Indonesian Journal of Electrical Engineering and Computer Science
Vol. 22, No. 1, April 2021, pp. 104~112
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v22.i1.pp104-112

Efficient TCAM design based on dual port SRAM on FPGA

Triet Nguyen, Kiet Ngo, Nguyen Trinh, Bao Bui, Linh Tran, Hoang Trang
Department of Electronics, Ho Chi Minh City University of Technology, VNU-HCM, Vietnam

Article Info

Article history:

Received Oct 20, 2020
Revised Jan 6, 2021
Accepted Jan 17, 2021

Keywords:

Algorithmic TCAM
FPGA TCAM
On-chip memory TCAM
RAM-based TCAM
Search engine

ABSTRACT

Ternary content addressable memory (TCAM) is a memory that allows high speed searching for data. Not only it is acknowledged as associative memory/storage but also TCAM can compare input searching content (key) against a collection of accumulated data and return the matching address which compatible with this input search data. SRAM-based TCAM utilizes and allocates blocks RAM to perform application of TCAM on FPGA hardware. This paper presents a design of 480×104 bit SRAM-based TCAM on altera cyclone IV FPGA. Our design achieved lookup rate over 150 millions input search data and update speed at 75 million rules per second. The architecture is configurable, allowing various performance trade-offs to be exploited for different ruleset characteristics.

Corresponding Author:

Hoang Trang
Department of Electronics
Ho Chi Minh City University of Technology, VNU-HCM
268 Ly Thuong Kiet Street, 10 District, Ho Chi Minh City, Vietnam
Email: hoangtrang@hcmut.edu.vn

1. INTRODUCTION

Ternary content-addressable memory (TCAM) is a memory type that can output the appropriate address containing matching data with the input key data. It will compare the search key with the entire stored TCAM words in parallel along with outputing the address of the matching word in one machine cycle [1, 2]. Instead of only storing binary data, TCAM can decode and store state X (don't care). Therefore, the TCAM performs partial matching, which is enabled by the wild-card state "x", may lead to a match with multiple words. TCAM top architecture is comprised of a SRAM cells' array and a priority encoder. With priority encoding, output of TCAM is the highest prioritized result. TCAM is widely uses in networking routers such as translations-look-aside buffers (TLBs) [3] caches in microprocessors, database accelerators in big data analytics [4] and in pattern recognition [5].

The flexibility of software and the near-ASIC performance associated by field programmable gate array (FPGA) known as reconfigurable hardware is undeniable. Ultramodern FPGA devices such as cyclone IV has the number of effective advantages such as high clock rate, low power dissipation, rich on-chip resources and large amounts of embedded with configurable word width. The reason for the demand for TCAM to be simple to integrate, there has been a growing importance in employing FPGA devices to implement TCAM-equivalent search engines.

The prime contributions in this paper is declared in the followings:

a) A detailed introduction to the RAM-based TCAM is given. Key ideas and algorithms behind it are also formalized. The RAM-based TCAM theoretical performance and the vital objections in implementing a large RAM-based TCAM are comprehensively analyzed.

b) A modular and scalable architecture comprised of small-size RAMs components, will be introduced. Additionally, update engine will be introduced in the detail diagram with further investigation and comprehensive experiments.

The remainder of this research paper is organized as follows. Section 2 gives a precise opening of the theoretic aspects of the RAM-based TCAM. Section 3 is futher investigation about the hardware architectures for RAM-based TCAM. Section 4 is a redicussion of the result achieved on FPGA-based TCAM designs. Section 5 states the conclusion of the research.

2. THE PROPOSED METHOD

2.1. List of notations used

The notations used in this paper are listed in Table 1.

