# Modular SRAM-based Binary Content-Addressable Memories

Ameer M.S. Abdelhadi and Guy G.F. Lemieux
Department of Electrical and Computer Engineering
University of British Columbia
Vancouver, B.C., V6T 1Z4, Canada
{ameer,lemieux}@ece.ubc.ca

***Abstract*—Binary Content Addressable Memories (BCAMs), also known as associative memories, are hardware-based search engines. BCAMs employ a massively parallel exhaustive search of the entire memory space, and are capable of matching a specific data within a single cycle. Networking, memory management, pattern matching, data compression, DSP, and other applications utilize CAMs as single-cycle associative search accelerators. Due to the increasing amount of processed information, modern BCAM applications demand a deep searching space. However, traditional BCAM approaches in FPGAs suffer from storage inefficiency. In this paper, a novel, efficient and modular technique for constructing BCAMs out of standard SRAM blocks in FPGAs is proposed. Hierarchical search is employed to achieve high storage efficiency. Previous hierarchical search approaches cannot be cascaded since they provide a single matching address; this incurs an exponential increase of RAM consumption as pattern width increases. Our approach, however, efficiently regenerates a match indicator for every single address by storing indirect indices for address match indicators. Hence, the proposed method can be cascaded and exponential growth is alleviated into linear. Our method exhibits high storage efficiency and is capable of implementing up to 9 times wider BCAMs compared to other approaches. A fully parameterized Verilog implementation is being released as an open source library. The library has been extensively tested using Altera's Quartus and ModelSim.**

***Keywords-content addressable memory; data addressable memory; associative memory; associative array; catalog memory***

## I. INTRODUCTION

Binary content addressable memories (BCAMs), also known as associative memories, are capable of searching the entire memory space for a specific value within a single clock cycle. While a standard RAM returns data located in a given memory address, a BCAM returns an address containing a specific given data, thus performing a memory-wide search for a specific value. BCAMs, being a hardware implementation of associative arrays, are massively parallel search engines accessing all memory content to compare with the searched pattern simultaneously. BCAMs are considered heavy power consumers due to the very wide memory bandwidth requirement and the concurrent compare.

A BCAM is actually a high-performance implementation of a very basic and widely used associative search, hence it's used almost in every science field requiring high-speed processing of associative search. Networking, associative caches and TLBs, pattern matching, data compression, DSP, bioinformatics, and other variety of scientific fields use CAMs as single-cycle associative search accelerators with millions of search lines. Yet, FPGAs lack an area-efficient soft CAM implementation. Current CAM approaches in vendor IP libraries achieve a maximum of 64K entries and utilize all the resources of a modern FPGA device.

BCAMs are usually designed at the transistor level [1]. Older devices, including Altera's FLEX, Mercury and APEX devices [9] employed minor architectural features to support small CAM blocks. However, FPGA vendors do not provide dedicated hard cores for CAMs in modern devices. These have been replaced with soft CAM cores that employ a brute-force approach of transposed RAM described in this paper as the traditional or transposed-indicators RAM approach.

While address space in modern databases can easily exceed millions of entries, traditional BCAM techniques in FPGAs cannot satisfy these requirements. Wide and shallow RAMs are needed to efficiently implement brute-force BCAMs. Shallow RAMs are required because each extra bit in the CAM pattern width doubles the required RAM depth, resulting in poor efficiency. Instead RAMs should be shallow; wider patterns can be matched by a cascade of AND'ing several matches in parallel. In addition, deeper CAMs can be built by increasing RAM width. However, BCAM requirements are getting deeper as FPGAs advance, yet individual FPGA RAM block width is growing slowly. For example, M4K blocks in Stratix II devices have minimal depth of 128 with maximal width of 36, M9K blocks in Stratix III and Stratix IV devices have minimal depth of 256 and maximal width of 36, M20K blocks in Stratix V devices have minimal depth of 512 and maximal width of 40. With the increasing depth of RAMs, and limited width growth, the brute-force approach is getting less efficient.

Hierarchical search BCAMs [3] efficiently reduce search space by dividing the address range into sets. First, a set with a match is found, and then the exact match location within this set is found. However, Hierarchical search BCAMs can generate only one match location, since only one set is searched for the exact location. Therefore, Hierarchical search BCAMs cannot be cascaded, which incurs exponential growth of RAM consumption as pattern width increases. Adversely, Hierarchical search BCAMs support only narrow patterns, hence they are impractical for most BCAM applications.

In this paper, a modular SRAM-based BCAM is proposed. Similar to hierarchical search BCAMs, our approach arranges the memory into two-dimensional data sets, hence the name two-dimensional BCAM. Our approach, however, is superior

to the hierarchical search approach since it efficiently regenerates match indicators for every single address by storing indirect indices for address match indicators. Thus, unlike hierarchical search, the proposed method can support wide patterns by cascading; this transforms exponential RAM growth into linear.

The proposed method is device-independent; hence, it can be applied to any FPGA device containing standard dual-ported BRAMs. The proposed approach dramatically improves CAM area efficiency compared to conventional methods. In contrast to algorithmic approaches (*e.g.* hashes and tries) or other BCAM techniques that require several nondeterministic cycles to write or match [5][6][7], our approach is high-throughput and can perform a pattern read (match) every cycle and a pattern write every two cycles.

Major contributions of this paper are:

- A novel highly efficient BCAM architecture. Compared to other BCAM approaches, the proposed technique provides up to 9 times wider BCAMs. To the authors' best knowledge, research and patent literature do not have similar BCAM techniques.
- A parameterized Verilog implementation of the our method, together with other approaches. A flow manager to simulate and synthesize various designs with various parameters in batch using Altera's ModelSim and Quartus is also provided. The Verilog modules and the flow manager are available online [2].
- A pipelined, FPGA-optimized wide priority-encoder used in our BCAM architecture.
- A CAM bypassing mechanism is also provided to write and match the same pattern in the same cycle.

To verify correctness, the proposed BCAM architecture is fully implemented in Verilog, simulated using Altera's ModelSim, and compiled using Quartus II [8]. A large variety of BCAM architectures and parameters, e.g. BCAM depth and pattern width are simulated in batch, each with over 1 million random BCAM write and match cycles. Stratix V, Altera's high-end FPGA, is used to implement and compare the proposed architecture with previous approaches.

