A Reconfigurable Content Addressable Memory

Steven A. Guccione, Delon Levi and Daniel Downs

Xilinx Inc.
2100 Logic Drive
San Jose, CA 95124 (USA)
Steven.Guccione@xilinx.com
Delon.Levi@xilinx.com
Daniel.Downs@xilinx.com

Abstract. Content Addressable Memories or *CAMs* are popular parallel matching circuits. They provide the capability, in hardware, to search a table of data for a matching entry. This functionality is a high performance alternative to popular software-based searching schemes. CAMs are typically found in embedded circuitry where fast matching is essential. This paper presents a novel approach to CAM implementation using run-time reconfiguration. This approach produces CAM circuits that are smaller, faster and more flexible than traditional approaches.

1 Introduction

Content Addressable Memories or *CAMs* are a class of parallel pattern matching circuits. In one mode, these circuits operate like standard memory circuits and may be used to store binary data. Unlike standard memory circuits, however, a powerful *match mode* is also available. This match mode permits all of the data in the CAM device to be searched in parallel.

While CAM hardware has been available for decades, its use has typically been in niche applications, embedded in custom designs. Perhaps the most popular application has been in cache controllers for central processing units. Here CAMs are often used to search cache tags in parallel to determine if a cache "hit" or "miss" has occurred. Clearly in this application performance is crucial and parallel search hardware such as a CAM can be used to good effect.

A second and more recent use of CAM hardware is in the networking area [1]. As data packets arrive into a network router, processing of these packets typically depends on the network destination address of the packet. Because of the large number of potential addresses, and the increasing performance demands, CAMs are beginning to become popular in processing network address information.

2 A Standard CAM Implementation

CAM circuits are similar in structure to traditional Random Access Memory (RAM) circuits, in that data may be written to and read from the device [2]. In addition to functioning as a standard memory device, CAMs have an additional

parallel search or *match* mode. The entire memory array can be searched in parallel using hardware. In this match mode, each memory cell in the array is accessed in parallel and compared to some value. If this value is found in any of the memory locations, a *match* signal is generated.

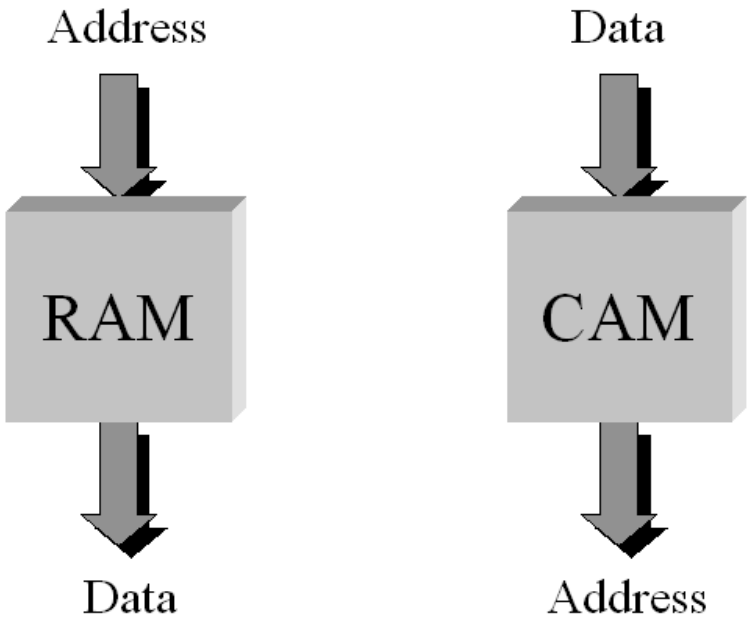

Fig. 1. RAM versus CAM functionality.

In some implementations, all that is significant is that a match for the data is found. In other cases, it is desirable to know exactly where in the memory address space this data was located. Rather than producing a simple "match" signal, some CAM implementations also supply the address of the matching data. In some sense, this provides a functionality opposite of a standard RAM. In a standard RAM, addresses are supplied to hardware and data at that address is returned. In a CAM, *data* is presented to the hardware and an *address* returned. Figure 1 shows this high level functionality.

At a lower level, the actual transistor implementation of a CAM circuit is very similar to a standard static RAM. Figure 2 shows transistor level diagrams of both CMOS RAM and CAM circuits. The circuits are almost identical, except for the addition of the *match* transistors to provide the parallel search capability.

