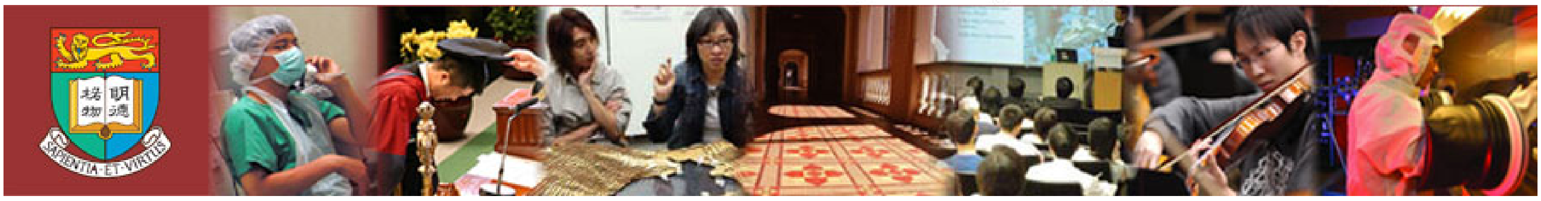

Title	Z-TCAM: An SRAM-based Architecture for TCAM
Author(s)	Ullah, Z; Jaiswal, MK; Cheung, RCC
Citation	IEEE Transactions on Very Large Scale Integration Systems, 2015, v. 23, p. 402-406
Issued Date	2015
URL	http://hdl.handle.net/10722/214167
Rights	Creative Commons: Attribution 3.0 Hong Kong License

Z-TCAM: An SRAM-based Architecture for TCAM

Zahid Ullah, Manish K. Jaiswal, and Ray C. C. Cheung

Abstract— **Ternary content addressable memories (TCAMs) perform high-speed lookup operation but when compared with static random access memories (SRAMs), TCAMs have certain limitations such as low storage density, relatively slow access time, low scalability, complex circuitry, and are very expensive. Thus, can we use the benefits of SRAM by configuring it (with additional logic) to enable it to behave like TCAM? This brief proposes a novel memory architecture, named Z-TCAM, which emulates the TCAM functionality with SRAM. Z-TCAM logically partitions the classical TCAM table along columns and rows into hybrid TCAM subtables, which are then processed to map on their corresponding memory blocks. Two example designs for Z-TCAM of sizes 512 × 36 and 64 × 32 have been implemented on Xilinx Virtex-7 field-programmable gate array. The design of 64 × 32 Z-TCAM has also been implemented using OSUcells library for 0.18 μm technology, which confirms the physical and technical feasibility of Z-TCAM. Search latency for each design is three clock cycles. The detailed implementation results and power measurements for each design have been reported thoroughly.**

Index Terms— **Application-specific integrated circuit (ASIC), field-programmable gate array (FPGA), memory architecture, priority encoder, static random access memory (SRAM)-based TCAM, ternary content addressable memory (TCAM).**

I. Introduction

Ternary content addressable memory (TCAM) allows its memory to be searched by contents rather than by an address and a memory location among matches is sent to the output in a constant time. A typical TCAM cell has two static random access memory (SRAM) cells and a comparison circuitry and has the ability to store three states – 0, 1, and x where x is a don't care state. The x state is always regarded as matched irrespective of the input bit. The constant time search of TCAM makes it a suitable candidate in different applications such as network routers, data compression, real-time pattern matching in virus-detection, and image processing [1].

TCAM provides single clock lookup; however, it has several disadvantages compared with SRAM. TCAM is not subjected to the intense commercial competition found in the RAM market [2]. TCAM is less dense than SRAM. The comparator's circuitry in TCAM cell adds complexity to the TCAM architecture. The extra logic and capacitive loading due to the massive parallelism lengthen the access time of TCAM, which is 3.3 times longer than the SRAM access time [3]. Inborn architectural barriers also limit the total chip capacity of TCAM. Complex integration of memory and logic also makes TCAM testing very time consuming [1].

Furthermore, the cost of TCAM is about 30 times more per bit of storage than SRAM [4]. RAM is available in a wider variety of sizes and flavors, is more generic and widely available, and enables to avoid the heavy licensing and royalty costs charged by some CAM vendors [5]. CAM devices have very limited pattern capacity and also CAM technology does not evolve as fast as the RAM technology [6].

