as LUTs, DSPs, and BRAMs) and processing latency. By comparing these approaches, this study provides a reference for selecting hardware configurations under specific 6G system constraints.

1.4 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 provides the theoretical framework for particle resampling and the mathematical basis for various resampling algorithms. Chapter 3 details the proposed hardware designs, specifically elaborating on the implementation of pre-fetching logic for the sequential model and the pipelined architecture of the MH PE array. Also this chapter includes a theoretical analysis about adding a crossbar to reuse the memory and explains why we use the current shared replicated BRAM design to avoid the timing and stall problems. Chapter 4 describes the experimental environment and the specific FPGA deployment process. Chapter 5 presents the hardware results, offering a rigorous analysis of the area, latency and accuracy tradeoffs of each architecture. Finally, Chapter 6 concludes the thesis with a summary of key findings and outlines potential directions for future work.

Chapter 2

Theoretical Framework and Mathematical Basis for Resampling

2.1 Functional Role of Resampling

The resampling stage presents the greatest difficulty when implementing particle filters in hardware [4]. The primary objective of resampling is to resolve the inevitable issue of weight degeneracy. As the filter progresses through successive time steps, the variance of the particle weights naturally increases. Eventually, almost all the weight concentrates on part of particles, leaving the vast majority with near zero weights [2]. If left unaddressed, the hardware wastes significant computational resources updating and propagating particles that represent highly improbable states. Resampling prevents this inefficiency by reconstructing the particle population based on their assigned weights. It actively discards the particles with negligible weights and replicates the ones with larger weights [7]. Consequently, the resampling successfully concentrates the available particles into domains of higher posterior probability [7]. This mechanism ensures that the limited hardware resources are focused on tracking the most likely states, maintaining the overall stability and accuracy of the filter.

2.2 Systematic Resampling

SR is widely adopted in hardware implementation because it provides a smaller sampling variance and is relatively simple to realize [4]. In this method, the selection of particles is based on a normalized cumulative sum of weights [4]:

$$w_k = \frac{W_k}{W_M} \tag{2.1}$$

where W_k is the prefix sum of particle weights up to the k-th particle, and W_M is the total sum.

The algorithm generates M ordered random numbers u_r using a single uniform offset $\tilde{u}$ according to the following formulation [4]:

$$u_r = \frac{1}{M}(r-1) + \tilde{u}, \quad \tilde{u} \sim U[0, 1/M) \tag{2.2}$$

The resampled particle set is formed by mapping each pointer u_r to a particle index k. For each $r \in \{1, \ldots, M\}$, the selected index k must satisfy the condition [4]:

$$w_{k-1} < u_r \leq w_k \tag{2.3}$$

The number of copies c_k of the k-th particle in the new set is determined by the number of pointers u_r that fall within its interval [4]:

$$c_k = \sum_{r=1}^{M} \mathbb{I}(w_{k-1} < u_r \leq w_k) \tag{2.4}$$

where $\mathbb{I}(\cdot)$ is the indicator function. This mechanism ensures that the expected number of copies is proportional to the particle weight. While the joint traversal between u_r and w_k can be theoretically parallelized using concurrent comparator arrays [4], such implementations are highly inefficient due to a massive number of redundant comparisons. Consequently, resolving this process in hardware often forces a reliance on serial execution. This inherently preserves a strong global data dependency, which remains the primary performance bottleneck for scaling high-speed hardware architectures.

2.3 Metropolis-Hastings Algorithm

To overcome the global data dependency inherent in SR, the MH algorithm is introduced as a localized alternative. Unlike standard methods, the MH approach is based on Markov Chain Monte Carlo (MCMC) theory and does not require the calculation of a global normalization factor or a prefix sum. To construct the Markov chain and explore the state space, each PE must first propose a candidate particle. The proposal distribution is designed as a discrete uniform random sampling over the existing pool of N particles. Specifically, a Pseudo-random generator (PRNG) produces an index $i \sim \text{Unif}\{1, \ldots, N\}$, and the particle stored at this corresponding memory location is fetched as the candidate. The transition to a new state is determined by the acceptance probability α [8]:

$$\alpha = \min\left(1, \frac{w(x^*)}{w(x)}\right) \tag{2.5}$$

where $w(x^*)$ is the weight of the proposed candidate particle fetched from memory, and $w(x)$ is the weight of the current particle held by the state machine. To implement this transition, a uniform random number $u \sim \text{Unif}[0, 1]$ is generated at each time step. The state at time $t+1$, denoted as $X(t+1)$, is determined by comparing u against the calculated α:

$$X(t+1) = \begin{cases} x^*, & \text{if } u \leq \alpha \\ X(t), & \text{if } u > \alpha \end{cases} \tag{2.6}$$

This mechanism ensures that the Markov chain is reversible and that its unique stationary distribution is proportional to the weights $w(x)$ [8]. In other words, each PE can perform its own transition independently, which enables high-efficiency

parallelization in hardware. As a consequence, the MH algorithm provides a mathematical foundation for resampling architectures to meet the low latency requirements of 6G applications.

2.4 Theoretical Trade-offs in Hardware Implementation

The selection of a resampling architecture involves a fundamental trade-off between resource utilization and processing latency. Given that parallelization of SR is redundant and hardware-inefficient, the design operates serially. Consequently, the algorithm requires N memory accesses due to the global dependencies, because next comparison cannot be resolved until the current comparison is completed. While optimization techniques, such as pre-fetching logic, can be used to mask this memory read latency and improve the throughput, they merely alleviate the issue without breaking the N sequential bottleneck. Alternatively, parallel architecture based on the MH algorithm removes this global dependency. Each PE requires N/P independent memory accesses per iteration, where N is the total number of particles and P is the total number of PEs. Over B iterations, each PE performs exactly $B \times (N/P)$ memory reads. Because each MH transition is entirely data-independent, these accesses can be perfectly parallel. However, this design requires more logic resources to implement multiple PEs, as each element must be equipped with its own PRNG, arithmetic datapath, and local memory. Understanding these theoretical boundaries guides the hardware design. SR is suitable when area is strictly limited and longer processing latency are acceptable. Conversely, the MH approach accepts a higher resource utilization in exchange for gains in speed. A detailed comparison based on hardware results will be provided in a subsequent section.

Chapter 3
Hardware Architecture Implementation

In this chapter, the hardware implementations of the resampling architectures are presented. Based on the theoretical analysis in Chapter 2, two distinct hardware designs are detailed: a sequential architecture utilizing a pre-fetching logic, and a parallel architecture based on the MH algorithm.

3.1 Sequential Architecture with Pre-fetching Buffer

As discussed in Section 2.2, the SR algorithm involves sequential data dependencies that lead to pipeline stalls during memory read operations. To mitigate this latency, a decoupled producer-consumer pipeline with a synchronous FIFO is implemented. In this architecture, the fetching of particle weights and the comparison logic are separated into two independent state machines.

3.1.1 Hardware Block Diagram

The top-level architecture of the sequential resampler consists of three primary modules: a Fetcher, a Synchronous FIFO, and a Consumer, as shown in Fig. 3.1.