In a CMOS static RAM circuit, as well as in the CAM cell, data is accessed via the *BIT* lines and the cells selected via the *WORD* lines. In the CAM cell,

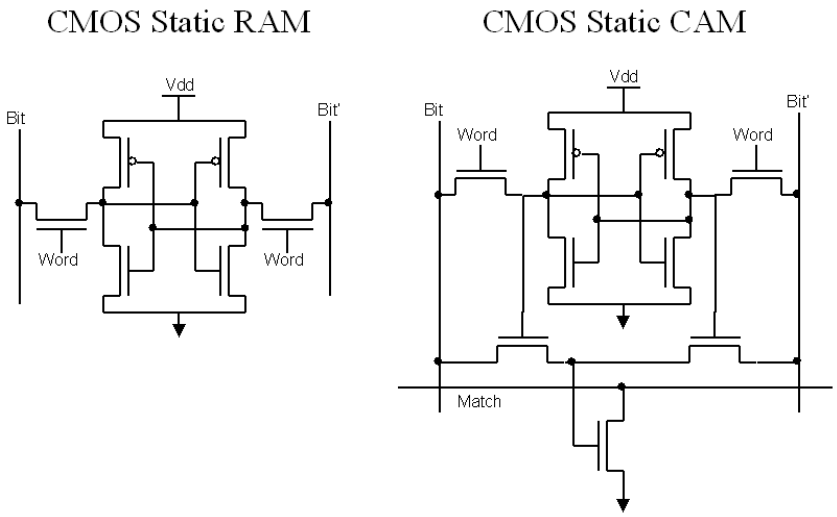

Fig. 2. RAM versus CAM transistor level circuits.

however, the *match* mode is somewhat different. Inverted data is placed on the *BIT* lines. If any cell contains data which does not match, the *MATCH* line is pulled low, indicating that no match has occurred in the array.

Clearly this transistor level implementation is efficient and may be used to produce CAM circuits which are nearly as dense as comparable static RAM circuits. Unfortunately, such transistor level circuits can not be implemented using standard programmable logic devices.

3 An FPGA CAM Implementation

Of course, a content addressable memory is just a digital circuit, and as such may be implemented in an FPGA. The general approach is to provide an array of registers to hold the data, and then use some collection of comparators to see if a match has occurred. Figure 3 shows a diagram of such a circuit.

While this is a viable solution, it suffers from the same sort of inefficiencies that plague FPGA-based RAM implementations. Like RAM, the CAM is efficiently implemented at the transistor level. Using gate level logic, particularly programmable or reconfigurable logic, often results in a substantial penalty, primarily in size.

Because the FPGA CAM implementation relies on flip-flops as the data storage elements, the size of the circuit is restricted by the number of flip flops in the device. While this is adequate for smaller CAM designs, larger CAMs quickly deplete the resources of even the largest available FPGA.

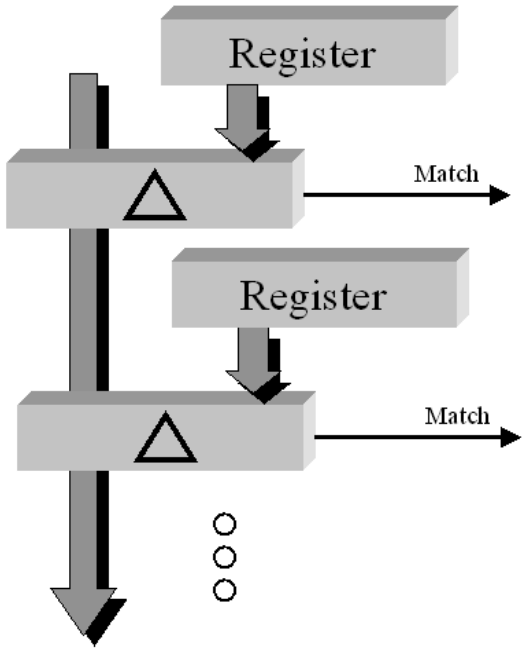

Fig. 3. An FPGA-based CAM circuit.

4 The Reconfigurable Content Addressable Memory (RCAM)

The *Reconfigurable Content Addressable Memory* or *RCAM* makes use of run-time reconfiguration to efficiently implement a CAM circuit. Rather than using the FPGA flip-flops to store the data to be matched, the RCAM uses the FPGA *Look Up Tables* or *LUTs*. Using LUTs rather than flip-flops results in a smaller, faster CAM.

The approach uses the LUT to provide a small piece of CAM functionality. In Figure 4, a LUT is loaded with data which provides a "match 5" functionality. That is, whenever the binary encoded value "5" is sent to the four LUT inputs, a *match* signal is generated. Note that using a LUT to implement CAM functionality, or any functionality for that matter, is not unique. An N-input LUT can implement any arbitrary function of N inputs, including a CAM.