Field-programmable gate array (FPGA) is used in many applications, for example, in networking systems [7] and [8] owing to several reasons that include its reconfigure-ability, massive hardware parallelism, and rapid prototyping capability. Recent FPGA devices such as Xilinx Virtex-7 [9] provide high clock rate and a large amount of on-chip dual-port memory with configurable word width. Currently, TCAMs are used in networking systems but they are expensive and not scalable with respect to clock rate or circuit area compared with RAMs [10]. The throughput of classical TCAMs is also limited by the relatively low speed of TCAMs [11]. Thus, SRAM- and FPGA-based TCAMs can be used in applications such as in networking chips to achieve high speed and high throughput.

With the potential advantages of SRAM over CAM, and feasibility of FPGA technology, we propose a memory architecture called Z-TCAM that emulates TCAM functionality with SRAM and has been successfully implemented on Xilinx Virtex-7 FPGA and also designed using OSUcells library for 0.18 μm technology. We assure that the proposed TCAM offers comparable search performance, scalability, and lower cost than classical TCAM devices, provided that SRAM devices are denser, cheaper, and operate faster than TCAM devices.

A. Related Work

We summarize RAM-based solutions for CAM in this section. The methods proposed in [2] and [12] use hashing to build CAM from RAM but these methods suffer from collisions and bucket overflow. If many records have been placed in an overflow area, then a lookup may not finish until many buckets are searched. In [12], when stored keys contain don't care bits in the bit positions used for hashing, then such keys must be duplicated in multiple buckets, which need increased capacity. On the other hand, if the search key contains don't care bits which are taken by the hash function, multiple buckets must be accessed that results in performance degradation. In [2], the performance of the method becomes gracefully degradable as the number of stored elements increases. Furthermore, it emulates binary CAM, not TCAM. Thus, hashing cannot provide deterministic performance owing to potential collisions and is inefficient in handling wildcard. Traditional algorithmic search solutions take multiple clock cycles [11] and also result in inefficient memory utilization [10]. In contrast, Z-TCAM has a deterministic search performance that is independent of data, efficiently handles the wild-cards, and has better memory utilization.

The method proposed in [13] combines RAM and CAM to develop the CAM functionality. This approach makes partitions of the conventional TCAM table using some distinguishing bits in CAM entries. But making partitions of totally random data is a very tedious and time consuming job. Because the method uses TCAM as a part of the overall architecture, it brings the intrinsic TCAM disadvantages in the overall architecture of [13] but Z-TCAM is generic and has an easy partitioning scheme.

RAM-based CAMs presented in [6] and [14] have an exponential increase in memory size with the increase in number of bits in CAM word, thus making them prohibitive. For instance, if a CAM word

Manuscript received August 13, 2012; revised March 28, 2013, July 11, 2013, and January 13, 2014; accepted February 19, 2014. Date of publication March 18, 2014; date of current version January 30, 2015. This work was supported in part by the Research Grant Council of the Hong Kong Special Administrative Region under Project CityU 123612 and in part by the Croucher Startup Allowance.

The authors are with the Department of Electronic Engineering, City University of Hong Kong, Hong Kong (e-mail: zullah2-c@my.cityu.edu.hk; mkjaiswal2-c@my.cityu.edu.hk; r.cheung@cityu.edu.hk).

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

Digital Object Identifier 10.1109/TVLSI.2014.2309350

TABLE I
TRADITIONAL TCAM TABLE AND ITS HYBRID PARTITIONS (HP)

Address	Ternary Data		Layer
0	00	11	
1	01 $\mathtt{HP}_{11}$	01 $\mathtt{HP}_{12}$	1
2	$0x$	11	
3	11 $\mathtt{HP}_{21}$	$1x$ $\mathtt{HP}_{22}$	2

TABLE II
Z-TCAM EXAMPLE: DATA MAPPING

Address	$\mathtt{VM}_{21}$	$\mathtt{VM}_{22}$	$\mathtt{OATAM}_{21}$	$\mathtt{OATAM}_{22}$	Original Address			
					$\mathtt{OAT}_{21}$		$\mathtt{OAT}_{22}$	
					2	3	2	3
0	1	0	0	–	1	0	0	1
1	1	0	1	–	1	0	1	1
2	0	1	–	0	0	1	0	0
3	1	1	2	1	0	0	0	0

has 36 bits, its size would be $2^{36} = 64$ GB in [6]. Furthermore, the method in [14] only works on ascended data but in typical CAM applications data are totally random. By arranging the data in ascending order, the original order of entries is disturbed. So, there must be a way to store the original addresses, which is lacked by [14]. If original addresses are considered, the memory and power requirements further increase. In contrast to [6] and [14], Z-TCAM supports an arbitrarily large bit pattern, considers the storage of original addresses, while using appropriate partitioning.

