Design and Implementation of Mechanisms for Faster Updates in FPGA-based Ternary Content-addressable Memory

Dr.A.Ranganayakulu[1] Mr.D.Satyanarayana[2] M.Venkata Sahithi[3] Yeluri Sai Bhanu[4] Vangepurapu Shainyunaisy[5] Yeruva Sujatha[6]

[1,2] Krishna Chaithanya Institute Of Technology & Sciences, Ece Department, Markapur, Andhra Pradesh.
[3,4,5,6,] Krishna Chaithanya Institute Of Technology & Sciences, UG Student-ECE, Markapur, Andhra Pradesh.

Abstract In this project, Hardware, power consumption, and speed of ternary content-addressable memory (TCAMs) based on field-programmable gate arrays (FPGAs) are always changing. These emulated TCAMs' low update latency is one drawback. Conventional FPGA-based TCAMs, where N is the TCAM depth, have an update-latency of N clock cycles as opposed to a lookup-latency of one clock cycle. Afterwards, the update-latency is reduced to t clock cycles, where t is the quantity of bits that don't matter. This project introduced two FPGA-based TCAM updating techniques that were effectively implemented on Xilinx Vivado FPGA: an economical LUT-Update mechanism and an expedited MUX-Update mechanism. W is the width of the TCAM, and MUX-Update uses just three input/output (I/O) pins to produce an update-latency of W+1 clock cycles. Using W I/O pins, LUT-Update produces a consistent update-latency of 2 clock cycles, regardless of TCAM size.

Keywords: LUT(Look Up Table),5G Communications, FPGA, Verilog HDL, T-CAM.

I.INTRODUCTION

Content-addressable memory (CAM) returns the position of given search input in a single clock cycle [1]. The stored bits classify CAM into binary CAM (BCAM) and ternary CAM (TCAM). BCAM stores ‘1’ and ‘0’ while TCAM can store ‘1’, ‘0’ as well as ‘X’ (a don’t care) bit. CAMs are implemented in many applications, such as networking, signal processing, pattern recognition, access control lists, and translation lookaside buffers (TLB) in microprocessors.
Field-programmable gate arrays (FPGAs) are becoming popular because of its massive hardware parallelism, softwarelike reconfigurability, and rapid prototyping. FPGAs are enriched with hardware resources, such as dedicated block random-access memory (BRAM), multiplexers, and so forth. FPGA-based CAM utilizes these components to emulate content-based memory [2], [3]. They have near to one clock cycle search-latency (lookup-latency), but the update-latency is very high, which is unable to form a balanced system in which the update and search operation can happen at nearly the same speed.To exploit the bandwidth of high-performance systems, updating of memory with new data should be done at a considerable speed. The update-latency of a typical FPGAbased CAM is O(2N) [4]. The architecture in [5] reduces it to O(N) where N is the number of CAM words. It is further reduced to O(N/k) in [6], where k is a factor depending on the number of groups of stored words. Still, update-latency is high and varying, compared to the low (one clock cycle) and fixed search-latency of FPGA-based CAMs.

In applications such as networking, the update of the CAM table is very frequent, which should be fast enough to reach the search speed of CAM to achieve high bandwidth. The performance of FPGA-based CAMs degrades with increasing clock cycles of update-latency, which depends on

the number of stored words in the existing architectures [7]. The data packets arriving at a node in IP networking need to be buffered due to the slow updates to avoid packet loss [8], [9]. Thus, slow updates cause an extra overhead on the system in the form of a large buffer, which is saved if the update process is accelerated. In this letter, we proposed two update mechanisms, MUX-Update and LUT-Update, which update the TCAM in fewer clock cycles and in a cost-effective way than the available FPGA-based CAM's updating procedures [5], [6], [10].

Key contributions of the proposed work are:

Two mechanisms for updating FPGA-based TCAM are proposed. MUX-Update saves the Input/Output (I/O) pins at the cost of extra clock cycles, while LUT-Update efficiently updates TCAM at the expense of additional I/O pins. • G-AETCAM [11] supports only static lookup tables in its original form. Our proposed update mechanisms enable G-AETCAM to update the TCAM entries during runtime. • The available update mechanisms for FPGA-based TCAMs increase with the size of TCAM [6], [5], while our proposed technique (MUX-Update) has an update latency of fixed two clock cycles. • I/O pins, which is a critical constraint in the FPGA design process, are reduced by introducing a simple multiplexer in the TCAM design.

