# DURE: An Energy- and Resource-Efficient TCAM Architecture for FPGAs With Dynamic Updates

Inayat Ullah, *Student Member, IEEE*, Zahid Ullah, *Member, IEEE*, Umar Afzaal, and Jeong-A Lee, *Senior Member, IEEE*

***Abstract*— Ternary content-addressable memory (TCAM) designed using static random-access memory (SRAM)-based field-programmable gate arrays (FPGAs) offers a promising lookup performance. However, the update process in a TCAM table poses significant challenges for efficiently employing SRAM-based TCAM. SRAM-based TCAM for FPGAs is designed using block RAM or distributed RAM resources in FPGAs. Such designs suspend search operations during an already high-latency update operation, rendering them infeasible in applications that require high-frequency updates. This paper presents a dynamically updateable energy- and resource-efficient TCAM design (DURE) based on FPGAs. DURE exploits the distributed RAM resources in FPGAs. More specifically, the lookup table RAMs (LUTRAMs) available in SLICEM resources are configured as quad-port RAM, which constitutes the basic memory (BM) block in the implementation of DURE. The contents of the TCAM table are divided into chunks of equal size and mapped onto the LUTRAMs of the proposed BM blocks. DURE implements dynamic updates by reconfiguring the LUTRAMs of only those BM blocks that are associated with the word being updated, thereby allowing search and update operations to be performed simultaneously. This achieves a lookup rate of 335 million lookups per second, with an update rate of 5.15 million updates per second on a 512 × 36 size TCAM on a Virtex-6 FPGA. Compared with the existing SRAM-based TCAMs, DURE has a smaller single-cycle search latency and achieves at least 2.5 times more energy efficiency and a 67% higher performance per area.**

***Index Terms*— Dynamic update, field-programmable gate array (FPGA), memory architecture, static random-access memory (SRAM)-based ternary content-addressable memory (TCAM).**

## I. Introduction

CONTENT addressable memory (CAM) is extensively employed for implementing high-speed lookup tables. CAM retrieves information based on content rather than addresses as in random access memory (RAM). CAM searches all of its stored words in parallel in a single clock cycle. Binary CAM stores only two states: "0" and "1." On the other hand, ternary CAM (TCAM) can store and compare a third state, known as the "don't care state x." The wild-card state "x" enables the TCAM to perform partial matching, which may result in a match with multiple words. TCAM architectures then use a priority encoder that is fed with the match/mismatch results from all the words to obtain the address of the matching word with the highest priority. The priority can be determined based on the physical location of the words, i.e., either the word with a lower address or a higher address has a higher priority. CAM enables a large amount of content to be searched in parallel at a high throughput. Therefore, it plays a crucial role in a wide variety of applications, such as network devices [1], [2], big-data analytics [3], [4], pattern recognition [5], [6], and data compression [7], [8].

Conventional TCAMs based on application-specific integrated circuits (ASICs) use a native hardware parallel search that offers low-deterministic search times and high performance. Despite its high-speed search operations, TCAM suffers from a low memory density and high power consumption. The dedicated bit-compare circuitry in each TCAM cell degrades the TCAM memory density, and the inherent massive parallelism makes TCAM expensive and power hungry [9], [10].

Contemporary static RAM (SRAM)-based field-programmable gate arrays (FPGAs) have emerged as an attractive platform for implementing configurable CAM architectures. Most FPGA-based TCAM solutions employ the on-chip SRAM for implementing the TCAM contents. The address of an SRAM word represents a distinct TCAM pattern, whereas the match information of a TCAM pattern against all the words of the TCAM table is stored in the corresponding SRAM word. In this way, all the words of an SRAM store the match/mismatch information for each word of the TCAM table against all the possible TCAM patterns.

A single update in SRAM-based TCAM requires rewriting of all the SRAM blocks configured by the design in the worst case owing to the TCAM's support for wild-card entries. The update process for a TCAM word in an SRAM-based TCAM architecture includes writing the update word and erasing the old TCAM word from SRAM blocks. This process involves the indexing of all the words of an SRAM block for writing. Such TCAMs implement dedicated update engines for writing the SRAM blocks in parallel, which introduces an excessive

Manuscript received July 28, 2018; revised November 8, 2018 and January 23, 2019; accepted February 24, 2019. Date of publication March 26, 2019; date of current version May 22, 2019. This work was supported in part by the National Research Foundation of Korea through the Ministry of Science and ICT under Grant NRF-2016R1A2B4010382 and in part by the Korea Institute of Energy Technology Evaluation and Planning and Ministry of Trade, Industry and Energy of the Republic of Korea under Grant 20184010201650. *(Corresponding author: Jeong-A Lee.)*

I. Ullah, U. Afzaal, and J. Lee are with the Department of Computer Engineering, Chosun University, Gwangju 61452, South Korea (e-mail: inayatmz@gmail.com; jalee@chosun.ac.kr).

Z. Ullah is with the Department of Electrical Engineering, CECOS University of IT and Emerging Sciences, Peshawar 25100, Pakistan.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2019.2904105

additional hardware overhead. Hence, the minimum number of cycles required for an update process is proportionate to the depth of the utilized SRAM blocks [11], [12].

Moreover, in order to ensure a consistent update of the TCAM table in SRAM-based TCAMs, search operations remain suspended during the update process as the match information of the word being updated is stored in all of the words of the SRAM [11], [13], [14]. If, however, a search operation is carried out during the update process, it will lead to an erroneous matching. Therefore, such TCAMs dedicate a substantial amount of clock cycles to supporting high-latency blocking updates in frequent update environments, thereby substantially lowering the arrival rate of input words for lookup [15]. Because lookup operations are locked during high-latency updates, SRAM-based TCAMs require a large buffer space to store input words in order to avoid the loss of these words [16], [17]. As such, despite the search performance of one search per cycle, these TCAMs are infeasible for use in high-churn-rate environments, where frequent TCAM updates are required. This critical TCAM table update issue in SRAM-based TCAM architectures has not been addressed so far. Therefore, further research on implementing dynamic updates in the FPGA-based TCAM architectures is required.

A translation look aside buffer (TLB) cache is an integral part of any processor architecture and is implemented using CAM [9], [18], [19]. Soft-core processors implemented in reconfigurable hardware, such as FPGAs, are widely employed in embedded system applications [20]. A low response time (single cycle) dynamically updateable CAM architecture is required in the design of a nonblocking TLB cache for soft-core processors [21]–[23]. However, existing RAM-based TCAM architectures on FPGAs suffer from high search latencies (high response times) and update operations during which searches are suspended.

In this paper, we present an FPGA-based dynamically reconfigurable TCAM design. This configures the lookup table RAMs (LUTRAMs) available in SLICEM resources on the FPGA as quad-port RAM. The match bits from the LUTRAMs are efficiently AND-cascaded by exploiting the hardwired fast carry-chains present in the modern FPGA, which are, in fact, intended for accelerated arithmetic operations. The proposed TCAM design offers high-speed single-cycle searches, achieves higher energy and resource efficiency, and supports dynamic updates without suspending search operations.

A summary of the main contributions of this paper is presented in the following.

