Implementing High-Speed Search Applications with APEX CAM

July 1999, ver. 1.01 Application Note 119

Introduction

Most memory devices store and retrieve data by addressing specific memory locations. For example, a system using RAM or ROM searches sequentially through memory to locate data. However, this technique can slow system performance since the search requires multiple clock cycles to complete.

You can considerably reduce the time required to find an item stored in memory by identifying stored data by content, rather than by its address. Memory accessed in this way is called content-addressable memory (CAM). CAM offers a performance advantage over other memory search algorithms, such as binary-based searches, tree-based searches, or look-aside tag buffers, because it compares the desired information against the entire list of pre-stored entries simultaneously. Thus, CAM provides an order-of-magnitude reduction in the search time.

CAM is ideally suited for many applications, including Ethernet address lookup, data compression, pattern recognition, cache tags, fast routing table lookup, high-bandwidth address filtering, user privileges, and security and encryption information.

This application note discusses the following topics:

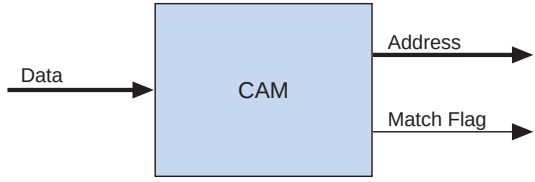
- CAM Fundamentals
- CAM in APEX™ 20KE Devices
- CAM Applications

CAM Fundamentals

CAM is based on RAM technology. RAM operates as a circuit that stores data at a particular address. When retrieving data from RAM, the system supplies the address and then receives the data. With CAM, the system supplies the data instead of the address. To locate stored data, CAM takes one clock cycle to search through all memory locations in parallel and returns the data's address. CAM drives a match flag high if the data is found, or low if the data is not found.

Figure 1 shows a block diagram of CAM operation.

Figure 1. CAM Block Diagram



CAM Accelerates Searches

CAM can accelerate applications requiring fast searches of databases, lists, or patterns, such as in image or voice recognition. For example, the search key could be a network user's internet protocol (IP) address, and the associated information could be a user's access privileges and location on the network. If the search key presented is available in CAM, CAM indicates a match and returns the associated information, i.e., the user's privileges.

CAM Integration

Currently, most applications requiring fast searches use discrete CAM. Designers must add a separate CAM device to their printed circuit board (PCB), which increases design time and reduces the amount of usable PCB space. Discrete CAM also reduces system performance because it introduces additional on-chip and off-chip delays.

APEX 20KE devices, which contain on-chip CAM built into their embedded system blocks (ESBs), eliminate the disadvantages of discrete CAM. APEX 20KE on-chip CAM has an access time of 4 ns, compared to a 20-ns access time for a typical discrete CAM. Because CAM is integrated inside an APEX 20KE device, it provides faster system performance than traditional discrete CAM. APEX 20KE device CAM is optimized for small- and medium-sized applications that are described in the "CAM Applications" section on page 8.

CAM in APEX 20KE Devices

In APEX 20KE devices, each ESB can implement a 32-word × 32-bit CAM block. Figure 2 shows CAM implemented in an ESB, and Figure 3 shows a block diagram of CAM.