Notation and abbreviations used for the rest of the paper are listed in Table I. The rest of this paper is organized as follows. In Section II, conventional BCAM techniques in embedded systems are reviewed. Our proposed indirectly indexed two-dimensional (II2D) BCAM approach is described in detail in Section III. Discussion of the suggested method and comparison to previous techniques are provided in Section IV. The experimental framework, simulation and synthesis results, are discussed in Section V. Future improvements and conclusions are drawn in Section VI.

TABLE I. LIST OF NOTATIONS AND ABBREVATIONS

| | | | |
|---|---|---|---|
| $R_D, C_D$ | RAM, CAM depth | $I_{p,a}$ | Single match indicator |
| $D_W, P_W, S_W$ | Data, pattern, set width | $I_{p,s}^{Sw}$ | Set match indicator |
| $P_{W,opt}$ | Optimal cascaded pattern width | WPatt, WAddr | Write pattern, address |
| $n_C$ | Number of cascades | MPatt, MAddr | Match pattern, address |
| $A, S, P$ | Address, set, pattern sets | MIndc | Match indicators |
| $R_{W,max}$ | Widest RAM configuration | RmPatt | Removed pattern |
| $R_{D,min}$ | Shallowest RAM configuration | MultiPatt | Multiple patterns |

## II. BACKGROUND ON FPGA-BASED CAMS

This section provides a review of current BCAM architectures in FPGAs. Using registers to create BCAMs is described in subsection A. The traditional brute-force BRAM-based approach is described in subsection B. BCAM cascading is described in section C. Hierarchical search BCAMs are explained in section D. A review of FPGA vendors' supports of BCAMs is placed in subsection E.

### A. Register-based BCAMs

The flexibility of reading and writing flip-flops makes it possible to concurrently read and compare all the patterns as depicted in Fig. 1. Similar to a register-based RAM, an address decoder is used to generate one-hot decoded address lines, enabling a single line for writing. Each registered pattern is compared with the matched pattern (MPatt) simultaneously; the comparison results drive the match line, also called match indicators (MIndc) followed by a priority-encoder to detect the first matching address (MAddr). The high demand for limited resources such as registers, comparators, address decoder and priority encoder (proportional to BCAM depth) make this approach infeasible for deep BCAMs; using Altera's high-end Stratix V device, only one byte-wide, 32K-depth BCAM can be generated.

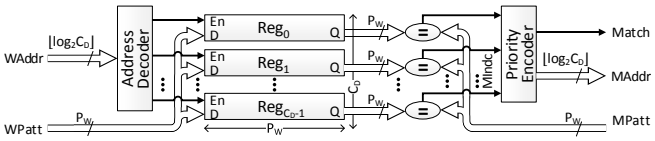


Figure 1. Register-based BCAM

### B. Brute-force Approach via Transposed-Indicators RAM

The brute-force approach creates BCAMs out of standard BRAMs. These BRAMs are addressed with the match pattern, thus allowing a single cycle pattern match. This approach is utilized by FPGA vendor IP libraries or application notes. As depicted in Fig. 2, a BRAM is addressed by the match pattern while each bit of the RAM data indicates the existence of the pattern of each BCAM address location. A RAM with $R_D$ lines in depth and $D_W$ bits data width allows a BCAM with $\mathrm{C_D} = \mathrm{D_W}$ lines depth and $P_W = \lfloor \log_2 R_D \rfloor$ bits pattern width. In this paper, this structure is called *Transposed-Indicators-RAM* (TIRAM) and is described as a matrix of indicators

$$TIRAM = \begin{bmatrix} I_{0,0} & I_{0,1} & \dots \\ I_{1,0} & I_{1,1} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \Big| \forall_{p \in P}^{a \in A} : I_{p,a} = (RAM[a]\ EQ\ p) \quad (1)$$

Where $A$ is the address space set and $P$ is the pattern set in the corresponding RAM structure.

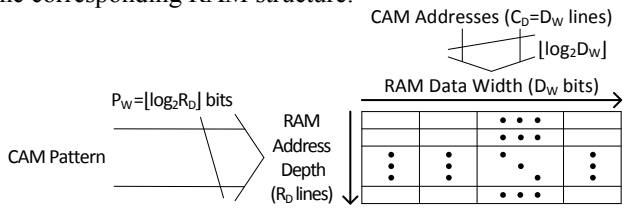


Figure 2. BCAM as Transposed-RAM

Writing to the TIRAM structure requires writing to a single bit in the TIRAM to set or clear a pattern indicator. As described in Fig. 3 (top), this can be achieved by employing the BRAM mixed-width capability in simple dual-ported mode supported by most FPGA vendors. The writing port width is set to a single bit, while the reading port is set to the maximum available width. The byte-enable functionality can also be utilized to write part of the data line as described in Fig. 3 (middle); however, usually byte-enables do not control fine-grained parts of the data, which make it impractical for TIRAM implementation. Both mixed-width and byte-enable methods can be combined as shown in Fig. 3 (bottom).

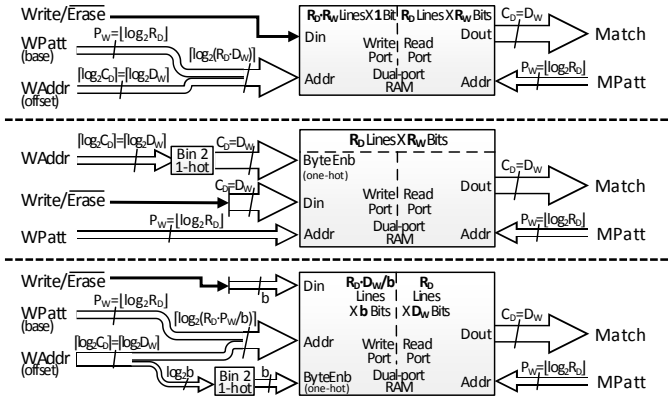


Figure 3. Transposed-Indicators-RAM (TIRAM) implementation using (top) Mixed-width (middle) Byte-enable (bottom) Combined methods

The complete system of the brute-force TIRAM approach is described in Fig. 4. Reading from the TIRAM is performed by providing the match pattern (MPatt) as address to read the match indicators (MIndc) for the entire BCAM address space for this specific match pattern. A priority encoder detects the first match address (MAddr) from the match indicators. However, writing (also called rewriting or updating) to the TIRAM structure requires more computation since it requires setting the new indicator and clearing the old indicator. As shown in Fig. 4, an ErRAM (Erase RAM) is used in parallel to the indicators RAM (TIRAM) in order to track the BCAM content and provide for a given address what pattern is already stored and should be removed. This is useful for BCAM writing operation where the old indicator should be cleared; ErRAM will provide the old pattern in the current written address. BCAM writing will consume two cycles as follows.

