22-bit combined address is sent to the 4-MB banked memory structure: 256 pages × 16 kB per page = 4 MB. These 22 bits are formed through the concatenation of the 8-bit bank address register and 14 of the microprocessor's low-order address bits, A[13:0]. The eight bank-address bits are changed infrequently whenever the microprocessor is ready for a new page in memory. The 14 microprocessor-address bits can change each time the window is accessed.

The details of a banking scheme can be modified according to the application's requirements. The bank access window can be increased or decreased, and more or fewer pages can be defined. If an application operates on many small sets of data, a larger number of smaller pages may be suitable. If the data or software set is widely dispersed, it may be better to increase the window size as much as possible to minimize the bank address register update rate.

While address banking can greatly increase the memory available to a microprocessor, it does so with the penalties of increased access time on page switches and more complexity in managing the segmented address space. Each time the microprocessor wants to access a location in a different page, it must update the bank address register. This penalty is acceptable in some applications. However, if the application requires both consistently fast access time and large memory size, a faster, more expensive microprocessor may be required that suits these needs.

The complexity of managing the segmented address space dissuades some engineers from employing address banking. Software usually bears the brunt of recognizing when necessary data resides in a different page and then updating the bank address register to access that page. It is easier for software to deal with a large, continuous address space. With the easy availability and low cost of 32-bit microprocessors, address banking is not as common as it used to be. However, if an 8-bit microprocessor must be used for cost reduction or other limitations, address banking may be useful when memory demands increase beyond 64 kB.

3.7 DIRECT MEMORY ACCESS

Transferring data from one region of memory to another is a common task performed within a computer. Incoming data may be transferred from a serial communications controller into memory, and outgoing data may be transferred from memory to the controller. Memory-to-memory transfers are common, too, as data structures are moved between subprograms, each of which may have separate regions of memory set aside for its private use. The speed with which memory is transferred normally depends on the time that the microprocessor takes to perform successive read and write operations. Each byte transferred requires several microprocessor operations: load accumulator, store accumulator, update address for next byte, and check if there is more data. Instead of simply moving a stream of bytes without interruption, the microprocessor is occupied mostly by the overhead of calculating new addresses and checking to see if more data is waiting. Computers that perform a high volume of memory transfers may exhibit performance bottlenecks as a result of the overhead of having the microprocessor spend too much of its time reading and writing memory.

Memory transfer performance can be improved using a technique called *direct memory access*, or DMA. DMA logic intercedes at the microprocessor's request to directly move data between a source and destination. A *DMA controller* (DMAC) sits on the microprocessor bus and contains logic that is specifically designed to rapidly move data without the overhead of simultaneously fetching and decoding instructions. When the microprocessor determines that a block of data is ready to move, it programs the DMAC with the starting address of the source data, the number of bytes to move, and the starting address of the destination data. When the DMAC is triggered, the microprocessor temporarily relinquishes control of its bus so the DMAC can take over and quickly move the data. The DMAC serves as a surrogate processor by directly generating addresses and reading and writing data. From the microprocessor bus perspective, nothing has changed, and data transfers proceed normally despite being controlled by the DMAC rather than the microprocessor. Figure 3.11 shows the basic internal structure of a DMAC.

A DMA transfer can be initiated by either the microprocessor or an I/O device that contains logic to assert a request to the DMAC. DMA transfers are generally broken into two categories: peripheral/memory and memory/memory. Peripheral/memory transfers move data to a peripheral or retrieve data from a peripheral. A peripheral/memory transfer can be triggered by a DMA-aware I/O-device when it is ready to accept more outgoing data or incoming data has arrived. These are called *single-address transfers,* because the DMAC typically controls only a single address—that of the memory side of the transfer. The peripheral address is typically a fixed offset into its register set and is asserted by supporting control logic that assists in the connectivity between the peripheral and the DMAC.

DMA transfers do not have to be continuous, and they are often not in the case of a peripheral transfer. If the microprocessor sets up a DMA transfer from a serial communications controller to memory, it programs the DMAC to write a certain quantity of data into memory. However, the transfer does not begin until the serial controller asserts a DMA request indicating that data is ready. When this request occurs, the DMAC arbitrates for access to the microprocessor bus by asserting a bus request. Some time later, the microprocessor or its support logic will grant the bus to the DMAC and temporarily pause the microprocessor's bus activity. The DMAC can then transfer a single unit of data from the serial controller into memory. The unit of data transfer may be any number of bytes. When finished, the DMAC relinquishes control of the bus back to the microprocessor.

Memory/memory transfers move data from one region in memory to another. These are called *dual-address transfers,* because the DMAC controls two addresses into memory—source and destination. Memory/memory transfers are triggered by the microprocessor and can execute continuously, because the data block to be moved is ready and waiting in memory.

Even when DMA transfers execute one byte at a time, they are still more efficient than the microprocessor, because the DMAC is capable of transferring a byte or word (per the microprocessor's data bus width) in a single bus cycle rather than the microprocessor's load/store mechanism with additional overhead. There is some initial overhead in setting up the DMA transfer, so it is not efficient to use DMA for very short transfers. If the microprocessor needs to move only a few bytes, it should probably do so on its own. However, the DMAC initialization overhead is more than compensated for if dozens or hundreds of bytes are being moved.

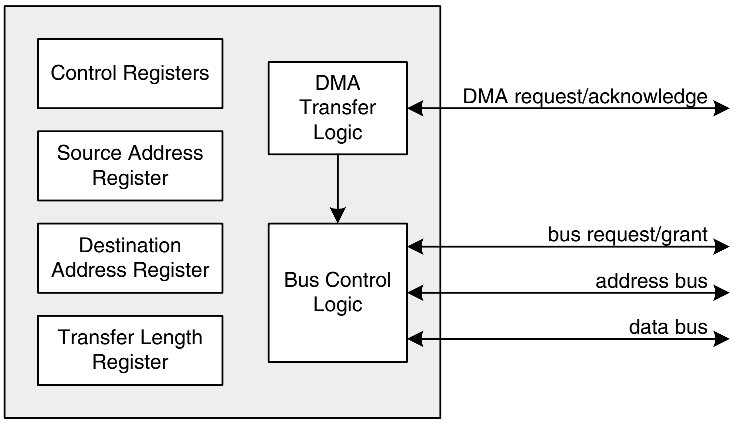

FIGURE 3.11 DMA controller block diagram.