B. Paper Organization

The rest of this brief is organized as follows: Section II elaborates hybrid partitioning. Section III discusses the architecture of Z-TCAM. Section IV explains Z-TCAM operations with examples. Section V provides implementation of Z-TCAM and Section VI concludes the brief along with highlighting the future work.

II. HYBRID PARTITIONING OF TCAM TABLE

Hybrid partitioning (HP) is a collective name given to vertical partitioning and horizontal partitioning of the conventional TCAM table. An example of HP is given in Table I. HP partitions conventional TCAM table vertically (columnwise) and horizontally (rowwise) into TCAM subtables, which are then processed to be stored in their corresponding memory units. This processing (data mapping) has been explained in Section IV-A with an example (Table II) to demonstrate the layer architecture of Z-TCAM. Vertical partitioning (VP) implies that a TCAM word of C bits is partitioned into N subwords; each subword is of w bits. VP is used in Z-TCAM to decrease memory size as much as possible. Horizontal partitioning (HrP) divides each vertical partition using the original address range of conventional TCAM table into L horizontal partitions. HrP cannot be used alone as it is area, power, and cost hungry but is used to create layers. HP results in a total of $L \times N$ hybrid partitions.

The dimensions of each hybrid partition are $K \times w$ where K is a subset from original addresses and w is the number of bits in a subword. Hybrid partitions spanning the same addresses are in the same layer. For example, HP_{21} and HP_{22} span the same address range and are in layer 2.

III. ARCHITECTURE OF Z-TCAM

A. Overall Architecture

The overall architecture of Z-TCAM is depicted in Fig. 1 where each layer represents the architecture shown in Fig. 2. It has L layers and a CAM priority encoder (CPE). Each layer outputs a potential match address (PMA). The PMAs are fed to CPE, which selects match address (MA) among PMAs.

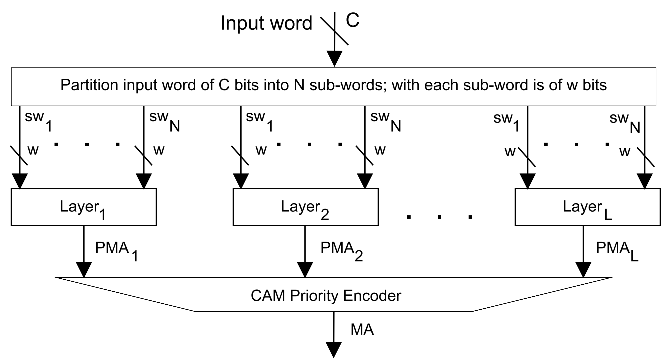
Input word
C
Partition input word of C bits into N sub-words; with each sub-word is of w bits
sw1
swN
w
Layer1
Layer2
LayerL
PMA1
PMA2
PMAL
CAM Priority Encoder
MA

Fig. 1. Architecture of Z-TCAM. (sw: subword, C: # of bits in the input word, PMA: potential match address, and MA: match address).

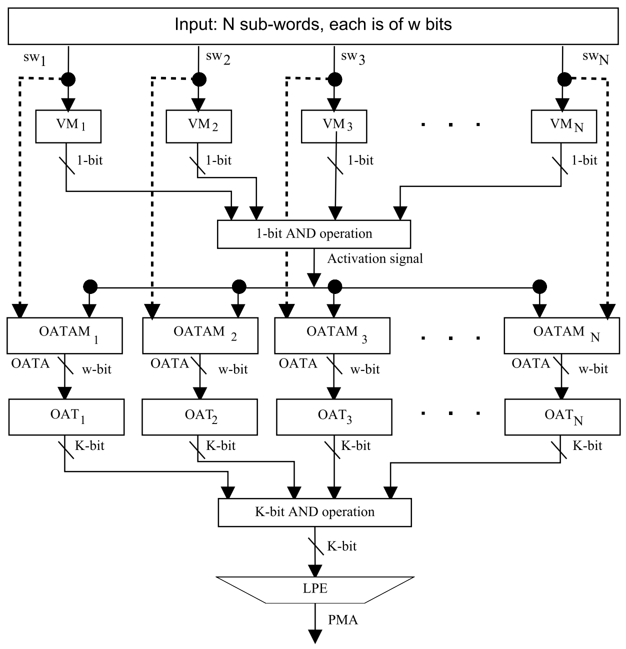

Fig. 2. Architecture of a layer of Z-TCAM. (sw: subword, VM: validation memory, OATAM: original address table address memory, OAT: original address table, and LPE: layer priority encoder).

