Adapting an Industrial Memory BIST solution for testing CAMs

Jais Abraham, Uttam Garg
Intel, Bangalore, India

Glenn Colon-Bonet,
Intel, Fort Collins, USA

Ramesh Sharma, Chennian Di
Intel, Santa Clara, USA

{jais.abraham, uttam.k.garg, glenn.t.colon-bonet, ramesh.sharma, chennain.di}@intel.com

Benoit Nadeau-Dostie, Etienne Racine
Mentor Graphics, Canada

Martin Keim
Mentor Graphics, USA

{benoit_nadeau-dostie, etienne_racine, martin_keim}@mentor.com

Abstract

Content Addressable Memories (CAMs) have found widespread use in applications that require high speed search capabilities. Each cell in the CAM array is associated with a storage unit and a comparator logic. Due to the various customized features in the CAM implementations, creation of an automated BIST solution for testing them has presented unique challenges. This paper shows that, with suitable modifications to the CAM test collar, an existing BIST solution and flow traditionally used to test embedded SRAMs can be used to test the CAMs.

Keywords: Binary CAM, Ternary CAM, BIST

1. Introduction

Content Addressable Memories (CAMs) have been employed widely for high speed search operations in applications such as networking and cache look up [1][2][3][4][5][6]. Compared to RAMs, which return the data based on the address provided to it, the CAMs compare the incoming data with the keys present in the array and return the address(es) of the matching entries in the array. A "hit" or "match" is said to occur when the incoming compare data can be found in the CAM array. Thus, the CAM cells have a comparator logic in addition to the storage element. CAMs are usually followed by a priority encoder and a payload RAM. The priority encoder resolves multiple matching entries in the array to the highest (or lowest) matching address and in many high speed implementations, the match lines serve as fully decoded address to the payload RAM.

CAMs are usually classified as Binary CAMs (BCAM) and Ternary CAMs (TCAM). Quaternary CAMs have recently been reported [7]. BCAMs generate a hit if the complete entry is matched. On the other hand, the Ternary CAMs (TCAM) provide the capability to match a part of the entry, while treating the remaining part as don't cares, which are specified using additional mask bits. TCAMs need an extra storage array to store the mask bits. In addition, different TCAM implementations provide the capability to mask specific bits of an entry or parts of the entries or the complete entry itself.

Several CAM BIST solutions have been presented in the past. Representative examples can be found in [8-11]. In our experience, CAM designs often include new features or features that are implemented differently in a way that would affect the BIST implementation and its automation. The contribution of this work is to demonstrate how an existing memory BIST solution for embedded SRAMs can be used to test CAMs in that context. Although the solution in this paper describes a particular CAM design, it will become clear that the solution can easily be extended to other implementations as well. The paper explains two exemplary CAM algorithms, providing insight into how these algorithms differ from those for SRAMs. In practice, variations of these algorithms should be considered, based on the presence or absence of specific CAM features. However, the treatment on the algorithms is not exhaustive and other works on CAM test [8-11] can be referred for this purpose. The only requirements for the existing BIST tool is to have a powerful algorithm programming capability used in combination with flexible operation definition of user-defined control signals.

The paper is organized as follows. Section 2 discusses the architecture of CAMs and provides a description of the generic ports that are available on the CAMs. Section 3 provides an overview of the fault models considered for testing the CAM comparator logic and the algorithms used to test them. Section 4 discusses the details of an example custom CAM collar. Section 5 discusses the differences in the design flow compared to the one used for RAM BIST. Section 6 concludes the paper.

2. CAM Architecture

The logical view of a Binary CAM array is shown in Fig. 1. The Binary CAM can be conceptualized as having two functions: storage and compare. The storage function behaves like a RAM and needs to be tested for the faults in the address decoder, storage cells and the read/ write circuitry. Associated with every bit is also a comparison logic. The result of the comparison is calculated across the complete entry to generate a match/hit.