1. Set cycle:
   1.1. Set (write '1') a new pattern indicator to TIRAM
   1.2. Read old data (pattern) from ErRAM
2. Clear cycle:
   2.1. Clear (write '0') the old indicator from the TIRAM (location is already provided by step 1.2)
   2.2. Write new data (pattern) to ErRAM

To implement a BCAM with $C_D$ lines and $P_W$ pattern width, namely a $C_D \times P_W$ BCAM, The brute-force TIRAM approach requires $C_D \cdot P_W$ SRAM cells for the ErRAM and $2^{P_W} \cdot C_D$ SRAM cells for the TIRAM, a total of

$$C_D \cdot P_W + 2^{P_W} \cdot C_D. \tag{2}$$

Assuming that ErRAM is fully utilized, and TIRAM uses the widest BRAM configuration, BRAM count is estimated as

$$\left\lceil \frac{C_D \cdot P_W}{R_{D,min} \cdot R_{W,max}} \right\rceil + \left\lceil \frac{2^{P_W}}{R_{D,min}} \right\rceil \cdot \left\lceil \frac{C_D}{R_{W,max}} \right\rceil. \tag{3}$$

Where $R_{W,max}$ and $R_{D,min}$ are BCAM parameters indicating the maximum width, and minimum depth, respectively.

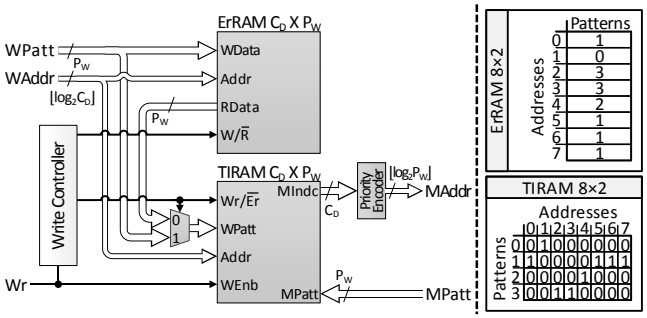


Figure 4. (left) Brute-force TIRAM approach (right) 8×2 example

## C. Cascading and Scaling of the Brute-force Approach

As shown in (2), SRAM cell usage for the brute-force TIRAM approach is exponential to pattern width $P_W$. A wide pattern width will make the SRAM requirements infeasible. BCAM cascading alleviates the SRAM usage from exponential into linear. As depicted in Fig. 5, the BCAM pattern (both matched and written pattern) is divided into smaller pattern segments; each segment is associated with a separate BCAM. The write operation writes every pattern segment into its corresponding BCAM, while match operation matches each pattern segment with its corresponding CAM. A match for the entire pattern is found if a match is found for all segments individually (hence the bitwise AND). The optimal pattern segment width is determined by the minimal depth $R_{D,min}$ of the BRAM, namely the shallowest and widest configuration, since choosing a wider pattern requires exponential growth. The optimal pattern width is therefore $P_{W,opt} = \lfloor \log_2(R_{D,min}) \rfloor$ and the total number of BCAM cascades is $n_C = \lceil P_W / P_{W,opt} \rceil$.

One stage of TIRAM will consume $\lceil C_D / R_{W,max} \rceil$ BRAMs and the total BCAM consumption of the TIRAM is therefore

$$n_C \cdot \left( \left\lceil \frac{C_D}{R_{W,max}} \right\rceil + \left\lceil \frac{C_D \cdot P_{W,opt}}{R_{D,min} \cdot R_{W,max}} \right\rceil \right). \tag{4}$$

The linear relation to $P_W$ and $C_D$ in (4) is clear, in contrary to the uncascaded version in (3) where the relation to $P_W$ is exponential.

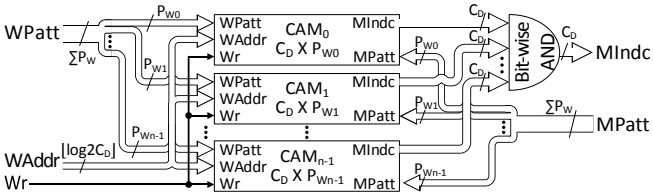


Figure 5. BCAM cascading

### D. Deep BCAMs via Hierarchical Search

The previously shown brute-force TIRAM approach requires a match indicator for each single address location. This structure is inefficient for deep BCAMs since it requires a RAM with a width equals to the BCAM depth to implement the TIRAM structure. On the other hand, BCAMs with hierarchical search [3] alleviate this requirement. Instead of storing match indicators for each address location separately, an indicator is generated for a set of $S_W$ addresses, indicating whether the pattern exists at any of these addresses in the set. For a BCAM with $C_D$ entries in depth, and $S_W$ set width, $\lceil C_D/S_W \rceil$ sets exist, and can be enumerated as

$$S = \{0,1,\dots,\lceil C_D/S_W \rceil - 1\}. \quad (5)$$

A set indicator $I_{p,s}^{S_W}$ indicates if any of the addresses in set $s$, hence addresses $S_W \cdot s$ upto $S_W \cdot (s+1) - 1$ contains the pattern $p$. The set transposed indicators RAM (STIRAM) is therefore,

$$STIRAM = \begin{bmatrix} I_{0,0}^{S_W} & I_{0,1}^{S_W} & \dots \\ I_{1,0}^{S_W} & I_{1,1}^{S_W} & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \Bigg\| \forall_{p \in P}^{s \in S} I_{p,s}^{S_W} = \bigvee_{a=S_W \cdot s}^{a=S_W \cdot (s+1)-1} I_{p,a}. \quad (6)$$

STIRAM identifies which sets hold the match pattern, not the exact location. To detect the exact pattern location, an auxiliary RAM stores the patterns associated with each set, each set's patterns packed in one RAM line, hence it called the sets RAM (SetRAM). Fig. 6 (left) provides an example of these two structures.

