ZKPU: A Novel NVMe-Based Accelerator for Scaling Zero-Knowledge Proofs in Blockchain Systems

Dingsen Shi[†], Chris Tsu[‡], Ying He[†], Alex Goss[§], Donger Chen[†], Harry Fu[†], Yanbo Dai[†], Song Fu[†]
[†]*Department of Computer Science and Engineering*
University of North Texas, Denton, Texas, USA
{DingsenShi, YingHe, DongerChen, HarryFu, YanboDai}@my.unt.edu, Song.Fu@unt.edu
[‡]*Initio Corporation, San Jose, California, USA. christsu@outlook.com*
[§]*University of Texas, Dallas, Richardson, Texas, USA. abgoss52@gmail.com*

***Abstract*—Zero-Knowledge Proofs (ZKPs) are becoming a foundational technology for scalable and privacy-preserving blockchain systems, especially through applications like zkRollups. However, the computational intensity of proof generation continues to limit real-world deployment. We present ZKPU, a hardware-software co-designed ZK accelerator that combines native NVMe integration—ensuring seamless compatibility across existing server and edge infrastructure—with a modular RISC-V System-on-Chip (SoC) architecture that opens the path to eliminating host–device communication bottlenecks. ZKPU is designed to flexibly support a wide range of ZK workloads; in this work, we demonstrate its capabilities by implementing and optimizing multi-scalar multiplication (MSM), a core bottleneck in many zk-SNARK systems. Built using the Chipyard framework and equipped with dedicated modular arithmetic units, ZKPU achieves significant performance and energy efficiency improvements over CPU, GPU, and FPGA baselines. Our results highlight ZKPU as a practical and forward-compatible foundation for scalable ZK acceleration in modern decentralized systems.**

***Index Terms*—ZKP, Blockchain, NVMe, RISC-V SoC, Computational SSD, Stablecoin**

1. Introduction

Zero-Knowledge Proofs (ZKPs) [1] represent a transformative advancement in cryptographic technologies, enabling entities to validate statements without disclosing sensitive underlying data. Their ability to support privacy-preserving transactions, secure identity verification [2], scalable distributed computations, and confidential data exchanges positions ZKPs as pivotal tools across diverse sectors, including finance, healthcare, supply chain management, and blockchain-based systems like Ethereum [3] and Stablecoin [4]. Nevertheless, widespread adoption of ZKPs has been hindered by the substantial computational overhead involved in proof generation, creating a significant performance bottleneck [5].

Current hardware implementations designed to accelerate ZKP workloads face considerable limitations. ASIC-based approaches such as PipeZK [6] demonstrate impressive speedups but require costly custom silicon and struggle with scalability to large problem sizes due to memory bandwidth bottlenecks. FPGA solutions like if-ZKP [7] and CycloneMSM [8] provide flexibility but often exhaust hardware resources, rely on proprietary cloud FPGA platforms, and introduce significant host–device communication overhead. GPU-based systems such as GZKP [9] achieve strong performance but demand extensive precomputation tables exceeding several gigabytes, creating storage and deployment challenges. Collectively, these solutions suffer from prohibitive costs, restricted interoperability with commodity infrastructures, and reliance on proprietary toolchains. They also lack conformity with widely adopted datacenter communication protocols, limiting integration into hyperscaler environments. Moreover, by frequently reimplementing similar MSM and NTT kernels in isolated designs, these systems exacerbate development overhead and operational inefficiencies.

To overcome these constraints, we propose a hardware-software co-design that accelerates Multi-Scalar Multiplication (MSM) over elliptic curves through a pipelined FPGA implementation integrated under a standard NVMe protocol interface. Our primary contribution lies in demonstrating the practicality and performance of a modular, infrastructure-compatible ZK accelerator. This accelerator integrates seamlessly with existing commodity systems using standardized communication interfaces and host tooling, such as nvme-cli, to significantly lower the barrier to deployment. Additionally, as a self-contained RISC-V SoC, our system also eliminates potential host-device communication overhead and establishes a foundation for future iterations to support full end-to-end proof generation entirely on-chip.

Our system integrates several architectural components into a unified datapath: a FIOS-based Montgomery multiplier optimized for efficient modular arithmetic, signed-digit scalar encoding to reduce accumulator contention, a deeply pipelined Twisted Edwards point adder, and a hazard-aware scheduler to coordinate pipeline stages. While not novel individually, these building blocks are combined and synthesized under a custom NVMe endpoint, enabling plug-and-play deployment in conventional server infrastructure.

Importantly, our design exemplifies a shift toward accessible and scalable ZK acceleration: by validating performance on a real-world FPGA prototype with standard Linux tooling, we demonstrate throughput on par with leading custom MSM implementations—without sacrificing portability or compatibility. This work provides a concrete step toward broader adoption of zero-knowledge computing by enabling "Proof-as-a-Service" (PaaS) models within existing data center environments.

The paper is organized as follows. Section II provides background and motivation, introducing ZKP applications in data exchanges, an overview of the proof generation [10] process, and a detailed discussion on ASIC adoption, including motivations and challenges. Section III describes the proposed system architecture for ZKP acceleration. Section IV presents the implementation details of our solution. Section V provides a comprehensive performance evaluation. Section VI and Section VII conclude the paper with a summary and implications of the findings and discuss directions for future research.

2. Zero-Knowledge Proof Construction

This section begins by introducing the challenges and opportunities associated with Zero-Knowledge Proofs (ZKPs) through a real-world data exchange use case (Section 2.1). It then examines the proof generation process (Section 2.2), emphasizing its computational demands and the corresponding hardware requirements. Finally, it discusses the limitations of current hardware solutions and the challenges they present (Section 2.3), highlighting the need for a more efficient and compatible approach to ZKP generation.

2.1. Zero-Knowledge Proofs in Data Exchange Applications

A Scenario: Company A, a financial institution, intends to sell aggregated credit risk data to Company B, a fintech startup, via a regulated Data Exchange.

