electronics

MDPI

Article

HLSCAM: Fine-Tuned HLS-Based Content Addressable Memory Implementation for Packet Processing on FPGA

Mostafa Abbasmollaei [1,*], Tarek Ould-Bachir [1,*] and Yvon Savaria [2,*]

[1] MOTCE Laboratory, Department of Computer Engineering, Polytechnique Montréal, Montréal, QC H3T 1J4, Canada
[2] Department of Electrical Engineering, Polytechnique Montréal, Montréal, QC H3T 1J4, Canada
* Correspondence: mostafa.abbasmollaei@polymtl.ca (M.A.); tarek.ould-bachir@polymtl.ca (T.O.-B.); yvon.savaria@polymtl.ca (Y.S.)

Abstract: Content Addressable Memories (CAMs) are pivotal in high-speed packet processing systems, enabling rapid data lookup operations essential for applications such as routing, switching, and network security. While traditional Register-Transfer Level (RTL) methodologies have been extensively used to implement CAM architectures on Field-Programmable Gate Arrays (FPGAs), they often involve complex, time-consuming design processes with limited flexibility. In this paper, we propose a novel templated High-Level Synthesis (HLS)-based approach for the design and implementation of CAM architectures such as Binary CAMs (BCAMs) and Ternary CAMs (TCAMs) optimized for data plane packet processing. Our HLS-based methodology leverages the parallel processing capabilities of FPGAs through employing various design parameters and optimization directives while significantly reducing development time and enhancing design portability. This paper also presents architectural design and optimization strategies to offer a fine-tuned CAM solution for networking-related arbitrary use cases. Experimental results demonstrate that HLSCAM achieves a high throughput, reaching up to 31.18 Gbps, 9.04 Gbps, and 33.04 Gbps in the 256 × 128, 512 × 36, and 1024 × 150 CAM sizes, making it a competitive solution for high-speed packet processing on FPGAs.

Keywords: Content-Addressable Memory; FPGA; High-Level Synthesis; packet processing

Academic Editors: Raffaele Giordano and Chen Yang

Received: 4 March 2025
Revised: 10 April 2025
Accepted: 23 April 2025
Published: 26 April 2025

Citation: Abbasmollaei, M.; Ould-Bachir, T.; Savaria, Y. HLSCAM: Fine-Tuned HLS-Based Content Addressable Memory Implementation for Packet Processing on FPGA. *Electronics* **2025**, *14*, 1765. https://doi.org/10.3390/electronics14091765

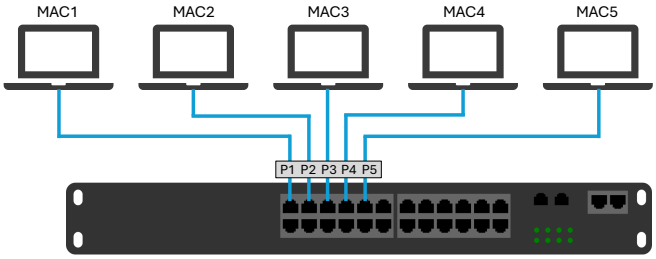
1. Introduction

Content-Addressable Memories (CAMs) are critical components in high-performance computing and data analysis. They enable fast data lookup operations, which are essential for many applications. For example, CAMs are used in packet routing and switching [1,2]. They also play a role in DNA sequence analysis [3,4]. In addition, CAMs support machine learning and data analysis, especially with approximate CAM designs [5].

Unlike traditional Random-Access Memories (RAMs), which access data using specific addresses, CAMs enable parallel content search. This significantly speeds up lookup operations and is especially crucial in data plane packet processing. Tasks like IP address lookup, access control, and flow classification demand fast and efficient search mechanisms to meet strict throughput and latency requirements [6].

Various CAM types serve distinct roles within packet processing pipelines. Binary CAMs (BCAMs) are optimized for exact match lookups, making them ideal for applications such as MAC address tables, as illustrated in Figure 1. Ternary CAMs (TCAMs) support wildcard-based matching, allowing flexible rule definitions that are indispensable in applications such as access control lists (ACLs) and packet classification. Additionally,

a specific type of TCAM could only consider prefix matching to handle Longest Prefix Match (LPM) operations, which are crucial in IP routing. In LPM, each subnet in a network defines a range of IP addresses with a common prefix, and STCAMs efficiently match these prefixes to determine the most specific routing entry for a given IP address. This specialized functionality makes STCAMs highly effective in network IP matching and routing table lookups, where hierarchical addressing schemes are prevalent.



MAC Address	Port
MAC1	P1
MAC2	P2
MAC3	P3
MAC4	P4
MAC5	P5

Figure 1. The use of CAM tables in L2 switching.

Despite their advantages, implementing CAMs for high-throughput environments presents several challenges. Traditional CAM designs, often realized through Application-Specific Integrated Circuits (ASICs), offer excellent performance but suffer from high development costs and lack flexibility for evolving network requirements. The adoption of Field-Programmable Gate Arrays (FPGAs) for CAM implementations has surged due to their inherent parallel processing capabilities and reconfigurability, offering flexibility over traditional Application-Specific Integrated Circuits (ASICs). FPGA-based CAMs are typically realized using four primary on-chip memory types: Ultra RAM (URAM), Block RAM (BRAM), Lookup Table RAM (LUTRAM), and flip-flops (FFs). Each memory type presents trade-offs in terms of resource utilization, power efficiency, and scalability. For example, BRAM-based CAMs offer high density but suffer from longer update latencies, whereas LUTRAM-based CAMs provide faster updates at the cost of increased routing complexity. FF-based CAMs, though less scalable, excel in applications requiring minimal hardware overhead and rapid updates.

FPGAs offer a reconfigurable alternative that can provide CAM performance while providing additional adaptability. However, traditional Register-Transfer Level (RTL) design methods for implementing CAMs on FPGAs present several challenges. These approaches are complex, time-consuming, and error-prone, which hinders rapid prototyping and design space exploration. High-Level Synthesis (HLS) addresses these limitations by introducing a new design paradigm. HLS allows hardware description using high-level programming languages like C/C++, significantly simplifying the development process.

HLS also facilitates accelerated development cycles, improved design portability, and simplified integration with software-driven workflows. While HLS has demonstrated its potential in various application domains, to the best of our knowledge, no prior work specifically addresses the design and implementation of CAM architectures optimized for packet processing using HLS.

This gap in the literature motivates the present paper, which proposes HLSCAM (https://github.com/abbasmollaei/HLSCAM, accessed on 24 April 2025) an HLS-based approach to implement CAM architectures optimized for data plane packet processing on FPGAs. The contributions of the paper include the following:

- It presents a templated HLS-based implementation of BCAMs for exact matching and TCAMs for ternary matching, supporting arbitrary table sizes for data plane packet processing.

- It explores the trade-offs between resource and latency to flexibly cover the design space for different FPGA-based packet processing use cases.
- It leverages FPGA parallelism through applying varied design parameters and optimization directives for CAM implementation in HLS to achieve high operating frequencies with minimal latency.
- It explores the design space for CAMs by varying memory depths, key widths, and optimization methods applied to evaluate performance and resource occupation comprehensively.

The remainder of this paper is structured as follows. Section 2 reviews related work on FPGA-based CAM implementations. Section 3 presents the proposed HLS-based CAM design, detailing the architectural framework, optimization strategies for BCAM, TCAM, and memory allocation schemes. Section 4 describes the experimental environment and provides a comprehensive analysis of the results, including design space exploration, performance evaluation based on various metrics, and a comparison with State-of-the-Art approaches. Finally, Section 5 concludes the paper by summarizing key findings and highlighting potential directions for future research.

2. Literature Review

Extensive research has been conducted to optimize FPGA-based CAM architectures, focusing on improving throughput, reducing power consumption, and enhancing resource efficiency. Early implementations, such as the Hybrid Partitioned TCAM (HP-TCAM) [7], introduced partitioning techniques to mitigate the exponential growth in resource requirements associated with brute-force approaches. Although HP-TCAM improved memory efficiency, it required pre-processed, ordered data and lacked dynamic update capabilities. Subsequent designs, such as Ultra-Efficient TCAM (UE-TCAM) [8], further optimized resource utilization by reducing redundant memory structures, even though these designs remained reliant on complex Register-Transfer Level (RTL) methodologies. An FPGA-based method for the construction of deep and narrow BCAMs using Segmented Transposed Indicators RAM (STIRAM) is proposed in [9]. This approach stores segment indicators and patterns in separate RAM structures, enabling a two-cycle match operation with concurrent reading. The authors in [10] introduce FMU-BiCAM algorithms for binary CAMs on FPGA, achieving two-cycle updates by directly using CAM keys as addresses. It reduces power consumption by eliminating lookup tables and limiting updates to necessary SRAM blocks.

LUTRAM-based CAM architectures, such as DURE-TCAM [11], D-TCAM [12], and ME-TCAM [13], have demonstrated improved update efficiency and reduced latency. DURE-TCAM [11] introduced a dynamically updateable TCAM architecture for FPGAs that uses LUTRAM primitives. DURE addresses the issue of blocking updates in existing SRAM-based TCAMs by enabling simultaneous search and update operations. The D-TCAM architecture [12] uses 6-input LUTRAMs to emulate 6-bit TCAM, and combines 64 of these LUTRAMs to create a 48-byte D-CAM block. These D-CAM blocks are cascaded horizontally and vertically to increase the width and depth of the TCAM, respectively. It also exploits the LUT-FF pair nature of FPGAs, using redundant flip-flops as pipeline registers.

ME-TCAM [13] introduces a Memory-Efficient Ternary Content Addressable Memory scheme using multipumping-enabled LUTRAM on FPGAs to enhance memory efficiency over traditional SRAM-based TCAMs. The TCAM table is partitioned into Hybrid Partitions (HPs), with each HP simulated using four LUTRAMs. However, increasing the multipumping factor improves memory efficiency at the cost of reduced throughput. RPE-TCAM [14] proposes a reconfigurable power-efficient TCAM for FPGAs that reduces power consumption by 40% using selective bank activation. It maintains one-cycle update latency

and supports BiCAM and TCAM configurations. A backup CAM (BUC) handles bank overflow, ensuring efficient memory use.