The Fetcher operates as a state machine that generates read addresses to the external Block RAM (BRAM). The fetched weights are then pushed into a synchronous FIFO with a depth of $D = 8$. To manage the data flow and prevent FIFO overflow, an `almost_full` backpressure signal is routed from the FIFO to the Fetcher. The Fetcher stalls its address generation when this signal is asserted, ensuring data integrity while maintaining a buffer of weights for the comparison stage.

3.1.2 Pipeline Timing and Latency Mitigation

The inherent read latency of the synchronous BRAM causes execution delays if weights are fetched strictly on demand. In this implementation, the latency is mitigated through a decoupled pipeline rather than complex memory interleaving. Specifically, the BRAM is configured with an internal output register to improve timing closure, which introduces a synchronous two-cycle read latency from the address being registered to the data being valid. A two-stage shift register, consisting of `valid_d1` and `valid_d2`, is utilized to align the data valid signal with

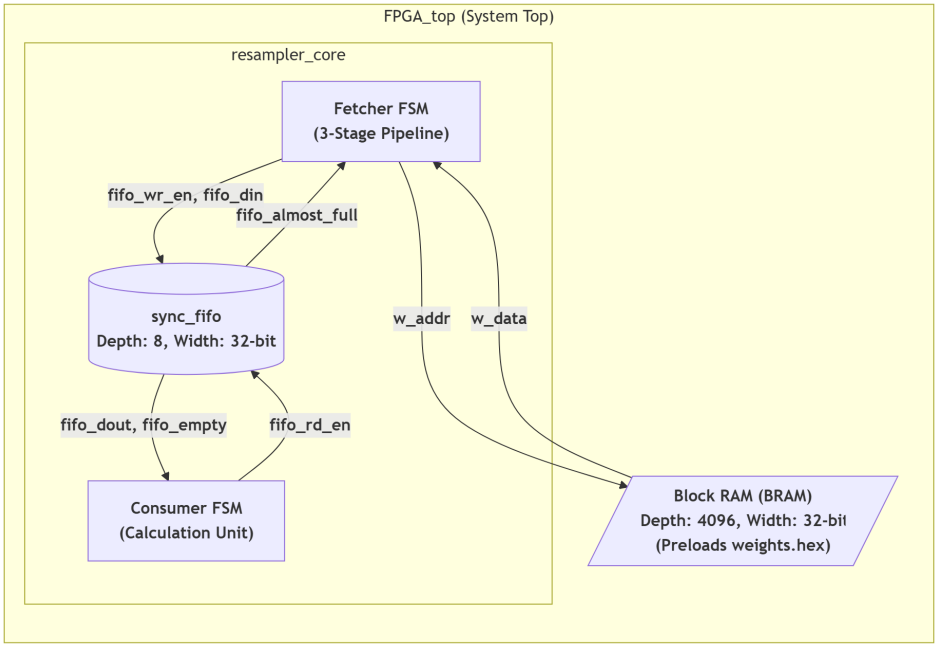

Figure 3.1: Hardware block diagram illustrating the Fetcher, Sync FIFO, and Consumer modules.

the fetched data. The shift register delays the write enable signal to match this inherent latency, ensuring that the data is correctly written into the FIFO exactly when it becomes available at the BRAM output. By enqueueing these weights into the FIFO in advance, the memory read latency is completely overlapped with the data consumption process. Consequently, when the Consumer requires a new weight, it reads directly from the FIFO with zero delay, effectively eliminating the pipeline stalls associated with memory access.

3.1.3 Consumer Execution Flow and Dynamic Accumulation

The Consumer module executes the SR comparison logic. Instead of storing the pre-calculated cumulative sum W_k in a dedicated memory block, the cumulative sum is calculated dynamically during execution. The hardware execution flow is illustrated in Fig. 3.2. In the processing state, the Consumer calculates the upper boundary of the current particle's interval by adding the raw weight from the FIFO to the accumulated sum register (`next_Q` = `Q_reg` + `fifo_dout`). A single hardware comparator evaluates the condition $T_{reg} < Q_{next}$, where T_{reg} represents the random threshold.

The execution follows two branches based on the comparison result:

HIT branch (Replication): If the threshold T_{reg} is less than the interval boundary, the current particle is replicated. The particle index is outputted, the write enable signal is asserted, and the threshold is incremented by a constant

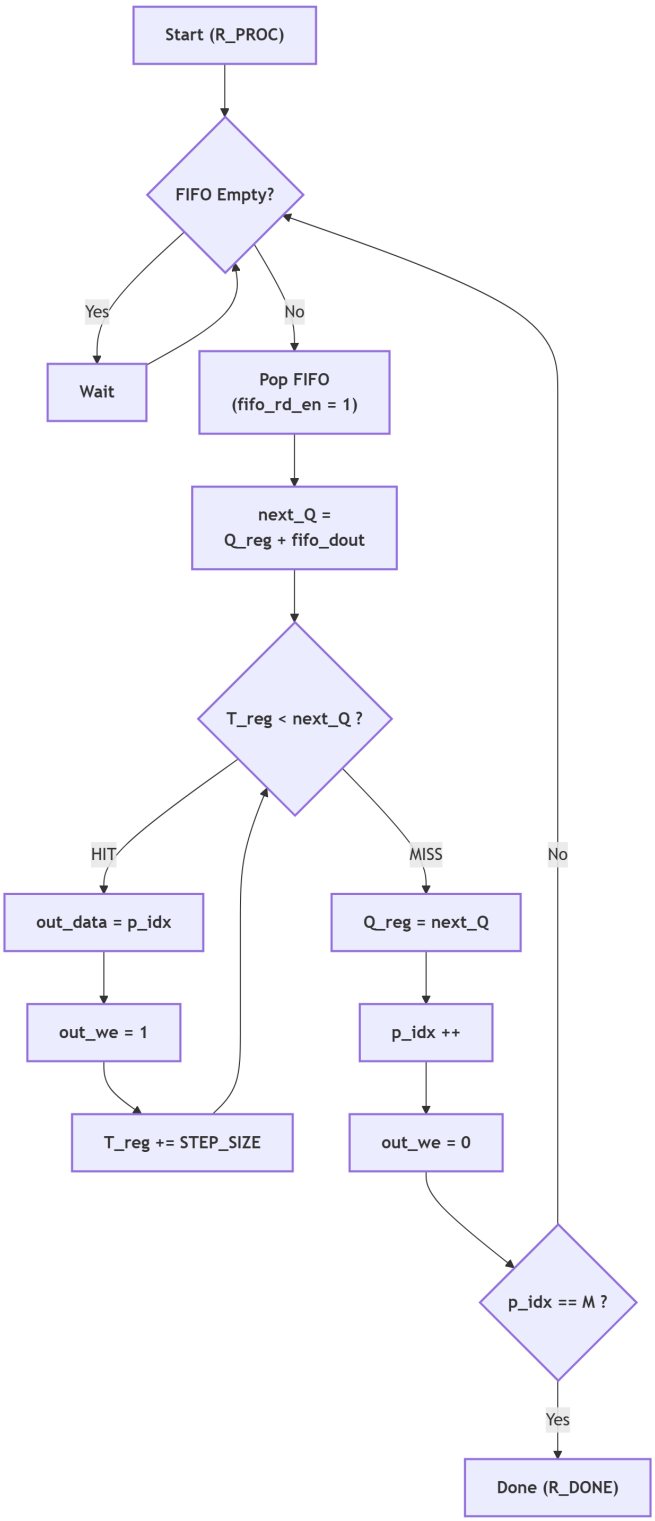

Figure 3.2: Execution flow chart of the Consumer state machine.