Challenges: Company A must protect its proprietary transactional data and adhere strictly to data privacy regulations. Conversely, Company B seeks reliable assurances regarding data quality, anonymization compliance, and regulatory adherence before committing to purchase.

A ZKP Solution: To address these requirements, Company A employs Zero-Knowledge Proofs (ZKPs) to verifiably demonstrate:

- The dataset aggregates information from at least 10,000 authenticated financial transactions.
- Data anonymization fully complies with regulatory privacy standards.
- Individual transaction details and sensitive information remain undisclosed throughout the verification process.

Outcomes: Company B proceeds confidently with the dataset acquisition, reassured by the robust assurances of compliance and data integrity. At the same time, Company A successfully preserves the confidentiality of its proprietary data while consistently adhering to stringent regulatory requirements.

2.2. Proof Generation

The illustrative scenario presented highlights the transformative potential of Zero-Knowledge Proof (ZKP) technology to efficiently overcome complex privacy and compliance challenges. Despite their considerable promise, ZKPs have not been widely adopted in practice, predominantly due to the computationally demanding nature of proof generation. Contemporary benchmarks indicate that proof generation on standard CPUs remains excessively time-consuming, frequently extending to several hours even for relatively simple computations, limiting their applicability. Leveraging specialized hardware accelerators emerges as a critical strategy to address this computational bottleneck. The subsequent sections delve deeper into the proof generation process and the specific computational obstacles involved.

2.2.1. Proof Generation Process Overview.

Proof generation [10] initiates with deterministic computations executed within a Zero-Knowledge Virtual Machine (ZKVM), exemplified by implementations such as RISC0 [11] or Scroll [5] [12]. The ZKVM yields an execution trace encapsulating every computational step, laying the foundation for subsequent proof generation. Contemporary benchmarks indicate that proof generation on standard CPUs remains excessively time-consuming, frequently extending to several hours even for relatively simple computations, limiting their applicability. Leveraging specialized hardware accelerators emerges as a critical strategy to address this computational bottleneck. The subsequent sections delve deeper into the proof generation process and the specific computational obstacles involved.

Following the creation of this trace, several computationally intensive tasks ensue, notably:

- Witness Generation: The process of constructing auxiliary data that confirms the validity of computational results.
- Multi-Scalar Multiplication (MSM): Conducting elliptic curve operations involving multiple scalar-vector multiplications and additions.
- Number-Theoretic Transform (NTT): Executing polynomial multiplications under modular arithmetic, integral for efficient polynomial evaluations.

Computational Complexity. Despite their theoretical simplicity, these computational stages necessitate intensive modular arithmetic involving elliptic curve elements and large integers, posing a significant obstacle to broader ZKP adoption. Each stage exhibits distinct hardware requirements:

- Witness Generation: Sequential execution, favoring flexible general-purpose processors such as CPUs.

- MSM: Highly parallelizable, arithmetic-intensive computations best accelerated by specialized ASIC hardware optimized explicitly for elliptic curve arithmetic.
- NTT: Memory-intensive operations demanding specialized hardware featuring robust memory bandwidth, ideally ASICs incorporating advanced memory solutions such as High Bandwidth Memory (HBM).

In this paper, we focus primarily on MSM acceleration, detailing our hardware-software co-design strategies and architecture optimizations that maximize throughput and efficiency in real-world ZK proving workloads.

2.3. Elliptic Curves

An elliptic curve over a finite field F_q, is defined by a non-singular cubic equation in the Weierstrass form:

$$E : y^2 = x^3 + ax + b \tag{1}$$

To ensure non-singularity, the curve parameters must satisfy the discriminant condition $4a^3 + 27b^2 \neq 0$. The set of F_q-rational points on this curve, denoted $E(F_q)$ forms an abelian group under a geometric addition law, with the point at infinity O serving as the identity element.

In cryptographic applications, we are typically interested in elliptic curves for which $E(F_q)$ contains a large prime-order subgroup. Let r be such a large prime and $G(F_q)$ denote the subgroup of order r. Operations in this group—especially scalar multiplication—are foundational to elliptic curve cryptography and zero-knowledge proof systems.

To optimize arithmetic and reduce the cost of group operations, alternative curve representations are often used. One such form is the Twisted Edwards curve [13], defined by the equation:

$$E_T : -x^2 + y^2 = 1 + \frac{k}{2}x^2y^2,\ k \in F_q \tag{2}$$

This representation offers complete addition formulas and more uniform behavior, which are advantageous for constant-time implementations. Although every Twisted Edwards curve is birationally equivalent to some Weierstrass curve, the reverse is not true in general.

A prominent example is the BLS12-377 curve, defined over a 377-bit prime field $F_q$with

$$E : y^2 = x^3 + 1 \tag{3}$$

$$E : y^2 = x^3 + 1$$

This curve is pairing-friendly with embedding degree 12 and supports a subgroup of prime order r suitable for cryptographic constructions. It also admits an equivalent Twisted Edwards form E_T as shown above, with a specific constant k. This duality enables the use of whichever representation is more efficient for a given operation—Weierstrass form for pairings, and Twisted Edwards form for scalar multiplication and MSM.

2.4. MSM and Pippenger Algorithm

In many cryptographic systems based on elliptic curves—such as SNARKs [14] and STARKs [15]—a core computational bottleneck is Multi-Scalar Multiplication (MSM). Given a list of elliptic curve points $P_1, P_2, \dots, P_n \in G$ and corresponding scalars $s_1, s_2, \dots, s_n \in F_r$, the goal of MSM is to compute the sum:

$$\text{MSM}(\{(s_i, P_i)\}_{i=1}^n) = \sum_{i=1}^{n} s_i P_i \tag{4}$$

A naive implementation would perform n scalar multiplications, each requiring $\mathcal{O}(\log r)$ point additions. However, this becomes inefficient for large n, particularly in proving systems where n can reach millions. To address this, specialized algorithms, such as Pippenger's algorithm, are employed to reduce the number of expensive curve operations.

