# FracTCAM: Fracturable LUTRAM-Based TCAM Emulation on Xilinx FPGAs

Ali Zahir, Shadan Khan Khattak, Anees Ullah, Pedro Reviriego, Fahad Bin Muslim, and Waleed Ahmad

***Abstract*— In this brief, we present FracTCAM, an efficient methodology for ternary content addressable memory (TCAM) emulation on Xilinx field-programmable gate arrays (FPGAs) by leveraging primitive architectural resources. The proposed methodology exploits the fracturable nature of lookup table random access memories (LUTRAMs) and built-in slice flip-flops for deeper pipelining. Multiple slices can be combined together to build deeper and wider TCAMs using ANDing operations. This results in TCAM implementations that achieve lower resources utilization, lower delay, and power consumption. A comparison with the existing schemes shows that FracTCAM consistently achieves the best performance per area (PA) and performance per area per watt (PAW).**

***Index Terms*— Field-programmable gate array (FPGA), packet classification, partial reconfiguration, ternary content addressable memories (TCAMs).**

## I. Introduction

Ternary content addressable memories (TCAMs) enable very fast content membership checking and are widely used in network switches and routers to find the best matching route on a table. If a content resides in a TCAM it would trigger a binary check flag and the location in memory. This is different from how a standard memory works and requires specialized circuitry which is more complex when compared to random access memories (RAMs) [1]. Traditionally, TCAMs are implemented as specialized memory blocks in application-specific integrated circuits (ASICs) for high-speed routers [2]. However, with the emerging requirements for programability in data planes for software-defined networks (SDNs), field-programmable gate arrays (FPGAs) are increasingly being adopted [3], [4]. Unfortunately, modern state-of-the-art FPGAs have no built-in blocks for TCAMs leaving the design to emulate them with other resources. TCAM emulation on an FPGA requires resources for storage, matching logic, and priority encoding. The most expensive part of a TCAM is the storage part, and its optimization has been actively pursued by researchers. TCAM-emulation on SRAM-based FPGAs has been explored with four different kinds of resources, that is, flip-flops (FFs), block-RAMs (BRAMs), lookup tables (LUTs), and LUT RAMs (LUTRAMs). FF-based TCAMs utilize in-slice FFs as TCAM storage memory [5]–[8]. Since each FF stores a single bit of information the interconnect complexity hugely increases and since many architectural constraints impose that an LUT–FF pair be used, many of the LUTs will be used as pass-through resulting in wastage of resources. BRAM-based TCAM emulation on SRAM-based FPGA has been widely investigated by researchers in [9], [10], and [11]–[17]. However, the efficient utilization of BRAM for TCAM emulation is limited by theoretical bounds, that is, we need at least an SRAM/TCAM bit ratio of $2^9/9$ when compared to LUTs or LUTRAM-based TCAMs that need $2^6/6$ [18] so 5× more. LUT-based TCAM has first been introduced by Reviriego *et al.* [19]. The authors showed that LUTs, although combinational in nature, could be utilized to emulate TCAMs combined with the reconfiguration capabilities of modern SRAM-based FPGAs. PR-TCAM [19] used dual-output LUTs (i.e., $5 \times 2$ LUTs) for storing TCAM rules while fine-grain frame-level reconfiguration for rule update. BPR-TCAM [20] improves upon PR-TCAM [19] by leveraging built-in slice carry-chain to reduce the match-logic required in TCAMs. However, both these approaches are based on partial reconfiguration for updating TCAM stored rules. This makes them slow for updates as partial reconfiguration is orders of magnitude slower when compared to LUTRAM-based approaches which update circuits work at the operating frequency. The last FPGA resource that can be used for TCAM-emulation is LUTRAM, also called distributed RAM [18], [21]–[26]. Ullah *et al.* [21] have used LUTRAMs in a $6 \times 1$ configuration for TCAM storage and carry-chains for match-logic reduction in the same slice to achieve better performance per area (PA) and resource efficiency. D-TCAM [22], on the other hand, used LUTRAMs in a $6 \times 1$ configuration for TCAM storage and fine-grain pipelining by leveraging the built-in slice register to achieve a much better throughput (TP). However, no existing work has utilized the LUTRAMs in the $5 \times 2$ configuration (i.e., dual-output LUTRAMs) and all the FFs in SLICEM which can greatly improve not only the storage density but also TP and PA.

