Indonesian Journal of Electrical Engineering and Computer Science
Vol. 22, No. 1, April 2021, pp. 89~96
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v22.i1.pp89-96

Algorithmic TCAM on FPGA with data collision approach

Nguyen Trinh, Anh Le Thi Kim, Hung Nguyen, Linh Tran
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

Content addressable memory (CAM) and ternary content addressable memory (TCAM) are specialized high-speed memories for data searching. CAM and TCAM have many applications in network routing, packet forwarding and Internet data centers. These types of memories have drawbacks on power dissipation and area. As field-programmable gate array (FPGA) is recently being used for network acceleration applications, the demand to integrate TCAM and CAM on FPGA is increasing. Because most FPGAs do not support native TCAM and CAM hardware, methods of implementing algorithmic TCAM using FPGA resources have been proposed through recent years. Algorithmic TCAM on FPGA have the advantages of FPGAs low power consumption and high intergration scalability. This paper proposes a scaleable algorithmic TCAM design on FPGA. The design uses memory blocks to negate power dissipation issue and data collision to save area. The paper also presents a design of a 256 x 104-bit algorithmic TCAM on Intel FPGA Cyclone V, evaluates the performance and application ability of the design on large scale and in future developments.

Corresponding Author:

Linh Tran
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: linhtran@hcmut.edu.vn

1. INTRODUCTION

Content addressable memory (CAM) is special type of memory which search the entire database in one clock cycle and output the address information of the input data [1-3]. The applications of CAM are routing Internet package, cache memory for microprocessors, artificial intelligence. Additionally, CAM also has applications in biological research, comparing and matching DNA sequences [4]. Although CAM has many advantages such as better searching speed of addressing data with software on microprocessors, it consumes more energy and has poor integration scalability. The poor integration scalability is caused by the needs of using a separated memory structure and being an independent IC connected peripherally to the microprocessors [5].

Ternary content addressable memory (TCAM) is CAM with support for ternary values. TCAM consumes 30 times more memory resources than DDR SRAM and 150 times more power consumption per memory bit than SRAM [5]. TCAM has always been a vital point for improvement in digital systems. However, the development for TCAMs in recent years seems to have slowed down. Very-high-speed searching applications still requires the refinements of TCAMs in many directions.

In recent hardware acceleration solutions, field-programmable gate arrays (FPGA) are commonly used as it can be designed quickly and updated directly on installed system. FPGAs applications are in accelerating Internet packages routing, artificial intelligence and biological research and DNA sequences [6, 7]. These applications require fast data searching hardware such as TCAM and CAM. Because most FPGAs do not have native support for CAM and TCAM hardware on their resources, methods have been proposed to use existing FPGA on-chip resources such as memory blocks and look-up tables.

Studies proposed using internal RAM blocks of FPGA to replace CAM and TCAM, however, with trade-off for a large amount of resources [5, 8]. In [9] also provide scaleable algorithmic TCAM which uses FPGA slices as its main resource. Xilinx also presents an intellectual property core for implementing TCAMs on their FPGA [10]. Methods to optimize the amount of resources and energy consumption are also proposed [11-17]. GreenTCAM [18] proved that algorithmic TCAM improves power efficiency with reliability on the rule set. Pseudo-TCAM [19] has simple and uniform hardware organization for all rulesets and high throughput, but need special mapping and relocation for updates. Algorithm aims for fast updating mentioned in [20] trade-off with performance. Multi-pumping SRAMs in [15] increases the operating frequency of internal RAM blocks and allows multiple access advantage with the same system clock. With high-frequency designs, it is difficult to have an internal RAM design that runs 5 times faster than the system [15]. Research [17] suggests a special way of positioning data, but only refers to replacing CAM without mentioning how to support ternary value often seen in content addressable memory when used in aforementioned applications. Multiple hash matching unit [21] is also an option to implement algorithmic TCAM, but it meets with hashing collision.

This study proposed an algorithmic TCAM structure using RAM blocks on FPGA, which would give the advantage of power consumption over orginal TCAM cells [14]. To save more area, data collision method for CAMs from [17] are used. In order to support ternary value, we use additional memory blocks structures and transform ternary value into mask vectors. The architecture shown in the paper is an improvement from [17] in the term of ternary matching functions.