Pippenger's algorithm [16] accelerates MSM by grouping scalars and points into buckets based on a windowed representation of the scalars. The algorithm proceeds as follows:

1) *Windowing Scalars:* Each scalar s_i is split into digits in base 2^w, forming a width-w representation:

$$s_i = \sum_{j=0}^{\lceil \log_2 r/w \rceil - 1} s_{i,j} \cdot 2^{jw}, \quad \text{with } s_{i,j} \in [0, 2^w) \tag{5}$$

2) *Bucket Accumulation:* For each digit position j, the algorithm creates $2^w - 1$ buckets. Each point P_i is placed into bucket $B_{s_{i,j}}$, skipping zeros.
3) *Summing Buckets:* For each window j, the buckets are summed in reverse order to minimize additions:

$$S_j = \sum_{k=1}^{2^w - 1} B_k \tag{6}$$

where the sum is computed using a ladder-style summation.
4) *Final Summation:* The MSM result is reconstructed as:

$$\sum_j 2^{jw} S_j \tag{7}$$

Pippenger's algorithm reduces the total number of group operations from $\mathcal{O}(n \log r)$ to approximately $\mathcal{O}(n)$ additions and $\mathcal{O}(n/w + 2^w)$ group operations. The window size w is selected to balance memory and performance. In practice, Pippenger's method enables scalable and efficient MSM suitable for high-throughput hardware implementations.

3. System Architecture of ZKPU

The objective of this paper is to develop ZKPU—a Zero-Knowledge Proof chip architecture designed for high-performance, cost-efficient proof generation. Our current focus is on accelerating Multi-Scalar Multiplication (MSM),

showcasing how specialized hardware modules integrated with NVMe connectivity can significantly improve throughput and integration flexibility. While future iterations will extend to full end-to-end proving—including Number-Theoretic Transform (NTT) and witness generation—the work presented here establishes a foundational step toward a modular, scalable, and cloud-compatible ZKP accelerator. By leveraging FPGA-based IP and a forward-compatible ASIC design, ZKPU aims to democratize access to ZKP capabilities in both datacenter and decentralized environments.

3.1. Key Challenges for ZKPU

3.1.1. NVMe Protocol Support.

One of the greatest challenges of existing ZK hardware solutions is their incompatibility with modern infrastructure. Current designs rely on proprietary, closed systems that are expensive, bulky, and difficult to maintain. These solutions lack support for standard interfaces, making them incompatible with existing motherboards and datacenter-based environments. This severely limits their accessibility, restricts adoption to large enterprises, and exacerbates centralization in ZKP applications.

To address these issues, our approach integrates support for the NVMe protocol, a universally adopted standard in modern computing. This ensures seamless compatibility with existing infrastructure while providing scalability, modularity, and adaptability to evolving workloads. Below, we delve into what NVMe is and how it helps solve these challenges.

Non-Volatile Memory Express (NVMe is a widely adopted storage protocol designed for high-performance and efficient access to non-volatile storage media, such as SSDs. It is supported across virtually all modern operating systems and computing platforms, making it a key component in ensuring compatibility with existing infrastructure.

To address the shortcomings of existing ASIC-based proof generation solutions, our architecture fully integrates support for the NVMe protocol, offering significant advantages over current designs that rely on proprietary communication systems:

Seamless Compatibility with Existing Infrastructure: Unlike closed and proprietary solutions, NVMe is a universally supported standard across modern computing platforms. Integrating NVMe allows our hardware to connect effortlessly with existing motherboards and systems, eliminating the need for custom drivers or complex configurations and significantly reducing deployment complexity and costs. For instance, in our demo, we showcase direct interaction with our MSM module on a VCU118 FPGA, leveraging NVMe's native support on an Ubuntu 22.04 system using nvme-cli tools, running on a regular SuperMicro motherboard.

Higher Server Capacity. A single server can support up to 24 NVMe cards or more, which is at least three times the typical number of PCIe slots available. This increased capacity enables each server to host a greater number of ZKPU cards, substantially boosting the overall proof generation throughput while ensuring efficient utilization of resources. Figure 1 illustrates such a deployment: a standard 4U SuperMicro server populated with multiple ZKPU cards in the front NVMe bays, while optional GPUs can still be installed in the rear PCIe slots. This configuration highlights the key advantage of NVMe-based ZKPU integration—servers can simultaneously host dozens of accelerators without being constrained by limited PCIe slots, while retaining flexibility for heterogeneous computing with GPUs when required.

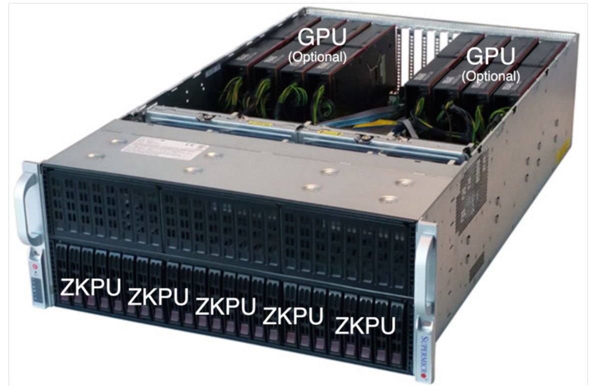

Figure 1. ZKPU in Server

Decentralized Enablement through Personal Provers. By adhering to the NVMe standard, our ZKPU can operate in any environment that supports modern operating systems—ranging from enterprise-grade servers to consumer desktops and edge devices. This accessibility enables individuals and smaller organizations to deploy personal provers without requiring specialized infrastructure or proprietary drivers. As a result, the barrier to participation in ZK systems is significantly lowered, fostering a broader and more decentralized proving ecosystem. While data center-scale deployments remain critical for high-throughput use cases, the ability to run provers at the edge complements centralized infrastructure and strengthens the overall resilience and trust model of decentralized networks.

