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 a wide range of applications, such as memory management, networks, data compression, DSP, and databases. 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 deep and narrow BCAMs out of standard SRAM blocks in FPGAs is proposed. This technique is most efficient for deep and narrow CAMs since the BRAM consumption is exponential to pattern width. Using Altera's Stratix V device, traditional methods achieve up to 64K-entry BCAM while the proposed technique achieves up to 4M entries. For the 64K-entry test-case, traditional methods consume 43 times more ALMs and achieves only one-third of the Fmax. A fully parameterized Verilog implementation is available[1]. This implementation has been extensively tested using Altera's tools.**

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 the specific given data, known as search pattern. Hence performing a memory-wide search for a specific value. BCAMs are massively parallel search engines accessing all memory content to compare with the search pattern simultaneously.

A BCAM is actually a high-performance implementation of a very basic and widely used associative search, hence it can be used in every science field requiring high-speed processing of associative search. Networks, associative caches and TLBs, pattern matching, data compression, DSP, bioinformatics, and a variety of other scientific fields can 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 [5] 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 described in this paper as the traditional or transposed-indicators RAM approach.

In this paper, a modular SRAM-based BCAM suitable for deep and narrow search 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) are stored in a second RAM structure.

The proposed method is device-independent and dramatically improves CAM area efficiency and operation frequency compared to conventional methods. In contrast to other attempts that require several cycles to write or match [3][4], the proposed approach is high-throughput and can perform a pattern match every single cycle and a pattern write every two cycles.

A parameterized Verilog implementation of the suggested methods and previous standard approaches, together with a flow manager are available online[1]. To verify correctness it is simulated using Altera's ModelSim, and compiled using Quartus II. A large variety of BCAM architectures and parameters are simulated in batch. Stratix V, Altera's high-end performance-oriented FPGA, is used to implement and compare the proposed architecture with previous approaches.

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 section III. The experimental framework, simulation and synthesis results, are discussed in section V. Conclusions are drawn in section VI.

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. A review of FPGA vendors' supports of BCAMs is placed in subsection C.

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

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 (PE) to detect the first matching address (MAddr). The high demand for limited resources such as registers, comparators, address decoder and PE (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.

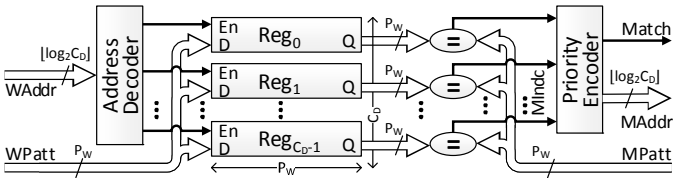


Fig. 1. Register-based BCAM

B. Brute-force Approach via Transposed-Indicators RAM

The RAM-based brute-force approach address the RAM by the match pattern while each bit of the RAM data indicates the existence of the pattern in 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).

The complete system of the brute-force TIRAM approach is described in Fig. 2. 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. A PE detects the first match address (MAddr) from the match indicators. However, rewriting the TIRAM requires more computation since it requires setting the new indicator and clearing the old indicator. As shown in Fig. 2, an ErRAM (Erase RAM) is used in parallel to the indicators RAM (TIRAM) in order to provide for a given address what pattern is already stored and should be removed. This is used for writing where the old indicator should be cleared; ErRAM will provide the old pattern in the current written address. Hence, the BCAM writing consumes 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

$$C_D \cdot (P_W + 2^{P_W}). \quad (1)$$

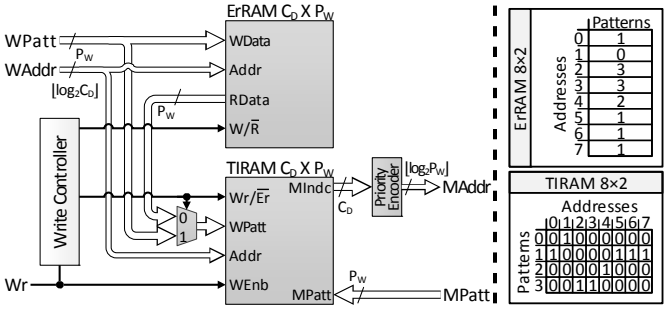


Fig. 2. (left) Brute-force TIRAM approach (right) 8×2 example

C. 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 [5] device family integrates intrinsic BCAM support into their embedded system blocks (ESBs). The ESB can be configured into a 32×32 BCAM. Since ESBs are limited to few hundred blocks in these devices, and due to routing complexity, 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 [8]. This approach is very slow at writing new patterns. Xilinx application notes also suggest utilizing BRAMs with the brute-force approach to create BCAMs [8].