2.LITERATURE

SRAM-based memory designs using brute force approach are unable to be implemented on FPGA due to its huge resource requirement [4]. The increase of one bit per CAM width doubles the hardware resource requirement on FPGA [7]. Later these emulated CAM architectures based on SRAM cells were re-designed using partitioning of the TCAM table to implement on limited resources of FPGA [12], [13].

HP-TCAM arranges the SRAM cells into sub-blocks and stored TCAM bits in such a way that it becomes implementable on the SRAM blocks inside FPGA [12]. REST [7] further reduced the hardware resource requirement compared to HP-TCAM, but one disadvantage of these FPGA-based CAMs is its high update-latency, which reduces the overall system efficiency.

F. Syed et al. discussed an update mechanism for TCAM, especially the HP-TCAM, to update one word in N clock cycles [5]. It requires less than N clock cycles if there are some words with no ternary (don't care) bits, but in worst case, update-latency remains N clock cycles where N is the number of CAM words.

Wang et al. [10] divides the TCAM table into high and low priority blocks, and shows improvement in the update process. Worst-case update-latency remains N clock cycles, where N is the total number of locations in TCAM. In one of our proposed update mechanisms (MUX-Update), introduction of a multiplexer (MUX) into the system reduces the hardware cost (I/Os) to almost half of the previously used hardware. Mishra et al. [14] proposed a design based on leaf TCAM and interior TCAM, which increases the complexity of the algorithm, and involves hardware overhead in the form of storing filters.

Xuan et al. presented an update mechanism in [6] which reduced the update-latency by a factor k, from O(N) to O(N/k) by centralized erasing technique, where k is the number of chunks transfer via data bus. TCAM cannot benefit from this technique and is only applicable to binary

CAM architecture, while our proposed techniques are to update FPGA-based binary as well as ternary CAM and provides low updatelatency with reduced hardware cost. Gate-based ternary CAM (G-TCAM) [11] is a logical FPGA-based CAM implemented on FPGA with improved hardware resources from previous TCAM architectures.

II.EXISTING SYSTEM AND PROPOSED SYSTEM

1.EXISTING SYSTEM: TCAM ARCHITECTURE

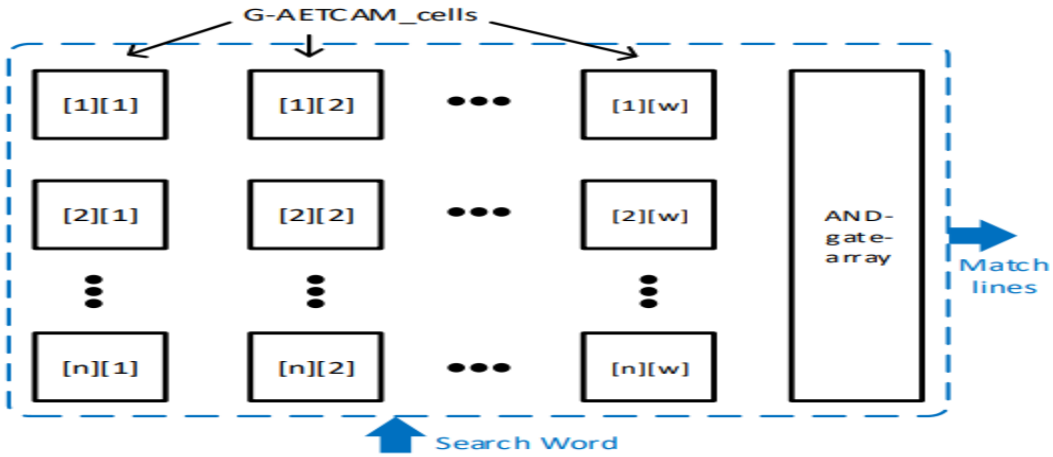

Fig.1. Simplified N×W G-AETCAM FOR N is depth of TCAM and W is width of TCAM).

TCAM architecture used to apply our proposed update mechanisms is a gate-based TCAM (G-AETCAM) implemented on flip-flops (FFs) as memory and gates as logical blocks of FPGA. To the best of our knowledge, G-AETCAM is the most efficient FPGA-based TCAM architecture which uses less logical resources than the available TCAM architectures. It consists of G-AETCAM cells that are capable of handling ‘masking bits’ and ‘storing bits’ to store TCAM bits, as shown in Fig. 1. FFs are used as storage media, which are also referred to as register file inside FPGAs. An N×W G-AETCAM architecture supports only static lookup tables and cannot be updated at runtime. This makes it inefficient for practical applications where the TCAM entries update dynamically. Our proposed designs enable G-AETCAM to dynamically update its entries during runtime. Two update mechanisms based on the availability of hardware resources are proposed in this work to achieve the comprehensive benefits of G-AETCAM in terms of integrating it with practical applications.