An SRAM-based TCAM architecture (REST) is proposed in [15]. It improves memory efficiency by optimizing address mapping using virtual blocks (VBs). By dividing SRAM into multiple VBs, REST increases the emulated TCAM capacity while reducing throughput. The design allows a trade-off between memory efficiency and throughput by adjusting the number of VBs. A multi-region SRAM-based TCAM architecture for longest prefix matching is presented in [16], reducing memory area usage by dividing TCAM entries into index and data fields. The index maps to SRAM addresses, while the data stores content, improving memory efficiency. Another work [17] presents power-efficient FPGA-based TCAM architecture using a segmented match line strategy to reduce dynamic power consumption. Each match line is divided into four nine-bit segments, activated sequentially to prevent unnecessary power usage.

Meanwhile, flip-flop-based designs like LH-CAM [18] and G-AETCAM [19] explored the use of FPGA slice registers to achieve low-latency, high-speed CAM operations. While these designs achieved impressive performance for small-scale implementations, they faced scalability challenges due to routing complexity and hardware overhead.

TeRa [2] introduces a ternary and range-based CAM architecture for packet classification, offering a power-efficient alternative to TCAMs. It features Ternary Match Logic (TML) with NAND-NOR encoding, a carry tree-based Range Match Logic (CTRC) for range comparisons, and Match Inversion Logic for efficient rule handling. TYPE1 is optimized for ASICs, while TYPE2 is adaptable for both ASICs and FPGAs, enhancing priority selection and memory efficiency.

A reconfigurable match table (RMT) design for FPGA-based switches has been proposed in [1], incorporating a TCAM-based implementation for flexible packet processing. The design employs a three-layer structure enabling dynamic reconfiguration of match table size and type without hardware modification. A segment crossbar facilitates PMT sharing and signal conversion, including a MAND operation for combining match lines across TCAM PMTs.

Despite these advances, existing research predominantly focuses on RTL-based design methodologies. While RTL offers fine-grained control over hardware resources, it is inherently time-consuming, complex, and less adaptable to evolving design requirements. HLS offers more flexibility, enables faster development, enhances design portability, and simplifies design space exploration. While CAM architectures have been extensively explored at the RTL and lower design levels, HLS-based implementations remain largely unexplored, particularly for designing match-action tables in packet processing applications.

3. Potential Network Applications

The proposed HLS-based CAM architecture is particularly suited for high-speed data plane operations in modern networking systems. Its flexibility and configurability enable efficient integration into various packet processing applications. Below are several use cases that illustrate how CAMs can be practically deployed in real-world networking environments.

3.1. Packet Classification and Filtering

CAMs are commonly used to match packet header fields (e.g., IP addresses, protocol numbers, port numbers) against a set of predefined rules. This operation is fundamental in routers, firewalls, and switches to determine the appropriate action for each packet, such as forwarding, dropping, or rate-limiting. For example, a TCAM can match the quintuple of

an incoming packet (source/destination IP and port, and protocol) with a set of entries in the Access Control List (ACL), allowing multi-field classification in a single clock cycle [20].

3.2. Firewall and Intrusion Detection Systems (IDS)

TCAMs are ideal for rule-based packet inspection, where wildcard matching is often required. For example, a firewall can use TCAM entries to define rules such as 'allow all traffic from subnet 192.168.0.X', using ternary bits to represent masked fields. IDS applications benefit similarly by matching known attack signatures or suspicious traffic behavior patterns across various protocol headers, providing real-time threat detection [21].

3.3. In-Band Network Telemetry (INT)

CAMs can be used to identify specific flows that require telemetry information insertion. Upon a CAM match, INT metadata is appended to the packet to track path information (e.g., latency, hop count) through the network. This enables operators to monitor network performance and detect anomalies with minimal overhead [22].

3.4. Network Address Translation (NAT)

In NAT-enabled devices, CAMs can serve as lookup tables to translate between internal private IP addresses and external public addresses. By maintaining a CAM of current translation entries, the device can efficiently identify and rewrite packet headers during ingress and egress processing [23].

3.5. Time-Sensitive Networking (TSN)

In deterministic networking environments, such as industrial automation or autonomous systems, CAMs help enforce strict scheduling and traffic-shaping rules. By matching packets against timing-based policies, CAMs enable precise control of transmission schedules to meet latency and reliability requirements [24].

3.6. User Management in 5G Core

CAMs play a crucial role in user session management and packet forwarding in 5G core network components, such as the User Plane Function (UPF) [25] and Access Gateway Function (AGF) [26]. In these 5G core sections, CAMs are used to match packet headers against subscriber session entries to enforce policy rules, quality-of-service (QoS) settings, and forwarding paths. This enables real-time user plane operations, including traffic steering, usage reporting, and IP anchor point management, all of which demand low-latency, high-throughput processing.

4. CAM Architectures, Implementations, and Trade-Offs

This work presents a templated HLS-based design and implementation of BCAM and TCAM architectures, adjusted for data plane packet processing applications on FPGAs. Our approach leverages the Vitis HLS tool for AMD (Xilinx) FPGAs, focusing on three distinct design perspectives: **Brute Force (BF)**, **Balanced (BL)**, and **High-Speed (HS)**. Each perspective addresses specific trade-offs between resource utilization, latency, and throughput, achieved by fine-tuning hardware directives provided by the Vitis HLS tool. Various design directives are applied to configure memory space allocation and access patterns based on application requirements to manage FPGA parallelism precisely. Additionally, the implementation is structured to allow the synthesis tool to optimize timing constraints and reach high operating frequencies.

4.1. Design Perspectives

4.1.1. Brute-Force CAM

The Brute-Force method implements sequential memory access, processing lookups two entries at a time. Figure 2 illustrates the hardware structure of the proposed CAM lookup in BF mode. The search engine is responsible for sequentially reading each entry and executing the required comparison and matching operations, depending on the CAM type. The allocated memory space consists of a single memory module with two ports for the entire CAM table. The first port is dedicated to read-only operations, while the second port supports both read and write functions.

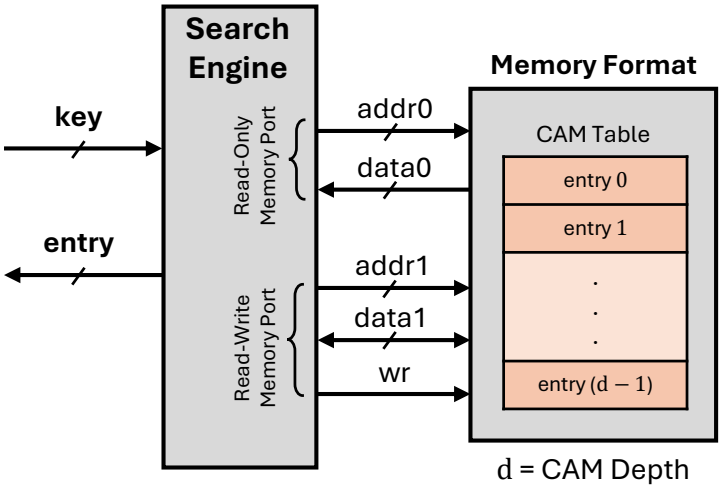


Figure 2. The hardware structure of CAM lookup in the BF mode on FPGA.

This approach prioritizes minimal resource utilization, making it ideal for applications with strict hardware limitations. Although it greatly reduces the consumption of FPGA resources, it introduces higher latency because of the restricted number of memory ports and the resulting serialized search procedure.

4.1.2. Balanced CAM

The Balanced method aims to provide a trade-off between resource utilization and performance. This architecture employs partial parallelism in memory access, where search operations are distributed across multiple parallel memory units. Figure 3 depicts the proposed CAM lookup hardware structure in BL mode. Consequently, the memory is divided into PF (Partition Factor) blocks, each containing PD (Partition Depth) entries. The relationship between PF and PD is defined by Equation (1).

$$Partition\ Depth\ (PD) = \frac{DEPTH}{Partition\ Factor\ (PF)} \tag{1}$$

This partitioning scheme increases the number of memory ports by a factor of PF, resulting in 2 $\times$ PF ports. Consequently, the search engine requires PD clock cycles to read all entries. Thus, the minimum achievable latency in the BL method is PD cycles, excluding the additional processing time required to identify the correct matching entry. During operation, the search engine schedules entry addresses and selects the appropriate address at each clock cycle using multiplexers, retrieving the entries for the matching process. The memory port signals are organized into two bundles: *bundle_r* for read-only ports and *bundle_wr* for read/write ports.

