applied sciences

MDPI

Article

MAToC: A Novel Match-Action Table Architecture on Corundum for 8 × 25G Networking

Jiawei Lin [1,2], Zhichuan Guo [1,2,*] and Xiao Chen [1,2]

[1] National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, No. 21, North Fourth Ring Road, Haidian District, Beijing 100190, China
[2] School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, China
* Correspondence: guozc@dsp.ac.cn

Abstract: Packet processing offloads are increasingly needed by high-speed networks. This paper proposes a high throughput, low latency, scalable and reconfigurable Match-Action Table (MAT) architecture based on the open source FPGA-based NIC Corundum. The flexibility and capability of this scheme is demonstrated by an example implementation of IP layer forwarding offload. It makes the NIC work as a router that can forward packets for different subnet and virtual local area networks (VLAN). Experiments are performed on a Zynq MPSoC device with two QSFPs and the results show that it can work at line rate of 8 × 25 Gbps (200 Gbps), within a maximum latency of 76 nanoseconds. In addition, a high-performance MAT pipeline with full-featured, resource-efficient TCAM and a compact frame merging deparser are presented.

Keywords: fast packet processing; FPGA Offloading; match action table; TCAM

check for updates

Citation: Lin, J.; Guo, Z.; Chen, X. MAToC: A Novel Match-Action Table Architecture on Corundum for 8 × 25G Networking. *Appl. Sci.* **2022**, *12*, 8734. https://doi.org/10.3390/app12178734

Academic Editor: Alexandros-Apostolos Boulogeorgos

Received: 4 July 2022
Accepted: 29 August 2022
Published: 31 August 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

With the exponential growth of Internet traffic, pure software network stacks with traditional ASIC NICs can hardly handle all incoming packets at high throughput. Hardware offloading has become an efficient way to catch up with the growing throughput and latency demands of modern applications in data centers [1]. A variety of protocols and jobs can be offloaded on hardware devices, which provide higher throughput up to line rate and lower latency down to sub-microsecond. Fast packet processing serves as an infrastructure for various types of offloading engines that require the processing of packetized data.

However, there are few solutions for packet processing on open-source prototyping platforms, and there is no way to support multiple connection ports while achieving 200 Gbps line rate processing. To meet the need for packet processing on an open-source platform for 200 Gbps network, we present a compact MAT-based offloading scheme for an open-source NIC named Corundum [2] and present an example implementation called MAToC that can perform time-to-live (TTL) decrement, MAC replacement, check summing, encapsulation of VLAN tag, and packets forwarding according to the destination IP address. The proposed scheme is scalable to multiple channels of either the transmit, the receive side or both. It is capable of working at realistic line rate in a link mode of 8 × 25G with two QSFPs. The delay of the processing scheme is tens of nanoseconds, which is low enough to meet the latency critical applications. The primary contributions in this paper are as follows:

- A MAT-based offloading scheme with throughput up to 200 Gbps is presented. It can handle traffic from multiple channels, processing and forwarding them to the designated ports according to reconfigurable matching tables.
- A resource efficient, low update latency, full featured TCAM with search-prior mechanism is implemented. The proposed TCAM offers decoupled search input and match

output interfaces, along with read and write interfaces for match rules. The delay for writing a rule is reduced to 32 × 2 clock cycles, whose preparation time is eight times reduced compared to the initial implementation. It utilizes the 32-deep LUT RAM to implement a 5-bit TCAM unit, achieving a small SRAM/TCAM bit ratio of 32:5 that can guarantee an efficient resource utilization. The search-prior mechanism guarantees the search performance.

- A compact frame merging deparser is proposed for merging of the processed header and payload. It can handle the concatenation of two traffic sources with different interface widths and variable lengths at run time. Moreover, the compact design achieves better timing performance than a straightforward implementation.

2. Background

There are many kinds of network interface controller (NIC) cards for high speed networks. They can generally be divided into dedicated ASIC NICs, network processors and reconfigurable NICs. We mainly focus on programmable devices that can be further divided into three categories. First is pure FPGA NIC, which contains solely FPGA for MAC/PHY and PCIe transmit and receive engines, such as Xilinx Alveo series SmartNIC [3]. Second is AFU NIC, which uses a conventional net device for MAC/PHY and PCIe engines while using FPGA or reconfigurable ASIC as acceleration function units (AFUs). Examples include Intel FPGA Programmable Acceleration Cards (PAC) [4] and Xilinx 8000 series Ethernet adapters [5]. Third is SoC NIC, which is much like the first kind but equipped with a multi-functional system-on-chip (SoC) that contains not only FPGA, but also general processors, graphic processing unit (GPU), memory subsystem, etc. Examples include Xilinx Zynq MPSoC and VERSAL [6].

Generally, AFU NIC provides hardware frameworks for packet processing, which saves labor on implementations of the complicated data path and packet transceiver engines, etc. Most of them are equipped with dedicated software packages, for example, the powerful user-space driver DPDK is provided for Intel PACs [4] and an application acceleration software named Onload [5] was developed for Xilinx 8000 series FPGA. However, these designs have limitations in flexibility for programming and extensibility for protocol design. For example, the XL710 net device on N3000 PAC [4] will filter out the malformed packets that do not comply with the Ethernet protocol. For exploration purposes, we turn to the other programmable NICs, and SoC NICs specifically. SoC NICs are much more functional than pure FPGA NICs and have more expansibility, providing a broader space for exploration.

Corundum [2] is an open-source FPGA-based NIC, which has splendid programmability, remarkable performance and efficient resource utilization. It is equipped with a vast amount of transmit and receive queues along with precise timing protocol (PTP) offloading, by which means it is capable of providing precise time division multiple access (TDMA) in the optical switch. It is also a wonderful prototyping platform for NIC functions and on-board packet processing applications development. With a framework module provided, substantial data and control interfaces are exposed to user logic. There are a few explorations of the potential of Corundum's flexibility. PANIC [7,8] is a multi-tenant isolation NIC equipped with a switch-based offload network that are all implemented on Corundum. Menshen [9] is an isolation extension for Reconfigurable Match Tables (RMT) packet processing pipeline, using Corundum as verification platform. Here, we will explore more interesting tasks performed on multiple Ethernet connection ports.

PANIC [7,8] uses switches and schedulers for chaining multiple offloads, running different applications simultaneously in multi-tenant servers. This is a novel way to support offloads with different throughput, by which means head-of-line blocking can be avoided in some ways. However, PANIC is not suitable for fast packet processing when the purpose is offloading certain simple functions or protocols. Besides, those tasks running below the line rate should be placed on host machines because they would not cause significant CPU

overhead when the traffic amount is rare. Otherwise, PANIC will not help in any way if many packets are passing through these inefficient offloads.