Lattice ispXPLD devices [9] have an integrated support for CAMs via their Multi-Function Block (MFB) which can be configured into a 128×48 Ternary CAM block (with don't care values). Alternatively, Actel application notes [10] 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

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 is used by a PE 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 require 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 exists at any of these addresses. A segment of width S_W holds a number of patterns in fixed positions. For a C_D-lines BCAM, and S_W segments width, $\lceil C_D/S_W \rceil$ segments exist. A segment indicator indicates if a segment contains the corresponding pattern in any of its positions.

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. 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 all patterns within this segment are compared concurrently to match pattern. Fig. 3 illustrates the complete STIRAM system. Two RAM structures are required, the first is STIRAM, a transposed RAM, addressed by patterns and storing segment indicators. The second is SegRAM, storing data patterns, with all patterns from a segment stored in one RAM line.

The match operation checks STIRAM for a match within a segment, detects the first matching segment using a PE, 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 detail as follows:

1. Detect match among segments:
   1.1. STIRAM detects which segments contains MPatt (MInd)
   1.2. The segments PE generates the binary address for the first segment with a matching pattern. This address composes the higher part of MAddr.
   1.3. Segments PE also generates the 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 1.2) is provided to the SegRAM to fetch the entire segment.
   2.2. Each pattern within the segment is compared to MPatt.
   2.3. The Intra-segment PE) detects the first matching pattern. The address of the first matching pattern composes the lower part of MAddr.

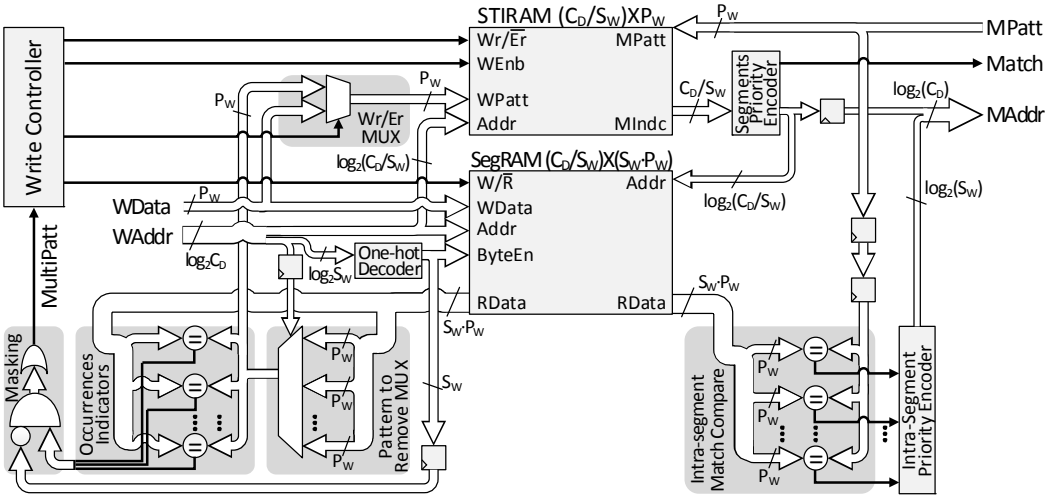
While detecting a match is completed by reading the STIRAM in one cycle, computing the exact match address requires another cycle to read the SegRAM. Hence, the match operation latency is two cycles. However, the throughput is one cycle since both STIRAM and SegRAM are read concurrently.

Similar to the brute-force approach, writing to the STIRAM 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, other occurrences of the old data should be checked. If other occurrences of the deleted pattern are found in the same segment, the segment indicator for this pattern in the STIRAM shouldn't be cleared. In detail, 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 from 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 rewritten from the 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 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 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

$$\lceil C_D/S_W \rceil \cdot (P_W \cdot S_W + 2^{P_W}). \quad (2)$$



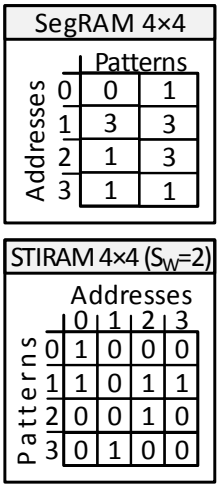


Fig. 3. (left) The complete STIRAM BCAM (right) 8×2; S_w=2 example

IV. Experimental Results

To verify and simulate the suggested approach and compare to standard techniques, fully parameterized Verilog modules have been developed, including register-based, brute-force TIRAM, and the proposed STIRAM method. 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 been developed. The Verilog modules and the flow manager are available online [2]. 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 using Altera's ModelSim. 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. 4 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. 4 (bottom). Even for shallow memories of 64K and 32K, the proposed approach demonstrates lower BRAM consumption than the TIRAM method. The columns in Fig. 4 (bottom) 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 PE as shown in Fig. 4 (middle and top). Register-based BCAM consumes the highest ALMs due to massive register usage.