3.1.2. RISC-V SoC.

While this paper focuses on accelerating MSM, we have designed our architecture with future extensibility in mind. One major bottleneck in existing ZK systems is the need to transfer witness data between the CPU and accelerator over PCIe—a costly step that underutilizes compute resources due to bandwidth limitations. To address this, our architecture incorporates a general-purpose RISC-V CPU [17] directly onto the ZKPU chip. This enables the possibility of running the witness generation process locally, avoiding expensive chip-to-chip data transfers. Though not implemented

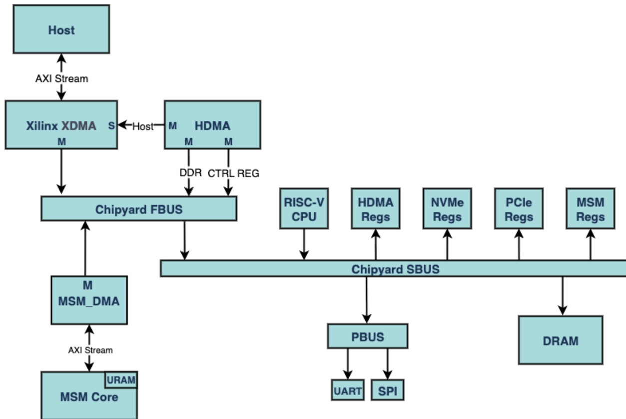

Figure 2. System Architecture

in this work, this direction allows future iterations of our system to support full end-to-end proof generation on-chip. By combining CPU-driven control logic with tightly integrated accelerators and a high-bandwidth on-chip interconnect, we aim to eliminate the data movement bottleneck and unlock significantly higher system efficiency.

3.2. Design of ZKPU

This section presents the overall design of our MSM acceleration system, focusing on two key aspects: the system-level architecture and the internal design of the MSM core. First, we describe the complete data movement flow between the host and device—how the host uploads the witness data, triggers computation, and retrieves results—through a standard NVMe interface and a Chipyard-based SoC. Second, we dive into the internal architecture of the MSM core itself, detailing how scalars and points are buffered, how the bucket-based algorithm is realized in hardware, and how we employ fast modular arithmetic—particularly the FIOS Montgomery method [18]—to accelerate elliptic curve operations. These two parts together form a tightly integrated, NVMe-compliant solution for accelerating ZK proof generation.

3.2.1. System Architecture and Workflow Operations.

Our MSM acceleration system is designed as a tightly integrated hardware-software co-design that exposes zero-knowledge proof (ZKP) primitives through the NVMe protocol. At a high level, the system consists of a host processor and a device composed of a custom RISC-V SoC implemented using the Chipyard framework [19]. The goal is to offload the computationally intensive multi-scalar multiplication (MSM) operation onto the device while maintaining host transparency through a standard NVMe interface. The overall system structure is illustrated in Figure 2.

The hardware architecture integrates the following major components:

- **RISC-V SoC (Chipyard-based).** We build the SoC using UC Berkeley's Chipyard platform, which provides a flexible and modular framework for RISC-V hardware development. Chipyard enables seamless integration of custom accelerators and DMA engines with the rest of the SoC, and provides robust support for TileLink and AXI interfaces. In our design, the RISC-V core is primarily responsible for lightweight firmware execution, register handling, and control sequencing.
- **Host DMA (HDMA).** A high-bandwidth DMA engine that handles bidirectional data movement between host memory and the on-chip DDR memory. HDMA is triggered by host-initiated NVMe commands and transfers witness data (scalars and elliptic curve points) as well as results.
- **MSM Core.** A dedicated hardware module optimized for elliptic curve MSM, attached to the SoC as a memory-mapped peripheral. It operates independently once configured and supports high-throughput scalar-point operations over large datasets.
- **On-chip DDR Memory.** Used as a shared buffer to store intermediate and final data, including input vectors and MSM results. It serves as the primary communication medium between HDMA, the RISC-V SoC, and the MSM accelerator.
- **NVMe Register Interface.** The system exposes its configuration and control registers using a PCIe BAR-compliant NVMe register map, allowing seamless integration with standard Linux NVMe drivers and tools like nvme-cli.

Chipyard plays a central role in our hardware design, serving as the backbone for system integration. Its object-oriented hardware generators and modular SoC infrastructure allow us to embed the MSM core and HDMA engine alongside the Rocket core with minimal glue logic. We leverage Chipyard's TileLink [20] buses for SoC-level interconnects and customize memory maps to expose all control interfaces through the NVMe BAR region. Chipyard's build system also simplifies simulation, synthesis, and eventual FPGA deployment of the full system.

The execution of an MSM task involves a carefully orchestrated sequence of interactions between the host and the device. The process proceeds as follows.

1) *Device Enumeration and Register Access*: Upon boot or insertion, the device is recognized as a standard NVMe endpoint. The host uses NVMe admin commands to access identification and custom vendor-defined registers mapped through the PCIe BAR.
2) *Data Upload via HDMA*: The host writes the input scalars and elliptic curve points to the device using standard NVMe I/O write commands. These are interpreted by firmware as HDMA transactions and streamed directly into DDR memory.
3) *MSM Triggering*: The host signals the start of computation by writing to a control register (ex-

posed through NVMe BAR). This action notifies the RISC-V firmware, which configures the MSM core and initiates execution using predefined memory addresses.

4) *Computation and Completion Notification*: Once launched, the MSM core operates autonomously. Upon completion, the device sends an asynchronous NVMe event to notify the host that results are ready.
5) *Result Retrieval*: The host issues an NVMe read to fetch the result vector from DDR memory via HDMA.

The design ensures minimal host intervention, eliminates the need for custom drivers, and enables efficient interaction using standard NVMe semantics. Importantly, our system is fully backward compatible with existing server infrastructure. As demonstrated in our working prototype [21], the accelerator operates seamlessly on an off-the-shelf SuperMicro X11SSZ-QF motherboard running Ubuntu 22.04, without any kernel patches or platform modifications. By combining this tightly coupled control model with Chipyard's extensibility, our system delivers a practical path toward scalable ZKP acceleration over industry-standard interfaces.

3.2.2. MSM Core Design.

Our MSM core is a deeply pipelined hardware unit built to accelerate elliptic curve multi-scalar multiplication (MSM) using the Pippenger algorithm. It integrates four key components that form a coordinated pipeline.