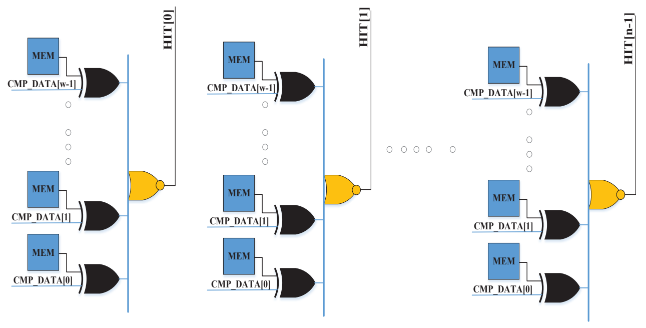

Figure 1: Binary CAM Architecture

The Ternary CAM is an extension of the Binary CAM. It has additional entries to store the mask bits and logic to provide the mask functionality. A simple implementation of the TCAM based on the data/ mask encoding is shown in Fig 2.

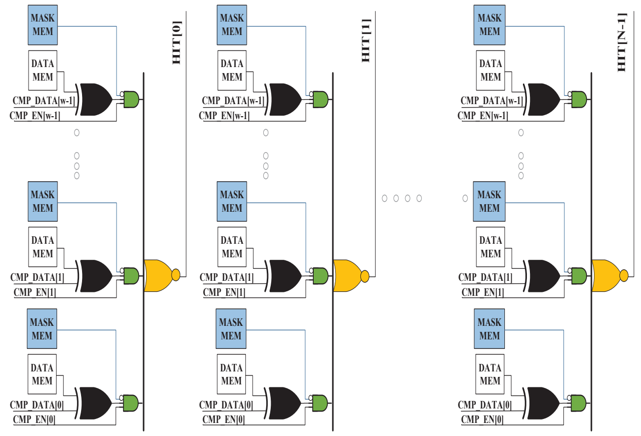

Figure 2: Ternary CAM Architecture

The TCAM has a global mask (CMP_EN) that applies to every entry. In more complex implementations, each bit of the global mask and internal mask bits could be applied to group of bits of an entry in an irregular manner i.e. each group could be of different size. Internal mask bits can't be equated to normal write masks in SRAMs as they are a function of the address; they can be shared or not by multiple addresses. Other features such as a valid bit per entry, bank mask bits, etc. makes the automated generation of the test logic more difficult.

3. CAM Fault Models & Algorithms

The CAM compare logic can be implemented using static or dynamic CMOS logic. A slice of the match logic implementation with static CMOS logic is shown in Fig. 3. In this implementation, the following faults are investigated for detection by the CAM test algorithm: stuck-at 0 & stuck-at1 faults and the slow-to-rise & slow-to-fall faults at the *m, d, c & h* nodes. Coupling faults are not considered in this discussion.

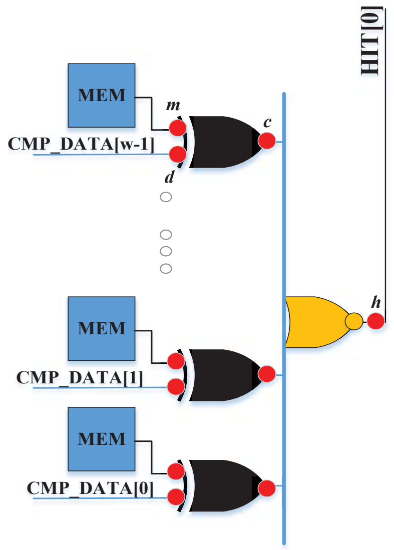

Figure 3: Fault locations in a Binary CAM static CMOS structure

Fig 4 shows the CAM compare logic implemented using dynamic logic. BL & $\overline{\text{BL}}$ are the bit lines of the storage cell and CD & $\overline{\text{CD}}$ are the bit lines for the compare. The match line is pre-charged prior to a compare operation. The stuck-on and stuck-open faults on the transistors T1, T2, T3 & T4 are considered for detection.

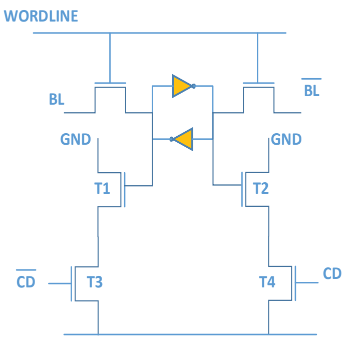

Figure 4: Dynamic CMOS implementation of compare logic

We describe two algorithms that can detect the faults on the CAM compare logic. In the descriptions of the algorithms, W refers to the width of the CAM array and N is the number of entries in the CAM array.