Table 1. List of notations used

Notation	Description
N	Depth of TCAM (number of words)
W	Width of TCAM (number of bits of TCAM word)
S	Size of TCAM (equal N * W)
D	Number of register of each RAM (equal 2*W)
K	Input key
Rule	Address of TCAM table
T	Ternary word (value of Rule)

2.2. Basic ideas

In *N*W* TCAM, a *W* bit binary input key is looked up and mapped exactly into *N* bit binary match vector. When using a 2^wN RAMs, it attains the similar function with the native TCAM. Each RAM stores TCAM word and utilizes the value of input key to search the address in RAM.

Algorithms that use RAM to implement TCAM:

a) Principle 1: Write to RAM
RAM [address][Rule] = 1 if *address = T*
RAM [address][Rule] = 0 if *address* $\neq$ *T*
With – address: address in RAM

b) Principle 2: Read from RAM
Output = RAM [address] if *address = K*
With – address: address in RAM

EXAMPLE: Assume that we use RAM to find KEY = 1100. The size of RAM needed to decode depends on the number of rules and the number of bits of each rule. With *W* = 4 and *Rule* = 4, we need $2^W = 24 = 16$ registers and number of bits of each register = number of rules = *4*. Thus, decoding the Basic Rule table in Table 2 we need a RAM of size 2^4x4 = 16x4 (16 registers with each register having a width of 4 bits). When applying principle 1 for writing to RAM, we have the results shown in Table 3.

Table 2. Basic Rule

Rule	Word
0	1001
1	x100
2	01xx
3	xx00

Table 3. Content of full-fill RAM

Address	R0	R1	R2	R3
0000[0]	0	0	0	1
0001[1]	0	0	0	0
0010[2]	0	0	0	0
0011[3]	0	0	0	0
0100[4]	0	1	1	1
0101[5]	0	0	1	0
0110[6]	0	0	1	0
0111[7]	0	0	1	0
1000[8]	0	0	0	1
1001[9]	1	0	0	0
1010[10]	0	0	0	0
1011[11]	0	0	0	0
1100[12]	0	1	0	1
1101[13]	0	0	0	0
1110[14]	0	0	0	0
1111[15]	0	0	0	0

Explaination:

a) With rule 0, there is *word = T0 = 1001* → According to the rule of Write we have*: RAM [1001][0] = 1,* RAM *locations [address ≠T0][0] = 0.*
b) With rule 1, there is *word = x100* → *word = 0100, 1100* → According to principle 1, we have : *RAM[0100][1] = 1* and *RAM[1100][1] = 1,* RAM location [*address ≠ T1][1] = 0.*
c) With rules 2 and 3, we fill in the same way as rules 0 and 1.

According to Principle 2, the output result will be retrieved when we get the address = key. For the example above, we have Output = RAM[1100] = 0101 →We have the key results that match rule 1 and rule 3. Depending on the encoder priority setting, we will choose one of the two results as the final result of TCAM searching, which is shown in Table 4.

Table 4. Result of TCAM searching

Address	R0	R1	R2	R3
0000[0]	0	0	0	1
0001[1]	0	0	0	0
0010[2]	0	0	0	0
0011[3]	0	0	0	0
0100[4]	0	1	1	1
0101[5]	0	0	1	0
0110[6]	0	0	1	0
0111[7]	0	0	1	0
1000[8]	0	0	0	1
1001[9]	1	0	0	0
1010[10]	0	0	0	0
1011[11]	0	0	0	0
1100[12]	0	1	0	1
1101[13]	0	0	0	0
1110[14]	0	0	0	0
1111[15]	0	0	0	0

Comments: The use of 1 RAM gives us advantages and disadvantages such as

a) Advantage: Simple and easy to use.
b) Disadvantages: extremely large memory usage for bigdata data types. For example with word = 100 bits we will have a RAM that has 2^{100} registers → impossible to solve.

Summary, it is necessary to split the number of bits of the word to process each part to minimize memory for each RAM.

