Deep and Narrow Binary Content-Addressable Memories using FPGA-based BRAMs

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

***Abstract*—Binary Content Addressable Memories (BCAMs) are massively parallel search engines capable of searching the entire memory space in a single clock cycle. BCAMs are used in wide range of applications, such as tag tables in associative caches or TLBs, network routing tables and intrusion detection, data compression, DSP, databases and bioinformatics. 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 and efficient technique for constructing BCAMs out of standard SRAM blocks in FPGAs is proposed. To achieve high storage efficiency, the suggested technique divides the address range into equal segments. A RAM structure stores for each pattern if it exists in any of the address segments individually. Another RAM stores for each segment, all patterns within this segment. Using Altera's Stratix V device, traditional design method achieves up to 64K-entry BCAM while the proposed technique achieves up to 4M entries. For the 64K-entry test-case, the traditional method consumes 43 times more ALMs, 18 times longer mapping runtime, and achieves only one-third of the Fmax of the proposed method. A fully parameterized Verilog implementation is being released as an open source hardware library. The library has been extensively tested using ModelSim and Altera's Quartus tools.[1]**

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 giving memory address, a BCAM returns an address containing a specific given data, hence performs a memory-wide search for a specific value. BCAMs, 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 [8] 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.

In this paper, a modular SRAM-based BCAM suitable for deep applications is proposed. The address range is divided into equal segments. CAM lookup depends upon two steps. First, a RAM structure stores hit or miss information for each segment, where a segment stores several patterns. Second, the specific segment is searched in parallel for a match. The segment data (the patterns themselves) is stored in a second RAM structure. To achieve a write and a match of the same pattern in the same cycle, a CAM bypassing mechanism is also provided.

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

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

CAM area efficiency and operation frequency compared to conventional methods. In contrast to other attempts that require several cycles to write or match [4][5][6], the proposed 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 BCAM architecture capable of producing millions of CAM lines. Compared to other conventional BCAM approaches, the proposed technique exhibits the highest area efficiency while providing improved overall performance. To the authors' best knowledge, research and patent literature do not have similar BCAM techniques.
- A parameterized Verilog implementation of the suggested methods, together with previous standard approaches. A flow manager to simulate and synthesize various designs with various parameters in batch using Altera's ModelSim and Quartus II is also provided. The Verilog modules and the flow manager are available online [2].

To verify correctness, the proposed BCAM architecture is fully implemented in Verilog, simulated using Altera's ModelSim, and compiled using Quartus II [7]. A large variety of BCAM architectures and parameters, e.g. BCAM depth, pattern width, bypassing, and pipelining are simulated in batch, each with over million random BCAM write and match cycles. Stratix V, Altera's high-end performance-oriented 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. The proposed segmented transposed RAM 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	n_C	Number of cascades
D_W, P_W, S_W	Data, pattern, segment width	WPatt, WAddr	Write pattern, address
$P_{W,opt}$	Optimal cascaded pattern width	MPatt, MAddr	Match pattern, address
$I_{p,a}$	Match indicator	MIndc	Match indicators
A, S, P	Address, segment, pattern set	RmPatt	Removed pattern
$R_{W,max}, R_{D,min}$	Widest, Shallowest RAM	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. A review of FPGA vendors' supports of BCAMs is placed in subsection D.

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) following 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), besides the increasing routing complexity, makes this approach infeasible for deep BCAMs. Using Altera's high-end Stratix V device, only 32K-line single byte pattern BCAM can be generated.