Our example implementation of MAToC is basically a programmable router. Cerovi et al. [10] divided the work of fast packet processing on routers into fast path and slow path in terms of the required speed and latency. They summarized most of the jobs in both paths. General operations in the fast path include validity checks, TTL decrement, packet forwarding and classification, most of which are also offloaded in our MAT pipeline. Complicated operations such as fragmentation, error packet handling and offloading for certain protocols like ARP and ICMP are put in the slow path. Their survey of the fast packet processing solutions mainly focuses on the Click modular router and lacks a perspective from register transfer level (RTL). In their cases, only limited offloads can be deployed yet without an insight into resources and timing.

The proposed packet processing scheme is based on MAT, which is a primitive invention in Programming Protocol-Independent Packet Processors (P4) [11]. P4 architecture is the most popular fashion for packet processing framework with its higher abstraction in design procedures, independence from specific protocols, and most importantly, merchant chips and devices, along with vast amounts of compilers, testers and optimizers. But unfortunately, P4 associated tools can hardly be applied to FPGA. Although there are some attempts [12] to map P4 into RTL or directly into netlist, those methods are either with poor performance or obscure the synthesized results, while there are needs for arrangement of resources and optimization of timing performance [13]. Nevertheless, MAT has become a paradigm for packet processing. Pat Bosshart etc. [14] extended this idea to a reconfigurable match tables (RMT) model in their powerful switch chip, supporting the match on arbitrary header fields with fairly large match table capacity. More research on packet processing based on RMT has emerged. FlowBlaze [15] replaces MAT with an extended finite state machine (EFSM) table, which can match on several subsets of a search key. They add an extra table and global registers with corresponding update logic to support stateful packet processing. PANIC [7,8] also uses RMT pipeline as the spin of its on-chip network message processing architecture, and achieves 100 Gbps throughput. FlowBlaze [15] and many other MAT solutions [12,14] design the customized ALUs as execution units along with compact instruction sets to support all possible packet process operations. However, general ALUs are not the best way to exploit the parallelism of FPGAs. A better method is customizing offload engines for target functions, which would result in a system with less latency.

Packet processing always requires several types of match tables in different conditions, such as masked match (MM) tables for broadcast, exact match (EM) tables for in-put admission of gateway entry, longest prefix match (LPM) tables for IP address lookup, also known as a routing table, and multi-field matching for packet classification and receive side scale (RSS). Most of these tables can be implemented based on the ternary content address memory (TCAM), except EM tables that can be simplified as Binary CAM. Therefore, there is a huge need for efficient and scalable TCAM in fast packet processing. TCAM can be divided in terms of the resources used in FPGA; for example, Xilinx offers intellectual property (IP) cores of several kinds of CAMs that use block RAM (BRAM), Ultra RAM (URAM) [16]. IITCAM [17] is also a BRAM-based TCAM targeting Intel Altera's platform. It uses two layers of hierarchical indexing to reduce the storage resource. More variants such as [18] optimize power by bank selection and filtering. They can be classified as the first kind that are characterized by a large memory unit (generally 36K-bit for BRAM and 288K-bit for URAM). Besides, D-TCAM [19] and its improved variant FracTCAM [20] utilize dual-port distributed LUT RAM that has a medium unit size of 64×4 bits. In addition, there are TCAMs that make use of a shift register look-up table (SRL) [16], LUT [21], and flip-flops [22–24].

Generally speaking, BRAM-based TCAM has longer update latency and lower resource efficiency as its SRAM/TCAM bit ratio is at least $2^9/9$ [20]. FF-based TCAM is only suitable for shallow TCAMs where update latency is critical [22]. The differences between these designs are directly related to their unit sizes. For instance, Altera's syn-

thesis cookbook [22] provides the earliest FF-based (Flip-Flop) TCAM that makes use of general-purpose flip-flops (registers). The later G-AETCAM [23–25] makes improvements in variant metrics such as power, resource efficient, and update latency. However, none of these could improve scalability, as FFs scatter over the FPGA chip, and managing large amounts of these units would lead to a failure in timing closure or even an inability to complete the routing process. Among all these TCAM designs, FracTCAM [20] is adopted as the infrastructure of our proposed MAT pipeline after the trade-off between resource utilization and scalability.

In the next pages, Section 3 presents the details of the proposed scheme and submodules, as well as its advantages. Section 4 summarizes the optimization of resource utilization and makes an analysis of throughput and timing performance. Section 5 lists and analyzes the experimental results. Section 6 concludes this paper and discusses future work.

3. Implementation

3.1. Overview

Corundum [2] can be configured to different link modes depending on the boards' capability. For the purpose of exploiting the maximum throughput with as many ports as possible, the 8 × 25G link mode is adopted in our implementation of MAToC. A block diagram in Figure 1 illustrates the proposed packet processing scheme. The top-level module of MAToC mainly contains infrastructure that constructs data paths for channels. Firstly, all the receive (RX) channels are adapted to the same data width of 512 bits, which is important to maintain the original throughput. This will be explained in Section 4.3. Then, the parsers would separate the packet into two parts. One part includes metadata that needs to be processed, which would be transferred to the MAT pipeline via a multiplexer. The other part includes the remaining payloads, which are saved in packet buffers and wait to be merged by the deparsers. The red lines in this figure marking the data are transmitted in AXI4-Stream protocol, while the green lines are in AXI4-Lite and the black lines are self-defined for certain uses, such as transferring of table config messages. The labels above the line mark different flows. For example, TX means transmitting from host to Ethernet ports, while RX means receiving in the opposite direction. The other labels will be explained in related modules.

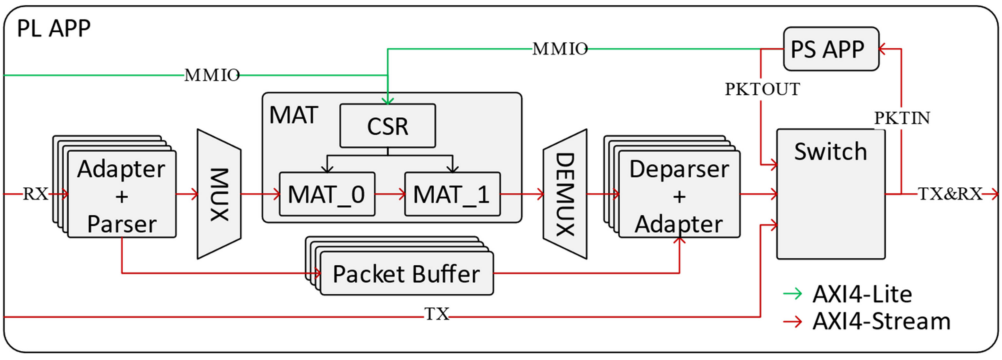

Figure 1. The proposed MAT architecture.