`STEP_SIZE`.

MISS branch (Consumption): If the threshold exceeds the boundary, the current weight is consumed. The cumulative sum register `Q_reg` is updated to `next_Q`, the particle index is incremented, and a read enable signal is asserted to pop the next weight from the FIFO.

This architecture utilizes a single comparator and dynamic accumulation to execute the resampling step, relying on the FIFO buffer to maintain throughput.

3.2 Parallel Metropolis-Hastings Architecture

While the sequential architecture detailed in Section 3.1 effectively mitigates the inherent read latency of the BRAM, its performance is fundamentally bounded by the $O(N)$ execution time. In scenarios demanding real-time tracking with a large number of particles, evaluating the resampling criterion sequentially incurs high latency. To overcome this bottleneck, it is necessary to shift the design paradigm from temporal iteration to spatial computation.

In this section, a parallel resampling architecture based on the MH algorithm is presented. By instantiating a distributed array of P independent PEs, the architecture evaluates candidate particles concurrently. The following subsections detail the hardware implementation of this parallel array, including the pseudo-random number generation, memory multiplexing, and the pipelined datapath.

3.2.1 Parallel Pseudo-Random Number Generation

In parallel architectures, the throughput of the random number generators directly affects the system performance. Instead of relying on traditional Linear Feedback Shift Registers (LFSRs) which may suffer from linear dependencies, this implementation adopts the Xorshift32 algorithm [9].

The Xorshift32 PRNG provides a period of $2^{32} - 1$ while requiring minimal hardware resources. According to the hardware implementation, each PRNG executes through three bitwise shifts (left 13, right 17, and left 5) and XOR operations. It should be noted that this calculation completes entirely within a single clock cycle without consuming any DSP slices. To ensure spatial uncorrelation across the processing array, each of the P address generators is initialized with a uniquely distributed deterministic seed (e.g., `32'hCAFE + i*32'hABC`). As a consequence, the random sequences generated by different PEs remain independent.

3.2.2 Top-Level Architecture and Resource Multiplexing

The parallel MH architecture requires dual memory spaces: one for the current state of the particles and another for the candidate pool. Assigning completely independent BRAMs to each PE would lead to a rapid depletion of on-chip memory resources. To address this issue, a specific memory architecture is implemented.

The memory resources are partitioned into two distinct categories: a private Working Memory and a Shared Source ROM, as illustrated in Figure 3.3. For the Working Memory, each PE is allocated a dedicated BRAM instance. This memory stores the accepted particle indices and weights, which implies that P

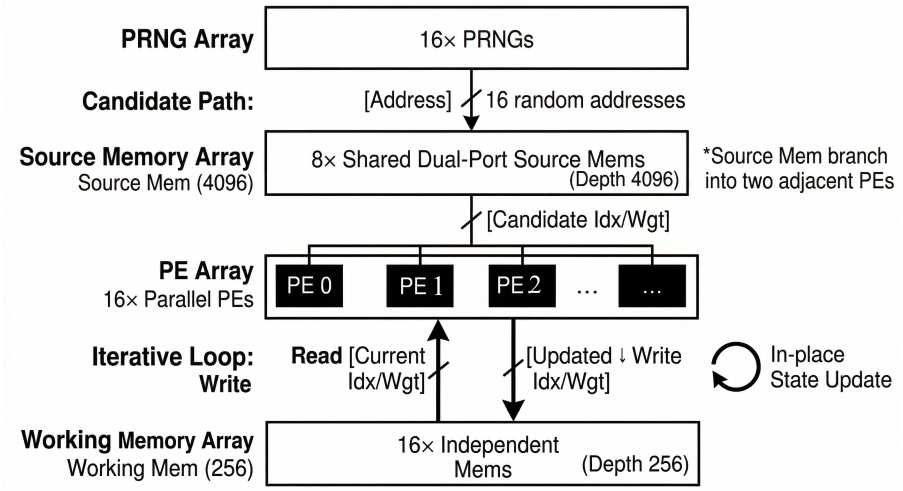

Figure 3.3: Top-level parallel MH hardware architecture, illustrating the 16-PE array, PRNG array, and the data paths for state updates.

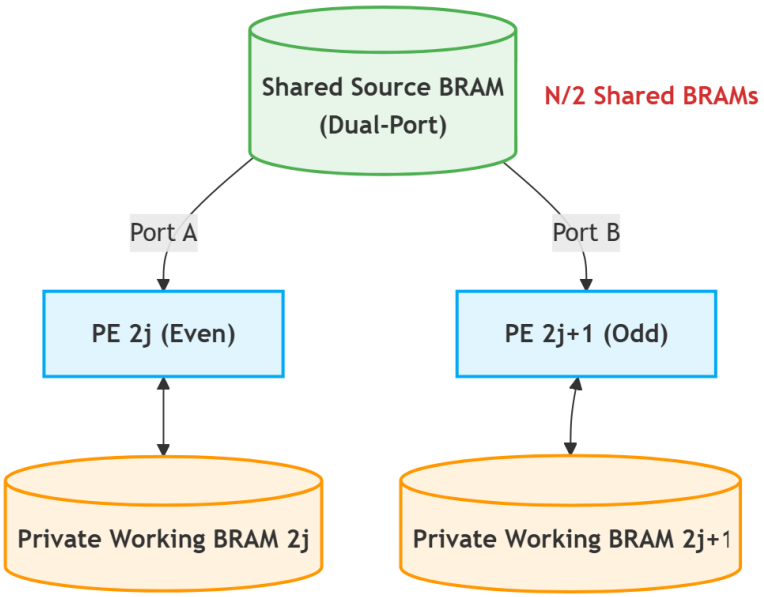

Figure 3.4: Memory architecture illustrating the private Working Memory and the shared Source ROM.

independent evolutionary trajectories can be written back concurrently without collision. On the other hand, to reduce the memory footprint of the candidate pool, the Shared Source ROM utilizes the inherent dual-port nature of the FPGA BRAM primitives. The top-level architecture instantiates $P/2$ Source ROMs. Port A of a given ROM serves the even-indexed PE $(2j)$, while Port B simultaneously serves the adjacent odd-indexed PE $(2j+1)$. This resource multiplexing strategy effectively halves the candidate memory utilization without introducing any stall cycles.

3.2.3 Interconnect Architecture Trade-Offs for Source Memory

To justify the adoption of the data replication strategy, an analysis of the interconnect architecture trade-offs is required. In the parallel MH architecture, P PEs must fetch candidate particles from the source memory simultaneously. If source memory is partitioned into B independent banks, the probability of any individual PE targeting a specific memory bank in a given clock cycle is $1/B$. However, a standard FPGA BRAM primitive only provides two independent read ports, a structural hazard occurs whenever a single bank receives strictly more than two requests ($k_i > 2$) simultaneously.