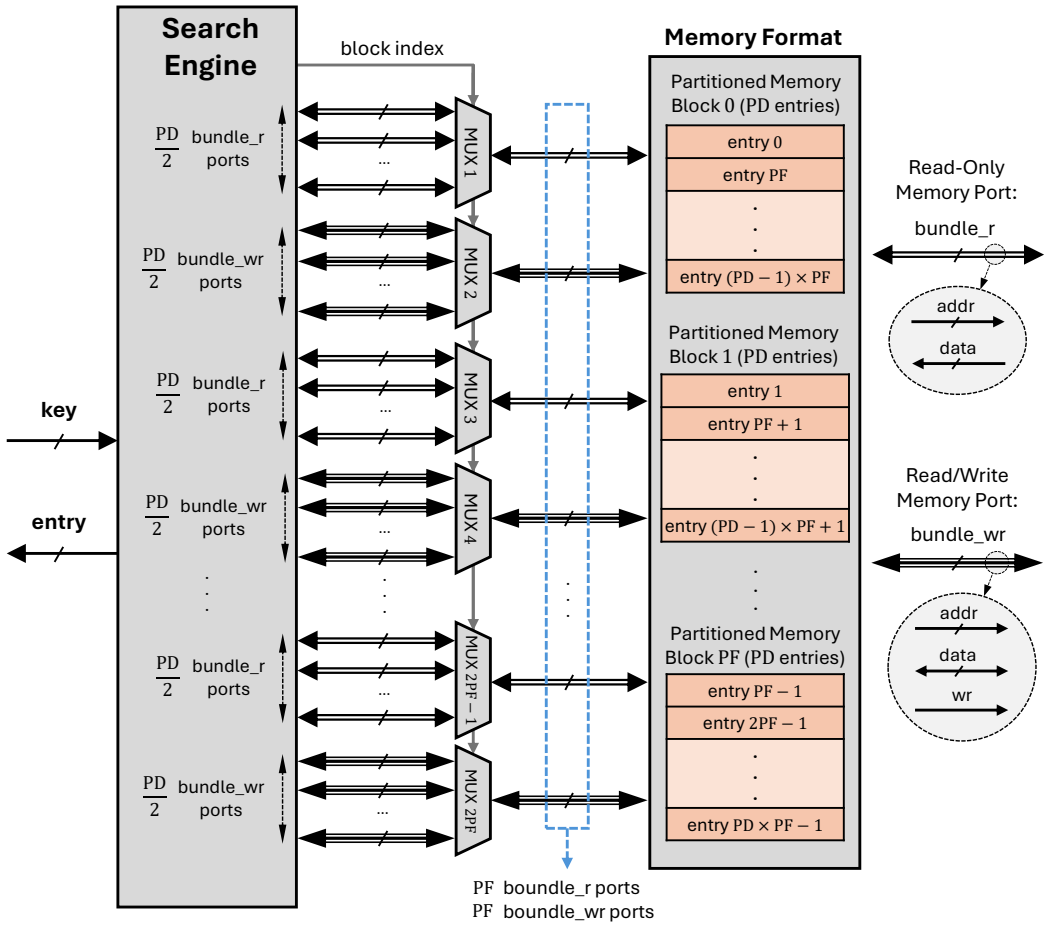


Figure 3. The hardware structure of CAM lookup in the BL mode on FPGA.

This approach reduces latency compared to the BF method while maintaining moderate resource usage. In Vitis HLS, we achieved this balance by selectively applying partial memory block partitioning, loop unrolling, and pipelining directives, allowing controlled parallel data paths. This design is well-suited for scenarios where both performance and resource efficiency are critical, such as mid-range packet processing applications.

4.1.3. High-Speed CAM

The High-Speed method is optimized for maximum throughput and minimal latency, fully exploiting the parallelism of FPGAs to enable simultaneous comparisons across multiple CAM entries. As illustrated in Figure 4, the hardware architecture in High-Speed mode features a flattened memory structure, where each entry is assigned a dedicated memory port. This configuration allows the search engine to access all entries within a single clock cycle, achieving the lowest possible latency compared to other modes. We extensively utilized directives such as full loop unrolling, aggressive pipelining, and array partitioning in Vitis HLS to maximize data flow concurrency. While this approach results in higher resource utilization, it achieves the lowest possible latency.

Given the parallel nature of operations in the HS mode, the intermediate wiring and routing logic consume significant FPGA resources. We introduce an enhanced version of the HS mode, referred to as HS-H, to improve resource efficiency. The HS-H architecture follows a hierarchical design. It matches the input key and returns the index of each matching entry, or an invalid value if no match is found. The valid indexes are then forwarded through tree-based multiplexers. This hierarchical approach reduces the number of intermediate signals propagating across multi-cycle matching operations, thereby saving logic resources

compared to the original HS mode. However, the trade-off shows a slight increase in the required number of clock cycles due to the added hierarchical structure.

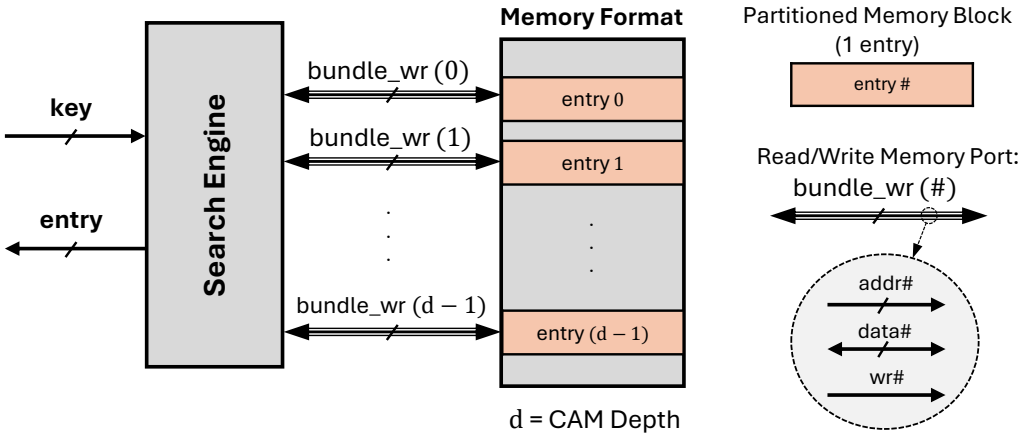
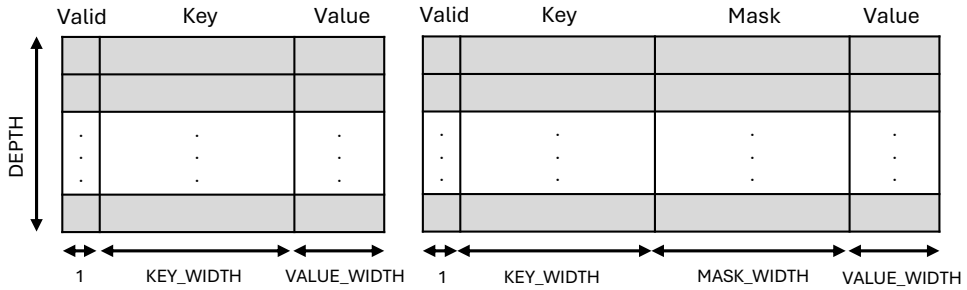


Figure 4. The hardware structure of CAM lookup in the HS mode on FPGA.

4.2. BCAM

The BCAM implementation follows a template-based structure, allowing for the flexible parameterization of key width, value width, and depth. The core structure of BCAM consists of an array of entries, where each entry is defined by a key, a corresponding value, and a validity flag as shown in Figure 5a. The BCAM is implemented as a class template (BCAM<KEY_WIDTH, VALUE_WIDTH, DEPTH>) to support a customizable key size, value size, and memory depth.

The `search_entry` function performs an exact match lookup by iterating through all entries, with its level of parallelism determined by the selected optimization mode. In HS mode, full parallelism is applied by distributing over distinct memory blocks, allowing all entries to be checked simultaneously for minimal latency. In BL mode, the degree of parallelism is proportionally determined by the PF, as higher PF values enable more memory partitions to be accessed concurrently, enhancing parallel lookup. In BF mode, the search is fully sequential, processing one entry at a time without parallelism. The function finally finds the matched index and returns the corresponding entry.



(a) BCAM (b) TCAM

Figure 5. The implemented structures of BCAM and TCAM.

4.3. TCAM

Ternary matching in TCAM allows each bit of a stored key to represent '0', '1', or a wildcard ('X'). This is implemented by associating each key bit with a corresponding mask bit. When a mask bit is enabled, the associated key bit becomes a wildcard, matching both '0' and '1' during lookup. For example, a stored IPv4 prefix of 192.168.1.X ($C0.A8.01.XX$) can be represented by a key and a mask where the last six bits are masked, allowing any

value in those positions. This would match IP addresses such as 192.168.1.10 $(C0.A8.01.0A)$ or 192.168.1.63 $(C0.A8.01.3F)$.

The proposed TCAM class provides a templated key-value storage system that supports ternary matching, allowing wildcard bits in key comparisons. Each entry consists of a key, a mask, a value, and a validity flag, as shown in Figure 5b. The `search_entry` function performs a lookup by iterating over all entries and checking for a ternary match. As illustrated in Figure 6, the matching process occurs in two stages. In the first stage, the function applies a bitwise XNOR between the stored and input keys to determine bit similarity. The result is then masked using the stored mask, and a validity check is performed. The outcome is stored as a Boolean value in an array, indicating whether each entry is a potential match. In the second stage, the Parallel Valid Index Finder (PVIF) module scans the array to locate the first valid match and retrieves the corresponding index and entry.

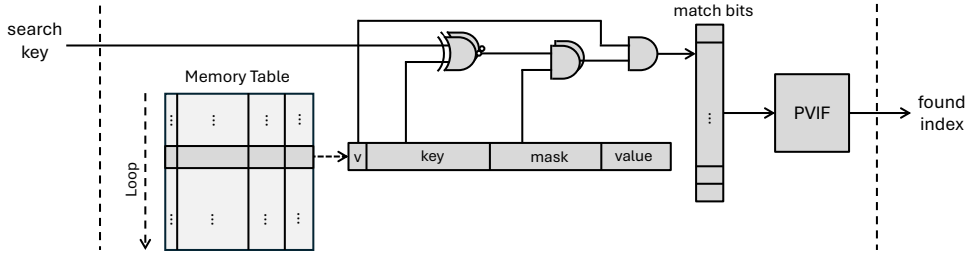


Figure 6. The TCAM search entry logic.

4.4. Memory Allocation

The proposed HLSCAM solution employs different memory allocation strategies in BF, BL, and HS modes, optimizing the use of FPGA resources based on performance and parallelism requirements. The primary memory resource utilized in the BF and the BL modes is LUTRAM, while the HS mode relies on register-based storage to maximize parallel access.