The MAT block undertakes most of the packet processing jobs. This block contains a number of offload engines implemented as reconfigurable match-action tables. They are cascaded with each other and the metadata is processed stage by stage in these submodules. Offloads could be either interdependent or not. For examples, the latter offload could take the destination MAC address (DMAC) as search key in its match table, while the DMAC could be replaced by a previous offload. Alternatively, the latter offload could be activated or disabled by the processing result from the previous ones. The exact structure of our proposed offload is shown in Sections 3.3 and 3.4. In addition, there are control and status registers (CSRs) to provide functions of the control plane, i.e., inserting, removing and querying table entries in MATs. The specific procedures of configuring are as follows. At

first, user calls a utility to config an offload with certain match and action rules. Then the utility writes the config data into CSRs via the miscellaneous device provided by the driver module. This mechanism is called memory-mapped IO (MMIO) and the related buses are indicated with labels above in Figure 1. These labels show the path from host/controller to CSRs. The config data includes match table entries, action instruction and operation code (opcode). In the third step, the modification of opcode registers would trigger the config process of read/write logic, which will be introduced in Section 3.3.

Each MAT offload could be divided into match stage and action stage. The former is responsible for extracting some of the metadata as search key to the match table, and then fetching the matched opcode with operands from an instruction table. The latter stage is used to execute these operations, such as modifying some packet fields. Moreover, the modified metadata is transferred to deparsers on its original channel via a demultiplexer. The deparser is responsible for reconstructing packets from the processed metadata and the buffered payload. Last but not least, packets are forwarded by a switch according to the channel ID set in the metadata. The transmitted packets form PCIe to Ethernet MACs are also involved in this switch, but keeping its original channel ID. Therefore, they are actually transmitted transparently but have bandwidth competition with other channels. Apart from receive and transmit channels, there is another channel to the processing system (PS) of the Zynq SoC [6] on which Petalinux programs and bare metal applications are running.

As the survey in [10] has analyzed, a packet process should be divided in two ways. The first is called the fast path, where all the processes need to be carried out at line rate. The other, called slow path, involves complex protocol processes with more latency and lower speed, such as Address Resolution Protocol (ARP) reply. In our design, almost all blocks are contained in the fast path, except the PS block shown on the top right of Figure 1. The PS block is used for procedures in the slow path that are not time critical. Similar to an SDN design in [26], the input and output of PS could be defined as packet-in (PKTIN) and packet-out (PKTOUT). It is worth mentioning that the PKTIN is actually a separate channel from all the other channels, so it would not occupy the bandwidth of the fast path. In this case, there is no packet that goes through the entire fast path more than once.

Stephens, etc. [7] classified NICs according to the arrangement of offload engines. They pointed out that pipeline designs would increases latency for packets that do not need to be processed and the reconfigurable MAT designs are lacking support for complex offloads that would stall the pipeline, such as compression and encryption. In the proposed scheme, however, those packets that are not processed by a MAT can be bypassed using control signals transferred with the metadata to reduce latency. Besides, complex offloads that cannot run at line rate can be implemented in the PS block or simply put on the host machine, as the DMA RAM is large enough to buffer some packets. Even so, when the traffic that needs to be processed by the complex offloads are continually transferred in, there is no other way to handle all of them than to replace the offload engines with higher performance versions.

3.2. Parser

The parser's duties normally include packet resolution, validation checks and metadata generation [10,27]. In MAToC, metadata generation is substituted with the separation of header frames. The first transmitted frame of 512 bits width is defined as the packet header, which would be transferred separately to the MAT pipeline. The original frames are stored in packet buffers at the same time. The IPv4 destination address used as the search key in MAT would be contained in this header frame, as its offset is generally 26 bytes, i.e., within the interface width. The resolution logic of the parser simply identifies whether packets are IPv4 or IPv6 and whether they contain a VLAN tag, then forms a user-defined tag named pkt_type [3:0], transferring it along with the header frame. What it does is wait until the correct receiving cycles of certain protocol fields and then stores them in the output registers.

3.3. Match Stage

The primary jobs in match stage include fetching certain fields in header as search key in TCAM, reading out an instruction from the table entry indicated by the TCAM output match line, where the TCAM match table could be the bottleneck of the whole scheme. An earlier attempt of MAT mapped on FPGA [12], implementing BRAM-based TCAM with several cycles' response delay which causes pipeline bandwidth cut down by more than a half. Improvement in throughput is sought by duplicating tables, and as a result, the more BRAM they used, the more throughput they obtained. Therefore, a TCAM design with enough throughput is important, i.e., its response speed should be at least equal or larger than the input rate in packets per second (pps). To sum up, the point is that all modules on fast path should not stall while there is no backpressure signal asserted. In other word, bubbles in the pipeline should not exist because they can waste bandwidth and even cause frame drops when the input is not ready to accept incoming frames.

MAToC adopts FracTCAM [20] to build its match tables mainly because of its high efficiency in resource utilization and low latency of match rule update. As mentioned before, the SRAM/TCAM bit ratio of FracTCAM is 2^5:5, which means it consumes 32 bits in memory resources to implement a 5-bit wide TCAM. By contrast, this metric of BRAM-based TCAMs is at least 2^9:9 [20], around nine times that of the former. Even with the surrounding update logic, LUTRAM-based TCAM is a better choice in terms of resource utilization. Besides, Corundum also uses many BRAMs in transmit and receive data paths together with descriptor and completion modules when configured with many PCIe interfaces, which can hardly be decreased and will be further discussed in Section 4.2. In this case, BRAM becomes much rarer than other memory resources. FF-based TCAM is good choice for shallow tables and is easy to realize. However, the routing complexity will increase sharply with deeper configuration. LUT RAM is in the middle of BRAM and register in term of unit size and it is spared nearly 86% of total LUTRAM in the ZU19EG chip, apart from those used by Corundum and PS block design. Therefore, it is well suited for TCAM in our implementation. This is one of the reasons for adopting FracTCAM [20].