The first theoretical option is to divide the source memory into $P/2$ independent banks, implemented via a $P \times P/2$ crossbar. Theoretically, $P/2$ banks provide P read ports, perfectly matching the number of PEs. However, the statistical collision becomes severe. Assuming uniform random memory access for a 16 PE array targeting 8 dual-port banks, the number of requests directed to a specific bank in a single cycle, X, follows a binomial distribution $X \sim B(16, 1/8)$. Since a dual-port BRAM can resolve two concurrent requests, port contention occurs when $X > 2$. The probability of such contention is expressed as:

$$Prob(X > 2) = 1 - \sum_{k=0}^{2} \binom{16}{k} \left(\frac{1}{8}\right)^{k} \left(\frac{7}{8}\right)^{16-k} \approx 32.3\% \tag{3.1}$$

This mathematical result dictates that in any given clock cycle, nearly one-third of the memory banks will experience a port conflict. More importantly, resolving the systemic collision requires evaluating the joint probability. Since 16 requests must be strictly distributed across 8 banks, with a maximum capacity of 2 ports per bank, a completely conflict-free cycle only occurs if every single bank receives exactly two requests. The probability of such an ideal state is low:

$$Prob(\text{No Conflict}) = \frac{P!}{(2!)^{P/2} \cdot (P/2)^{P}} \approx 0.029\% \quad (\text{for } P = 16). \tag{3.2}$$

The denominator $(P/2)^P$ defines the total random sample space, representing all possible combinations of P independent PEs targeting $P/2$ memory banks. The numerator isolates the favorable outcomes by taking the full permutations of all incoming requests $P!$ and dividing it by $(2!)^{P/2}$ to eliminate the internal ordering redundancy. This result dictates that around 99.97% of clock cycles will experience at least one port collision, forcing ungranted PEs into non-deterministic stall cycles. To quantify the latency penalty introduced by these stalls, we evaluate

the expected throughput. The expected number of served requests, Y, per bank per cycle is given by

$$E[Y] = 1 \cdot Prob(X = 1) + 2 \cdot Prob(X \geq 2) \approx 1.494. \tag{3.3}$$

Across 8 banks, the actual system throughput drops to 11.95 particles per cycle, compared to the theoretical peak of 16. This intrinsic stochastic contention degrades pipeline throughput by over 25%, inflating execution latency by 34%, even before accounting for the physical delay of the crossbar logic.

To mitigate the massive stall penalties caused by port contention, the next step is to increase the number of read ports by dividing the source memory into P independent banks with $2P$ read ports, utilizing a $P \times P$ crossbar. For an array of 16 PEs and 16 banks, the probability of a cycle without conflict is shown below:

$$Prob(\text{No Conflict}) = \frac{1}{P^P} \sum_{\substack{\sum k_i = P \\ 0 \leq k_i \leq 2}} \frac{P!}{k_1! k_2! \dots k_P!} \approx 19.48\% \quad (\text{for } P = 16). \tag{3.4}$$

Applying the same multinomial model, this formulation sums across all valid distribution states where no single bank exceeds its dual-port capacity ($0 \leq k_i \leq 2$). The denominator updates to P^P, representing the enlarged sample space of P PEs targeting P banks. This mathematical result reveals that there is an over 80% probability of experiencing at least one collision in any given clock cycle. Let X denote the number of requests a single bank receives in this configuration, following a binomial distribution $X \sim B(16, 1/16)$, because the memory accesses driven by the PRNGs are completely independent. The expected number of successfully served requests per bank is $E[Y] \approx 0.908$. Across 16 banks, the actual throughput drops to 14.53 particles per cycle, inflating the execution latency by over 10%. Furthermore, a large crossbar scales quadratically. The programmable routing fabric dominates both the logic delay and area overhead [10]. This heavy routing challenges timing closure in FPGA, making the option remain sub-optimal.

To fundamentally bypass these global interconnect bottlenecks, we adopt the decentralized local memory strategy. Instead of using a shared topology, we replicate the candidate particles into $N/2$ independent BRAM instances. By directly assigning Port A to an even-indexed PE and Port B to an adjacent odd-indexed PE, we prioritize execution throughput at the cost of on-chip memory utilization. It completely avoids the routing delay of a crossbar and eliminates the considerable pipeline stall penalty. This design choice guarantees a deterministic pipeline latency and makes the parallel array highly robust.

3.2.4 Pipelined Processing Elements

To achieve a high frequency, the core evaluation logic within each PE is structured into a strictly aligned pipeline. The hardware datapath is shown in Fig. 3.5.

The evaluation logic inside the PE is divided into a three-stage pipeline. In the first stage, known as input buffering, the candidate and current particle data fetched from the BRAM are immediately registered into buffer signals (e.g., `cand_weight_s1`). This operation absorbs the routing delays from the BRAMs

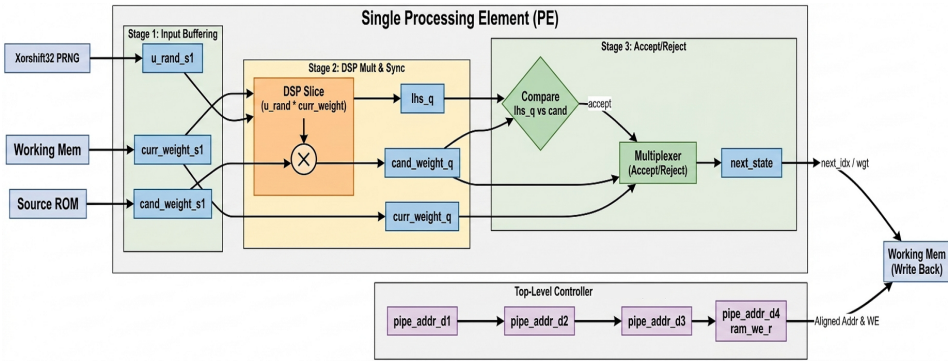

Figure 3.5: Hardware block diagram of the pipelined PE.

to the logic fabric, ensuring stable operands. Following this, the second stage executes the multiplication and synchronization. Because the evaluation criterion requires multiplying the current weight by a uniform random variable, this calculation (`u_rand_s1 * curr_weight_s1`) is explicitly mapped to dedicated DSP slices using the `use_dsp = "yes"` attribute to minimize combinational delay. Concurrently, the associated index and weight signals are delayed by one clock cycle to maintain data synchronization. Finally, in the third stage, a combinational logic block evaluates the MH acceptance condition based on the stable outputs from the preceding stage. The pipeline concludes by multiplexing the next state to either the candidate or current particle based on the binary acceptance decision.

Chapter 4

Experimental Setup and Verification Methodology

In order to rigorously evaluate the effectiveness and hardware efficiency of the architectures proposed in Chapter 3, an experimental platform and a verification methodology are required. This chapter details the specific FPGA development environment, the generation mechanism of customized stimulus data, and the proposed verification framework, which is utilized to guarantee both the logical correctness and the statistical fidelity of the hardware implementations.