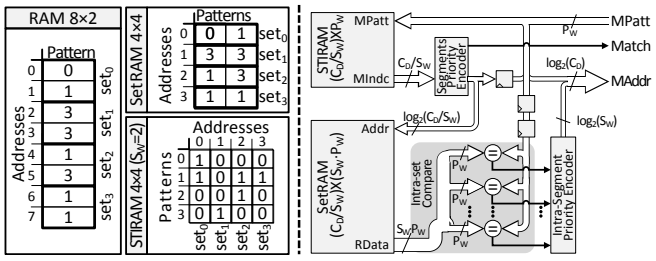


Figure 6. (left) An example of STIRAM and SetRAM structures for a 8×2; $S_w$=2 BCAMs (right) Hierarchical search matching mechanism

Fig. 6 (right) illustrates the matching mechanism of the hierarchical search system; the writing mechanism is omitted for simplicity. The match operation will search the STIRAM for a set with a matching pattern. A priority-encoder generates the address of the first set with matching pattern; this address is used to address the SetRAM and fetch the entire set patterns. Finally, the match pattern is compared concurrently with all the patterns is set to find the exact location.

Similar to the brute-force TIRAM approach, writing to the STIRAM BCAM requires two cycles, one cycle for new pattern insertion, and a second cycle for old pattern deletion.

To implement a BCAM with $C_D$ entries and $P_W$ pattern width, namely a $C_D \times P_W$ BCAM, this approach requires $\lceil C_D/S_W \rceil \times P_W \cdot S_W$ SRAM cells for the SetRAM and $2^{P_W} \times \lceil C_D/S_W \rceil$ SRAM cells for the STIRAM, a total of

$$\left\lceil \frac{C_D}{S_W} \right\rceil \cdot P_W \cdot S_W + 2^{P_W} \cdot \left\lceil \frac{C_D}{S_W} \right\rceil. \quad (7)$$

Assuming a wide RAM requirement, an upper bound estimate for the BRAMs needed to construct the STIRAM is

$$\left\lceil \frac{2^{P_W}}{R_{D,min}} \right\rceil \cdot \left\lceil \frac{\lceil C_D/S_W \rceil}{R_{W,max}} \right\rceil. \quad (8)$$

The SetRAM is a true dual-port RAM. $R_{W,byte}$ describes the width of data controlled by a single byte-enable. A single BCAM accommodates $R_{W,max}/R_{W,byte}$ individual byte-enable controlled data, and each pattern requires $\lceil P_W/R_{W,byte} \rceil$ of them. Finally, $S_W$ lines of this structure are required. Therefore, the number of BCAMs needed to construct the SetRAM is

$$\frac{R_{W,max}}{R_{W,byte}} \cdot \left\lceil \frac{P_W}{R_{W,byte}} \right\rceil \cdot \left\lceil \frac{S_W}{R_{D,min}} \right\rceil. \quad (9)$$

Since a set of address match indicators can be lumped into a single set indicator, the hierarchical search approach can support very deep BCAMs. However, cascading is not possible since it requires the entire match indicators for every address, while only a single priority-encoded match address is produced. Therefore, the memory consumption of this method is exponential to the pattern with. Only very narrow patterns are be supported, which makes the hierarchical search approach impractical for most applications.

### E. Vendor Support of BCAMs

Modern FPGAs provide plenty of embedded hard-coded blocks, such as block RAM, external memory controllers, processors, DSP blocks/multipliers, and fast I/O transceivers. However, hard CAM blocks do not exist in modern FPGAs – presumably due to their high area and power consumption, and their highly specialized nature. While most FPGA vendors provide simple register-based or brute-force SRAM-based CAMs, some old devices provide partial support for CAM construction. Altera's legacy FLEX, Mercury and APEX [9] device family integrates intrinsic BCAM support into their embedded system blocks (ESBs). The ESB can be configured into several modes; a 2Kbits RAM/ROM mode with configurable width and depth, 32 product terms with 32 literal inputs, or a 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. Since ESBs are limited to a few hundred blocks in these devices, deep CAMs are infeasible. Furthermore, BCAMs can only be implemented as soft IP in modern Altera devices.

On the other hand, Xilinx devices do not provide native support for BCAMs. However, partial configuration capabilities in Xilinx Virtex devices can be utilized to create a CAM as described in Xilinx application notes [12][13] and this approach is very slow at writing new patterns. Other Xilinx application notes suggest utilizing BCAMs with the brute-force approach to create BCAMs [13][14].

Lattice ispXPLD devices [15] have an integrated support for CAMs via their Multi-Function Blocks (MFBs) which can be configured into 128×48 Ternary CAM block (with don't care values). Alternatively, Actel application notes [16] recommend using multi-cycle CAMs by searching BRAM in parallel; for a single-cycle CAM, using registers is suggested.

## III. Indirectly Indexed Two-Dimensional BCAMs

In this section, the proposed Indirectly Indexed Two-Dimensional (II2D) BCAM method is described in detail. The motivation and key idea for this work are explained in subsection A. The design method is described in subsection B. The construction of wide priority encoders in FPGAs, which are crucial for BCAMs in general, is described in subsection C. BCAM bypassing techniques are described in subsection D. Subsection E describes a device-specific instance for Altera's Stratix device family.

### A. Motivation and Key Idea

As shown in subsection II.D, Hierarchical search BCAMs cannot be cascaded, hence suffer from exponential growth of the required memory size as function of pattern width. This limitation is crucial since the vast majority of applications require wide patterns. To support BCAM cascading, all match indicators for every single BCAM address shall be generated, as in the brute-force approach (subsection II.C). However, storing all match indicators requires wide RAM and incurs high memory overhead.