The original implementation example of FracTCAM [20] provided is not parametrized nor practical, i.e., one cannot config width and depth of TCAM tables easily at synthesis time. The initial update logic has not provided functions for inquiry, nor configuring rules for masked search key. Besides, its write operation costs 33×9 cycles to complete, which is absolutely unnecessary and would lead to a performance decrease of search operations as the two operations would compete for address channels. In our match stage, we proposed a parametrized and full-featured TCAM based on the genius idea of FracTCAM [20]. Our TCAM modules use Verilog instantiation parameters to generate the table at an arbitrary width and depth, and offers independent search and match interfaces with backpressure signals. Furthermore, we implemented a compact read/write logic for match rules. It takes $32 + 1$ cycles to response a read operation and $32 \times 2 + 1$ cycles to write, and the latency is only one cycle for matching of the search key. The extra 32 cycles of write operations come from the 32-bit shift right register, which is used to buffer the calculated match rules. Last but not least, according to the principle in [10], the search operations in the fast path always take precedence over write or read operations of match rules, by which means it can eliminate the search performance decrease caused by update or query of rules mentioned in [24]. In other words, the reconfiguring of our TCAM might be stalled by query or search operations, while the packet processing would never be stalled. This is applicable in the assumption that the total input rate would not rise to the upper bound of the pipeline's capability, which is 250 Mpps at 250 MHz clock domain. The assumption is always true as long as packet length is larger than 76 bytes. In that case, there are spare clock cycles for update logic to write match rules.

Figure 2 shows how the surrounding logic is designed for a single TCAM unit. A large TCAM is combined from many of these units instantiated from Xilinx configurable logic block (CLB) primitive RAM32M. This element is a 32-bit deep by 8-bit wide random-access memory implemented using the LUT memory resources, named SelectRAM. To clarify,

match rules are the actual data stored in LUT RAM, which has a mapping relationship with the masked search key in the form of data bits and care bits, e.g., a five bits width search key of 5′b00001 maps to a 32 bits match rule of 32′h00000002, and 5′b0000X maps to 32′h00000003, where X is a 'don't care' bit. The read and write logic blocks are responsible for this mapping work.

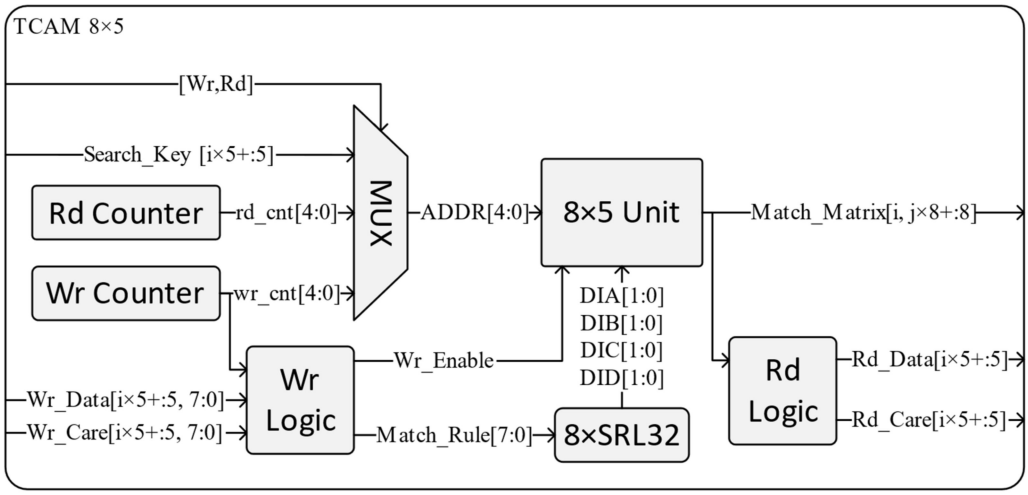

Figure 2. The proposed TCAM update logic.

The write process contains two parts: first is preparation of the match rules, then storing the rules into the quad of 32-deep LUT RAMs. More specifically, its logic uses a counter, counting from 0 to 31 to calculate the 32-bit match rules from given data and care bits. The calculated match rules are buffered in a 32-bit shift right register, then written into the LUT RAM with its counter counting one more round. The counting cycle is limited to the address space of a single LUT RAM unit, which is 32 in our case. Read logic needs no preparation, so the response latency is the time it takes to traverse the LUT RAM, which is 32 cycles, plus one more cycle to register the output. Logic in the first write counting cycles is isolated from search or read logic, which means that either of them can be processed at this temporal interval.

The match process is straightforward. Take a 10-bit wide TCAM with a depth of 32 as an example. It is composed of eight units shown in Figure 2, and it is arranged in two columns and four rows. It is able to store $4 \times 8 = 32$ match rules and match for keys in $2 \times 5 = 10$ bits width. Every time the match process begins, the search key is segmented into two 5-bit wide pieces, and each is sent to the whole column of units. Then the column takes the input 5-bit piece as the address of its RAM32Ms and output is a 32-bit deep indicator. The indicators from each column are sent to an AND gate, whose outputs report whether all the segmented pieces are matched. In this example, the input of AND gate is 32×2 matrix, and the output is a 32-bit deep vector. The final output is called the match line, which is an array indicating valid entries in the instruction table. One indicator is selected as priority and used to fetch an entry from the instruction table, then the fetched instruction is sent to the action stage in our scheme.

3.4. Action Stage

There are all kinds of functions that can be offloaded onto the NIC, e.g., checksum, PTP, and various protocol offloading; even data moving engines such as DMA and RDMA could be counted as offloads. Unlike FlowBlaze [15] and the other MAT solutions [11,12,14] whose execution units are ALU based implementations, here we use straightforward processing of packet headers to efficiently exploit the programmability of FPGA, and achieve a lower latency, as ALU based implementations require more cycles. Cerovi et al. in [10] conclude the essential steps for the fast path in a router, which we adapt in our action stage, including decrement of time to live (TTL), rewriting of the destination and source

mac address, and changing the output port according to a route table. The three steps are included in the proposed pipeline as the first, second and fourth actions. Furthermore, MAToC also offloads the virtual local area network (VLAN) extension of IEEE 802.1Q, operations including VLAN tag insertion, modification and removal. This is enabled by the capability of the variable length header merging of the deparser.

As shown in Figure 3, the first action of the pipeline is TTL decrement. In the cases of IPv4 and IPv6, with and without VLAN, the position of TTL is separately handled according to the packet type information resolved from the parsers. The second action is the replacement of the destination and source MAC address, which is controlled by the operation code from the match stage. The third action is the operation of the VLAN tag. The fourth action is forwarding the packet to a corresponding channel, which might be one of the eight MAC ports or that to the slow path, i.e., the applications running on the processing system of SoC.