Designing a Ternary Content-Addressable Memory (TCAM) based on FPGA (Field-Programmable Gate Array) involves creating a hardware implementation that leverages the reconfigurability and parallel processing capabilities of FPGAs. Below are the general steps involved in implementing TCAM on an FPGA:

1. Define TCAM Requirements:

Clearly define the requirements of the TCAM, including the number of entries, bit width, and the ability to store ternary values (0, 1, and "don't care").

2. FPGA Architecture Selection:

Choose an FPGA with sufficient resources (look-up tables, flip-flops, and memory blocks) to accommodate the TCAM design. The FPGA should also support the required I/O standards and clock frequencies.

3. Create TCAM Core:
Develop a custom TCAM core that consists of ternary memory cells and associated control logic. Each memory cell should be capable of storing ternary values and support parallel search operations.
4. Parallel Search Mechanism:
Implement a parallel search mechanism within the TCAM core. This involves designing the logic to simultaneously compare input search keys with multiple entries in parallel.
5. Memory Storage Implementation:
Implement the storage of ternary values in memory cells. You may use dedicated memory blocks on the FPGA or a combination of lookup tables and flip-flops to store the TCAM entries.
6. Write and Read Operations:
Develop the logic for write and read operations to allow the programming of TCAM entries and the retrieval of matching entries based on input search keys.
7. Interface Integration:
Integrate the TCAM core with other components of the FPGA design. This includes interfaces for external communication, such as input and output ports, and any necessary control and configuration interfaces.
8. Clock Domain Considerations:
Manage clock domains appropriately to ensure synchronization and avoid potential issues related to clock domain crossings.
9. Testing and Verification:
Implement comprehensive testbenches and verification procedures to ensure the correct functionality of the TCAM design under various conditions. FPGA simulation tools and hardware testing can be used for verification.
10. Synthesis and Implementation:
Use FPGA synthesis tools provided by the FPGA vendor to convert the hardware description (e.g., written in Verilog or VHDL) into a bitstream that can be programmed onto the FPGA.
11. Optimization:
Optimize the TCAM design for area, speed, and power consumption. FPGA tools often provide optimization options to achieve better performance.
12. Integration into Larger Systems:
Integrate the TCAM design into the larger FPGA-based system. This may involve additional interfaces, interconnects, and coordination with other FPGA components.
13. Deployment and Iteration:
Program the synthesized bitstream onto the FPGA and deploy the system. Iterate through the design process as needed to address any issues or make improvements.

2.PROPOSED SYSTEM

A. Proposed MUX-Update Mechanism

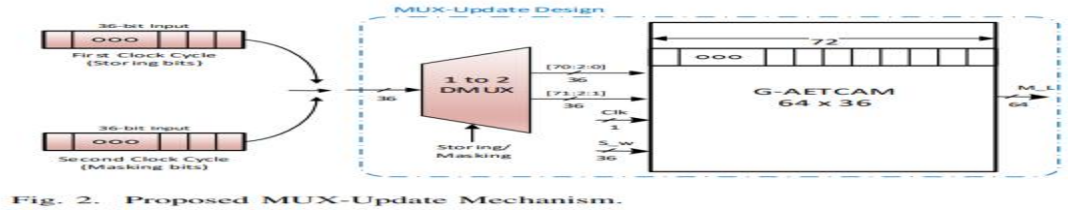

Fig. 2. Proposed MUX-Update Mechanism.

Fig. 2 shows the proposed architecture for MUX-Update mechanism to update the TCAM.

It enables the TCAM to update an entry in fixed two clock cycles. The first clock cycle is to store the "storing bits" of G-AETCAM at even bit positions of each word location while the second clock cycle is to save the "masking bits" of G-AETCAM at odd bit positions of each word location. The even/odd positions are used to reduce the I/O pins to half because of the limited resources on FPGA.

The select line of 1-to-2 DMUX for storing bits is '0', which for masking bits is '1' as shown in Fig 2. MUX Update mechanism updates the TCAM word in two clock cycles represented by line # 4 and 8 in Algorithm 1.