Figure 2. Implementing CAM in an ESB

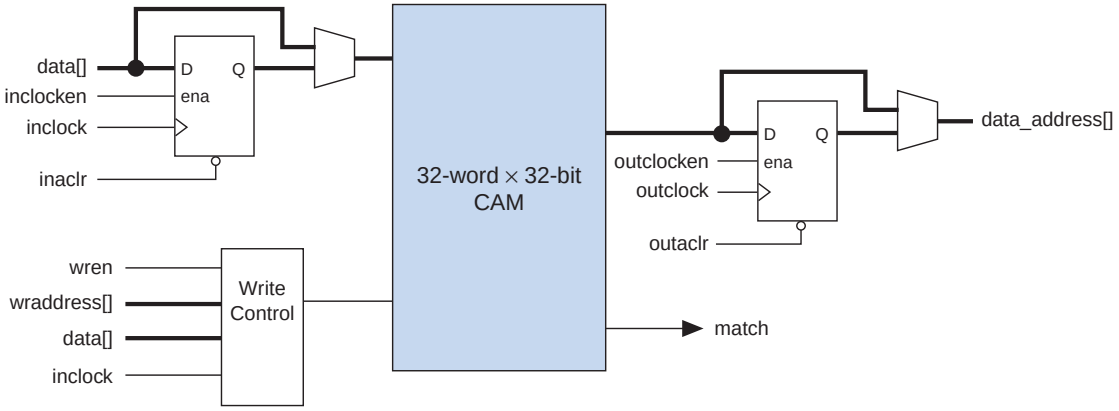
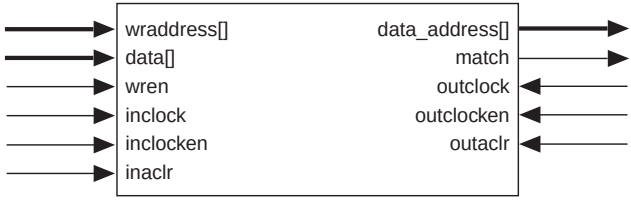


Figure 3. APEX 20KE CAM Block Diagram

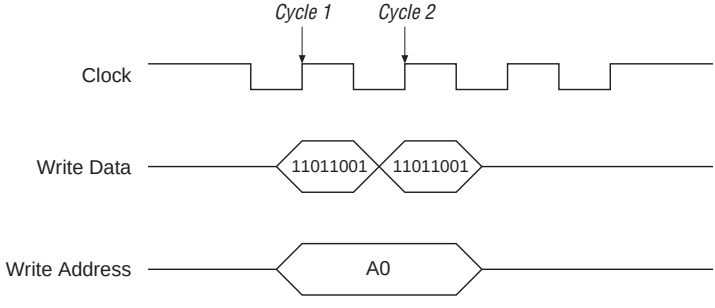


Writing to APEX CAM

You can either pre-load CAM with data during configuration, or you can write data during system operation. In most cases, two clock cycles are required to write each word into CAM.

Figure 4 shows the waveform for an 8-bit input written to address A0 of a CAM block. The data is driven to CAM for two clock cycles.

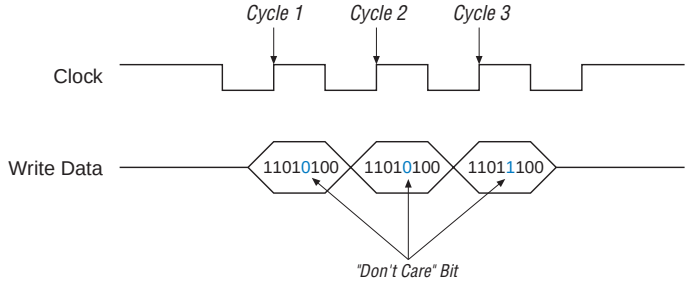
Figure 4. Writing an 8-Bit Input to CAM



A design can write "don't care" bits into CAM words; bits set to "don't care" do not affect matching. The "don't care" bits can be used as a mask for CAM comparisons. A third clock cycle is required when "don't care" bits are used. The "don't care" bits are signified by inverting them on the third clock cycle.

Figure 5 shows the waveform for a 1101X100 word (with a "don't care" bit) written to CAM.