- **Scalar Representation Transformation** converts incoming scalars into a signed-digit form, reducing the number of point operations since negative digits can be handled by trivial point negation. The transformed scalar–point pairs are buffered in the Input FIFO.
- **Scheduler** reads scalar–point pairs from the Input FIFO and assigns them to the appropriate buckets according to Pippenger's algorithm. If multiple operations target the same bucket simultaneously, it temporarily redirects one into the Stall FIFO to resolve conflicts, ensuring high pipeline utilization without hazards.
- **Point Arithmetic** performs elliptic curve point additions using Twisted Edwards mixed-coordinate formulas, which minimize data conversion overhead.
- **Field Arithmetic** underlies all point operations, providing modular multiplications, additions, and reductions via a Montgomery-based design. The Adder block in Figure 2 instantiates these field operations as the computational backbone of the MSM pipeline.

Figure 3 shows how the four components are realized as a streaming pipeline on FPGA. Scalars, once transformed into signed-digit form, enter the Input FIFO, which buffers host-to-core transfers. The Scheduler reads from the FIFO and directs operations to the correct bucket; conflicts are deferred to the Stall FIFO to keep the pipeline hazard-free. The

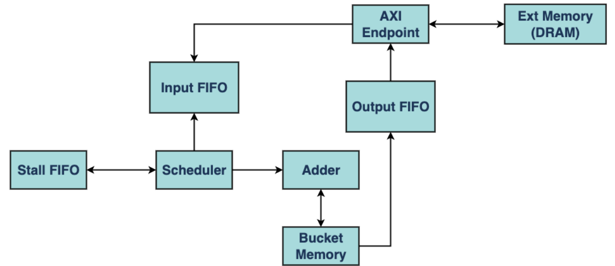

Figure 3. MSM Core Architecture

Adder, powered by Montgomery-based Field Arithmetic, performs Twisted Edwards point additions and updates the Bucket Memory. Partial results are then moved through the Output FIFO and AXI Endpoint into external DRAM, enabling support for large-scale MSM workloads.

To make the operation of the MSM core clearer, we next elaborate on each of the four components in detail. Specifically, we explain how field arithmetic forms the computational foundation, how scalars are preprocessed before entering the pipeline, how the scheduler coordinates bucket assignments, and how the point arithmetic integrates these building blocks to execute MSM efficiently.

Scalar Representation Transformation. To reduce storage and arithmetic overhead in the MSM pipeline, we transform all input scalars into a signed-digit representation on-chip, before they reach the point adder. Given a scalar k represented as $k = \sum_{i=0}^{N-1} 2^{o_i} d_i$ with unsigned digits $d_i \in [0, 2^{b_i} - 1]$, we iteratively convert them into a signed-digit form $d_i \in [-2^{b_i-1} - 1, 2^{b_i-1} - 1]$ by applying a windowed rebalancing transform. The transformation is fully unrolled in hardware and implemented as a pipelined module with optional skid buffers to manage backpressure.

Scheduler Design for Conflict Resolution. In the Pippenger algorithm [16], each scalar contributes to a bucket stored in RAM. The scheduler's job is to coordinate these additions to ensure correctness in the presence of pipeline latency. While the underlying task—adding a point to a bucket—is conceptually straightforward, the deeply pipelined nature of our point addition unit introduces hazards that must be carefully managed. To overcome this, we develop a hazard-aware scheduler [22] which works as follows while the steps are specified in Algorithm 1.

- We track all bucket indices currently in flight.
- The stalled entries are revisited under certain conditions, such as when a batch of in-flight operations has sufficiently cleared or when the FIFO reaches a pressure threshold.
- When a new scalar is scheduled, we check for conflicts with the in-flight bucket set.
- If a conflict is detected,
  - The scalar and associated point are placed into a stalled FIFO.

– A bubble (NOP) is inserted into the pipeline for that cycle.

Each window of the Pippenger algorithm has its own set of FIFOs and hazard tracking logic. This enables multiple windows to be processed in parallel, without introducing inter-window dependencies.

This design avoids the need for probabilistic or delayed rescheduling techniques. Instead, it relies on efficient real-time conflict tracking and selective reattempts, maintaining high throughput while ensuring correctness.

Algorithm 1 Hazard-Aware Scheduling with Conflict Tracking

```
1:  Input: Stream of (s_i, P_i) pairs for window w
2:  Data: InFlightBuckets[0..T-1], StalledFIFO
3:  for each cycle t do
4:      if ShouldRetryStalled() then
5:          (s', P') ← StalledFIFO.pop()
6:          k' ← GetBucketIndex(s')
7:          if k' ∉ InFlightBuckets then
8:              LaunchToPipeline(s', P')
9:              InFlightBuckets.insert(k')
10:         else
11:             StalledFIFO.pushBack(s', P')
12:         end if
13:     else
14:         if NewInputAvailable() then
15:             (s, P) ← ReadNextScalarPoint()
16:             k ← GetBucketIndex(s)
17:             if k ∈ InFlightBuckets then
18:                 StalledFIFO.push(s, P)
19:                 InsertBubbleIntoPipeline()
20:             else
21:                 LaunchToPipeline(s, P)
22:                 InFlightBuckets.insert(k)
23:             end if
24:         else
25:             InsertBubbleIntoPipeline()
26:         end if
27:     end if
28:     AdvancePipeline()
29:     RetireCompletedBucket()
30: end for
```

Point Arithmetic. The core of our MSM datapath is a fully pipelined point addition unit based on Twisted Edwards curves [13]. Specifically, we implement a mixed-coordinate point adder, where the accumulator point is represented in extended coordinates (X_1, Y_1, Z_1, T_1) and the incoming point from the scalar multiplication is in affine coordinatesX_2, Y_2 This format enables efficient computation without the need to convert each point into the same coordinate system, significantly reducing the latency and data movement in the MSM pipeline.

