ATTINA: Spatial Acceleration of *Additive* Number Theoretic Transform for Energy-Efficient zkSNARKs at the Edge

Yanze Wu
George Mason University
Fairfax, Virginia, USA
ywu42@gmu.edu

Devin Park
George Mason University
Fairfax, Virginia, USA
dpark35@gmu.edu

Md Tanvir Arafin
George Mason University
Fairfax, Virginia, USA
marafin@gmu.edu

ABSTRACT

Modern zero-knowledge proof (ZKP) schemes are moving from costly elliptic curve-based systems towards algebraic proof systems. This transition is ushering in a new era of broad adoption of hardware-friendly ZKPs, implementable even on resource-constrained edge devices. Algebraic structures, such as towers of binary fields, are core components of such schemes, and they require hardware and algorithmic innovation to realize advanced ZKPs, such as Binius and Binius-FRI, efficiently.

Hence, this paper introduces the ATTINA framework, a high-throughput additive number theoretic transform (ANTT) accelerator operating over extended binary fields. ANTT is the most computationally intensive task for realizing Binius. ATTINA partitions ANTT tasks into multiple subtasks and strategically allocates them to processing elements (PEs) and hardware programmable logic (PL) kernels in an interleaved computing pattern. ATTINA features a scalable architecture that enables dynamic parameter changes for different cryptographic applications. To understand its applicability, this paper implements ATTINA on an edge-deployable heterogeneous Versal adaptive system-on-chip (ASoC) platform. ATTINA on Versal ASoC maintains a throughput of 10 Gb/s. ATTINA outperforms a benchmark CPU implementation of ANTT on a high-end processor (i9-14900K) by **139**× and a PE-only implementation by **38**× in terms of latency while operating at ≤5W for a 4096-point ANTT. ATTINA's code is published at https://github.com/SPIRE-GMU/binius-spire for evaluation and reproducible research.

KEYWORDS

Binius, Additive Number Theoretic Transform, Coarse-grained reconfigurable architecture (CGRA), Heterogeneous Architecture.

ACM Reference Format:
Yanze Wu, Devin Park, and Md Tanvir Arafin. 2026. ATTINA: Spatial Acceleration of *Additive* Number Theoretic Transform for Energy-Efficient zkSNARKs at the Edge. In *63rd ACM/IEEE Design Automation Conference (DAC '26), July 26–29, 2026, Long Beach, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770743.3804075

1 INTRODUCTION

Zero-knowledge proofs (ZKP) represent a privacy-preserving paradigm that allows non-interactive authentication, *i.e.*, via mathematical statements, rather than the disclosure of personal information. The hidden privacy capability makes ZKPs key cryptographic components in multiple applications [6, 22]. Among recent variants of ZKP, succinct non-interactive arguments of knowledge (SNARK) [8] are popular due to their small proof size, fast verification, and viable scalability. However, the traditional elliptic curve-based proof involves multi-scalar multiplication on large integers, *e.g.*, 256 bits, limiting the performance of the protocols in practice.

Scalable transparent argument of knowledge (STARK) makes a significant improvement upon SNARK by replacing elliptic curve-based proof with hash-based ZKP protocols [25, 26, 31, 34]. Hence, there is a shift in the ZKP domain from large integers to smaller sizes, *e.g.*, 64-bit integers in the Goldilocks field [24], and even 31-bit in Mersenne31 [14] and Babybear[23]. Such improvements enable more efficient computation under typical computing architectures. For instance, Plonky2 employs Goldilocks and achieves hundreds of times the speedup compared to the prior ZKP system [12].

Interestingly, new protocols such as Binius [9, 10] follow this trend and push the blind proof to a smaller binary field, where the proof systems operate directly over zeros and ones. The arithmetic in the binary field directly maps to Boolean logic, *e.g.*, addition is equivalent to the XOR gate, and multiplication to AND gates, which fundamentally differs from computations in other fields. The hardware-friendly property offers Binius further efficiency in computing. However, given the foundation of the specific data construction, the core components of Binius, *i.e.*, IANTT and ANTT, consume the majority of the time, as shown in the benchmark of the Binius model Fig.1. The development of specialized hardware accelerators for this transform remains little investigated in the current literature, which presents a fundamental gap in this field.

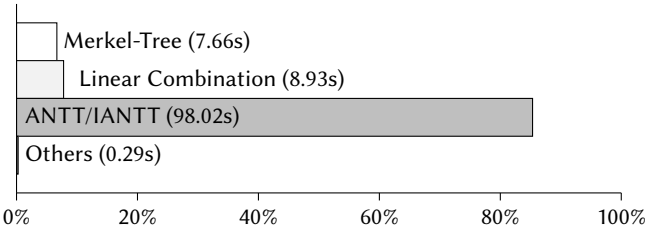


Figure 1: Compute-time profile for the Binius implementation by Vitalik et al. [1, 7].