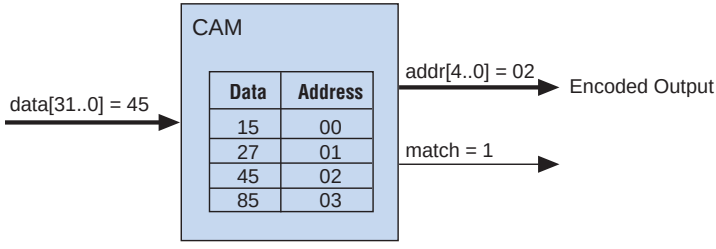
Figure 5. Writing 1101X100 to CAM



Reading from APEX CAM

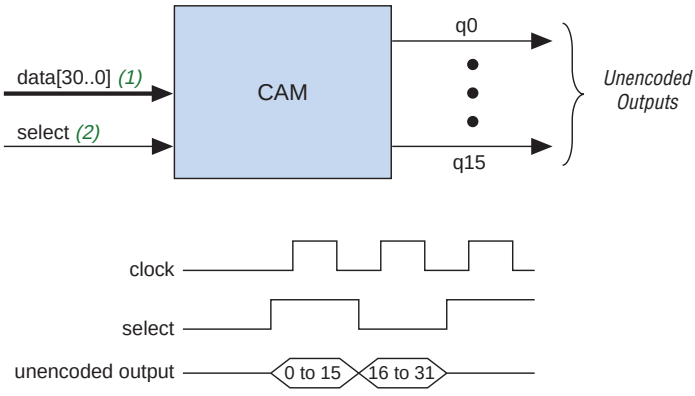
The CAM output is either encoded or unencoded. When encoded, the ESB outputs the data's location as an encoded address. Encoded data is better suited for designs without duplicate data in the memory, and reading an encoded output requires only one clock cycle. Figure 6 shows the encoded output for CAM in APEX 20KE devices.

Figure 6. Encoded CAM Output



You should use an unencoded output if the same data may be written to multiple locations in the memory. In this mode, an ESB uses 16 outputs and reads the outputs in two cycles: 16 bits at each cycle to represent the 32 words in the CAM block. Each output represents one word of the CAM. There is no match output in this mode; if any of the outputs go high, there is a match (e.g., if the data is located in address 15, the fifteenth output line goes high). Figure 7 shows the CAM's unencoded output in APEX 20KE devices.

Figure 7. Unencoded CAM Output



Notes:

(1) For an unencoded output, the ESB only supports 31 input data bits. One input bit is used by the `select` line to choose one of the two banks of 16 outputs.

(2) If the `select` input is a `1`, then CAM outputs words 0 through 15. If the `select` input is a `0`, CAM outputs words 16 through 31.

