D-TCAM
A High-Performance Distributed RAM Based TCAM Architecture on FPGAs
Irfan, Muhammad; Ullah, Zahid; Cheung, Ray C. C.

Published in:
IEEE Access

Published: 01/01/2019

Document Version:
Final Published version, also known as Publisher's PDF, Publisher's Final version or Version of Record

License:
CC BY

Publication record in CityU Scholars:
Go to record

Published version (DOI):
10.1109/ACCESS.2019.2927108

Publication details:
Irfan, M., Ullah, Z., & Cheung, R. C. C. (2019). D-TCAM: A High-Performance Distributed RAM Based TCAM Architecture on FPGAs. IEEE Access, 7, 96060-96069. https://doi.org/10.1109/ACCESS.2019.2927108

Citing this paper
Please note that where the full-text provided on CityU Scholars is the Post-print version (also known as Accepted Author Manuscript, Peer-reviewed or Author Final version), it may differ from the Final Published version. When citing, ensure that you check and use the publisher's definitive version for pagination and other details.

General rights
Copyright for the publications made accessible via the CityU Scholars portal is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights. Users may not further distribute the material or use it for any profit-making activity or commercial gain.

Publisher permission
Permission for previously published items are in accordance with publisher's copyright policies sourced from the SHERPA RoMEO database. Links to full text versions (either Published or Post-print) are only available if corresponding publishers allow open access.

Take down policy
Contact lbscholars@cityu.edu.hk if you believe that this document breaches copyright and provide us with details. We will remove access to the work immediately and investigate your claim.

Download date: 28/05/2024
D-TCAM: A High-Performance Distributed RAM Based TCAM Architecture on FPGAs

MUHAMMAD IRFAN1, (Student Member, IEEE), ZAHID ULLAH2, (Member, IEEE), AND RAY C. C. CHEUNG1, (Member, IEEE)
1Department of Electrical Engineering, City University of Hong Kong, Hong Kong
2Department of Electrical Engineering, CECOS University of IT and Emerging Sciences, Pakistan

Corresponding author: Muhammad Irfan (m.irfan@my.cityu.edu.hk)

This work was supported by the City University of Hong Kong Collaborative Research Fund (CRF) under Grant 8730043.

ABSTRACT Ternary content-addressable memory (TCAM) is a high-speed searching device that searches the entire memory in parallel in deterministic time, unlike random-access memory (RAM), which searches sequentially. A network router classifies and forwards a data packet with the aid of a TCAM that stores the routing data in a table. Field-programmable gate arrays (FPGAs), due to its hardware-like performance and software-like reconfigurability, are widely used in networking systems where TCAM is an essential component. TCAM is not included in modern FPGAs, which leads to the emulation of TCAM using available resources on FPGA. Several emulated TCAM designs are presented but they lack the efficient utilization of FPGA’s hardware resources. In this paper, we present a novel TCAM architecture, the distributed RAM based TCAM (D-TCAM), using D-CAM as a building block. One D-CAM block implements a 48-bytes TCAM using 64 lookup tables (LUTs), that is cascaded horizontally and vertically to increase the width and depth of TCAM, respectively. A sample size of 512 × 144 is implemented on Xilinx Virtex-6 FPGA, which reduced the hardware utilization by 60% compared to the state-of-the-art FPGA-based TCAMs. Similarly, by exploiting the LUT-flip-flip (LUT-FF) pair nature of Xilinx FPGAs, the proposed TCAM architecture improves throughput by 58.8% without any additional hardware cost.

INDEX TERMS Content-addressable memory (CAM), field-programmable gate array (FPGA), lookup table (LUT), memory, random-access memory (RAM).

I. INTRODUCTION

Content-addressable memory (CAM) is a special type of associative array that performs parallel searching of the stored rules. It returns the address of the matched rule in one clock cycle [1]. CAMs are classified into two major types: binary CAM (BiCAM) supports only two memory states (‘0’ and ‘1’) and ternary CAM (TCAM) supports three memory states (‘0’, ‘1’, and ‘X’). ‘X’ state is known as don’t care bit or wildcard.

TCAM is widely used in many applications where the searching speed is imperative [2], [3], i.e., digital signal processing, artificial intelligence [4], translation lookaside buffers (TLBs) in soft processors [5], cache memory, and gene pattern recognition [6] in bioinformatics. Packet classification and packet forwarding maintain the high throughput of the system using TCAM in networking applications [7], e.g., software defined networks (SDNs) [8]. A routing table is maintained in the TCAM which stores the destination addresses and the ports to which the data packet needs to be forwarded. The router compares the destination address (“0 1 0 1”) of the incoming packet with the stored addresses and chooses the appropriate port (Port A), as shown in Fig. 1. The match line of TCAM acts as a pointer to the corresponding word (Port number) in static random-access memory (SRAM).

Field-programmable gate arrays (FPGAs) provide a large amount of logic and memory components which make them extremely favorable for the development of emerging systems [9]. Although TCAM is vital element in networking applications, modern FPGAs lack a hard core for TCAM because of its many other applications where TCAM would not be used. It was supported by some old FPGAs such as those from Altera (FLEX) [10] and Lattice in the form of partial memory blocks and multi-function blocks (MFBs), but TCAM is not included in the latest release of these.
devices. What if we need a TCAM for the implementation of networking system inside an FPGA?