V. Conclusions

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, our approach maintains a single pattern match indicator for each segment, where a segment holds a number of patterns. While traditional methods cannot exceed 64K-line, our method is capable of implementing a 4M-line CAM and significantly improves area and performance. However, the proposed approach is not cascadable, leading to an exponential growth of BRAM consumption as pattern width increases; hence it is most efficient for deep and narrow CAMs where it completely dominates all past designs in area and Fmax.

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] 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.

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

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

[6] F. Heile, A. Leaver, and K. Veenstra, "Programmable memory blocks supporting content-addressable memory*," in Proc. of the ACM/SIGDA symposium on Field programmable gate arrays*, 2000, Monterey, CA.

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

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

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

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

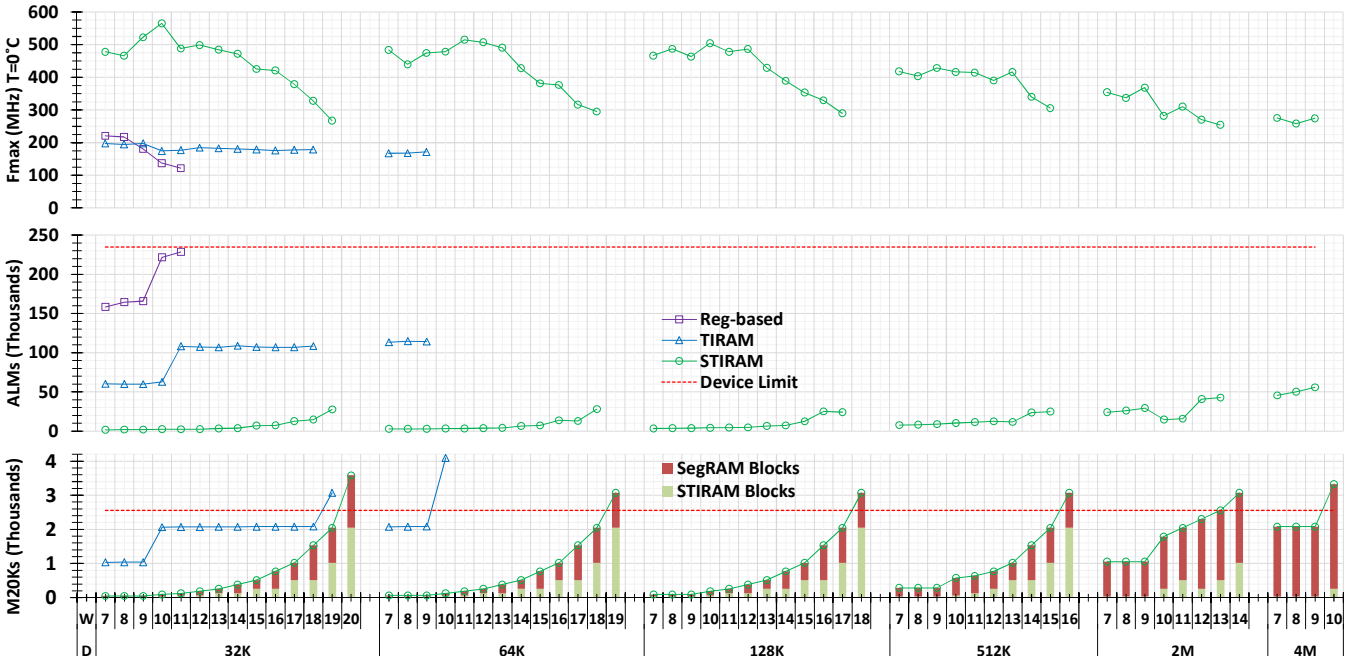


Fig. 4. Results for several BCAM depth and pattern width sweeps (bottom) M20K count (1000's) (middle) ALMs count (1000's) (top) Fmax (MHz) T=0°C