Algorithm 1 Updating 64×36 TCAM using MUX-Update Mechanism

Input: *W* TCAM bits (X, 1, 0), where *W* is 36.
Output: G-AETCAM bits (1, 0)

```
1:  G-AETCAM [0:71] = "000...00";
2:  if SM == '0' then
3:     for i ← 0 to i ← 35 do
4:        G-AETCAM [2*i] = TCAM [i];
5:     end for
6:  else
7:     for i ← 0 to i ← 35 do
8:        if TCAM [i] == '0' or TCAM [i] == '1' then
9:           G-AETCAM [(2*i)+1] = '0';
10:       else if (TCAM [i] == 'x') then
11:          G-AETCAM [(2*i)+1] = '1';
12:       end if
13:    end for
14: end if
```

Note: SM: Storing/Masking bit.

B. Proposed LUT-Update Mechanism

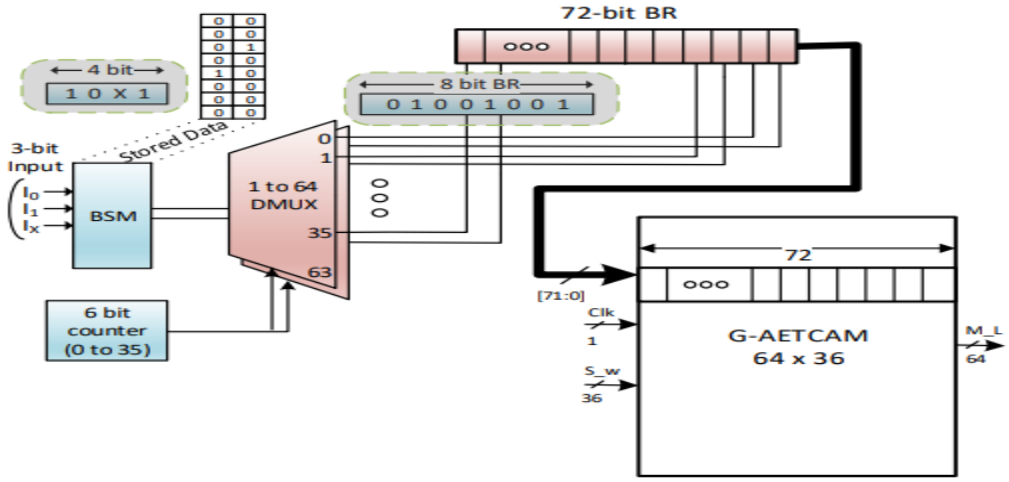

Fig. 3. Proposed LUT-Update. (BR: Buffer register, BSM: Bit-Select-Memory).

Fig. 3 shows the proposed architecture for LUT-Update mechanism which updates the TCAM in W+1 clock cycles, where W is the width of TCAM. Bit-Select-Memory (BSM) consists of two 3-input LUTs which outputs the appropriate value to the input of MUX for 0, 1 and X bits of TCAM, using I0, I1 and Ix input pins, respectively. For instance, an entry "10X1" needs to update in TCAM as highlighted in Fig. 3. The BSM stores "01", "00", "10", and

"01" in 8-bit BR (buffer register) which is transferred to the corresponding location of TCAM in next clock cycle. The update-latency, in this case, is five clock cycles, where W is 4. For a 36-bit word, this process is repeated 36 times to fill the BR with storing & masking bit. In the next clock cycle, the content of BR transfers to the TCAM location, which is represented by line # 7 in Algo. 2.

The Switch statement in Algo. 2 handles different input values of BSM. One of the three cases is selected to provide input values to two demultiplexers (DMUXes) for the corresponding location of BR. The number of iterations in the for loop is dependent on the width of TCAM memory.

Algorithm 2 Updating TCAM using LUT-Update Mechanism
Input: TCAM bits (X, 1, 0)
Output: G-AETCAM bits (1, 0)

```
1: for i ← 0 to i ← 35 do
2:     Switch I_x I_1 I_0 do
3:         Case "001":
               BR [2*i, (2*i)+1] = "00";
4:         Case "010":
               BR [2*i, (2*i)+1] = "01";
5:         Case "100":
               BR [2*i, (2*i)+1] = "10";
6: end for
7: G-AETCAM [0:71] = BR [0:71];
```

Note: BR: Buffer Register; I_x, I_1 and I_0 are the three inputs of Lookup table shown in Fig. 3.