Recently, the increasing demand for computation in artificial intelligence (AI) workloads has driven significant development of energy-efficient hardware design at the edge. For example, specialized accelerators such as Google's Edge TPU [17], NVIDIA's Jetson platform [15], Tenstorrent's GraySkull [29], and Apple's Neural Engine [13] are proposed for high-performance AI computation with minimal power consumption. Among them, Tenstorrent's

Please use nonacm option or ACM Engage class to enable CC licenses
This work is licensed under a Creative Commons Attribution 4.0 International License.
DAC '26, July 26–29, 2026, Long Beach, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804075

Authors Copy

GraySkull and AMD-Xilinx's Versal [3] feature processor array-based designs, *i.e.*, two-dimensional processing elements (PE) on the chip. Such a coarse-grained reconfigurable architecture (CGRA) not only meets the high-throughput demands of AI tasks but also supports computation-intensive cryptographic applications.

Therefore, in this paper, we investigate Xilinx's Versal devices and construct a high-throughput additive NTT accelerator, titled ATTINA. We develop software kernels for PEs and hardware kernels that run on the programmable logic (PL) domain. The main contributions of this work are given below.

C1. Based on the extended binary fields constructed in the Binius protocol, we propose ATTINA, a dedicated ANTT accelerator on a heterogeneous reconfigurable architecture. The proposed design partitions ANTT into two parts, *i.e.*, computation and permutation, which are accelerated on the PL and PE sides, respectively.
C2. On the PE side, we introduce the permutation unit to rearrange the data stream between butterfly operation stages. These permutation kernels are mapped into the PE array of the Versal devices, known as the adaptive intelligent engine (AIE) domain.
C3. On the PL side, we designed specific multipliers for field elements in a binary field tower. Then, we integrate the components on the PL and PE sides to deliver the full ATTINA design.
C4. We evaluate ATTINA in terms of throughput and scalability. We find that ATTINA is capable of maintaining a throughput of up to 10 Gb/s for a continuous additive NTT process. In addition, the architectural flexibility enables ATTINA to be extended to an arbitrary length.
C5. We open-source our codes at **https://github.com/SPIRE-GMU/binius-spire** for artifact evaluation and reproducible research.

2 PRELIMINARY

2.1 Binary Field and Extension

Polynomials are key components in cryptography and are typically defined over a finite field $\mathbb{F}_p$. Specifically, Binius operates over a binary finite field with $p = 2$, and extends the field for m times ($m \geq 1$), to generate field $\mathbb{F}_{2^{2^m}}$ by adding conjugates. For instance, $\mathbb{F}_{2^2}$ is constructed from $\mathbb{F}_2$ by adjoining a root x of the irreducible polynomial $x^2 = x + 1$, which yields the extended field elements from $\{0, 1\}$ to $\{0, 1, x, x + 1\}$.

To operate on elements within larger finite fields, Binius constructs a tower of field extensions defined recursively by:

$$\tau_m = \tau_{m-1}[X_{m-1}]/(X_{m-1}^2 + X_{m-1}X_{m-2} + 1) \tag{1}$$

$$\tau_0 = \mathbb{F}_2 \tag{2}$$

, where each successive extension introduces a new layer to the existing field, building hierarchically upon the previous field. An intuitive diagram of the tower is shown in Fig.2, which enables the representation and manipulation of elements with arbitrary length.

$$\begin{aligned}
\tau_m &= \tau_{m-1}[X_{m-1}]/(X_{m-1}^2 + X_{m-1}X_{m-2} + 1)\\
&\vdots\\
\tau_{m-1} &= \tau_{m-2}[X_{m-2}]/(X_{m-2}^2 + X_{m-2}X_{m-3} + 1)\\
&\vdots\\
\tau_1 &= \tau_0[X_0]/(X_0^2 + X_0 + 1)\\
&\vdots\\
\tau_0 &= \mathbb{F}_2
\end{aligned}$$

Figure 2: Visualizing Binary Towers as a sequence of field extensions starting from base field $\mathbb{F}_2$.

The binary tower field offers significant hardware advantages for mathematical operations, particularly for adding numbers of arbitrary bit width. Any bit position in a binary tower field implies that $1 + 1 = 0$ (in fields of characteristic 2) rather than being carried over as conventional digital circuits do. In other words, binary-tower representation reduces addition to bitwise XOR, *i.e.*,

$$Add(A, B) = A \oplus B \tag{3}$$

Multiplication in a binary field proceeds through a recursive process. To multiply elements $x * y$, each number is split into two halves $x = L_x + R_x * X_k$, and $y = L_y + R_y * X_k$. Then multiply as:

$$\begin{aligned}
x * y &= (L_x + R_x * X_k) * (L_y + R_y * X_k)\\
&= (L_x * L_y) + (L_x * R_y) * X_k +\\
&\quad (R_x * L_y) * X_k + (R_x * R_y) * X_k^2
\end{aligned} \tag{4}$$

, where the last item is reduced within the construction rule ($X_k^2 = X_k X_{k-1} + 1$).