The remainder of this paper is structured as follows. The second section shows the algorithms and calculations that organize the database and resources according to mathematical calculations. In the third section, an example of a 256 x 104-bit ternary content addressable memory on FPGA Cyclone V is presented. The forth section estimates, calculates and evaluates the feasibility of large-scale implementation of the proposed algorithm and future development. The final section draws conclusions.

2. THE PROPOSED METHOD

Table 1 shows the basic database of a CAM, including rows of data sorted in order of priority. Rule 0 is the default rule when the retrieved data does not match with any of the remaining rules in the memory. That typical database can be loaded into a content addressable memory that supports ternary values or a content addressable memory that does not support the ternary values with controlled prefix expansion (CPE), simply separating the ternary value into its binary values. Thus, with a content addressable memory that does not support ternary values, the demand for memory resources is extremely large.

Table 1. Example of TCAMs basic database

Rule	TCAM Rules
0	x.x.x.x
1	11.x.x.x
2	11.06.x.x
…	…
n	11.06.19.07

The example of controlled prefix expansion (CPE) is shown in [22]. With a large number of rules and many arbitrary values like a FIB router, using CPE consumes a huge amount of resources. To build a content addressable memory on FPGA, the main resource used in the study is internal RAM on FPGA. The internal RAM memory on the FPGA is provided as memory cells as shown in Figure 1. Each memory cell supports storing a certain amount of kilobit of data. Intel's M20K memory cell supports 20k-bit and Xilinx M9K, M10K or B36K are similar. These memory cells support the number of address bits and memory cell lengths that vary by technology. However, when using to build content addressable memory, it must be noted that these memory cells do not support ternary values but only support binary values.

To understand the algorithm, first we need to understand how the content addressable memory works. The basic operation of a content addressable memory is depicted in Figure 2. The data as a key is input into the content addressable memory and the output is returned. The output is the address of that data in the content addressable memory. In general cases, the result is either the data matches a rule that is in the content addressable mmemory or not, and the action apply to that data. In building TCAM memory on FPGA, as mentioned in [7, 16], the best way to reduce the amount of RAM consumed and optimize the system speed is to cut the key into many pieces with a certain bit length (may or may not be equal with each other). The most optimal method [7, 16] is achieved when the bit length of the key fragments is the minimum address bit length of the memory cell on the FPGA. For example, for Intel's M20K memory cell, this length is 9 bits. In the method implemented in the algorithm of this study, we will also cut the key into pieces to optimize speed and reduce memory consumption.

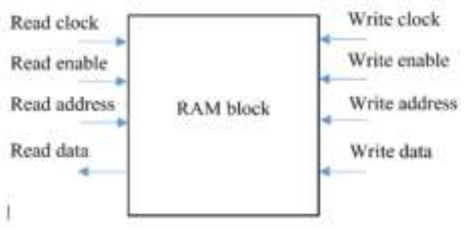


Figure 1.Typical RAM block cell

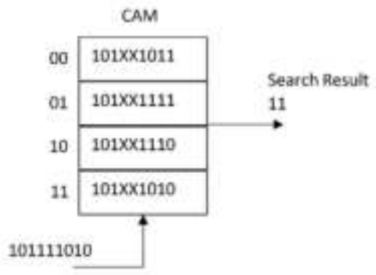


Figure 2. Basic operation of a content addressable memory

Figure 3 shows the key cut into pieces of a certain bit length. These pieces access each predefined cell cell and form a vector of results, as shown in the study [17]. However, research [17] does not support rules that contain ternary values. Once the vector of the result was obtained, they only analyze and verify it to produce the result of the matching key. The research presented in this paper modifies and reorganizes the data collision structure which was presented in research [17]. The modifications and additional modules allow rules to use ternary values and reduce the burden of CPE on memory consumption. Figure 4 shows the basic dataflow of this TCAM structure. The data follows 3 main steps to achieve result.