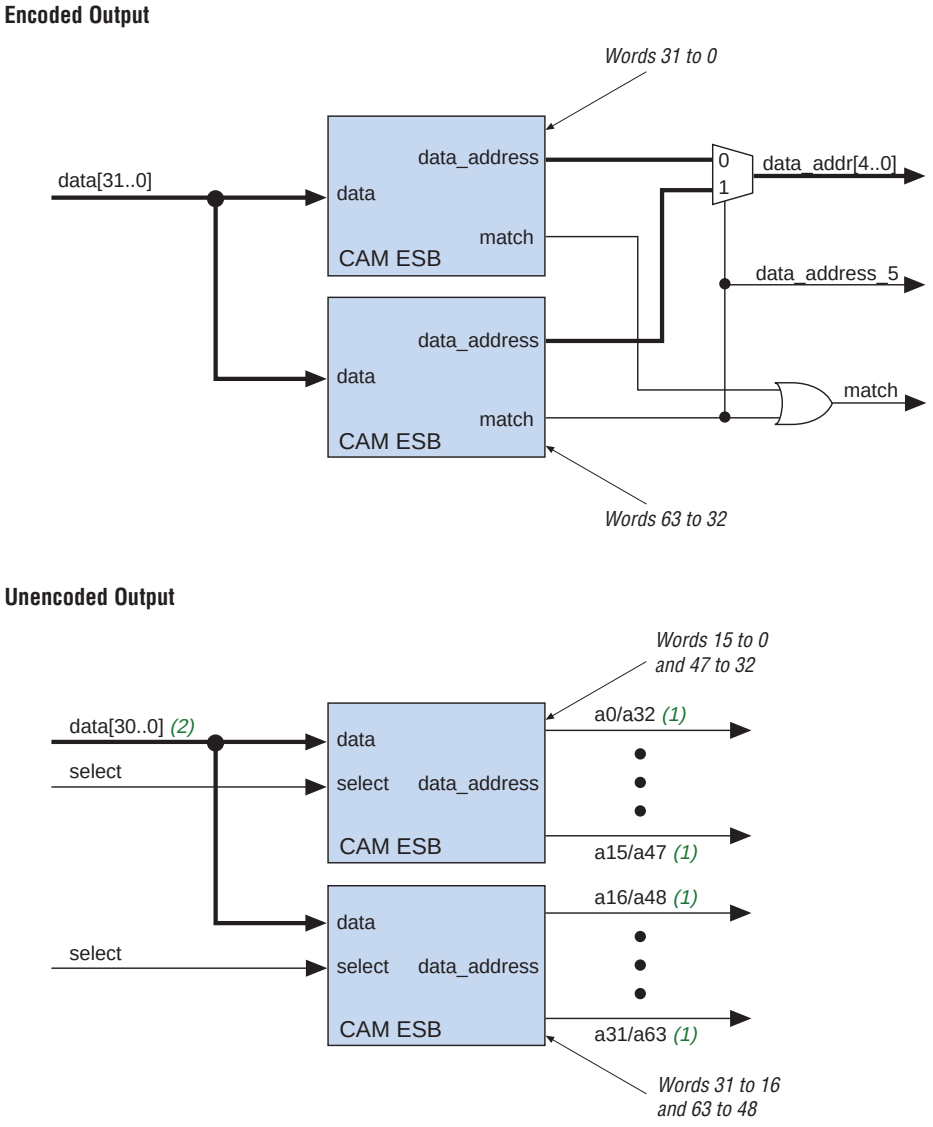
Deeper & Wider CAM Blocks

Each ESB in an APEX 20KE device supports a 1-Kbit CAM block (32 words of 32 bits each). You can implement wider or deeper CAM by combining multiple CAM blocks using logic elements (LEs). The Quartus™ software combines ESBs and LEs automatically to create larger CAM blocks. There is no intrinsic limit to cascading APEX 20KE ESBs; all ESBs in a device can be combined into one very large CAM block. For example, by cascading 64 (out of 104) ESBs in an EP20K400E device, you can generate a 2,048-word × 32-bit or 1,024-word × 64-bit block of CAM. Large APEX 20KE devices, such as the EP20K1000E device (with 160 ESBs), can generate a 4,096-word × 32-bit CAM block using 128 ESBs.

Creating Deeper CAM

To create deeper CAM blocks, the Quartus software cascades the output of each ESB. Both encoded and unencoded CAM outputs are used to create deeper CAM blocks. In an encoded implementation, a multiplexer selects the output of one of the ESBs and drives it out. The select line of the multiplexer is controlled by the match flags of the ESBs. Figure 8 shows an example of 64-word × 32- or 31-bit CAM implemented with encoded and unencoded outputs.

Figure 8. Creating Deeper CAM with Encoded & Unencoded Outputs



Notes:

