Modular Block-RAM-Based Longest-Prefix Match Ternary Content-Addressable Memories

Ameer M.S. Abdelhadi
School of Engineering Science
Simon Fraser University
Burnaby, BC Canada, V5A 1S6
aabdelha@ensc.sfu.ca

Guy G.F. Lemieux
Dept. of Electrical & Computer Engineering
The University of British Columbia
Vancouver, BC Canada V6T 1Z4
lemieux@ece.ubc.ca

Lesley Shannon
School of Engineering Science
Simon Fraser University
Burnaby, BC Canada, V5A 1S6
lshannon@ensc.sfu.ca

***Abstract*—Ternary Content Addressable Memories (TCAMs) are massively parallel search engines enabling the usage of "don't care" wildcards when searching for data. TCAMs are used in a wide variety of applications, such as routing tables for IP forwarding, which have been recently implemented using FPGAs. However, traditional "brute force" CAM architectures that use FPGA SRAM blocks (BRAMs) involve swapping address and data lines and are very inefficient. In this paper, a novel, efficient and modular technique for Longest-Prefix Match (LPM) TCAMs using FPGA BRAMs is proposed. Hierarchical search is exploited to achieve a linear storage growth and high storage efficiency. Compared to other methods, our LPM-TCAM design accommodates 5.5x more data for the same SRAM area without degrading the performance. A fully parameterized Verilog implementation is being released as an open source library.[1] The library has been extensively tested using Altera's Quartus and ModelSim.**

***Index Terms*—ternary content addressable memory, associative memory, longest-prefix match, routing table, packet forwarding**

I. Introduction

Content addressable memories (CAMs) are capable of searching the entire memory space for a specific value within a single clock cycle. As a hardware implementation of associative arrays, CAMs are massively parallel search engines. They compare the searched "pattern" against all memory content simultaneously as shown in Fig. 1. While a standard RAM returns data located at a given memory address, a CAM returns an address containing a specific given datum using a memory-wide search for that value.

CAMs can be classified into two major classes: Binary CAMs (BCAMs) and Ternary CAMs (TCAMs). While BCAMs hold binary values only, TCAMs can hold "don't care" wildcards (X's). TCAMs can be categorized into two subclasses. The general case is the Priority-Encoded TCAM (PE-TCAM), where wildcards can be placed anywhere in the written pattern. The Longest-Prefix Match TCAM (LPM-TCAM), which is the focus of of this work, is a special case of PE-TCAMs, where wildcards have to be placed as a single contiguous prefix in the written pattern.

A CAM is a high-performance implementation of the widely used associative array. Hence, it is used in almost every science field requiring high-speed processing of associative search. Yet, FPGAs lack an area-efficient soft CAM implementation. Current BCAM approaches in vendor IP libraries [1]–[3] use a brute-force approach, where SRAM data lines are swapped with address lines, to achieve a maximum of 64K entries in a modern high-density FPGA device. FPGA-based TCAM techniques are either brute-force SRAM-based or register-based [4]–[6], both of which are inefficient.

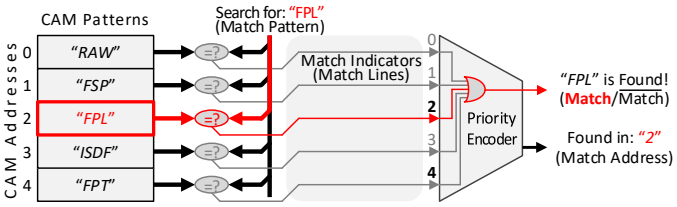

Fig. 1: CAM abstraction as a massively parallel search engine accessing all memory content to compare with the searched pattern simultaneously.

The more complex LPM-TCAMs are the building blocks of the Internet backbone Border Gateway Protocol (BGP) routers, where they are used as routing tables for IP forwarding. As of August 2014, BGP IPv4 routing tables have exceeded the 512K limit [7]. Clearly the brute-force approach cannot be used to meet this demand.

Alternatively, algorithmic approaches can be used to build a CAM. For example, they can be implemented as a linear scan by traversing the memory space sequentially, incurring a worst-case runtime of $O(n)$. An implementation using hash tables [8] distributes entries across the memory and reduces the average runtime to $O(1)$, but the worst case is still $O(n)$. Self-balancing or height-balanced Binary Search Trees (BST), *e.g.*, AVL trees and red-black trees [8], can also be used to algorithmically construct associative arrays, with a worst-case runtime of $O(\log(n))$. Algorithmic heuristics to emulate CAMs for specific applications are widely available, but often require a non-deterministic number of cycles per lookup.

Traditionally, TCAMs are built into cell-based Application-Specific Integrated Circuits (ASICs) as predesigned IP blocks. Such IPs must be specialized to each application and must use custom-designed memory cells at the transistor-level specific to each foundry. Although TCAMs can be built as IP blocks on FPGAs, it is not cost effective to build one that is flexible enough to meet the varied demands of different applications. An alternative to custom transistor-level TCAMs is needed for both FPGAs and cell-based ASICs.

This research has been funded by the National Sciences and Engineering Research Council of Canada (NSERC) Chair for Women in Science and Engineering Grant (British Columbia and Yukon) PDF Funding. This research has also been funded by the Computing Hardware for Emerging Intelligent Sensory Applications (COHESA) project. COHESA is financed under the NSERC Strategic Networks grant number NETGP485577-15.

[1]https://github.com/AmeerAbdelhadi/II-LPM-TCAM

In this paper, a novel, efficient and modular technique for constructing Longest-Prefix Match (LPM) TCAMs out of standard SRAM blocks in FPGAs is proposed. To achieve high storage efficiency, hierarchical search with data compression, previously used only to implement BCAMs [9], [10], is extended in this paper to implement LPM-TCAMs. It eliminates inefficiencies found in current SRAM-based TCAMs and brings tangible performance gains and higher storage efficiency. The same technique can also be used to design TCAMs in cell-based ASICs out of dual-ported memories.

The proposed LPM-TCAM method is scalable and fast. To build larger CAMS, the design can be cascaded in a way that exhibits a linear storage growth. It also has high storage efficiency; compared to other methods, the suggested LPM-TCAM accommodates 5.5× more data for the same SRAM area without degrading performance. For speed, one LPM pattern search can be completed every clock cycle, while pattern updates take multiple cycles. This is especially useful for IP lookup/packet-forwarding engines where fast searches are required, but slower writes are tolerated since BGP routing tables have fewer than 10k updates/second [11].