III. RESULTS AND ANALYSIS DISCUSSION

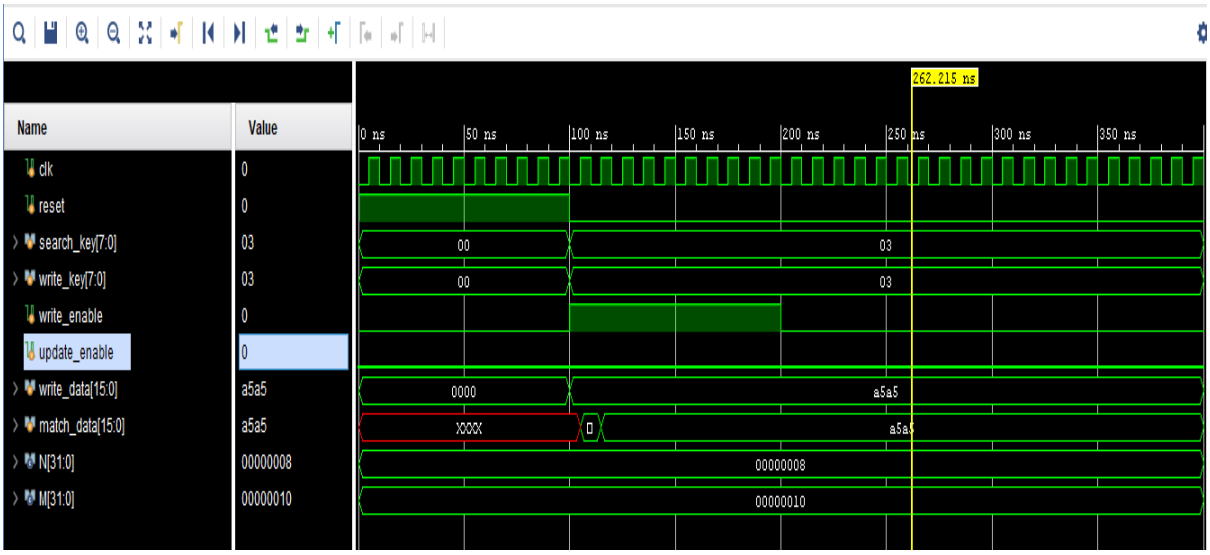

FIG1.TACM SIMULATION OUTPUT

```
# run 1000ns
Time=5000 search_key=00 write_key=00 write_data=0000 match_data=xxxx
Time=15000 search_key=00 write_key=00 write_data=0000 match_data=xxxx
Time=25000 search_key=00 write_key=00 write_data=0000 match_data=xxxx
Time=35000 search_key=00 write_key=00 write_data=0000 match_data=xxxx
Time=45000 search_key=00 write_key=00 write_data=0000 match_data=xxxx
Time=55000 search_key=00 write_key=00 write_data=0000 match_data=xxxx
Time=65000 search_key=00 write_key=00 write_data=0000 match_data=xxxx
Time=75000 search_key=00 write_key=00 write_data=0000 match_data=xxxx
Time=85000 search_key=00 write_key=00 write_data=0000 match_data=xxxx
Time=95000 search_key=00 write_key=00 write_data=0000 match_data=xxxx
Time=105000 search_key=03 write_key=03 write_data=a5a5 match_data=xxxx
Time=115000 search_key=03 write_key=03 write_data=a5a5 match_data=0000
Time=125000 search_key=03 write_key=03 write_data=a5a5 match_data=a5a5
Time=135000 search_key=03 write_key=03 write_data=a5a5 match_data=a5a5
Time=145000 search_key=03 write_key=03 write_data=a5a5 match_data=a5a5
Time=155000 search_key=03 write_key=03 write_data=a5a5 match_data=a5a5
Time=165000 search_key=03 write_key=03 write_data=a5a5 match_data=a5a5
Time=175000 search_key=03 write_key=03 write_data=a5a5 match_data=a5a5
Time=185000 search_key=03 write_key=03 write_data=a5a5 match_data=a5a5
Time=195000 search_key=03 write_key=03 write_data=a5a5 match_data=a5a5
Time=205000 search_key=03 write_key=03 write_data=a5a5 match_data=a5a5
Time=215000 search_key=03 write_key=03 write_data=a5a5 match_data=a5a5
Time=225000 search_key=03 write_key=03 write_data=a5a5 match_data=a5a5
Time=235000 search_key=03 write_key=03 write_data=a5a5 match_data=a5a5
Time=245000 search_key=03 write_key=03 write_data=a5a5 match_data=a5a5
```