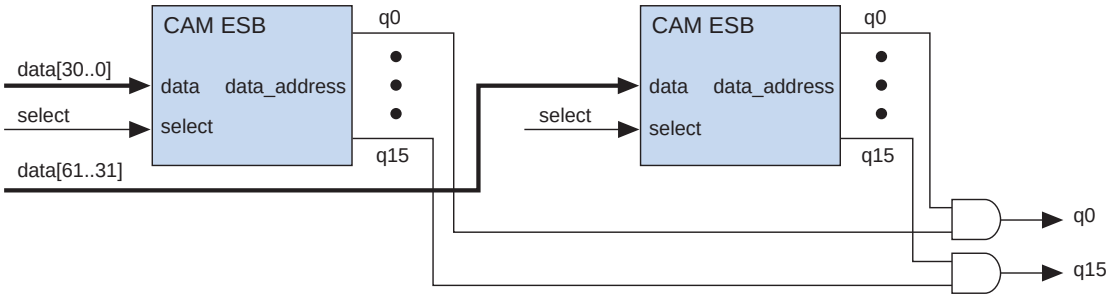
(1) Words 0 through 31 are driven out in parallel in the first clock cycle, and words 32 through 63 are driven out in parallel in the second clock cycle.

(2) For an unencoded output, the ESB only supports 31 input bits. One input bit selects one of the two banks of 16 outputs.

Creating Wider CAM

To increase the width of a CAM block, the Quartus software cascades the ESB's unencoded outputs. Encoded outputs cannot be used, because two different data words may coincidentally contain matching portions and cause an incorrect output. To cascade the ESBs, each bit of the first ESB is ANDed with the corresponding bit of the second ESB. When both ESBs report a match, the entire word matches the stored word. Figure 9 shows an example of 32-word × 62-bit CAM implemented with unencoded outputs.

Figure 9. Creating Wider CAM with an Unencoded Output



CAM Applications

CAM is used to accelerate a variety of applications such as local-area networks (LANs), database management, file-storage management, table look up, pattern recognition, artificial intelligence, fully associative and processor-specific cache memories, and disk cache memories. CAM can also perform any search operation.

This section discusses the following applications:

- Data Compression
- Network Switch
- IP Filters
- ATM Switch
- Cache Tags
- Peripheral Component Interconnect (PCI) and Other Bus Applications

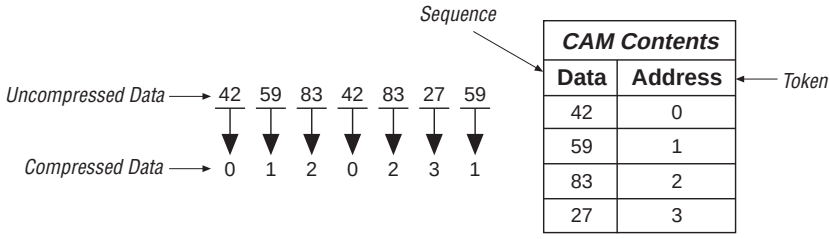
Data Compression

Data compression removes redundancy in a given piece of information, producing an equivalent, but shorter, message. Data compression is particularly useful in communications because it allows devices to transmit the same amount of data using fewer bits.

CAM implements data compression efficiently because it can quickly search through the data structure containing the compression information. Because a good portion of a compression algorithm's time is spent searching and maintaining this data structure, a hardware search engine can greatly increase the algorithm throughput.

CAM look-up is performed after each word is presented. If the specific code is not found in CAM, another word is shifted in. When the code is found, CAM outputs the appropriate token and the input register is flushed. CAM generates a result in a single transaction regardless of the table size or length of the search list. This process makes CAM ideal for data compression schemes that use sparsely populated tables as part of their algorithm. Figure 10 shows an example of data compression using CAM.