B. Layer Architecture

Layer architecture is shown in Fig. 2. It contains N validation memories (VMs), 1-bit AND operation, N original address table address memories (OATAMs), N original address tables (OATs), K-bit AND operation, and a layer priority encoder (LPE).

1) Validation Memory: Size of each VM is $2^w \times 1$ bits where w represents the number of bits in each subword and 2^w shows the number of rows. A subword of w bits implies that it has total combinations of 2^w where each combination represents a subword. For example, if w is of 4 bits, then it means that there are total of $2^4 = 16$ combinations. This explanation is also related to OATAM and OAT. Each subword acts as an address to VM. If the memory location be invoked by a subword is high, it means that the input subword is present, otherwise absent. Thus, VM validates the input subword, if it is present. For example, Table II shows that subwords $00, 01$, and 11 are mapped in VM_{21}. This states that memory locations $00, 01$, and 11 should be high in VM_{21} and the remaining memory locations are set to low because their corresponding subwords do not exist.

2) 1-Bit AND *Operation:* It ANDs the output of all VMs. The output of 1-bit AND operation decides the continuation of a search operation. If the result of 1-bit AND operation is high, then it permits the continuation of a search operation, otherwise mismatch occurs in the corresponding layer.

3) Original Address Table Address Memory: Each OATAM is of $2^w \times w$ bits where 2^w is the number of rows and each row has w bits. In OATAM, an address is stored at the memory location indexed by a subword and that address is then used to invoke a row from its corresponding OAT. If a subword in VM is mapped, then a corresponding address is also stored in OATAM at a memory location accessed by the subword. For example, Table II shows OATAM_{21} where addresses are stored at the memory locations 00, 01, and 11. The output of OATAM is called as OATA. Hyphen "-" indicates that the corresponding memory location has no data because the corresponding subword for the memory location is not present in VM.

4) Original Address Table: Dimensions of OAT are $2^w \times K$ where w is the number of bits in a subword, 2^w represents number of rows, and K is the number of bits in each row where each bit represents an original address. Here K is a subset of original addresses from conventional TCAM table. It is OAT, which considers the storage of original addresses. An example of OAT is given in Table II, where 1 shows the presence of a subword at an original address.

5) K-Bit AND *Operation:* It ANDs bit-by-bit the read out K-bit rows from all OATs and forwards the result to LPE.

6) Layer Priority Encoder: Because we emulate TCAM and multiple matches may occur in TCAM [15], the LPE selects PMA among the outputs of K-bit AND operation.

IV. Z-TCAM Operations

A. Data Mapping Operation

Classical TCAM table is logically partitioned into hybrid partitions. Each hybrid partition is then expanded into a binary version. Thus, we first expand x into states 0 and 1 to be stored in SRAM. For example, if we have a TCAM word of $010x$, then it is expanded into 0100 and 0101. Each subword, acting as an address, is applied to its corresponding VM and a logic "1" is written at that memory location. The same subword is also applied to its respective OATAM and w bits data are written at that memory location. During search, these w bits data act as an address to the OAT. The K bits data are also written at the memory location in OAT determined by its corresponding OATA. Thus, in this way, all hybrid partitions are mapped.

A subword in a hybrid partition can be present at multiple locations. So, it is mapped in its corresponding VM and its original address(es) is/are mapped to its/their corresponding bit(s) in its respective OAT. Since a single bit in OAT represents an original address, only those memory locations in VMs and address positions/original addresses in OATs are high, which are mapped while remaining memory locations and address positions are set to low in VMs and OATs, respectively.