The rest of this paper is organized as follows. Section II reviews conventional block-RAM-based CAM techniques in FPGAs. Section III describes our indirectly-indexed LPM-TCAM approach. Section IV provides a discussion and comparison. Section V presents our experimental framework and results and Section VI concludes the paper with future suggestions.

II. Block-RAM-Based CAMs for FPGAs

This section reviews current FPGA CAM architectures. Existing CAM architectures in FPGAs can be classified into three categories, based on the FPGA memory resources they utilize. The first is register-based, where registers are used to store patterns and concurrently compare all register values. However, register resources are limited; a modern Intel's high-density Stratix V device [12] can only implement a 16K-entry byte-wide TCAM. The second method is the Reconfiguration-Memory-Based CAM (RCAM), where LUT configuration memory is utilized. This method can implement a 128K-entry single-byte CAM on the same Stratix V device. Finally, SRAM blocks are utilized in a brute-force approach to store locations for each pattern. The same Intel Stratix V device can implement a 64K-entry single-byte CAM. Our work explores using SRAM blocks with a more efficient hierarchical search.

Subsection A reviews previous work using reconfiguration memory to create CAMs. Next, we summarize traditional brute-force approach in Subsection B and FPGA vendor CAM support in Subsection C. Finally, we explain hierarchical search and indirectly-indexed CAMs in Subsection D.

A. Reconfiguration Memory Based CAMs (RCAMs)

FPGA configuration memory is an SRAM chain loaded with the configuration bit-stream and is used to configure the device. A modern FPGA device accommodates several Mbits of configuration SRAM cells, for instance, Intel's Stratix V device contains 22Mbits for LUT configuration [12]. The

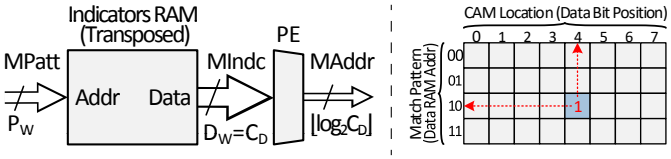

Fig. 2: (left) CAM as a Transposed-RAM. (right) Example of a pattern indicator for pattern '10' in address 4.

SRAM reconfiguration memory in FPGAs can be utilized as a wide and shallow memory to generate RCAMs [13], [14].

RCAMs are impractical in many cases; they require multiple-cycles to write the bit-stream and update their content, exhaust logic resources, and are not portable across different devices. Alternatively, Intel's Stratix devices provide accessibility to LUT configuration memory as SRAM blocks called MLABs [12]. MLABs can be used to create CAMs in a similar method as the brute-force approach we describe next.

B. Brute-force CAMs via Transposed Indicators RAM

As depicted in Fig. 2, a BRAM is addressed by the match pattern while each bit of the data indicates the existence of the pattern. The data bit position corresponds to the BCAM address location. Thus, the CAM depth, d_c, must match the RAM width, w_r. Also, the pattern width of must match the address width of the RAM, *i.e.*, $w_p = \lceil \log_2 d_r \rceil$.

In this paper, we call this structure a Transposed Indicators RAM (TIRAM) and describe it as a matrix of indicators

$$TIRAM = \begin{bmatrix} I_{0,0} & I_{0,1} & \cdots & I_{0,d_c-1} \\ I_{1,0} & I_{1,1} & \cdots & I_{1,d_c-1} \\ \vdots & \vdots & \ddots & \vdots \\ I_{|P|-1,0} & I_{|P|-1,1} & \cdots & I_{|P|-1,d_c-1} \end{bmatrix} \quad (1)$$
$$\forall a \in A, p \in P : I_{p,a} = (RAM\,[a] \quad EQ \quad p)\,,$$

where A is the address space set and P is the pattern set.

This BRAM-based brute-force approach is adopted by Xilinx [1], [2] and Intel [3] to create soft CAMs as described in their application notes. Zerbini and Finochietto [4] apply the cascaded brute-force approach to emulate TCAMs for packet classification; however, updating the TCAM content is not discussed. Jiang [5] also uses the brute-force approach to emulate TCAMs. However, a pattern update requires a sequential rewriting of all RAM addresses for each pattern. Ullah *et al* [6] also use the brute-force approach, but their TCAMs can be partitioned. This allows the search of specific TCAM fragments, but the required storage is still similar to the brute-force approach. Furthermore, rewriting the TCAM requires a serial rewriting of all RAM locations.

The Cascaded Brute-Force Approach: SRAM cell usage for the brute-force approach is exponential to pattern width w_p; making a wide pattern width is therefore infeasible. CAM cascading relaxes SRAM growth from exponential to linear. The CAM pattern and prefix are divided into smaller pattern segments; each segment is associated with a separate CAM. A pattern is located in the CAM if all its segments are found in the same address of all segmented CAMs. CAM cascading is used by Xilinx [1], [2] and Intel [3] to create soft scalable CAMs described in their application notes.

C. Vendor Support of CAMs

FPGA support for CAMs has taken two approaches: direct support in hard IP, implemented in some legacy FPGAs, and as pure soft IP, the modern approach.

For hard IP approaches, Intel's legacy FLEX, Mercury and APEX [15] device families integrated intrinsic partial CAM support into their memory blocks. This required additional transistors in each memory cell which could be used to build a small 32×32 BCAM. These BCAM blocks can be used in parallel to increase the address space, and can be cascaded as described in the previous subsection to increase pattern width. In addition, Lattice ispXPLD devices [16] have integrated support for CAMs via their Multi-Function Blocks (MFBs) which can be configured into 128×48 Ternary CAM block. All of these FPGAs have been discontinued. No modern FPGAs have any hard IP for CAM support.

Xilinx does not use hard IP, and instead relies upon soft IP with the brute-force approach for creating CAMs [1], [2]. Alternatively, Microsemi recommends registers for a single-cycle CAM, or a multi-cycle search of BRAMs in parallel [17].

D. Hierarchical Search BCAMs (HS-BCAMs)

Hierarchical search BCAMs [9] efficiently reduce search space by dividing the CAM into equal-size pattern sets. HS-BCAMs are composed of two RAM hierarchies; the first stores hit/miss indicators for each set, whereas the second stores the sets. A lookup operation consists of finding a set with a match from the first RAM, then fetching the matching set from the second RAM, then searching its entries in parallel for a match. Thus, HS-BCAMs group addresses into sets and maintain one pattern match indicator for each set.

HS-BCAMs are highly efficient with narrow patterns. However, the lack of pattern match indicator for each address prevents cascading, causing exponential memory growth as pattern width increases. This limitation is crucial as the vast majority of applications require wide patterns.