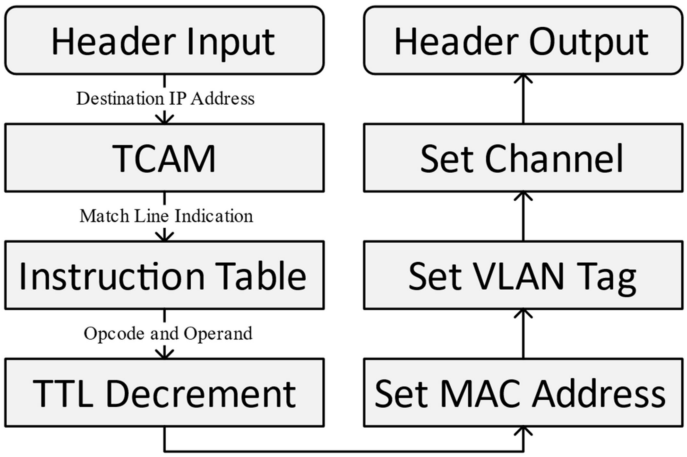

Figure 3. Match-Action pipeline.

Technically, the exact actions could be any packet processing offload engine in any order dedicated by users. As long as it does not block any incoming header frame while there is no backpressure signal asserting in the output, the throughput will not be affected.

3.5. Deparser

A traditional deparser is responsible for reassembling the processed packet headers from protocol fields, then reconstructing the packet frames and sending them to a buffer [13], with additional work such as replication for multicast [10]. The main job of the deparser in the proposed scheme is simplified to concatenate the processed header and payloads for each packet. In that way, it can be described as a byte-based barrel shifter, which is usually implemented using multiplexers. This is a simple job, but due to its own complexity and its position as the last stage of the fast path, its timing performance is highly critical.

In MAToC, the main challenge of deparser is in reducing the delay in concatenate logic. As MATs demand variation in header length, a deparser needs to merge a longer or shorter header frame with other payload frames. With a longer header as an example, the deparser ought to store the excess part into a temporary buffer, and then concatenate it with the lower part of the payload frame in the next transfer cycle, while the remainder of the payload frame is stored in the same temporary buffer. The concatenating is carried out over and over again until the packet transfer comes to an end. The variable length concatenating could be straightforwardly implemented using a logic shifter and bitwise OR gate, but that would result in an unacceptable timing performance. This is because the synthesizer always maps these shifting as multiplexers with input of all the bits along the header frame, the current payload frame, and the temporary buffer. The number of inputs as well as routing complexity increase with the variation range in header length, which could be too complex to achieve timing closure.

In our implemented deparser, we reduce the candidate bits of the multiplexers from the whole frame to a limited number according to a given variation range. Based on the fact that the shifting displacement is a multiple of byte width, its candidate bits should only contain the corresponding position in every eight bits. As an example, when setting the frame size to 16 bytes and limiting the header's variation range to ±4 bytes, the accepted header's size should range from 12 to 20 bytes. The example concatenation design is illustrated in Figure 4. The whole process begins with storing the shifted header frame into a temporary buffer, while the lower 16 bytes would be transferred out if the header length is increased or unchanged. After the first transfer cycle, the buffer would continue to store the payload frames, and its output would concatenate with the incoming payloads. There are 16 multiplexers in the last shaded block, generating all possible concatenations and being controlled by a select signal. The select signal is calculated according to the variation of header length (var) and the state of the finite state machine (FSM) defined in the Concat Logic block.

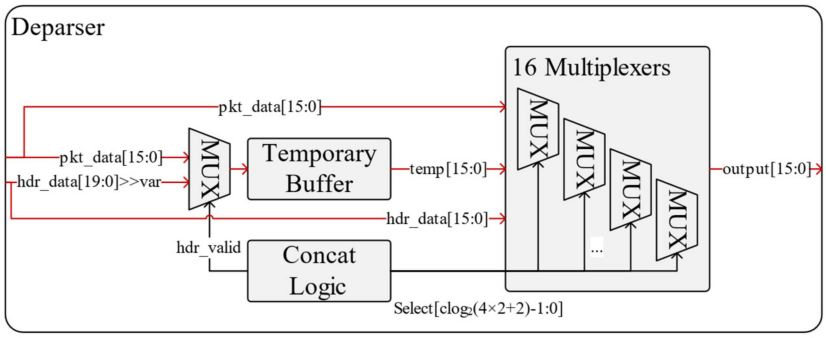

Figure 4. Concatenation design in deparser. For brevity, indexes here refer to bytes instead of bit, except for the Select signal.

All the possible concatenated outputs are listed in the select table in Figure 5, which explains the inner logic of the 16 multiplexers in Figure 4. The bottom row is the lower part of the header frames, which is selected as the first output frame when header length is increased or unchanged (select signal is 0, i.e., sel = 0), and it can be selected once at most. The top row is the initial payload data and would be selected if the header length is unchanged (var = 0, sel = 9). The others are the concatenation of the buffered data and the incoming payload, in which circumstance the select single varies from 1 to 8 depending on the variation of packet length. By carefully designing the multiplexer for each output data byte, our deparser achieved better resource and timing performance than the straightforward implementation, because the selection candidates are reduced.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	var	sel
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	9
14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	1	8
13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	2	7
12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	3	6
11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	4	5
3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	-4	4
2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	-3	3
1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	-2	2
0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	-1	1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		0

: Header Frame : Temporary Buffer : Payload Frame : Output Frame

Figure 5. Select table for concatenated output.

4. Implementation

The example implementation of the proposed scheme is tested on a commercial off-the-shelf SmartNIC Stargate F1000 with a Zynq MPSoC chip produced by ResNIC Co., Shanghai, China. We have ported Corundum of 10G and 25G link mode onto the board with most of the management features disabled for convenience. The 100G mode is provided by the vendor. The board can operate as a normal NIC without any user application.

4.1. Simulation

For verification purposes, the Cocotb [28] is used. This is a python verification framework with adequate support on bus interfaces and a capacity for wide test coverage. The functional verification and stress tests are performed separately for match stage, action stage, MAToC top module and at PCIe level. The simulation at PCIe level is done using a simulating driver in python provided by Corundum.

The implementation is coded in Verilog. Many of the components come from the Corundum library, including the multiplexer that constructs the packet processing data path. A few improvements have been made. For example, the original multiplexer introduces a bubble between packets when there is only one single channel that keeps entering. This is because the arbiter excludes the channel granted for its decision, so the channel would not be granted again until the transfer of the current packet was completed. In order to eliminate bubbles without adding extra delays, a prediction grant result is assigned to the currently granted channel by the arbiter.