3.1 Walking 0(/1) algorithm

The walking 0(/1) algorithm fills the CAM array with a particular background (D) and then walks a 0(/1) on the compare data. The walking 0(/1) algorithm performs operations described in short notation below.

^wD1; >cWS2; ^wI3; >cWS4;

Where:
Operation (^wD1) initializes array with D
Operation (>cWS2) performs the following three steps for each bit position (b) of word width (W)

1. *set CMP_DATA = D; expect match on every entry*
2. *set CMP_DATA= D xor (2 ^ b); expect mismatch on every entry*
3. *set CMP_DATA = D; expect match on every entry*

Steps 1 and 2 are performed at-speed. The operations described above are repeated with inverted background

The algorithm detects the *m, d, c, h* stuck-at 0 & stuck-at 1 faults & *d, c, h* slow-to-rise & slow-to-fall faults in the static CMOS implementation. In the dynamic CMOS implementation, it detects the presence of stuck-open faults in transistors T1, T3 when the background D is set to FF, and in transistors T2, T4 with a background set to 00.

3.2 Unique Match algorithm

The unique match algorithm fills the array with a particular background. It then inverts each entry and creates a match for the entry. The algorithm performs the operation described in following short notation:

^wD1; ^cD2 ; ^wI3 ; ^cI4

Where:
Operation (^wD1) *initializes whole array with a solid background (D)*
Operation (^cD2) performs following steps for each entry of the array while CMP_DATA has been set to inverted word-line storage data.

1. *Write inverted D to the Entry. Expect match on the line*
2. *Write (D) to the Entry. Expect mismatch across the entire array*

Both steps 1 and 2 are performed at-speed. The operations described above are repeated with inverted background.

The unique match algorithm additionally detects the *m* slow-to-fall & slow-to-rise fault. In the dynamic CMOS implementation, it detects the stuck-on fault on each transistor.

An extension of these algorithms can be found to detect the faults in the mask logic of TCAMs. The description of the TCAM algorithms is, however, beyond the scope of this paper and other works on this topic can be referred for that purpose [10-11]. The algorithms above, although not comprehensive, have been presented to provide an insight into why specialized hardware is required to test the CAMs, as described in the next section.

4. Custom CAM collar

The variations in the BCAM & TCAM architectures make it difficult to create a generic BIST hardware for testing them. However, with the addition of test hardware in the CAM collar, it can be tested with solutions existing for RAM-BIST by making the CAM appear like a 2-port SRAM to the memory BIST controller. The main components of this hardware are a shift register, an expected data generator and a compactor/expander. These components are shown in Fig. 5 and explained below:

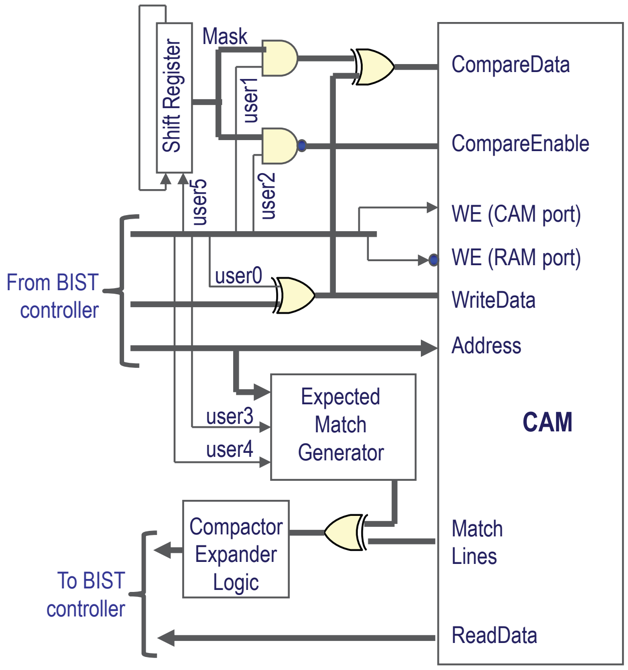

Figure 5: Custom CAM collar

Shift Register: The walking 1/(0) algorithm requires the creation of a one-hot (/ one-cold) pattern. Therefore a shift register is created in each CAM collar which has a size equal to the width of the CAM being tested. This also makes it difficult to share this hardware across different CAMs in the design.