Because a LUT can be used to implement any function of N variables, it is also possible to provide more flexible matching schemes than the simple match described in the circuit in Figure 4. In Figure 5, the LUT is loaded with values which produce a match on any value but binary "4". This circuit demonstrates the ability to embed a *mask* in the configuration of a LUT, permitting arbitrary

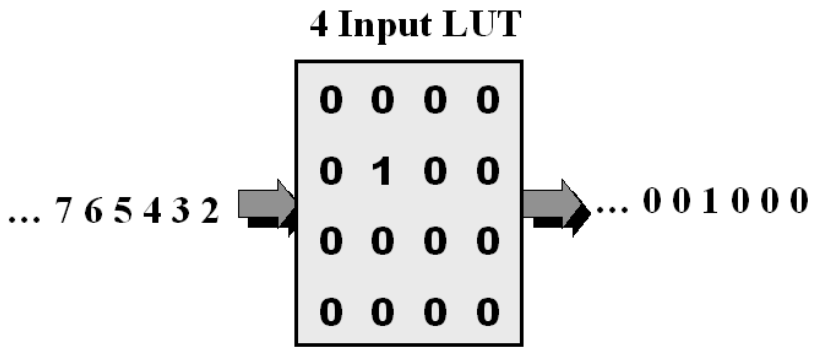

Fig. 4. Using a LUT to match 5.

disjoint sets of values to be matched, within the LUT. This function is important in many matching applications, particularly networking.

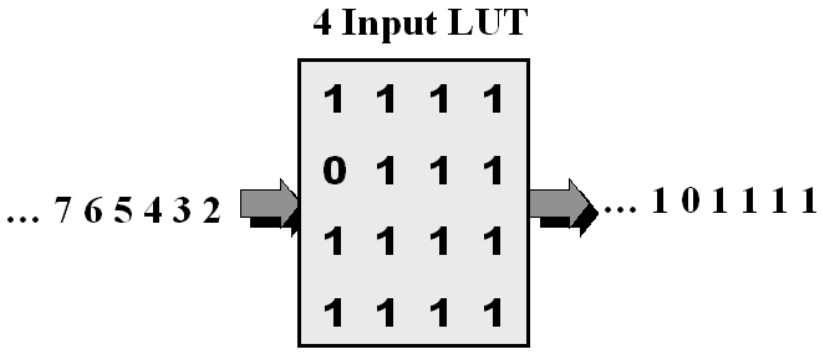

Fig. 5. Using a LUT to match all inputs except 4.

This approach can be used to provide matching circuits such as *match all* or *match none* or any combination of possible LUT values. Note again, that this arbitrary masking only applies to a single LUT. When combining LUTs to make larger CAMs, the ability to perform such masking becomes more restricted.

While using LUTs to perform matching is a powerful approach, it is somewhat limited when used with traditional design tools. With schematics and HDLs, the LUT contents may be specified, albiet with some difficulty. And once

specified, modifying these LUTs is difficult or impossible. However, modification of FPGA circuitry at run-time is possible using a run-time reconfiguration tool such as *JBits* [3]. *JBits* permits LUT values, as well as other parts of the FPGA circuit, to be modified arbitrarily at run time and in-system. An *Application Program Interface (API)* into the FPGA configuration permits LUTs, for instance, to be modified with a single function call. This, combined with the partial reconfiguration capabilities of new FPGA devices such as Virtex (tm) permit the LUTs used to build the RCAM to be easily modified under software control, without disturbing the rest of the circuit.

Finally, using run-time reconfiguration software such as *JBits*, RCAM circuits may be dynamically sized, even at run-time. This opens the possibility of not only changing the contents of the RCAM during operation, but actually changing the size and shape of the RCAM circuit itself. This results in a situation analogous to dynamic memory allocation in RAM. It is possible to "allocate" and "free" CAM resources as needed by the application.

5 An RCAM Example

One currently popular use for CAMs is in networking. Here data must be processed under demanding real-time constraints. As packets arrive, their routing information must be processed. In particular, destination addresses, typically in the form of 32-bit *Internet Protocol (IP)* addresses must be classified. This typically involves some type of search.