2.2 Additive NTT

Extended binary fields are vital in the encoding and decoding of the Reed-Solomon extension (RSE), and Lin *et al.* [19] propose ANTT to speed up the RSE process. ANTT is defined as the evaluation of a polynomial $P(x) \in \mathbb{F}_{2^n}[x]/(x^{2^m} - x), m \leq n$, over an additive subgroup coset $\{w_i + a\}_{i=0}^{2^m-1}$, where $a \in \mathbb{F}_{2^n}$ and $\{w_i\}_{i=0}^{2^m-1}$ is an additive subgroup of $\mathbb{F}_{2^n}$. Since $\{w_i + a\}_{i=0}^{2^m-1}$ is a coset of the additive subgroup $\{w_i\}_{i=0}^{2^m-1}$ and is in $\mathbb{F}_{2^n}$, a new polynomial $\hat{P}(x) \in \mathbb{F}_{2^n}[x]/x^{2^n} - x$ can be constructed by padding zero to $P(x)$ and then evaluate it over $\mathbb{F}_{2^n}$ [11].

Algorithm 1 presents the procedure of ANTT. It starts with a polynomial $P(x) = \sum_{i=0}^{2^m-1} p_i X_i(x), 1 \leq m \leq n$ expressed under a set of *novel* polynomial bases $\{X_i(x)\}_{i=0}^{2^n-1}$. The *novel* basis $X_i(x)$ is defined as

$$X_i(x) = \prod_{j=0}^{n-1} \left(\frac{s_j(x)}{s_j(w_{2^j})}\right)^{i_j} \tag{5}$$

, where i_j implies the $j-$th bit of i, and $\{s_j(x)\}_{j=0}^{n-1}$ are defined as subgroup vanishing polynomials

$$s_j(x) = \prod_{i=0}^{2^j-1} x + w_i \tag{6}$$

, which obey the polynomial linearity $s_j(x+y) = s_j(x) + s_j(y)$ [19].

Authors Copy

Algorithm 1 Additive Number Theoretic Transform

procedure ANTT($P(x), a, m$)
input: $P(x) = \sum_{i=0}^{2^m-1} p_i X_i(x)$, $a, p_i \in \mathbb{F}_{2^n}, m \leq n$
output: $[P(w_0 + a), P(w_1 + a), ..., P(w_{2^m-1}+a)]$
if m=0 **then**
  return p_0
end if
// Split the polynomial
$\tau \leftarrow 2^{m-1}$
$P_0(x) \leftarrow \sum_{i=0}^{\tau-1} p_i X_i(x)$
$P_1(x) \leftarrow \sum_{i=0}^{\tau-1} p_{\tau+i} X_i(x)$
// Butterfly operation
$Q_0(x) \leftarrow P_0(x) + \frac{s_{m-1}(a)}{s_{m-1}(w_\tau)} P_1(x)$
$Q_1(x) \leftarrow Q_0(x) + P_1(x)$
// Recursively compute with a smaller scale polynomial
return concatenate (ANTT($Q_0(x), a, m-1$), ANTT($Q_1(x), w_\tau + a, m-1$))

By recursively decomposing the *novel* basis following

$$X_{\tau+i}(x) = \frac{s_{k-1}(x)}{s_{k-1}(w_\tau)} X_i(x) \tag{7}$$

, ANTT eventually evaluates the polynomial $P(x)$ of size 2^m, over the additive coset $\{w_i + a\}_{i=0}^{2^m-1}$. Inverse ANTT (IANTT) is a symmetric process of ANTT that constructs the polynomial of the *novel* basis from the evaluation points. ANTT and IANTT form an efficient approach that supports RSE operation on a characteristic-2 finite field with complexity $O(n \cdot log(n))$.

ANTT versus classical NTT. ANTT and NTT operate over distinct fields. ANTT is constructed over an extended binary field $\mathbb{F}_{2^{2^m}}$, whereas the NTT typically operates over a finite field $\mathbb{F}_p$. Although they share a similar computing pattern, *i.e.*, the butterfly unit, their handling of operands differs. For example, the butterfly unit in ANTT is calculated directly with *gate logic*, in contrast to the NTT, which is computed with modular operations. This work implements IANTT as an example, but the methodology is equally applicable to ANTT, as they are symmetrical processes with identical computational complexity, and the terms will be used interchangeably.

2.3 Adaptive System on Chip

ATTINA works on AMD-Xilinx's Versal series of hardware, which is a heterogeneous computing platform composed of three computing elements: (1) processing elements, *i.e.*, PEs in the AI engine (AIE) domain, (2) the processing system (PS) domain, and (3) the programmable logic (PL) domain. This architecture paradigm distinguishes itself from the conventional FPGA architecture (PS + PL) by introducing a two-dimensional array of 400 programmable processing elements (called PEs in the rest), as depicted in Fig. 3.

The AIE domain represents a significant advancement in heterogeneous computing, where each PE features a 36 KB dedicated data memory, 16 KB programmable memory, and a 7-way very long instruction word (VLIW) supporting two read, two move, one write, scalar, and vector execution concurrently. The interconnect capabilities of a PE include direct memory access (DMA) to adjacent PEs (in four directions), dual 32-bit AXI-stream interfaces for

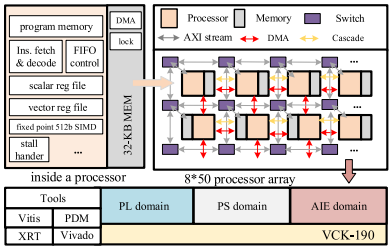


Figure 3: Overview of target platform VCK-190, which consists of AI engine (AIE), processing system (PS), and programmable logic (PL) domains.

non-adjacent communication, and high-bandwidth 384-bit cascade channels for data streams.