Expected Data Generator: Based on the algorithms described in Section 3, the CAM hit output can generate a match on all entries, or a mismatch on all entries, or a unique match on only one entry. The expected data generator is a combinational decoder that creates one of the above expected outputs based on inputs that indicate the algorithm being exercised on the CAM. For the unique match, the expected data generator creates the expected data from the address lines that are provided by the BIST engine.

Compactor / Expander Logic: A unique compare result is available at every match line of the CAM array, thereby generating a total of "N" compare bits. However, the BIST controller is to be provided a compare vector equal to the width of the array. Hence, the compare result has to be suitably compacted or expanded before it is sent back to the

BIST controller. In cases, where the number of entries in the CAM is greater than its width (i.e. N > W), the compare results are ORed to provide a compacted compare vector to the BIST engine. In the case where the number of entries is less than the width of the array (i.e. N < W), the remaining bits are tied off to '0'. This logic, though simple to implement, has limitations in providing diagnostic accuracy for low yield debug.

4.1 Support for Improved Diagnosis

To improve the diagnostic resolution of failures in the comparator, the compactor/expander logic is replaced by the logic shown in Fig. 6. The BIST_MASK bus is the output of the shift register of Fig. 5 used to generate the Walking 1 pattern for the compare data or compare enable CAM inputs. The mask is applied to test comparators located in a same column or even individual comparators when the BIST_CD_MASK_ENABLE (compare data mask enable) or BIST_CE_MASK_ENABLE (compare enable mask enable) signal is enabled. The value of the mask itself and enable signals are pipelined and gated together to generate a mask for the CAM collar output bus. There is at least one level of pipelining for clocked compare ports which is the most common case. Additional pipeline stages are required if the match line outputs are pipelined. When at least one of the match line output doesn't generate the expected output, the MatchError signal goes high. If the compare operation was not performed using the Walking 1 mask, then all CAM collar outputs take the value of 1. However, if the mask was used, then only the bit of the CAM collar output corresponding to the column being tested will take the value 1.

This modification allows to use a conventional RAM Built-In Repair Analysis (BIRA) module for repairing columns based on the results of both the RAM and CAM ports.

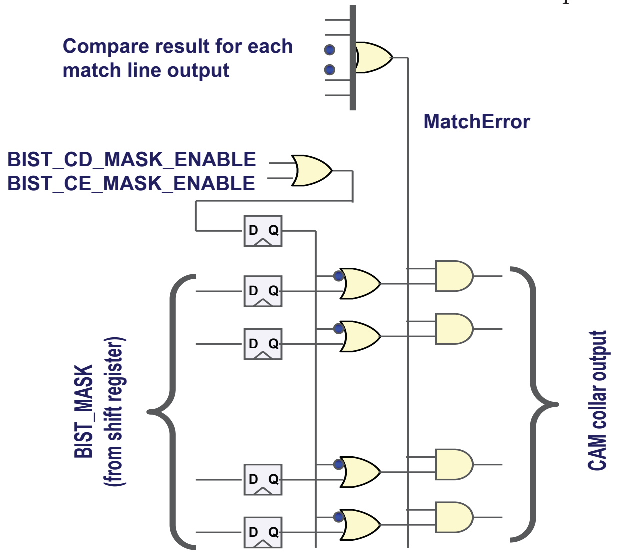
Compare result for each match line output
MatchError
BIST_CD_MASK_ENABLE
BIST_CE_MASK_ENABLE
D Q
BIST_MASK (from shift register)
CAM collar output