2.3. Main ideas

There are many research on optimizing algorithmic TCAM on FPGA. The solution provided in [6, 7] gives good mapping technique to optimize storage space but trade-off with throughput speed. In [8] use multi-pumping, which is overclocking the memory unit to reach desired number of access. In practice, the method [8] is infeasible with high speed system. The pipelined-lookup method in [2, 9] is efficient and could be used to speed up grid of BRAMs. We decided not to use decision tree and binary tree method in [10-14] as this increases complexity and require more software solution. Other type of solution involves splitting and cutting the original rule set into many subsets [15, 16] which faces the same complexity as tree based method. We aim to use fully hardware for searching and updating without difficult computation, such as hashing [17]. Openflow 5 tuples key length [18] is considered to be used in our hardware implementation. FPGA-based TCAM on researches by Ullah *et al.* [19-23] with many different examples helped us decided on memory efficiency, energy consumption and speed. Using pipelined RAM blocks [24] is possible to achieve very high throughput. To avoid using large RAM memory, we have to divide Word into separate pieces to process. Several small RAM memory are used instead of one huge RAM memory. Depth division method.

Depth division [25] is a technique whiche chain pieces match vectors collected from several small TCAMs. One TCAM which has N depth can be splited into two TCAMs. This idea comes from the assumption that if N Depth is too huge and no single RAM or TCAMs can process, it need to utilize numerous “shallower” RAM or TCAM to solve this issue. Therefore, this method is:

$$N * WTCAM = (N1+N2) * WTCAM = N1 * WTCAM_1 + N2 * WTCAM_2 = N1 * 2^w RAM_1 + N2 * 2^w RAM_2$$

After dividing depth of TCAM, it need an action to concatenate the output for linking the complete N-bit match vector. With depth division method, we do not only optimize the memory of each single RAM but also utilize appropriately several RAMs in FPGA devices. Width extension method:

Using the similar idea with depth division method, width division [25] is a technique which chain pieces W-bit word vectors collected from several small TCAMs. One TCAM which has W width can be splited into two TCAMs. This method solved the problem that if W width is too huge and no single RAM or TCAMs can process, it need to utilize numerous "narrower" RAM or TCAM to encounter this issue.

Therefore, this method is:

$$N*WTCAM = N*(W1+W2)TCAM = N*W1TCAM_1+N*W2TCAM_2 = N*2^{w1}RAM_1+ N * 2^{w2}RAM_2$$

After dividing width of TCAM, it need an action to connect all the output with the AND-gate for find the final *W-bit* match vector. With Width Division Method, we not only optimize the memory of each single RAM but also utilize appropriately several RAMs in FPGA devices.

We reuse instance in Basic idea to prove this idea. Suppose we use RAM to decode a rule table with the following content in Table 5 with a key value 1100. With using several RAMs, we break down the word of the rule and the Key into 2 parts, 2 bits each for processing. With Principle 1, we in turn fill the first 2 bits and the following 2 bits of the rules into 2 Tables as follows in Tables 6 and 7. With Principle 2, we will read 2 registers *Output1 = RAM[Key[3 : 2]] = 11* and *Output0 = RAM[Key[1 : 0]] = 00* from the tables. After reading 2 outputs from the 2 RAM above, we will perform AND operations of the outputs to get the final result: *OUTPUT = Output1&Output2 = 0101&0111 = 0101*. The result in two separate *AM(Word[1:0])* is shown in Table 8.

Table 5. Content of Full-fill two separate RAM (Word[3:2])

Address	R0	R1	R2	R3
00	0	0	0	1
01	0	1	1	1
10	1	0	0	1
11	0	1	0	1

Table 6. Content of Full-fill two separate RAM (Word[1:0])

Address	R0	R1	R2	R3
00	0	1	1	1
01	1	0	1	0
10	0	0	1	0
11	0	0	1	0

Table 7. Result of TCAM in two separate RAM (Word[3:2])

Address	R0	R1	R2	R3
00	0	0	0	1
01	0	1	1	1
10	1	0	0	1
11	**0**	**1**	**0**	**1**

Table 8. Result of TCAM in two separate RAM (Word [1:0])

Address	R0	R1	R2	R3
00	**0**	**1**	**1**	**1**
01	1	0	1	0
10	0	0	1	0
11	0	0	1	0

Comments: the positive and the negatvive effect of using several RAMs

a) Positive aspect: use small amount of RAM to solve the decoding
b) Negative effect: need more RAM to use

Using several small RAMs memory is more effective and feasible than using only large RAM.