Figure 10. Using CAM for Data Compression

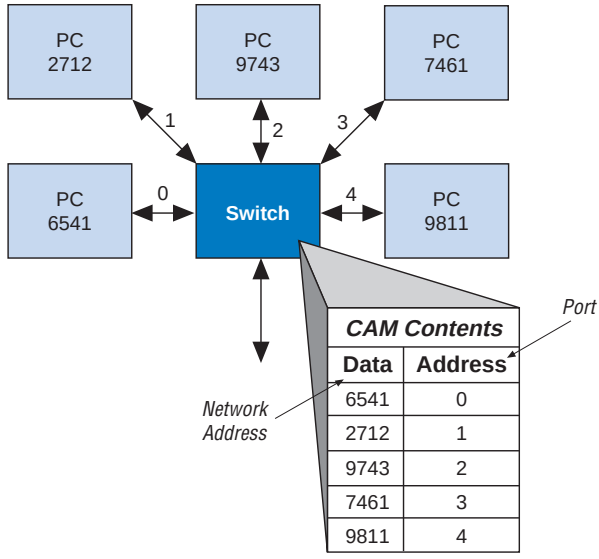


Network Switch

Switch applications use CAM to process the address information from incoming packets. To switch a packet to the correct outgoing port, the incoming packet address is compared with a table of network addresses stored in CAM. CAM outputs the destination for each data packet based on its address.

CAM can store network address and switch port numbers (see Figure 11). CAM in the switch compares gathered data against its stored table. If the comparison yields a match, CAM outputs the destination, and routing control forwards the packet to the correct port.

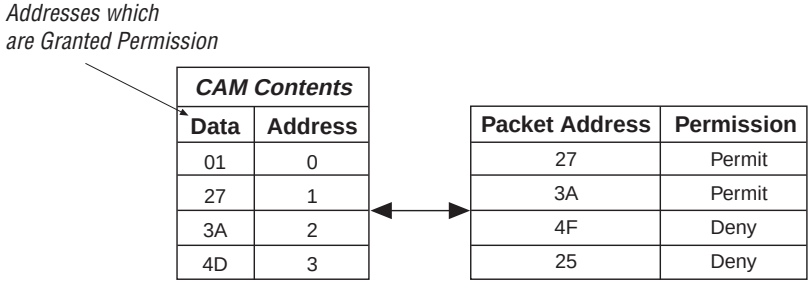
Figure 11. Using CAM as a Network Switch



IP Filters

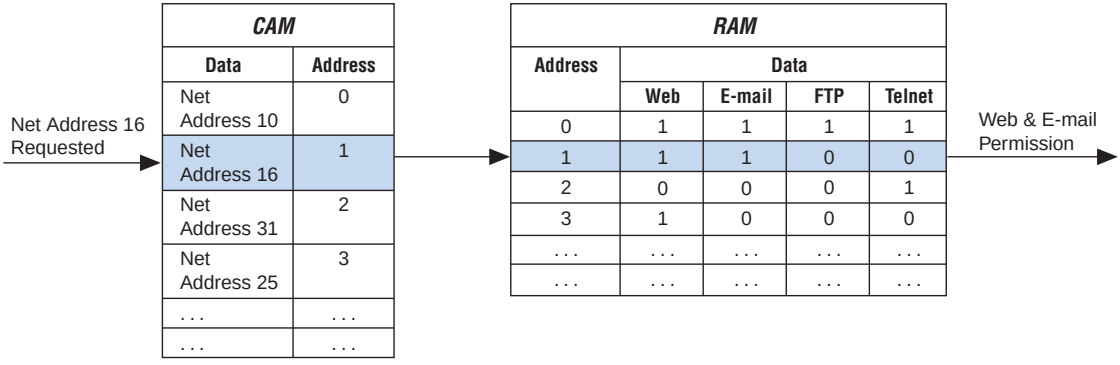
An IP filter is a security feature that prohibits unauthorized users from accessing LAN resources. It can also restrict IP traffic over a wide-area network (WAN) link. With an IP filter, LAN users can be restricted to specific applications on the Internet (such as e-mail). CAM works as a filter to block all access except for packets that have permission. The addresses that have permission are stored in CAM; when an address is sent to memory, CAM reports whether it contains the address. If the address resides within CAM, it has permission for a particular activity. Figure 12 shows an example of an IP filter.

Figure 12. Using CAM as an IP Filter



When multiple permissions are required, a combination of CAM and RAM enables this operation. Figure 13 shows a sample application that regulates access to e-mail, the web, file transfer protocol (FTP), and telnet. This application uses a 4-bit RAM block; each bit of RAM refers to one permission, or access.