In the HS mode, the CAM array is fully flattened and mapped into FPGA registers, enabling simultaneous access to all entries. This approach eliminates the sequential memory access bottleneck associated with LUTRAM, allowing each lookup to be processed in a single cycle. However, storing CAM entries in registers instead of LUTRAM significantly increases FF utilization, which may become a limiting factor in large-scale CAM deployments. Each bit of a CAM entry in HS mode is mapped to a dedicated FF, allowing true parallel access and one-cycle lookup latency. This direct mapping ensures that all entries are immediately accessible without sequential memory traversal. However, such register-based implementation requires individual addressing for each bit, which can introduce additional logic overhead for managing lookup control and can complicate placement and routing during synthesis. This trade-off between speed and resource consumption must be carefully balanced, particularly for large-scale CAMs.

In the BF and BL modes, the CAM architecture utilizes LUT6 (LUT 6-input) elements configured as LUTRAM, where each LUT6 can store 64 bits. Figure 7 illustrates the LUT6 organization for a BCAM depth of 64 entries, each with a 25-bit width (1-bit valid flag, 16-bit key, and 8-bit value) in the BF mode. To accommodate the dual-port memory access used in BF mode, the entries are divided into two groups:

- **Even-indexed entries:** Mapped to the left group of LUTs.
- **Odd-indexed entries:** Mapped to the right group of LUTs.

Each group contains 32 entries, and all LUTs within a group share the same address input to provide simultaneous access. Each bit of an entry is stored at the same LUT address across different LUTs. Given the 25-bit entry width, 25 LUTs are required per

group, resulting in a total of 50 LUTs for this BCAM configuration. However, due to the fixed 64-bit capacity per LUT, half of the bits in each LUT remain unused, leading to lower LUTRAM utilization efficiency. If the CAM architecture were configured with a single memory port instead of two, the LUT usage would be reduced by half, requiring only 25 LUTs instead of 50. However, this would come at the cost of approximately doubling the lookup latency, as each cycle could only access one entry at a time instead of two.

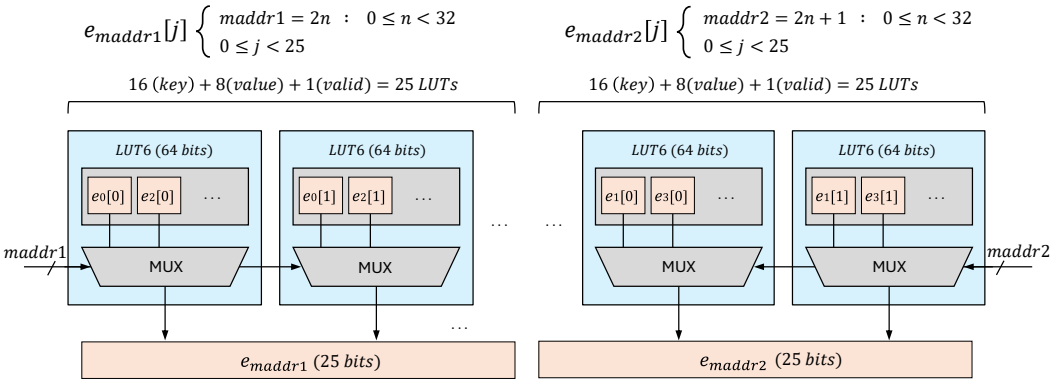


Figure 7. Memory allocation for the proposed BCAM in the BF mode with double memory ports and depth = 64, entry width = 25 bits (16-bit key, 8-bit value, 1-bit valid).

In this mode, the HLSCAM employs two memory ports to divide entries into two groups: even indexes and odd indexes. The even entries are mapped at the left group of LUTs, and the odd entries are mapped in the right group of LUTs, placing 32 entries into each group. The address input for the LUTs of each group is the same. Each bit of entries is distributed to the same address at LUTs. Since the entry width is 25 bits, 25 LUTs are needed for each group, occupying 50 LUTs in total. In this case, half the bits of each LUT remains unused. If we had single memory ports, the LUT usage would be half, and the latency would be around doubled.

Although double-port memory used in the BF mode provides an efficient memory structure, it suffers from limiting the available memory bandwidth, resulting in linearly growing latency by increasing the CAM depth. The CAM storage is partitioned across more than two LUTs to enable parallel memory access, a process controlled by the PF parameter. The PF determines the number of LUTs allocated per CAM configuration, directly influencing memory access parallelism and resource utilization. Figure 8 shows a

$$\begin{aligned} \text{LUTRAM usage} &= \text{ENTRY WIDTH} \times \lceil \frac{\text{CAM DEPTH}}{2^6} \rceil \times \text{\# of memory ports} \\ &= \text{ENTRY WIDTH} \times \lceil \frac{\text{CAM DEPTH}}{2^6} \rceil \times \text{PF} \times 2 \end{aligned} \tag{2}$$

Higher PF values allocate more LUTs to store CAM entries, increasing parallel read/write access and reducing lookup latency. However, higher PF values also lead to LUTRAM fragmentation, as each partitioned LUT may contain unused bits that cannot be reassigned to other memory segments. This results in lower LUTRAM utilization efficiency despite improved parallelism.

In the BF mode, sequential lookups require minimal partitioning, leading to higher LUT utilization efficiency but increased lookup latency due to serial memory access. In contrast, the BL mode introduces a moderate level of partitioning, striking a balance between parallel memory access and LUTRAM efficiency.

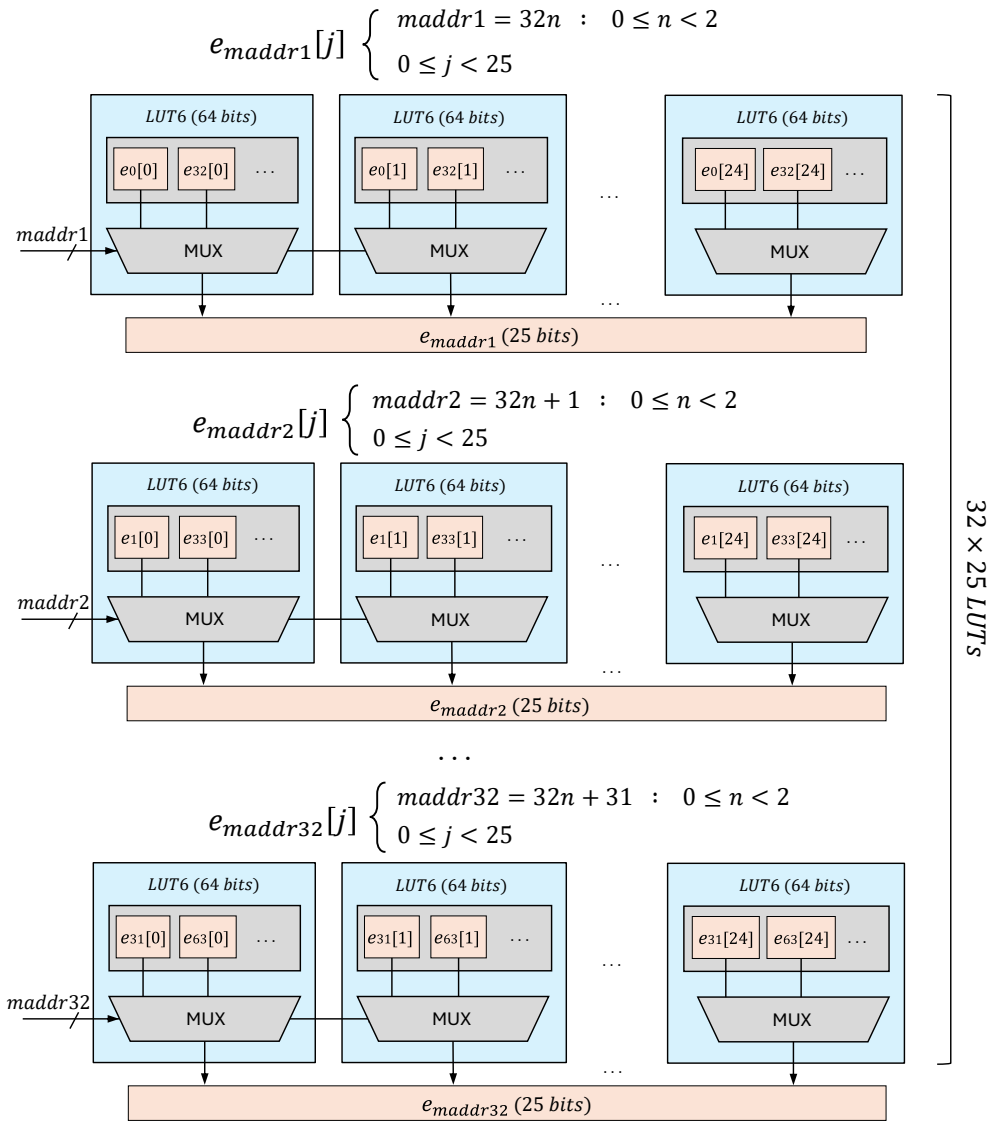


Figure 8. Memory allocation for the proposed BCAM in the BL mode with $PF = 16$, depth = 64, entry width = 25 bits (16-bit key, 8-bit value, 1-bit valid).

5. Evaluation

This section evaluates the HLS-based BCAM and TCAM implementations under various design spaces. The evaluation focuses on measuring latency, maximum frequency, throughput, and FPGA resource utilization, highlighting the trade-offs between performance and hardware efficiency. Latency was measured in terms of the number of clock cycles required for a single lookup operation. At the same time, throughput was computed based on the operating frequency and parallelism in memory access.