Example of data mapping is shown in Table II. We use Table I to be mapped to Z-TCAM. We take $N = 2$, $L = 2$, $K = 2$, and $w = 2$. After necessary processing, HP_{11}, HP_{12}, HP_{21}, and HP_{22} are mapped to their corresponding memory units. In the example, we map hybrid partitions of layer 2 to their corresponding memory units. Hybrid partitions of layer 1 can be easily mapped in similar way.

B. Search Operation

1) Searching in a Layer of Z-TCAM: Algorithm 1 describes searching in a layer of Z-TCAM. N subwords are concurrently applied to a layer. The subwords then read out their corresponding memory locations from their respective VMs.

Algorithm 1 Pseudocode for Searching in a Layer of Z-TCAM

```
INPUT: N sub-words
OUTPUT: PMA
 1: → Apply N sub-words
 2: → Apply all sub-words simultaneously to their VMs
 3: → Read all VMs concurrently
 4: if all VMs validate their corresponding sub-words then
 5:     → Sustain search operation
 6:     → a. Read all OATAMs in parallel
 7:     → b. Read all OATs simultaneously
 8:     → c. AND bit-wise all K-bits rows
 9:     → d. Select PMA/mismatch occurs
10: else
11:     → Mismatch occurs
12: end if
```

TABLE III
Example of a Search Operation in Layer 2 of Z-TCAM

Steps	Activity
1	$\text{Sub-word}_1 = 00$ $\text{Sub-word}_2 = 11$
2	Sub-word_1 is applied to VM_{21} Sub-word_2 is applied to VM_{22}
3	Read out bit from $\text{VM}_{21} = 1$ Read out bit from $\text{VM}_{22} = 1$
4	Have all VMs validated their corresponding sub-words?
5	Yes, so sustain search operation
6	Read out data from $\text{OATAM}_{21} = 0$ Read out data from $\text{OATAM}_{22} = 1$
7	Read out data from $\text{OAT}_{21} = 10$ Read out data from $\text{OAT}_{22} = 11$
8	K-bit AND operation result = 10
9	PMA = 2

Algorithm 2 Pseudocode for Searching in Z-TCAM

```
INPUT: Search Key
OUTPUT: MA
 1: → Apply search key
 2: → Divide search key into N sub-words
 3: → All layers use algorithm 1 in parallel
 4: → Select MA among PMAs/mismatch occurs
```

If all VMs validate their corresponding subwords (equivalent to 1-bit AND operation in Fig. 2), then searching will continue, otherwise mismatch occurs in the layer. Upon validation of all subwords, the subwords read out their respective memory locations from their corresponding OATAMs concurrently and output their corresponding OATAs. All OATAs then read out K-bit rows from their corresponding OATs simultaneously, which are then bitwise ANDed. LPE selects PMA from the result of the K-bit AND operation. Example of a search operation in layer 2 is shown in Table III, following Algorithm 1. Memory blocks in Table II need to be searched.

2) Searching in Z-TCAM: Search operation in the proposed TCAM occurs concurrently in all layers, which follows Algorithm 2. Search key is applied to Z-TCAM, which is then divided into N subwords. After searching, PMAs are available from all layers. CPE selects MA among PMAs; otherwise a mismatch of the input word occurs.

Table IV provides overall search operation in Z-TCAM, which follows Algorithm 2. We use input word 0011 to be searched.

TABLE IV
EXAMPLE OF A SEARCH OPERATION IN Z-TCAM

Steps	Activity
1	Search key = 0011

2	Sub-word$_1$ = 00
	Sub-word$_2$ = 11

3	PMA$_1$ = 0
	PMA$_2$ = 2

4	CAM priority encoder selects address 0 as MA

TABLE V
IMPLEMENTATION RESULTS OF Z-TCAM ON VIRTEX-7 FPGA

Cases (L, N)	BRAMs 36K	BRAMs 18K	FFs	LUTs	Speed (MHz)	Power (mW)
Size: 512 × 36						
Case 1 (2, 4)	32	8	593	1781	176	75.06
Case 2 (4, 4)	32	16	665	1986	165	85.14
Case 3 (2, 3)	174	18	521	1628	124	158.07
Case 4 (4, 3)	180	36	521	1666	129	168.45
Size: 64 × 32						
Case 1 (2, 4)	0	16	134	301	196	23.20
Case 2 (4, 4)	0	32	198	447	190	35.69

In Table IV, for explanation, we assume that we have also mapped layer 1.