3. RESEARCH METHOD

3.1. TCAM without update logic

When the search engine starts to work, all the data in memory is cleared serially by each address. The machine states at ready stage and prepares to write data. Firstly, it loads the initial address (x0) and brings it to compare with WORD which is processed include don't care bits. If the data is match, write enable pin's of ram will be kicked off. In while, read data at output of RAM is added with rule which is decoded from original data. These data will be stored in RAM if write enable bit is turned on. The process has been repeated until it loads the last address of the memory. After that, the engine will load another rule and implement like the previous process. This TCAM structure without update logic is shown in Figure 1. To read the look up data, write data process need to be finished and the machine backs to the ready stage. Look up data is selected to be read by mux instead of the result from control address as presented in Figure 2.

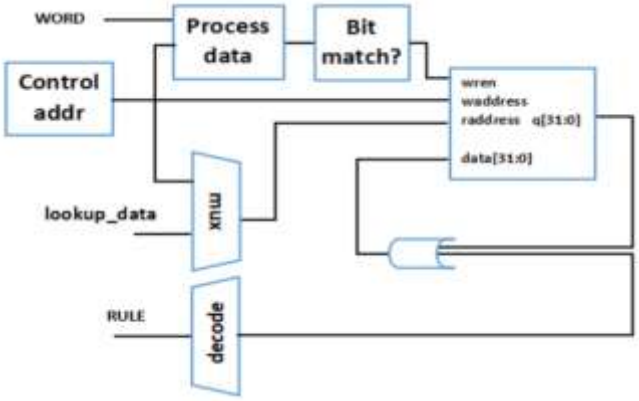

Figure 1. Hardware Architecture of TCAM without update logic

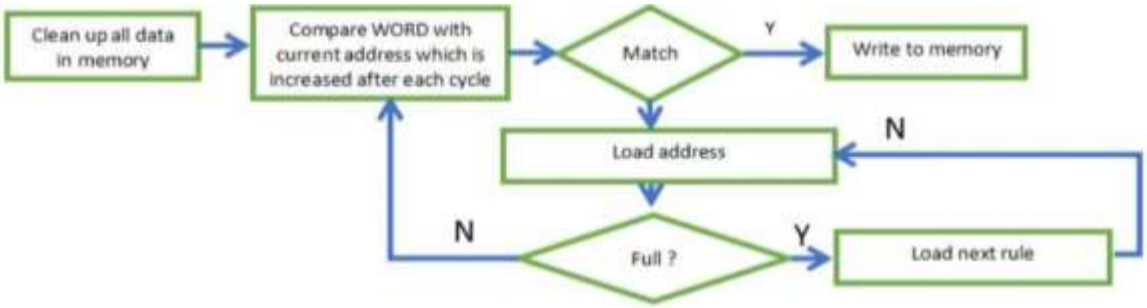

Figure 2. TCAM without update logic component flow

3.2. TCAM with update logic

The search engine without update logic consumes the amount of time for writing full data into the memory. Thus, if the user wants to get some new rules to their system, they need to load full data again. Therefore, the engine needs to have some update logic components to solve these disadvantages. The TCAM structure with additional update logic component is shown in Figure 3.