1) A dynamically updatable FPGA-based TCAM architecture using LUTRAM FPGA primitives is proposed. The proposed architecture targets TCAM applications, where high-frequency updates are required. The main challenge in the efficient use of FPGA-based TCAM is the high-latency blocking update operation, which is resolved by the proposed solution.
2) We show how dedicated carry chain FPGA primitives, which are actually intended for arithmetic operations, are exploited in our design for AND-cascading the match outputs from the LUTRAMs. Thus, the use of extra logic and routing resources for this operation is avoided, resulting in a reduced delay time, which enables a higher design clock rate.
3) The performance of the proposed TCAM design is scalable, which has been evaluated with respect to the lookup rate, power consumption, device utilization, and update rate. It achieves a lookup rate of 175 million searches per second and update rate of 2.7 million updates per second for a table with 1 k TCAM words of 144 bits.
4) The proposed design is resource efficient. It achieves a higher TCAM bit storage density and saves a considerable amount of LUT resources by implementing the TCAM ANDing logic in carry chains. Compared with the existing state-of-the-art TCAM solutions, it achieves at least 67% higher performance per area.
5) The proposed design is highly energy efficient. Compared with the existing state-of-the-art TCAM solutions, it attains at least a 2.5 times better energy efficiency.
6) The single-cycle search latency of the proposed design makes it an attractive choice for applications requiring low-response-time search operations, such as TLB caches in microprocessors.

The remainder of this paper is organized as follows. Section II discusses existing FPGA-based TCAM solutions. The architecture of the proposed TCAM design is then detailed in Section III. Section IV discusses the implementation of the proposed TCAM on FPGA, evaluates the performance of the proposed design, and presents comparative studies with other existing TCAM solutions. Finally, Section V concludes this paper.

## II. Related Work

There is a large body of literature dealing with fast incremental updates in conventional TCAMs. These studies focus on issues related to supporting the longest prefix matching in IP lookups, where the TCAM words are arranged in an ordered list such that those with higher priority are stored at lower TCAM addresses [16], [24]–[26]. However, this paper focuses on FPGA-based TCAM architectures and targets the critical issue of blocking updates in existing SRAM-based TCAM architectures. The proposed approach resolves this issue by presenting a dynamically reconfigurable TCAM architecture that implements nonblocking updates.

Existing FPGA-based TCAM architectures are mostly RAM-based solutions employing block-RAM (BRAM) or distributed RAM resources for implementing TCAM. The BRAM-based TCAM solutions presented in previous work [13], [14], [27] store the knowledge about the presence of TCAM words and their address information in separate sets of BRAM units and thus suffer from inefficient memory usage. The update process for such TCAMs includes updating a copy of the TCAM table, which is then remapped to the BRAM configured by the design on the FPGA. As such, all the BRAMs configured by the TCAM are rewritten, which is time-consuming and expensive in terms of the additional logic overhead.

The BRAM-based TCAM solutions presented in [11], [12], [15], and [28]–[31] store the knowledge about the presence of the TCAM word and their address information in the same BRAM block. These approaches suffer from reduced memory efficiency because of the limited storage capacity of BRAM for storing the TCAM bits owing to a higher SRAM/TCAM ratio of $2^9/9$ [31]. The update process in the SRAM-based TCAMs from [11], [13]–[15], and [27]–[31] requires dedicated update engines to write all of the SRAM blocks used by the design in parallel, which results in an additional logic overhead. In this manner, the write cycle in the update process becomes proportional to the depth of the design-configured BRAMs, which requires 513 clock cycles [11], [12].

The distributed RAM-based TCAM solutions presented in the previous work [11], [15], [29] achieve a higher memory efficiency because of increased storage capacity for TCAM bits in distributed RAM with an SRAM/TCAM ratio of $2^5/5$. However, when implementing a large TCAM table, the bitwise ANDing of a large number of wide bit vectors becomes time-critical, resulting in a reduced achievable throughput. Although these designs have a relatively small update latency of 33 cycles, they suffer from an expensive rewriting of the entire configured SRAM memory on account of supporting wild-card entries.

Xilinx has presented several implementation techniques for binary CAM using BRAM and for TCAM using the shift register (SRLE16) on FPGAs in its application note [30]. The BRAM-based binary CAM technique is storage-inefficient. The TCAM implementation technique, which employs LUT resources as shift registers, has reduced resource efficiency, as only two TCAM bits are encoded and mapped on a single 16-bit shift register LUT primitive SRLE16. For larger TCAM table implementations, this technique suffers from timing problems arising because of routing congestion.

The multiplexer-based TCAM design presented in the patent [32] suffers from a degraded system throughput owing to the sequential searching of the TCAM data for the input word, i.e., a single TCAM word is searched per cycle. This technique stores the contents of the TCAM table in the LUTRAMs from the SLICEM resources, while the comparison circuitry is implemented using programmable FPGA interconnects. The outputs of the comparison circuits are further AND-cascaded to implement multiple bits of the TCAM word. In contrast, the proposed TCAM architecture achieves a higher system throughput, as the contents of the TCAM table are searched in parallel for an input word.

A fast content updating mechanism for SRAM-based TCAM is presented in [33]. Its update latency depends on the number of "don't care" bits in the subwords of an update word. However, this TCAM approach, which is implemented using Xilinx BRAM resources, has the worst case update latency, being 513 cycles.

As mentioned earlier, the update process in existing SRAM-based TCAMs requires the rewriting of all the SRAM blocks, which is highly expensive in terms of both the logic overhead and the reduction of the search rate. The decrease in the search rate results from the TCAM being locked for search

TABLE I
3 × 4 TCAM TABLE

| Address | TCAM words |
|---|---|
| 0 | 1001 |
| 1 | x100 |
| 2 | 01xx |

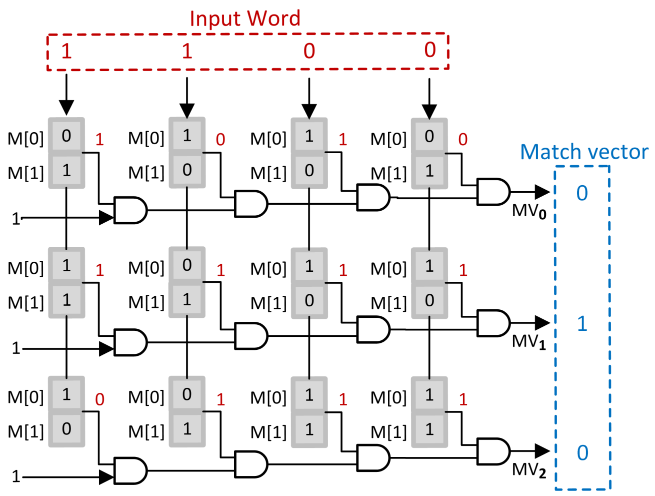


Fig. 1. Simplified implementation of DURE.

operations in already high-latency rewrite cycles. The time-consuming blocking update process comprises the main challenge in existing SRAM-based TCAMs on FPGAs. In this paper, a dynamically updateable energy- and resource-efficient TCAM design (DURE) is presented as a solution to the blocking update issue in SRAM-based TCAMs. DURE maps the contents of a TCAM table onto the LUTRAMs available in SLICEM resources, which together with the available fast carry chains implements the desired TCAM functionality, achieving a single-cycle search latency. DURE dynamically reconfigures only the LUTRAMs associated with the word being updated, without suspending search operations.

## III. HARDWARE ARCHITECTURE OF PROPOSED TCAM: DURE

### A. Basic Idea

The basic design of the DURE TCAM implementation technique is shown in Fig. 1, which shows a simplified implementation of the $3 \times 4$ TCAM table presented in Table I. A $1 \times 1$ TCAM can be implemented using a $2 \times 1$ RAM such that the match knowledge for the presence of a "0" value is represented by storing a "1" at RAM[0], a value of "1" by storing a "1" at RAM[1], and "x" state by storing a "1" at both the RAM[0] and RAM[1] locations. A $C$-bit word of a TCAM table can be implemented using a 1-bit RAM of $2^C$ positions. In general, the words of the TCAM table are divided into smaller chunks of $C$ bits, and each TCAM word is then implemented using a row of AND-cascaded RAMs, as shown in Fig. 1. When an input word is applied to the rows of RAM blocks, the partial match/mismatch information read is AND-cascaded to obtain the final match result.