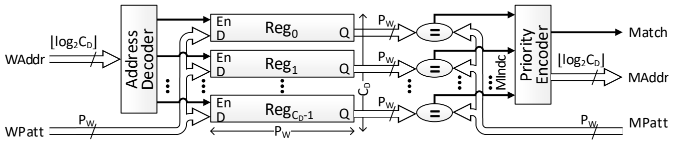


Fig. 1. Register-based BCAM

B. Brute-force Approach via Transposed-Indicators RAM

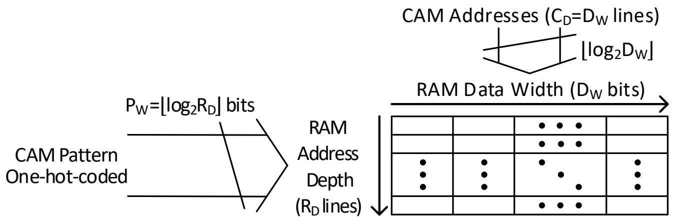
The brute-force approach for creating a BCAM out of standard SRAM block is to address these SRAM blocks with the match pattern, thus allowing a single cycle pattern search. This is the approach taken by FPGA vendor IP libraries or application notes. As depicted in Fig. 2, the RAM is addressed by the 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 $C_D = D_W$ 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 as follows

$$TIRAM = \begin{bmatrix} I_{0,0} & I_{0,1} & \cdots & I_{0,C_D-1} \\ I_{1,0} & I_{1,1} & \cdots & I_{1,C_D-1} \\ \vdots & \vdots & \ddots & \vdots \\ I_{max(P),0} & I_{max(P),1} & \cdots & I_{max\,(P),C_D-1} \end{bmatrix}, \quad (1)$$

$$\forall a \in A, p \in P\colon\ I_{p,a} = (RAM[a]\ EQ\ p).$$

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



CAM Addresses (C_D=D_W lines)
⌊log₂D_W⌋
RAM Data Width (D_W bits)
P_W=⌊log₂R_D⌋ bits
CAM Pattern One-hot-coded
RAM Address Depth (R_D lines)

Fig. 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).

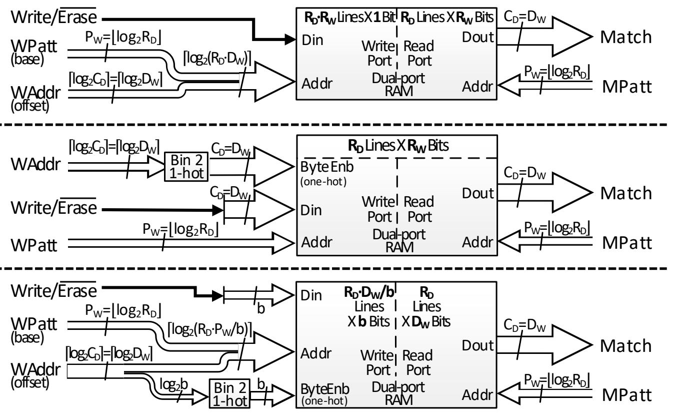


Fig. 3. Transposed-Indicators-RAM (TIRAM) implementation using (top) Mixed-width BRAM (middle) Byte-enable (bottom) Combined methods

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. Hence, the 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 form 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

$$C_D \cdot P_W + 2^{P_W} \cdot C_D. \qquad (2)$$

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

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

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

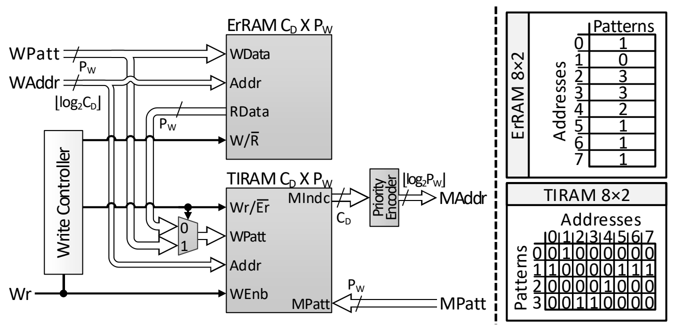


Fig. 4. Brute-force TIRAM approach (left) 8×2 example (right)

C. BCAM Cascading and Scaling

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 determined by the minimal depth $R_{D,min}$ of the BRAM, namely the shallowest and widest configuration, since choosing 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

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

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.