In this brief, FracTCAM a TCAM emulation scheme that utilizes the fracturable LUTRAMs available in SLICEM on Xilinx FPGA and the built-in slice FFs is proposed. In our scheme each SLICEM implements an $8 \times 5$ TCAM when compared to the authors in D-TCAM [22], DURE [21], and BPR-TCAM [20] which can implement only a $4 \times 6$ TCAM, $1 \times 18$ TCAM, and $2 \times 16$ TCAM in the slice. It should be noted that BPR-TCAM [20] utilizes SLICEL in contrast to SLICEM used by D-TCAM [22], DURE [21], and FracTCAM. Therefore, our proposed method has significantly lower resource usage. In particular, when considering the widely used PA metric, FracTCAM achieves an improvement that ranges from 25% (for D-TCAM I versus FracTCAM I) to 311% (for D-TCAM II versus FracTCAM II) over the best existing scheme for the TCAM configurations considered. FracTCAM achieves the lowest normalized slice utilization for one of the configuration tested while for the other two it is only worse by 10% (for BPR-TCAM II versus FracTCAM II) and 15% (for BPR-TCAM III versus FracTCAM III),

Manuscript received April 7, 2020; revised August 3, 2020; accepted September 7, 2020. Date of publication October 8, 2020; date of current version November 24, 2020. This work was supported in part by the Higher Education Commission (HEC) Pakistan and the Ministry of Planning, Development and Special Initiatives under the National Centre for Cyber Security. The work of Pedro Reviriego was supported by Architecting Intelligent Cost-effective Central Offices to enable 5G/6G Tactile Internet (ACHILLES) under Project PID2019-104207RB-I00, in part by the Spanish Ministry of Science and Innovation for the Go2Edge Network under Grant RED2018-102585-T, and in part by the Madrid Community Research under Project TAPIR-CM-P2018/TCS-4496. *(Corresponding author: Anees Ullah.)*

Ali Zahir is with the Department of Electrical and Computer Engineering, COMSATS University Islamabad, Abbottabad Campus, Abbottabad 220101, Pakistan (e-mail: alizahir@cuiatd.edu.pk).

Shadan Khan Khattak is with the Department of Computer Engineering, College of Computer Science and Information Technology, King Faisal University, Al Ahsa 31982, Saudi Arabia (e-mail: snasrullah@kfu.edu.sa).

Anees Ullah is with the Department of Electronics Engineering, University of Engineering and Technology Peshawar, Abbottabad Campus, Abbottabad 220101, Pakistan (e-mail: aneesullah@uetpeshawar.edu.pk).

Pedro Reviriego is with the Department of Telematic Engineering, Universidad Carlos III of Madrid, 28911 Madrid, Spain (e-mail: revirieg@it.uc3m.es).

Fahad Bin Muslim is with the Department of Electronics Engineering, Iqra University, Islamabad 44000, Pakistan (e-mail: fahad@iqraisb.edu.pk).

Waleed Ahmad is with the Department of Electrical and Computer Engineering, University of Poonch, Rawalakot 12350, Pakistan (e-mail: drwaleedahmad@upr.edu.pk).

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

Digital Object Identifier 10.1109/TVLSI.2020.3026840

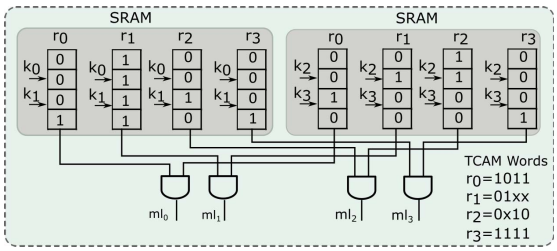


Fig. 1. Example of TCAM emulation using SRAMs.