Figure 6: Hardware for improved diagnosis and repair

For bit-level diagnosis, there will still be some ambiguity about the address that caused an error if the algorithm is such that a match or mismatch can occur for multiple entries. This is because the standard RAM diagnosis tool doesn't know if the address is considered or not in the generation of the expected values for the match lines. Algorithms must be chosen to only select operations that will cause a unique match or mismatch to avoid potentially ambiguous diagnosis reports.

```
MicroProgram {
 Instruction ( InitializeRAM ) {
   BranchToInstruction : InitializeRAM;
   OperationSelect       : WriteCAM;
   WriteDataCmd         : Zero;
   X1AddressCmd         : Increment;
   NextConditions {
      X1_EndCount      : On;
   } // end NextConditions
 } // end instruction InitializeRAM
Instruction ( WalkingBit ) {
 BranchToInstruction  : WalkingBit;
 OperationSelect            : CfgCbgCfg;
 WriteDataCmd               : Zero;
 ExpectDataCmd              : Zero;
 CounterACmd                : Increment;
    NextConditions {
        CounterAEndCount    : On;
    } // end NextConditions
} // end instruction WalkingBit
Instruction ( WalkingAddress ) {
 BranchToInstruction    : WalkingAddress;
 OperationSelect           : WfgCfgWbg;
 WriteDataCmd              : Zero;
 ExpectDataCmd             : Zero;
 X1AddressCmd              : Increment;
 NextConditions {
    X1_EndCount               : On;
    RepeatLoop (A) {
       BranchToInstruction  : InitializeRAM;
      Repeat {
         WriteDataSequence : Inverse;
      } // end repeat
    } // end repeatLoop
   } // end NextConditions
  } // end instruction WalkingAddress
} // end MicroProgram
```

Figure 7: Microprogram example

4.2 Operation of the CAM collar

Several "user bits" are added to the CAM collar to steer the operation of the hardware under different algorithms. These user bits allow driving inputs to the custom CAM wrapper with arbitrary waveforms, under full control of the algorithms. Fig. 7 shows how the two algorithms described in section 3 (Walking 0/1 and Unique Match) could be coded in the same microprogram which can be either hard coded in the BIST controller or loaded in the controller at run time using the existing capabilities of [12].

The core portion of each algorithm only requires a single instruction of the microprogram shown in Fig. 7. Each instruction calls a complex operation. The three complex operations used by this microprogram are shown in Fig. 8. The waveforms show the behaviour of the signals of Fig. 5. Some control signals are not standard RAM control signals like write enable, read enable or select. Those are labelled user0 to user5 in Fig. 5 and 8. They are generated by the BIST controller and used in the custom CAM collar to generate the WriteData, CompareData and CompareEnable CAM inputs.

The first instruction (InititializeRAM) makes use of the WriteCAM operation to initialize all entries with a background of 0s in the first iteration of the microprogram. The WalkingBit instruction makes use of the CbgCfgCbg complex operation which performs 3 compare operations on all entries. In the first cycle, the compare data (CompareData) is the same as the background data used to initialize the CAM and all hit lines are checked to be active by setting the AllMatch (user3) input of the expected match generator to 1. In the second cycle, the compare data differ from the background data by one bit due to the application of the mask contained in the shift register which was initialized with a single 1. The mask is applied to the compare data by setting CDMaskEnable (user1) to 1. There should be no match in these this cycle i.e. all hit lines are checked to be inactive. The third cycle is the same as the first one and the hit lines are checked to be inactive. The third cycle is also used to shift the mask by one position in the shift register. This is done by setting RotateMask (user5) to 1. The instruction is repeated for a number of times determined by the number of bits in an entry.

Note that inputs to the expected match generator need to be pipelined by at least one when the CAM has a clock input. Match lines could however be pipelined which adds to the latency. The operation set definition remains the same in all cases except that some control signals can be identified as requiring pipelining. The pipeline stages are added when the BIST controller is generated.

The WalkingAddress instruction makes use of the WfgCfgWbgCfg complex operation. For each address, the BIST engine first writes a foreground pattern obtained by inverting the background pattern with the InvertData (user0) control signal. A compare using the foreground pattern is then performed and a hit at this specific address is expected by setting the SingleMatch (user4) input of the expected match generator to 1. The background pattern is restored at the address in the third cycle followed by a compare to the foreground pattern in the fourth cycle where no hit is expected.