5.1. Experimental Setup

The BCAM and TCAM architectures were implemented using AMD Vitis HLS 2023.2, which enables high-level hardware description and optimization for FPGAs. The synthesized designs were evaluated using AMD Vivado to collect detailed resource utilization reports. The AMD Alveo U280 FPGA (xcu280-fsvh2892-2L-e) was targeted as a high-performance acceleration card well-suited for high-throughput networking applications. The initial clock frequency was set to 312.5 MHz to align with standard high-speed networking requirements, such as those in 10G and 25G Ethernet systems. The maximum operating frequency (f_{max}) was determined based on the critical path identified during

FPGA synthesis and post-place-and-route analysis. All reported results were obtained after placement and routing to ensure accurate performance evaluation.

5.2. Design Space Exploration

The design space of the proposed CAM architectures was explored by varying CAM depth, key width, and optimization methods. The evaluation considered three different table depths: 64, 128, and 256 entries. The 32, 64, and 128 key widths were chosen to analyze the impact of various configurations on performance and resource utilization. The stored value word width remained constant, as the complexity of the matching process depends mostly on the key rather than the stored value. Moreover, the *PF* was fixed to 16 to divide CAMs into 16 partitions and provide 32 distinct memory ports. These variable parameters were selected to restrict the design space for focused analysis. At the same time, the proposed templated HLSCAM can go beyond these parameters that we evaluate with higher parameters in the Section 5.5. This exploration allowed us to identify optimal configurations that balance efficiency, scalability, and application-specific requirements.

The synthesis performance and utilization of resources of the proposed BCAM architecture was evaluated in different configurations, as presented in Table 1. In the BF mode, both LUT and register usage increase proportionally with growing table depth and key width. Latency directly correlates with CAM depth, as each lookup requires sequential comparisons. The measured latencies are 33, 65, and 129 clock cycles for 64, 128, and 256 entries, respectively. The maximum operating frequency varies between 346 MHz and 397 MHz, depending on key width and resource constraints.

In the BL mode, LUT Logic and LUTRAM usages grow linearly with key width but LUTRAM usage remains constant as the BCAM depth increases. This behavior results from *PF* being fixed at 16 across all depths. As described by Equation (2), the number of required LUTRAMs remains unchanged until the partitioned memory blocks can no longer accommodate additional bits, increasing LUTRAM usage. Register utilization follows a similar trend, changing slightly with depth variations but linearly increasing with key width. Depending on key width and depth, the maximum lookup latency varies from 4 to 10 clock cycles. The maximum operating frequency is variable from 306 MHz to 349 MHz, which is relatively lower than that of the BF mode.

In the HS mode, the tool shifts from LUTRAM storage to register-based storage for fully partitioned entries, reducing total LUT utilization. While LUT usage in HS mode is lower than in BL mode for depths of 64 and 128 entries, it increases significantly for 256-entry configurations due to the higher intermediate logic and wiring requirements for parallel processing. The use of registers grows more rapidly than in the BL mode, reflecting the use of registers to store BCAM entries. The lookup latency is minimized to one cycle for 64 entries with a key width of 32, while it increases to two cycles for larger configurations. The maximum operating frequency ranges from 314 MHz to 345 MHz, which are close to the BL mode.

The TCAM architecture exhibits synthesis performance trends similar to BCAM as presented in Table 2, with increased resource utilization due to the added complexity of ternary matching and additional storage required for mask bits. Compared to BCAM, LUT, and register usage are approximately doubled across all configurations. Latency remains identical to BCAM, maintaining 33, 65, and 129 cycles in the BF mode and ranging between 5 and 10 cycles in the BL mode. In the HS mode, lookup latency remains at one or two cycles, depending on the key width and table depth. The range of maximum operating frequencies is slightly wider than in BCAM, varying from 294 MHz to 412 MHz.

Table 1. Resource Utilization and Performance Analysis of HLSCAM BCAM in the BF, BL, and HS modes. The value width is fixed to 8-bit.

Mode	Depth	Key Width (Bit)	Latency (Cycle)	LUT Logic	LUTRAMs	SRs [1]	f_{max} (MHz)
BF	64	32	33	311	82	456	397
		64	33	318	146	521	389
		128	33	488	274	661	383
	128	32	65	527	164	828	357
		64	65	594	292	902	346
		128	65	967	548	1061	356
	256	32	129	1025	328	1574	373
		64	129	1097	584	1657	360
		128	129	1563	1096	1852	364
BL	64	32	4	951	1312	1619	341
		64	4	1552	2336	2692	334
		128	5	2769	4384	4771	322
	128	32	6	1142	1312	1675	349
		64	6	1798	2336	2719	345
		128	7	2952	4384	5074	325
	256	32	10	1289	1312	1776	338
		64	10	2092	2336	3086	326
		128	10	3136	4384	5582	306
HS	64	32	1	1229	0	2627	345
		64	2	1934	0	4759	344
		128	2	3267	0	8940	337
	128	32	2	2497	0	5308	329
		64	2	3899	0	9472	316
		128	2	6561	0	17,746	314
	256	32	2	4992	0	10,574	317
		64	2	7717	0	18,873	321
		128	2	12,997	0	35,386	320

[1] Slice Registers.

Table 2. Resource Utilization and Performance Analysis of HLSCAM TCAM in the BF, BL, and HS modes. A TCAM entry holds key and mask portions having the same length. The value width is fixed to 8-bit.

Mode	Depth	Key Width (Bit)	Latency (Cycle)	LUT Logic	LUTRAMs	SRs [1]	f_{max} (MHz)
BF	64	32	33	303	146	520	412
		64	33	373	274	649	404
		128	33	626	530	917	372
	128	32	65	566	292	892	386
		64	65	627	548	1030	382
		128	65	1114	1060	1317	346
	256	32	129	1100	584	1636	337
		64	129	1140	1096	1783	340
		128	129	2046	2120	2108	336

Table 2. *Cont.*

Mode	Depth	Key Width (Bit)	Latency (Cycle)	LUT Logic	LUTRAMs	SRs [1]	f_{max} (MHz)
BL	64	32	4	1563	2336	2644	395
		64	5	2921	4384	4741	360
		128	5	5633	8480	8945	357
	128	32	6	1971	2336	2691	343
		64	7	3368	4384	4894	341
		128	7	6128	8480	9296	328
	256	32	10	2138	2336	2871	335
		64	10	3289	4384	5122	377
		128	10	6197	8480	9565	315
HS	64	32	1	2552	0	4676	365
		64	2	3361	0	8794	346
		128	2	5985	0	17,004	339
	128	32	2	4437	0	9371	333
		64	2	10,287	0	17,589	324
		128	2	11,904	0	34,000	320
	256	32	2	11,371	0	18,740	325
		64	2	17,356	0	35,183	294
		128	2	24,109	0	67,996	304

[1] Slice Registers.

5.3. BCAM Versus TCAM

This subsection compares the proposed BCAM and TCAM across different optimization modes, focusing on FPGA resource utilization and maximum clock frequency. The comparison is based on BCAM and TCAM architectures with depths of 128 and 256, while keeping the key width fixed at 128-bit and the value width at 8-bit. Figure 9 compares BCAMs and TCAMs in BF mode in terms of LUT and slice register consumptions, as well as frequency. The LUT consumption is a combination of LUT logic and LUTRAMs, which are shown in the chart separately. As expected, LUTRAM usage in TCAM is approximately twice that of BCAM, reflecting the increased memory demand for storing mask bits. In contrast, LUT logic usage is nearly identical for both architectures, with only minor differences, since both require the same logic for sequential memory iteration and matching operations. Accordingly, TCAM consumes more total LUTs than BCAM. TCAM also requires 24% more registers for 128-depth configurations and 14% more for 256-depth configurations than BCAM. Moreover, BCAM achieves slightly higher maximum operating frequencies, surpassing TCAM by approximately 3% in the 128-depth case and 8% in the 256-depth case.

In the BL mode, the resource utilization gap between BCAM and TCAM becomes more pronounced, as shown in Figure 10. While LUTRAM usage follows the same trend as in the BF mode, LUT logic consumption in TCAM is approximately 98% and 108% higher than in BCAM for depths of 128 and 256, respectively. This increase is attributed to the semi-parallel processing approach in the BL mode, which requires additional logic to handle and schedule the partitioned block memory units. Registers are also increasingly demanded for the TCAM to store more and wider intermediate signals in the pipeline. Despite these differences, the maximum operating frequencies of both architectures remain within a similar range.

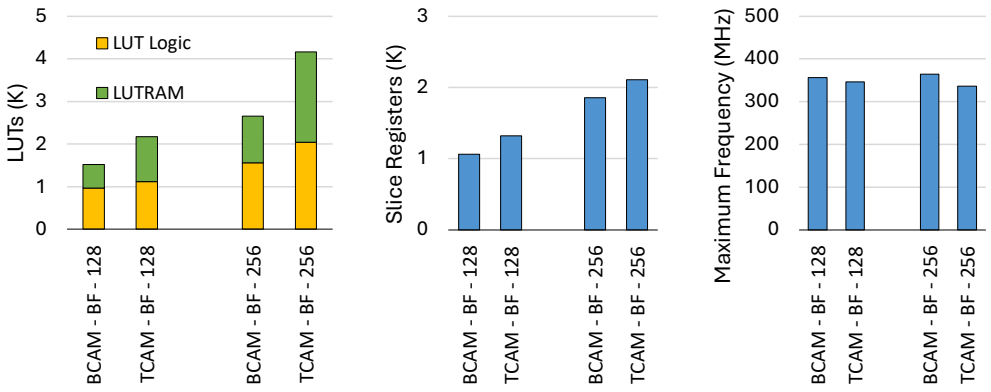


Figure 9. Comparison of the proposed BCAM and TCAM in the mode BF for 128 and 256 CAM depth (128-bit key, 8-bit value, 1-bit valid).