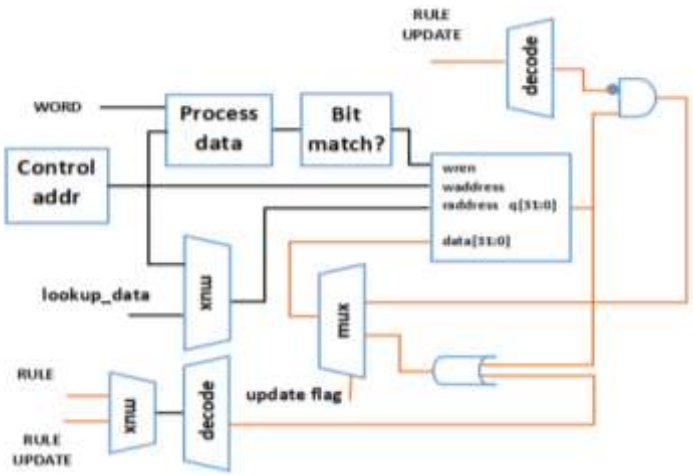

Figure 3. Hardware Architecture of TCAM with update logic

When the writing process finished, the machine is in ready stage and waiting for the update signal. Initially, rule update is loaded and decoded instead of original rule. These inputs are used for deleting and overwriting all the data in columns rule. The updated process consumes the same amount of time writing one rule to the table. Its process is shown in Figure 4.

Figure 4. TCAM with update logic component flow

3.3. A 480x104-bit Implementation

The top-level architecture of our 480x104-bit RAM-based TCAM consist of 195 blocks RAM M9K [26], which aligned 15 columns each of which contains 13 units, shown in Figure 5. According to two Principles and two division method, rule table divide sequentially rules into each blocks in order from left to right. The first rule is the highest priority and the last rule is the lowest priority. The matching result of each RAM which read by *address = (W bitinputkey)/13* is connected with AND-gate to filter the final result of each column, so entire output of AND-gates are inputs of module priority encoder. It chooses the final result and show the rule match with *W-bit* input key.