4.1 Hardware Platform and Toolchain

It is widely acknowledged that the selection of the hardware platform directly impacts the timing closure and resource evaluation of digital systems. Therefore, the physical deployment and synthesis of the proposed resampling architectures are targeted at the AMD Xilinx Artix-7 FPGA family. Specifically, the `xc7a100tcsg324-1` device is deliberately selected as the primary testbed. It is worth mentioning that this specific chip provides a highly balanced distribution of on-chip resources, including abundant BRAMs and DSP48E1 slices, making it exceptionally suitable for evaluating both the sequential architecture and the parallel architecture.

Furthermore, the entire hardware design is implemented using SystemVerilog at the Register Transfer Level (RTL). Subsequently, the synthesis, physical implementation, and timing analysis are performed entirely within the Xilinx Vivado Design Suite 2022.2 Standard Edition. Compared with traditional behavioral estimations, the synthesis strategies in Vivado are strictly configured to optimize for timing closure, allowing for the extraction of highly accurate maximum operating frequencies (F_{max}) and reliable resource utilization reports. Meanwhile, MATLAB is adopted as the primary software environment to generate the input test vectors and establish the theoretical Golden Model for cross-verification. Finally, generative AI tools are used as secondary writing and formatting assistants. Their application is restricted to proofreading the English manuscript, refining sentence fluency, and troubleshooting LaTeX errors.

4.2 Stimulus Data Generation and Formatting

In the context of particle filtering, the distribution characteristics of the particle weights heavily dictate the execution latency, particularly in architectures featuring dynamic hardware loops or FIFO buffers. Specifically, in order to comprehensively simulate realistic tracking scenarios and evaluate the hardware robustness under extreme conditions, a customized MATLAB script is developed to generate a diverse set of weight distributions. First and foremost, a uniform distribution is generated to serve as the theoretical worst-case scenario for the sequential SR architecture. In this specific case, the absence of high-weight super particles maximizes the sequential traversal depth, thereby pushing the processing latency to its absolute upper bound. Subsequently, highly peaked distributions, modeled via Log-Normal functions, are synthesized to produce a small subset of super particles accompanied by a massive number of negligible noise particles. The fundamental motivation behind this peaky design is to intentionally induce severe replication bursts, thereby stress-testing the pipeline stalls and the `almost_full` backpressure logic of the synchronous FIFO. Furthermore, standard normal distributions are also incorporated to evaluate the baseline throughput and average latency levels of both the sequential and parallel architectures under typical operating conditions. By systematically injecting these diverse stimulus datasets into the FPGA, the processing capabilities of the proposed designs can be thoroughly analyzed across the entire spectrum of operational extremes. In addition, to successfully interface the floating-point software domain with the fixed-point FPGA environment, the generated weights must undergo a strict quantization process. The normalized floating-point weights are converted into a Q0.32 unsigned fixed-point format by applying a constant scale factor of 2^{32}. It should be noted that an explicit boundary check is simultaneously performed during this conversion to saturate any potential overflow values to the 32-bit maximum limit (`0xFFFFFFFF`). Finally, as a consequence of this formatting, the quantized data is exported as multiple hexadecimal memory initialization files (e.g., `weights_uniform.hex`, `weights_peaked.hex`), which are directly mapped into the FPGA BRAM primitives during the hardware synthesis phase.

4.3 Functional and Statistical Verification

Due to the distinct algorithmic natures of the two proposed architectures, verifying the hardware correctness requires a highly tailored approach rather than a universal output comparison. Therefore, a comprehensive verification strategy, precisely divided into a functional consistency check for the deterministic sequential design and a statistical consistency check for the stochastic parallel array, is implemented in this thesis.

First and foremost, the functional consistency check is exclusively conducted for the sequential architecture based on SR. It is acknowledged that SR exhibits a strictly deterministic execution flow under a fixed initial threshold. Accordingly, a Bit-True Golden Model is constructed in MATLAB to replicate the exact cycle-by-cycle state machine behavior of the RTL design. By directly comparing the 12-bit output indices generated by the FPGA against this Golden Model, a

zero mismatch confirms that the hardware timing logic and fixed-point accumulations of the sequential resampler strictly mirror the theoretical design without any state-machine deviations. On the contrary, the MH algorithm utilized in the parallel architecture is inherently based on MCMC stochastic transitions. As a result, expecting an exact cycle-by-cycle index match is impractical due to the stochastic nature of the MCMC transitions. To overcome this verification bottleneck, the statistical consistency is guaranteed by introducing a Probabilistic RMSE (P-RMSE) metric. This step calculates the histogram of the particles produced by the MH PE array and compares this hardware-generated frequency distribution directly against the theoretical input probability distribution. To elaborate, the continuous input probability distribution must first be mapped to expected counts, which is derived by multiplying the total particle population (N) by its corresponding input probability, and this often results in a non-integer value (e.g., expecting a particle to be duplicated 2.7 times). However, the hardware is constrained to duplicate particles in integer quantities (e.g., generating 2 or 3 copies). Although this mandatory rounding introduces an inherent quantization noise, it does not invalidate the evaluation metric. A sufficiently low P-RMSE explicitly confirms that the intensive hardware optimizations, such as BRAM multiplexing and parallel Xorshift32 pseudo-random number generation, preserve the foundational statistical fidelity required for successful convergence, without demanding rigid index-level synchronization. It is important to note that while this metric validates the hardware functionality at the module level, the definitive end-to-end data performance can only be fully characterized upon integration into the complete system. The detailed quantitative results of this evaluation will be presented in the subsequent sections.

4.4 Hardware Verification Environment

To comprehensively verify the effectiveness and feasibility of the proposed hardware architectures in a realistic physical environment, a hardware testing environment is constructed. The overall experimental setup block diagram is clearly illustrated in Fig. 4.1. As shown in the figure, the testbed fundamentally consists of a host personal computer (PC) and the target Artix-7 FPGA development board. The pre-calculated fixed-point particle weights are synthesized into the on-chip BRAMs as memory initialization files. A hardware push-button is configured to trigger the global reset and initiate the resampling process.

In order to bridge the communication gap between the physical FPGA and the host PC, a custom UART transmission controller is integrated into the top-level design. Once the core algorithms assert the done flag, the top-level finite state machine sequentially reads the 12-bit results from the Working Memory. Subsequently, these binary indices are converted into ASCII-encoded hexadecimal characters and transmitted to the host PC via the physical UART TX pin at a standard baud rate.

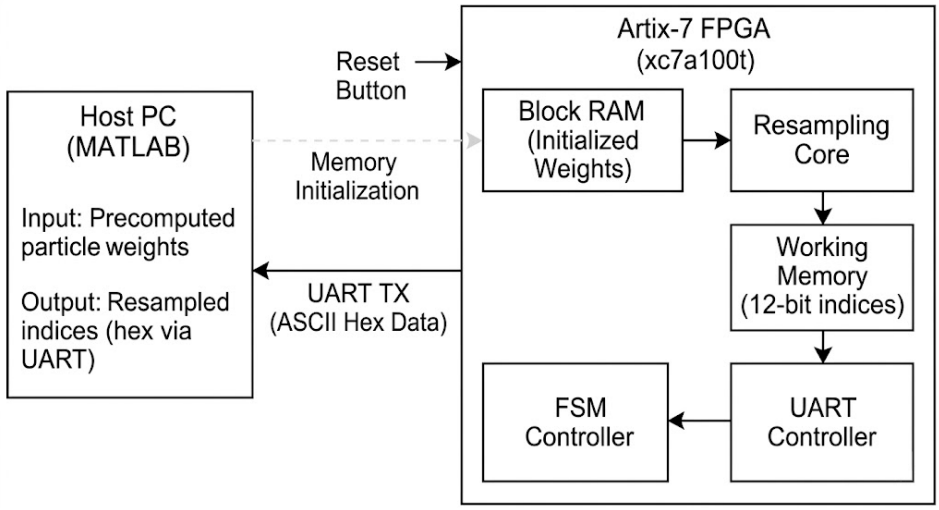

Figure 4.1: Block diagram of the testing platform, illustrating the data interaction between the host PC and the FPGA device.