FIG2.OUTPUT OF TCAM

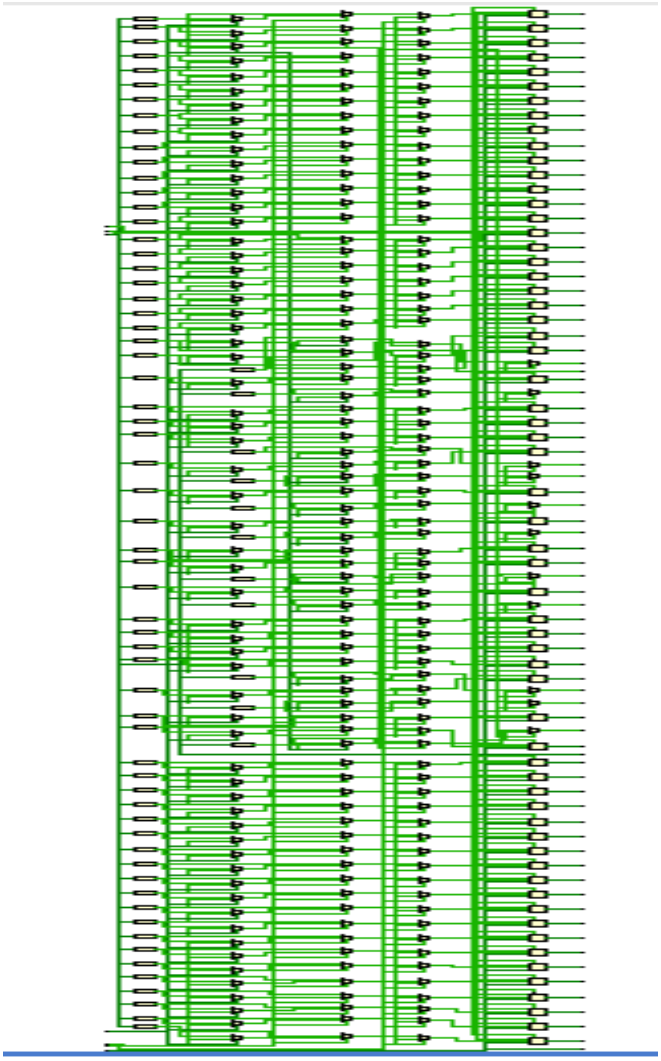

FIG3.RTL SCHEMATIC OF TCAM.