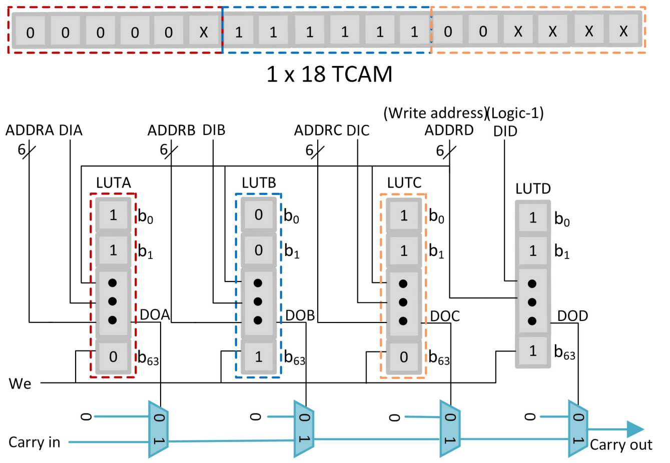


Fig. 2. Architecture of the BM block of DURE for implementing a $1 \times 18$ TCAM on the FPGA.

### B. Building Blocks of DURE on FPGA

DURE utilizes the on-chip-distributed RAM available in the state-of-the-art Xilinx FPGAs. The SLICEM resources of a Xilinx FPGA comprise four LUTRAM cells, which are configured as quad-port RAM (three read/one write) in DURE. The four LUTRAM cells are called LUTA, LUTB, LUTC, and LUTD in Xilinx terminology, as shown in Fig. 2. The LUTRAM cells from the same SLICEM share a common write address port. Four distinct bits are written to these LUTRAM cells using data from the ports DIA, DIB, DIC, and DID, while the write address is applied to LUTD. At the same time, the address ports of the three LUTRAM cells LUTA, LUTB, and LUTC are available for reading. In this manner, the structure creates quad-port RAM (three read/one write) with three read ports (read addresses are applied in parallel to LUTA, LUTB, and LUTC) and one write port (the write address is applied to LUTD).

The quad-port RAM structure shown in Fig. 2 implements a $1 \times 18$ TCAM using the LUTA, LUTB, and LUTC LUTRAM cells. Each of these 64-bit LUTs implements six bits of the TCAM. The input words are applied to the address ports of the LUTRAMs' LUTA, LUTB, and LUTC in parallel, and match information bits are read at their outputs. The hardwired fast carry chain structure present in the slice is then used for ANDing match bits to yield the final match result. This SLICEM configuration constitutes a basic memory (BM) block in DURE. The architecture of the BM block dictates that LUTD cannot be used for storing TCAM bits, as the address port of LUTD acts as the common write address port. To implement wide TCAM words, this implementation scheme must transfer the partial match results from the carry chain in the current slice to the carry chain in the next one. To forward the match result to the carry chain structure of the next slice, the output of the LUTD DOD must be permanently set to logic-1. For this purpose, all the bits of the LUTD are initialized to "1," and a logic-high is connected to its DID input.

### C. Architecture of the Proposed DURE

A TCAM table of word width $W$ and depth $D$ is shown in Fig. 3, where it is partitioned into chunks of equal size.

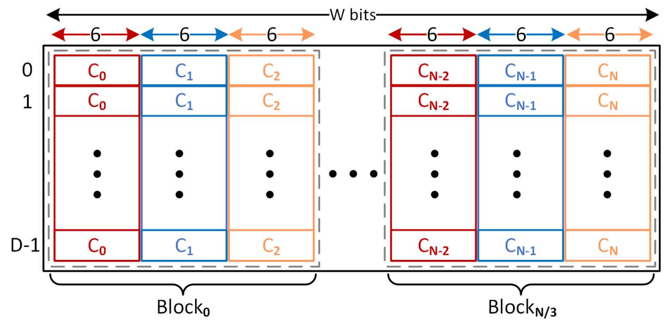


Fig. 3. Partitioning TCAM table contents into six-bit chunks.

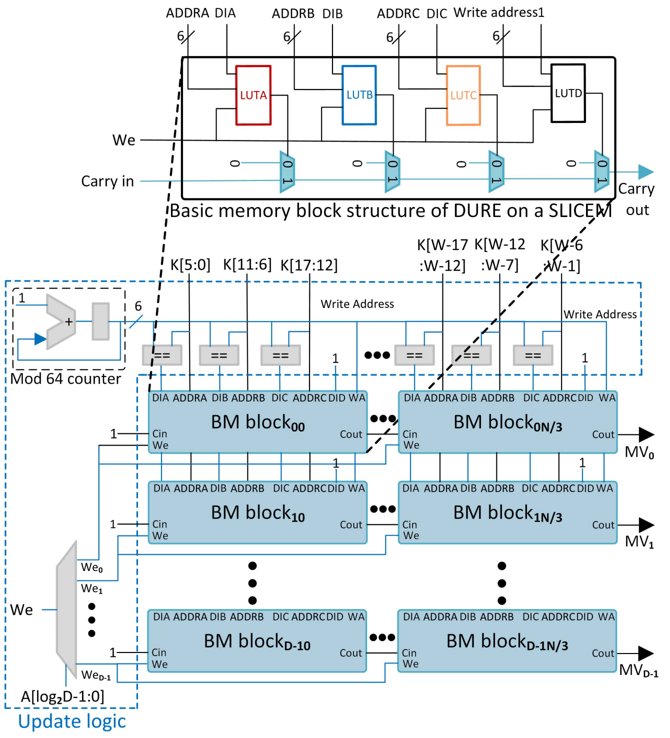


Fig. 4. Architecture of the proposed DURE.

The $D$ words of this TCAM table are divided into chunks of six bits, denoted by $C_0$, $C_1$,..., $C_{N-1}$, where $N = W/6$. Each three successive chunks of the TCAM table are annotated as basic blocks. Therefore, each basic block consists of $D$ subwords with a word width of 18 bits.

The architecture of the proposed DURE is shown in Fig. 4. This primarily includes a cascaded array of the proposed BM blocks implementing the TCAM and a separate update logic. The entire contents of the $D \times W$ TCAM table are mapped onto the array of BM blocks on the FPGA. The subwords of the basic blocks are mapped onto the LUTRAMs of the BM blocks with each six-bit chunk of the TCAM mapped onto a 64-bit LUTRAM cell. For example, the three chunks of the first basic TCAM block, $C_0$, $C_1$, and $C_2$, are implemented using LUTA, LUTB, and LUTC from the BM block, as shown in Figs. 3 and 4.

The input word bits are applied in parallel to the $D$ rows of the $N/3$ BM blocks. The match bits from the LUTRAMs' LUTA, LUTB, and LUTC of each BM block are AND-cascaded through the dedicated hardwired carry chain

**Algorithm 1** Update Operation in DURE

```
INPUT: A $W$-bit update TCAM word: $U_0, U_1, \ldots, U_{W-1}$.
INPUT: A $log_2 D$-bit update address $A$.
INPUT: Update operation: Add or Delete.
OUTPUT: Updated $D \times W/k$, $2^k$ bit arrays, where $k$=6: $M_{i,j} = b_{i,j,0}, b_{i,j,1}, \ldots, b_{i,j,2^{k-1}}$, $i = 0, 1, \ldots, D-1$, $j = 0, 1, \ldots, W/k - 1$
1:  for $i = 0, 1, \ldots, D-1$ do
2:      for $j = 0, 1, \ldots, W/k - 1$ do
3:          for $l = 0, 1, \ldots, 2^k - 1$ do
4:              if (Operation==Add) then
5:                  if ($U[j * k : (j+1) * k]$ matches $l$) then
6:                      $M_{A,j,l} \leftarrow' 1'$
7:                  else
8:                      $M_{A,j,l} \leftarrow' 0'$
9:                  end if
10:             else if (Operation==Delete) then
11:                 $M_{A,j,l} \leftarrow' 0'$
12:             end if
13:         end for
14:     end for
15: end for
```

structures, which extend into multiple slices [34], [35]. This implementation technique cascades the BM blocks of each row, BM Block$_0$, BM Block$_1$,..., BM Block$_{N/3}$, through carry chains. Consequently, the match bits from the LUTRAMs, in which each implement a TCAM word, are AND-cascaded to obtain the final match result. The use of these extended fast computation structures allow the cascading operations to be performed without consuming additional logic, thus also saving any associated routing resources.