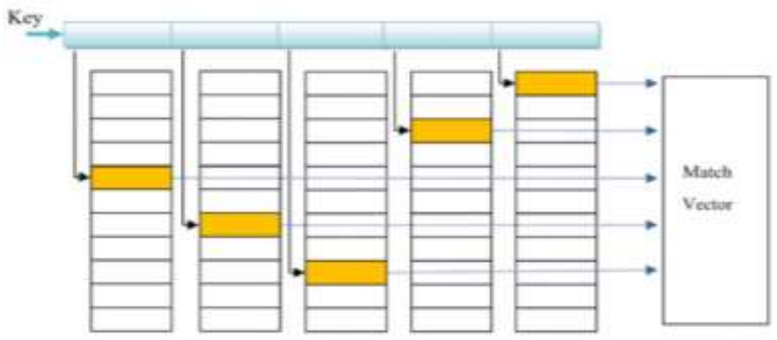


Figure 3. The CAM Key is cut into pieces and used to create a match vector at the end

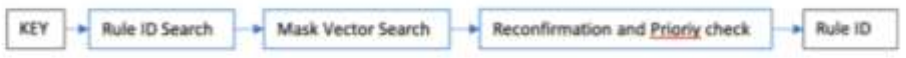


Figure 4. Basic dataflow of structure

Figure 5 illustates the process of setting rules into the memory. Here we take an example of 10-bit rules, divided into 5 2-bit pieces. A table of this content addressable memory consists of cells containing two parts, the status of the cell and the identification number (ID number or ID) for the rule. The way to write the rules into the table is as follows. First, the rules must be in proper form to be added to the table. The appropriate form to be included in the table is the rules with fragments containing two arbitrary value bits that go together as "**" or have no arbitrary values. If the rule has fragments in the form of only one arbitrary value, the rule must be separated into two rules. To put a rule into a table, write the rule's ID into the cell, respectively, with the address corresponding to the value of that rule on the table and transfer the state of the cell to written. If the cell already has a rule written in first, the cell will be in collision state. After finishing writing the rules, each cell in the table has one in three states: blank, filled, or collision. When writing a rule, it is necessary to ensure that there are at least n cells that do not collide, usually, n only needs to be 1. To remove a rule from the table, remove that rule from the cell it has filled in, the collision cell will return to be a normal filled cell if there is only one fill rule left, the filled cell will return to be a blank cell.

To find the address for a key, few more components are needed. However, at this step, a data vector can already be found to move on. Figure 6 illustrates how to find a key from the written rule data table and in some invalid cases, quickly conclude the result. Bad cases are predictable during table construction and can be prevented with proper software optimization. According to [16], for a table with n addresses and containing m random rules, the collision rate for a column is 26%. For z fragments, the rate of having an entire z-piece vector is all in collided state is $26\%^z$, this is very small with a long bit-length key.

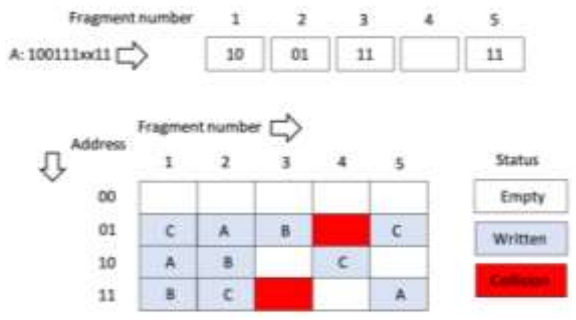


Figure 5. Setting rules into the database of the TCAM

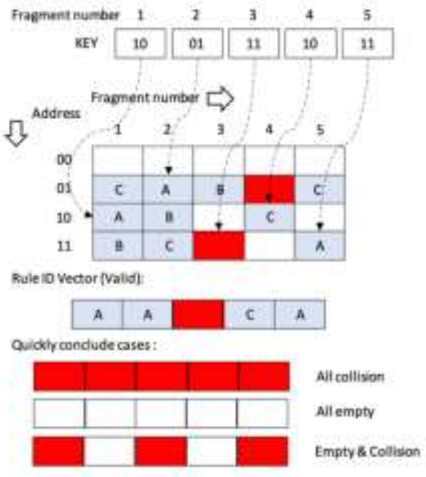


Figure 6. Searching for the match vector from the database

In the next step, the rules have been written in the table. For each of these rules, we have a vector that marks the corresponding ternary values (mask vector). Each bit in this vector marks a fragment of arbitrary value ”**”. For each piece that contains a written ID (not in an collision state), we need to find a corresponding mask vector to compare. There are many ways that can be applied to search for custom vectors in this case, each with different advantages and disadvantages when applying on hardware. The mask vectors and the marking process are described in Figure 7. The mask vectors are presented in the figure as 5-bit bit strings with each asserted bit represent a masked segment.