Chapter 5
Hardware Results and Analysis

This chapter presents the physical evaluation results of the implemented resampling architectures. After validating the hardware outputs with the MATLAB software models, it provides a comparative analysis of resource utilization, execution latency and the statistical sampling quality.

5.1 Resource Utilization and Architectural Scalability

This section presents the hardware results under a 100 MHz clock constraint. We compare the sequential design and parallel architectures with 4, 8 and 16 PEs. The main purpose is to show the resource scaling ability and the hardware consumption data.

Table 5.1 shows the detailed resource consumption on the Artix-7 FPGA. We can see that the sequential architecture consumes 3296 LUTs and 4675 FFs but only uses 4 BRAMs. On the other hand the parallel 16 PE architecture consumes a similar amount of LUTs and FFs but requires 64 DSP slices and 48 BRAMs. This shows the hardware strategy of trading memory and computing resources for latency reduction.

Table 5.1: Resource Utilization for Different Resampling Architectures

Architecture	LUT	LUTRAM	FF	BRAM	DSP
Sequential	3296	435	4675	4	0
Parallel 4 PE	910	38	1154	14	16
Parallel 8 PE	1691	69	2188	24	32
Parallel 16 PE	3245	132	4259	48	64

When analyzing the parallel configurations, a non linear BRAM utilization discontinuity is observed. From 16 PEs to 8 PEs the BRAM usage drops from 48 to 24. This is a linear scaling. However when we scale down from 8 to 4 PEs the BRAM usage drops to 14 instead of the theoretically expected 12. After investigating the synthesis and Xilinx silicon architecture we found that this is a direct result of the physical geometry constraints of the Artix-7 BRAMs.

To understand this non linear scaling the hardware premise must be clarified. The fundamental memory building block in Artix-7 is the 36 Kb BRAM primitive known as RAMB36E1. All BRAM utilization reported by Vivado is calculated based on how many of these 36 Kb physical blocks or their 18 Kb halves are consumed.

The 16 PE baseline is analyzed first. It allocates 32 BRAMs for the source memory and 16 BRAMs for the working memory. The architecture employs 8 shared dual-port RAMs for the source. Each RAM must store the 4096 particle dataset with a 32 bit width. Given that a single 36 Kb block in a 32 bit width has a maximum depth of 1024, the synthesis tool must cascade 4 blocks in depth for each RAM. This configuration results in 32 BRAMs in total. For the working memory each PE requires a dedicated space to store its local state which demands a 44 bit data width. This width comprises a 12 bit index and a 32 bit weight. For the 16 PE configuration the required depth is 256. Since a 36 Kb block supports up to a 72 bit width for depths up to 512 the 256 by 44 data dimension fits into 1 block per PE. This accounts for 16 BRAMs and brings the system total to 48 BRAMs.

The scaling for 8 PEs is analyzed next. According to the Xilinx 7 Series Memory Resources Guide when the architecture scales down to 8 PEs the workload per PE doubles. This means the working memory depth increases to 512. Because a 36 Kb block supports a 512 depth with a maximum width of 72 bits, the 512 by 44 requirement still fits into a single block. Therefore the source memory and working memory both scale down linearly achieving a 50% resource reduction to a total of 24 BRAMs.

Finally the physical bottleneck for 4 PEs is discussed. When the architecture scales down to 4 PEs the working memory depth doubles again to 1024. According to the physical specifications once the depth reaches 1024 the maximum supported width of a single 36 Kb block drops to 36 bits. Because the data is 44 bits wide it exceeds this hardware limit. To accommodate the remaining 8 bits Vivado is forced to cascade an additional 18 Kb block side by side for each PE. As a result the working memory for a single PE now consumes 1.5 BRAMs instead of 1. The total BRAM for 4 PEs becomes 8 source memories plus 4 times 1.5 working memories which equals 14 BRAMs.

This architectural analysis confirms that the 8 PE configuration maximizes the depth to width ratio of the physical BRAM primitives. Scaling down further triggers a penalty due to the memory width constraints of the target silicon device. In conclusion this proves the current design accounts for the underlying silicon geometry and provides an objective resolution based on physical resource constraints.

5.2 Execution Latency and Throughput

This section analyzes the execution latency and throughput of different architectures under a 100 MHz clock frequency and a baseline of 4096 particles. Figure 5.1 illustrates the overall execution time comparison among the evaluated hardware designs.

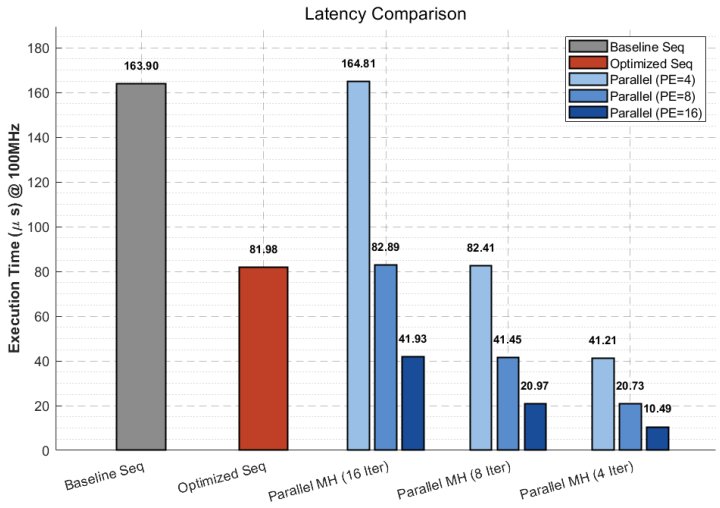

Figure 5.1: Execution time and latency comparison for the sequential and parallel architectures at 100 MHz with 4096 particles.

In the optimized sequential architecture, the total execution latency can be mathematically modeled by Equation 5.1[4].

$$C_{seq} = 2N + L_{seq\_pipe} \tag{5.1}$$

where N is the total number of particles and L_{seq_pipe} denotes the constant cycle overhead required for pipeline flushing and state machine transitions. The $2N$ multiplier dictates the absolute physical throughput ceiling of the system, indicating that the architecture consumes up to two clock cycles to process a single particle under worst-case data distributions. This represents the physical throughput limit of a single PE. According to the measurement data shown in Figure 5.1 the optimized sequential architecture consumes 8198 cycles. Substituting N equals 4096 into the equation shows that the core computation consumes 8192 cycles. Comparing this with the total latency reveals that the pipeline flushing and state machine transition overhead is extremely small which is exactly 6 cycles. This indicates that the timing optimization and throughput of the single core architecture have reached the theoretical throughput bound.