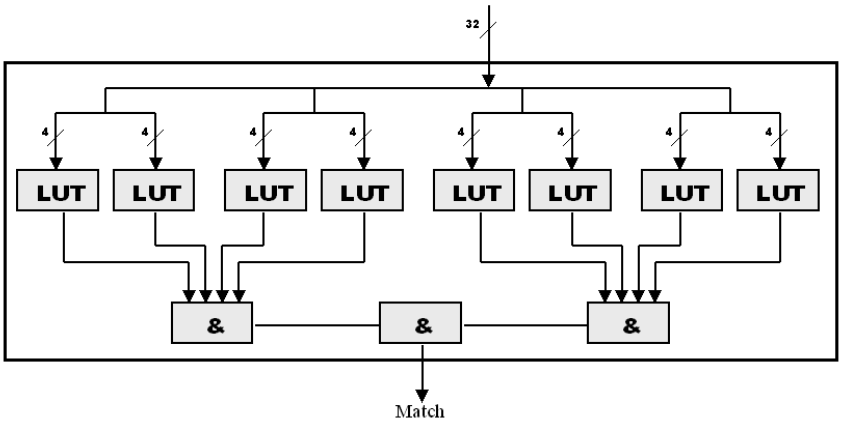

Fig. 6. Matching a 32-bit IP header.

Current software based approaches rely on standard search schemes such as hashing. While effective, this approach requires a powerful processor to keep

up with the real-time demands of the network. Offloading the computationally demanding matching portion of the algorithms to external hardware permits less powerful processors to be used in the system. This results in savings not only in the cost of the processor itself, but in other areas such as power consumption and overall system cost.

In addition, an external CAM provides networking hardware with the ability to achieve packet processing in essentially constant time. Provided all elements to be matched fit in the CAM circuit, the time taken to match is independent of the number of items being matched. This provides not only good scalability properties, but also permits better real-time analysis. Other software based matching schemes such as hashing are data-dependent and may not meet real-time constraints depending on complex interactions between the hashing algortihm and the data being processed. CAMs suffer no such limitations and permit easy analysis and verification.

Figure 6 shows and example of an IP Match circuit constructed using the RCAM approach. Note that this example assumes a basic 4-input LUT structure for simplicity. Other optimizations, including using special-purpose hardware such as carry chains are possible and may result in substantial circuit area savings and clock speed increases.

This circuit requires one LUT input per matched bit. In the case of a 32-bit IP address, this circuit requires 8 LUTs to provide the matching, and three additional 4-input LUTs to provide the ANDing for the MATCH signal. An array of this basic 32-bit matching block may be replicated in an array to produce the CAM circuit. Again, note that other non-LUT implementations for generating the MATCH circuit are possible.

Since the LUTs can be used to mask the matching data, it is possible to put in "match all" conditions by setting the LUTs to all ones. Other more complicated masking is possible, but typically only using groups of four inputs. While this does not provide for the most general case, it appears to cover the popular modes of matching.

6 System Issues

The use of run-time reconfiguration to construct, program and reprogram the RCAM results in some significant overall system savings. In general, both the hardware and the software are greatly simplified.

Most of the savings accrue from being able to directly reconfigure the LUTs, rather than having to write them directly as in standard RAM circuits. Reconfiguration rather than direct access to the stored CAM data first eliminates all of the read / write access circuitry. This includes the decode logic to decode each address, the wiring necessary to broadcast these addresses, the data busses for reading and writing the data, and the IOBs used to communicate with external hardware.

It should be pointed out that this interface portion of the circuitry is substantial, both is size and complexity. Busses typically consume tri-state lines,

which are often scarce. Depending on the addressing scheme, tens of IOBs will necessarily be consumed. These also tend to be valuable resources. The address decoders are also somewhat problematic circuits and often require special purpose logic to be implemented efficiently. In addition, the bus interface is typically the most timing sensitive portion of the circuit and requires careful design and simulation. This is eliminated with the use of run-time reconfiguration.

Finally, the system software is simplified. In a standard bus interface approach, device drivers and libraries must be written, debugged and maintained to access the CAM. And when the system software or processor changes, this software must be ported to the new platform. With the RCAM, all interfacing is performed through the existing configuration port, at no additional overhead.

The cost of using the configuration port rather than direct hardware access is primarily one of setup speed. Direct writes can typically be done in some small number of system cycles. Reconfiguration of the RCAM to update table entries may take substantially longer, depending on the implementation. Partial reconfiguration in devices such as Virtex permit changes to be made more rapidly than in older bulk configuration device, but the speed may be orders of magnitude slower than direct hardware approaches. Clearly the RCAM approach favors applications with slowly changing data sets. Fortunately, many applications appear to fit into this category.

7 Comparison to Other Approaches

While CAM technology has been in widespread use for decades, there has been little interest in producing commercial CAM devices. This recent interest in CAMs, driven primarily by the high-performance networking market, has resulted in commercially available CAM devices. Music Semicondictor [4] and Net Logic [5] are two companies which provide CAM devices tailored specifically for the networking market.