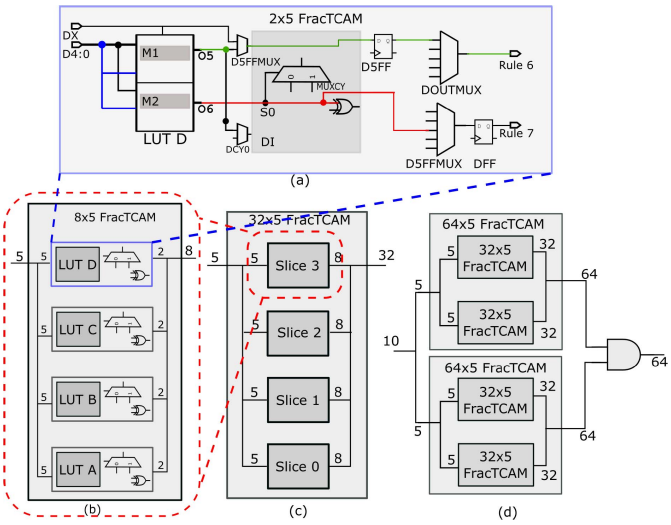


Fig. 2. (a) Architecture of the proposed method for mapping a $2 \times 5$ TCAM to a slice. (b) Combining four LUTs into $8 \times 5$ TCAM. (c) Combining four slices into a TCAM block. (d) Combining blocks to build larger TCAMs.

respectively. As for speed, FracTCAM is the best in two of the configuration tested and for the other two it is only worse by 2% (for D-TCAM III versus FracTCAM III) and 12% (for D-TCAM I versus FracTCAM I), respectively. This combination of high speed and low resource usage makes FracTCAM an attractive option for TCAM emulation on FPGAs.

The rest of this brief is organized as follows. Section II presents the proposed FracTCAM architecture. Section III evaluates FracTCAM and compares the results with existing works. Section IV concludes the brief.

## II. Proposed FracTCAM Architecture

TCAM emulation on SRAM-based FPGAs can be better understood with an example. Let us consider $N = 4$ and $W = 4$, that is, a $4 \times 4$ TCAM as depicted in Fig. 1, where $N$ is the depth or the number of rules and $W$ is the width or key size. The basic SRAM structure in Fig. 1 is a $4 \times 1$ SRAM or two-input LUT. Each rule r0, r1, r2, and r3 is mapped to a single two-input LUT or $4 \times 1$ SRAM memory. Since the key size is 4 and each $4 \times 1$ SRAM or two-input LUT has two input address lines, we need to split the TCAM into two blocks as shown in Fig. 1. The first block is indexed by k0 and k1 while the second by k2 and k3. The outputs from the four pairs of SRAM are merged with AND gates called match logic. Note that the priority encode logic is not shown here. The design of efficient TCAMs on FPGA boils down to the selection of SRAM implementation primitives and its depth and width extension. The proposed FracTCAM leverages the on-chip distributed LUTRAM located in SLICEM of the Xilinx FPGAs. To better understand the implementation of FracTCAM on FPGA, let us consider an $8 \times 5$ TCAM block. This TCAM has eight rules and a key width of 5, which can be implemented on a single SLICE. For example, LUTD can be used to implement $2 \times 5$ TCAM as shown in Fig. 2(a).

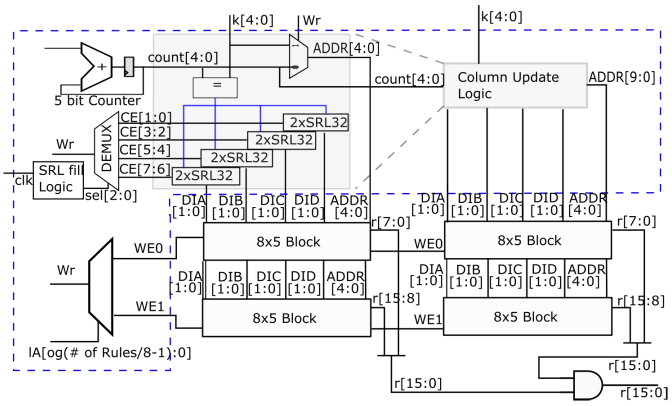


Fig. 3. Architecture of the update logic.

The keyword is connected to 5-bit LUT input (A4:A0) and two rules (Rule 6 and Rule 7) are read through O5 and O6. Memory M1 saves Rule 6 and memory M2 saves Rule 7. The rules can be updated using write address inputs, represented by blue lines in Fig. 2(a), through DI ports which are not shown here. To implement $8 \times 5$ TCAM, the four LUTs (LUTA, LUTB, LUTC, and LUTD) are stacked with common keyword and eight different rules through O5 and O6. This is shown in Fig. 2(b). FracTCAM utilizes 6-LUTs in dual-output mode along with the eight FFs in a single slice giving rise to $8 \times 5$ configuration (compared to $4 \times 6$ when single output LUTs are used). Moreover, this configuration efficiently utilizes the local routing matrix due to shared slice inputs. As shown in Fig. 2(a), O6 can be connected to DFF through DFFMUX and O5 can be routed to D5FF through D5FFMUX. In this way, a fully pipelined FracTCAM structure is designed which improves the performance while the resource utilization remains the same as that of a nonpipelined structure.