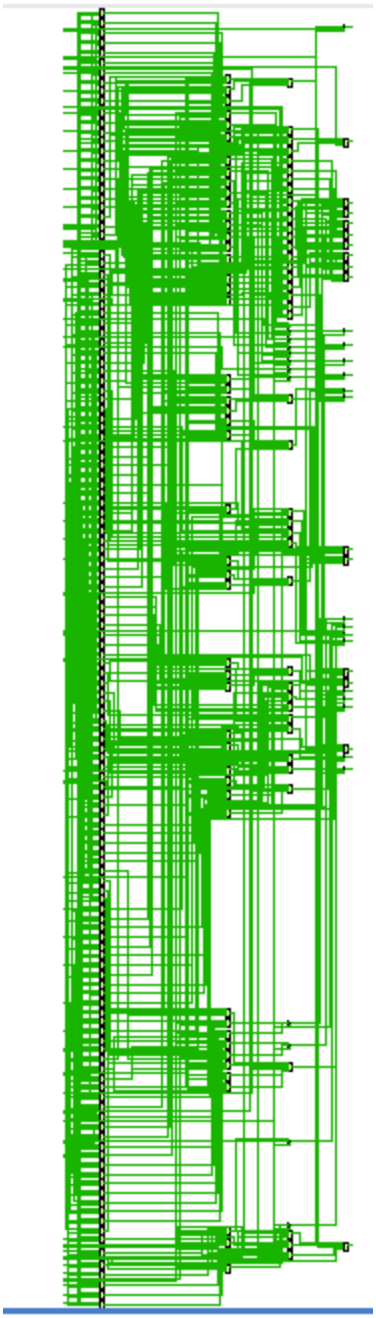

FIG.4.SYNTHESIS DESIGN OF TCAM

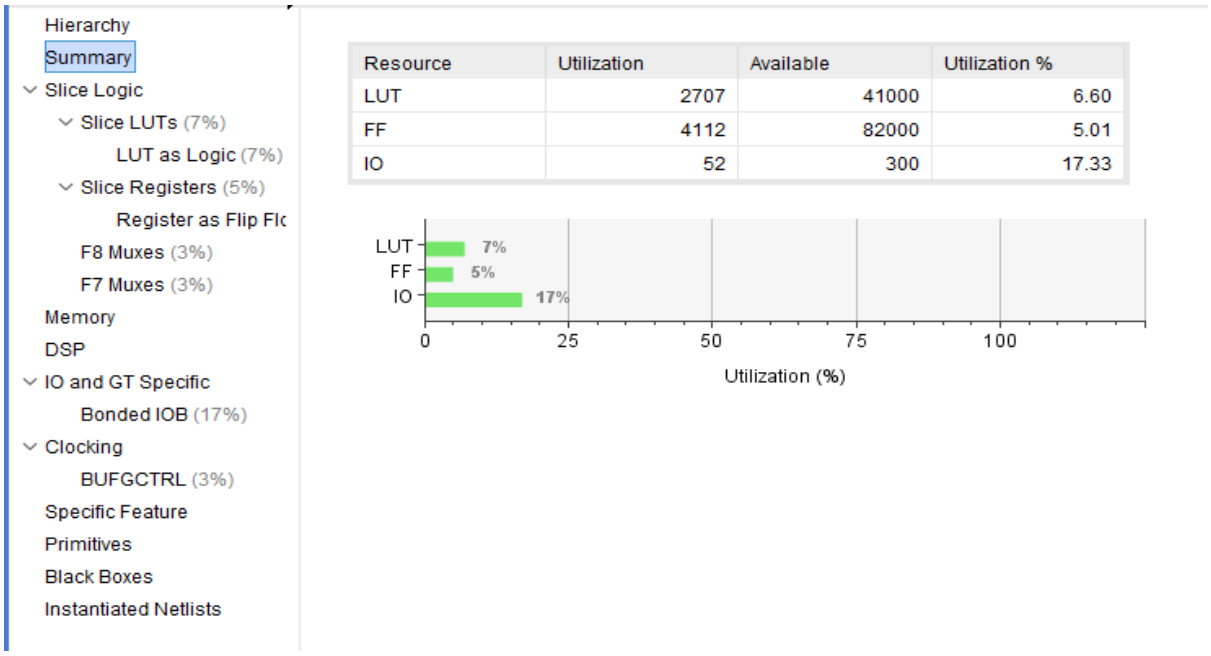

FIG.5.UTILIZATION OF LUTS REPORT

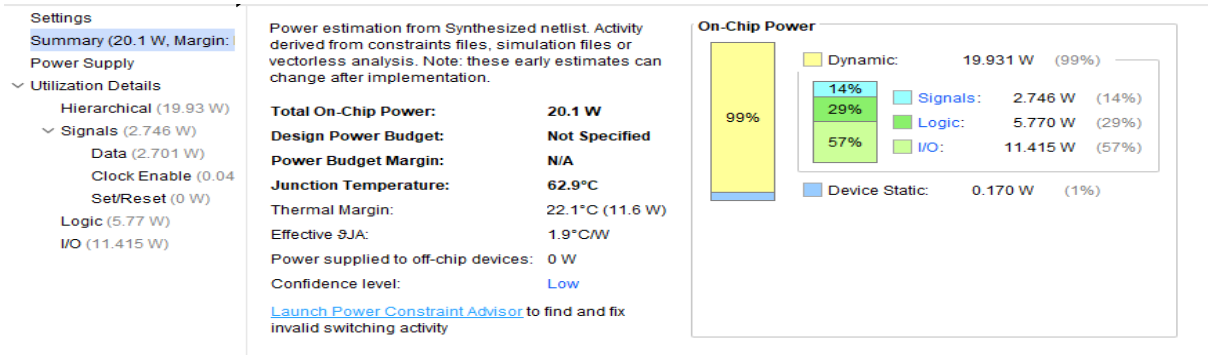

FIG.6.POWER REPORT OF TCAM.