In addition, at least one FPGA manufacturer, Altera, has begun to embed CAM hardware into their Apex(tm) devices. While this circuitry is embedded in an FPGA, it is special purpose and not part of the general configurable fabric. It is included here for comparison, but it should be pointed out that special purpose hardware is readily inserted into FPGAs. The cost here is inflexibility. The special purpose hardware must be used for a specific circuit, at a specific physical location, or not used at all. In this sense, this embedded CAM has more in common with custom solutions than programmable solution. But the specifications are included here for comparison.

Figure 7 gives some sizes for current commercially available devices. While these are custom CAM implementations and can expected to be denser than FPGA implementations, the RCAM sizes are within the general range of that available from custom implementations. In addition, the RCAM circuits are more flexible and may be placed at any location within the FPGA and may be integrated with other logic in the design. Finally, the RCAM approach is approximately 3-4 times denser than attempting to implement a CAM using an FPGA

CAM (Virtex V1000)	768 x 32	384 x 64
RCAM (Virtex V1000)	3K x 32	1K x 64
Quality Semiconductor	1K x 64	2K x 64
Net Logic	16K x 64	8K x 128
Music Semiconductor	2K/4K/6K x 32	2K/4K/6K x 64
Altera APEX	1K-8K x 32	500-4K x 64

Fig. 7. Some commercially available CAM devices.

and traditional design appraoches. Optimizations using logic such as the Virtex carry chain also indicate improvements of an additional 40%.

8 Associative Processing

Today, advances in circuit technology permit large CAM circuits to be built. However, uses for CAM circuits are not necessarily limited to niche applications like cache controllers or network routers. Any application which relys on the searching of data can benefit from a CAM-based approach. A short list of some potential application areas that can benefit from fast matching are:

- Artificial intelligence
- Database
- Computer Aided Design
- Graphics
- Computer Vision

Much of the work in using parallel matching hardware to accelerate algorithms was carried out in the 1960s and 1970s, when several large parallel matching machines were constructed. An excellent survey of so-called *Associative Processors* can be found in Yau and Fung [6].

With the rapid growth both in size and speed of traditional processors in the intervening years, much of the interest in CAMs has faded. However, as real-time constraints in areas such as networking become impossible to meet with traditional processors, solutions such as CAM-based parallel search will almost certainly become more prevalent.

In addition, the use of parallel matching hardware in the form of CAMs can provide another more practical benefit. For many applications, the use of CAM-based parallel search can offload much of the work done by the system processor. This should permit smaller, cheaper and lower power processors to be used in embedded applications which can make use of CAM-based parallel search.

9 Conclusions

The *RCAM* is a flexible, cost-effective alternative to existing CAMs. By using FPGA technology and run-time reconfiguration, fast, dense CAM circuits can be easily constructed, even at run-time. In addition, the size of the *RCAM* may be tailored to a particular hardware design, or even temporary changes in the system. This flexibility is not available in other CAM solutions.

In addition, the *RCAM* need not be a stand-alone implementation. Because the *RCAM* is entire a software solution using state of the art FPGA hardware, it is quite easy to embed *RCAM* functionality in larger FPGA designs.

Finally, we believe that existing applications, primarily in the field of network routing, are just the beginning of *RCAM* usage. Once other applications realize that simple, fast, flexible parallel matching is available, it is likely that other applications and algorithms will be accelerated using this approach.

10 Acknowledgements

Thanks to Kjell Torkellesson and Mario Dugandzic for discussions on networking. And thanks especially to Paul Hardy for early RCAM discussions.

References

1. R. Neale, "Is content addressable memory (CAM) the key to network success?," *Electronic Engineering* **71**, pp. 9–12, February 1999.
2. N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley Publishing Company, 1985.
3. S. A. Guccione and D. Levi, "XBI: A Java-based interface to FPGA hardware," in *Configurable Computing Technology and its use in High Performance Computing, DSP and Systems Engineering, Proc. SPIE Photonics East*, J. Schewel, ed., pp. 97–102, SPIE – The International Society for Optical Engineering, (Bellingham, WA), November 1998.
4. "Music semiconductor." World Wide Web page http://www.music-ic.com/, 1999.
5. "Net logic microsystems." World Wide Web page http://www.netlogicmicro.com/, 1999.
6. S. S. Yau and H. S. Fung, "Associative processor architecture – a survey," *Computing Surveys* **9**, pp. 3–27, March 1977.
7. Xilinx, Inc., *The Programmable Logic Data Book*, 1996.