We adopt the madd-2008-hwcd-3 formula set from Hisil et al. [23], which is both strongly unified and optimized for hardware. Under the assumptions $Z_2 = 1$ and $k = 2d$, the formulas compute the group addition using the formula below. These formulas require 7 multiplications (M), 1 multiplication by a constant (k), 8 additions/subtractions, and 1 doubling, assuming the second point is in affine form and the constant $k = 2d$ is precomputed.

The use of signed-digit scalars in the upstream MSM engine complements this design: since point negation on Twisted Edwards curves is trivial (a sign flip on the x-coordinate), the adder only needs to support point additions and subtractions efficiently, without branching or coordinate re-encoding.

Field Arithmetic. In our implementation, field operations over F_q are performed using 374-bit integers in Montgomery form, with the Montgomery constant $R = 2^{384}$, ensuring fast reduction using shifts. To optimize modular multiplication—which dominates performance—we adopt the Finely Integrated Operand Scanning (FIOS) algorithm, which interleaves multiplication and reduction at the finest granularity. Compared to other methods such as CIOS or SOS, FIOS introduces slightly more additions and reads, but offers greater pipelining potential for our fine-grained, hardware-oriented datapath [24].

We compute a full 374-bit × 374-bit multiplication using a composition of 48-bit × 48-bit signed base multipliers. These base multipliers are mapped to DSP blocks (e.g., 6 DSPs per 48×48 multiply), enabling highly parallelized execution. The 374-bit operands are broken into 8 limbs, aligned with the base size, and processed in a multi-level accumulation structure within the FIOS Montgomery framework. This design avoids Karatsuba [25] or Toom-Cook [26] recursion, which—although popular in software—is less hardware-friendly due to irregular data dependencies and increased control complexity. Our approach achieves deterministic latency and fits well within both ASIC and FPGA pipelines.

This format allows the downstream MSM accumulator to reuse existing points for negative digits via simple coordinate negation. In twisted Edwards affine coordinates, negation is a free operation (requiring only a sign flip of the x-coordinate). As a result, the number of buckets and additions required during MSM is reduced, improving both area and cycle efficiency.

4. Performance Evaluation

4.1. Experiment Setup

To demonstrate the universality and practicality of our ZKPU architecture, we have implemented NVMe-MSM on a Xilinx VCU118 FPGA board—representative of a high-performance PC or server environment. This setup showcases our modular MSM accelerator integrated with a custom NVMe endpoint, communicating with a host system using open-source nvme-cli [27] tools on Linux. The system is built around a RISC-V SoC generated using the Chipyard framework and runs at 100 MHz. Our configuration highlights the seamless integration of hardware acceleration

and NVMe compatibility, underscoring the accessibility and deployability of our approach on widely available infrastructure.

4.2. Experimental Results

Since the MSM algorithm itself is well-studied and not the focus of this work, our goal is not to propose a new blazingly fast algorithm, but to demonstrate a hardware-software system that is broadly compatible with modern infrastructure while still delivering performance comparable to state-of-the-art MSM implementations. Through this demo, we aim to validate the practicality, modularity, and integration capability of our NVMe-based MSM accelerator under realistic conditions.

In our setup, the **NVMe-MSM system** operates as follows.

1) The host system writes a batch of scalars and elliptic curve points (on BLS12-377) to the FPGA over the NVMe interface.
2) An `NVMe Async Event` command is issued to the device to initiate the MSM computation.
3) The MSM accelerator inside the SoC performs the computation, and upon completion, signals the host.
4) The host then issues a standard NVMe read to retrieve the resulting elliptic curve point.

To keep host-side software integration minimal and transparent, we have implemented a Python-based driver that wraps Linux's open-source `nvme-cli` commands using the `subprocess` module. This ensures compatibility with any Linux environment supporting NVMe, reinforcing the portability of our solution[1].

4.3. Experimental Results

Table 1 lists the measured MSM execution times of our NVMe-based accelerator for $n = 5$–22. Our measurements report only the hardware MSM module latency, excluding host communication or setup overhead. The results follow an almost perfect exponential scaling, well-fitted by:

$$T_{\text{ZKPU_MSM}}(n) \approx 9.26417 \times 10^{-5} \cdot (2.00835)^n \text{ ms}.$$

Comparison with CycloneMSM and gnark-crypto.. The public data for CycloneMSM (AWS F1 FPGA) and gnark-crypto (CPU) [8] are only available for $n = 22$–26. To enable comparison over the full range, we fit their results to:

$$T(n) = a \cdot 2^n + b$$

obtaining:

$$T_{\text{Cyclone}}(n) \approx 7.6864 \times 10^{-5} \cdot 2^n + 477.96 \text{ ms},$$
$$T_{\text{gnark}}(n) \approx 2.12185 \times 10^{-3} \cdot 2^n + 967.08 \text{ ms}.$$

1. A video demonstration of our NVMe-MSM system is available in [28].

TABLE 1. MEASURED MSM TIMES FOR OUR NVME-BASED IMPLEMENTATION (EVALUATED FROM SIMULATION).

n	ZKPU MSM (ms)	n	ZKPU MSM (ms)
5	0.0030	14	1.6333
6	0.0057	15	3.2371
7	0.0123	16	6.5438
8	0.0247	17	13.0650
9	0.0502	18	26.0817
10	0.1009	19	52.1813
11	0.2014	20	104.5091
12	0.4095	21	209.0369
13	0.8184	22	418.0618

The constants $b \approx 478\,\text{ms}$ (CycloneMSM) and $b \approx 967\,\text{ms}$ (gnark-crypto) indicate fixed costs such as host–device communication and setup overhead, which are absent from our reported numbers.