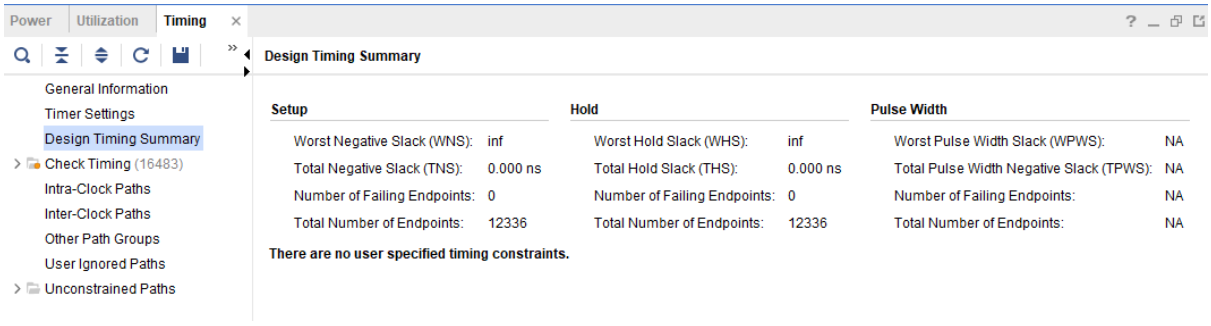

FIG.7.TIMING REPORT OF TCAM

CONCLUSION

Longer update-latency of TCAM affects the efficiency of searching-based systems, e.g., software-defined networks (SDN) on FPGAs. Our proposed techniques enable FPGAbased TCAM architecture to update the TCAM entries at lower update-latency and in a cost-effective way in terms of I/O pins usage.

In conclusion, Ternary Content Addressable Memory (TCAM) stands as a pivotal technology in the realm of computing and networking, offering unparalleled capabilities in fast and efficient data searching. Its unique ability to perform content-based searches in a single clock cycle has made it indispensable in a variety of applications, ranging from network routing and security to telecommunications and storage systems.

REFERENCES

[1] I. Ullah, Z. Ullah, and J.-A. Lee, "Efficient TCAM design based on multipumping-enabled multiported SRAM on FPGA," IEEE Access, vol. 6, pp. 19 940–19 947, 2018.

[2] R. Karam, R. Puri, S. Ghosh, and S. Bhunia, "Emerging trends in design and applications of memory-based computing and content-addressable memories," Proceedings of the IEEE, vol. 103, no. 8, pp. 1311–1330, 2015.

[3] P. Reviriego, A. Ullah, and S. Pontarelli, "PR-TCAM: Efficient TCAM emulation on xilinx FPGAs using partial reconfiguration," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, pp. 1–5, 2019.

[4] M. Somasundaram, "Circuits to generate a sequential index for an input number in a pre-defined list of numbers," Dec. 26 2006, US Patent 7,155,563.

[5] F. Syed, Z. Ullah, and M. K. Jaiswal, "Fast Content Updating Algorithm for an SRAM based TCAM on FPGA," IEEE Embedded Systems Letters, 2017.

[6] X.-T. Nguyen, T.-T. Hoang, H.-T. Nguyen, K. Inoue, and C.-K. Pham, "An efficient I/O architecture for RAM-based content-addressable memory on FPGA," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 66, no. 3, pp. 472–476, 2018.

[7] A. Ahmed, K. Park, and S. Baeg, "Resource-Efficient SRAM-Based Ternary Content Addressable Memory," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 25, no. 4, pp. 1583–1587, 2017.

[8] C. A. Zerbini and J. M. Finochietto, "Performance evaluation of packet classification on FPGA-based TCAM emulation architectures," in Global Communications Conference (GLOBECOM), 2012 IEEE. IEEE, 2012, pp. 2766–2771.

[9] T. Nguyen-Viet and D.-H. Le, "TCAM-based flow lookup design on FPGA and its applications," in 2015 International Conference on Advanced Technologies for Communications (ATC). IEEE, 2015, pp. 378– 382.

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

[11] M. Irfan and Z. Ullah, "G-AETCAM: Gate-Based Area-Efficient Ternary Content-Addressable Memory on FPGA," IEEE Access, vol. 5, pp. 20 785–20 790, 2017.

[12] Z. Ullah, M. K. Jaiswal, R. C. Cheung, and H. K. So, "UE-TCAM: An ultra efficient SRAM-based TCAM," in TENCON 2015-2015 IEEE Region 10 Conference. IEEE, 2015, pp. 1–6.

[13] W. Jiang, "Scalable ternary content addressable memory implementation using FPGAs," in Proceedings of the ninth ACM/IEEE symposium on Architectures for networking and communications systems. IEEE Press, 2013, pp. 71–82.

[14] T. Mishra, S. Sahni, and G. Seetharaman, "PC-DUOS: Fast TCAM lookup and update for packet classifiers," in 2011 IEEE Symposium on Computers and Communications (ISCC). IEEE, 2011, pp. 265–270.

[15] Z. Ullah, K. Ilgon, and S. Baeg, "Hybrid partitioned SRAM-based ternary content addressable memory," IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 59, no. 12, pp. 2969–2979, 2012.

[16] H. Mahmood, Z. Ullah, O. Mujahid, I. Ullah, and A. Hafeez, "Beyond the limits of typical strategies: Resources efficient FPGA-based TCAM," IEEE Embedded Systems Letters, 2018.