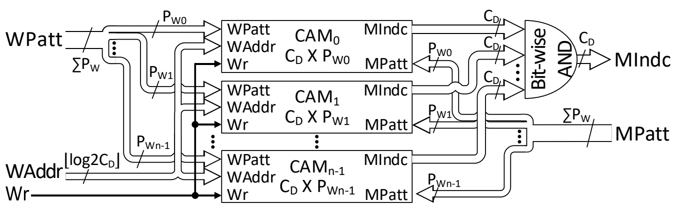


Fig. 5. BCAM cascading

D. *Vendor Support for 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 [8] 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 few hundred blocks in these devices, and due to routing complexity, deep CAMs are unfeasible. Furthermore, BCAMs can only be implemented as soft IP in modern Altera devices.

On the other hand, Xilinx devices do not provide a 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 [11][12][13], however 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][15].

Lattice ispXPLD devices [16] 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 [17] recommend using multi-cycle CAMs by searching BRAM in parallel. For a single-cycle CAM, using registers is suggested.

III. Segmented Transposed Indicators RAM Deep BCAMs

In this section, the proposed segmented transposed indicators RAM (STIRAM) method to generate deep BCAMs is described in detail. The motivation and method of this work is explained in subsection A. The construction of wide priority encoders in FPGAs, which are crucial for BCAMs in general, is described in subsection B. BCAM bypassing techniques are described in subsection C.

A. *Segmented Transposed Indicators Approach*

The proposed Segmented Transposed Indicators (STIRAM) approach is a modular BRAM-based BCAM approach suitable for deep applications. The address range is divided into equal size segments; each segment is a very wide word that stores a set number of patterns. BCAM pattern lookup works in two stages. First, a RAM structure is indexed by the pattern; it stores match information for each pattern. This can be used by a priority encoder to identify the address of a segment containing the pattern; it also produces the upper bits of the match address. The address indexes into a second RAM structure to produce the segment. Second, the wide segment is searched in parallel for a match to the pattern; the location of the match within the segment produces the lower bits of the match address.

While cascadable BCAMs requires indicators from every address location at every stage, this requirement can be alleviated if the BCAM will not be cascaded. Instead of storing pattern indicators for each address location separately, an indicator is generated for a group of addresses, indicating if the pattern exist at any of these addresses. An address segment of width S_W is a group of successive S_W addresses, hence an address range of width S_W. For a C_D-lines BCAM, and S_W segments width, $\lceil C_D/S_W \rceil$ segments exist, and can be enumerated as

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

A segment indicator $I_{p,s}$ indicates if any of the addresses in segment s, hence addresses $S_W \cdot s$ upto $S_W \cdot (s+1)-1$ contains the pattern p, namely,

$$\forall s \in S, p \in P\colon\ I_{p,s} = \bigvee_{a=S_W \cdot s}^{a=S_W \cdot (s+1)-1} (RAM[a]\ EQ\ p). \quad (6)$$

Were P is the patterns set. The segmented transposed indicators RAM (STIRAM) is therefore