HS-BCAMs cannot be cascaded. While cascadable CAMs require indicators from every address at every stage, this requirement can be alleviated if the CAM will not be cascaded. Instead of storing match indicators for each address separately, as in the brute-force approach, an indicator is generated for a set of w_s addresses, indicating whether the pattern exists at any of these addresses in the set.

An address set is a set of successive w_s addresses. A d_c-entry BCAM is divided into $\lceil d_c/w_s \rceil$ sets. A set indicator $I_{p,s}$ indicates if any of the addresses in set s, *i.e.*, addresses $w_s \cdot s$ upto $w_s \cdot (s+1) - 1$, contains the pattern p, namely,

$$\forall s \in S, p \in P : I_{p,s} = \bigvee_{a=w_s \cdot s}^{a=w_s \cdot (s+1)-1} (RAM\,[a] == p)\,. \quad (2)$$

where S is the set of all address sets. Set indicators are stored in a Set Transposed Indicators RAM (STIRAM) as follows:

$$STIRAM = \begin{bmatrix} I_{0,0} & I_{0,1} & \cdots & I_{0,|S|-1} \\ I_{1,0} & I_{1,1} & \cdots & I_{1,|S|-1} \\ \vdots & \vdots & \ddots & \vdots \\ I_{|P|-1,0} & I_{|P|-1,1} & \cdots & I_{|P|-1,|S|-1} \end{bmatrix}. \quad (3)$$

Indirectly Indexed Hierarchical Search (II-HS) BCAM:

Similar to HS-BCAMs, the II-HS-BCAM approach [10] arranges memory addresses into sets. However, the lack of a pattern match indicator for each address prevents HS-BCAM cascading, causing exponential memory growth as pattern width increases.

Instead, the II-HS CAM regenerates match indicators for every single address by storing indirect indices for address match indicators. Hence, II-HS-BCAMs can be cascaded and the exponential RAM growth becomes linear as pattern width increases. This method exploits the sparsity of the transposed indicators RAM to store indices to subsets of the match indicators. It relies upon specific properties of BCAMs that allow a bounded compression of BCAMs match indicators.

III. The Proposed Indirectly-Indexed Longest-Prefix Match TCAM (II-LPM-TCAM)

We now present the proposed II-LPM-TCAM. Subsection A motivates and explains the key idea for this work. We describe the design method in Subsection B. Subsection C discusses a device-specific instance for Intel's Stratix device family.

A. Motivation and Key Idea

In this paper, we prove that the same bounded compression applied to HS-BCAMs also applies to LPM-TCAMs, enabling a significant increase in storage efficiency compared to brute-force methods. The following theorem is the basis of the compression technique. This theorem employs properties of pattern prefixes in LPM-TCAMs to show that TCAMs' match indicators can be compressed into a smaller subset of indicators and thereby stored in smaller distributed memories.

Definition III.1 (Pattern sets)**.** The set of all possible binary patterns of length w_p is denoted by P_{w_p}, where the cardinality of P_{w_p} is $|P_{w_p}| = 2^{w_p}$. For simplicity, we denote the set of all possible patterns of a specific length merely as P. The following defines two types of pattern subsets. One, P^a is a subset of P that includes all patterns in address a, thus

$$P^a = \left\{ \tilde{p} \in P \;\middle|\; I_{\tilde{p},a} = 1 \right\}. \quad (4)$$

Two, $P^{p,l}$ is a subset of P that includes all patterns matching the leftmost l bits of pattern p (a prefix length of l), thus

$$P^{p,l} = \left\{ \tilde{p} \in P \;\middle|\; \tilde{p}\langle w_p{-}1 : w_p{-}l \rangle = p\langle w_p{-}1 : w_p{-}l \rangle \right\} \quad (5)$$

where the angle brackets are used for bit selection. The cardinality of $P^{p,l}$ is $|P^{p,l}| = 2^{w_p - l}$ since only l bits (the prefix) are constant, whereas all the other $w_p - l$ bits are X's.

Examples. $P_3 = \{111, 110, 101, 100, 011, 010, 001, 000\}$, $P^{\underline{10}1,2} = \{\underline{10}1, \underline{10}0\}$, $P^{\underline{0}11,1} = \{\underline{0}11, \underline{0}10, \underline{0}01, \underline{0}00\}$.

Lemma III.1 (The relation between patterns in LPM-TCAM addresses)**.** *The patterns included in two LPM-TCAM addresses are either a subset or a disjoint. Given two addresses a_i and a_j, and their included pattern sets P^{a_i} and P^{a_j}, respectively, the relation between P^{a_i} and P^{a_j} is (i) P^{a_i} is a subset of P^{a_j},* i.e.*, $P^{a_i} \subseteq P^{a_j}$, (ii) P^{a_j} is a subset of P^{a_i},* i.e.*, $P^{a_j} \subseteq P^{a_i}$, or (iii) P^{a_i} and P^{a_j} are disjoint,* i.e.*, $P^{a_i} \cap P^{a_j} = \emptyset$.*

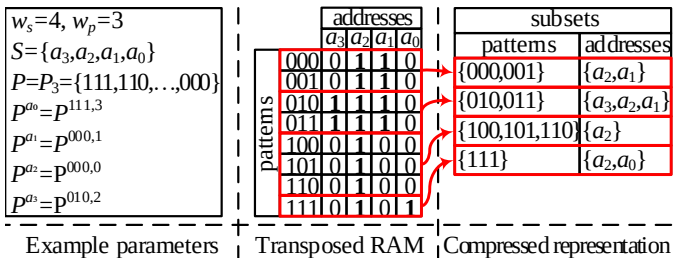

Fig. 3: A toy example demonstrating Theorem III.2. The compressed representation requires only $w_s = 4$ entries.

Proof. Let $P^{p_i,l_i} = P^{a_i}$, $P^{p_j,l_j} = P^{a_j}$, and without loss of generality $l_i > l_j$. If $p_i\langle l_j - 1 : 0\rangle = p_j\langle l_j - 1 : 0\rangle$, P^{a_i} will be a subset of P^{a_j}, namely, $P^{a_i} \subseteq P^{a_j}$, otherwise they will be disjoint, namely $P^{a_i} \cap P^{a_j} = \emptyset$ ■

Theorem III.2 (A bound on the number of disjoint pattern subsets, where each pattern subset is located in a different address subset)**.** *Suppose S is a set of w_s addresses. It is possible to cluster all patterns located in S into at most w_s disjoint pattern subsets, whereas all patterns in the same pattern subset are located in the same address subset of S.*

Example. A toy example is provided in Fig. 3 where patterns are clustered into $w_s = 4$ disjoint subset such that each pattern subset is located in the same address subset (Fig. 3 (right)).