3 DESIGN OF ATTINA: A SCALABLE ANTT ON HETEROGENEOUS ARCHITECTURE

In this section, we implement scalable ANTT/IANTT with the following steps. First, we derive a data dependence graph (DG) by analyzing the ANTT process. Then, a signal flow graph (SFG) is orchestrated from the DG, which demonstrates PE and PL kernels and the flow of the data stream. After that, we optimize the PE and PL kernels on both the hardware and the software sides. Finally, we map and integrate the PE and PL kernels on ASoC to deliver the complete ATTINA framework.

3.1 Data Flow Analysis

Alg.1 outlines the overall procedure of ANTT, while several implementation details warrant further discussion. Fig. 4 (a) depicts the data dependence graph of IANTT as a representative instance, which allows a clearer illustration. The first stage processes the input coefficients $(p_1, p_2, ..., p_n)$ using the minimal two-point butterfly unit (BU). Each subsequent stage doubles the BU size, *e.g.*, stage two utilizes a four-point BU, *etc.*

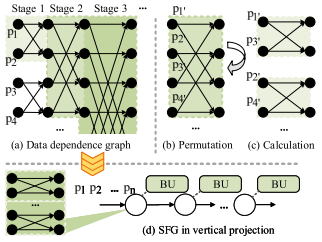


Figure 4: Data dependence graph. (a) The overview of the procedure can be considered as a series of stacked butterfly units across stages. The butterfly calculation is further split into permutation (b) and calculation (c). The DG is projected to SFG (d) in vertical projection.

Various sizes of BUs essentially share the same logic across stages. As an illustrative case, a four-point BU (Fig. 4.b) is transformed into two two-point BUs (Fig. 4.c) through an appropriate permutation. The transformation indicates the inherent property of ANTT, that it is not solely a computational task but rather a dual problem of permutation and calculation.

Authors Copy

Algorithm 2 Pseudo-code for Naive Permutation

```
procedure PERMUTATION(S_out, S_in)
while 1 do
    // Load one data point per cycle
    int32 a[i],b[i] ← Data_loading_from_FIFO(S_in);
    for i =0; i<n/2; i++ do
        // Point-wise permutation
        uint16_t la = a[i] & 0xFFFF;
        uint16_t lb = b[i] & 0xFFFF;
        uint16_t hb = (b[i] >> 16) & 0xFFFF;
        a[i] = (a[i] & 0xFFFF0000) | hb;
        b[i] = ((uint32_t)la << 16) | lb;
    end for
    for i =0; i<n/2; i++ do
        Data_squeeze_out(S_out,a[i],b[i]);
    end for
end while
```

3.2 Orchestrating the ATTINA Wavefront Architecture

The DG reveals spatial relationships among data points along a vertical direction, representing a stage-wise aggregation. And a SFG is derived by projecting DG along the vertical direction, as depicted in Fig. 4 (d). The SFG is similar to the single-path delay feedback (SDF) architecture or multipath delay commutator (MDC) architecture in other literature [16, 32], as shown in Fig. 5 (a), where BU represents the execution of butterfly operation with varying sizes and MEM is the dual port memory bank.

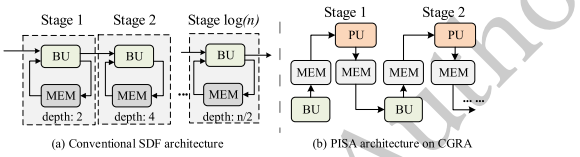


Figure 5: The development from conventional SDF to ATTINA, a wavefront architecture. Green rectangles are butterfly units and orange rectangles are permutation units. Memory (MEM) blocks function as a cache between kernels.

As mentioned in Section 3.1, ANTT can be further partitioned into computation and permutation problems. However, the classic SDF focuses on the pipelined structure but ignores the bottleneck in memory access, rendering it inefficient in large-scale NTT.

In contrast, the ATTINA architecture differs from the classical SDF by introducing permutation units (PU) between BU. On the one hand, a more efficient strategy can be employed to rearrange the data. On the other hand, it alleviates the burden of BU on address selection and enables smooth data access. As a result, the data stream goes as a wavefront across these processing kernels.

3.3 Strategy for Kernel Mapping

Both the PL and AIE domains can implement wavefront architecture, but they have distinct characteristics: the PL constructs circuits using Boolean logic, while kernels in the AIE domain execute tasks through instructions, similar to a conventional processor. Therefore, where to implement the kernels depends on the properties of the tasks.