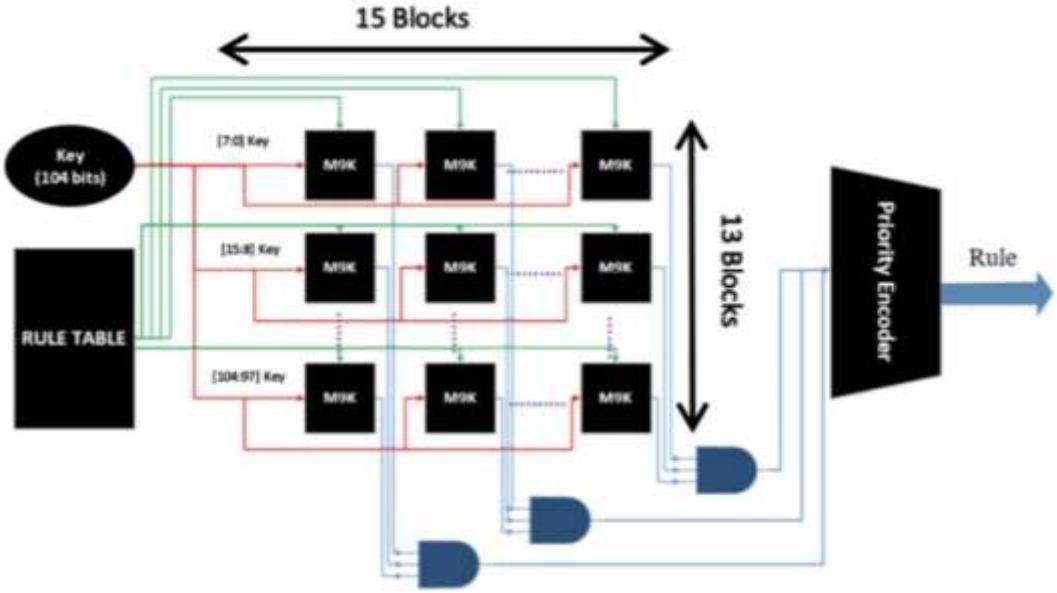

Figure 5. TCAM Top-level architecture

4. RESULT AND DISCUSSION

4.1. Synthesist result of TCAM on FPGA Intel cyclone IV

We synthesize the proposed design on FPGA chip EP4CE115F29C7 intel cyclone IV and the result is presented in Figure 6. The design uses 40% of the available memory on the chip (195 M9Ks). The registers and logic elements usage is minimal. The timing synthesis results at 150 MHz clock speed. The throughput of the algorithmic TCAM would be at 150 million packets per second. The speed of updating the rule set is at 75 million updates per second for each operation.

Figure 6. TCAM resource synthesized on EP4CE115F29C7

4.2. Update content and expected result

After writing 480 rules into RAM, we update some rules to test the update function. The steps is shown in Table 9. The result, which is as expected, is presented in Table 10.

Table 9. Rule changes in update process

Rule Update Change rule 10 → 124
Change rule 15 → 0
Delete rule 31
Delete rule 480
Change rule 99 → new rule
Delete rule 373
Delete rule 40

Table 10. Rule match result after updating

No.	Key (104 bits) (Hex)	Rule Match Before Update	Rule Match After Updates
0	39 e6 8b 07 39 27 bd 44 9c 9a 3a 2a 2a	2	2
1	b4 fd 2c 9d d8 67 0f bb 18 7b c3 55 d4	15	NOT MATCH
2	74 11 b8 ba 90 8c 83 79 d0 7e 9b 17 71	40	400
3	41 b3 a1 91 cb f2 c5 49 de 7d 89 37 28	88	88
4	40 44 1f 6c 55 f2 a3 80 d9 b9 e6 e4 f8	102	102
5	00 31 68 ed 95 31 75 32 e7 83 93 32 d0	124	10
6	e8 d1 1f cb b4 15 f7 80 d8 86 3d 1c 1a	175	175
7	44 94 2e 89 42 ae 53 88 6e 51 f7 6d f8	210	210
8	98 70 01 d1 7c 07 41 21 e1 38 96 c8 35	222	222
9	a7 aa 8f 7f 0e aa 9f 56 09 cc aa 03 dd	252	252
10	f0 cd e0 c1 16 00 aa 10 5f 3d 7e ba 33	NOT MATCH	NOT MATCH
11	66 76 59 16 32 61 57 d1 30 59 12 34 56	306	306
12	97 a7 eb d7 98 76 65 54 32 10 64 8d 81	373	NOT MATCH
13	ac 89 bd 8f ef 67 ee 6f fa 74 af 00 28	436	436
14	de 3c f6 39 03 13 57 68 97 f2 50 87 c9	478	478
15	7b 3b ed 5d 3e 30 d1 43 B0 7f ec c6 14	NOT MATCH	99

4.3. Modelsim result and throughput discussion

With Modelsim 10.5b, we simulate with clock period = 200ps (f = 5GHz). From t = 0ps to t = 24, 819, 500ps is time to write 480 rules into RAM. At t = 24, 819, 500ps, we use 16 keys to search and the

result is displayed in Figure 7. After that we continue to update some rule like 4.2 and at t = 25, 287, 900ps, the real result after update is the same with the expected result, we show this in Figure 8.