Our proposed Indirectly Indexed Two-Dimensional (II2D) BCAM is based on hierarchical search BCAMs and utilize the sparsity of the set indicators RAM to store only the required match indicators, and is able to regenerate all match indicators for every BCAM address, hence can be cascaded. The following theorem is the basic keystone of our technique.

**Lemma 1** The number of binary '1' (matches) in each column (address) of the TIRAM matrix from (1), is exactly 1, namely,

$$\forall a \in A\colon \sum_{p\in P} I_{p,a} = 1. \tag{10}$$

**Proof** Similar to RAM, each address of a BCAM contains one and only one valid pattern. A pattern $p`$ located at address $a`$ means $I_{p`,a`} = 1$ and $I_{p,a`} = 0$ for every $p \neq p`$. Hence, the corresponding column of address $a`$ in the TIRAM matrix has a binary '1' only in $I_{p`,a`}$ and zeros for all the other patterns. ■

**Theorem 1** The number of binary '1' (matches) in each column (set) of the STIRAM matrix from (6) is limited to the set width $S_W$, namely,

$$\forall s \in S\colon \sum_{p\in P} I_{p,s}^{S_W} \leq S_W. \tag{11}$$

**Proof** A match indicator of a set is defined in (6) and indicates if any of the addresses in the set has a match. Given a set $s`$ of $S_W$ addresses, (6) provides $I_{p,s`}^{S_W} = \bigvee_{a=S_W\cdot s`}^{a=S_W\cdot(s`+1)-1} I_{p,a}$. Lemma 1 insures that each address $a \in \{S_W \cdot s`, \ldots, S_W \cdot (s`+1) - 1\}$ has one and only one $p$ such that $I_{p,a} = 1$. Since the set $s`$ has $S_W$ addresses, exactly $S_W$ different $I_{p,a} = 1$ exist. Hence, in the entire set, maximum $S_W$ patterns have a match (less than $S_W$ in case two addresses or more has the same pattern). ■

The significance of theorem 1 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 pairs as in brute-force approach (Fig. 7 (left)), or set indicators as in hierarchical search approach (Fig. 7 (middle)), we store all address indicators only for sets with a match, as they are limited to $S_W$. To reduce memory consumption, address match indicators are saved in another auxiliary structure, while the original STIRAM will hold indices to the auxiliary structure (Fig 7. (right)).

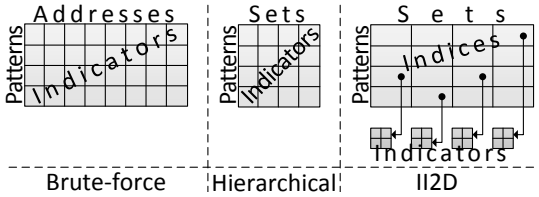


Figure 7: Indicators arrangment for three different approaches

### B. II2D-BCAM: Design and Functionality

As depicted in Fig. 8, a single-stage of the proposed II2D-BCAM consist of three parts. First, the *match RAM* where the set indices and match indicators are stored; this is the most memory consuming structure. Second, the *status RAM* where the system status is stored and feeds the control logic with system status. Finally, the *control and steering logic*, which generates control signals to control the entire structure (based on system status), feeds the match RAM with indicators and indices, and updates the status RAM.

#### 1) Match RAM:

As described in the previous subsection, instead of storing set match indicators in the STIRAM, we store only indirect indices for other auxiliary RAM, which holds the match indictors for all the addresses in the set, hence it called the indicators RAM (IndRAM). Theorem 1 shows that maximum $S_W$ different patterns can have a match in a set; hence, the depth of IndRAM is $S_W$ at most. Each set has $S_W$ addresses, therefore IndRAM should have $S_W$ address indicators for each set and its width should also be $S_W$. The address space consists of $\lceil C_D/S_W \rceil$ sets; hence, $\lceil C_D/S_W \rceil$ IndRAM $S_W \times S_W$ blocks are required.

The STIRAM hold indices for all pattern and set pairs. To represent all patterns the required depth is $2^{P_W}$. For each of the $\lceil C_D/S_W \rceil$ sets, $\lceil \log_2(S_W) \rceil$ bits are required for each index. In total, the STIRAM width is $\lceil C_D/S_W \rceil \cdot \lceil \log_2(S_W) \rceil$ bits.

#### 2) Status RAM:

The status RAM is updated at each write to reflect the system status and consists of three RAM structures as follows.

##### a) Sets RAM (SetRAM):

Similar to the hierarchical search method, this RAM holds the entire CAM patterns, each set's patterns packed in one row, hence, can be fetched in a single cycle. Its size is therefore $\lceil C_D/S_W \rceil \times (P_W \cdot S_W)$.

##### b) Indices RAM (IdxRAM):

The Indices RAM stores the index of each pattern in the BCAM, arranged similar to the SetRAM, namely each set's indices in one row. Its size is $\lceil C_D/S_W \rceil \times (\lceil \log_2(S_W) \rceil \cdot S_W)$.

*c) Vacancy RAM (VacRAM):*

The Vacancy RAM indicates for each row of the IndRAM whether it is vacant or holds valid indicators. The IndRAM consists of $\lceil C_D/S_W \rceil$ RAM blocks (for each set), each with $S_W$ rows. The Vacancy RAM hold the status of each IndRAM block in one row, hence its depth is $\lceil C_D/S_W \rceil$ and marks the validity of all the $S_W$ IndRAM rows, hence its width is $S_W$.

*3) Control and steering logic:*

Based on current system status and the required write pattern and write address, it generates write indicators to IndRAM and indices to IndRAM and STIRAM. Furthermore, it updates the IdxRAM and VacRAM status.

TABLE II. BRAM USAGE OF A SINGLE[1] STAGE II2D-BCAM

| Structure | Blocks # | Depth | Width |
|---|---|---|---|
| STIRAM | 1 | $2^{P_W}$ | $\lceil C_D/S_W \rceil \cdot \lceil \log_2(S_W) \rceil$ |
| IndRAM | $\lceil C_D/S_W \rceil$ | $S_W$ | $S_W$ |
| SetRAM | 1 | $\lceil C_D/S_W \rceil$ | $P_W \cdot S_W$ |
| IdxRAM | 1 | $\lceil C_D/S_W \rceil$ | $\lceil \log_2(S_W) \rceil \cdot S_W$ |
| VacRAM | 1 | $\lceil C_D/S_W \rceil$ | $S_W$ |

1. For cascaded II2D-BCAM, cascading method from II.C. is employed; $P_W = P_{W,opt} = \lfloor \log_2(R_{D,min}) \rfloor$ is used for each stage, while $n_C = \lceil P_{W,total}/P_{W,opt} \rceil$ stages are required.