Algorithm 3 Optimized Permutation on AIE

```
// Extract even-index elements
filter_even (v64int16 in):
    return v32int16 out = {in[0], in[2], ..., in[62]};
// Interleave two vectors
interleave(v32int16 a, v32int16 b):
    return v64int16 out = {a[0], b[0], a[1], b[1], ...};
procedure PERMUTATION((S_out, S_in):)
while 1 do
    // Loading data stream to vector from the previous PE
    v64int16 a1,b1 ← Vector_data_loading_from_FIFO(S_in)
    // Pipelined vector operation of filter & interleave
    v32int16 a1_high = filter_even(a1)
    v32int16 a1_low = filter_even(a1)
    v32int16 b1_high = filter_even(b1)
    v32int16 b1_low = filter_even(b1)
    v64int16 r1 = interleave(a1_high, b1_high)
    v64int16 r2 = interleave(a1_low, b1_low)
    Vector_data_squeeze_out(S_out, r1, r2)
end while
```

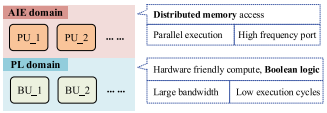


Figure 6: Mapping SFG to ASoC. Two types of kernels, *i.e.*, BU and PU, are allocated to different domains.

Butterfly units are preferably placed in the PL domain, as illustrated in Fig. 6. The primary rationale is that ANTT operates in binary field arithmetic, where mathematical operations are aligned to gate-level logic. Meanwhile, multiplication follows a recursive structure, as shown in Eq.4, which benefits from pipelining optimizations in hardware. Permutation units are initialized in the AIE domain due to their distributed memory architecture, *i.e.*, each processing element is equipped with a dedicated 36KB memory bank. Another advantage lies in the SIMD technique, which enables PU to shuffle multiple data points in parallel, whereas address rearrangement in the PL domain is built on dual-port memory banks, which only supports limited data access per cycle.

3.4 PL Kernel Optimization

ATTINA develops pipelined **BU kernels** on **PL domain** for binary tower field arithmetic as shown in Fig.7. This butterfly is inspired by the Open Binius implementation[28]. As mentioned, multiplication in the binary tower employs a recursive decomposition, *i.e.*, a 16-bit operand is divided into two 8-bit segments, which are further partitioned into smaller bit-width components, as shown in Eq.4.

Therefore, the multiplication process is implemented using a pipelined architecture, where each pipeline stage handles the decomposition of one-level operands. This approach enables high-throughput multiplication when handling the data stream. Noting

Authors Copy

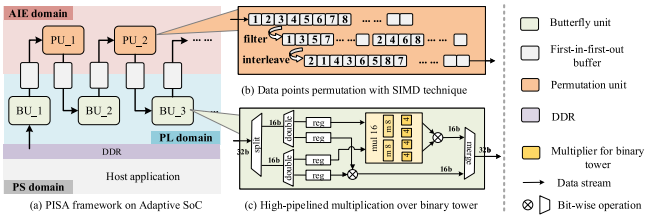


Figure 7: The overview of the ATTINA framework. (a) The data stream is loaded from a real-time processing unit and flows across different processing elements, *i.e.*, BU in the PL domain and PU in the AIE domain, respectively. (b) Detailed data shuffle procedure in the permutation unit. (c) High-pipelined multiplier over binary tower in butterfly unit.

that the frequency of the multiplication primitives in [28] is 880 MHz, whereas BU kernels in ATTINA are constrained at around 320 MHz to match the throughput of PU kernels in AIE.

3.5 AIE Kernel Optimization

The **permutation unit (PU) kernels** are realized on the **AIE domain**, as shown in Fig.7. This kernel reorganize the data streams and push them to the PL domain through a 32-bit AXI-stream channel. Alg.2 illustrates a baseline permutation for n data points in the $log_2(n)$-th stage of processing. The rearrangement goes in three phases: (1) initial data loading (Alg.2, line 3) where all data points are stored in memory; (2) permutation through bit decomposition and splicing (Alg.2, line 4-10); (3) squeezes out the reordered data stream Alg.2, line 12). The reference Binius implementation [1] operates on 16-bit operands, while the minimal AXI interface width is 32 bits. Hence, the PU kernel accesses two data points per cycle to fully utilize the data bus.

Two approaches are employed to further enhance the throughput of PU kernels: pipelining and single-instruction, multiple-data (SIMD) processing. **Pipelining.** By properly crafting the instructions, the PE executes the instructions smoothly [4]. A key difference between the naive and optimized permutations is instruction efficiency. The naive implementation relies on complex bitwise operations (*e.g.*, Alg.2 lines 6-9), which compile into an inefficient sequence of instructions and cause pipeline stalls during execution. In contrast, the optimized kernel utilizes well-organized built-in instructions to execute with high efficiency. **SIMD.** Moreover, it supports vector-wise operation rather than point-wise operation, which significantly improves the throughput of the kernel.

Table 1: Resource utilization for ATTINA on VCK-190.