4.2. Resource Utilization Analysis

As mentioned earlier, the only resource that matters in MAToC is BRAM, as the primitive Corundum allocates several buffers in the data path, together with consumption of DMA RAM for packet data, descriptor and completion in both receive and transmit direction of each independent channel. This has already slightly exceeded the BRAM capacity of this part (984 for zu19eg-ffvc1760-2e) in the 8×25G mode, leading to failure in placement. Even if resources are reduced below capacity, the heavy use of distributed resources in a centralized design will lead to the increase of propagation time in all paths, resulting in an unacceptable drop of maximal clock frequency. In this section, we collect some architecture parameters that should be carefully analyzed and set in order to achieve optimal resource utilization and meet the timing constraints.

In order to reduce the use of BRAM, we disabled functions such as PTP timestamp and statistics gathering. We also managed to set parameters for buffers and DMA RAM. The instantiations of block RAM can be configured as 36×1024 or 64×512 (Width $\times$ Depth). In some cases, improper parameters can lead to waste in space utilization. For example, let the size parameter of a buffer with data width of 16 bytes equal 8192 bytes; its depth is resolved as $8192 \div 16 = 512$ and it would be synthesized as two BRAM configured as 64×512. If the SIZE is larger than 8192, then the synthesis could result in four BRAM as 32×1024, which is double that of the former one. If it is 4096 bytes, the synthesizer would result in the same numbers of instantiated BRAM as the original setting, wasting half of its address space. By carefully setting the buffer size, BRAMs would be configured to the width of 64 as much as possible, and making full use of the memory space, i.e., letting the buffer depth equal 512 can benefit to the utilization result. As for the packet buffer for each channel, their depths are preset according to the delay cycles in the MAT pipeline and the maximum frame length. Specifically, it should be the larger one of the maximum frame length and the internal data size multiplies the delay cycles. Based on this result, trade-off between resources and performance is then to be considered by the designer.

4.3. Thoughput Analysis

Throughput is the main metric in our experiments. The physical limitation in optical modules is called the line rate, which is around $8 \times 25 = 200$ Gbps in our case and it is also the target throughput. The throughput of the internal data path depends on data

width and clock frequency. Parameters such as data width are automatically calculated to ensure sufficient bandwidth of the on-board data path. For example, the data width is bound to 512 bits for 100G link mode and 64 bits for 10G, 128 for 25G with the same clock frequency of 250 MHz. Therefore, the original infrastructure modules would not be a restriction. As we mentioned earlier, when traffic from all eight channels is merged into a single pipeline for processing purposes, the pipeline becomes a bottleneck for the entire application. This is why we emphasize that modules in the fast path should not stall packets when the backpressure signal is not asserted. In addition, to maintain a throughput margin, data width adaptation is necessary.

In MAToC, data width adapting is performed for the MAT pipeline, from 128 bits in the ingress and egress, to 512 bits internally, in order to ensure the headers are transferred as one frame per packet. The working frequency is 250 MHz, which means the total throughput is limited to 250 Mpps. We will show that this choice is a trade-off between routing complexity and performance.

With eight channels combined, the total throughput seems to be 200 Gbps. However, the actual data rate varies with packet length (PL) due to the bandwidth consumption of interframe gap (IFG, normally 12 bytes), preamble (abbreviated as P, 7 bytes), and start frame delimiter (SFD, 1 byte). The maximum receive rate can be calculated by formula (1), which is coincident with the actual traffic rate from the IXIA generator, tagged as transmit rate upper bound (TX UB) in Figure 3.

$$\text{Rx Rate} = \frac{\text{PL}}{\text{PL} + \text{IFG} + \text{P} + \text{SFD}} \times \text{Line Rate}, \quad (1)$$

4.4. Timing Analysis

As mentioned before, the timing performance is critical for deparsers, so an analysis on the logic delay of deparsers is necessary. In the logic level, multiplexers (MUXs) are implemented using look up tables (LUTs) and dedicated multiplexers in CLB. Each Ultrascale architecture CLB has eight 6-input LUTs (LUT6), four F7MUX, and two F8MUX. One LUT6 can be used to implement a 4:1 MUX. Combining with the adjacent LUT6 and their F7MUX, an 8:1 MUX can be implemented. With all of the LUTs in one CLB, the widest multiplexer it can implement is a 32:1 MUX. A lower bound of logic delay can be calculated from this perspective.

For example, the proposed deparser is configured to fit the variation range of 15 bytes, which means 32 (=15 $\times$ 2 + 2) candidates are arranged for each output bit. In the case of the lowest logic delay, only one CLB is utilized for one-bit output, and its logic delay includes one LUT6, one F7MUX, one F8MUX and one F9MUX. However, for a general barrel shifter that takes the header, payload and temporary buffer as candidates, the number of inputs is 1656 (=632 + 512 + 512). Hence, at least 54 CLBs constructed into three layers of 32:1 MUXs are needed for one single output bit.

5. Result

The function verification and performance tests are performed in a DELL PowerEdge R740 server machine with a PCIe gen 3.0 x16 slot supported. On the client side, a traffic generator named IXIA Network is used, which can generate dedicated traffic to provide performance and stress tests.

Throughput is measured using uniform traffics with different search keys equally distributed to achieve peak performance. Tests are done in variable packet lengths to draw a curve of throughputs versus frame lengths. Those ingress packets from IXIA would go through the whole pipeline and all the actions are performed, among which VLAN encapsulation is specified. Results in the figure below show that the proposed scheme can process packets at line rate when packet lengths are larger than 76 bytes. It should be noted that curves are printed according to the ingress frame size, while the egress frame sizes are actually enlarged by the encapsulation of VLAN tags.

Figure 6 shows some performance decrease for small packet lengths compared with the egress throughput upper bound as a red line drawn from formula (1). This is because of the deficiency of single pipeline designs, whose total processing rate in packet per second (pps) is limited to their working frequency. In the case of small packets, the actual egress throughput can be estimated by formula (2), where PL stands for packet length from ingress and the extra four bytes come from encapsulated VLAN tags. The intersection of the two formulas (1) and (2) implies a minimal packet length that makes the total throughput reach its upper bound, which is 76 bytes in our proposed scheme. To wipe out the performance decline for small packets, solutions can be increasing the clock speed or instantiating another pipeline to discharge the heavy traffic, i.e., making two identical pipelines of 512 bits for every four 128 bits wide input channels, either increasing routing complexity or design complexity.

$$\text{Tx Rate} = \text{Freq} \times (4\text{B} + \text{PL}) = 250\ \text{MHz} \times (4 + \text{PL}) \times 8\ \text{bit} \quad (2)$$

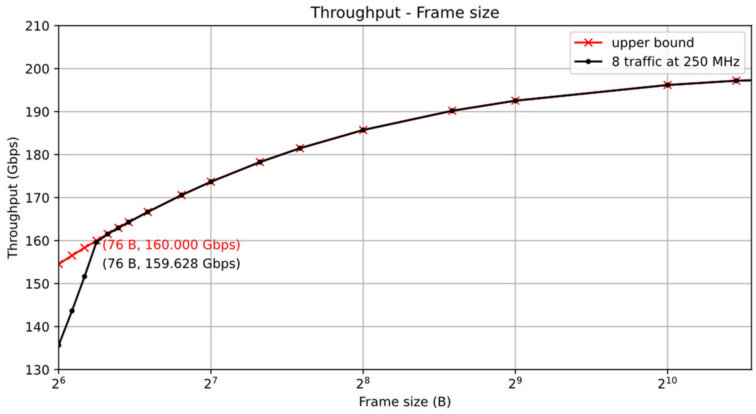

Figure 6. Throughput in reference to frame size.