Emulated TCAMs are the alternatives which are proposed by many researchers to utilize the hardware resources of FPGA to design a TCAM inside FPGAs [11], [12]. FPGA vendors including Altera [13] and Xilinx [11], [14] also provide soft IP (intellectual property) cores for TCAM to support its applications, but the memory utilization is inefficient in their current versions. Xilinx recently released an emulated TCAM to support the SDNs [11] which lacks efficient usage of memory resources on FPGA.

In this work, we propose a novel TCAM architecture using distributed-RAM (D-TCAM) which outperforms the state-of-the-art FPGA-based TCAMs by efficiently utilizing the hardware resources. We exploit the physical structure of FPGA slices to improve the throughput of TCAM by introducing pipeline registers to the design. The improvements are in the form of 60% better hardware utilization and 58.8% enhanced throughput compared to the state-of-the-art FPGA-based TCAMs [11], [15], [16].

The rest of the paper is arranged as follows: Section II provides the motivations and key contributions of the proposed design. Section III discusses the related work. Section IV describes the proposed TCAM architecture. Section V illustrates the technical feasibility of the proposed design and comparisons with state-of-the-art TCAMs. Section VI concludes this work and highlights future directions.

II. MOTIVATIONS AND CONTRIBUTIONS

A. MOTIVATIONS

TCAM is an essential element in modern networking applications where it provides the functionality of packet classification and packet forwarding in the form of implementing a routing table. FPGAs, having considerable amount of memory and logical resources, are extremely favorable for complex reconfigurable systems, e.g., software defined networks (SDNs). The presence of TCAM in FPGAs is crucial to implement such complex systems.

Some legacy FPGA devices, such as FLEX and APEX families, previously provided support for CAM and it was possible to implement a BCAM using an additional transistor. Lattice ispXPLD also supported TCAM up to size 128 × 48 using its multi-function blocks (MFBs) [17]. But all of these are discontinued. Modern FPGAs including those from Xilinx does not have a hard IP to support TCAM. Thus, several emulated TCAMs are proposed using the memory resources inside FPGAs, which are known as FPGA-based CAMs. In this paper, we propose a novel FPGA-based TCAM architecture which efficiently utilizes the available hardware resources on FPGA and improves throughput compared to the state-of-the-art FPGA-based TCAMs [11], [15].

B. KEY CONTRIBUTIONS

Key contributions of the proposed work are as follows:

- The proposed TCAM design, D-TCAM, outperforms the available FPGA-based TCAMs in terms of hardware utilization on FPGA and shows an improvement of 60% compared to the available state-of-the-art TCAMs [11].
- The available architectures use either lookup tables (LUT) [15] or flip-flops (FF) [18], [19] to implement TCAM and the other one is inferred by synthesizer but not part of the design; thus LUT-FF pairs are inefficiently utilized in such designs. D-TCAM efficiently exploits the paired nature of LUTs and FFs in modern FPGAs, and utilize it coherently, such that, none of it infers useless. Speed of the design is improved by using these redundant FFs.
- D-TCAM provides better performance as per throughput of the TCAM design and shows an improvement of 58.8% compared to the state-of-the-art FPGA-based TCAMs [11], [15].

III. RELATED WORK

Conventional TCAMs based on application-specific integrated circuits (ASICs) have low storage density, less reconfigurable, and challenging to integrate with FPGA-based systems [20], [21]. Emulation of TCAM using FPGA’s hardware resources is a better option to integrate with FPGA-based searching applications. The focus of this work is on emulated TCAMs which is a mature research area and consists of a large number of different architectures collectively known as FPGA-based TCAMs.

Based on the memory resources that are utilized to emulate TCAM functionality, FPGA-based TCAMs are classified into three categories.

1) The first is block RAM (BRAM)-based TCAMs [15], [16], [22]–[25] which use FPGA’s BRAMs to store the information of TCAM data. HP-TCAM [22] divided the TCAM table into horizontal and vertical blocks and stored it separately in BRAMs of FPGA. It was not possible without division to implement on FPGA resources because the BRAMs required for the architecture presented in patent [26] increases exponentially with increase in width of TCAM. The information about the presence of a rule and its address is stored in different BRAMs which inefficiently utilize BRAMs to emulate a small size (512 × 36) of TCAM. Z-TCAM [27] and E-TCAM [28] are slightly improved ver-
sions of HP-TCAM, but they also suffer from inefficient memory utilization. The major drawback of HP-TCAM, Z-TCAM, and E-TCAM is the requirement of TCAM rules to be in ascending or descending order; otherwise, it fails to provide the match address. UE-TCAM [23] reduced the hardware utilization by storing the information about the presence of TCAM rule and its address in the same BRAM. The TCAM rules are not needed to be arranged in ascending or descending order.