Kernel	Freq. (MHz)	PL domain FF/LUT/BRAM/DSP	AIE domain PE
BU	320	485/2083/1/0	-
PU	1250	-	1
ATTINA	320	11642/10290/3/0	12

Integrating the components together, we present the ATTINA framework, an optimized ANTT accelerator illustrated in Fig. 7. The ATTINA framework systematically presents a development flow of the ANTT, consisting of four phases: algorithmic formulation and DG analysis, orchestrate wavefront architecture, mapping kernels to ASoC, and kernel optimization.

4 EXPERIMENTAL RESULTS

The experiments for ATTINA are conducted on an Intel(R) Core(TM) i9-14900K and a VCK-190 ASoC. The development tool is Vitis 2025.1 on Ubuntu 24.04.



Figure 8: Layout of the proposed 4096-point additive NTT on the platform. The PEs are connected to a FIFO via a reconfigurable network.

4.1 Resource Utilization

Table 1 lists the resource utilization of kernels on ASoC. Each BU requires little hardware resource as the butterfly operation in ANTT is gate-level logic rather than modulo reduction, as conventional NTT does. Moreover, no DSP is needed, as it only evolves bitwise operation to perform multiplication over binary tower. ATTINA employs 12 butterfly units and 12 permutation units for a 4096-point additive NTT implementation. The layout of ATTINA on the VCK-190 platform is shown in Fig.8.

4.2 Power Analysis

We evaluate the power consumption of ATTINA using Power Design Manager [5] on Versal VCK-190 and provide detailed static and dynamic measurements. The measurements span PL, AI engine, power management controller, and network-on-chip (NoC). The total power consumption is 8.353 W, comprising static and dynamic power of 4.406 W and 3.947 W, respectively. The PL domain dominates the static power at 3.148 W, while AIE and NoC account for the dynamic power at 2.284 W and 1.133 W.

4.3 Performance & Discussion

We first profile the core of the butterfly computation, *i.e.*, 16-bit multiplications on the extended binary fields, as listed in Table 4. Despite operating at a relatively lower frequency, the PL kernel achieves the highest throughput due to its efficient Boolean operation, which outperforms both the CPU and PE implementations. The ATTINA architecture copes with the data stream (a batch of 4096 points) that sequentially crosses stage 0 to subsequent kernels, with the 32-bit AXI in/out ports. The measurement results show that each kernel maintains a consistent processing latency of 6.5 μs, in a throughput of 10 Gb/s.

$$T_{antt} = \frac{points \times width}{latency} = 10.08\ Gb/s \tag{8}$$

Since it is a continuous data stream, the throughput of ATTINA is equivalent to $T_{antt} = Freq. \times width$. The final performance is bounded by the frequency of the PL kernel and the width of the AXI-stream interface, which together determine the maximum theoretical data throughput that the system can achieve.

Authors Copy

Table 2: Comparison of the proposed architecture with other state-of-the-art works on FPGA.

Reference	Platform	Input Size		Area			Timing			Throughput
		Length	Width	LUT	FF	BRAM	Freq.(MHz)	Cycles	Latency (μs)	(Gb/s)
ISCAS'24[27]	Zynq-UltraScale+	1024	17	15834	1147	-	150	320	2.13	8.16
VLSI'24[18]	Kintex-UltraScale	1024	14	22648	15030	24	200	343	1.72	8.34
TCAS-II'22[33]	Virtex-7	1024	14	2151	1475	2.5	232	2104	4.5	1.581
VLSI'24[20]	Virtex-7	4096	24	5.5k	4.9k	24	307	3081	13.5	7.28
TCAD'22[21]	Virtex-7	4096	24	14.6k	6.5k	12	121	1543	12.8	7.68
VLSI'24[16]	Virtex-7	4096	28	6.3K	3.4K	8	150	8.3K	55.8	2.06
VLSI'24[16]	Virtex-7	4096	28	8.4K	4.0K	8	150	4.2K	28.5	4.02
CF'22[32]	XCU200	4096	30	54.1k	56.2k	84	250	-	24.7	5.51
VLSI'25[30]	Virtex-7	4096	24	4.2k	4.6k	8	214	3107	14.5	6.78
ASHES'24[2]	VCK190	4096	14	-	-	-	1250	30175	24.14	2.38
ATTINA	VCK-190	4096	16	10290	11642	3	320	512	**6.5**	**10.08**

Table 3: Detailed power consumption ofATTINA on VCK-190.

Resource	Static		Dynamic	
	Power (W)	Ratio %	Power (W)	Ratio %
PMC	0.047	0.57	0.247	2.96
PL	3.148	37.68	0.284	3.40
AIE	1.060	12.69	2.284	27.34
NoC	0.152	1.82	1.133	13.56
Total	4.406	52.75	3.948	47.26

Table 4: Runtime of 16-bit ANTT on different platforms.

Platform	Freq. (MHz)	Latency (μs)	Throughput (Mb/s)
CPU (i9-14900K)	3100	905.46	74.19
PE only	1250	248.82	270.5
ATTINA (PL+PE)	320	6.50	10082

4.4 Comparison with Recent NTT Works

Since little literature is available on the ANTT accelerator, we provide a thorough performance evaluation by benchmarking ATTINA against other state-of-the-art (SOTA) NTT implementations. In particular, ATTINA operates in extended binary fields, while there are few hardware implementations involved. Although operating in different mathematical fields, all these implementations operate with similar input lengths and widths for a fair comparison.