The latency time is 19 clock cycles as shown in Table 1 where detailed timing for each stage is given. The overall latency is 76 nanoseconds in the 250 MHz clock domain and it shows that MAToC is capable of meeting the latency requirements of most applications. Most of the modules in MAToC only introduce one cycle delay as skid buffers are generally used. Adapters before parsers take four cycles to generate one 512 bits output transaction from four 128-bit input transactions. The multiplexer adopts a round-robin arbiter, whose latency depends on the number of inputs when they are all asserting for transmission, which is at most eight in MAToC. The four delay cycles of the match stage are separately from search key preparation, TCAM response, instruction data fetch, and at the last registered output. The instruction table is made of block RAM and its latency comes from the inner output registering in RTL's view. It is a fairly compact design, but it can still be improved with a more integrated design, which would be a future work. The latency of action stages depends on these internal offload engines. The time needed to pass through the switch is variable depending on the competition conditions. It would be only one cycle in a uniform test traffic with search keys equally distributed. In addition, traffic in the latter adapters could be buffered with a FIFO to avoid the throughput reduction caused by pipeline stalling in the switch.

The resource utilization of MAToC is listed in Table 2. There is still a large number of unused LUT resources to ensure the future development of more offload engines. The TCAM size is 35 $\times$ 1024 here, and it is the only module that consumes the LUT RAM (LUT as Memory). Block RAM is used for packet FIFO buffers (8.5 for each channel) and the instruction table (4 in this config).

A series of experiments are designed to evaluate the hardware cost of the deparser. Deparsers are implemented in different configs and resources utilization reports are generated. The Figure 7 below shows that the logic cost in terms of some common types of resources increase along with the variation range of header length.

Table 1. Latency.

Module	Latency (Cycle)
Adapter ahead	4
Parser	1
Multiplex	1
Match Stage	4
Action Stage	5
Demultiplexer	1
Deparser	1
Adapter after	1
Switch	1
Total	19

Table 2. Utilization.

Resources	Number	Percentage %
LUT as logic	76453	14.63
LUT as Memory	3640	1.64
Block RAM	72	7.33
Flip-Flops	73537	7.03
CARRY8	57	0.09
F7 Muxes	6595	2.52
F8 Muxes	2406	1.84

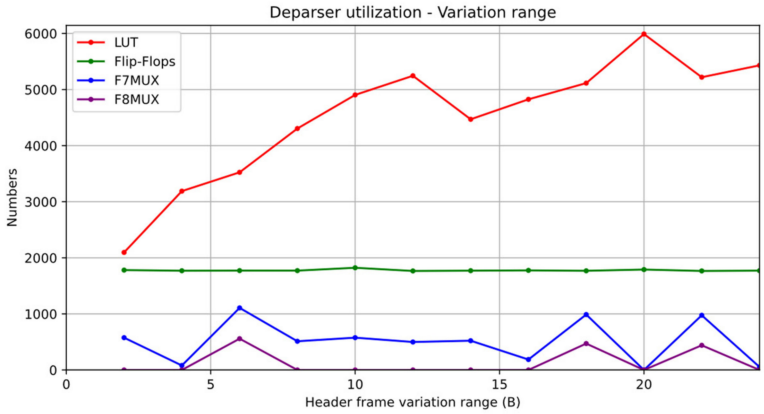

Figure 7. Utilization of deparsers in different variation range.

6. Conclusions

In this paper, we propose MAToC as a high throughput, low latency, MAT-based packet processing scheme for 200 Gbps network based on the open source NIC Corundum. Moreover, we also design a full-featured, low latency TCAM based on FracTCAM and a compact frame merging deparser, supporting the merger of two variable length frames. Performance and resource utilization are also analyzed in detail. The throughput test shows that MAToC has realized line rate processing and IP forwarding for packets larger than 76 bytes. The overall latency of the processing pipeline is 19 clock cycles that results in 76 ns using a 250 MHz clock signal.

There are two major limitations to this design. One is that the processing capability cannot cope with small packets. One possible solution is to increase the clock rate for the entire application block, which would also bring the challenge of timing closure. The asynchronous crossing between different clock domains is addressed by an asynchronous FIFO buffer in this project. It is built from the dedicated logic in block RAM, which can provide two accessing ports for asynchronous read and write. The speed should be high enough that the pipeline can handle the smallest packets of 64 bytes at line rate. For a board

with eight ports in 25GBASE-R, the total throughput is about 200 Gbps. The lower bound on the clock rate is the total throughput divided by the sum of IFG, preamble and packet length, which is approximately 300 MHz. The other limitation is that TCAM cannot be implemented in a larger size due to limited resources and timing issues. We plan to design a multi-layer match table that uses on-board DDRs, which have much more capacity and longer latency than the memory for FPGA. This will be a future work.

Author Contributions: Conceptualization, Z.G. and X.C.; methodology, implementation and validation, J.L; writing-original draft preparation: J.L.; writing-review and editing, Z.G., X.C. and J.L.; supervision, Z.G. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Strategic Leadership Project of Chinese Academy of Sciences: SEANET Technology Standardization Research and System Development (Project No. XDC02070100) and IACAS Frontier Exploration Project (Project No. QYTS202006).

Acknowledgments: The authors would like to thank the reviewers for their valuable feedback. The authors also would like to thank Xiaoying Huang for his helpful comments.