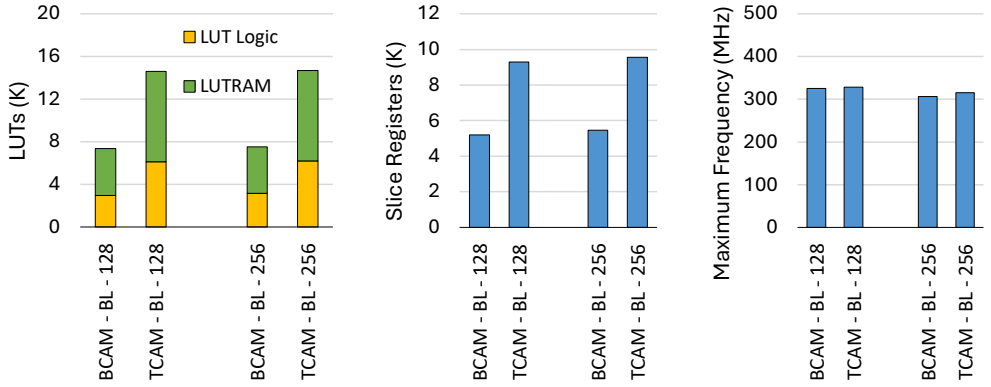


Figure 10. Comparison of the proposed BCAM and TCAM in the mode BL for 128 and 256 CAM depth (128-bit key, 8-bit value, 1-bit valid).

Figure 11 presents the comparison in HS mode, where LUTRAM is no longer used, making registers the primary resource for storage. The LUT logic and register utilization trends in HS mode follow the same pattern as in BL mode, but with higher overall values due to the increased parallelism and partitioning required for high-speed operation. The maximum clock frequency follows a similar trend to BL mode, varying from 300 MHz to 320 MHz.

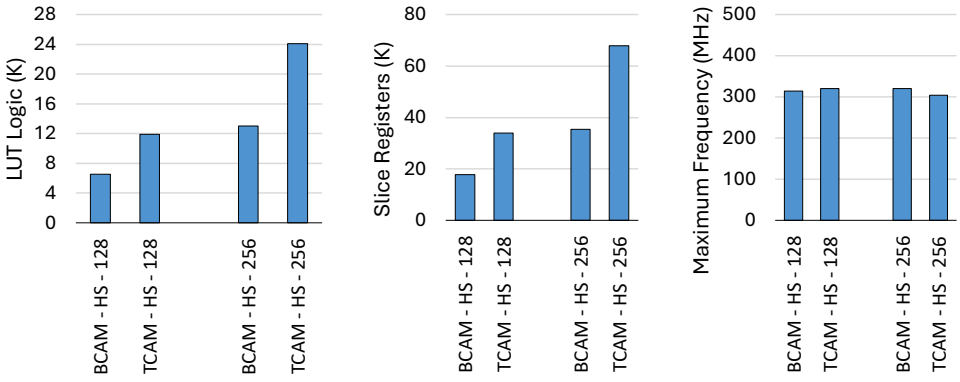


Figure 11. Comparison of the proposed BCAM and TCAM in the mode HS for 128 and 256 CAM depth (128-bit key, 8-bit value, 1-bit valid).

5.4. High-Speed (HS) Optimization Mode Variations

Furthermore, we compare the HS mode and its enhanced version, HS-H mode, across different CAM depths for the proposed TCAM design. Figure 12 presents a detailed

comparison of resource utilization, latency, and maximum clock frequency for both modes, using TCAM configurations with depths of 64, 128, and 256, while keeping the key width fixed at 128 bits. The results demonstrate that the HS-H mode achieves up to 7% higher efficiency in LUT utilization and up to 4% higher efficiency in register utilization compared to the HS mode. The most significant improvement is observed in the maximum clock frequency, which increases by up to 32.9% for a TCAM depth of 256. However, as expected, the latency increases by one cycle when the TCAM depth is 256.

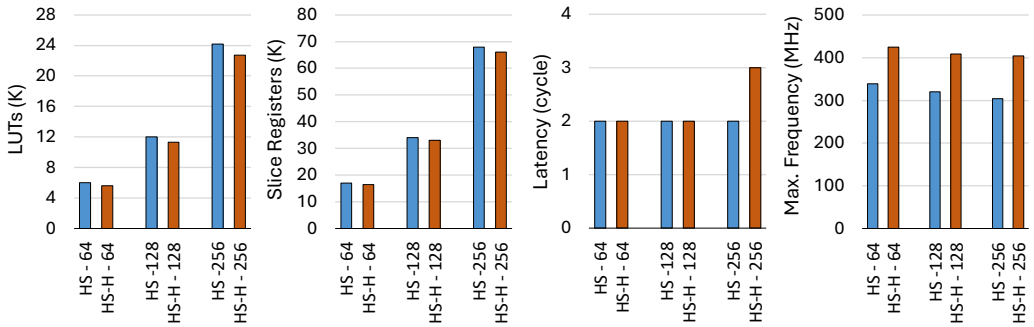


Figure 12. Comparison of the HS and HS-H modes for the proposed TCAM (128-bit key, 8-bit value, 1-bit valid).

5.5. Comparison

To evaluate the proposed HLSCAM architectures, we consider the HS-H mode, which offers improved resource efficiency and higher operating frequencies compared to the HS mode. The comparison considers key performance metrics, including FPGA resource utilization, which accounts for the combination of LUTRAMs, LUT logic, slice registers, maximum clock frequency, and throughput. Throughput TP is defined as the rate of lookup operations completed per second, which is measured in bits per second (bps) and calculated using Equation (3) [11]:

$$Throughput = f_{max} \times W_{CAM} \tag{3}$$

where the f_{max} is the maximum frequency and W_{CAM} is the key width of the CAM. Since CAM architectures in the literature are implemented on different FPGA devices, a normalized throughput (N.TP) metric is introduced to allow for a fair comparison. It is defined in Equation (4) [11]:

$$Normalized\ Throughput = Scaling\ Factor \times Throughput \tag{4}$$

where *Scaling Factor* is a coefficient that compensates for differences in FPGA technology speed, making performance metrics more comparable across different devices. The *Scaling Factor* is computed using Equation (5) [11]:

$$Scaling\ Factor = \frac{Technology\ (\text{nm})}{40\ (\text{nm})} \times \frac{1.0}{VDD} \tag{5}$$

where *Technology* refers to the FPGA CMOS technology size and the VDD refers to the corresponding supply voltage. The Virtex-6 family is used as the baseline FPGA, as it is fabricated with a 40 nm process and operates at 1.0 V supply voltage. Table 3 lists the related FPGA parameters used to calculate the normalized throughput. It provides the relevant FPGA parameters, including those for Virtex-6, Kintex-7, Virtex-7, and Virtex Ultrascale+, enabling the calculation of scaling factors for each. The comparison results are compiled to reflect post-place-and-route implementations.

Table 3. Different FPGA parameters.

FPGA Family	CMOS Technology	VDD	Scaling Factor
Virtex-6	40 nm	1.0 V	1.0
Virtex-7, Kintex-7	28 nm	1.0 V	0.7
Virtex Ultrascale+	16 nm	0.85 V	0.47

Table 4 shows the results of the proposed HLSCAM architectures in the HS-H mode and other existing work. The comparison results are obtained after post-place-and-route implementation for both Virtex-7 (xc7v2000tflg1925-2) and Virtex Ultrascale+ (xcu280-fsvh2892-2l-e). Resource usage is reported in terms of LUTs, FFs, and BRAMs. Additionally, CLBs (Configurable Logic Blocks) are used for Virtex UltraScale+ device, while Slices are reported for older FPGA families. The TP and $N.TP$ represent the throughput and normalized throughput, respectively. However, a direct quantitative comparison is still challenging, as not all works report the same set of resource utilization metrics, as well as variations in FPGA devices, synthesis tools, and experimental setups. Additionally, since each FPGA slice contains multiple LUTs and registers, comparing slices alone does not fully reflect differences in resource efficiency across different architectures. Despite these challenges, a qualitative comparison is made by analyzing key performance characteristics.

Some existing TCAM implementations, such as Scalable TCAM and Zi-TCAM, report high slice usage, whereas the proposed architectures demonstrate more efficient resource allocation while achieving significantly higher operating frequencies. Unlike Z-TCAM and UE-TCAM, which rely on BRAM-based storage, the proposed architectures avoid BRAM dependency, improving scalability for high-speed large CAM sizes. Since the HLSCAM uses high-level partitioning, particularly in the HS mode, using BRAM would lead to underutilized memory blocks.

The proposed HLSCAM architectures achieve higher maximum operating frequencies than most existing designs, such as Hierarchical TCAM (109 MHz), Zi-TCAM (38 MHz), and HP-TCAM (118 MHz). Compared to G-AETCAM, which operates at 358 MHz, the proposed 512 $\times$ 36 TCAM architecture sustains competitive clock speeds while handling larger key widths and CAM depths. Compared to D-TAM, the corresponding HLSCAM configuration exhibits slightly lower operating frequency, resulting in lower throughput. Nevertheless, in the wider CAMs such as 256 $\times$ 128, the HLSCAM achieves up to 31.18 Gbps normalized throughput.

For similar CAM sizes, HLSCAM achieves higher operating frequencies than Xilinx CAM IPs. For TCAM 256 $\times$ 128, HLSCAM runs at 404 MHz versus 316 MHz. For TCAM 512 $\times$ 36, it operates at 365 MHz compared to 323 MHz. This results in greater normalized throughput 24.30 Gbps versus 19.04 Gbps and 6.12 Gbps versus 5.47 Gbps. However, Xilinx CAM IPs use BRAM for balanced resource utilization, while HLSCAM relies on distributed memory, increasing CLB usage.

Although HLSCAM TCAM and BCAM consistently achieve significantly higher maximum frequencies and related throughput TP than most existing works, the resulting normalized throughput $N.TP$ gains are not proportionally higher. This is due to the scaling factor introduced by the normalization equation, which is not ideal for accurately reflecting performance improvements across different architectures and configurations.