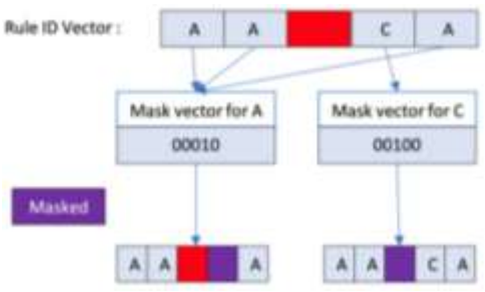


Figure 7. Mask vectors from the Rule ID Vector

The following step is to compare the segments in the masked rule ID vector and give the final result including whether the key matches the rule database and if so, the ID of the rule. In the case there is a match, it still has to go through another memory to confirm again as [16], because of the false positive rate. The reconfirmation is done before comparing the priority order because if compare the priority first, there will be cases of false positive eliminating true positives. The step is shown in Figure 8.

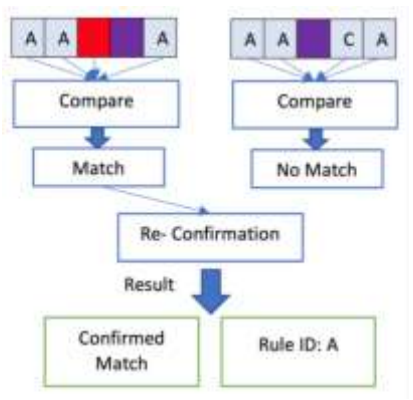


Figure 8. Re-confirmation of the vector to get the correct result

The priority is included in the reconfirmation table and with rules sets in large content addressable memory can be hierarchically preprioritized between regions to reduce the pressure of comparing priorities. After comparing the priority and getting the highest-priority match, the result is the ID of the rule that matched the key or show that the key does not match. This ID is used by an external data memory to guide further action for the packet that ties with the key.

3. NUMERICAL EXPERIMENTATION

The algorithm is built on FPGA Cyclone V DE-10 chip, using FPGA chip 5CSXFC6D6F31C6. The main memory resource of the chip is M10K memory unit. The minimum length of address bit the M10K memory supports is 8 bits (256 x 40) for the Simple Dual Port mode for independent write and read control. Using this mode eases the writing process, allows updating the database into memory without affecting the reading process to search and retrieve data [23].

The memory access memory size selected for simulation and testing on Kit is 256 x 104-bit. The minimum bit length of the address is chosen. The chosen key length is 104 bits, 8 bit fragments to match the address bit length of the address. The 104-bit length is suitable for practical applications used in OpenFlow's 5-tuple network routers consisting of: source IP address (32-bit), destination IP address (32-bit), source port (16 -bit), destination Port (16-bit), protocol (8-bit) [24]. With the 8-bit fragmentation option, the total number of fragments in the vector is 13 pieces.

For calculation on memory resources, 256 rules and assuming with proper arrangement, no rule with all the piece of that rule in a collision state requires 13 M10K memory units. Each unit contains the information of 1 piece. The pieces information consists of 8 bits to contain the rule ID and 2 bits to indicate the status of the memory cell, so 10 bits in total. With 40 data bit-length of M10K memory, it is possible to hold an additional 3 vector fragments without consuming any additional resources, supporting 1024 rules. But to simplify the simulation process, the test only uses the first 10 bits of memory data. Thus, saving the fragments of the rule ID vector consumes 13 M10K. With 13 fragments, for direct searching implementation, 13 M10K memories are used to store mask vectors of rules. Although there are only 256 rules, but for direct searching, to search for mask vectors in parallel with ID vector, it is required to be perform on 13 pieces at the same time, each memory contains 256 x 13-bit of mask vector data of the rules. The resources for simple and direct confirmation memory are 13 256 x 125-bit of memory. Each 125-bit strings consists of 104 rule data bits, 13 mask bits and 8 priority bits. In this step, 52 M10K memory units for confirmation and priority comparison.

Simulation and synthesis results on the FPGA kit showed that the amount of memory resources consumed is 78 M10K. Simulate the result on the testbench shows acceptable latency. The design was relatively optimized in terms of clock speed, resulting in 200 MHz on the FPGA 5CSXFC6D6F31C6 chip. The result confirmed that the structure works as intended in correspondence with our theory. Overall block diagram of the design as shown in Figure 9.