### *D. Dynamic Update*

The procedure for updating a TCAM word in DURE is presented in Algorithm 1. The update logic in DURE includes an MOD-64 counter, decoder logic, and $N = W/6$ number of six-bit comparison modules, as shown in Fig. 4. The update address $A$ of the $log_2$ $D$-bits is decoded to give the write enable signals, which select the LUTRAM cells of the rows of the intended BM blocks for writing. The MOD-64 counter generates a new sequence of six bits for every cycle, which is matched with the six-bit chunks of the update word. If the match result is "TRUE," then a bit value of "1" is stored in the corresponding LUTRAM on the index specified by the six-bit sequence from the MOD-64 counter. The write address is applied using the common write address port of the LUTRAMs (the address port of LUTD) in each BM block, as explained earlier. DURE modifies all the LUTRAMs from the BM blocks corresponding to the update word in parallel, thus requiring 65 cycles.

The DURE TCAM implementation technique is highly efficient because all the LUTs from an occupied SLICEM are utilized and connected to the available three read/one write ports. Thus, the existing interconnections are exploited for the search and update processes with very little overhead for update logic in the proposed DURE architecture.

DURE exclusively writes the LUTRAM from the BM blocks corresponding to the word being updated without affecting the LUTRAM from the BM blocks implementing other words of the TCAM table. During the update process, the BM blocks are available for search operations, thereby allowing dynamic updates in DURE. DURE is able to maintain a consistent TCAM table during the update process by adapting the approach from [16]. The match result of the LUTRAMs from the BM blocks corresponding to the word being updated is invalidated. At the same time, a redundant copy of the update word is implemented using the flip-flop resource of the FPGA, which takes only a single cycle [36]. The search operations are not allowed during this single-cycle update process. In this way, DURE is able to perform search operations on a consistent TCAM table maintained throughout the update process, which can be used in all those partial matching search engine applications where maintaining the order of the TCAM words is not required [7], [8]. However, existing SRAM-based TCAM approaches are not able to support search operations during the update process. Furthermore, the update process of a single word in related SRAM-based TCAM approaches involves writing all the SRAM words, as the matching information of a TCAM word is stored among all the words of the SRAM.

DURE does not propose any solution to maintain a consistent TCAM table during the update process for applications, where the TCAM table words are arranged in an ordered list known as ordered TCAM. However, DURE is able to support ordered TCAM through an update of the dynamic priority encoder when the priority of the update word is different from that of the old TCAM word. The update process of the dynamic priority encoder cannot overlap the search operations. In this way, DURE maintains a consistent TCAM table and ensures the correctness of the matching addresses by suspending search operations during the priority encoder update for applications which require arranging the TCAM table words in an order [16], [24]–[26].

## IV. FPGA Implementation and Performance Evaluation

### *A. FPGA Implementation and Results*

DURE was implemented in a Xilinx Virtex-6 FPGA device (XC6VLX760). This FPGA has 33 120 SLICEMs each of which contains four six-input LUTs, constituting a total of 132 480 six-input LUTRAM cells. The LUTRAM cells and the carry chain structure available in the SLICEM resources are configured using DURE's BM blocks, as described in Fig. 2. This BM block structure cannot be inferred using synthesis and implementation tools. We implemented the proposed BM blocks by instantiating the LUTRAM cells as RAM64M primitives and initialized them with the appropriate bits by defining their INIT attributes. The DOA, DOB, DOC, and DOD outputs of the LUTRAMs were connected as select inputs to the carry chains by defining the CARRY4 primitives. For example, the implementation

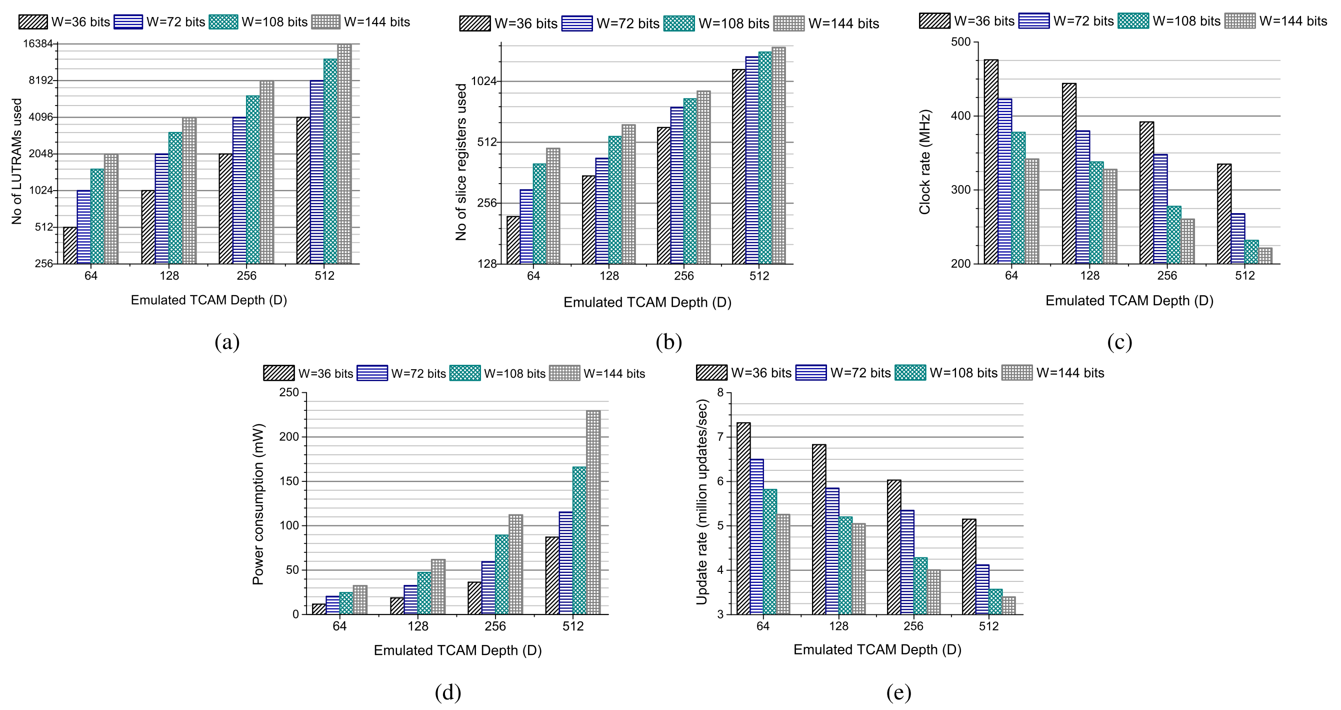


Fig. 5. Performance trend for increasing TCAM depth ($D$) and width ($W$). (a) LUTRAM utilization. (b) SR usage. (c) Clock rate. (d) Power consumption. (e) Update rate.