REST [24] proposed a hardware-efficient architecture by accessing the BRAM multiple times which affected the throughput of TCAM. Jiang [15] presented a scalable TCAM architecture in a sophisticated manner and utilized a large number of BRAMs to implement \(1024 \times 150\) TCAM. However, the design is entirely not clear from the description of the paper that how such a large size is achieved. Our proposed architecture utilizes the LUTRAM resources on FPGA according to its structure (LUT-FF pairs) which improves the throughput as well as hardware utilization. Multi-pumping [29] approach is used to access the BRAM memory multiple times in one system clock cycle to achieve a reduction in hardware utilization, but the speed is reduced with multiple accesses. Our proposed design is not compromising on the speed of the TCAM architecture and improves the hardware utilization as well as the throughput of the emulated TCAM design.

2) The second type of FPGA-based TCAM is the emulation using distributed-RAM resources on FPGAs [11], [14]–[16]. Most of the previous works that are designed initially using BRAMs are implemented using distributed-RAMs. Xilinx in 2012 presented a TCAM architecture in its application note using SRL16E which uses 4-input LUTRAM as a shift register [14]. One SRL16E implements 2-bits of TCAM. If the same approach is applied to 6-input LUTRAM, only 4-bit TCAM can be emulated. Our proposed TCAM architecture utilizes 6-input LUTRAM to emulate 6-bit TCAM and combines 64 LUTRAMs to emulate 46 bytes of TCAM. Jiang [15] also proposed a distributed RAM-based TCAM architecture which uses a huge number of slices to implement a TCAM of size \(1024 \times 150\). It uses 20526 slices to implement \(1024 \times 150\) TCAM on Xilinx Virtex-7 FPGA while our proposed TCAM design uses 4835 slices to implement \(512 \times 144\) TCAM on Xilinx Virtex-6 FPGA. The throughput of Jiang TCAM design and our proposed D-TCAM are 20.9 and 37.3 Gb/s, respectively.

Xilinx in 2017 released a soft IP core for SDNet which uses 12011 slices and 3 BRAMs (36Kb) to implement \(512 \times 128\) TCAM [11]. The architecture’s internal design is not disclosed in the IP data sheet, and only implementation results are mentioned. Our proposed TCAM architecture shows improvement than Xilinx design in slice utilization as well as throughput by 60% and 58.8%, respectively. In DURE [30] one slice is utilized to implement \(1 \times 18\) block of TCAM and carry chain logic is used to roll the AND logic of the match lines for the subsequent slices. It used 4 LUTRAMs to emulate 18 bits of TCAM while our proposed TCAM architecture uses 4 LUTRAMs to emulate 24 bits TCAM. Our proposed design provides a building block of \(64 \times 6\) TCAM which can be expanded horizontally and vertically to implement large sizes of TCAM.

Distributed RAM-based TCAM architectures waste a large number of flip-flops in FPGAs due to the LUT-FF pairs as it only needs LUTs to implement the design which we call as inefficient utilization of memory resources. We have designed the proposed architecture by keeping in mind the hardware structure of FPGA slices and its other components which is explained in the upcoming sections.

3) The third type of FPGA-based TCAMs in the literature are register-based TCAMs [18], [19], [31], [32] which use FFs to store the TCAM rules and perform parallel comparisons of each flip-flop with the input search key. They are also known as gate-based TCAMs. FFs in FPGA terminology are known as slice registers (SRs). Gate-based TCAMs reduces the hardware resources in terms of slices per TCAM bit but lacks scalability because of its high routing complexity. LH-CAM [18] and G-AETCAM [19] are implemented as \(64 \times 36\) BiCAM and TCAM, respectively, on Xilinx FPGA. It showed improvement in hardware resources and speed for small size TCAM, but for larger sizes, it is not a good option to emulate TCAM on FPGA.

Some other sub-types include hash-based TCAMs [7], algorithmic TCAMs, pseudo-TCAM [2], and cuckoo hashing [33] which have non-deterministic search latency and have the drawback of collision as well as bucket overflow. Algorithmic TCAMs perform best for a specific set of rules as they are based on a specific pattern in the input search key [34]. Updating procedures [35], [36] and soft errors detection [9] is another research direction in FPGA-based TCAMs. The scope of this paper is to propose a novel TCAM design which outperforms in hardware utilization and throughput compared to the state-of-the-art TCAM architectures.

Our proposed architecture is based on RAM of FPGA, so only type 1 and type 2 FPGA-based TCAMs are part of our comparisons as both are using the RAM available in FPGAs. To make a fair comparison between the BRAMs and distributed RAM, we computed the number of normalized slices for all of the TCAM architectures which are discussed in Section V.

**IV. PROPOSED ARCHITECTURE**

**A. TERMINOLOGY**

Table 1 lists the important terminologies used in the proposed TCAM design.