Figure 13. Multiple-Permission IP Filter



ATM Switch

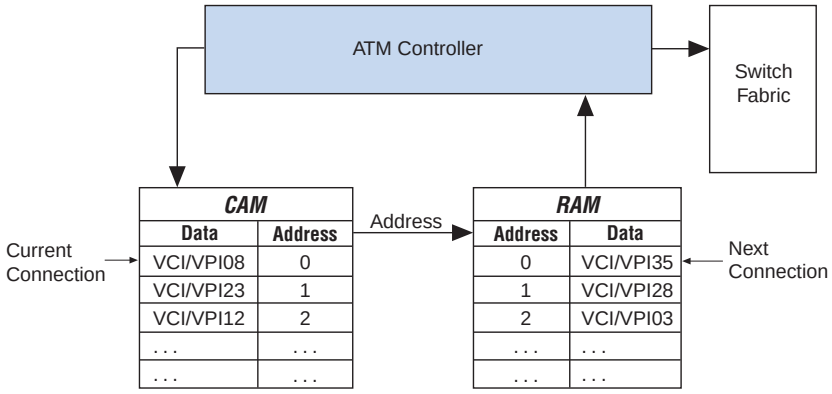
CAM can be used in asynchronous transfer mode (ATM) switching network components as a translation table. Because ATM networks are connection-oriented, virtual circuits must be set up before transferring data. There are two kinds of ATM virtual circuits: virtual path (identified by a virtual path identifier (VPI)) and channel path (identified by a channel path identifier (VCI)). VPI/VCI values are localized; each segment of the total connection has a unique VPI/VCI combination.

Whenever an ATM cell travels through a switch, its VPI/VCI value must change the next segment of connection through a process called VPI/VCI translation. It is critical to optimize the translation speed to improve the performance of high throughput ATM networks. CAM acts as an address translator in an ATM switch and performs the VPI/VCI translation very quickly. During the translation process, CAM processes the incoming VPI/VCI values in ATM cell headers and generates addresses that access data stored in RAM. RAM stores the VPI/VCI mapping data and other connection information.

VPI/VCI fields from the ATM cell header are compared against a list of current connections stored in the CAM array. From the comparison, CAM generates an address that is used to access the RAM. A combination of CAM and RAM implements the translation tables with fully parallel search capability.

The ATM controller modifies the cell header using the VPI/VCI data from the RAM, and the cell is sent to the switch. This application is shown in Figure 14. For optimal performance, both CAM and RAM should be embedded into the same device.