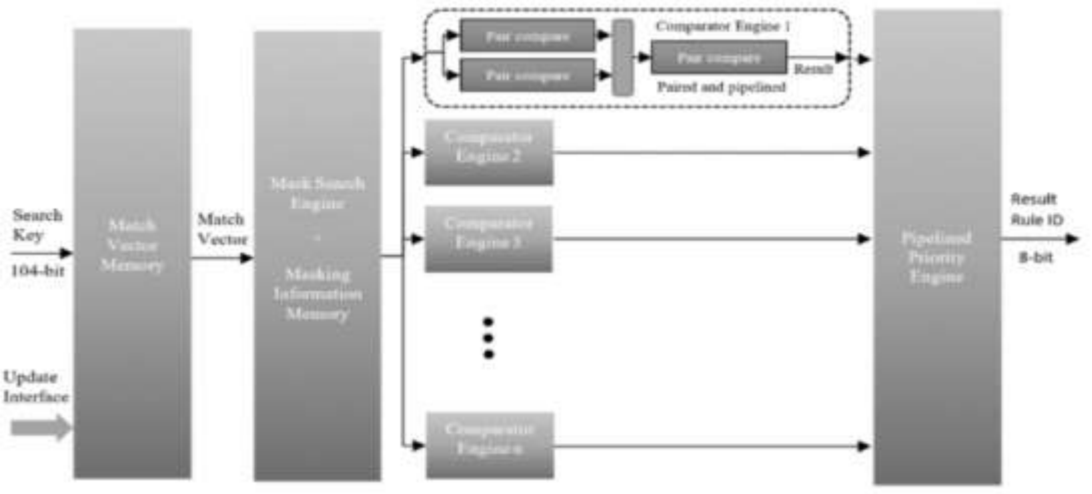


Figure 9. Overall block diagram of the design

4. RESULT AND DISCUSSION

The algorithm is built on the purpose of improvement [17] and provides the future development directions of the application capabilities based on the need to implement data access memory to support custom values with a larger scale. Because the FPGA’s internal memory is used and accessed many times in the process of searching, trade-off with latency, the power consumption of this architecture is lower than that of traditional TCAM [17]. In addition, the entire use of static random-access memory (SRAM) also gives the maximum clock speed of the structure higher than traditional content addressable memory.

Compared to other algorithmic TCAM structures, on a large scale and on industry-used FPGAs, the algorithm achieves better performance optimization. On high-speed FPGAs such as Intel Arria and Stratix series, the M20K memory has a minimum of 9-bit address bit length, in [5] has difficulties with large internal memory consumption and because it needs to access a lot of memory in parallel, and every time a new rule is written in, another 1-bit column in memory is used. It is difficult to optimize memory usage.

For a 1024 x 104-bit content addressable memory, building on [5] uses an average of 302 M20K. The proposed structure does not consume more M20K unit than in the example (using only 8 bits of 9 bit and use the remaining 1 bit to write-copy, allowing to change the entire database of memory access without affecting the retrieval process) for rule ID vector search. The memory section contains mask vectors and the reconfirmation memory consumes 4 times more resources, for a total of 273 M20K units. The difference in resource usage will be greater with logical rules arrangement, optimizing the ability to overlap rules on software. In addition, memory containing ternary vectors and recertified memory can be built in other ways of direct search and will save much more resources, with popular search methods instead of direct search, such as hash tables, binary trees, and bit vectors.

With using the SRAM True Dual Port mode, we have two memory access as read or write per cycle. The throughput is double for searching purpose. The consumed resources would stay the same due to data words width is halved compared to SDP. Updating the table would slow the throughput of the algorithmic content addressable memory during writing cycles which could be fix with an update-copy method but it would also require additional resources.

Fragments bit-length could also be vary based on the applications. In routers, parsing IP in the range of /16 to /24 subnet mask would make the default 8-bit fragment becoming heavy on the expansion and resources. Instead of one fragment represents for the 8-bit range /16 to /24, we could split it into two 4-bit fragments, shown in Figure 10. Many other ways of segmenting the bit strides apply to different applications. Masking memory searching step could also be deducted from the structure by include the masking information in the rule ID vector table. Example is illustrated in Figure 11. For the cell in collision or empty state, there would be blank masking information.