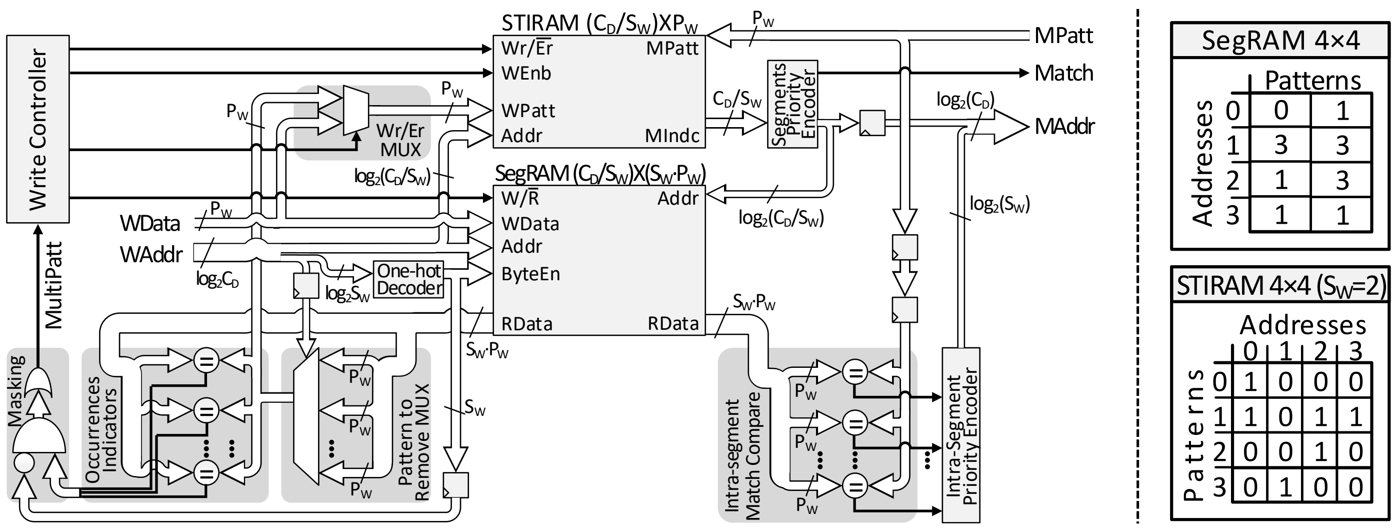
$$STIRAM = \begin{bmatrix} I_{0,0} & I_{0,1} & \cdots & I_{0,max\,(S)} \\ I_{1,0} & I_{1,1} & \cdots & I_{1,max\,(S)} \\ \vdots & \vdots & \ddots & \vdots \\ I_{max(P),0} & I_{max(P),1} & \cdots & I_{max\,(P),max\,(S)} \end{bmatrix}. \quad (7)$$

STIRAM provides information for matched patterns within a segment, not the exact location. To detect the exact pattern location, an auxiliary RAM stores the patterns associated with each segment, each segment's patterns in one RAM address, hence it called the segments RAM (SegRAM). If a match is found in a specific segment, this segment will be fetched from the SegRAM and patterns within this segment are compared concurrently to match pattern.

Fig. 6 illustrates the complete STIRAM system. Two RAM structures are required, the first is STIRAM, a transposed RAM, addressed by patterns and stores segments indicators. The second is SegRAM, stores data patterns, each segment's patterns in one RAM line.

The match operation checks STIRAM for a match within a segment, detects the first matching segment using a priority encoder, then fetches the corresponding segment patterns (with a match) from SegRAM, then compares all patterns with the match pattern in parallel to detect the exact match location. The match operation is described in details as follows.

1. Detect match among segments:
   - 1.1. MPatt is provided to STIRAM for search.
   - 1.2. STIRAM detects which segments contains MPatt (One-hot MInd)
   - 1.3. Segments priority-encoder (PE) generates the binary address for the first segment with a matching pattern. This address composes the higher part of MAddr.
   - 1.4. Segments PE also generates the binary Match signal, which indicates that a match was found.
2. Detect match exact location within the segment:
   - 2.1. The address of the first matched segment (step 1.3) is provided to the SegRAM to fetch the entire segment.



Addresses	Patterns	
0	0	1
1	3	3
2	1	3
3	1	1

Patterns \ Addresses	0	1	2	3
0	1	0	0	0
1	1	0	1	1
2	0	0	1	0
3	0	1	0	0

Fig. 6. (left) The complete STIRAM BCAM (right) 8×2; S_w=2 example

2.2. Each pattern within the segment is compared to MPatt

2.3. A priority-encoder (intra-segment PE) detects the first matching pattern. The address of the first matching pattern composes the lower part of MAddr.