TABLE II
FPGA RESOURCE UTILIZATION FOR PROPOSED DURE

| TCAM size | TCAM | | | Update logic | |
|---|---|---|---|---|---|
| (D×W) | LUTRAMs | SLICEM | FFs | LUTs | SLICEL |
| 512 × 36 | 4096 | 1024 | 1165 | 624 | 210 |
| 1024 × 144 | 32768 | 8192 | 2690 | 1220 | 582 |

of a 1 × 18 TCAM data (00000x11111100xxxx), shown in Fig. 2, requires the instantiation of the LUT64M primitives with (64'h0000000000000003) for LUTA, (64'h8000000000000000) for LUTB, and (64'h000000000000ffff) for LUTC.

We employed the Xilinx ISE 14.7 synthesis and implementation tool to design and evaluate our proposed architecture. The Xilinx post-place-and-route results were used to report the maximum achievable clock rate and resource consumption of DURE. The power consumption for this implementation was estimated using the XPower Analyzer tool.

Table II lists the FPGA resource utilization for two different TCAM table sizes (case I: 512 × 36 and case II: 1024 × 144) using the DURE technique. This implementation of DURE does not contain a priority encoder. The DURE implementation cases I and II utilize 4096 and 32 768 LUTRAM cells, respectively, for storing the contents of the respective TCAM tables with 624 and 1220 LUTs, respectively, for implementing the update logic. As can be observed, the resource utilization overhead for implementing the update logic using the proposed DURE is considerably small.

## B. Scalability of DURE

The scalability of DURE is evaluated through important metrics, such as the FPGA resource utilization, clock rate, power consumption, and update rate for TCAM table implementations of different sizes. To this end, TCAM tables of width $W = 36$, 72, 108, and 144 against depths of $D = 64$, 128, 256, and 512 are compared in Fig. 5.

The utilization of FPGA resources, such as LUTRAM and slice registers (SRs) for different TCAM sizes implemented using DURE, is shown in Fig. 5(a) and (b) using a logarithmic scale. It is clear from these figures that the hardware consumption of DURE increases proportionally with an increase in the width or depth of the TCAM table size. DURE efficiently utilizes the LUTRAM resources for storing TCAM bits by implementing 18 TCAM bits in the four LUTRAM cells of an SLICEM. The LUTRAM consumed in the implementation of a TCAM table of depth $D$ and width $W$ using DURE can be calculated from

$$\text{No. of LUTRAMs used} = D \times [(W/18) \times 4]. \tag{1}$$

The clock rates achieved by DURE for different TCAM sizes are shown in Fig. 5(c). DURE is able to achieve a linear decrease in clock frequency with a linear increase in the depth or width of the implemented TCAM. The critical path delay in DURE increases owing to an increase in the lengths of the longest wires connecting the LUTRAM cells. Fig. 5(c) also shows that DURE can achieve considerably high operating speeds from 221 up to 475 MHz for the given TCAM sizes. Fig. 5(d) shows that the dynamic power consumption of DURE increases linearly with an increase in the TCAM table width or depth. In SRAM-based TCAM implementations, the primary source of power consumption is the SRAM memory, which stores the TCAM contents. As previously mentioned, DURE efficiently utilizes the LUTRAM resources to store the TCAM contents. Moreover, avoiding the

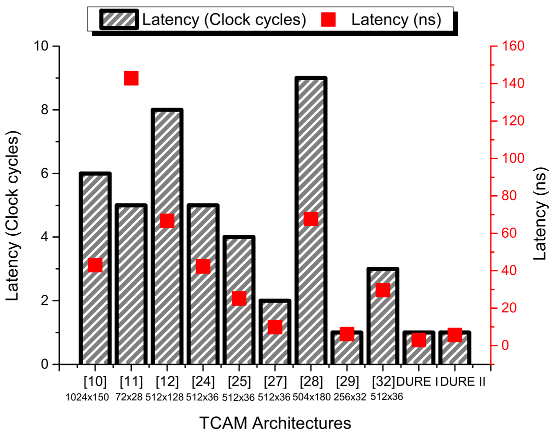


Fig. 6. Search latency comparison.

use of LUTs and their associated programmable interconnects for AND-cascading the potential match bits and instead using the carry chains for this purpose leads to minimal power consumption in DURE.

The update rate performance achieved in DURE is shown in Fig. 5(e) and is calculated according to

$$\text{Update rate} = \frac{\text{Clock rate (MHz)}}{\text{\# of cycles required for an update}}. \tag{2}$$

The update operation takes 65 clock cycles. The update rate also decreases linearly with an increase in the width or depth of the implemented TCAM tables. DURE can achieve high update rates of 7.3∼3.4 million updates per second.

### C. Comparison With Existing FPGA-Based TCAM Solutions

We compared the proposed DURE architecture with existing TCAM architectures for FPGAs in terms of the search latency, update rate, FPGA resource utilization, performance per area, and energy efficiency. We implemented the two TCAM sizes of 18 kbit (case I: $512 \times 36$) and 144 kbit (case II: $1024 \times 144$) in DURE for the sake of appropriate comparisons with the related methods.

*1) Search Latency:* The comparison of DURE with other FPGA-based TCAM architectures in terms of the search latency is shown in Fig. 6. The FPGA-based TCAM architectures in [11]–[15], [28]–[30], and [33] achieved search latencies of 6, 5, 8, 5, 4, 2, 9, 1, and 3 clock cycles, respectively, and thus, the match results against input words are processed in 43, 143, 67, 42, 25, 10, 68, 6, and 29.7 ns, respectively. DURE is able to achieve a search latency of a single clock cycle. For the two cases implemented in DURE, the delay time achieved in case I is 2.987 ns and that in case II is 5.723 ns. Thus, the time required by the search operation for DURE is the lowest among the existing FPGA-based TCAM approaches. Thus, the single-cycle low access time search makes the proposed DURE method a promising architecture for implementing applications for which low response times are desired, such as TLB caches.

*2) Update Rate:* DURE is compared with the related TCAM architectures in terms of the update rate in Table III. DURE cases I and II, implementing TCAM tables of size $512 \times 36$ and $1024 \times 144$, respectively, achieve clock rates of 335 and 175 MHz. The update operation in DURE is performed in 65 clock cycles. Thus, the update rates achieved in DURE cases I and II are 5.15 million and 2.7 million updates per second, respectively.

TABLE III
UPDATE RATE COMPARISON

| Architecture | Speed (MHz) | Update cycles | Update rate (Million updates per second) |
|---|---|---|---|
| Jiang [11] | 139 | 33 | 4.21 |
| REST [12] | 35 | 513 | 0.07 |
| Xilinx 2017 [15] | 120 | 33 | 3.64 |
| HP-TCAM [14] | 118 | 513 | 0.23 |
| Z-TCAM [13] | 159 | 513 | 0.31 |
| UE-TCAM [28] | 202 | 513 | 0.4 |
| Qian [29] | 109 | 33 | 3.3 |
| Locke [30] | 100 | 17 | 9.6 |
| Syed [33] | 101 | 513 | 0.2 |
| DURE case I | 335 | 65 | 5.15 |
| DURE case II | 175 | 65 | 2.7 |

The BRAM-based TCAM architectures presented in the previous studies [12]–[14], [28], [33] achieve clock rates of 35, 118, 159, 202, and 101 MHz, respectively, while the high update latency of 513 cycles in BRAM-based TCAM architectures results in lower update rates of 0.07, 0.23, 0.31, 0.4, and 0.2 million updates per second, respectively. The TCAM architectures [11], [15], [29], implemented in distributed RAM with an update latency of 33 cycles, achieve the update rates of 4.21, 3.64, and 3.3 million updates per second, respectively. The $256 \times 32$ size TCAM table implemented in [30] with an update latency of 17 cycles achieves an update rate of 9.6 million updates per second.

