burst, the overall system design is likely to be easier, because tight DDR timing on the address/control interface is not required as it would be with a two-word burst QDR device.

As can be readily observed, a QDR device can truly provide four times the bandwidth of a conventional SDR SRAM, but only when the read and write bandwidths are symmetrical. If an application requires very high bandwidth for a long set of writes and then the same equivalent bandwidth for a long read, QDR technology will not provide any real advantage over a DDR SRAM. QDR is useful in many communications applications where it serves as an in-line buffer or FIFO between two data processing elements. Such applications exhibit symmetrical bandwidth, because they cannot store data for long and must rapidly drain data buffer as fast as data is stored to prevent an overflow or underflow.

8.5 CONTENT ADDRESSABLE MEMORY

Most types of memory are constructed from an array of data storage locations, each of which is indexed with a unique address. The set of addresses supported by the memory array is a continuous, linear range from 0 to some upper limit, usually a power of 2. For a memory array size, W, the required address bus width, N, is determined by rounding up $N = \log_2 W$ to the next whole number. Therefore, $W \leq 2^N$. Memory arrays usually store sets of data that are accessed in a sequential manner. The basic paradigm is that the microprocessor requests an arbitrary address that has no special meaning other than the fact that it is a memory index, and the appropriate data is transferred. This scheme can be modified to implement a lookup table by presenting an index that is not an arbitrary number but that actually has some inherent meaning. If, for example, a network packet arrives with an eight-bit identification tag (perhaps a source address), that tag can be used to index into a memory array to retrieve or store status information about that unique type of packet. Such status information could help implement a filter, where a flag bit indicates whether packets with certain tags should be discarded or allowed through. It could also be used to implement a unique counter for each tag to maintain statistics of how many packets with a particular tag have been observed. As shown in Fig. 8.17, when the packet arrives, its relevant eight-bit tag is used to access a single memory location that contains a filter bit and a count value that gets incremented and stored back into the memory array. This saves logic, because 256 unique counters are not required. Instead, a single +1 adder accomplishes the task with the assistance of the memory's built-in address decoding logic.

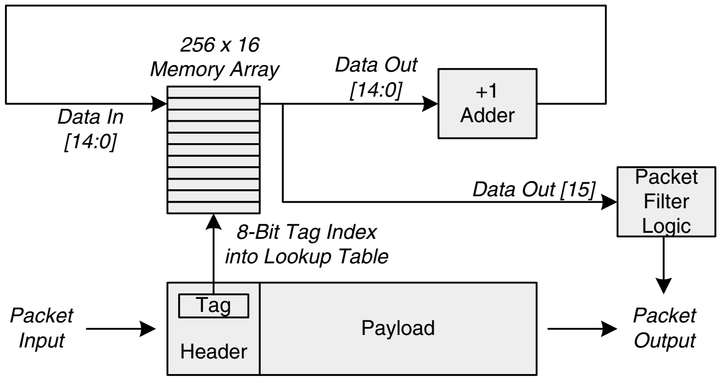

FIGURE 8.17 Using a memory array as a lookup table.

Such lookup tables are common in communication systems where decisions are made and statistics gathered according to the unique tags and network addresses present in each packet's header. When the size of a tag is bounded at a manageable width, a conventional memory array can be used to implement a lookup table. However, as tags grow to 16, 32, 64, 128, or more bits, the required memory size becomes quite impractical. The example in Fig. 8.17 would require 8 GB of memory if the tag width increased from 8 to 32 bits! If all 2^{32} tag permutations need to be accounted for independently, there would be no avoiding a large memory array. However, the majority of such lookup table applications handle a small fraction of the total set of permutations. The working set of tags sparsely populates the complete defined set of tags. So the question becomes how to rapidly index into a memory array with an N-bit tag where the array size is much less than 2^N.

A *content addressable memory* (CAM) solves this problem with an array of fully associative tags and optional corresponding data entries as shown in Fig. 8.18. Instead of decoding 2^N unique locations based on an N-bit tag, each CAM entry simultaneously matches its own tag to the one presented. The entry whose tag matches is the one that presents its associated data at the output and the one that can have its data modified as well. Alternatively, a CAM may simply return the index of the matched or winning entry in the array, if the specific device does not have any data associated with each entry. There is substantial overhead in providing each entry with a unique tag and matching logic, making CAMs substantially more expensive than conventional memories on a per-bit basis. Their increased cost is justified in those applications that require rapid searching of large yet sparsely populated index ranges.

Unlike a conventional memory, a CAM must be managed by the system's hardware and/or software to function properly. The system must load the CAM entries with relevant tags and data. Care should be taken to keep tags unique, because there is no standard means of resolving the case in which two entries' tags match the tag input. Individual CAM implementations may specify how such conflicts are resolved. Some CAMs handle read/write maintenance functions through the same interface that tags are presented and matched. Other implementations provide a separate maintenance port. Having a separate maintenance port increases the number of pins on the CAM, adding complexity to the circuit board wiring, but it may decrease overall system complexity, because maintenance logic and data logic paths do not have to be shared.

CAM tags and matching logic can be constructed in either a *binary* or *ternary* manner. A binary CAM implements a standard tag of arbitrary width and a valid bit. These CAMs are well suited to situations in which exact tag matches are desired. A ternary CAM doubles the number of tag bits to

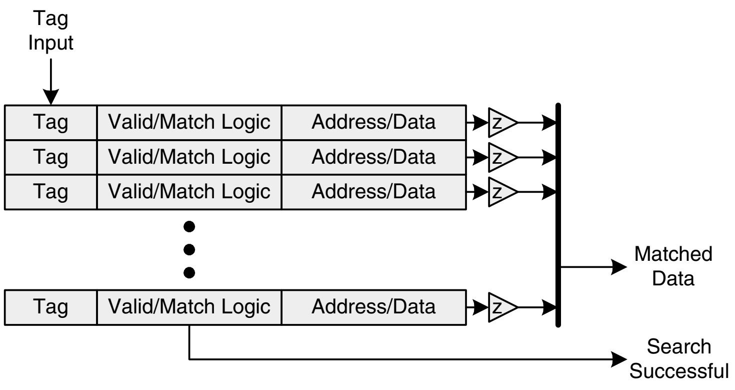

FIGURE 8.18 Basic CAM architecture.