For the parallel architecture the execution latency is inherently determined by the algorithm parameters and the hardware scale. The mathematical model for the parallel architecture is shown in Equation 5.2 [7].

$$C_{par} = \frac{B \cdot N}{P} + L_{par\_pipe} \tag{5.2}$$

where B is the iteration parameter, P is the number of instantiated PEs and L_{par_pipe} represents the pipeline overhead. The measurement data is substituted into the equation for verification. When the iterations are set to $B = 16$, the theoretical computation cycles ($\frac{B \cdot N}{P}$) for $P = 4, 8$, and 16 are 163.84 μs, 81.92 μs, and

40.96 μs, respectively. Correlating these with the measured latencies, 164.81 μs, 82.89 μs, and 41.93 μs, the pipeline overhead L_{par_pipe} is explicitly isolated to exactly 97 cycles when $B = 16$. Similarly, when $B = 8$, the overhead consistently measures 49 cycles for all arrays; at $B = 4$, it measures 25 cycles. This mathematical results demonstrates two facts: First, scaling the PEs introduces zero routing or contention penalties. Second, the overhead is mathematically predictable, scaling as $L_{par\_pipe} = 6 \times B + 1$ cycles. This reflects the architectural pipeline depth, requiring 6 cycles of fill and flush time per iteration round.

Both analysis and Equation 5.2 provide objective proofs of the system latency determinism. In this mathematical model no variable is related to the input particle weight variance. This confirms from both mathematical and hardware perspectives that the execution latency of the system is locked to the constant calculated by the equation regardless of the input data distribution. Furthermore, Equation 5.2 reveals a linear relationship between the total execution cycles and the iteration parameter B. This linear characteristic provides the system with adjustability in different application scenarios. It allows the system to objectively reduce the execution time by decreasing the number of iterations under extreme latency constraints.

5.3 Accuracy under Variance Stress and Iteration Trade-Offs

This section evaluates the sampling accuracy of the hardware architectures under different levels of variance stress. Three specific weight distributions are defined to simulate realistic particle filtering scenarios. The uniform distribution represents a zero variance condition. The normal distribution represents medium variance and the peaky distribution simulates extreme weight degeneracy with high variance. Figure 5.2 presents a grouped bar chart comparing the P-RMSE of SR against the MH algorithm configured with 16 iterations.

A phenomenon must first be addressed regarding the SR baseline. We can see from the data that the SR algorithm achieves an absolute zero P-RMSE under the Uniform distribution. This mathematically validates that the evaluation metric isolates algorithmic behavior from measurement noise. However the SR P-RMSE under the Normal distribution (9.6×10^{-5}) is measurably higher than under the extreme Peaky distribution (7.26×10^{-5}). This is a direct manifestation of the algorithm's rounding errors when mapping continuous probabilities to integer replications. In the Peaky scenario, the vast majority of particles possess near zero weights, resulting in exact zero replications with no rounding error. Conversely, the Normal distribution assigns more balanced weights, generating thousands of non-integer expected replication counts. As detailed in Section 4.3, these fractional expectations inevitably trigger the quantization noise. The accumulation of these independent rounding errors naturally elevates the total P-RMSE, confirming that the accuracy fluctuations of SR are driven by integer quantization costs rather than statistical divergence.

Against this baseline the proposed Parallel MH architecture demonstrates convergence in regular operating conditions. Under both Uniform (2.46×10^{-4}) and Normal (2.42×10^{-4}) distributions the MH maintains a stable error margin. This

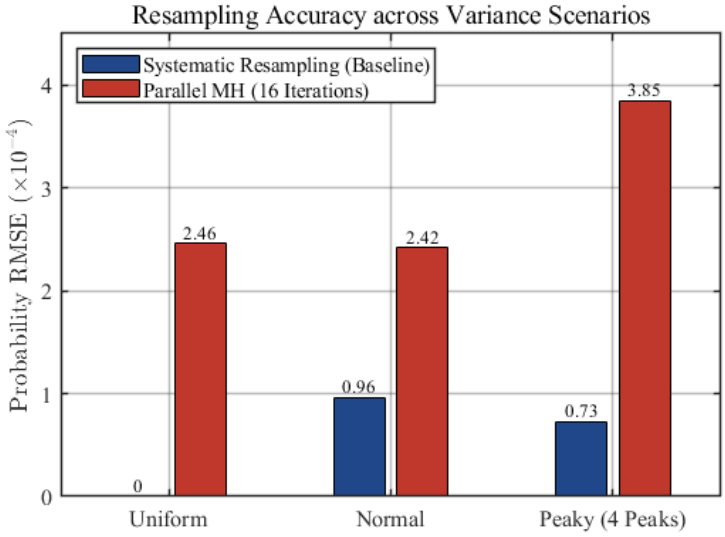

Figure 5.2: P-RMSE comparison between SR and MH at 16 iterations under different weight distributions.

consistency indicates that at an iteration depth of $B = 16$ the parallel MCMC transitions have thoroughly mixed and converged. The hardware error is suppressed near the physical lower bound of single particle quantization noise, rendering it robust against moderate weight fluctuations.

Under the high variance of the Peaky distribution, the MH P-RMSE degrades to 3.85×10^{-4}. This stems from the fact that MH resampling acts as a biased sampler that requires extended convergence times when navigating highly distorted probability spaces. When faced with an extremely steep probability peak, the localized state transitions of the MCMC process become susceptible to local optima within a finite number of steps [7]. However by sustaining $B = 16$ iterations across the 16 PE array this degradation is strictly contained within the same 10^{-4} magnitude order as the baseline, preventing algorithmic divergence.

This marginal loss in statistical precision under extreme scenarios is not a hardware flaw, but rather the intentional and necessary theoretical cost paid to completely sever the global data dependency inherent in the calculations of SR. By accepting this tightly bounded accuracy trade-off, the system unlocks the capability to deploy a fully decentralized 16 PE architecture, yielding the low latency required for next generation real time applications.

The trade-off dynamics between the number of iterations and the resulting accuracy are examined next to determine the optimal configuration. Figure 5.3 displays a line chart showing the P-RMSE across 4, 8, 16 and 32 iterations for the three weight distributions.