Table 4. Resource Utilization and Throughput Comparison with Existing FPGA-based CAM Architectures

Work	CAM Size ($D \times W$)	Device	LUTs	SRs [1]	BRAMs (18k/36k)	Slices/ CLBs [2]	f_{max} (MHz)	TP (Gbps)	N.TP [3] (Gbps)
HP-TCAM [7]	512 × 36	Virtex-6	6546	2670	0/56	NA	118	4.25	4.25
UE-TCAM [8]	512 × 36	Virtex-6	3652	1758	0/32	NA	202	7.27	7.27
FMU-BiCAM [10]	512 × 36	Virtex-6	1257	732	64/0	NA	284	10.22	10.22
DURE [11]	512 × 36	Virtex-6	NA	NA	0/0	1668	335	12.06	12.06
D-TCAM [12]	512 × 36	Virtex-6	NA	NA	0/0	968	460	16.56	16.56
REST [15]	72 × 28	Kintex-7	130	390	0/1	77	50	1.40	0.98
LH-CAM [18]	64 × 36	Virtex-6	1259	70	0/0	NA	340	12.24	12.24
G-AETCAM [19]	64 × 36	Virtex-6	3067	4672	0/0	NA	358	12.89	12.89
Sca. TCAM [27]	1024 × 150	Virtex-7	NA	37,556	0/0	20,526	199	29.85	20.9
Hie. TCAM [28]	512 × 72	Virtex-6	16,720	NA	0/0	NA	109	7.85	7.85
Zi-TCAM [29]	512 × 36	Virtex-6	24,551	NA	0/0	NA	38	1.37	1.37
Z-TCAM [30]	512 × 36	Virtex-6	NA	NA	0/40	1116	159	5.72	5.72
Mul. TCAM [31]	512 × 32	Virtex-6	1,515	1593	0/16	NA	237	7.58	7.58
Xilinx BCAM [32]	256 × 128	Virtex-US+	2583	4159	0/6	641	383	49.04	23.05
Xilinx BCAM [32]	512 × 36	Virtex-US+	1972	4159	0/2	460	400	14.41	6.77
Xilinx BCAM [32]	1024 × 150	Virtex-US+	2950	4488	0/12	723	333	49.95	23.48
Xilinx TCAM [33]	256 × 128	Virtex-US+	15,669	19,033	0/48	3361	316	40.51	19.04
Xilinx TCAM [33]	512 × 36	Virtex-US+	19,033	12,407	0/24	2332	323	11.63	5.47
Xilinx TCAM [33]	1024 × 150	Virtex-US+	21,113	25,900	0/60	4873	320	47.98	22.58
HLSCAM BCAM (proposed)	256 × 128	Virtex-7	11,786	33,776	0/0	8410	335	42.88	30.02
	512 × 36	Virtex-7	8138	19,722	0/0	5872	358	12.91	9.04
	1024 × 150	Virtex-7	54,735	157,331	0/0	34,169	298	44.74	31.32
	256 × 128	Virtex-US+	12,841	33,297	0/0	3933	423	54.10	25.43
	512 × 36	Virtex-US+	7194	19,461	0/0	2402	392	14.11	6.63
	1024 × 150	Virtex-US+	54,572	155,748	0/0	15,881	374	56.03	26.34
HLSCAM TCAM (proposed)	256 × 128	Virtex-7	22,724	66,329	0/0	14,719	348	44.54	31.18
	512 × 36	Virtex-7	12,755	38,663	0/0	10,435	339	12.21	8.55
	1024 × 150	Virtex-7	105,055	311,431	0/0	66,213	315	47.20	33.04
	256 × 128	Virtex-US+	22,642	66,119	0/0	6632	404	51.70	24.30
	512 × 36	Virtex-US+	13,086	38,266	0/0	5805	365	13.12	6.12
	1024 × 150	Virtex-US+	105,147	309,613	0/0	29,766	333	49.97	23.48

[1] Slice Registers; [2] CLB for Virtex-US+ and Slice for older FPGA families; [3] Normalized Throughput.

6. Discussion

The proposed HLSCAM framework introduces a flexible, high-performance CAM implementation in HLS, enabling users to define their CAM table sizes according to specific application requirements. Unlike fixed hardware designs, this adaptability allows for customized trade-offs between resource utilization and performance, making it a versatile solution for FPGA-based packet processing.

To further enhance flexibility, HLSCAM provides multiple optimization strategies, offering users a range of configurations to balance throughput, latency, and resource consumption. The three primary operational modes (BF, BL, and HS) are designed to meet diverse performance demands. The BF mode is particularly suited for resource-constrained scenarios, offering a highly compact implementation. In contrast, the HS mode prioritizes minimal latency by leveraging parallel registers at the cost of higher resource consumption. The BL mode provides a balanced trade-off between latency and resource efficiency through partitioning memory blocks into multiple LUTRAMs. The *PF* parameter plays a critical role in this trade-off, acting as a control knob to shift the design towards either a resource-efficient or low-latency CAM table.

Furthermore, we introduced an improved version of the HS mode, termed HS-H, which retains the fundamental parallel processing benefits of the HS mode but adopts

a hierarchical structure. This architectural enhancement significantly reduces intermediate signal propagation, leading to up to 7% LUT reduction and up to 4% reduction in register utilization, while also achieving up to 32.9% improvement in maximum operating frequency.

HLSCAM is specifically tailored for dataplane packet processing, distinguishing it from traditional CAM architectures. The entry format is adapted to accommodate network rule storage, incorporating a valid bit to determine entry validity and a value field for packet processing upon a successful match. For TCAM implementations, mask bits are explicitly defined for the entire key width, leading to higher memory consumption compared to conventional CAM solutions. As a result, for the same CAM dimensions, HLSCAM requires more than double the memory units used in existing works.

Despite its relatively higher FPGA resource consumption, HLSCAM outperforms many competitors in terms of frequency and throughput, particularly in the 512×36 configuration. Additionally, it delivers substantial throughput in configurations such as 256×128 and 1024×150. A key factor contributing to HLSCAM's resource consumption is its HLS-based implementation. Unlike HDL-based solutions, which allow fine-grained control over hardware resources, HLS-based designs introduce abstraction layers that can lead to additional overhead. While this abstraction enhances design flexibility, it often results in less efficient resource utilization than RTL implementations. This inefficiency becomes more pronounced in larger designs that rely on deep pipelining, where mapping high-level behavioral code to FPGA logic and memory introduces complexity. One way to mitigate this issue is to reduce memory partitioning, simplifying the design to lower memory and logic overhead. A hybrid approach that combines semi-partitioned memory with a hash-based lookup scheme could further optimize resource efficiency by reducing the number of required comparisons. However, potential hash conflicts must be carefully considered. Investigating the feasibility and effectiveness of these approaches, along with their impact on resource utilization and lookup performance, remains an area for future research.

Nevertheless, HLSCAM enables the easy generation of arbitrary CAM configurations, while automatic scheduling mechanisms optimize logic placement and pipelining to achieve high operating frequencies. These features are beneficial for high-speed networking and packet processing applications, including rule-based packet flow filtering, Intrusion Detection Systems (IDS), In-band Network Telemetry (INT), timestamping, and packet classification.

7. Conclusions

This paper presented HLSCAM, a fine-tuned HLS-based implementation of BCAM and TCAM for high-speed packet processing on FPGAs. By leveraging High-Level Synthesis, the proposed approach enables configurable, scalable, and resource-efficient CAM architectures, reducing development complexity compared to traditional RTL-based implementations. The design methodology employed different design parameters to provide a fine-tuned FPGA parallelism management. This approach, combined with a structured implementation, allowed the synthesis tool to decrease the critical path. The Brute Force (BF), Balanced (BL), and High-Speed (HS) modes offer a spectrum of design trade-offs between latency, resource utilization, and throughput, allowing customization for different networking applications.

The evaluation results demonstrated that HLSCAM achieves significantly higher operating frequencies than most existing FPGA-based CAM designs, reaching up to 423 MHz. The HS mode achieved the lowest lookup latencies, while the BL mode provided a balance between resource efficiency and performance. Despite higher resource consumption due to

extensive partitioning and parallelism, the proposed architectures delivered competitive throughput across various configurations. A comparison with previous work highlighted HLSCAM's advantages in flexibility, scalability, and high-speed performance, particularly in large-scale CAM applications. By avoiding BRAM dependency, the proposed architectures improve scalability, making them well-suited for high-performance packet classification and filtering.

Future work will further optimize HLS-based CAM architectures to save resources while maintaining rapid lookup. Approaches like hash-based methods could prevent CAMs from comparing with all entries to reduce logic and the need for memory partitioning, thereby improving efficiency. Additionally, integrating these CAMs into real-world network processing pipelines will provide further insights into their applicability in next-generation networking systems.

Author Contributions: Conceptualization, M.A. and T.O.-B.; methodology, M.A., T.O.-B. and Y.S.; software, M.A.; validation, M.A., T.O.-B. and Y.S.; formal analysis, M.A., T.O.-B. and Y.S.; investigation, M.A., T.O.-B. and Y.S.; resources, M.A.; data curation, M.A.; writing—original draft preparation, M.A. and T.O.-B.; writing—review and editing, M.A., T.O.-B. and Y.S.; visualization, M.A.; supervision, T.O.-B. and Y.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by NSERC Kaloom-Intel-Noviflow. The grant number is IRCPJ-548237-18 CRSNG.

Data Availability Statement: The data presented in this study will be available on Github at https://github.com/abbasmollaei/HLSCAM accessed on 24 April 2025.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following list of acronyms and abbreviations is provided to improve the readability of the paper.