Multiple slices can be combined when the size of the TCAM is large. For example, increasing the depth of the FracTCAM requires more slices stacked vertically with the words of same width. This is illustrated in Fig. 2(c) for a FracTCAM with 32 rules of 5 bits. Four slices are stacked vertically, where each slice implements an $8 \times 5$ TCAM. The keywords to all four slices are common. The proposed methodology terms a $32 \times 5$ TCAM as a "block." The blocks are combined to produce larger TCAM dimensions. For instance, to increase the depth from 32 to 64, two blocks are vertically stacked with the same word size. Similarly, the width extension of FracTCAM is possible by using multiple blocks with the same depth in parallel. Fig. 2(d) demonstrates this as a FracTCAM with 64 rules of 10 bits. In this configuration, two blocks with the same depth are connected in parallel and each of the two blocks compares to the corresponding key bits and gives a match of a 64-bit vector. The final match vector is then obtained by doing a bitwise AND of the two match-vectors from $64 \times 5$ FracTCAMs. It is worth mentioning that the multibit wide AND was manually optimized (17% improvement in slice utilization for $512 \times 80$ configuration) using a tree-like structure instead of relying on the vendor design tool. Fig. 3 shows the update logic (highlighted in a blue dotted bounding box) for a $16 \times 10$ FracTCAM. It should be noted that an $8 \times 5$ block that is, single SLICEM has write enable "WE" line short for all the LUTRAMs. Furthermore, the "WE" lines are short for the same rules, for example, r[7:0] in first and second column have WE shorted into WE0 and r[15:8] into WE1. These lines are demultiplexed with row ID to identify which row is to be updated in current write cycle. The column update logic is responsible for the blocks in the same column that is, it has the same key lines. The column update logic consists of serial shift registers realized as SRL32 in

TABLE I
RESOURCE UTILIZATION OF UPDATE LOGIC FOR FRACTCAM

| TCAM Size (DxW) | | 512 x 40 | 1024 x 160 |
|---|---|---|---|
| TCAM | LUTs as Logic | 1024 | 7168 |
| | LUTRAM | 2048 | 16384 |
| | FFs | 4096 | 32768 |
| Update logic | LUTs as Logic | 150 | 382 |
| | LUTRAM | 64 | 256 |
| | FFs | 23 | 23 |

TABLE II
RESOURCE UTILIZATION, POWER (MILLIWATT), AND SPEED (MEGAHERZ) FOR DIFFERENT CONFIGURATION OF FRACTCAM

| Width | Parameters | Depth | | | |
|---|---|---|---|---|---|
| | | 64 | 128 | 256 | 512 |
| 20 | LUT as Logic | 64 | 128 | 256 | 512 |
| | LUT RAM | 128 | 256 | 512 | 1024 |
| | FFS | 256 | 512 | 1024 | 2048 |
| | Speed(MHz) | 874.9 | 862.8 | 803.9 | 741.8 |
| | Power(mW) | 11 | 16 | 27 | 53 |
| 40 | LUT as Logic | 128 | 256 | 512 | 1024 |
| | LUT RAM | 256 | 512 | 1024 | 2048 |
| | FFS | 512 | 1024 | 2048 | 4096 |
| | Speed(MHz) | 676.6 | 672 | 652.7 | 588.9 |
| | Power(mW) | 14 | 23 | 39 | 65 |
| 80 | LUT as Logic | 192 | 384 | 768 | 1536 |
| | LUT RAM | 512 | 1024 | 2048 | 4096 |
| | FFS | 1024 | 2048 | 4096 | 8192 |
| | Speed(MHz) | 721 | 664 | 726.7 | 677.5 |
| | Power(mW) | 28 | 34 | 73 | 119 |
| 160 | LUT as Logic | 448 | 896 | 1792 | 3584 |
| | LUT RAM | 1024 | 2048 | 4096 | 8192 |
| | FFS | 2048 | 4096 | 8192 | 16384 |
| | Speed(MHz) | 555.9 | 526.9 | 387.4 | 363.5 |
| | Power(mW) | 30 | 345 | 78 | 148 |