**B. FPGA STRUCTURE**

Modern FPGAs consist of reconfigurable hardware components, to implement any digital system, supported by the interconnection matrix. The main parts of Xilinx FPGAs are block RAMs (BRAMs), digital signal processing (DSP) blocks, configurable logic blocks (CLBs), and input/output
TABLE 1. Terminologies (notations for D-TCAM).

| Notation | Definition                        |
|----------|-----------------------------------|
| Sk       | Search key (W bits)              |
| W        | Width of the D-TCAM              |
| D        | Depth of the D-TCAM              |
| m        | Number of D-CAM blocks in a column |
| n        | Number of D-CAM blocks in a row  |
| MLs      | Match lines. Output of D-TCAM going into the priority encoder |
| MLs_Vs   | Match line vectors of 64 bits. Output of each D-CAM block |
| Sw       | Sub-word formed by dividing Sk into a 6-bit words |

(I/O) blocks (IOBs) as shown in Fig. 2. CLB consists of two types of slices; SLICEM and SLICEL. Each slice has four lookup tables (LUTs) and eight flip-flops (also known as SRs). LUTs inside a SLICEL can be configured as a logic element which implements any function up to 6-inputs (using one LUT), and are known as function generators. LUTs inside a SLICEM can be configured as a logic element as well as a memory element, and are known as LUTRAMs. Each LUTRAM is a column memory of 64 cells which can be used as 64-bit RAM. Multiple LUTRAMs combine to form a distributed RAM.

C. LUTRAM AS TCAM

A 3-input LUTRAM saves the information of $1 \times 3$ TCAM. The address input of the LUTRAM is treated as search key (Sk) of $1 \times 3$ TCAM. Only those location(s) stores ‘logic-1’ whose address corresponds to the rule present in TCAM, otherwise ‘logic-0’ is stored. For instance, 3-input LUTRAM saves the TCAM rule “0 1 1” which made the 3rd location of LUTRAM to be ‘logic-1’, and all others ‘logic-0’ as shown in Fig. 3(a). If the rule in TCAM has ‘X-bit(s), multiple locations of LUTRAM are ‘logic-1’. For instance, 3-input LUTRAM saves the TCAM rule “0 X 1”, which make the 1st and 3rd location of LUTRAM ‘logic-1’, as shown in Fig. 3(b).

Two 3-input LUTRAMs are cascaded to form a $2 \times 3$ TCAM. LUTRAMs in Fig. 3(c) store “1 0 X” and “X 0 1” as two 3-bit TCAM rules. Similarly, two 3-input LUTRAMs are connected in parallel with output ANDed to form $1 \times 6$ TCAM. LUTRAMs in Fig. 3(d) stores “1 0 X 0 1 0” as one 6-input rule. The input search key (Sk) equal to “1 0 0 1 0” and “1 0 0 1 0” matches with the stored rule to provide match line (ML) equal to ‘logic-1’ as output.

Four 3-input LUTRAMs are connected such that two LUTRAMs in parallel and two in cascade form to emulate a $2 \times 6$ TCAM. In Fig. 3(e) two TCAM rules of 6 bits each (“1 0 0 X 1 0” and “1 X 0 0 1”) are stored.

To simplify the understanding, we explained the emulation of LUTRAMs into TCAM using 3-input LUTRAMs. Modern
FPGA have mostly 6-input LUTRAM as in Xilinx FPGA. In the proposed TCAM design we use 6-input LUTRAMs which save information of 1 × 6 TCAM in a similar way as described for 3-input LUTRAM in Fig. 3(a). A D-CAM block has 64 6-input LUTRAMs which implements a 64 × 6 TCAM. Multiple D-CAM blocks are connected to form larger TCAMs similarly as described for multiple 3-input LUTRAMs in cascade and parallel form as shown in Fig. 3(e).

**D. ONE D-CAM BLOCK**

D-CAM block is the basic building block of the proposed architecture, D-TCAM. One D-CAM block consists of 64 6-input LUTs. These LUTs are from SLICEM slices of Xilinx FPGA, which are also known as LUTRAMs. Fig. 4(a) shows a D-CAM block which is a distributed memory of 64 × 64 consisting of 64 LUTRAMs. All of the memory locations can be read and write using the Address and Data_in as inputs, and Data_out as the output of the whole memory block (D-CAM). The same memory of Fig. 4(a) is used in Fig. 4(b) with a priority encoder, and it is changed into a TCAM memory.

Address in Fig. 4(a) represents the search key (Sk) of TCAM memory in Fig. 4(b). Data_out in Fig. 4(a) represents the match lines (MLs) of TCAM memory in Fig. 4(b). The final output of a TCAM is always an address where the search key is found. A priority encoder (PE) is normally used to translate match lines into an address. PE in itself has different implementation designs and is not part of the proposed design. Fig. 4(b) shows PE to ease the understanding of a typical TCAM data flow.

**E. MODULAR STRUCTURE OF D-TCAM**

Each D-CAM block implements 48 bytes of TCAM with configuration as 64 × 6, where 64 is the depth (D) of TCAM and 6 is the width (W) of TCAM. Increasing the width of TCAM from 6 bits to 12 bits requires an additional D-CAM block by keeping the depth of TCAM constant, as shown in Fig. 5(a). Similarly, increasing the depth of TCAM from 64 to 128 requires an additional D-CAM block by keeping the width of TCAM constant, as shown in Fig. 5(b). Horizontal expansion of D-CAM blocks is done to increase the width while the vertical expansion of D-CAM blocks is done to increase the depth of emulated D-TCAM. Expansion in both directions is done to create the desired size configuration of TCAM on FPGA. Fig. 6 shows the formation of 128 × 12 TCAM using four D-CAM blocks connected in parallel and cascade form.

Scalability of the proposed design is shown by emulating a D-TCAM of size 512 × 144 using 192 D-CAM blocks connected in parallel and cascade form.