V. Z-TCAM IMPLEMENTATION AND RESULTS

We have implemented two example designs with different design parameters (L: # of layers and N: # of vertical partitions) of Z-TCAM for sizes 512×36 and 64×32 using Verilog-HDL on Xilinx Virtex-7 (xc7v2000t-2flg1925) FPGA as the target using Xilinx 13.2 Synthesis and Implementation Tool. We have verified its functionality using different test vectors using Xilinx ISim Simulator. Implementation flow exactly follows Fig. 1. Resource utilization and maximum frequency of the example designs are given in Table V.

We have measured power consumption using Xilinx Xpower Analyzer [16]. We have generated the switching activity interchange format (SAIF) file, which is required for more accurate power estimation. Total dynamic power consumption with 1.0 v core voltage and 100 MHz operation for different cases of the design examples is tabulated in Table V. We analyze from Table V that if we increase value of L for the same value of N, then there is an increase in the memory size and power consumption. Similarly, if we decrease value of N for the same value of L, then there is also an increase in the memory size and power consumption. Thus, a smaller value of w reduces area and power consumption of Z-TCAM.

We have also synthesized and placed and routed 64×32 Z-TCAM using Synopsys tools using OSUcells library for 0.18 μm technology [17], which confirm its physical and technical feasibility. The design has been automatically placed and routed with Synopsys Astro tool, and finally a graphic database system (GDS) file has been created. The final layout and floorplan (preplacement) are shown in Fig. 3. The floorplan (preplacement) is highlighted with various memories used in layer 1. Details of the application-specific integrated circuit (ASIC) implementation are shown in Table VI. The cell/core ratio for final chip is 76.894%. The OSUcells library does not have memory IPs, and memories are mainly constructed using basic cells and Flip Flops. The current ASIC implementation has been provided to show the physical realization of the proposed methodology. The proposed design mostly consists of memory blocks (RAM); using optimized SRAM standard cell memory intellectual property (IP) the design metrics can be drastically improved in terms of area, with possible further improvement in speed and power cost.

Latency of Z-TCAM up to PMA is three clock cycles, which is higher than that of conventional CAM; however, several literatures,

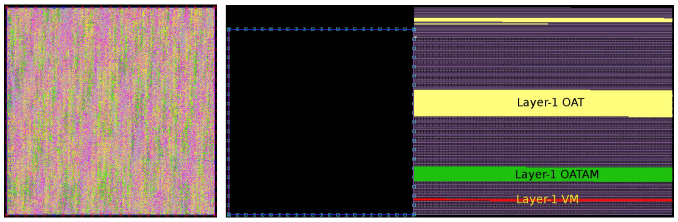

Fig. 3. ASIC implementation for 64×32 of Z-TCAM, with $L = 4$, $N = 4$. Layout (left) and (preplacement) floorplan (right).

TABLE VI
IMPLEMENTATION RESULTS FOR 64×32 Z-TCAM IN OSUCELLS LIBRARY FOR 0.18 μm TECHNOLOGY

Cases (L, N)	Case 2 (4, 4)
Speed (MHz)	223
Total Area (μm^2)	18707925
Power Consumption (W) @ 100 MHz and 1.0v	0.376

for example [12], [18], and [19], take multiple cycles for a lookup, thus making CAM latency even longer. With the inclusion of priority encoder, latency becomes four clock cycles. Latency can be easily compromised as long as throughput is achieved. The throughput of the proposed TCAM is one word comparison per clock cycle. For larger TCAM size, there may be larger values for L and N but it is expected that the throughput is not affected by larger size of TCAM. Because all layers are accessed simultaneously, latency is independent of number of layers. Key benefits of Z-TCAM over conventional TCAM are given below.

1) The proposed Z-TCAM is simpler, and easily scalable (owing to easy scalability of SRAM) for large size TCAM. The proposed one can be easily composed in ASIC or FPGA environment and the feasibility has been demonstrated successfully.
2) The proposed TCAM follows the development trends of SRAM, which are much faster than conventional TCAM. Thus, the development in FPGA and SRAM technologies will give much better values for the proposed Z-TCAM.
3) Classical TCAM uses match-line and XOR gates for comparison operations. The match-lines in classical TCAM are very capacitive and consume much time for charging and discharging. There is the physical limit in increasing speed for classical TCAM because of such intrinsic structure. However, the proposed approach mainly uses SRAM read operations for comparisons. The speed of the proposed TCAM is only limited by the read speed of SRAMs. This speed can be much higher than the speed of classical TCAM.