Proof. We prove by weak induction on the number of addresses w_s in a set of address S that at most w_s disjoint sets of patterns will be located in different address subsets of S.

Base case. This is the trivial case where $w_s = 1$, S includes a single address, say a. All patterns that are located in a form a single set P^a. Thus the base case holds for $w_s = 1$.

Induction step. Suppose that this theorem holds for $w_s = k$, *i.e.*, patterns can be clustered into at most $w_s = k$ disjoint sets, whereas all patterns in the same pattern set are located in the same address subset of S, and S is a set of $w_s = k$ addresses. The same property should be proven for $w_s = k + 1$. In other words, we should prove that the addition of an address to S, say a_{k+1}, with its corresponding pattern set $P^{a_{k+1}}$, will add at most a single disjoint pattern subset.

Lemma III.1 shows that the relation of $P^{a_{k+1}}$ and any other address's (say a_i) pattern set P^{a_i} is one of three options as follows. (i) $P^{a_{k+1}}$ is a subset of P^{a_i}, *i.e.*, $P^{a_{k+1}} \subseteq P^{a_i}$, (ii) P^{a_i} is a subset of $P^{a_{k+1}}$, *i.e.*, $P^{a_i} \subseteq P^{a_{k+1}}$, or (iii) $P^{a_{k+1}}$ and P^{a_i} are disjoint, *i.e.*, $P^{a_{k+1}} \cap P^{a_i} = \emptyset$. (See Fig. 4.)

In case (i) and case (iii), a new address subset may be generated to include a_{k+1}. In case (ii), a_{k+1} will be added to all address subsets where a_i is included, hence no additional address subset will be added. However, a new address subset may be generated to include a_{k+1}, where a_i is not included.

To summarize, any of these three cases may introduce one address subset at most, hence the addition of address a^{k+1} will add a single address subset at most. ■

While each pattern can be located in a subset of S, Theorem III.2 signifies that there are at most w_s subsets of S, given that S is a set of w_s addresses. Thus, the key idea is to store these

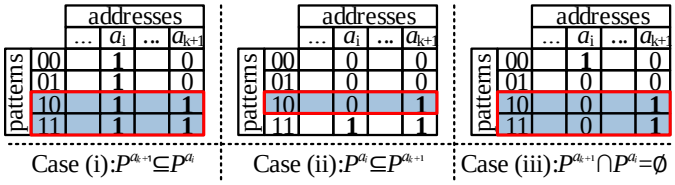

Fig. 4: An example of a transposed RAM showing the three cases of pattern set relationship. Subsets generated due to the addition of a_{k+1} are highlighted.

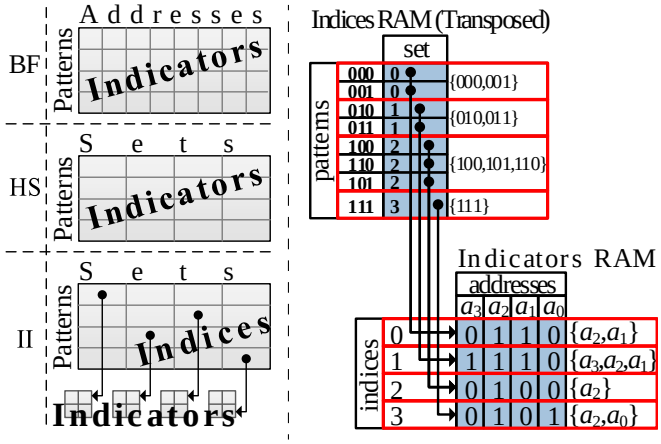

Fig. 5: (left) Indicators arrangement for three different approaches. (right) The toy example from Fig. 3. arranged as Indirectly Indexed (II) tables.

w_s subsets into smaller distributed memories, and only keep indices to addresses in the distributed memories. We call this concept "Indirectly-Indexed."

The significance of Theorem III.2 lies in the measurement it provides for the STIRAM matrix sparsity. Namely, it provides an upper bound of the number of binary '1' (matches) for a set (a column in STIRAM). Instead of storing match indicators for every address and pattern pair as in the brute-force approach (Fig. 5 (upper left)), or set indicators as in the hierarchical search approach (Fig. 5 (middle left)), we store all address indicators only for sets with a match, as they are limited to w_s. To reduce memory consumption, address match indicators are saved in another auxiliary structure, whereas the original STIRAM will hold indices to the auxiliary structure (Fig. 5 (lower left)). This mechanism is demonstrated in Fig. 5 (right) using the toy example from Fig. 3.

B. Design and Functionality of II-LPM-BCAM

As depicted in Fig. 6, a single stage of the proposed II-HS-TCAM consists of four parts. First, the *SetRAM*, where the patterns and their corresponding prefix length (as a mask) are stored in sets of w_s words. The same structure includes the mask and compare logic, where all patterns in the current set are masked and compared in parallel with an input pattern. The main output of this structure is *SetIndc*, a vector containing a match indicator for each pattern in the current set. The second stage is the *Indirectly-Indexed Transposed RAM (IITRAM)*, where patterns are used as addresses for the *IndcRAM*. This table is a Set-Transposed Indicators RAM, where sets of addresses share indices to a smaller structure of distributed memories (MLABs, or LUTRAMs), enabling the compression mechanism. Its primary output are the match indicators for

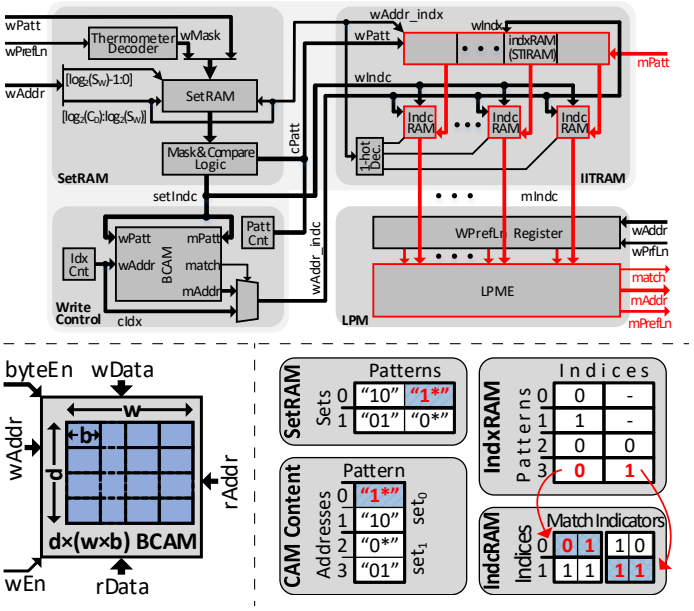

Fig. 6: (top) II-LPM-TCAM single-stage architecture. The match datapath is highlighted. (bottom-left) Dual-port block-RAM connectivity used throughout the paper. (bottom-right) Toy example of II-TCAM tables with $d_c = 4, w_p = 2, w_s = 2$. Writing "1*" and matching "3" are highlighted.