Figure 14. CAM in an ATM Switch



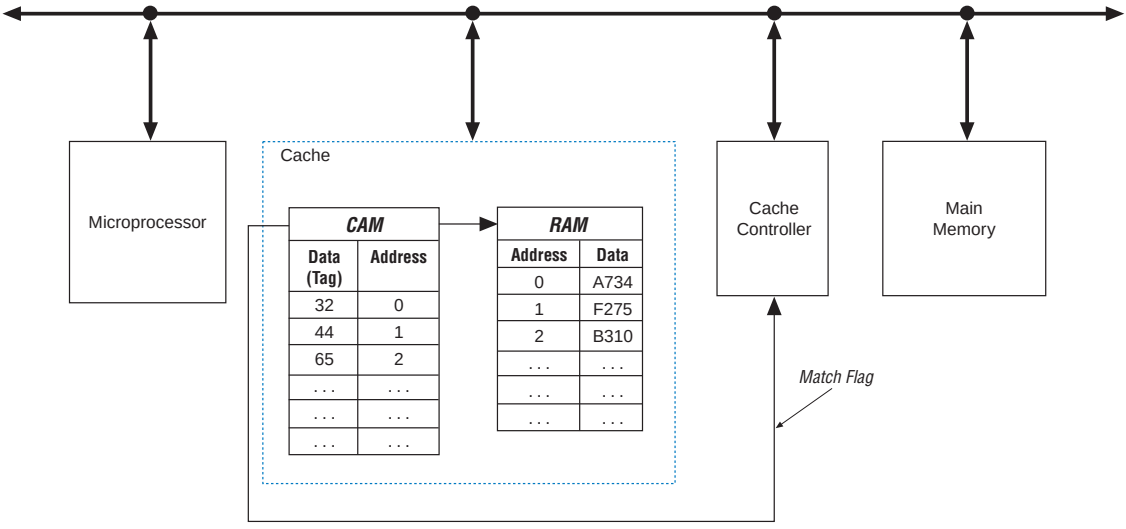
Cache Tags

Cache is high-speed memory that enables a microprocessor to quickly access a subset of data from the main memory. The microprocessor can access data stored in cache much faster than data located in the main memory. Cache stores recently used items in a small amount of fast memory; recently accessed words replace previously used words. Cache uses both CAM and RAM to store data; CAM stores the address, or tag, where the data can be found in RAM, and RAM contains the actual data. For optimal performance, both CAM and RAM should be embedded into the same device.

When requesting data, the microprocessor submits a data tag to the cache. The cache compares the tag requested by the microprocessor with tags stored in the CAM tag field. All tags in CAM are compared simultaneously (in parallel) with the requested tag. If the tag is located in the CAM block (i.e., a match is found), CAM's match flag goes high. CAM also sends the address of the data to RAM, which in turn outputs the requested data to the microprocessor. Figure 15 diagrams this process.

If CAM does not find a match, CAM's match flag goes low, and the cache controller transfers the requested data from the main memory into cache. The new data and address are stored in RAM and CAM, respectively, and replace previously used data. Only the most recently used data is stored in cache.

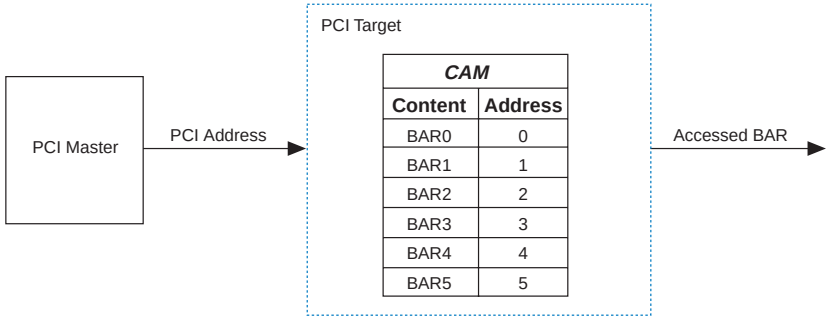
Figure 15. Searching CAM with the Tag Field in Cache



PCI Applications

In a system that utilizes a dynamic memory map, you can use CAM to store memory addresses for faster access. For example, in a peripheral component interconnect (PCI) system, a single PCI device may contain up to six memory spaces allocated in system memory. The exact location of these spaces is determined on power-up, and their starting locations are written into the PCI interface's six base address registers (BARs). When a PCI master requests access to a PCI device's memory location, CAM can be used to match the request to the address in memory quickly, as shown in Figure 16. To recognize which BAR is being called, CAM compares the requested address against the stored base addresses.

Figure 16. PCI Master Request



By using CAM for PCI applications, the PCI master can locate the requested BAR faster. The CAM implementation also requires fewer device resources than a LE implementation.

Conclusion

CAM can be used to accelerate a variety of search applications. By embedding CAM into the APEX 20KE architecture, Altera improves the performance of memory searches without on- and off-chip delays.

Revision History

The information contained in *Application Note 119 (Implementing High-Speed Search Applications with APEX CAM)* version 1.01 supersedes information published in previous versions. *Application Note 119 (Implementing High-Speed Search Applications with APEX CAM)* version 1.01 contains an update to Figure 2 on page 3.