**F. LUT-FF PAIRS**

FPGA’s hardware components are utilized in the form of slices (CLBs), BRAMs, DSP blocks, and I/O blocks to synthesize any digital design. LUTs and FFs are arranged as LUT-FF pair inside the slices. They cannot be used independently. For instance, if a digital design requires 200 LUTs and no FF, the synthesizer will assign 50 slices to it, which contains FFs as well. The 200 FFs are utilized (synthesized/inferred) by the digital design but not used in real,
which is a wastage of area in FPGA. Can we use these flip-flops to improve the performance of the system? Yes, we can! Let’s use these flip-flops for pipelining the design. The motivation is to design the digital systems according to the available hardware to efficiently utilize the area and hardware resources. The idea is to use these redundantly inferred FFs of the LUT-FF pairs for pipelining.

In this work, we are exploiting the LUT-FF pair nature of the hardware resources inside FPGAs and propose a novel design to utilize the available hardware resources efficiently. We use the redundant flip-flops as pipeline registers which improved the throughput of the proposed TCAM design.

G. PIPELINING

The available designs [11], [15], [16] that are based on distributed RAM use LUTRAMs to implement TCAM and need only LUTs from the LUT-FF pairs inside a slice. A large number of LUT-FF pairs are inferred by the synthesizer in the form of slices to implement the distributed RAM-based TCAM designs. LUTs are needed by these design, but FFs are not. However, due to the LUT-FF pair nature of FPGA, FFs are also inferred by the synthesizer while they are not part of the design. It is a waste of area on the reconfigurable device, i.e., FPGA. We call those FFs as redundant flip-flops. Our proposed TCAM architecture uses redundant FFs as pipeline registers which improved the throughput of the proposed TCAM design.

Algorithm 1 Populating Algorithm (D × W TCAM)

Input: TCAM Table (D × W)
Output: D-TCAM Table (m*n sub-tables)

1: for $t \leftarrow 0$ to $t \leftarrow m*n$ do
2:   for $i \leftarrow 0$ to $i \leftarrow m$ do
3:     for $j \leftarrow 0$ to $j \leftarrow n$ do
4:       D-CAM[i][j] $<=$ TCAM sub-table [i]
5:     end for
6:   end for
7: end for

$m \times n$: Total number of sub-tables in TCAM which is equal to the number of D-CAM blocks in D-TCAM. $m$: Number of sub-tables in a column. $n$: Number of sub-tables in a row.

H. POPULATING OF D-TCAM

How is the data from a TCAM table mapped into D-TCAM which consists of D-CAM blocks? The mapping of TCAM data into the emulated TCAM is known as populating. TCAM table is partitioned into $m$ rows and $n$ columns of sub-tables according to the dimensions of D-CAM block, i.e., $64 \times 6$. Each row of the partitioned TCAM table consists of 64 rules while each column consists of 6 bits. The $m \times n$ sub-tables of TCAM are mapped into the corresponding D-CAM blocks to form a D-TCAM of size D × W, where D is the depth of TCAM and W is the width of TCAM, as shown in Algorithm 1. Variable ‘i’ in the algorithm represents the corresponding sub-table of TCAM mapping to D-CAM blocks as well as the address where it is found in LUTRAMs in an efficient way to reduce the number of SRAM cells per TCAM bit compared to other LUTRAM-based TCAMs. Similarly, the redundant FFs are used as pipeline registers to divide the TCAM design into two stages which improved the throughput compared to the state-of-the-art TCAM designs.

The number of slices utilized by the TCAM design without pipelining and with pipelining is same, but the performance is improved in the latter case, which is the beauty of this work and is shown with FPGA implementation in the next section.
of D-TCAM. If we apply algorithm 1 on Fig. 6, \( t \) is 4. Theoretically, a total of \( m \times n \times 64 \) LUTRAMs are required to implement \( D \times W \) D-TCAM because each D-CAM block constitutes of 64 LUTRAMs.

Update latency of RAM-based TCAMs is proportional to the depth of the utilized RAM blocks. BRAM-based TCAMs have a 513 clock cycle update latency while distributed-RAM based TCAMs have an update latency of 65 clock cycles. Our proposed design is based on LUTRAMs which can be correspondingly updated in 65 clock cycles as a typical distributed RAM-based TCAM.

Algorithm 2 Searching of D-TCAM for Potential Match

\[
\begin{align*}
&\text{Input: Search key (Sk)} \\
&\text{Output: } m \text{ Match line vectors (MLVs) of 64 bits each which is converted to potential match address by PE.} \\
&1: \text{for } i \leftarrow 0 \text{ to } i < n \text{ do} \\
&\quad \text{sub-word } [i] \leftarrow Sk [6 \times n : (6 \times n + 5)]; \\
&2: \text{end for} \\
&3: \text{for } j \leftarrow 0 \text{ to } j < m \text{ do} \\
&\quad \text{MLV } [j] \leftarrow D\text{-CAM }[j][\text{subword(0)}] \text{ AND } D\text{-CAM }[j][\text{subword(1)}] \text{ AND} \\
&\text{..} \\
&\text{..} \\
&\text{D-CAM }[[\text{subword(n)}]; \\
&4: \text{end for} \\
&5: \text{PE } \leftarrow \text{MLV } [m:0]; \\
&\text{m: Number of sub-tables in a column.} \\
&\text{n: Number of sub-tables in a row.} \\
&\text{PE: A priority encoder which converts the match line vectors into the potential match address.}
\end{align*}
\]