each single address. This subsystem is the most memory-consuming structure. The third stage is the Longest-Prefix Match, where match indicators from the *IITRAM* are received. Lengths of pattern prefixes are stored in one register. The longest of these prefixes with a positive match indicator is reported by the Longest-Prefix Match Encoder (LPME). The fourth stage, the write control unit, generates control signals to control the entire structure, feeds the IITRAM with indicators and indices, and controls the writing sequence. The write control unit has two counters to iterate over all patterns and all indices. A small BCAM is used to control the compression mechanism and search for Set Indicators (SetIndc) that have already been stored in the distributed memories. The following is a detailed description of each of these four stages.

(1) *Indirectly-Indexed Transposed RAM (IITRAM) unit*: As described in the previous subsection, instead of storing set match indicators in the STIRAM, we store only indirect indices to an auxiliary distributed RAM, which holds the match indictors for all the addresses in the set, called the *Indicators RAM (IndcRAM)*. Theorem III.2 shows that a maximum of w_s different patterns can match in a set; hence, the depth of IndcRAM is set to w_s. Each set has w_s addresses, therefore IndcRAM should have w_s address indicators for each set and its width is also w_s. The address space consists of $\lceil d_c/w_s \rceil$ sets; thus, $\lceil d_c/w_s \rceil$ IndcRAM $w_s \times w_s$ blocks are required.

The IndxRAM (based on STIRAM) holds indices for all pattern and set pairs. To represent all patterns the required depth is 2^{w_p}. For each of the $\lceil d_c/w_s \rceil$ sets, $\lceil \log_2 w_s \rceil$ bits are required for each index, in addition to one valid bit. In total, the IndxRAM width is $\lceil d_c/w_s \rceil \cdot (\lceil \log_2 w_s \rceil + 1)$ bits. The IITRAM structure is described in detail in Fig. 7.

Pattern matching mechanism (TCAM search): The pattern match operation activates the IITRAM and the LPME

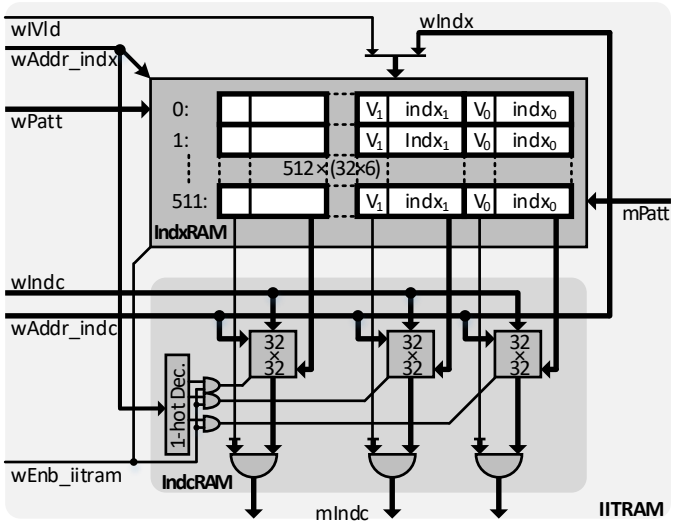

Fig. 7: Indirectly-Indexed Transposed RAM (IITRAM) unit.

structures only. As depicted in Fig. 7, the matched pattern $mPatt$ is used to address IndxRAM, where a complete set (line) of indices and their corresponding valid bits are read. Each index from IndxRAM is used to address a separate IndcRAM. The outputs of the IndcRAM blocks are then masked with the corresponding valid bit to generate the final match indicators (match lines) which feed the LPM stage. The II-TCAM is capable of performing one match operation every cycle. However, the total latency of the TCAM lookup includes the latency of the pipelined LPME, one cycle to access IndxRAM, and another cycle to access IndcRAM. Thus, $throughput = 1$ and $latency = 2 + latency(LPME)$. The detailed match operation is described in Algorithm 1.

(2) *Sets RAM (SetRAM) Unit*: Similar to the hierarchical search method, the SetRAM unit holds all patterns and their corresponding masks, and a valid bit for each address. Pattern masks are derived from the pattern's prefix length using a thermometer decoder. Patterns, masks and valid bits for each set of w_s addresses are packed into a single SetRAM line. Thus, its size is $\lceil d_c/w_s \rceil \times (w_s(2w_p + 1))$.

Once a new pattern is written to SetRAM, we read the complete set where the same pattern was written, and re-evaluate the set indicators (SetIndc). SetIndc's are re-evaluated based on a counted pattern (cPatt) that iterated over all possible patterns. If cPatt's prefix and the pattern read from SetRAM are equal and the data is valid, the corresponding bit of SetIndc

Algorithm 1: Matching $mPatt$ in a single-stage II-TCAM

input : $mPatt$: a pattern to match
output: $mIndc$: match indicators (match lines)

$\{V_{w_s-1}, indx_{w_s-1}, \cdots, V_0, indx_0\} \leftarrow IndxRAM[mPatt]$
for $i = 0$ **to** $w_s - 1$ **do**
  if ($V_i =$ '1') **then**
    $mIndc\langle w_s(i+1)-1 : w_s i\rangle = IndcRAM_i[indx_i]$
  else
    $mIndc\langle w_s(i+1)-1 : w_s i\rangle = 0$
return $mIndc$

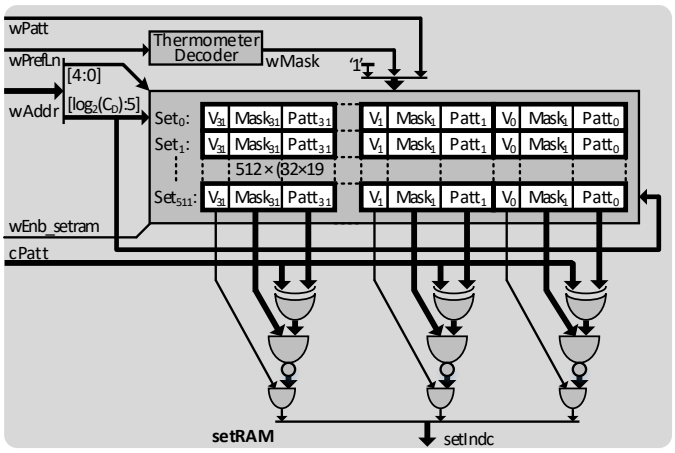

Fig. 8: Sets RAM (SetRAM) unit.

will be set. The SetRAM subsystem is depicted in Fig. 8.