Overhead removal and algorithmic scaling. By subtracting these constants:

$$T'_{\text{Cyclone}}(n) = T_{\text{Cyclone}}(n) - 477.96,$$
$$T'_{\text{gnark}}(n) = T_{\text{gnark}}(n) - 967.08,$$

Table 2 presents the adjusted MSM execution times for ZKPU_MSM, CycloneMSM, and gnark-crypto after removing the fixed overhead term b from the latter two implementations. The adjustment reveals the pure MSM kernel performance, allowing a fair, algorithmic-level comparison. Across all n from 2^5 to 2^{26}, ZKPU_MSM maintains execution times that are more than an order of magnitude faster than the CPU-based gnark-crypto, with the gap widening significantly at larger n. Compared to CycloneMSM, ZKPU_MSM shows competitive performance throughout, with near-matching runtimes for small and mid-range n, and a modest slowdown at the largest problem sizes. This difference stems from our non-recursive ASIC design choice, which increases DSP usage and pipeline length to improve design robustness and reduce error-prone complexity, at the cost of a small performance trade-off. Importantly, ZKPU_MSM's native NVMe interface allows it to integrate directly into standard server and edge platforms without requiring dedicated PCIe accelerator slots, offering deployment flexibility that is absent in traditional FPGA card solutions.

Figure 4 visualizes these results on a logarithmic scale, highlighting the scaling trends after fixed-overhead removal. The three curves exhibit nearly parallel slopes, indicating similar exponential scaling behavior in their MSM kernels. ZKPU_MSM consistently tracks close to CycloneMSM across the entire range while staying far below the gnark-crypto curve, reinforcing the substantial efficiency advantage of hardware acceleration over CPU execution. The parallelism of the curves also validates that the algorithmic scaling is preserved despite architectural differences, and the small vertical offset between ZKPU_MSM and CycloneMSM reflects the aforementioned design trade-offs. This visualization underscores that ZKPU_MSM delivers state-of-the-art MSM performance while preserving broad system compatibility through NVMe integration.

TABLE 2. ADJUSTED MSM TIMES (MS) WITH FIXED b REMOVED FROM CYCLONEMSM AND GNARK-CRYPTO FITS.

n	ZKPU_MSM (ms)	CycloneMSM adj. (ms)	gnark-crypto adj. (ms)
5	0.0030	0.0025	0.0679
6	0.0061	0.0049	0.1358
7	0.0122	0.0098	0.2716
8	0.0246	0.0197	0.5432
9	0.0493	0.0394	1.0864
10	0.0991	0.0787	2.1728
11	0.1989	0.1574	4.3456
12	0.3994	0.3148	8.6911
13	0.8019	0.6297	17.3822
14	1.6102	1.2593	34.7644
15	3.2330	2.5187	69.5289
16	6.4914	5.0374	139.0578
17	13.0337	10.0747	278.1155
18	26.1698	20.1494	556.2310
19	52.5450	40.2988	1112.4620
20	105.5028	80.5976	2224.9241
21	211.8341	161.1953	4449.8481
22	425.3318	322.3906	8899.6962
23	854.0041	644.7812	17799.3925
24	1714.7152	1289.5624	35598.7849
25	3442.8970	2579.1247	71197.5699
26	6912.8332	5158.2495	142395.1397

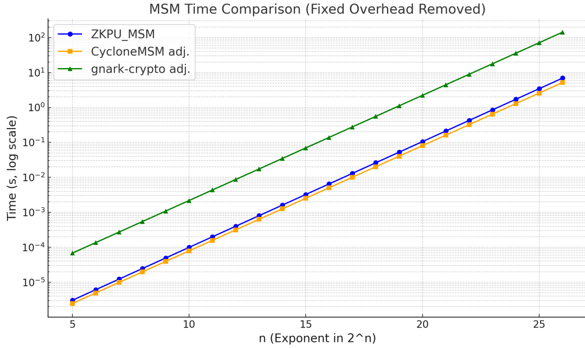

Figure 4. MSM time comparison after removing fixed overhead b from CycloneMSM and gnark-crypto fits (log scale).

5. Discussion

5.1. Flash Memory Integration for Precomputation

One of the most pressing scalability challenges in large-scale MSM arises from the memory overhead associated with storing precomputed elliptic curve points. As discussed in prior work such as GZKP [9], the storage footprint for precomputed tables reaches over 5 GB at an MSM size of 2^{21}, and grows further with larger workloads. This imposes significant constraints on current FPGA- and ASIC-based MSM accelerators, which typically rely on limited on-chip SRAM or expensive external DRAM.

A promising direction to address this challenge is the *integration of non-volatile flash memory directly onto the MSM accelerator chip*. This would enable persistent, high-capacity storage of precomputed points, eliminating the need to reload large tables between runs. Flash is especially well-suited for this task given the read-only, structured nature of MSM memory access patterns.

Crucially, this direction aligns naturally with our architecture's reliance on the NVMe protocol, which was originally developed to expose the performance of flash storage through a standard high-throughput interface. In modern data centers, NVMe is synonymous with flash, and by designing our accelerator to speak NVMe natively, we lay the groundwork for future MSM chips to integrate flash directly under the same protocol. This would unify the data movement and storage layers under a single, flash-native interface, simplifying system integration and improving overall efficiency.

5.2. NVMe-Based Peer-to-Peer Communication

As Zero-Knowledge (ZK) proof systems evolve toward larger, recursive, and more parallelizable constructions, the demand for high-throughput inter-device communication becomes critical. One promising direction is to leverage NVMe's support for peer-to-peer (P2P) communication, which allows devices to exchange data directly across the PCIe fabric [29] without routing through the host CPU. This functionality, already exploited in technologies such as NVIDIA's GPUDirect, opens the door to new architectural models for distributed ZK systems.

A promising direction is the development of ZKLink, an NVMe-based protocol that could enable direct, low-latency communication between ZKPUs, GPUs, and other accelerators. Rather than treating each accelerator as a passive endpoint managed entirely by the CPU, ZKLink establishes a peer fabric where computational devices can coordinate sub-tasks autonomously. This is especially useful in recursive ZK proof architectures (e.g., Halo 2 [30]), where a large proof is decomposed into smaller sub-proofs that must be generated and later aggregated with minimal overhead. Fast, reliable P2P communication is essential to achieve this at scale.