I. SEARCHING OPERATION

The input search key (Sk) is divided into \( n \) 6-bit sub-words (Sw) and are provided to its corresponding \( n \) D-CAM blocks. Sw is searched in \( m \) D-CAM blocks in parallel to generate 64-bit match line vectors (MLVs). The \( m \) MLVs from each of the \( n \) D-CAM blocks are ANDed to form \( m \times 64 \) match lines (MLs) which are converted to an address by the priority encoder (PE) as shown in Algorithm 2.

V. PERFORMANCE EVALUATION

A. FPGA IMPLEMENTATION

To evaluate the feasibility of the proposed design, D-TCAM is successfully implemented on Xilinx Virtex-6 FPGA for the size \( 512 \times 144 \). The FPGA device XC6VLX760 is used with −2 speed grade using Xilinx ISE Design Suite 14.5. A large number of TCAM dimensions are implemented by varying the depth (D) and width (W) of the emulated TCAM which proves the scalability and modular structure of the proposed TCAM architecture. The number of slices used by the emulated design on Xilinx Virtex-6 FPGA is given in Table 2, where the left and right values are for the design without pipeline registers and with pipeline registers, respectively. Similarly, the speed of the emulated design is given in Table 3, where the left and right values are for the design without pipeline registers and with pipeline registers, respectively.

B. COMPARISON WITH OTHER TCAM ARCHITECTURES

FPGA-based TCAMs discussed in Section 3 use different memory resources, such as distributed RAM, BRAM, and registers (gate-based TCAMs). Number of BRAMs cannot be directly compared with the number of distributed RAMs because of its different structure and memory capacity. To balance it, DURE [30] adapted an equation to calculate the normalized slices (Slices') which is given in (1).

\[
\text{Slices'} = \text{No. of slices} + \text{No. of BRAMs(36k)} \times 24
\]  

For instance, HP-TCAM is implemented using BRAM, and its utilization is published in the form of BRAMs. So we converted the BRAMs to normalized slices using (1) which is provided in the comparison table as the number of slices (Slices'). Most of the TCAM designs under comparison are implemented using Virtex-6. We also implemented D-TCAM on Virtex-6 to make the comparisons fair. Some of the architectures are implemented on different devices which use different technology, e.g., Virtex-7 is used by Jiang [15] and Xilinx [11] while Kintex-7 is used by REST [24]. Virtex-6 is a 40 nm device while Virtex-7 and Kintex-7 are 28 nm devices, which are also different in speed. For comparisons, the normalized speed is calculated using (2), where \( \text{Speed'} \) is the normalized speed on the target device and \( \text{Vdd} \) is always equal to 1.0 V.

\[
\text{Speed'} = \text{Speed} \times \left[ \frac{\text{Device (nm)}}{\text{Virtex 6 (nm)}} \right]
\]  

Throughput (in Gbits/s) of the proposed design is found using (3) where \( \text{Speed} \) is the maximum clock rate of the
emulated TCAM and \( W \) is the width of the emulated TCAMs.

\[
\text{Throughput} = \text{Speed} \times W \tag{3}
\]

### C. IMPACT OF PIPELINING

The TCAM architectures that used LUTRAMs for emulating TCAM are not using FFs from the LUT-FF pairs of FPGA’s slices. We have used those redundant FFs as pipeline registers to improve the speed of the proposed TCAM architecture. The pipeline registers are FFs which requires additional slices to implement in a design, but in our case, almost no additional slices are used, but the speed improvement is prominent. To visualize it more clearly we have plotted four graphs showing the change in slice usage and change in the speed of the proposed design by using and not using pipeline registers. The horizontal dotted lines in Fig. 8(a) and Fig. 8(b)
shows that the increase in slices is almost negligible, but the horizontal dotted lines in Fig. 8(c) and Fig. 8(d) shows that the increase in speed is prominent. Speed of all emulated designs of D-TCAM is greater than the speed of the design without pipeline registers. Our proposed architecture shows improvement in speed of TCAM without using additional hardware resources on FPGAs. The ideal case of this implementation can be seen from Table 2 and 3 in case of D-TCAM size 128 × 36. The slices used in both cases (with pipeline registers and without pipeline registers) are the same (254), but the speed is improved by 52% (from 459 to 701 MHz).

Table 4 summarizes the state-of-the-art FPGA-based TCAM architectures, which show that our proposed architecture improves the hardware resources by 60% in terms of slices on FPGA. The throughput is improved from 29.8 Gb/s to 37.3 Gb/s by using the redundant FFs of the LUT-FF pairs as pipeline registers. The state-of-the-art TCAM design [11] has a throughput of 15.36 Gb/s while our proposed architecture significantly improves it to 37.3 Gb/s, which is 58.8% improvement because of the novel TCAM design using LUT-FF pairs.