(3) *Control Unit*: The write control unit is activated on a write request, where it updates the entire data structure based on the new write operation. Since a TCAM write operation may place a pattern in all memory addresses (*e.g.*, when the prefix is 0), rewriting memory lines may be required. Therefore, a write operation requires $d_{r,min}$ cycles at most, where $d_{r,min}$ is the depth of the shallowest BRAM configuration.

The write control unit has two counters. (1) Patterns counter (cPatt), used to iterate over all possible patterns, addresses and updates the IndxRAM accordingly. (2) Indices counter (cIndx) is used to assigned a different index for each different indicators in the IndcRAM. Theorem III.2 assures that at most w_s indices will be used. cIndx is used in conjunction with a BCAM to assure that indicators that have been stored will not be associated with a new index.

The write control unit employs an FSM to control the whole structure. The write control unit is depicted in Fig. 9. The write control unit's writing mechanism is described in detail below.

Pattern writing mechanism (TCAM update): The pattern writing mechanism is described in Algorithm 2. The written address is sliced into two portions. (1) The lower portion up to $\log_2 w_s - 1$ bits of the address is used as a byte-enable for intra-set addressing. (2) The higher portion includes the other bits and is used for set addressing. The prefix length is converted into a thermometer mask to accelerate the parallel comparison afterward.

The written pattern and mask are written into SetRAM, then the entire updated set is read and compared with cPatt,

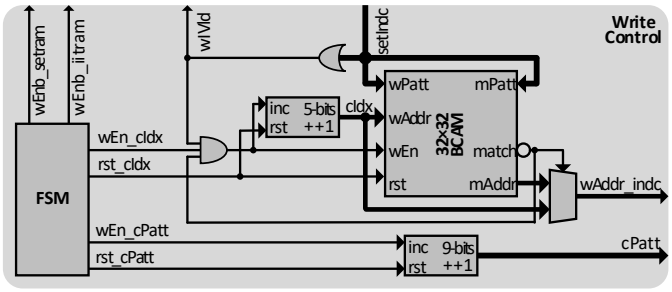

Fig. 9: Write controller, with pattern and index counters, FSM, and BCAM.

Algorithm 2: Writing a pattern $wPatt$ with a prefix length $wPerfLn$ in address a of a single-stage of the II-TCAM

input : $wPatt$: a pattern to write
$wPrefLn$: The written prefix length
$wAddr$: a writing location of $wPatt$

$wAddr_{low} \leftarrow wAddr\,\langle log_2 w_s - 1 : 0\rangle$
$wAddr_{hight} \leftarrow wAddr\,\langle log_2 d_c : log_2 w_s\rangle$
$wMask \leftarrow Thermometer(wPrefLn)$
$SetRAM[wAddr] \leftarrow \{V = \text{'1'}, wMask, WPatt\}$
for $i = 0$ **to** $w_s - 1$ **do**
  $\{V_i, mask_i, patt_i\} \leftarrow SetRAM[wAddr_{high}, i]$
for $cPatt = 0$ **to** $d_{r,min}$ **do**
  for $i = 0$ **to** $w_s - 1$ **do**
    $SetIndc\,\langle i\rangle \leftarrow V_i \& \parallel \overline{(mask_i \& (cPatt \oplus Patt_i))}$
  $wVld \leftarrow OR(SetIndc)$ (SetIndc is not all zeros)
  $cIndx \leftarrow 0$
  for $j = 0$ **to** $w_s - 1$ **do**
    if *(SetIndc in BCAM)* **then**
      $wAddr\_indc \leftarrow$ 'SetIndc Location in BCAM'
    else
      $wAddr\_indc \leftarrow cIndx$
      if $wVld =$ '1' **then**
        $cIndx++$
        BCAM$[cIndx] \leftarrow SetIndc$
  $IndxRAM[cPatt, wAddr_{High}] \leftarrow wAddr\_indc$
  $IndcRAM_{wAddr_{High}}[wAddr\_indc] \leftarrow SetIndc$

a pattern counter that iterates over all patterns in range. If the masked portion of cPatt and the SetRAM pattern are equal and valid, the set indicator (SetIndc) for this specific set is enabled. If the SetIndc is not found in the logic control BCAM, it will be written there with a unique address cIndx. cIndx will increase each time SetIndc is not found in the BCAM. IndxRAM is a transposed memory, hence it will be addressed with cPatt. A specific set is chosen from IndxRAM using the higher portion of the address as a byte-enable. wAddr_indc is used to index IndcRAM, it is also written to the IndxRAM. wAddr_indc is read from the BCAM if the appropriate SetIndc is found in the BCAM, otherwise the cIndx will be used.

(4) *Longest-Prefix Match Encoders (LPME)* Given a match indicator and a prefix length for each location, an LPME finds the location of the longest prefix match. An LPME receives n match indicators ($mIndc_i : 0 \leq i < n$), and n prefix length words ($PrefLn_i : 0 \leq i < n$). The LPME reports if there is a match ($match = \parallel mIndc$), the match location $mAddr$, and the prefix of the match $mPrefLn$. The following formalizes the encoding operation.

$$mAddr = i \left| \begin{array}{l} mIndc_i = 1,\ and \\ PrefLn_i > PrefLn_j \ \forall \ \begin{array}{l} j \neq i \mid \\ match_j = 1 \end{array} \end{array} \right. \quad (6)$$

Fig. 10 illustrates a recursive implementation of an LPME. While Priority Encoders (PEs) steer the index of the first match, LPMEs steer the index of the first largest matching

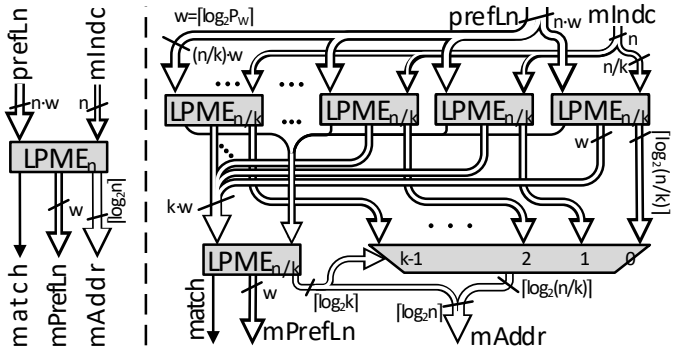

Fig. 10: Recursive LPME: (left) Symbol, and (right) recursive definition

prefix. Compared to PEs, LPMEs have additional logic area overhead due to comparators used to find longest prefixes. Furthermore, pattern prefixes are stored in $\lceil \log_2 w_p \rceil \cdot d_c$ registers to be able to fetch them in parallel.