However, it should be noted that search operations remain suspended during the update process in existing SRAM-based TCAMs [11]–[15], [28], [29], [33]. This means that the update rates listed in Table III for these works are achievable when all of the system processing cycles are dedicated to update, exclusively, making the TCAM completely unavailable for any search operations. This scenario is not affordable since the availability of a TCAM for search operations is of prime importance. Therefore, these TCAMs accommodate the search and update operations by dedicating a major portion of the processing cycles toward searches and by managing updates in the remaining cycles called bubbles [15]. Hence, the effective update rates of these TCAMs are very low compared with DURE which supports a nonblocking update process. DURE does not affect the search rate of the system, since the search operations can be performed simultaneous to the update operations. Moreover, the effective search rates of the existing TCAMs are also lower than achievable clock rates owing to their blocking updates support.

*3) FPGA Resource Utilization:* Table IV compares the FPGA resource utilization in terms of LUTRAMs, LUTLs, SRs, BRAMs, and slices along with the corresponding utilization percentages for the various TCAM architectures. Column 2 lists the FPGA device used to implement the TCAM table sizes listed in Column 3. The LUTRAM utilization of architectures implemented in distributed RAM is listed in Column 4, while the BRAM utilization of BRAM-based architectures is given in Column 7. The maximum TCAM

TABLE IV
FPGA RESOURCE UTILIZATION FOR IMPLEMENTING VARIOUS TCAM ARCHITECTURES

| Architecture | FPGA | TCAM Size ($D \times W$) | LUTRAMs usage (%) | LUTLs usage (%) | Slice registers usage (%) | BRAMs (36 kbit) usage (%) | FPGA Slices usage (%) |
|---|---|---|---|---|---|---|---|
| Jiang [11] | XC7V2000T | $1024 \times 150$ | 20480 (5.94%) | 61624 (7%) | 37556 (1.54%) | 0 | 20526 (6.72%) |
| REST [12] | XC7K70T | $72 \times 28$ | 8 (0.06%) | 130 (0.47%) | 390 (0.48%) | 1 (0.74%) | 77 (0.75%) |
| Xilinx 2017 [15] | XC7V2000T | $512 \times 128$ | 8875 (2.58%) | 27559 (3.14%) | 35068 (1.44%) | 3 (0.23%) | 12011 (3.93%) |
| HP-TCAM [14] | XC6VLX760 | $512 \times 36$ | 0 | 6546 (1.92%) | 2670 (0.28%) | 56 (7.78%) | 1637 (1.38%) |
| Z-TCAM [13] | XC6VLX760 | $512 \times 36$ | 0 | 4462 (1.30%) | 2178 (0.23%) | 40 (5.56%) | 1116 (0.94%) |
| UE-TCAM [28] | XC6VLX760 | $512 \times 36$ | 0 | 3652 (1.07%) | 1758 (0.19%) | 32 (4.44%) | 913 (0.77%) |
| Qian [29] | XC6VLX75T | $504 \times 180$ | 15552 (93%) | 24715 (83%) | 35342 (38%) | 0 | 10067 (86%) |
| Locke [30] | XC5VLX220 | $256 \times 32$ | 4096 (11.23%) | 1527 (1.50%) | 341 (0.25%) | 0 | 1406 (4.07%) |
| Syed [33] | XC6VLX760 | $512 \times 36$ | 0 | 3013 (0.88%) | 552 (0.06%) | 32 (4.44%) | 754 (0.64%) |
| DURE case I | XC6VLX760 | $512 \times 36$ | 4096 (3%) | 1605 (0.47%) | 1174 (0.13%) | 0 | 1668 (1.40%) |
| DURE case II | XC6VLX760 | $1024 \times 144$ | 32768 (24%) | 3039 (0.89%) | 2700 (0.28%) | 0 | 9654 (8.14%) |

TABLE V
PERFORMANCE PER AREA COMPARISON WITH EXISTING FPGA-BASED TCAM ARCHITECTURES

| Architecture | TCAM size ($D \times W$) | FPGA Device (Technology) | BRAMs (36 kbit) | Slices usage | Normalized Area (slices) | Speed (MHz) | Normalized speed (MHz) | Throughput (Gbit/s) | P/A$^{(a)}$ |
|---|---|---|---|---|---|---|---|---|---|
| Jiang [11] | $1024 \times 150$ | Virtex-7 (28 nm) | 0 | 20526 | – | 199 | 139 | 20.9 | 1.04 |
| REST [12] | $72 \times 28$ | Kintex-7 (28 nm) | 1 | 77 | 101 | 50 | 35 | 0.98 | 0.7 |
| Xilinx 2017 [15] | $512 \times 128$ | Virtex-7 (28 nm) | 3 | 12011 | 12083 | 171 | 120 | 15.36 | 0.65 |
| HP-TCAM [14] | $512 \times 36$ | Virtex-6 (40 nm) | 56 | 1637 | 2981 | 118 | – | 4.25 | 0.73 |
| Z-TCAM [13] | $512 \times 36$ | Virtex-6 (40 nm) | 40 | 1116 | 2076 | 159 | – | 5.72 | 1.41 |
| UE-TCAM [28] | $512 \times 36$ | Virtex-6 (40 nm) | 32 | 913 | 1681 | 202 | – | 7.26 | 2.21 |
| Qian [29] | $504 \times 180$ | Virtex-6 (40 nm) | 0 | 10067 | – | 109 | – | 19.26 | 0.98 |
| Locke [30] | $256 \times 32$ | Virtex-5 (65 nm) | 0 | 1406 | – | 100 | 163 | 5.2 | 0.95 |
| Syed [33] | $512 \times 36$ | Virtex-6 (40 nm) | 32 | 754 | 1522 | 101 | – | 3.64 | 1.22 |
| DURE case I | $512 \times 36$ | Virtex-6 (40 nm) | 0 | 1668 | – | 335 | – | 12.06 | 3.7 |
| DURE case II | $1024 \times 144$ | Virtex-6 (40 nm) | 0 | 9654 | – | 175 | – | 25.2 | 2.67 |

(a) **P/A**: Performance per Area ((Mbit/s)/$Slice$)

bit storage capacity of the TCAM architectures [11], [15], [29] implemented in distributed RAM is higher, because these configure the LUTRAMs of a SLICEM as a 32×6 simple dual-port RAM, thereby implementing 30 TCAM bits per SLICEM compared with 18 TCAM bits per SLICEM in the case of DURE. However, the given architectures achieve this higher TCAM bit storage capacity at the cost of higher logic consumption in the bitwise ANDing of a large number of wide bit vectors, as shown in Column 5. On the other hand, the proposed DURE architecture utilizes the carry chain structures available in FPGA slices to AND-cascade the match bits from the LUTRAMs, thereby dispensing of the extra logic and associated routing resources consumed in bitwise ANDing.

*4) Performance per Area:* The performances of the TCAM architectures are metricized in terms of the processing throughput Mbit/s, described using (3), and the area, which is measured as the number of FPGA slices required per word for the implemented TCAM table

$$\text{Throughput (Gbit/s)} = \text{Clock rate (MHz)} \times \text{Width of the TCAM words.} \quad (3)$$

Because the FPGA-based TCAM architectures under consideration were implemented in different embedded memory resources, such as BRAM or distributed RAM, in order to fairly compare the given TCAM architectures, we measured their normalized area based on the observation that 384 BRAM bits correspond to one six-input LUT, while 1536 BRAM bits are similar to one slice comprising four six-input LUTs on a Virtex-6 FPGA. This is described using (5), which has been adapted and modified from [37]–[39]

$$\text{Normalized Area(slices)} = \#\text{ of FPGA slices} + [\#\ of\ 36\text{ kbit size BRAMs} \times 24]. \quad (4)$$

Table V presents a comparison between DURE and the other architectures in terms of the performance per area metric, which is calculated according to (5), modified from [37]–[39]

$$\frac{\text{Performance}}{\text{Area}} = \frac{\text{Throughput (Mbit/s)}}{\frac{\text{Normalized Area (slices)}}{\text{Depth of the TCAM table}}}. \quad (5)$$

The clock rates of the TCAM designs implemented in different CMOS technologies are normalized using (6), which has been modified from [40], [41]

$$F^* = F \times \left[\frac{\text{Technology (nm)}}{40\text{ (nm)}}\right] \times \left[\frac{1.0}{VDD}\right] \quad (6)$$

where $F$ represents the clock rate and $F^*$ denotes the normalized clock rate for 40-nm CMOS technology (Virtex-6) and a supply voltage of 1.0 V.