VI. CONCLUSIONS AND FUTURE WORK

The proposed TCAM design efficiently utilizes the hardware resources on FPGA and exploits the internal structure of slices inside FPGA. Pipelining improved the performance of the TCAM, in terms of throughput, without extra hardware cost compared to the state-of-the-art FPGA-based TCAMs.

Future work targets on further optimization of emulated TCAM for networking applications which can more efficiently utilize the hardware resources on FPGAs.

REFERENCES

[1] R. Karam, R. Puri, S. Ghosh, and S. Bhunia, “Emerging trends in design and applications of memory-based computing and content-addressable memories,” Proc. IEEE, vol. 103, no. 8, pp. 1311–1330, Aug. 2015.

[2] W. Yu, S. Sivakumar, and D. Pao, “Pseudo-TCAM: SRAM-based architecture for packet classification in one memory access,” IEEE Netw. Lett., vol. 1, no. 2, pp. 89–92, Jun. 2019.

[3] X. Chen, H. Hua, J. Huan, and J. S. Vitter, “Efficient graph similarity search in external memory,” IEEE Access, vol. 5, pp. 4551–4560, 2017.

[4] E. I. Guerra-Hernandez, A. Espinal, P. Batres-Mendoza, C. H. Garcia-Capulin, R. De J. Romero-Troncoso, and H. Rostro-Gonzalez, “A FPGA-based neuromorphic locomotion system for multi-legged robots,” IEEE Access, vol. 5, pp. 8301–8312, 2017.

[5] S. Mittal, “A survey of techniques for architecting TLBs,” Concurrency Comput. Pract. Exper., vol. 29, no. 10, p. e4901, 2017.

[6] O. Mujadid, Z. Ullah, H. Mahmood, and A. Hafeez, “Fast pattern recognition through an LBP driven CAM on FPGA,” IEEE Access, vol. 6, pp. 39525–39531, 2018.

[7] Y. Yu, D. Belazougui, C. Qian, and Q. Zhang, “Memory-efficient and ultra-fast network lookup and forwarding using othello hashing,” IEEE/ACM Trans. Netw., vol. 26, no. 3, pp. 1151–1164, Jun. 2018.

[8] T.-S. Chen, D.-Y. Lee, T.-T. Liu, and A.-Y. Wu, “Dynamic reconfigurable ternary content addressable memory for OpenFlow-compliant low-power packet processing,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 63, no. 10, pp. 1661–1672, Oct. 2016.

[9] A. Ullah, P. Reviriego, A. Sánchez-Macián, and J. A. Maestro, “Multiple cell upset injection in BRAMs for Xilinx FPGAs,” IEEE Trans. Device Mater. Rel., vol. 18, no. 4, pp. 636–638, Dec. 2018.

[10] F. Altera, “10k embedded programmable logic device family data sheet,” Tech. Rep., 2003.

[11] Ternary Content Addressable Memory (TCAM) Search IP for SDNet, Xilinx Product Guide PG190, Xilinx, San Jose, CA, USA, Nov. 2017.

[12] P. Alfke, “Creative uses of block RAM,” Virtex and Spartan FPGA Families, Xilinx, San Jose, CA, USA, White Paper, 2008.

[13] Altera, “Implementing high speed search applications with Altera CAM,” Appl. Note 119 V2.1, Jul. 2001.

[14] K. Locke, “Parameterizable content-addressable memory,” Xilinx, San Jose, CA, USA, Appl. Note XAPP1151, 2011.

[15] W. Jiang, “Scalable ternary content addressable memory implementation using FPGAs,” in Proc. 9th ACM/IEEE Symp. Archit. Netw. Commun. Syst., Oct. 2013, pp. 71–82.

[16] Z. Qian and M. Margala, “Low power RAM-based hierarchical CAM on FPGA,” in Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig), Dec. 2014, pp. 1–4.

[17] A. M. S. Abdelhadi, G. G. F. Lemieux, and L. Shannon, “Modular block-RAM-based longest-prefix match ternary content-addressable memories,” in Proc. 28th Int. Conf. Field Program. Logic Appl. (FPL), Aug. 2018, pp. 243–250.

[18] Z. Ullah, “LH-CAM: Logic-based higher performance binary CAM architecture on FPGA,” IEEE Embedded Syst. Lett., vol. 9, no. 2, pp. 29–32, Jun. 2017.

[19] M. Irfan and Z. Ullah, “G-AETCAM: Gate-based area-efficient ternary content-addressable memory on FPGA,” IEEE Access, vol. 5, pp. 20785–20790, 2017.

[20] V. M. Zackriya and H. M. Kittur, “Precharge-free, low-power content-addressable memory,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 24, no. 8, pp. 2614–2621, Aug. 2016.

[21] S. Mishra and A. Dandapat, “Energy-efficient adaptive match-line controller for large-scale associative storage,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 64, no. 6, pp. 710–714, Jun. 2017.

[22] Z. Ullah, K. Ilgon, and S. Baeg, “Hybrid partitioned SRAM-based ternary content addressable memory,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 59, no. 12, pp. 2969–2979, Dec. 2012.