C. Implementation in Intel Stratix V Devices

An area-efficient implementation of the proposed II-TCAM requires both large and small RAMs on the FPGA. Relatively small RAMs will be used to construct IndcRAM and store match indicators. A perfect candidate is the LUT configuration RAM, known as LUTRAM, and is supported by both Intel and Xilinx devices. In addition, the larger RAMs will be used to construct other wide structures.

Intel's Stratix V memory architecture provides a 640-bit LUTRAM memory called MLAB (Memory Logic Array Block) as well as 20Kb BRAMs called M20Ks.

Both MLABs and M20Ks are used in their shallowest/widest configuration modes. Hence, the MLABs are 32×20, and the M20Ks are 512×40. Since the depth of MLABs is 32, and they are used to implement the IndcRAM, we set $w_s = 32$. With an index width of $\lceil \log_2 w_s \rceil = 5$, a single M20K line can store up to 8 indices. M20K's mixed-width port is used to write a single 5-bit index.

IV. Comparison and Discussion

The CAM storage efficiency μ_s is defined as the SRAM cell utilization for a specific CAM implementation. In other words, μ_s is the ratio of the total CAM bits realized to the total SRAM cells used to implement the CAM. μ_s for the brute-force TCAM is estimated as follows:

$$\mu_s^{-1}(BF) \approx \begin{cases} 1 + 2^{w_p}/w_p & uncascaded \\ 1 + d_{r,min}/\log_2(d_{r,min}) & cascaded. \end{cases} \quad (7)$$

Equation (7) shows that the storage efficiency of the uncascaded BF implementation is inversely proportional to the exponent of w_p and decays rapidly as w_p increases. The cascaded BF is not related to w_p, hence the efficiency is not affected when w_p increases. However, the efficiency is affected by $d_{r,min}$, an intrinsic BRAM characteristic.

As shown in Fig. 11 (top), the efficiency is inversely proportional to $d_{r,min}$. Hence, shallow and wide BRAMs exhibit higher storage efficiency. Using the area information

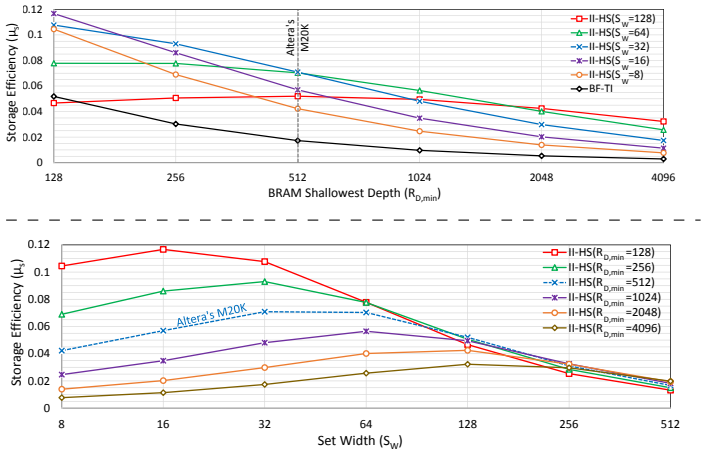

Fig. 11: Storage Efficiency μ_s as function of (top) BRAM shallowest depth ($d_{r,min}$), and (bottom) set width (w_s) for HS-CAM and the cascaded BF.

from Section III-B, the storage efficiency μ_s of the proposed II-TCAM can derived as follows:

$$\mu_s^{-1}(II) \approx 1 + \frac{1 + w_s + \log_2 w_s \cdot \left(1 + \frac{d_{r,min}}{w_s}\right)}{\log_2 d_{r,min}}. \quad (8)$$

Equation (8) shows that the efficiency of our II-TCAM approach is not related to the pattern width w_p. Also, it can be adjusted by tuning the set width w_s. The set width w_s is restricted by internal RAM parameters, namely the depth of the LUTRAM and the BRAM allowable write-port width in mixed-width mode. As described in Section III-C, Intel's Stratix V MLAB minimal depth is 32, and $\log_2 32 = 5$ is allowed as write data width for M20K mixed-width configuration. Hence, w_s=32 is the suitable set width. This restriction is in agreement with Fig. 11 (bottom) where the storage efficiency μ_s is given as function of the set width w_s. Fig. 11 (bottom) shows that the storage efficiency decreases for narrow sets since the STIRAM portion of the TCAM is not efficiently compressed. Conversely, storage efficiency decreases for wide sets due to the increase of the IndcRAM portion.

V. Experimental Results

To verify and simulate our proposed TCAM and compare it to earlier techniques, fully parameterized Verilog modules have been developed for register-based, both cascaded and uncascaded Brute Force Transposed Indicator (BF-TI), and the proposed II-TCAM methods. A run-in-batch flow manager was developed to simulate and synthesize these designs with various parameters using Intel's ModelSim and Quartus II. The Verilog modules and the flow manager are available online.

To verify correctness, the proposed architecture is simulated using Intel's ModelSim. A large variety of different TCAM architectures and parameters, *e.g.*, TCAM depth and pattern width, are swept and simulated in batch, each with over a million random cycles. All different TCAM design modules are synthesized using Quartus II targetting a Stratix V `5SGXMABN1F45C2`. This '-2' speed grade device has 360k ALMs and 2,640 M20K BRAMs. Half of the ALMs can be used as MLABs, where eache MLAB replaces 10 ALMs.