Case I, where a $512 \times 36$ size TCAM table is implemented in 1668 FPGA slices, achieves a throughput of 12.06 Gbit/s. Therefore, the performance per area achieved in case I is 3.7 Mbit/s/slice. Thus, DURE case I is able to achieve a performance per area that is 67% higher than that of [28], which was already the highest among the related TCAM architectures.

TABLE VI
EDP COMPARISON FOR VARIOUS FPGA-BASED TCAM SOLUTIONS

| Architecture | TCAM size (D × W) | FPGA (Technology) | Delay (ns) | Search cycles | Power (W) | Normalized delay T*(ns) | Normalized power P*(W) | Energy (fJ/bit/search) | EDP (ns.fJ/bit/search) |
|---|---|---|---|---|---|---|---|---|---|
| Jiang [11] | 1024×150 | Virtex-7 (28 nm) | 5.03 | 6 | 1.9 | 7.18 | 2.7 | 180 | 1290 |
| REST [12] | 72×28 | Kintex-7 (28 nm) | 20 | 5 | 0.11 | 28.57 | 0.16 | 798 | 22817 |
| Xilinx 2017 [15] | 512×128 | Virtex-7 (28 nm) | 5.85 | 8 | 1.01 | 8.34 | 1.442 | 220 | 1834 |
| HP-TCAM [14] | 512×36 | Virtex-6 (40 nm) | 8.47 | 5 | 0.19 | – | – | 102.2 | 865 |
| Z-TCAM [13] | 512×36 | Virtex-6 (40 nm) | 6.29 | 4 | 0.11 | – | – | 58.9 | 371 |
| UE-TCAM [28] | 512×36 | Virtex-6 (40 nm) | 4.95 | 2 | 0.08 | – | – | 42.3 | 210 |
| Qian [29] | 504×180 | Virtex-6 (40 nm) | 7.52 | 9 | 2.5 | – | – | 263 | 2414 |
| Locke [30] | 256×32 | Virtex-5 (65 nm) | 10 | 1 | 0.09 | 6.15 | 0.055 | 68 | 413 |
| Syed [33] | 512×36 | Virtex-6 (40 nm) | 9.9 | 3 | 0.043 | – | – | 23.3 | 231 |
| DURE case I | 512×36 | Virtex-6 (40 nm) | 2.99 | 1 | 0.05 | – | – | 28 | 84 |
| DURE case II | 1024×144 | Virtex-6 (40 nm) | 5.72 | 1 | 0.48 | – | – | 32.8 | 187 |

Case II implements a larger TCAM table of size 144 kbit ($1024 \times 144$) in 9654 FPGA slices and achieves a throughput of 25.2 Gbit/s. Therefore, it achieves a performance per area of 2.67 Mbit/s/slice. The similar sized TCAM design in [11] achieves a performance per area of 1.04 Mbit/s/slice. Table V shows that DURE case II is able to achieve a performance per area that is about 2.5 times higher than that of [11], which was already the highest among the other large-sized TCAM architectures [11], [15], [29]. Thus, overall DURE provides a high-speed and resource-efficient TCAM architecture.

*5) Energy Efficiency:* Table VI compares the performances of the two cases implemented with DURE with other FPGA-based TCAM solutions in terms of the energy-delay product (EDP). The power consumption results of previous methods for implementing the TCAM architectures in different CMOS technologies are normalized using (7), which has been taken from [41] and modified

$$P^* = P \times \left[\frac{40\ (\text{nm})}{\text{Technology}\ (\text{nm})}\right] \times \left[\frac{1.0}{VDD}\right]^2 \quad (7)$$

where $P$ represents the dynamic power consumption, while $P^*$ represents the normalized dynamic power consumption for 40- nm CMOS technology (Virtex-6) and a supply voltage of 1.0 V.

Case I consumes a dynamic power of 52 mW and has a delay time of 2.99 ns. Thus, the energy consumption in DURE I is 28 fJ/bit/search and the EDP is 84 ns.fJ/bit/search. The EDP achieved in case I is 2.5 times lower than that of UE-TCAM [28], which is the lowest among the other FPGA-based TCAM architectures. Case II, which implements a larger TCAM, consumes a dynamic power of 483 mW and has a delay time of 5.72 ns. Thus, its EDP is 187 ns.fJ/bit/search, which is about seven times lower than that of the similar size 150-kbit TCAM implementation in [11]. Thus, DURE is also a highly energy-efficient TCAM architecture.

## V. CONCLUSION

This paper presented a technique for implementing a dynamically reconfigurable TCAM in SRAM-based FPGAs with a higher energy consumption and resource utilization efficiency. The proposed TCAM design methodology, called DURE, employs an FPGA's distributed RAM resources and configures the LUTRAMs of an SLICEM as quad-port RAM in its implementation. DURE dynamically reconfigures only the LUTRAM associated with the word being updated and, at the same time, allows lookup operations to be performed. DURE achieved a better performance than other methods in terms of the lookup and update operations when tested on a Virtex-6 FPGA device. It achieved an energy efficiency and performance per area that were at least 2.5 times and 67% better than those of the other FPGA-based TCAM solutions.

## REFERENCES

[1] J.-Y. Huang and P.-C. Wang, "TCAM-based IP address lookup using longest suffix split," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 976–989, Apr. 2018.

[2] G. Brebner and W. Jiang, "High-speed packet processing using reconfigurable computing," *IEEE Micro*, vol. 34, no. 1, pp. 8–18, Jan. 2014.

[3] L.-Y. Huang *et al.*, "ReRAM-based 4T2R nonvolatile TCAM with 7x NVM-stress reduction, and 4x improvement in speed-wordlength-capacity for normally-off instant-on filter-based search engines used in big-data processing," in *Proc. Symp. VLSI Circuits Digest Tech. Papers*, 2014, pp. 1–2.

[4] X.-T. Nguyen, T.-T. Hoang, H.-T. Nguyen, K. Inoue, and C.-K. Pham, "An FPGA-based hardware accelerator for energy-efficient bitmap index creation," *IEEE Access*, vol. 6, pp. 16046–16059, 2018.

[5] M. Imani, A. Rahimi, and T. S. Rosing, "Resistive configurable associative memory for approximate computing," in *Proc. Design, Automat. Test Europe Conf. Exhib. (DATE)*, 2016, pp. 1327–1332.

[6] H.-J. Tsai *et al.*, "Energy-efficient TCAM search engine design using priority-decision in memory technology," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 3, pp. 962–973, Mar. 2017.

[7] 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.

[8] A. G. Hanlon, "Content-addressable and associative memory systems," *IEEE Trans. Electron. Comput.*, vol. EC-15, no. 4, pp. 509–521, Aug. 1966.

[9] I. Arsovski *et al.*, "1.4 gsearch/s 2-Mb/mm$^2$ TCAM using two-phase-pre-charge ML sensing and power-grid pre-conditioning to reduce Ldi/dt power-supply noise by 50%," *IEEE J. Solid-State Circuits*, vol. 53, no. 1, pp. 155–163, Jan. 2018.