Once all addresses of the third instruction have been processed, the microprogram points back to the first instruction to re-initialize the RAM with a different background pattern as specified by the RepeatLoop wrapper.

These simple algorithms only test the basic capabilities of a binary CAM. Additional algorithms can be developed to test additional features that the CAM might have using the same infrastructure. For example, the global compare enable inputs (CompareEnable) of the CAM require the control the CEMaskEnable (user2) signal. CompareEnable and CEMaskEnable are shown in Fig. 8 but were held constant for executing the two example algorithms.

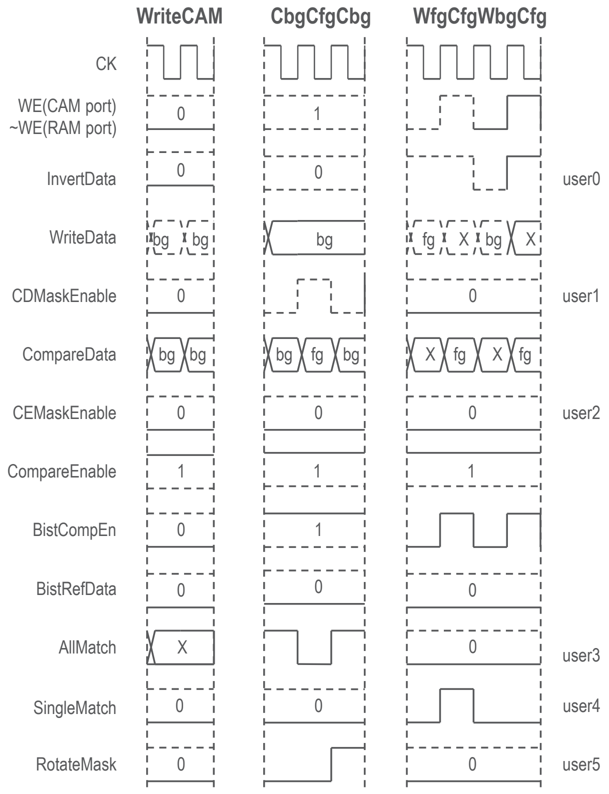

Figure 8: Complex operation set example

5. Design flow considerations

The methodology has a minimal impact on the design flow once the CAM collars has been generated and manually instantiated in the design. Note that the CAM collars have to be generated for each family of CAMs; however given the small number of CAM families present in a design, this is not expected to pose a major challenge. The remaining steps are essentially the same as for all other RAMs.

The CAM collar includes two modules: the CAM macro itself and a second module including the custom logic described in Section 4. The collar is generated with the exact same footprint as the CAM macro so that it is instantiated as usual in the design. Fig. 9 shows the hierarchical arrangement of these modules.

The custom logic module is modeled as a multi-port memory for the BIST software. The exact port configuration depends on the CAM macro functionality. For a CAM with a single compare port and separate data inputs for the RAM and CAM functions allowing simultaneous read/write and compare operations, the memory is modeled as a 2RW.

Dummy ports are added to the custom logic module as needed. For example, a dummy address port is added for the compare port even though it doesn't exist on the CAM macro. The output data bus associated to the compare port is also adjusted to be of the same width as the RAM port as explained in the compactor/expander logic section. The CAM always appears as having embedded BIST multiplexers as the test input coming from the controller needs to be modified before being applied to the CAM macro as shown in Fig. 9. The custom collar module is parameterized so that it can adapt to the CAM size or other options of the CAM macro. The BIST controller and memory interface modules are automatically added during test insertion. The BIST controller can be shared with other memories, RAM or CAM.

Verification test bench and manufacturing pattern generation is done like for any other multi-port RAM except that it is mandatory to run tests for the RAM and CAM ports in separate steps, if tested by the same memory BIST controller, since the algorithms and operation sets used are completely different for each port.