[23] Z. Ullah, M. K. Jaiswal, R. C. C. Cheung, and H. K. H. So, “UE-TCAM: An ultra efficient SRAM-based TCAM,” in Proc. IEEE Region 10 Conf. TENCON, Nov. 2015, pp. 1–6.

[24] A. Ahmed, K. Park, and S. Baeg, “Resource-efficient SRAM-based ternary content addressable memory,” IEEE Trans. Very Large Scale Integr. Syst., vol. 25, no. 4, pp. 1583–1587, Apr. 2017.

[25] C. A. Zerbini and J. M. Finocchietto, “Performance evaluation of packet classification on FPGA-based TCAM emulation architectures,” in Proc. Global Commun. Conf. (GLOBECOM), Dec. 2012, pp. 2766–2771.

[26] A. Almasanudar, “Circuits to generate a sequential index for an input number in a pre-defined list of numbers,” U.S. Patent 7 155 563 B1, Dec. 26, 2006.

[27] Z. Ullah, M. K. Jaiswal, and R. C. C. Cheung, “Z-TCAM: An SRAM-based architecture for TCAM,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 23, no. 2, pp. 402–406, Feb. 2015.

[28] Z. Ullah, M. K. Jaiswal, and R. C. C. Cheung, “E-TCAM: An efficient SRAM-based architecture for TCAM,” Circuits, Syst. Signal Process., vol. 33, no. 10, pp. 3123–3144, Oct. 2014.

[29] I. Ullah, Z. Ullah, and J.-A. Lee, “Efficient TCAM design based on multipumping-enabled multiported SRAM on FPGA,” IEEE Access, vol. 6, pp. 19940–19947, 2018.

[30] I. Ullah, Z. Ullah, U. Afzaal, and J.-A. Lee, “DURE: An energy- and resource-efficient TCAM Architecture for FPGA with dynamic updates,” IEEE Trans. Very Large Scale Integ. (VLSI) Syst., vol. 27, no. 6, pp. 1298–1307, Jun. 2019.

[31] M. Irfan and A. Ahmad, “Impact of initialization on gate-based area efficient ternary content-addressable memory,” in Proc. Int. Conf. Comput. Electron. Commun. Eng. (ICCECE), Aug. 2018, pp. 328–332.

[32] H. Mahmood, Z. Ullah, O. Mujadid, I. Ullah, and A. Hafeez, “Beyond the limits of typical strategies: Resources efficient FPGA-based TCAM,” IEEE Embedded Syst. Lett., to be published.

[33] S. Pontarelli, P. Reviriego, and J. A. Maestro, “Parallel D-pipeline: A cuckoo hashing implementation for increased throughput,” IEEE Trans. Comput., vol. 65, no. 1, pp. 326–331, Jan. 2016.

[34] A. Bremler-Barr, Y. Har chol, D. Hay, and Y. Hel-Or, “Encoding short ranges in TCAM without expansion: Efficient algorithm and applications,” IEEE/ACM Trans. Netw., vol. 26, no. 2, pp. 835–850, Apr. 2018.
MUHAMMAD IRFAN received the B.Sc. degree in electrical engineering from the University of Engineering and Technology, Peshawar, Pakistan, in 2013, and the M.S. degree in electrical engineering from the Lahore University of Management Sciences, Lahore, Pakistan, in 2016. He is currently pursuing the Ph.D. degree in electrical engineering with the City University of Hong Kong, Hong Kong.

He was with the Department of Electrical Engineering, CECOS University of IT and Emerging Sciences, Peshawar, Pakistan, for two years, and is currently on leave from there. His research interests include FPGA-based digital systems designs, low-power computer architectures, and memory design.

ZAHID ULLAH received the B.Sc. degree (Hons.) in computer system engineering from the University of Engineering and Technology, Peshawar, Pakistan, in 2006, the M.S. degree in electronic, electrical, control, and instrumentation engineering from Hanyang University, South Korea, in 2010, and the Ph.D. degree in electronic engineering from the City University of Hong Kong, Hong Kong, in 2014.

He is currently serving as an Associate Professor with the Department of Electrical Engineering, CECOS University of IT and Emerging Sciences, Peshawar, Pakistan. He has authored prestigious journal and conference papers and holds patents in his name in the field of FPGA-based TCAM. His research interests include low-power/high-speed CAM design on FPGA, low-power/high-speed VLSI design, and embedded systems.

RAY C. C. CHEUNG received the B.Eng. and M.Phil. degrees in computer engineering and computer science and engineering from The Chinese University of Hong Kong (CUHK), Hong Kong, in 1999 and 2001, respectively, and the Ph.D. degree and DIC in computing from Imperial College London, London, U.K., in 2007.

In 2009, he was a Visiting Research Fellow with the Department of Electrical Engineering, Princeton University, Princeton, NJ. He is currently an Associate Professor with the Department of Electronic Engineering, City University of Hong Kong (CityU). He is the author of more than 150 journal and conference papers. His research team, CityU Architecture Lab for Arithmetic and Security (CALAS), focuses on the following research topics: reconfigurable trusted computing, applied cryptography, and high-performance biomedical VLSI designs. He received the Hong Kong Croucher Foundation Fellowship for the postdoctoral study with the Electrical Engineering Department, University of California, Los Angeles (UCLA).