VI. CONCLUSION

In this brief, we have presented a novel SRAM-based TCAM architecture of Z-TCAM. We have implemented two example designs of 512×36 and 64×32 of Z-TCAM on Xilinx Virtex-7 FPGA. We have also designed 64×32 Z-TCAM in OSUcells library for 0.18 μm technology, which confirms its technical feasibility. FPGA implementation is a big plus for Z-TCAM. Resources utilization, speed, and power consumption for different situations for the example designs on FPGA as well as in ASIC have been tabulated. Z-TCAM also ensures large capacity TCAM whereas this capability is lacked by conventional ones. Moreover, the proposed TCAM has a simpler structure, and very importantly, has a deterministic search performance of one word comparison per clock cycle.

SRAM-based TCAM is a rich field of research and further investigation is necessary to find out more SRAM-based TCAMs. Our future work aims to investigate the field in depth and achieve more designs for SRAM-based TCAM.

References

[1] N. Mohan, W. Fung, D. Wright, and M. Sachdev, "Design techniques and test methodology for low-power TCAMs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 14, no. 6, pp. 573–586, Jun. 2006.

[2] P. Mahoney, Y. Savaria, G. Bois, and P. Plante, "Parallel hashing memories: An alternative to content addressable memories," in *Proc. 3rd Int. IEEE-NEWCAS Conf.*, Jun. 2005, pp. 223–226.

[3] S. Dharmapurikar, P. Krishnamurthy, and D. Taylor, "Longest prefix matching using bloom filters," *IEEE/ACM Trans. Netw.*, vol. 14, no. 2, pp. 397–409, Apr. 2006.

[4] D. E. Taylor, "Survey and taxonomy of packet classification techniques," ACM Comput. Surveys, New York, NY, USA: Tech. Rep. WUCSE-2004-24, 2004.

[5] P. Mahoney, Y. Savaria, G. Bois, and P. Plante, "Transactions on high-performance embedded architectures and compilers II," in *Performance Characterization for the Implementation of Content Addressable Memories Based on Parallel Hashing Memories*, P. Stenström, Ed. Berlin, Germany: Springer-Verlag, 2009, pp. 307–325.

[6] S. V. Kartalopoulos, "RAM-based associative content-addressable memory device, method of operation thereof and ATM communication switching system employing the same," U.S. Patent 6 097 724, Aug. 1, 2000.

[7] W. Jiang and V. Prasanna, "Scalable packet classification on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 9, pp. 1668–1680, Sep. 2012.

[8] M. Becchi and P. Crowley, "Efficient regular expression evaluation: Theory to practice," in *Proc. 4th ACM/IEEE Symp. Archit. Netw. Commun. Syst.*, Nov. 2008, pp. 50–59.

[9] Xilinx, San Jose, CA, USA. *Xilinx FPGAs* [Online]. Available: http://www.xilinx.com

[10] W. Jiang and V. K. Prasanna, "Large-scale wire-speed packet classification on FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, 2009, pp. 219–228.

[11] W. Jiang and V. Prasanna, "Parallel IP lookup using multiple SRAM-based pipelines," in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, Apr. 2008, pp. 1–14.

[12] S. Cho, J. Martin, R. Xu, M. Hammoud, and R. Melhem, "CA-RAM: A high-performance memory substrate for search-intensive applications," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2007, pp. 230–241.

[13] M. Somasundaram, "Memory and power efficient mechanism for fast table lookup," U.S. Patent 20 060 253 648, Nov. 2, 2006.

[14] 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, Dec. 26, 2006.

[15] 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.

[16] Xilinx, San Jose, CA, USA. *Xilinx Xpower Analyzer* [Online]. Available: http://www.xilinx.com

[17] OSUCells, Stillwater, OK, USA [Online]. Available: http://vlsiarch.ecen.okstate.edu

[18] S.-J. Ruan, C.-Y. Wu, and J.-Y. Hsieh, "Low power design of precomputation-based content-addressable memory," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 3, pp. 331–335, Mar. 2008.

[19] H. Noda *et al.*, "A cost-efficient high-performance dynamic TCAM with pipelined hierarchical searching and shift redundancy architecture," *IEEE J. Solid-State Circuits*, vol. 40, no. 1, pp. 245–253, Jan. 2005.