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

References

1. Sha, M.; Guo, Z.; Song, M.; Wang, K. A Review of FPGA's Application in High-speed Network Processing. *Netw. New Media Technol.* **2021**, *10*, 6.
2. Forencich, A.; Snoeren, A.C.; Porter, G.; Papen, G.; Soc, I.C. Corundum: An Open-Source 100-Gbps NIC. In Proceedings of the 28th IEEE International Symposium on Field-Programmable Custom Computing Machines, Fayetteville, AR, USA, 3–6 May 2020; pp. 38–46. [CrossRef]
3. Adaptable Accelerator Cards for Data Center Workloads. Available online: https://www.xilinx.com/products/boards-and-kits/alveo.html (accessed on 4 July 2022).
4. Intel FPGA Acceleration Card Solutions. Available online: https://www.intel.com/content/www/us/en/products/details/fpga/platforms/pac.html (accessed on 4 July 2022).
5. 8000 Series Ethernet Adapters—10/40GbE Network Adapters. Available online: https://www.xilinx.com/products/boards-and-kits/8000-series.html (accessed on 4 July 2022).
6. Xilinx Adaptive SoCs. Available online: https://www.xilinx.com/products/silicon-devices/soc.html (accessed on 4 July 2022).
7. Stephens, B.; Akella, A.; Swift, M.M.; Comp, M.A. Your Programmable NIC Should be a Programmable Switch. In *Hotnets-Xvii: Proceedings of the 2018 Acm Workshop on Hot Topics in Networks*; Association for Computing Machinery: New York, NY, USA, 2018; pp. 36–42. [CrossRef]
8. Lin, J.; Patel, K.; Stephens, B.E.; Sivaraman, A.; Akella, A.; Assoc, U. PANIC: A High-Performance Programmable NIC for Multi-tenant Networks. In Proceedings of the 14th Usenix Symposium on Operating Systems Design and Implementation (Osdi '20), Banff, Alberta, Canada, 4–6 November 2020; Usenix Association: Berkley, CA, USA, 2020; pp. 243–259.
9. Wang, T.; Yang, X.; Antichi, G.; Sivaraman, A.; Panda, A. Isolation mechanisms for high-speed packet-processing pipelines. In Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation, Renton, WA, USA, Online, 4–6 April 2022; Volume 22. [CrossRef]
10. Cerovi, D.; del Piccolo, V.; Amamou, A.; Haddadou, K.; Pujolle, G. Fast Packet Processing: A Survey. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 3645–3676. [CrossRef]
11. Bosshart, P.; Daly, D.; Gibb, G.; Izzard, M.; McKeown, N.; Rexford, J.; Schlesinger, C.; Talayco, D.; Vahdat, A.; Varghese, G.; et al. Programming Protocol-Independent Packet Processors. *ACM Sigcomm. Comput. Commun. Rev.* **2014**, *44*, 87–95. [CrossRef]
12. Kekely, M.; Korenek, J. Mapping of P4 Match Action Tables to FPGA. In Proceedings of the 27th International Conference on Field Programmable Logic and Applications (FPL), Ghent, Belgium, 4–8 September 2017.
13. Luinaud, T.; Stimpfling, T.; da Silva, J.S.; Savaria, Y.; Langlois, J.M.P. Bridging the Gap: FPGAs as Programmable Switches. In Proceedings of the 21st IEEE International Conference on High Performance Switching and Routing (IEEE HPSR), Newark, NJ, USA, 11–14 May 2020.
14. Bosshart, P.; Gibb, G.; Kim, H.-S.; Varghese, G.; McKeown, N.; Izzard, M.; Mujica, F.; Horowitz, M. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. *ACM Sigcomm Comput. Commun. Rev.* **2013**, *43*, 99–110. [CrossRef]
15. Pontarelli, S.; Bifulco, R.; Bonola, M.; Cascone, C.; Spaziani, M.; Bruschi, V.; Sanvito, D.; Siracusano, G.; Capone, A.; Honda, M.; et al. FlowBlaze: Stateful Packet Processing in Hardware. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*; USENIX Association: Boston, MA, USA, 2019; pp. 531–548. Available online: https://www.usenix.org/conference/nsdi19/presentation/pontarelli (accessed on 4 July 2022).
16. Content Addressable Memory (CAM). Available online: https://www.xilinx.com/products/intellectual-property/ef-di-cam.html (accessed on 4 July 2022).

17. Abdelhadi, A.M.S.; Lemieux, G.G.F.; Shannon, L. Modular Block-RAM-Based Longest-Prefix Match Ternary Content-Addressable Memories. In Proceedings of the 28th International Conference on Field Programmable Logic and Applications (FPL), Dublin, Ireland, 27–31 August 2018; pp. 243–250. [CrossRef]
18. 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. Syst.* **2020**, *28*, 1925–1929. [CrossRef]
19. 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]
20. Zahir, A.; Khattak, S.K.; Ullah, A.; Reviriego, P.; Muslim, F.B.; Ahmad, W. FracTCAM: Fracturable LUTRAM-Based TCAM Emulation on Xilinx FPGAs. *IEEE Trans. Very Large Scale Integr. Syst.* **2020**, *28*, 2726–2730. [CrossRef]
21. Reviriego, P.; Ullah, A.; Pontarelli, S. PR-TCAM: Efficient TCAM Emulation on Xilinx FPGAs Using Partial Reconfiguration. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2019**, *27*, 1952–1956. [CrossRef]
22. Advanced Systhesis Cookbook. Available online: http://www.altera.com/literature/manual/stx_cookbook.pdf (accessed on 4 July 2022).
23. Irfan, M.; Ullah, Z. G-AETCAM: Gate-Based Area-Efficient Ternary Content-Addressable Memory on FPGA. *IEEE Access* **2017**, *5*, 20785–20790. [CrossRef]
24. Irfan, M.; Ullah, Z.; Sanka, A.I.; Cheung, R.C.C. Accelerated Updating Mechanisms for FPGA-Based Ternary Content-Addressable Memory. *IEEE Embed. Syst. Lett.* **2021**, *13*, 37–40. [CrossRef]
25. Mahmood, H.; Ullah, Z.; Mujahid, O.; Ullah, I.; Hafeez, A. Beyond the Limits of Typical Strategies: Resources Efficient FPGA-Based TCAM. *IEEE Embed. Syst. Lett.* **2019**, *11*, 89–92. [CrossRef]
26. Kohler, T.; Duerr, F.; Rothermel, K. ZeroSDN: A Highly Flexible and Modular Architecture for Full-range Network Control Distribution. In Proceedings of the 2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems, Beijing, China, 18–19 May 2017; pp. 25–37. [CrossRef]
27. Gibb, G.; Varghese, G.; Horowitz, M.; McKeown, N. Design Principles for Packet Parsers. In Proceedings of the Symposium on Architectures for Networking and Communications Systems, San Jose, CA, USA, 21–22 October 2013; pp. 13–24.
28. Use Cocotb to Test and Verify Chip Designs in Python. Productive, and with a Smile. Available online: https://www.cocotb.org/ (accessed on 4 July 2022).