6. Conclusion

Multi-Scalar Multiplication (MSM) remains the dominant computational bottleneck in generating pairing-based zero-knowledge proofs. While prior work has focused on algorithmic improvements, this paper addresses a complementary challenge: architectural integration and system-level compatibility. We present a modular, infrastructure-friendly ZK accelerator built as a self-contained RISC-V SoC and integrated under a standard NVMe interface. Our design integrates a FIOS-based Montgomery multiplier, signed-digit scalar encoding, a pipelined Twisted Edwards adder, and a hazard-aware scheduler into a unified datapath deployable via standard tools like nvme-cli. Our FPGA prototype demonstrates performance on par with leading MSM implementations, validating NVMe-based ZK acceleration as a practical and scalable direction. Looking ahead, we plan

to extend our architecture with embedded flash for persistent precomputations and support for peer-to-peer NVMe communication, enabling fully on-chip, distributed proof generation. Our work lays the foundation for cloud-native, modular, and production-ready ZK hardware systems.

Acknowledgment

We thank the reviewers for their insightful and constructive feedback, which has strengthened this paper.

References

[1] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, February 1989.

[2] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct {Non-Interactive} zero knowledge for a von neumann architecture. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 781–796, 2014.

[3] Introduction to zk-rollups. https://ethereum.org/developers/docs/scaling/zk-rollups/, 2025.

[4] Circle Internet Financial, LLC. Episode 12: Financial privacy, digital currency and stablecoin payments. https://www.circle.com/the-money-movement/episode-12-financial-private-digital-currency-and-stablecoin-payments, 2020.

[5] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. Cryptology ePrint Archive, Paper 2019/1047, 2019.

[6] Ye Zhang, Shuo Wang, Xian Zhang, Jiangbin Dong, Xingzhong Mao, Fan Long, Cong Wang, Dong Zhou, Mingyu Gao, and Guangyu Sun. Pipezk: Accelerating zero-knowledge proof with a pipelined architecture. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 416–428, 2021.

[7] Shahzad Ahmad Butt, Benjamin Reynolds, Veeraraghavan Ramamurthy, Xiao Xiao, Pohrong Chu, Setareh Sharifian, Sergey Gribok, and Bogdan Pasca. if-zkp: Intel fpga-based acceleration of zero knowledge proofs. In *2024 IEEE 32nd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 212–212, 2024.

[8] Kaveh Aasaraai, Don Beaver, Emanuele Cesena, Rahul Maganti, Nicolas Stalder, and Javier Varela. FPGA acceleration of multi-scalar multiplication: CycloneMSM. Cryptology ePrint Archive, Paper 2022/1396, 2022.

[9] Weiliang Ma, Qian Xiong, Xuanhua Shi, Xiaosong Ma, Hai Jin, Haozhao Kuang, Mingyu Gao, Ye Zhang, Haichen Shen, and Weifang Hu. Gzkp: A gpu accelerated zero-knowledge proof system. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS 2023, page 340–353, New York, NY, USA, 2023. Association for Computing Machinery.

[10] Scroll.io. The anatomy of proof generation. https://scroll.io/blog/proof-generation, December 2022.

[11] RISC Zero. Universal zero knowledge – risc zero. https://risczero.com/, 2025.

[12] Scroll.io. Scroll – zkevm ethereum layer2 rollup. https://scroll.io/, 2025.

[13] Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson. Twisted edwards curves revisited. Cryptology ePrint Archive, Paper 2008/522, 2008.

[14] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, 2014.

[15] StarkWare Industries. Starknet: The l2 validity rollup scaling ethereum. https://starkware.co/starknet/, 2025. StarkNet is a permissionless, Ethereum L2 scaling solution using off-chain STARK proofs.

[16] Nicholas Pippenger. On the evaluation of powers and related problems. In *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, SFCS '76, page 258–263, USA, 1976. IEEE Computer Society.

[17] Andrew Waterman. *Design of the RISC-V Instruction Set Architecture*. PhD thesis, EECS Department, University of California, Berkeley, Jan 2016.

[18] C. Kaya Koc, T. Acar, and B.S. Kaliski. Analyzing and comparing montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996.

[19] Berkeley Architecture Research and Chipyard Contributors. *Chipyard Documentation (latest)*. Chipyard Project, 2025.

[20] SiFive, Inc. TileLink Specification, Version 1.7-draft. Technical Report 1.7-draft, SiFive, Inc., San Francisco, CA, August 2017.

[21] OpenProof. Nvme-msm: Demo of nvme-accelerated multi-scalar multiplication. https://docs.open-proof.com/demo/nvme-msm, July 2025.

[22] Hardcaml Team. Msm overview — hardcaml zprize. https://zprize.hardcaml.com/msm-overview, 2022.

[23] dodvlad Mihăilescu and Tim Sheerman-Chase and Craig Costello and Patrick Longa. "madd-2008-hwcd-3: Mixed addition on twisted edwards extended coordinates". https://hyperelliptic.org/EFD/g1p/auto-twisted-extended-1.html#addition-madd-2008-hwcd-3, 2008.

[24] C Kaya Koc, Tolga Acar, and Burton S Kaliski. Analyzing and comparing montgomery multiplication algorithms. *IEEE micro*, 16(3):26–33, 1996.

[25] Brilliant.org. Karatsuba algorithm. https://brilliant.org/wiki/karatsuba-algorithm/, 2025.

[26] Stephen A. Cook. On the time–space tradeoff for sorting. Technical report, Technical Report, University of Toronto, 1966.

[27] linux-nvme. nvme-cli: NVMe management command line interface. https://github.com/linux-nvme/nvme-cli, 2025.

[28] OpenProof. NVMe-MSM: Demo of nvme-accelerated multi-scalar multiplication. https://docs.open-proof.com/demo/nvme-msm#benchmark, July 2025.

[29] Fabric Cryptography. Fabric cryptography– verifiable processing unit for next-gen cryptography. https://www.fabriccryptography.com/, 2025.

[30] Electric Coin Company. Ecc releases code for halo 2. https://electriccoin.co/blog/ecc-releases-code-for-halo-2/, September 2020.