SLICEM. For an $8 \times 5$ column, we need eight SRL32 which can be implemented within a single SLICEM. In the write operation, a global 5-bit counter value is compared with the incoming key value and the binary value is written into either SRL32 enabled by a $1 \times 8$ demultiplexer. This demux is controlled by a 3-bit counter inside SRL fill logic which increments every 33 cycles of the global 5-bit counter once. All the SRL32s are filled in $8 \times 33$ cycles which are then written to an $8 \times 5$ block in 33 cycles. Therefore, a total of 297 cycles are required to update 8 rules in an $8 \times 5$ block that is, 38 cycles per rule.

## III. EVALUATION

The proposed architecture of FracTCAM was implemented on the Xilinx Virtex-7 28-nm XC7V2000TFHG1761-2L FPGA device with $-2$ speed grade. This device contains 1 221 600 LUTs, 344 800 LUTRAMs, 2 443 200 FFs, and 305 400 SLICEs. Vivado HLx 2016.3 design suite was used for the performance evaluation of different sizes of TCAMs. The key size of TCAM was varied from 20 to 160 bits and the number of rules from 64 to 512. It is worth mentioning that these sizes are chosen because of the constrains imposed by our basic building block that is, a SLICE able to implement a $8 \times 5$ TCAM. Therefore, keys are multiplicative factors of 5 and rules are multiplicative factors of 8. All the reported results are based on postplace and postroute implementation available in [28].

Table I shows the resources required for the update logic and TCAM storage for two FracTCAM configurations, that is, $512 \times 40$ and $1024 \times 160$. It can be noted from the table that the resources required for update logic are significantly less than those of the storage part of FracTCAM. In fact, looking at the resources used in DURE [21] for the update logic, our update logic resource usage is much lower. Table II shows resource utilization for FracTCAM of different sizes. It is worth mentioning that the table does not contain a match reduction or priority encoder. As previously discussed, the architecture of FracTCAM utilizes three FPGA resources: LUTRAMs for storing the TCAM rules, logic LUTs for implementing AND gates, and FFs registers for pipelining. It can be observed that resource utilization is directly related to the size of the TCAM. For example, the $64 \times 20$ TCAM can be divided into four $64 \times 5$ FracTCAM blocks. Each block consumes 32 LUTRAMs (8 SLICEM) and 64 FF for pipelining. It is worth mentioning that FracTCAM utilizes FFs within the same SLICE. Thus, each $64 \times 20$ block requires 8 SLICEMs per block and a total of four blocks, that is, 32 SLICEMs to implement TCAM cells. The logic LUTs in Table II include the implementation of AND-tree for match-logic.

The utilization of FFs corresponds to the number of blocks times the depth of TCAM. For example, $64 \times 20$ configuration takes $4 \times 64$, that is, 256 FFs. Similarly, $64 \times 80$ configuration takes $16 \times 64$ FFs. It is worth to mention here that Xilinx Virtex-7 FPGA supports eight FFs within a single SLICE. The proposed FracTCAM fully exploits this feature to maximize the SLICE resources utilization. In this way, a fully pipelined FracTCAM architecture is implemented without using extra SLICEs for pipeline registers.

The speed achieved by FracTCAMs of different sizes is also shown in Table II. It should be noted that FracTCAM inserts register between input and FracTCAM and between FracTCAM and Reduction OR Logic. It can be noted that FracTCAM achieves speeds from 363.5 up to 874.9 MHz for different sizes. The speed of FracTCAM degrades with its size however this degradation is mild and does not double as its size doubles. For example, the speed decreases by 10.1 and 29 MHz while moving from $64 \times 20$ to $128 \times 20$ and from $64 \times 160$ to $128 \times 160$, respectively. Similarly, this degradation is 198.3 and 165.1 MHz while moving from $64 \times 20$ to $64 \times 40$ and from $64 \times 80$ to $64 \times 160$, respectively. The maximum degradation is 339.2 MHz, which is 2.14 times, as we move from $256 \times 80$ to $256 \times 160$. Thus, the FPGA resource utilization and speed results in Table II shows that FracTCAMs scale well with size. Table II also represents the dynamic power consumption in milliwatts for different configurations of FracTCAM. These values are reported post implementation by Vivado power analyzer with default switching activity. It can be noted that the power consumption increases according to the size of the TCAM. For the smallest configuration, that is, $64 \times 20$, the dynamic power consumption is 11 mW, while for the largest design, that is, $512 \times 160$, it is 148 mW. Therefore, it can be concluded that the power consumption increases incrementally with configuration size.