While detecting a match is completed by reading the STIRAM in one cycle, computing the exact match address requires another cycle to read the SegRAM. Hence, the match operation latency is two cycle. The match operation throuput is a single cycle since both STIRAM and SegRAM are read concurrently.

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. However, before clearing the old data indicator in the STIRAM, the segment should be checked to detect other occurrences of the old data. If other occurrences of the deleted pattern found in the same segment, the segment indicator for this pattern in the STIRAM shouldn't be cleared. In details, the STIRAM BCAM writing is performed as follows.

1. Cycle 1: STIRAM write; SegRAM read

1.1. Write STIRAM with WPatt and the higher $\lceil \log_2(C_D/S_W) \rceil$ bits of WAddr to set the corresponding segment indicator.

1.2. Read the entire corresponding segment for SegRAM (addressed by the higher $\lceil \log_2(C_D/S_W) \rceil$ bits of WAddr)

1.2.1. The "Pattern to remove MUX" selects the pattern that is being rewriting from the corresponding segment. The selector is the lower $\lceil \log_2 S_W \rceil$ bits of WAddr.

1.2.2. The "Occurrences Indicators" are a compare of the currently rewritten pattern with all the other patterns in the segment to detect other occurrences.

1.2.3. The final masking stage masks the indicator of the currently rewritten pattern, since only other occurrences should be detected. All the indicators are OR'ed to detect any other occurrence

2. Cycle 2: SegRAM write; STIRAM conditional erase

2.1. The SegRAM is written with WPatt. Byte-enable is used to write only the corresponding pattern in the segment.

2.2. If no other occurrences of the currently rewritten pattern are detected (stage 1.2.3 above), the STIRAM indicator for the replace pattern and the current address is cleared.

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

$$\left\lceil \frac{C_D}{S_W} \right\rceil \cdot P_W \cdot S_W + 2^{P_W} \cdot \left\lceil \frac{C_D}{S_W} \right\rceil. \tag{8}$$

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

$$\left\lceil \frac{2^{P_W}}{R_{D,min}} \right\rceil \cdot \left\lceil \frac{\lceil C_D/S_W \rceil}{R_{W,max}} \right\rceil. \tag{9}$$

The SegRAM is a true-dual-port RAM. $R_{W,byte}$ describes the width of data controlled by a single btye-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 SegRAM is

$$\frac{R_{W,max}}{R_{W,byte}} \cdot \left\lceil \frac{P_W}{R_{W,byte}} \right\rceil \cdot \left\lceil \frac{S_W}{R_{D,min}} \right\rceil. \tag{10}$$

B. *Wide Priority Encoders in FPGAs*

The proposed STIRAM technique successfully reduces the width of the C_D wide priority-encoder used by the brute-force approach and splits it into two narrower priority-encoders. However, priority-encoder delay still affects overall performance. Hence, a fast priority-encoder design is essential.

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.

As depicted in Fig. 7, 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 employed 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.

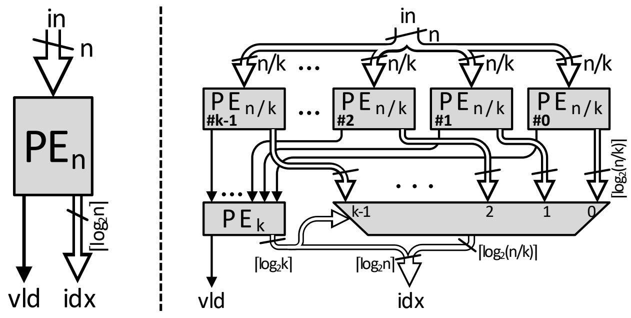


Fig. 7. Priority-encoder (left) symbol (right) recursive definition

C. BCAM Bypassing

In a register-based BCAM, writing a new pattern 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.