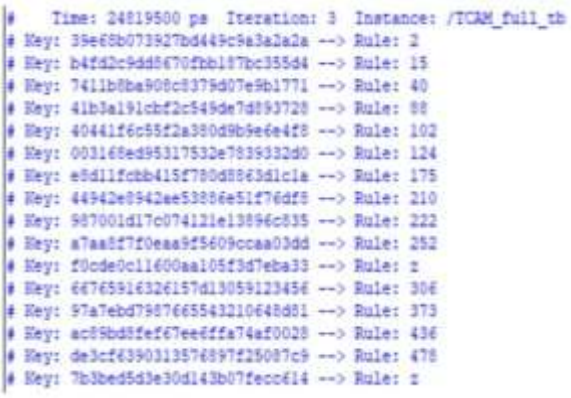

Figure 7. Search result before update process

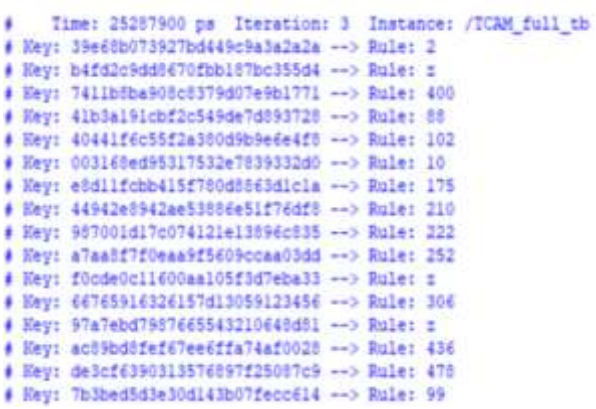

Figure 8. Searching result after update process

4.4. Future scope

The design clock speed could be improved by pipelining effectively between RAM blocks. In application, the ruleset could also be optimized on software for easy updating and data searching. The update function could be improved in speed by allowing arbitrary location updating. We believe this would relies on the characteristic of the ruleset and is difficult to achieve generally. One important future direction of build TCAM on FPGA is to implement them on external DDRAM. This would allow the structure to cost less on-chip resources of the FPGA and offload the ruleset table to external memory which is less limited by size.

5. CONCLUSION

This paper presents a TCAM design on FPGA, utilizing FPGAs advantages in reconfiguration and flexibility effectively. Moreover, the TCAM can be scaled easily to fit with a design specification. However, the memory resource is a challenge for large searching engine integration. We expected the design provides a general perspective in RAM-based TCAM structure on FPGA.

ACKNOWLEDGEMENTS

This research is funded by Ministry of Science and Technology under grant number KC.01.24/16-20.

REFERENCES

[1] Wade, Jon P. and Charles G. Sodini, "A ternary content addressable search engine," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 4, pp. 1003-1013, 1989, doi: 10.1109/4.34085.
[2] Meiners, Chad R., Alex X. Liu and Eric Torng, "Algorithmic approaches to redesigning tcam-based systems," *Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, 2008, doi: 10.1145/1375457.1375524.
[3] S. Mittal, "A survey of techniques for architecting TLBs," *Concurrency Comput., Pract. Exper*., vol. 29, no. 10, pp. e4061, 2017, doi: 10.1002/cpe.4061.
[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, doi: 10.1109/ACCESS.2018.2816039.
[5] M. Imani, A. Rahimi and T. S. Rosing, "Resistive configurable associa- tive memory for approximate computing," in *Proc. Design, Automat. Test Europe Conf. Exhib. (DATE)*, pp. 1327-1332, 2016.
[6] 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, doi: 10.1109/TVLSI.2016.2636294.
[7] 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, doi: 10.1109/TENCON.2015.7372837.
[8] 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, doi: 10.1109/ACCESS.2018.2822311.