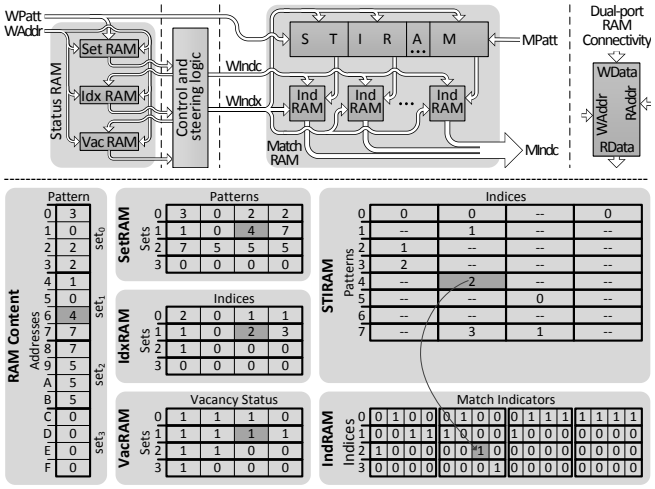


Figure 8: II2D-BCAM single-stage (top) high-level architicture (bottom) 8×3; $S_w$=4 example; pattern '4' is highlighted with all related RAM content

## C. Wide Priority Encoders in FPGAs

A priority-encoder, also called leading zero detector (LZD) or leading zero counter (LZC), receives an $n$-bit input vector and detects the index of the first binary '1' in the input vector. A valid signal indicates if any binary '1' was detected in the input vector, hence the index is valid.

Our proposed II2D-BCAM consists of a $\lceil C_D/S_W \rceil$ wide priority-encoder, for deep BCAMs the priority-encoder delay considerably affects the overall performance; hence, a fast priority-encoder design is essential.

As depicted in Fig. 9, the suggested priority-encoder is recursively constructed. The input vector is split into $k$ equal fragments with $n/k$ bits. A priority encoder $PE_{n/k}$ with a narrower width of $n/k$ is applied for each fragment. The valid bit of each of the $k$ $PE_{n/k}$'s goes to a $k$ bit $PE_k$ to detect the first valid fragment. The location of this fragment is the higher part of the overall index, and steers the exact location within the fragment itself to produce the lower part of the overall index.

The depth of the proposed structure is $\lceil \log_k n \rceil$, while the hardware area complexity is $O(n)$. If Altera's Stratix V or equivalent device is used, $k = 4$ is recommended to achieve higher performance and area compression, since the mux can be implemented using 6-LUT, hence an entire ALM.

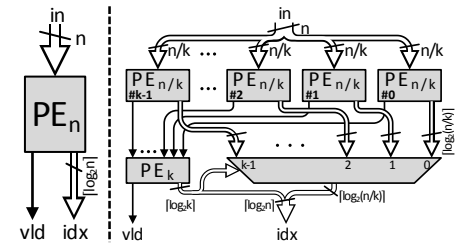


Figure 9. Priority-encoder (left) symbol (right) recursive definition

## D. BCAM Bypassing

Writing a new pattern to a register-based BCAM is immediate on the triggering clock edge and it is ready to be matched immediately. In contrast, BRAM-based BCAM methods, namely TIRAM and STIRAM, require two cycles for writing. Hence, matching a pattern that is being written in the same cycle will match old indicators, introducing a read-after-write hazard. However, some applications, *e.g.* caches and TLBs, require immediate matching of the recently written patterns, hence, pattern bypassing is required.

Fig. 10 shows the bypassing circuitry for both TIRAM and STIRAM. In both, the writing address (WAddr) is one-hot encoded; the insertion mask (bitwise OR) forces '1' into the matched indicators (MIndc) in location MAddr. Similarly, the removal mask (bitwise AND) forces '0' into the matched indicators (MIndc) in location MAddr. In the TIRAM approach, if the matched pattern (MPatt) is equivalent to the written pattern (WPatt), the insertion mask output is passed to the matching indicators output (MIndc); otherwise the removal mask output is passed. In the STIRAM approach, there is another case. The removal mask output is allowed to be passed only if MPatt equals to the removed pattern (RmPattern), and there are no other occurrences of the removed pattern in the same set (negated MultiPatt).

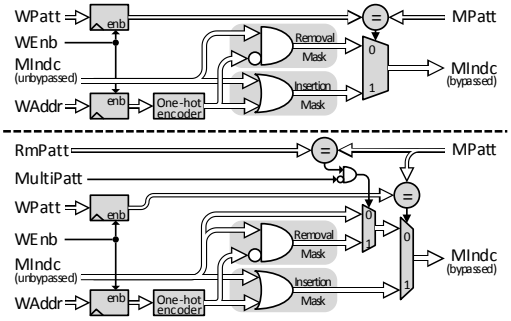


Figure 10. Bypassing logic (top) brute-force approach (bottom) proposed II2D-BCAM approach

## E. Feasibility on Altera's Stratix Devices

An area efficient implementation of the proposed II2D-BCAMs requires heterogeneous BRAMs on FPGA. The relatively small BRAMs will be used to construct IndRAM and store match indicators. A perfect candidate is the LUT

configuration RAM, known as LUT RAM, and is supported by both Altera and Xilinx devices. On the other hand, the relatively large BRAMs will be used to construct other structures; M20K BRAMs will be used for this purpose.

Altera's Stratix V memory architecture provides a 640-bit LUT RAM memory called MLAB (Memory Logic Array Block) as well as 20Kb BRAMs called M20K.

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 IndRAM, we set $S_W = 32$. With an index width of $\lceil \log_2(S_W) \rceil = 5$, a single M20K line can store up to 8 indices. M20K's mixed-width port capability is used to write a single 5-bit index.