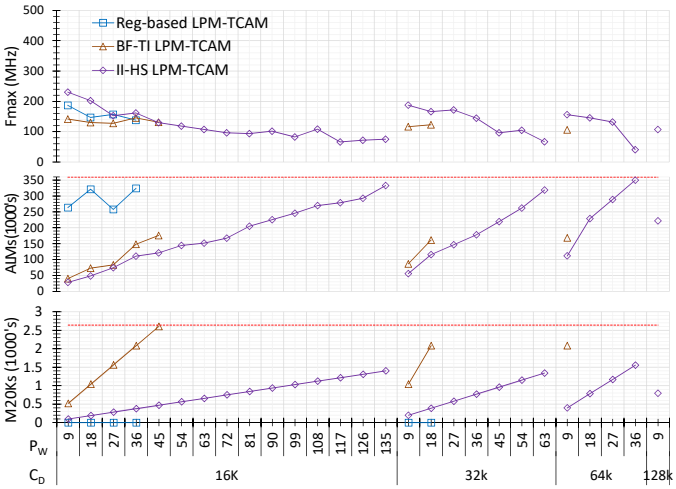

Fig. 12: Results for several TCAM depth and pattern width sweeps (bottom) M20K count (middle) ALMs count (top) F_{max} at T=0°C.

Fig. 12 plots feasible sweeps of TCAM depth and pattern width. With 16K entries, the proposed II-TCAM can implement a pattern width of 135b, while other BF-based approaches cannot exceed 45b. The number of M20K BRAMs is plotted in Fig. 12 (bottom). The BF-TI and II-TCAM exhibit a linear growth of M20K consumption as pattern width increases since both methods are cascaded. However, the II-TCAM growth rate is lower since addresses are grouped as sets. The inverse of the curve slope represents the storage efficiency μ_s. A linear regression shows that $\mu_s(BF) = 1.4\%$, whereas $\mu_s(II) = 7.7\%$. This is with agreement with (7) and (8), and shows 5.5× improvement in storage efficiency. The efficiency of the BF-CAM in Fig. 12 matches the reported CAM instances by Xilinx [1] and Intel [3].

As shown in Fig. 12 (middle), the proposed II-TCAM method and the BF-based methods also exhibit linear ALM count growth as pattern width increases. The proposed II-TCAM approach uses ALMs as MLABs, hence it has higher ALM consumption. The register-based TCAM consumes the most ALMs due to massive register usage.

Fig. 12 (top) plots the F_{max} of all TCAM architectures. The II-TCAM exhibits the highest F_{max} for most configurations compared to register-based and BF-based TCAMs. However, F_{max} drops dramatically as the CAM depth increases since the LPME delay becomes dominant. As can be seen, F_{max} decreases mildly as pattern width increases due to cascading.

Similar to BF-based approaches, the II-TCAM matches a pattern every cycle and requires multiple cycles to write a pattern. Pipelining is employed to increase F_{max} and this adversely increases latency. The longest combinational path goes through the LPME, and is pipelined every stage. For the device-specific LPME described in Section III-B, $\lceil \log_4 \lceil d_c/w_s \rceil \rceil$ pipe stages are required. On the other hand, sets are not used in BF-based approaches, hence its match latency is $\lceil \log_4 d_c \rceil$.

VI. Conclusions and Future Work

In this paper, a novel modular LPM-TCAM architecture for FPGAs is proposed. The approach employs hierarchical search to reduce BRAM consumption. While hierarchical search methods have been previously employed to BCAMs, we show how to compress TCAM tables using hierarchical search and provide a mathematical proof of correctness.

Our approach efficiently regenerates a match indicator for every address by storing indirect indices to address indicators. Hence, the proposed method can be cascaded, resulting in a linear growth rate instead of exponential. Compared to other methods, our LPM-TCAM is capable of storing 5.5x more data for the same SRAM without degrading the performance. We also perform a detailed comparison for different FPGA-based LPM-TCAM architectures. A fully parametrized open source Verilog implementation of the suggested methods is provided.

As future work, the implementation would benefit by applying power reduction techniques. Hierarchical search can be power efficient since it performs the search in steps; the implementation needs to selectively disable inactive memories to realize a power savings.

References

[1] K. Locke, "Parameterizable Content-Addressable Memory," San Jose, CA, USA, 2011, xilinx Inc. Application Note XAPP1151.

[2] J. Brelet, "Methods for implementing CAM functions using dual-port RAM," Mar. 5 2002, US Patent 6,353,332.

[3] Altera, "Implementing High-Speed Search Applications with Altera CAM," San Jose, CA, USA, July 2001, Applications Note 119, V2.1.

[4] C. A. Zerbini and J. M. Finochietto, "Performance evaluation of packet classification on FPGA-based TCAM emulation architectures," in *IEEE Global Communications Conf.*, Dec 2012, pp. 2766–2771.

[5] W. Jiang, "Scalable ternary content addressable memory implementation using FPGAs," in *ACM/IEEE Symp. on Architectures for Networking and Communications Systems*, Oct 2013, pp. 71–82.

[6] Z. Ullah, M. K. Jaiswal, and R. C. C. 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, Feb 2015.

[7] (2014) CIDR Report. [Online]. Available: http://www.cidr-report.org

[8] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[9] A. Abdelhadi and G. Lemieux, "Deep and narrow binary content-addressable memories using FPGA-based BRAMs," in *Int. Conf on Field-Programmable Technology*, Dec 2014, pp. 318–321.

[10] ——, "Modular SRAM-Based Binary Content-Addressable Memories," in *IEEE Int. Symp. on Field-Programmable Custom Computing Machines (FCCM)*, May 2015, pp. 207–214.

[11] (2014) The BGP Instability Report. [Online]. Available: http://bgpupdates.potaroo.net/instability/bgpupd.html

[12] Altera, *Stratix V Device Handbook*, San Jose, CA, USA, May 2013.

[13] S. Guccione, D. Levi, and D. Downs, "A Reconfigurable Content Addressable Memory," in *Proceedings of the 15 IPDPS Workshops on Parallel and Distributed Processing*, London, UK, 2000, pp. 882–889.

[14] A. Jamadarakhani and S. K. Ranchi, "Implementation and Design of High Speed FPGA-based Content Addressable Memory," *Int. J. for Sci. Research and Development*, vol. 1, no. 9, pp. 1835–1842, Dec 2013.

[15] Altera, *APEX 20K Programmable Logic Device Family Data Sheet*, San Jose, CA, USA, March 2004, version 5.1.

[16] Lattice Semi, "Content Addressable Memory (CAM) Applications for ispXPLD Devices," Hillsboro, OR, 2002, application Note AN8071.

[17] Actel, "Content-Addressable Memory (CAM) in Actel Devices," Mountain View, CA, USA, 2002, application Note AC194.