The masking information could further be encoded to reduce memory consumption. For example, in IPv4 case, the real-world rule set could be found in [25] only have the masking from mostly /8. With 32-bit data key, 4-bit fragment normally it would require 8 bits to inform the masks (each bit for each segment). But the case is that IPv4 would only either be mask in /0, /4, /8, /12, /16, /20, /24, /28, /32. Those are 9 masking cases, so would require at most 4 bits to represent, which has halved the resources consumption.

The situation where range matching is needed could be done with registers in combination with the above structure or range conversion into masks could also be used but it would cost a huge amount of resources. The problem that need to be considered with range conversion is that the small range would collide with the large range data, causing many collision cells to appear in the rule ID table.

Many hash-table chaining and bucket methods could also be implement into the structure such as linear chaining or bucket cell rule ID table as shown in Figure 12. One important direction of improving algorithmic TCAM on FPGA is to make it possible to be im- plement on external memories with an interface that is fast enough to support industry-level requirement. The external memory could be DDR, HBM or even eSRAM.

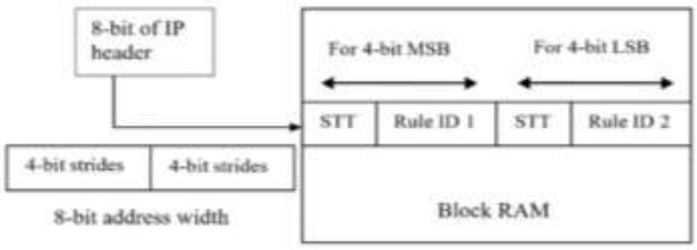
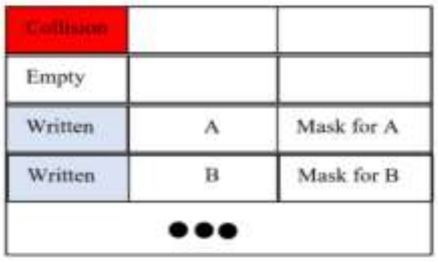


Figure 10. Splitting the original 8-bit fragment into two 4-bit fragments



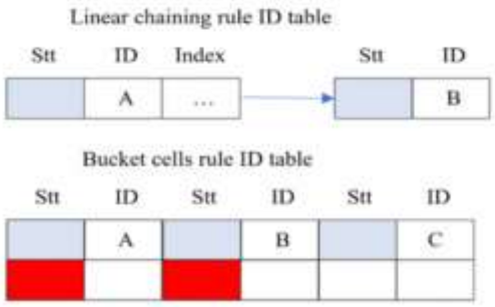


Figure 11. The mask vector information is included with the database that is used to search match vector

Figure 12. Many techniques that are used in hash-table searching could also be applied to the proposed structure

5. CONCLUSION

The paper presents a structure of RAM-based TCAM on FPGA. It has advantages in scalabilty and integration compare to traditional TCAMs. Its function has been improved and resource consumption is more optimized compare to other FPGA-based TCAM structure. We hope to achieve further improvements of FPGA-based TCAM design and even general TCAM in the future.

ACKNOWLEDGEMENTS

This research is funded by Ho Chi Minh City University of Technology-VNU-HCM under grant number T-ĐĐT-2020-44. We gratefully acknowledge their support.

REFERENCES

[1] TRW Computer Division Archived August 5, 2011, at the Wayback Machine, p. 17, 1963.
[2] Hucaby, David, “CCNP BCMSN exam certification guide: CCNP self-study,” Cisco Press, 2004.
[3] Pagiamtzis, Kostas, and Ali Sheikholeslami. ”Content-addressable memory (CAM) circuits and architectures: A tutorial and survey.” *IEEE journal of solid-state circuits*, vol. 41, no. 3, pp. 712-727, 2006.
[4] D. V. Garro, C. V. Caldero ́n and C. S. Yeung, ”Using a programmable network switch TCAM to find the best alignment of two DNA sequences,” *2016 IEEE 36th Central American and Panama Convention (CONCAPAN XXXVI), San Jose*, pp. 1-5, 2016.