## IV. Comparison and Discussion

The BCAM storage efficiency $\mu_s$ is first introduced in this paper, and is defined as the SRAM cell utilization for a specific BCAM implementation. In other words, $\mu_s$ is the ratio between the total BCAM bits and the total SRAM cells used to implement this BCAM.

Table III shows storage efficiency estimation for different BCAM architectures and is derived from (2), (4), (7) and Table II. The storage efficacy of the uncascaded brute-force implementation (UBF) is inversely proportional to the exponent of $P_W$, hence, decays rapidly with $P_W$ increase. The cascaded brute-force (CBF) is not related to $P_W$, hence the efficiency is not affected when $P_W$ increases. However, the efficiency is affected by an intrinsic BRAM characteristic, the minimal depth $R_{D,min}$ (associated with the maximum width). The efficiency is inversely proportional to $R_{D,min}$, hence, shallow and wide BRAMs with smaller $R_{D,min}$ (and larger $R_{W,max}$) will exhibit higher storage efficiency. Similar to the uncascaded brute-force (UBF), the efficiency of the hierarchical search approach (HS) is inversely proportional to the exponent of $P_W$. However, the exponential relation is mitigated by the set width $S_W$, an external and user-driven parameter. On the other hand, the efficiency of our II2D-BCAM approach is not related the pattern width $P_W$, but is augmented by $S_W$. To generalize, HS-BCAM provides the highest efficiency up to

$$P_W = -\frac{W_{-1}\left(-\frac{\ln 2}{a}\right)}{\ln 2}, a = \frac{S_W \cdot R_{D,min}}{\log_2(R_{D,min})}, \quad (12)$$

Where $W_{-1}$ is the lower branch of the Lambert Product Logarithm function (also called Omega function).

For the HS-BCAM approach, the set width $S_W$ has a limitation due to the minimal width of the SetRAM it is stored in. The SetRAM depth is $\lceil C_D/S_W \rceil$ and is limited by $R_{D,min}$. Hence, to achieve maximum efficiency, $S_W$ is bounded by

$$S_W \leq \frac{C_D}{R_{D,min}}. \quad (13)$$

On the other hand, the set width $S_W$ for our II2D-BCAM is bounded by internal RAM parameters. Specifically, the depth of the LUT RAM and the BRAM allowable write port width in mixed-width mode. As described in subsection III.E, Altera's Stratix V MLAB depth (for the widest configuration) is 32, and $\log_2(32) = 5$ is allowed as write data width for M20K mixed-width configuration, hence, $S_W = 32$ is the suitable set width.

TABLE III. Storage Efficiency (Inversed)

| | Uncascaded | Cascaded |
|---|---|---|
| Brute-force | $\mu_s(UBF)^{-1} \approx 1 + \frac{2^{P_W}}{P_W}$ | $\mu_s(CBF)^{-1} \approx 1 + \frac{R_{D,min}}{\log_2(R_{D,min})}$ |
| Hierarchical Search | $\mu_s(HS)^{-1} \approx 1 + \frac{2^{P_W}}{S_W \cdot P_W}$ | $\mu_s(II2D)^{-1} \approx 1 + \frac{1 + \frac{R_{D,min} \cdot \log_2(S_W)}{S_W^2}}{\log_2(R_{D,min})}$ |

## V. Experimental Results

To verify and simulate the suggested II2D approach and compare to standard and previous techniques, fully parameterized Verilog modules have been developed. Register-based, cascaded and uncascaded brute-force TIRAM, Hierarchical search BCAM, and the proposed II2D-BCAM methods have been implemented. A run-in-batch flow manager has also been developed to simulate and synthesize these designs with various parameters in batch using Altera's ModelSim and Quartus II. The Verilog modules and the flow manager are available online [2].

To verify correctness, the proposed architecture is simulated using Altera's ModelSim. A large variety of different BCAM architectures and parameters, e.g. BCAM depth and pattern width, are swept and simulated in batch, each with over a million random cycles. All different BCAM design modules were implemented using Altera's Quartus II on Altera's Stratix V 5SGXMABN1F45C2 device. This is a speed grade 2 device with 360k ALMs and 2640 M20K blocks. Half of the ALM's can be used to construct MLABs, while a single MLAB consists of 10 ALMs.

Fig. 11 plots feasible BCAM depth and pattern width sweeps. Within the device limitation, the proposed II2D-BCAM approach is able to reach 153-bits pattern width, while other approaches cannot exceed 45-bits for 16K entries. The number of Altera's M20K blocks used to implement each BCAM configuration is plotted in Fig. 11 (bottom). The BF-BCAM and our II2D-BCAM exhibit a linear growth of M20K consumption as pattern width increases since both methods are cascaded. However, the II2D-BCAM growth rate is lower since addresses are grouped as sets. On the other hand, HS-BCAM suffers from exponential growth, hence it cannot exceed a very narrow 19-bits pattern width.

As shown in Fig. 11 (middle), the proposed II2D-BCAM method and the BF-BCAM also exhibit linear ALM count growth as pattern width increases. The priority-encoder in the HS-BCAM approach is split in two, hence the ALM count is lower. Furthermore, our II2D-BCAM approach uses ALMs as MLABs, hence it has higher ALM consumption. The register-based BCAM consumes the most ALMs due to massive register usage.

Figure 11 (top) plots the Fmax of all BCAM architectures. The HS-BCAM exhibit the highest Fmax for very narrow patterns, where it is feasible. However, Fmax drops dramatically as the pattern width increases due to massive

increase of M20K usage. On the other hand, II2D-BCAM performs better than BF-CAM and register-based BCAM for shallow memory. As can be seen, Fmax decreases mildly as pattern goes wider due to cascading.

Similar to BF-BCAM and HS-BCAM, our approach is capable of matching a pattern every cycle and writing a pattern every two. Pipelining is employed to increase Fmax and this adversely increases latency. The longest combinational path goes through the output priority-encoder, and is pipelined every stage. For the device-specific priority-encoder from subsection III.C, $\lceil \log_4 \lceil C_D/S_W \rceil \rceil$ pipe stages are required. HS-BCAM benefits from higher $S_W$, hence it can exhibit lower latency. On the other hand, sets are not used in BF-BCAM, hence its match latency is $\lceil \log_4 C_D \rceil$.