Table 2 summarizes recent FPGA-based designs for comparative analysis. NTT designs with a length of 1024 points typically achieve extremely low latency, as seen in [18, 27]. However, the difficulty increases exponentially along with the NTT scales in either length. A 4096-point NTT requires a greater workload and more complex memory access, resulting in increased latency. On the one hand, Mu *et al.* [21], Liu *et al.* [20], and Wu [30] focus on developing a conflict-free scheme to alleviate bottlenecks in memory access by reordering the address of data points after completing butterfly units in each stage. They also initialize multiple computing kernels (usually 2×2 or 4×2) to enable parallel computation. On the other hand, Florian *et al.* [16] and Yang *et al.* [32] propose high-pipelined architectures, *i.e.*, single-path delay feedback (SDF) or multipath delay commutator (MDC), which enable bandwidth-efficient and high-throughput implementations. Fahim *et al.* [2] uses the same platform as ours; however, they put all computing units onto the AIE domain, rendering relatively high computing cycles. Among SoTAs, the ATTINA framework maintains a competitive throughput of around 10 Gb/s.

Interestingly, the comparison with recent NTT/INTT accelerators reveals two primary optimization directions. To address the substantial computational workload, high-throughput architectures like SDF or MDC are used to pipeline the computation stage by stage. In contrast, another branch of research focuses on alleviating the challenge of memory access by designing conflict-free schemes [20, 21, 30]. The proposed ATTINA framework incorporates the advantages of both approaches. It leverages the properties of binary towers and computes butterfly units with gate-level logic (PL domain). Furthermore, ATTINA allocates data permutation on PEs (AIE domain), as these PEs support multiple loads and shuffle simultaneously, while the memory bank in PL logic typically accesses single or dual addresses per cycle. In sum, the ATTINA framework leverages the advantages of two hardware components on an emerging ASoC platform, and distributes the two distinct types of tasks across the PL and AIE domains, respectively, to process ANTT/IANTT with high efficiency.

5 CONCLUSION

This work presents a scalable, hardware-friendly ANTT accelerator design, ATTINA, for enabling Binius operations at emerging heterogeneous edge devices. The proposed ATTINA exemplifies how coordinated hardware-software design methodologies can harness the programmable resources of modern adaptive SoC. This research provides a framework for hardware architects, computer designers, and cryptographers to explore novel strategies in heterogeneous architecture that build efficient high-performance primitives in edge computing devices.

6 ACKNOWLEDGMENT

This material is based upon work partially supported by the National Science Foundation under grant #2523805, and Commonwealth Cyber Initiative grant N-2Q26-007. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Authors Copy

REFERENCES

[1] 2024. Ethereum. https://github.com/ethereum/research/tree/master/binius.

[2] Fahim Ahmed and Md Tanvir Arafin. 2024. SANTA: A Spatial Accelerator Design for Efficient Number Theoretic Transform (NTT) on Heterogeneous System-on-Chips. In *Proceedings of the 2024 Workshop on Attacks and Solutions in Hardware Security*. 2–10.

[3] AMD. 2020. Versal: The First Adaptive Compute Acceleration Platform. https://docs.amd.com/v/u/en-US/wp505-versal-acap

[4] AMD. 2024. Overview AI Engine Kernel and Graph Programming Guide (UG1079). https://docs.amd.com/r/en-US/ug1079-ai-engine-kernel-coding.

[5] AMD. 2025. Power Design Manager User Guide (UG1556). https://docs.amd.com/r/en-US/ug1556-power-design-manager

[6] M Vijay Bhasker Reddy and Soujanya Duvvi. 2024. Blockchain Privacy through Zero-Knowledge Proofs: A Survey of Techniques and Use Cases. *Frontiers in Health Informatics* 13, 3 (2024).

[7] Vitalik Buterin. 2024. Binius: highly efficient proofs over binary fields. https://vitalik.eth.limo/general/2024/04/29/binius.html.

[8] Thomas Chen, Hui Lu, Teeramet Kunpittaya, and Alan Luo. 2022. A review of zk-snarks. *arXiv preprint arXiv:2202.06877* (2022).

[9] Benjamin E Diamond and Jim Posen. 2024. Polylogarithmic proofs for multilinears over binary towers. *Cryptology ePrint Archive* (2024).

[10] Benjamin E Diamond and Jim Posen. 2025. Succinct arguments over towers of binary fields. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 93–122.

[11] Yuval Domb. 2024. NTT 201: A Practical Introduction to Number-Theoretic Transforms. (August 2024). https://github.com/ingonyama-zk/papers/blob/main/ntt_201_book.pdf

[12] Electron-Labs. 2024. ETH-lc-plonky2. https://github.com/Electron-Labs/eth-lc-plonky2.

[13] Yonis Gulzar. 2023. Fruit image classification model based on MobileNetV2 with deep transfer learning technique. *Sustainability* 15, 3 (2023), 1906.