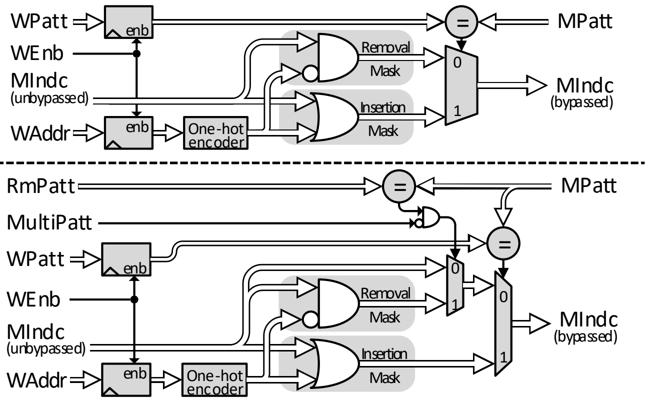


Fig. 8. Bypassing logic (top) brute-force TIRAM (bottom) proposed STIRAM

Fig. 8 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 segment (negated MultiPatt).

IV. Comparison and Discussion

The BCAM area efficiency μ_a is first introduced in this paper, and is defined as the SRAM cell utilization for a specific BCAM implementation. In other words, μ_a is the ratio between the total BCAM bits and the total SRAM cells used to implement this BCAM. Using (2), (4) and (8), μ_a is estimated as follows.

$$\mu_a(UTI) \approx \frac{1}{1+\frac{2^{P_W}}{P_W}}; \mu_a(CTI) \approx \frac{1}{1+\frac{R_{D,min}}{\log_2(R_{D,min})}}; \mu_a(STI) \approx \frac{1}{1+\frac{2^{P_W}}{S_W \cdot P_W}} \quad (11)$$

Equation (11) shows that the area efficiency of the uncascaded TIRAM implementation (UTI) is inversely proportional to the exponent of P_W, hence, decays rapidly with P_W increase. The cascaded TIRAM (CTI) is not related to P_W, hence the efficiency is not affected when P_W increases. However, the efficiency is affected by 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 area efficiency. Similar to the uncascaded TIRAM (UTI), the efficiency of the proposed approach (STI) is inversely proportional to the exponent of P_W. However, the exponential relation is mitigated by the segment width S_W, an external and user-driven parameter. For example, with $S_W = 4K$, $R_{D,min} = 512$ (as in Altera's M20K), and a pattern width of $P_W = 12$, the area efficiency of the proposed STIRAM is $\mu_a = 0.923$ while using the C-TIRAM approach provides $\mu_a = 0.017$. The STIRAM approach provides the highest efficiency up to $P_W = 22$. To generalize, STIRAM is superior 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).

Practically, the segment width S_W has a limitation due to the minimal width of the RAM it stored in (SegRAM). The SegRAM depth is $\lceil C_D/S_W \rceil$ and is limited by $R_{D,min}$, hence, to achieve maximum area efficiency, S_W is bounded by

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

For deep memories, S_W can be set to higher values, allowing higher area efficiency. Furthermore, providing shallow and wide BRAM, namely a lower $R_{D,min}$, will allow higher S_W values, hence a higher area efficiency.

To reduce ALM consumption it is recommended to divide the priority-encoders in the STIRAM design equally, hence, $S_W \approx \sqrt{C_D}$. Each priority encoder will be of width $\sqrt{C_D}$. Furthermore, dividing the priority-encoders equally will balance the priority-encoders depth, hence increasing Fmax and reducing the maximum pipe stages in the pipelined design.

V. Experimental Results

To verify and simulate the suggested approach and compare to standard techniques, fully parameterized Verilog modules have been developed. Register-based, cascaded and uncascaded brute-force TIRAM, and the proposed STIRAM BCAM methods have been implemented. To simulate and synthesize these designs with various parameters in batch using Altera's ModelSim and Quartus II, a run-in-batch flow manager has also been developed. 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. bypassing, depth, pattern width, and segment width, are swept and simulated in batch, each with over million random cycles. All different BCAM design modules were implemented using Altera's Quartus II on Altera's Stratix V 5SGXMA7N1F45C1 device. This is a high-performance device with 235k ALMs and 2560 M20K blocks.