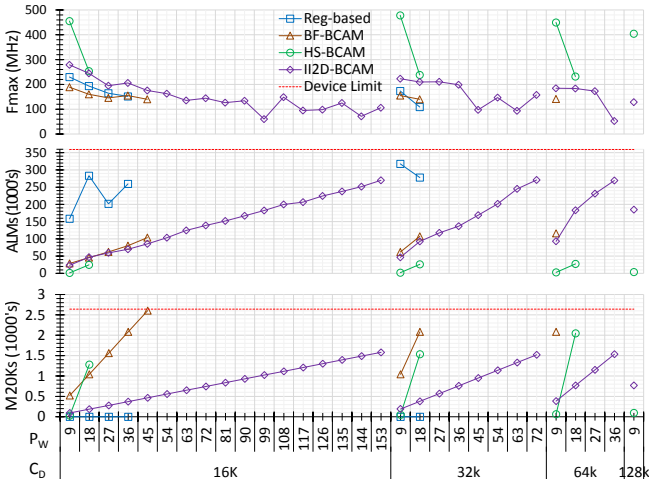


Figure 11: Results for several BCAM depth and pattern width sweeps (bottom) M20K count (middle) ALMs count (top) Fmax at T=0° C

## VI. Conclusions and Future Work

In this paper, a novel BCAM architecture for FPGAs is proposed. The approach is fully BRAM-based and employs hierarchical search to reduce BRAM consumption. While traditional brute-force approaches have a pattern match indicator for every single address, the proposed approach groups addresses into sets and maintains a single pattern match indicator for every set. Current hierarchical search BCAMs cannot be cascaded since they provide a single matching address; this incurs an exponential increase of RAM consumption as pattern width increases. On the other hand, our approach efficiently regenerates a match indicator for every single address by storing indirect indices for address match indicators. Hence, the proposed method can be cascaded and the exponential growth is alleviated into linear. Our technique supports up to 4 times wider patterns compared to brute-force BCAM and up to 9 times wider patterns compared to hierarchical search BCAM.

Furthermore, we provide a high-performance area-wise recursive priority-encoder as an essential part of our BCAM. To match a pattern that is being written at the same cycle, a bypassing mechanism for our BCAM and HS-BCAMs in general is also provided. In addition, we perform a detailed comparison for different FPGA-based BCAM architectures.

A fully parameterized Verilog implementation of the suggested methods is provided as open source hardware [2]. As future work, the suggested BCAMs can be tested with other FPGA vendors' tools and devices. Furthermore, these methods can be tested for ASIC implementation using dual-ported RAMs as building blocks, and compared against customary designed BCAMs. Excessive pipelining and time-borrowing techniques can be used to improve Fmax. The goal would be to recover the frequency drop due to the comparators and priority-encoder. One possible approach uses shifted clocks to provide more reading and writing time [17]. However, adapting this method to BCAMs is not trivial due to internal timing paths across the BCAM. To fit some applications that require single-cycle writing, e.g. caches and TLBs, the proposed technique can be enhanced to support single cycle write by using multi-write RAM [18].

## References


[1] K. Pagiamtzis, A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: a tutorial and survey," *Solid-State Circuits, IEEE Journal of* , vol.41, no.3, pp.712–727, March 2006.

[2] http://www.ece.ubc.ca/~lemieux/downloads/.

[3] A. M.S. Abdelhadi and G.F. Lemieux, "Deep and Narrow Binary Content-Addressable Memories using FPGA-based BRAMs," *IEEE International Conference on Field-Programmable Technology (FPT)*, Dec. 2014, Shanghai, China.

[4] Altera Corporation, *Stratix V Device Handbook*, May 2013.

[5] S. J.E. Wilton, W. Jones, and J. Lamoureux, "An embedded flexible content-addressable memory core for inclusion in a Field-Programmable Gate Array", *in Proceedings of the 2004 International Symposium on Circuits and Systems (ISCAS '04)*, may 2004.

[6] C.W. Jones and S. J.E. Wilton, "Content-Addressable Memory with Cascaded Match, Read and Write Logic in a Programmable Logic Device," U.S. Patent 6 622 204 B1, Sep. 16, 2003.

[7] G.R. Schlacter ,"Emulation of Content-Addressable Memories," U.S. Patent 6 754 766 B1, Jun. 22, 2004.

[8] Altera Corporation, *Quartus II Handbook*, Version 13.1, Nov. 2013.

[9] "APEX 20K Programmable Logic Device Family," Data Sheet, March 2004, ver. 5.1, Altera Corporation, San Jose, CA.

[10] F. Heile, A. Leaver, and K. Veenstra, "Programmable memory blocks supporting content-addressable memory*," in Proceedings of the 2000 ACM/SIGDA eighth international symposium on Field programmable gate arrays*, pp. 13-21, February 10-11, 2000, Monterey, CA.

[11] "Implementing High-Speed Search Applications with Altera CAM," Application Note 119, ver. 2.1, July 2001, Altera Corp., San Jose, CA.

[12] J.-L. Brelet and B. New, "Designing Flexible, Fast CAMs With Virtex Family FPGAs," Application Note XAPP203, 1999, Xilinx, Inc., CA.

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

[14] J.-L. Brelet, "Methods for Implementing CAM Functions Using Dual-Port RAM," U.S. Patent 6 353 332 B1, Mar. 5, 2002.

[15] "Content Addressable Memory (CAM) Applications for ispXPLD Devices," Application Note 8071, 2002, Lattice Corp., Hillsboro, OR.

[16] "Content-Addressable Memory (CAM) in Actel Devices," Application Note AC194, December 2003, Actel Corp., Mountain View, CA.

[17] A. Brant, A. Abdelhadi, A. Severance, G. Lemieux, "Pipeline Frequency Boosting: Hiding Dual-Ported Block RAM Latency using Intentional Clock Skew," *IEEE International Conference on Field-Programmable Technology (FPT)*, Dec. 2012, Seoul, South Korea.

[18] A. M.S. Abdelhadi and G. G.F. Lemieux, "Modular Multi-ported SRAM-based Memories," *In Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays*, pp. 35-44, February 26-28, 2014, Monterey, CA.