[10] T. Mishra and S. Sahni, "PETCAM—A power efficient TCAM architecture for forwarding tables," *IEEE Trans. Comput.*, vol. 61, no. 1, pp. 3–17, Jan. 2012.

[11] 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.

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

[13] 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.

[14] 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.

[15] *Ternary Content Addressable Memory (TCAM) Search IP for SDNet Version 1.0*, San Jose, CA, USa, Xilinx, Xilinx Product Guide PG190, Nov. 2017.

[16] Z. Wang, H. Che, M. Kumar, and S. K. Das, "CoPTUA: Consistent policy table update algorithm for TCAM without locking," *IEEE Trans. Comput.*, vol. 53, no. 12, pp. 1602–1614, Dec. 2004.

[17] P. He, W. Zhang, H. Guan, K. Salamatian, and G. Xie, "Partial order theory for fast TCAM updates," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 217–230, Feb. 2018.

[18] S. Mittal, "A survey of techniques for architecting TLBs," *Concurrency Comput., Pract. Exper.*, vol. 29, no. 10, p. e4061, 2017.

[19] J. R. Haigh and L. T. Clark, "High performance set associative translation lookaside buffers for low power microprocessors," *Integration*, vol. 41, no. 4, pp. 509–523, 2008.

[20] S. Tamimi, Z. Ebrahimi, B. Khaleghi, and H. Asadi, "An efficient SRAM-based reconfigurable architecture for embedded processors," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 38, no. 3, pp. 466–479, Mar. 2019.

[21] K. Aasaraai and A. Moshovos, "NCOR: An FPGA-friendly nonblocking data cache for soft processors with runahead execution," *Int. J. Reconfigurable Comput.*, vol. 2012, p. 6, Jan. 2012.

[22] M. Biglari, K. M. Barijough, M. Goudarzi, and B. Pourmohseni, "A fine-grained configurable cache architecture for soft processors," in *Proc. 18th CSI Int. Symp. Comput. Archit. Digit. Syst. (CADS)*, 2015, pp. 1–6.

[23] H. Park, H. So, and H. Lee, "Application specific cache design using STT-RAM based block-RAM for FPGA-based soft processors," *IEICE Electron. Exp.*, vol. 15, no. 10, 2018, Art. no. 20180330.

[24] P. He, W. Zhang, H. Guan, K. Salamatian, and G. Xie, "Partial order theory for fast TCAM updates," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 217–230, Feb. 2018.

[25] D. Shah and P. Gupta, "Fast updating algorithms for TCAM," *IEEE Micro*, vol. 21, no. 1, pp. 36–47, Jan. 2001.

[26] T. Mishra and S. Sahni, "DUOS—Simple dual TCAM architecture for routing tables with incremental update," in *Proc. IEEE Symp. Comput. Commun.*, Jun. 2010, pp. 503–508.

[27] 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.

[28] 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 Conf. (TENCON)*, Nov. 2015, pp. 1–6.

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

[30] K. Locke. (2011). *Xilinx Application Note: XAPP1151—Parameterizable Content-Addressable Memory*. [Online]. Available: http://www.xilinx.com

[31] 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.

[32] P. Maidee, "Multiplexer-based ternary content addressable memory," U.S. Patent 9 653 165, May 16 2017.

[33] F. Syed, Z. Ullah, and M. K. Jaiswal, "Fast content updating algorithm for an SRAM-based TCAM on FPGA," *IEEE Embedded Syst. Lett.*, vol. 10, no. 3, pp. 73–76, Sep. 2018.

[34] *Virtex-6 FPGA Configurable Logic Block User Guide UG364 (Version 1.2)*, Xilinx, San Jose, CA, USA, 2012. [Online]. Available: http://www.xilinx.com

[35] H. Parandeh-Afshar, G. Zgheib, P. Brisk, and P. Ienne, "Routing wire optimization through generic synthesis on FPGA carry chains," in *Proc. 20th Int. Workshop Logic Synth.*, San Diego, CA, USA, 2011, pp. 1–8.

[36] I. Ullah, U. Afzaal, Z. Ullah, and J.-A. Lee, "High-speed configuration strategy for configurable logic block-based TCAM architecture on FPGA," in *Proc. 21st Eur. Conf. Digit. Syst. Design (DSD)*, 2018, pp. 16–21.

[37] H. Nakahara, T. Sasao, H. Iwamoto, and M. Matsuura, "LUT cascades based on edge-valued multi-valued decision diagrams: Application to packet classification," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 6, no. 1, pp. 73–86, Mar. 2016.

[38] H. Nakahara, T. Sasao, and M. Matsuura, "An update method for a CAM emulator using an LUT cascade based on an EVMDD (k)," in *Proc. IEEE 44th Int. Symp. Multiple-Valued Log.*, May 2014, pp. 1–6.

[39] I. Sourdis, D. N. Pnevmatikatos, and S. Vassiliadis, "Scalable multigigabit pattern matching for packet inspection," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 2, pp. 156–166, Feb. 2008.

[40] P.-T. Huang and W. Hwang, "A 65 nm 0.165 fJ/bit/search 256 × 144 TCAM macro design for IPv6 lookup tables," *IEEE J. Solid-State Circuits*, vol. 46, no. 2, pp. 507–519, Feb. 2011.

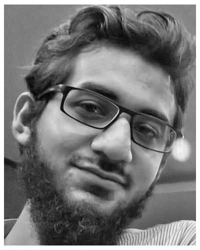
[41] O. T.-C. Chen and R. R.-B. Sheen, "A power-efficient wide-range phase-locked loop," *IEEE J. Solid-State Circuits*, vol. 37, no. 1, pp. 51–62, Jan. 2002.

**Inayat Ullah** (S'18) received the bachelor's degree in computer systems engineering from the University of Engineering and Technology, Peshawar, Pakistan, in 2007. He is currently working toward the Ph.D. degree at the College of Electronics and Information Engineering, Chosun University, Gwangju, South Korea.

He has 11 years of experience, including eight years of teaching experience. His current research interests include memory architecture, parallel processing, reconfigurable computing, and embedded systems.

**Zahid Ullah** (M'16) received the B.Sc. degree (Hons.) in computer systems 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, Seoul, 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. He has published prestigious journal and conference papers, and he holds patents in his name in the field of field-programmable gate array (FPGA)-based ternary content-addressable memory. His current research interests include low-power/high-speed CAM design on FPGA, low-power/high-speed VLSI design, and embedded systems.



**Umar Afzaal** received the B.E. degree in electrical engineering from the NED University of Engineering and Technology, Karachi, Pakistan, in 2016. He is currently working toward the Ph.D. degree at the Department of Computer Engineering, Chosun University, Gwangju, South Korea.

He is also a Research Assistant with the Computer Systems Lab, Department of Computer Engineering, Chosun University. His current research interests include approximate computing, fault-tolerant computing, and genetic design engineering.

**Jeong-A Lee** (M'84–SM'01) received the B.S. degree (Hons.) in computer engineering from Seoul National University, Seoul, South Korea, in 1982, the M.S. degree in computer science from Indiana University Bloomington, Bloomington, IN, USA, in 1985, and the Ph.D. degree in computer science from the University of California at Los Angeles, Los Angeles, CA, USA, in 1990.

From 1990 to 1995, she was an Assistant Professor with the Department of Electrical and Computer Engineering, University of Houston, Houston, TX, USA. Since 1995, she has been with Chosun University, Gwangju, South Korea. From 2008 to 2009, she served as a Program Director of the ECE Division, National Research Foundation of Korea. She has authored or coauthored more than 100 reviewed journal and conference papers. Her current interests include high-performance computer architectures, memory architecture, approximate computing, self-aware computing, and reliable computing.

Dr. Lee is a member of the National Academy of Engineering in South Korea.