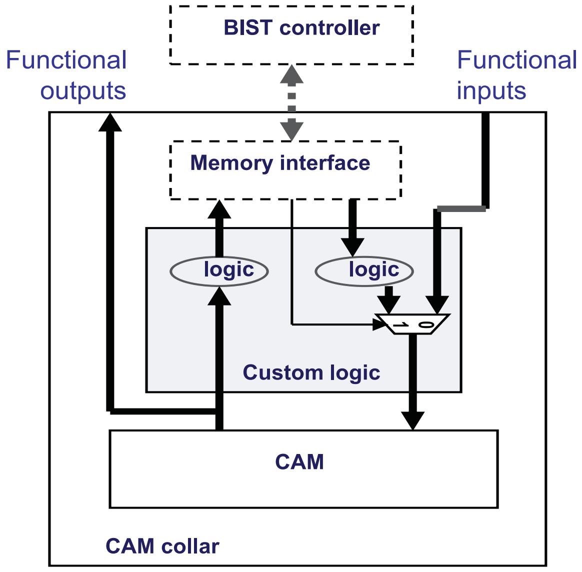

Figure 9: Connections of CAM collar and BIST logic

The resultant overhead of implementing the CAM collar on a few array BCAM configurations is provided below. In the table, the (normalized) area of the CAM collar is compared with the MBIST controller area (normalized) for the same array.

Array Width	Array Depth	CAM collar area	MBIST cntlr area
72	128	1.0	14.2
140	252	2.7	23.1
72	512	2.3	15.8

It can be seen that the overhead of implementing the CAM collar using the techniques described in this paper, is between 10-15% of the MBIST controller.

6. Conclusions

A new methodology to implement BIST for CAMs has been presented in this paper. Capabilities of existing commercial tools implementing BIST for RAMs such as algorithm programming, automatic assembly, pattern generation and diagnosis are leveraged. The amount of custom work is limited to the generation of a collar making the CAM look like a multi-port memory. This methodology has been successfully applied to a number of CAMs.

References

[1] A. G. Hanlon, "Content-addressable and associative memory systems", IEEE Trans. On Electronic Computers, vol. EC-15, no. 4, August 1966

[2] K. Pagiamtzis, Ali Sheikholeslami, "Content-Addressable Memory (CAM) circuits and architectures: a tutorial and survey", IEEE Journal of solid-state circuits, vol. 41, no. 3, March 2006

[3] N. Babu, F. Noorbasha, B. Shankar, "A Survey on Content Addressable Memory", International Journal of Emerging Trends & Technology in Computer Science, vol. 2, issue 3, May-June 2013

[4] R. Prashidha, G. Shanthi, K. Priyadharshini, " A survey on design of content addressable memories", International Journal of Science, Engineering and Technology Research, vol. 3, issue 10, October 2014

[5] H. Noda et al, "A 143MHz 1.1 4.5Mb dynamic TCAM with hierarchical searching and shift redundancy architecture ", IEEE International Solid-State Circuits Conference, 2004, pp. 208-209

[6] H. Jarollahi, N. Onizawa, V. Gripon, W. J. Gross, "Architecture and implementation of an associative memory using sparse clustered networks", Proc. IEEE International Symposium on Circuits and Systems, May 2012, pp. 2901-2904

[7] H.-Y. Yang et al., "Testing methods for quaternary content addressable memory using charge-sharing sensing scheme", IEEE International Test Conference, 2015, paper 19.1

[8] S. Kornachuk, L. McNaughton, R. Gibbins, B. Nadeau-Dostie, "A high speed embedded cache design with non-intrusive BIST", IEEE International Workshop on Memory Technology, Design and Testing, 1994, pp. 40-45

[9] H. Grigoryan, G. Harutyunyan, S. Shoukourian, V. Vardanian, Y. Zorian, "Generic BIST architecture for testing of content addressable memories", IEEE 17th International On-Line Testing Symposium, 2011, pp. 86-91

[10] D.K. Bhavsar, "A built-in self-test method for write-only content addressable memories", 23rd IEEE VLSI Test Symposium, 2005, pp. 9-14

[11] H.-H. Wu et al., "A comprehensive TCAM test scheme: an optimized test algorithm considering physical layout and combining scan test with at-speed BIST design", IEEE International Test Conference, 2009, paper 7.3

[12] Tessent Memory BIST user's and reference manual, v2015.1