[9] Pao, Derek and Ziyan Lu, ”A multi-pipeline architecture for high-speed packet classification,” *Computer communications*, vol. 54, pp. 84-96, 2014, doi: 10.1016/j.comcom.2014.08.004.
[10] Li, Xianfeng and Yuanxin Lin,” TaPaC:ATCAM-assisted algorithmic packet classification with bounded worst-case performance,” *2016 IEEE Global Communications Conference (GLOBECOM). IEEE*, 2016, doi: 10.1109/GLOCOM.2016.7842313.
[11] Vamanan, Balajee and T. N. Vijaykumar, ”TreeCAM: decoupling updates and lookups in packet classifi-cation,” *Proceedings of the Seventh Conference on emerging Networking EXperiments and Technologies*, 2011.
[12] Hatami, Rashid and Hossein Bahramgiri, ”Fast SDN updates using tree-based architecture,” *International Journal of Communication Networks and Distributed Systems*, vol. 25, no. 3, pp. 333-346, 2020, doi: 10.1504/IJCNDS.2020.109566.
[13] He, Peng, *et al*., ”Toward predictable performance in decision tree based packet classification algorithms,” *2013 19th IEEE Workshop on Local & Metropolitan Area Networks (LANMAN). IEEE*, 2013, doi: 10.1109/LANMAN.2013.6528270.
[14] Chang, Yeim-Kuan and Chao-Yen Chien, ”Layer partitioned search tree for packet classification,” *2012 IEEE 26th International Conference on Advanced Information Networking and Applications. IEEE*, 2012, doi: 10.1109/AINA.2012.122.
[15] Li, Wenjun, *et al*., ”Cutsplit: A decision-tree combining cutting and splitting for scalable packet classification,” *IEEE INFOCOM 2018-IEEE Conference on Computer Communications. IEEE*, 2018, doi: 10.1109/INFOCOM.2018.8485947.
[16] Pnevmatikou, Arsinoe, *et al*., ”Fast Packet Classification using RISC-V and HyperSplit Acceleration on FPGA,” *2020 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE*, 2020, doi: 10.1109/ISCAS45731.2020.9180588.
[17] Reviriego, Pedro, *et al*., ”Multiple Hash Matching Units (MHMU): An Algorithmic Ternary Content Ad- dressable Memory Design for Field Programmable Gate Arrays,” *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR). IEEE*, 2018, doi: 10.1109/HPSR.2018.8850764.
[18] Yang, Tong, *et al*., ”Fast open flow table lookup with fast update,” *IEEEINFOCOM2018-IEEE Conference on Computer Communications. IEEE*, 2018, doi: 10.1109/INFOCOM.2018.8485878.
[19] Ullah, Zahid, Manish K. Jaiswal and Ray CC Cheung, ”Z-TCAM: an SRAM-based architecture for TCAM,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 2, pp. 402-406, 2014, doi: 10.1109/TVLSI.2014.2309350.
[20] Ullah, Zahid, ”LH-CAM: Logic-based higher performance binary CAM architecture on FPGA,” *IEEE Embedded Systems Letters*, vol. 9, no. 2, pp. 29-32, 2017, doi: 10.1109/LES.2017.2664378.
[21] Ullah, Zahid, Manish K. Jaiswal and Ray CC Cheung, ”Design space explorations of Hybrid-Partitioned TCAM (HP-TCAM),” *2013 23rd International Conference on Field programmable Logic and Applications. IEEE*, 2013, doi: 10.1109/FPL.2013.6645583.
[22] Ullah, Inayat, *et al*., ”DURE: An energy-and resource-efficient TCAM architecture for FPGAs with dynamic updates,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1298-1307, 2019, doi: 10.1109/TVLSI.2019.2904105.
[23] Mahmood, Hassan, *et al*., ”Beyond the limits of typical strategies: Resources efficient FPGA-based TCAM,” *IEEE Embedded Systems Letters*, vol. 11, no. 3, pp. 89-92, 2018, doi: 10.1109/LES.2018.2888889.
[24] Jiang, Weirong, Qingbo Wang and Viktor K. Prasanna, ”Beyond TCAMs: An SRAM-based parallel multi-pipeline architecture for terabit IP lookup,” *IEEE INFOCOM 2008-The 27th Conference on Computer Communications. IEEE*, 2008, doi: 10.1109/INFOCOM.2008.241.
[25] Weirong Jiang, ”Scalable Ternary Content Addressable Memory Implementation Using FPGAs,” *Architectures for Networking and Communications Systems, San Jose, CA*, pp. 71-82, 2013, doi: 10.1109/ANCS.2013.6665177.
[26] Intel Company, RAM/ROM ’s user guide, June 2014.