Table III compares FracTCAM with state-of-the-art FPGA TCAMs in terms of several parameters defined in DURE [21] represented in (1)–(6). In case BRAMs are used, the number of normalized slices is calculated using the following equation:

$$N_{\text{slices}} = S + (\text{BRAMs}_{36\ \text{KBits}} * 24). \tag{1}$$

For fair comparison across different technology nodes used in FPGA, the normalized speed is derived from the following equation:

$$N_{\text{speed}} = \text{Speed} * \frac{\text{Technology(nm)}}{40\ \text{(nm)}} * \frac{1.0}{\text{VDD}}. \tag{2}$$

TP, another important parameter for TCAMs comparison, is calculated with the following equation:

$$\text{TP(Gbit/s)} = F(\text{MHz}) * \text{TCAM}_{\text{Width}}. \tag{3}$$

Upate rates are normally expressed in clock cycles and a million of updates per seconds (MUPS) as follows:

$$\text{Updaterate(MUPS)} = \frac{\text{Clockrate(MHz)}}{\text{clockcycles}}. \tag{4}$$

TABLE III

COMPARISON OF FRACTCAM WITH STATE-OF-THE-ART TCAMS: LUTS AS LUTRAMS/LUTS, SLICES IN REAL/NORMALIZED, SPEED AS REAL/NORMALIZED IN MEGAHERZ, UPDATE AS CLOCK CYCLES/RATE IN MPPS, POWER IN MILLIWATT, (TP IN Gbit/s), (PA IN (Mbit/s)/SLICE), AND (PAW IN (Mbit/s)/SLICE/WATT)