Under the Uniform and Normal distributions, the architecture exhibits a state of convergence at early iteration stages. For the Normal distribution, the P-RMSE achieves 2.54×10^{-4} at $B = 4$ and strictly locks at 2.42×10^{-4} from $B = 8$ through

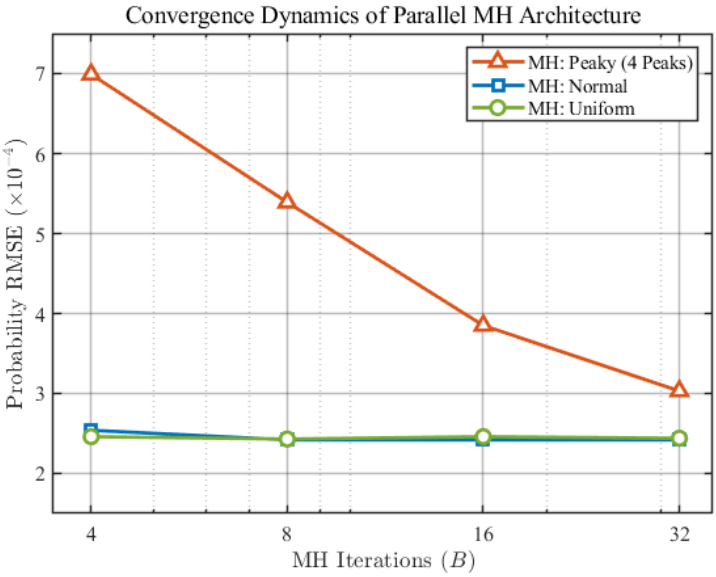

Figure 5.3: Convergence dynamics of the Parallel MH architecture across varying iteration counts under different variance scenarios.

$B = 32$. This precise plateau is not an algorithmic stall, but the physical manifestation of the system hitting its quantization noise floor. In a discrete hardware system of $N = 4096$ particles, the fundamental tracking resolution is physically constrained by the $\Delta = 1/N$ quantization step (approximately 2.44×10^{-4}). The data proves that in regular probability spaces, the proposed architecture exhausts its statistical potential in merely 8 iterations, with any subsequent transitions yielding zero additional mathematical benefit.

Conversely, the extreme Peaky distribution demands deeper Markov chain mixing. At a highly constrained configuration of $B = 4$, the P-RMSE elevates to 6.99×10^{-4}. Crucially, this proves that under severe iteration limits, the algorithm undergoes a controlled degradation rather than a statistical divergence. As iterations increase, the error monotonically decreases. However, a severe diminishing return is observed after $B = 16$. Transitioning from $B = 16$ to $B = 32$ doubles the hardware execution latency, yet yields a marginal statistical improvement of merely 0.82×10^{-4}.

The empirical convergence trends establish $B = 16$ as the optimal configuration. In this case, the system conservatively allocates mixing depth to ensure stability for regular distributions, while providing a necessary and sufficient buffer to suppress divergence during unpredicted peaky scenarios. Furthermore, these convergence dynamics mathematically validate the architecture's run-time elasticity. In strict physical layer scenarios facing hard deadlines, the system can dynamically downscale the iteration depth to $B = 4$ (achieving 10.49 μs execution time). This aggressive scheduling intentionally sacrifices a bounded, predictable

degree of precision to guarantee system maintaining timing compliance under strict latency constraints.

Chapter 6
Conclusion and Future Work

6.1 Summary of Contributions

This thesis presents the hardware design and implementation for Particle Filter resampling. Two distinct hardware architectures are proposed and evaluated on an Artix-7 FPGA platform.

An optimized sequential architecture is developed first. A producer consumer state machine and a synchronous FIFO buffer are implemented to hide the BRAM read latency. The experimental results demonstrate approximately a 2.0 times speedup compared to the standard baseline. The execution time is 81.98 μs at a 100 MHz clock frequency.

To bypass the sequential bottleneck completely a parallel architecture based on the MH algorithm is proposed. Utilizing the MCMC method eliminates the need for global normalization calculations. By modeling memory structural hazards through a multinomial distribution, the severe latency penalties of interconnection is quantified. Based on the theoretical proof, a decentralized local memory strategy is adopted. An array of PEs is constructed with multiplexed dual-port memory resources allowing concurrent processing of all candidate particles.

The physical evaluation reveals a direct area speed trade-off. The parallel architecture consumes more BRAMs and DSP slices than the sequential design in exchange for significant latency reduction. When the iteration parameter is set to 4 the latency is reduced to 1049 clock cycles equating to 10.49 μs. Compared to the baseline latency of 163.90 μs the parallel array achieves a 15.6 times speedup. Statistical evaluation confirms that the algorithmic bias introduced by the parallel architecture is restricted to a low magnitude. Trading mathematical convergence for hardware acceleration provides a viable structural solution for future advanced 6G applications.

6.2 Future Work

Based on the outcomes of this thesis several directions for future research are identified to further enhance the system capabilities.

The first direction is full system integration. The current evaluation focuses on the standalone resampling module. The next phase could involve integrating this hardware with particle generation and weight update stages to construct a

complete System. This will enable empirical evaluation using real channel data to assess the overall accuracy.

The second direction explores dynamic iteration scaling. While the current hardware utilizes a fixed iteration parameter, future implementations could dynamically adjust this number based on the immediate channel environment. This mechanism aims to conserve power consumption during stable tracking phases without sacrificing performance.

The third direction concerns Application Specific Integrated Circuit (ASIC) implementation. While the FPGA platform validates the logic functionality, transitioning the RTL design to a dedicated silicon process will yield significantly lower power consumption and higher operating frequencies.

Finally advanced random number generators should be investigated. The current architecture employs the Xorshift 32 algorithm. Future designs may incorporate the Mersenne Twister [11] or True Random Number Generators [12] to improve the statistical properties of the random sequences fed to the parallel array.

References

[1] D. Iancu, L. Liu, O. Edfors, E. Leitinger, and X. Li, "Low-latency D-MIMO localization using distributed scalable message-passing algorithm," *arXiv:2508.09546*, 2025.

[2] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Trans. Signal Process.*, vol. 50, no. 2, pp. 174–188, Feb. 2002.

[3] A. Hazarika and M. Rahmati, "Towards an evolved immersive experience: Exploring 5G- and beyond-enabled ultra-low-latency communications for augmented and virtual reality," *Sensors*, vol. 23, no. 7, Art. no. 3682, Apr. 2023.

[4] S. A. Alam and O. Gustafsson, "Improved particle filter resampling architectures," *J. Signal Process. Syst.*, vol. 91, no. 11, pp. 1234–1245, Nov. 2019.

[5] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEE Proc. F, Radar Signal Process.*, vol. 140, no. 2, pp. 107–113, Apr. 1993.

[6] M. Bolić, P. M. Djurić, and S. Hong, "Resampling algorithms and architectures for distributed particle filters," *IEEE Trans. Signal Process.*, vol. 53, no. 7, pp. 2442–2450, Jul. 2005.

[7] S. Liu, G. Mingas, and C.-S. Bouganis, "Parallel resampling for particle filters on FPGAs," in *Proc. Int. Conf. Field-Program. Technol. (FPT)*, pp. 191–198, 2014.

[8] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, Apr. 1970.

[9] G. Marsaglia, "Xorshift RNGs," *J. Stat. Softw.*, vol. 8, no. 14, pp. 1–6, Jul. 2003.

[10] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 26, no. 2, pp. 203–215, Feb. 2007.

[11] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator," *ACM Trans. Model. Comput. Simul.*, vol. 8, no. 1, pp. 3–30, Jan. 1998.

[12] T. E. Tkacik, "A Hardware Random Number Generator," in *Proc. International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pp. 450–453, 2002.

Printed by Tryckeriet i E-huset, Lund 2026

LUND
UNIVERSITY

Series of Master´s theses
Department of Electrical and Information Technology
LU/LTH-EIT 2026-1123
http://www.eit.lth.se