[5] Z. Ullah, M. K. Jaiswal, Y. C. Chan and R. C. C. Cheung, ”FPGA Implementation of SRAM-based Ternary Content Addressable Memory,” *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, Shanghai*, pp. 383-389, 2012.
[6] Chiou, Derek, ”The microsoft catapult project,” *2017 IEEE International Symposium on Workload Characterization (IISWC). IEEE*, 2017.
[7] Sotiriades, Euripides, Christos Kozanitis, and Apostolos Dollas, ”FPGA based architecture for DNA se- quence comparison and database search,” *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium. IEEE*, 2006.
[8] M. Irfan, Z. Ullah and R. C. C. Cheung, ”D-TCAM: A High-Performance Distributed RAM Based TCAM Architecture on FPGAs,” *in IEEE Access*, vol. 7, pp. 96060-96069, 2019.
[9] Ullah, Anees, *et al*., ”BPR-TCAM-Block and Partial Reconfiguration based TCAM on Xilinx FPGAs.” *Electronics* vol. 9, no. 2, p. 353, 2020.
[10] Reviriego, Pedro, Anees Ullah, and Salvatore Pontarelli, ”PR-TCAM: Efficient TCAM emulation on Xilinx FPGAs using partial reconfiguration,” *IEEE Transactions on Very Large Scale Integration (VLSI)* Systems, vol. 27, no. 8, pp. 1952-1956, 2019.
[11] Panigrahy, Rina, and Samar Sharma. ”Reducing TCAM power consumption and increasing throughput.” *Proceedings 10th Symposium on High Performance Interconnects. IEEE*, 2002.
[12] N. U. Rehman, O. Mujahid, M. Irfan, A. HafeezandZ. Ullah, ”Low Power Pre-comparison Configuration Strategy for a Logic-based Binary CAM on FPGA,” *2019 Second International Conference on Latest trends in Electrical Engineering and Computing Technologies (INTELLECT), Karachi, Pakistan*, pp. 1-5, 2019.
[13] Z. Qian and M. Margala, ”Low power RAM-based hierarchical CAM on FPGA,” *2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig14), Cancun*, pp. 1-4, 2014.
[14] Ullah, Inayat, Zahid Ullah, and Jeong-A. Lee, ”EE-TCAM: An Energy-Efficient SRAM-Based TCAM on FPGA,” Electronics, vol. 7, no. 9, p. 186, 2018.
[15] Ullah, Inayat, Zahid Ullah and Jeong-A Lee, “Efficient TCAM Design Based on Multipumping-Enabled Multiported SRAM on FPGA,” *IEEE* Access, vol. 6, pp. 19940-19947, 2018.
[16] W. Jiang, ”Scalable Ternary Content Addressable Memory implementation using FPGAs,” *Architectures for Networking and Communications Systems, San Jose, CA*, pp. 71-82, 2013.
[17] Y. Sato, K. Otsuka, K. Kobayashi, T. Kouchi, M. Uwai and M. Nishizawa, ”Novel approach for search engine,” *2016 11th International Microsystems, Packaging, Assembly and Circuits Technology Confer- ence (IMPACT), Taipei, 2016, pp. 69-72. Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, Rhodes Island*, pp. 8, 2006.
[18] Li, Xianfeng, Yuanxin Lin, and Wenjun Li. ”GreenTCAM: A memory and energy efficient TCAM based packet classification.” *2016 International Conference on Computing, Networking and Communications (ICNC). IEEE*, 2016.
[19] W. Yu, S. Sivakumar and D. Pao, ”Pseudo-TCAM: SRAM-Based Architecture for Packet Classifica- tion in One Memory Access” *in IEEE Networking Letters*, vol. 1, no. 2, pp. 89-92, 2019.
[20] Syed, Farkhanda Ullah, Dr. Zahid Jaiswal, Manish, “Fast Content Updating Algorithm for an SRAM based TCAM on FPGA”. *IEEE Embedded Systems Letters.* pp. 1-1, 2017.
[21] Reviriego, Pedro, ”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.
[22] V. Srinivasan and G. Varghese. “Fast address lookups using controlled prefix expansion,” *ACM Trans. Comput. Syst*., vol. 17, no. 1, pp. 1-40, 1999.
[23] Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug ram rom.pdf
[24] Nygren, Anders, ”Openflow switch specification version 1.5. 1.” *Open Networking Foundation, Tech. Rep*, 2015.
[25] http://bgp.potaroo.net/as2.0/bgp-active.html