| Architecture | LUTS | FFs | BRAMs | Slices | Speed | TP | Update | Power | PA/PAW |
|---|---|---|---|---|---|---|---|---|---|
| REST-72x28[1] [14] | 8/130 | 390 | 1 | 77/101 | 50/35 | 0.98 | 513/0.07 | 113 | 0.7/6.2 |
| Locke-256x32[3] [24] | 4096/1527 | 341 | 0 | 1406/1406 | 100/162.5 | 5.2 | 17/9.6 | 253 | 0.95/3.8 |
| Qian-504x180[2] [23] | 15552/24715 | 35342 | 0 | 10067/10067 | 109/109 | 19.62 | 33/3.3 | 2548 | 0.98/0.4 |
| G-AETCAM-512x36[2] [5] | NA/NA | NA | NA | NA | 358/358 | NC | 1/358 | 119 | NC |
| RPE-TCAM-512x36[2] [27] | NA/NA | NA | NA | NA | 319/319 | NC | **1/319** | 71 | NC |
| HP-TCAM-512x36[2] [10] | 0/6546 | 2670 | 56 | 1637/2981 | 118/118 | 4.248 | 513/0.23 | 188 | 0.73/3.9 |
| Z-TCAM-512x36[2] [11] | 0/4462 | 2178 | 40 | 1116/2076 | 159/159 | 5.724 | 513/0.31 | 109 | 1.41/12.9 |
| UE-TCAM-512x36[2] [13] | 0/3652 | 1758 | 32 | 913/1681 | 202/202 | 7.272 | 513/0.4 | 78 | 2.21/28.3 |
| Syed-512x36[2] [16] | 0/3013 | **552** | 32 | 754/1522 | 101/101 | 3.636 | 513/0.2 | NA | 1.22/NC |
| Xilinx-512x128[1] [25] | 8875/27559 | 35068 | 3 | 12011/12083 | 171/119.7 | 15.3216 | 33/3.64 | 90 | 0.65/7.2 |
| DURE-I-512x36[2] [21] | 4096/1605 | 1174 | 0 | 1668/1668 | 335/335 | 12.06 | 65/5.15 | **50** | 3.7/74 |
| D-TCAM I-512x 36[2] [22] | NA/NA | NA | 0 | 968/968 | 460/**460** | **16.56** | NA | NA | 8.76/NC |
| BPR-TCAM I-512x40[1] [20] | 0/2560 | 1105 | 0 | 768/768 | 360/252.07 | 10.08 | NA | NA | 6.72/NC |
| *Frac-TCAM I-512x40[1]* | 2048/**1024** | 4096 | 0 | **768/768** | **588**/411.6 | 16.46 | 38/15.47 | 65 | **10.98/168.9** |
| D-TCAM II-512x 72[2] [22] | NA/NA | NA | 0 | 2357/2357 | 214/214 | 15.41 | NA | NA | 3.35/NC |
| BPR-TCAM II-512x80[1] [20] | 0/5120 | **1185** | NA | **1280/1280** | 188/79.8 | 12.77 | NA | NA | 2.55/NC |
| *Frac-TCAM II-512x80[1]* | 4096/**1536** | 8192 | 0 | 1408/1408 | **677/473.9** | **37.91** | **38/17.11** | **119** | **13.79/115.9** |
| D-TCAM III-512x 144[2] [22] | NA/NA | NA | 0 | 4835/4835 | 259/**259** | 37.3 | NA | NA | 3.95/NC |
| BPR-TCAM III-512x160[1] [20] | 0/10240 | **1345** | NA | **2560/2560** | 114/79.8 | 12.768 | NA | NA | 2.55/NC |
| *Frac-TCAM III-512x160[1]* | 8192/**3585** | 16384 | 0 | 2944/2944 | **364**/254.8 | **40.77** | **38/9.58** | **148** | **7.09/47.9** |
| DURE-II-1024x144[2] [21] | 32768/3039 | **2700** | 0 | 9654/9654 | 175/175 | 25.2 | 65/2.69 | 230 | 2.67/11.6 |
| Jiang-1024x150[1] [18] | 20480/61624 | 37556 | 0 | 20526/20526 | 199/139.3 | 20.895 | 33/4.21 | NA | 1.04/NC |
| *Frac-TCAM-IV-1024x160[1]* | **16384/7168** | 32786 | 0 | **5888/5888** | **357.1/250** | **39.95** | **38/9.40** | **190** | **6.95/36.6** |

[1]28nm, [2]40nm, [3]65nm, NA: Not Available, NC: Non Computable

A metric that is widely used to compare FPGA TCAMs in literature is PA expressed by the following equation:

$$\text{PA(Mbit/s/slices)} = \frac{\text{TP(Mbit/s)}}{\frac{N_{\text{slices}}}{\text{TCAM}_{\text{Depth}}}}. \tag{5}$$

Finally, the performance per area per watt (PAW) is calculated with the following equation:

$$\text{PAW (Mbit/s/slices/watt)} = \frac{\text{PA (Mbit/s/slices)}}{\text{Power(W)}}. \tag{6}$$

FracTCAM-I, II, and III, for slice resource usage, respectively, are 20%, 40%, and 39% more efficient than D-TCAM-I, II and III. Similarly, FracTCAM-I, II, and III are 22%, 68%, and 27% faster than D-TCAM-I, II, and III, respectively. Comparing with BPR-TCAM I, II, and III for slice utilization, FracTCAM-I, II, and III are shown to be the same, 10%, and 15% worse, respectively. However, in speed comparison, FracTCAM I, II, and III outperform BPR-TCAM I, II, and III by 22%, 68%, and 27%, respectively. Comparing with DURE I and DURE II, FracTCAM I and FracTCAM IV are 54% and 39% better in slice utilization while at the same time 23% and 43% better in speed. Update rate is not reported by BPR-TCAM and D-TCAM, however, in comparison to DURE I and DURE II, FracTCAM I and FracTCAM IV show more than 2 and $x$ time improvement in MUPS. G-AETCAM and RPE-TCAM have a very high MUPS but the authors did not reported resource utilization which is huge as FFs are used to design TCAM. Therefore, due to higher speed and resource utilization efficiency, FracTCAM shows significant improvement in metrics such as TP and PA as can be seen in Table III. The power consumption is also improved due to better slice utilization resulting in better PAW. Comparing DURE I and DURE II, FracTCAM I and FracTCAM IV show 128% and 215% improvements in PAW. Therefore, FracTCAM shows improvements in all the compared metrics and can be used for large FPGA-based TCAMs.