[14] Ulrich Haböck, David Levit, and Shahar Papini. 2024. Circle STARKs. Cryptology ePrint Archive, Paper 2024/278. https://eprint.iacr.org/2024/278

[15] Hassan Halawa, Hazem A Abdelhafez, Andrew Boktor, and Matei Ripeanu. 2017. Nvidia jetson platform characterization. In *Euro-Par 2017: Parallel Processing: 23rd International Conference on Parallel and Distributed Computing, Santiago de Compostela, Spain, August 28–September 1, 2017, Proceedings 23*. Springer, 92–105.

[16] Florian Hirner, Ahmet Can Mert, and Sujoy Sinha Roy. 2024. Proteus: A pipelined ntt architecture generator. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 32, 7 (2024), 1228–1238.

[17] Seyedehfaezeh Hosseininoorbin, Siamak Layeghy, Mohanad Sarhan, Raja Jurdak, and Marius Portmann. 2023. Exploring edge TPU for network intrusion detection in IoT. *J. Parallel and Distrib. Comput.* 179 (2023), 104712.

[18] Bin Li, Yunfei Yan, Yuanxin Wei, and Heru Han. 2024. Scalable and Parallel Optimization of the Number Theoretic Transform Based on FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 32, 2 (2024), 291–304. https://doi.org/10.1109/TVLSI.2023.3312423

[19] Sian-Jheng Lin, Wei-Ho Chung, and Yunghsiang S Han. 2014. Novel polynomial basis and its application to reed-solomon erasure codes. In *2014 ieee 55th annual symposium on foundations of computer science*. IEEE, 316–325.

[20] Si-Huang Liu, Chia-Yi Kuo, Yan-Nan Mo, and Tao Su. 2023. An area-efficient, conflict-free, and configurable architecture for accelerating NTT/INTT. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 32, 3 (2023), 519–529.

[21] Jianan Mu, Yi Ren, Wen Wang, Yizhong Hu, Shuai Chen, Chip-Hong Chang, Junfeng Fan, Jing Ye, Yuan Cao, Huawei Li, et al. 2022. Scalable and conflict-free NTT hardware accelerator design: Methodology, proof, and implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 5 (2022), 1504–1517.

[22] Yusuf Ozmiş. 2025. Applications Of Zero-Knowledge Proofs On Bitcoin. *arXiv preprint arXiv:2507.21085* (2025).

[23] Plonky3. 2025. Plonky3. https://github.com/Plonky3/Plonky3.

[24] PolygonZero. 2022. Plonky2: Fast Recursive Arguments with PLONK and FRI. https://github.com/0xPolygonZero/plonky2/blob/main/plonky2/plonky2.pdf

[25] Riccardo Salvatelli. 2024. *VerComp: A Framework for Verifiable Computation in Cloud Environments Using zk-STARK Proofs*. Ph. D. Dissertation. Politecnico di Torino.

[26] Srinath Setty. 2020. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Annual International Cryptology Conference*. Springer, 704–737.

[27] Nana Sutisna, Elkhan J Brillianshah, Infall Syafalni, M Ogin Hasanuddin, Trio Adiono, and Tutun Juhana. 2024. Low-complexity and high-throughput number theoretic transform architecture for polynomial multiplication in homomorphic encryption. In *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[28] Wu Tony and Shlomtz. 2025. Open-binius. https://github.com/ingonyama-zk/open-binius/tree/main.

[29] Jasmina Vasiljevic, Ljubisa Bajic, Davor Capalija, Stanislav Sokorac, Dragoljub Ignjatovic, Lejla Bajic, Milos Trajkovic, Ivan Hamer, Ivan Matosevic, Aleksandar Cejkov, et al. 2021. Compute substrate for Software 2.0. *IEEE micro* 41, 2 (2021), 50–55.

[30] Zhenyang Wu, Ruichen Kan, Jianbo Guo, and Hao Xiao. 2025. Scalable and Low-Cost NTT Architecture With Conflict-Free Memory Access Scheme. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2025).

[31] Tiancheng Xie, Yupeng Zhang, and Dawn Song. 2022. Orion: Zero knowledge proof with linear prover time. In *Annual International Cryptology Conference*. Springer, 299–328.

[32] Yang Yang, Sanmukh R. Kuppannagari, Rajgopal Kannan, and Viktor K. Prasanna. 2022. NTTGen: a framework for generating low latency NTT implementations on FPGA. In *Proceedings of the 19th ACM International Conference on Computing Frontiers* (Turin, Italy) *(CF '22)*. Association for Computing Machinery, New York, NY, USA, 30–39. https://doi.org/10.1145/3528416.3530225

[33] Zewen Ye, Ray CC Cheung, and Kejie Huang. 2022. PipeNTT: A pipelined number theoretic transform architecture. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 10 (2022), 4068–4072.

[34] Hadas Zeilberger, Binyi Chen, and Ben Fisch. 2024. BaseFold: efficient field-agnostic polynomial commitment schemes from foldable codes. In *Annual International Cryptology Conference*. Springer, 138–169.

Authors Copy