FPGA	Field Programmable Gate Logic
ASIC	Application-Specific Integrated Circuit
HLS	High-Level Synthesis
RTL	Register-Transfer Level
CAM	Content-Addressable Memory
BCAM	Binary CAM
TCAM	Ternary CAM
BF	Brute-force optimization
BL	Balanced optimization
HS	High-Speed optimization
HS-H	High-Speed optimization (enHanced version)
PD	Partition Depth
PF	Partition Factor
PVIF	Parallel Valid Index Finder
RAM	Random-Access Memory
LUT	Look Up Table
LUT5/LUT6	LUT 5-input/LUT 6-input
LUTRAM	Distributed Logic RAM
BRAM	Block RAM in FPGA
URAM	Ultra RAM in FPGA
FF	Flip-Flop
SR	Slice Register
MUX	Multiplexer
maddr/addr	Memory address

wr	Write enable
e_i/e_j	i_{th}/j_{th} entry
TP	Throughput
N.TP	Normalized Throughput
f_{max}	Maximum Operating Frequency
CLB	Configurable Logic Block
Gbps	Gigabits per second
IP	Intellectual Property
IDS	Intrusion Detection System
INT	In-band Network Telemetry
ACL	Access Control List
NAT	Network Address Translation
AGF	Access Gateway Function
UPF	User Plane Function
QoS	Quality-of-Service

References

1. Song, X.; Guo, Z. An Implementation of Reconfigurable Match Table for FPGA-Based Programmable Switches. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2024**, *32*, 2121–2134. [CrossRef]
2. Dhayalakumar, M.; Sk, N.M. TeRa: Ternary and Range based packet classification engine. *Integration* **2024**, *96*, 102153. [CrossRef]
3. Irfan, M.; Vipin, K.; Qureshi, R. Accelerating DNA Sequence Analysis using Content-Addressable Memory in FPGAs. In Proceedings of the 2023 IEEE 8th International Conference on Smart Cloud (SmartCloud), Tokyo, Japan, 16–18 September 2023; pp. 69–72. [CrossRef]
4. Kaplan, R.; Yavits, L.; Ginosar, R.; Weiser, U. A Resistive CAM Processing-in-Storage Architecture for DNA Sequence Alignment. *IEEE Micro* **2017**, *37*, 20–28. [CrossRef]
5. Garzón, E.; Yavits, L.; Teman, A.; Lanuzza, M. Approximate Content-Addressable Memories: A Review. *Chips* **2023**, *2*, 70–82. [CrossRef]
6. Irfan, M.; Sanka, A.I.; Ullah, Z.; Cheung, R.C. Reconfigurable content-addressable memory (CAM) on FPGAs: A tutorial and survey. *Future Gener. Comput. Syst.* **2022**, *128*, 451–465. [CrossRef]
7. Ullah, Z.; Jaiswal, M.K.; Cheung, R.C.C. Design space explorations of Hybrid-Partitioned TCAM (HP-TCAM). In Proceedings of the 2013 23rd International Conference on Field programmable Logic and Applications, Porto, Portugal, 2–4 September 2013; pp. 1–4. [CrossRef]
8. Ullah, Z.; Jaiswal, M.K.; Cheung, R.C.; So, H.K. UE-TCAM: An ultra efficient SRAM-based TCAM. In Proceedings of the TENCON 2015—2015 IEEE Region 10 Conference, Macao, China, 1–4 November 2015; pp. 1–6. [CrossRef]
9. Abdelhadi, A.M.S.; Lemieux, G.G.F. Deep and narrow binary content-addressable memories using FPGA-based BRAMs. In Proceedings of the 2014 International Conference on Field-Programmable Technology (FPT), Shanghai, China, 10–12 December 2014; pp. 318–321. [CrossRef]
10. Qazi, A.; Ullah, Z.; Hafeez, A. Fast Mapping and Updating Algorithms for a Binary CAM on FPGA. *IEEE Can. J. Electr. Comput. Eng.* **2021**, *44*, 156–164. [CrossRef]
11. Ullah, I.; Ullah, Z.; Afzaal, U.; Lee, J.A. DURE: An Energy- and Resource-Efficient TCAM Architecture for FPGAs with Dynamic Updates. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *27*, 1298–1307. [CrossRef]
12. Irfan, M.; Ullah, Z.; Cheung, R.C.C. D-TCAM: A High-Performance Distributed RAM Based TCAM Architecture on FPGAs. *IEEE Access* **2019**, *7*, 96060–96069. [CrossRef]
13. Shi, Z.; Yang, H.; Liu, R.; Yan, J.; Qiao, P.; Wang, B. ME-TCAM: Memory-Efficient Ternary Content Addressable Memory Based on Multipumping-Enabled LUTRAM on FPGA. In Proceedings of the 2020 International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI), Sanya, China, 4–6 December 2020; pp. 38–42. [CrossRef]
14. Irfan, M.; Ullah, Z.; Chowdhury, M.H.; Cheung, R.C.C. RPE-TCAM: Reconfigurable Power-Efficient Ternary Content-Addressable Memory on FPGAs. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2020**, *28*, 1925–1929. [CrossRef]
15. Ahmed, A.; Park, K.; Baeg, S. Resource-Efficient SRAM-Based Ternary Content Addressable Memory. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2017**, *25*, 1583–1587. [CrossRef]
16. Zou, Q.; Zhang, N.; Guo, F.; Kong, Q.; Lv, Z. Multi-region SRAM-Based TCAM for & Longest Prefix. In Proceedings of the Science of Cyber Security: 4th International Conference, SciSec 2022, Matsue, Japan, 10–12 August 2022; Revised Selected Papers; Springer: Berlin/Heidelberg, Germany, 2022; pp. 437–452. [CrossRef]

17. Ur Rehman, N.; Mujahid, O.; Ullah, Z.; Hafeez, A.; Fouzder, T.; Ibrahim, M. Power Efficient FPGA-based TCAM Architecture by using Segmented Matchline Strategy. In Proceedings of the 2019 International Conference on Advances in the Emerging Computing Technologies (AECT), Al Madinah Al Munawwarah, Saudi Arabia, 10 February 2020; pp. 1–4. [CrossRef]
18. Ullah, Z. LH-CAM: Logic-Based Higher Performance Binary CAM Architecture on FPGA. *IEEE Embed. Syst. Lett.* **2017**, *9*, 29–32. [CrossRef]
19. Irfan, M.; Ullah, Z. G-AETCAM: Gate-Based Area-Efficient Ternary Content-Addressable Memory on FPGA. *IEEE Access* **2017**, *5*, 20785–20790. [CrossRef]
20. Krishnan, A.S.; Sivalingam, K.M.; Shami, G.; Lyonnais, M.; Wilson, R. Flow classification for network security using P4-based Programmable Data Plane switches. In Proceedings of the 2023 IEEE 9th International Conference on Network Softwarization (NetSoft), Madrid, Spain, 19–23 June 2023; pp. 374–379. [CrossRef]
21. Khraisat, A.; Gondal, I.; Vamplew, P.; Kamruzzaman, J. Survey of intrusion detection systems: Techniques, datasets and challenges. *Cybersecurity* **2019**, *2*, 20. [CrossRef]
22. Xu, Z.; Lu, Z.; Zhu, Z. Information-Sensitive In-Band Network Telemetry in P4-Based Programmable Data Plane. *IEEE/ACM Trans. Netw.* **2024**, *32*, 5081–5096. [CrossRef]
23. Paim, E.C.; Schaeffer-Filho, A. P4-TURNet: Enabling NAT Traversal through P2P Relay Networks and Programmable Switches. In Proceedings of the 2024 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Natal, Brazil, 5–7 November 2024; pp. 1–6. [CrossRef]
24. Ihle, F.; Lindner, S.; Menth, M. P4-PSFP: P4-Based Per-Stream Filtering and Policing for Time-Sensitive Networking. *IEEE Trans. Netw. Serv. Manag.* **2024**, *21*, 5273–5290. [CrossRef]
25. Wen, Z.; Yan, G. HiP4-UPF: Towards High-Performance Comprehensive 5G User Plane Function on P4 Programmable Switches. In Proceedings of the 2024 USENIX Annual Technical Conference (USENIX ATC 24), Santa Clara, CA, USA, 10–12 July 2024; pp. 303–320.
26. Abbasmollaei, M.; Ould-Bachir, T.; Savaria, Y. Normal and Resilient Mode FPGA-based Access Gateway Function Through P4-generated RTL. In Proceedings of the 2024 20th International Conference on the Design of Reliable Communication Networks (DRCN), Montreal, QC, Canada, 6–9 May 2024; pp. 32–38. [CrossRef]
27. Jiang, W. Scalable Ternary Content Addressable Memory implementation using FPGAs. In Proceedings of the Architectures for Networking and Communications Systems, San Jose, CA, USA, 21–22 October 2013; pp. 71–82. [CrossRef]
28. Qian, Z.; Margala, M. Low power RAM-based hierarchical CAM on FPGA. In Proceedings of the 2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14), Cancun, Mexico, 8–10 December 2014; pp. 1–4. [CrossRef]
29. Irfan, M.; Ullah, Z.; Cheung, R.C.C. Zi-CAM: A Power and Resource Efficient Binary Content-Addressable Memory on FPGAs. *Electronics* **2019**, *8*, 584. [CrossRef]
30. Ullah, Z.; Jaiswal, M.K.; Cheung, R.C.C. Z-TCAM: An SRAM-based Architecture for TCAM. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2015**, *23*, 402–406. [CrossRef]
31. Ullah, I.; Ullah, Z.; Lee, J.A. Efficient TCAM Design Based on Multipumping-Enabled Multiported SRAM on FPGA. *IEEE Access* **2018**, *6*, 19940–19947. [CrossRef]
32. AMD/Xilinx Inc. *LogiCORE IP Binary Content-Addressable Memory (BCAM) v2.6: Product Guide (PG317)*, AMD, v2.6 ed.; Vivado Design Suite 2023.2; AMD/Xilinx: Santa Clara, CA, USA, 2023.
33. AMD/Xilinx Inc. *LogiCORE IP Ternary Content-Addressable Memory (TCAM) v2.6: Product Guide (PG318)*, AMD, v2.6 ed.; Vivado Design Suite 2023.2; AMD/Xilinx: Santa Clara, CA, USA, 2023.