## IV. CONCLUSION

FracTCAM leverages the architectural features of Xilinx FPGAs to efficiently emulate TCAMs. LUTRAMs, configured in dual-output mode combined with built-in slice registers, found in the latest seven-series FPGAs, are utilized to propose a modular and scalable TCAM architecture. The basic building block of the proposed architecture is able to map an $8 \times 5$ TCAM compared to existing $4 \times 6$ TCAM, thus, almost doubling the utilization density. Furthermore, the utilization of in-slice registers to pipeline LUTRAM outputs enables high speed operation. Therefore, not only the logic utilization but also TP is enhanced leading to better PA when compared to existing approaches. The proposed update logic requires significantly less resources than existing FPGA TCAMs and is able to update all the rules in an $8 \times 5$ block simultaneously. Due to better dynamic power consumption, the proposed solution outperforms existing approaches in PA and PAW, which is very significant considering large size TCAM-emulation on SRAM-based FPGAs.

## REFERENCES

[1] K. Pagiamtzis and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: A tutorial and survey," *IEEE J. Solid-State Circuits*, vol. 41, no. 3, pp. 712–727, Mar. 2006.

[2] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2013, pp. 99–110.

[3] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 gbps as research commodity," *IEEE Micro*, vol. 34, no. 5, pp. 32–41, Sep. 2014.

[4] Xilinx. *SDNet Packet Processor User Guide*; UG1012 (v2018.1); Xilinx, San Jose, CA, USA, 2018.

[5] 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.

[6] 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.

[7] 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)*, Southend, U.K., Aug. 2018, pp. 328–332.

[8] H. Mahmood, Z. Ullah, O. Mujahid, I. Ullah, and A. Hafeez, "Beyond the limits of typical strategies: Resources efficient FPGA-based TCAM," *IEEE Embedded Syst. Lett.*, vol. 11, no. 3, pp. 89–92, Sep. 2019.

[9] M. Somasundaram, "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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] I. Ullah, Z. Ullah, and J.-A. Lee, "EE-TCAM: An energy-efficient SRAM-based TCAM on FPGA," *Electronics*, vol. 7, no. 9, p. 186, Sep. 2018, doi: 10.3390/electronics7090186.

[18] W. Jiang, "Scalable ternary content addressable memory implementation using FPGAs," in *Proc. Architectures Netw. Commun. Syst.*, Oct. 2013, pp. 71–82.

[19] P. Reviriego, A. Ullah, and S. Pontarelli, "PR-TCAM: Efficient TCAM emulation on xilinx FPGAs using partial reconfiguration," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1952–1956, Aug. 2019.

[20] A. Ullah, A. Zahir, N. A. Khan, W. Ahmad, A. Ramos, and P. Reviriego, "BPR-TCAM—Block and partial reconfiguration based TCAM on Xilinx FPGAs," *Electronics*, vol. 9, no. 2, p. 353, 2020.

[21] I. Ullah, Z. Ullah, U. Afzaal, and J.-A. Lee, "DURE: An energy- and resource-efficient TCAM architecture for FPGAs with dynamic updates," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 6, pp. 1298–1307, Jun. 2019.

[22] M. Irfan, Z. Ullah, and R. C. C. Cheung, "D-TCAM: A high-performance distributed RAM based TCAM architecture on FPGAs," *IEEE Access*, vol. 7, pp. 96060–96069, 2019.

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

[24] K. Locke, *Parameterizable Content-Addressable Memory*, Xilinx, San Jose, CA, USA, 2011.

[25] Xilinx. *Ternary Content Addressable Memory (TCAM) Search IP for SDNet*. SmartCORE IP Product Guide; PG190 (v1.0); Xilix, San Jose, CA, USA, 2017.

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

[27] M. Irfan, Z. Ullah, M. H. Chowdhury, and R. C. C. Cheung, "RPE-TCAM: Reconfigurable power-efficient ternary content-addressable memory on FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 8, pp. 1925–1929, Aug. 2020.

[28] *FracTCAM Source Code*. Accessed: Sep. 29, 2020. [Online]. Available: https://github.com/alixahir/FRACTCAM_SOURCECODE.git