Fig. 9 plots feasible BCAM depth and pattern width sweeps implemented on Altera's Stratix V device. Within the device limitation, the proposed STIRAM approach is able to reach 4M lines of CAM, while the brute-force TIRAM and the register-based approach cannot exceed 64K and 32K lines, respectively. The number of Altera's M20K blocks used to implement each BCAM configuration is plotted in Fig. 9 (a). Even for shallow memories of 64K and 32K, the proposed approach demonstrates lower BRAM consumption compared to other methods. The columns in Fig.9 (a) show the BCAM consumption of the two RAM structures that compose the STIRAM approach; the SegRAM and the STIRAM. The SegRAM stores the patterns themselves; hence, if the SegRAM BRAM consumption is dominating the STIRAM consumption, the area efficiency will be higher.

The proposed STIRAM method exhibits significantly lower ALM count and higher Fmax due to splitting the priority-encoder as shown in Fig. 9 (b and c). Register-based BCAM consumes the highest ALMs due to massive register usage. To achieve high Fmax, all testcases are fully pipelined. Pipelining and BRAM access latency increase the overall system latency in both traditional and proposed approaches. Brute-force TIRAM approach has a lower latency by maximum one cycle in some shallow testcases. The latency of both approaches in these shallow testcases is 9 or 10 cycles. The growth of latency as depth increases is logarithmic, since PE depth is logarithmic. The latency of the deepest 4M-lines STIRAM is 13 cycles.

Fig. 9 (d) plots the optimal segment width. As the depth increases, optimal segment width is larger. Similar with pattern width; if pattern width increases, optimal segment width increases to overcome the increase in STIRAM BRAM consumption to wide patterns.

The area efficiency is plotted in Fig. 9 (e). The STIRAM overcomes the brute-force TIRAM in 32K and 64K CAMs. Furthermore, area efficiency is inversely related to pattern width as expected from STIRAM. As STIRAM goes deeper, the efficiency increases, the 4M×9 BCAM test case demonstrates a high area efficiency of 0.9.

Fig. 9 (f) plots the full Quartus II flow runtime. STIRAM synthesis is faster than register-based or TIRAM. STIRAM synthesis runs up to 3 hours in its largest 4M testcase.

VI. Conclusions and Future Work

In this paper, a novel BCAM architecture for FPGAs is proposed. The approach is fully BRAM-based and employs segmentation to reduce BRAM consumption. While traditional brute-force approach have a pattern match indicator for each single address, the proposed approach maintains a single pattern match indicator for each address segment. The suggested method is capable of implementing a 4M-line CAM and significantly improves area and performance. In contrast, traditional methods cannot exceed 64K in depth. The suggested STIRAM BCAM design *completely dominates* all past designs in BRAM and ALM consumption, Fmax and runtime.

A fully parameterized and 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. 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 [18]. 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 [19].

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

[4] 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.

[5] 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.

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

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

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

[9] 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.

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

[11] J.-L. Brelet, "An OvervieW of Multiple CAM Designs in Virtex Family Devices," Application Note XAPP201, 1999, Xilinx, Inc., San Jose, CA.

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

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

[14] J.-L. Brelet, "Using Block RAM for High Performance Read/Write CAMs," Application Note XAPP204, 2000, Xilinx, Inc., San Jose, CA.

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

[16] "Content Addressable Memory (CAM) Applications for ispXPLD Devices," Application Note AN8071, July 2002, Lattice Semiconductor Corporation, Hillsboro, OR.

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

[18] 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)*, December 10-12, 2012, Seoul, South Korea.

[19] 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.

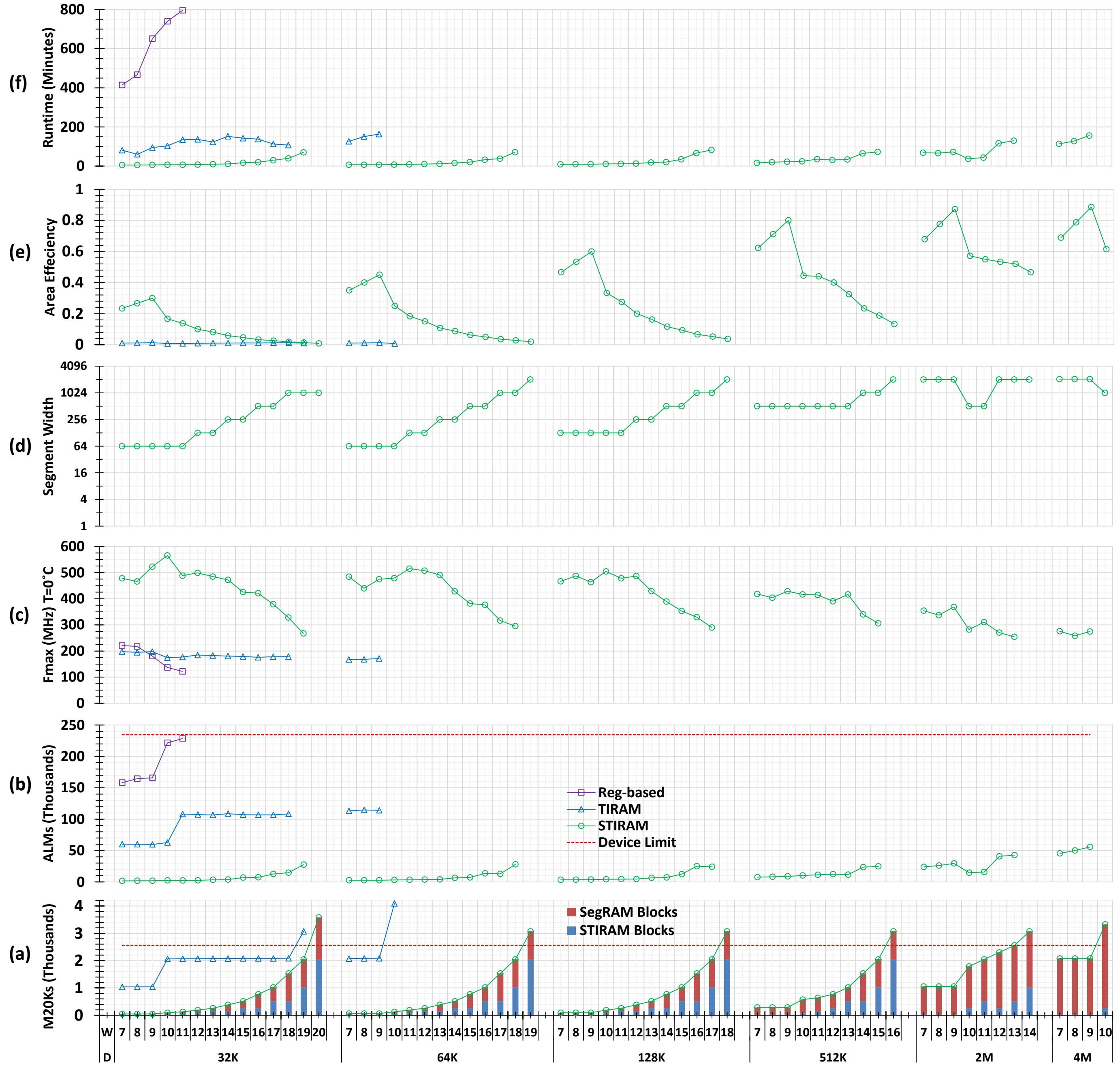


Fig. 9. Results for several BCAM depth and pattern width sweeps (a) M20K count (b) ALMs count (c) Fmax (d) Segment width (e) Area effeciency (f) Runtime