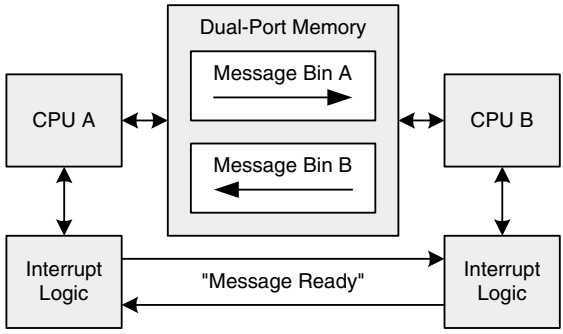

FIGURE 4.17 Dual microprocessor message passing architecture.

4.8 THE FIFO

The memory devices discussed thus far are essentially linear arrays of bits surrounded by a minimal quantity of interface logic to move bits between the port(s) and the array. *First-in-first-out* (FIFO) memories are special-purpose devices that implement a basic queue structure that has broad application in computer and communications architecture. Unlike other memory devices, a typical FIFO has two unidirectional ports without address inputs: one for writing and another for reading. As the name implies, the first data written is the first read, and the last data written is the last read. A FIFO is not a random access memory but a sequential access memory. Therefore, unlike a conventional memory, once a data element has been read once, it cannot be read again, because the next read will return the next data element written to the FIFO. By their nature, FIFOs are subject to *overflow* and *underflow* conditions. Their finite size, often referred to as *depth*, means that they can fill up if reads do not occur to empty data that has already been written. An overflow occurs when an attempt is made to write new data to a full FIFO. Similarly, an empty FIFO has no data to provide on a read request, which results in an underflow.

A FIFO is created by surrounding a dual-port memory array—generally SRAM, but DRAM could be made to work as well for certain applications—with a write pointer, a read pointer, and control logic as shown in Fig. 4.18.

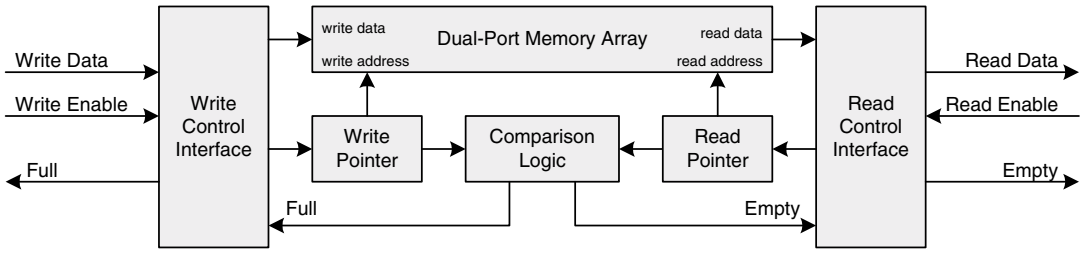

FIGURE 4.18 Basic FIFO architecture.

A FIFO is not addressed in a linear fashion; rather, it is made to form a continuous ring of memory that is addressed by the two internal pointers. The fullness of the FIFO is determined not by the absolute values of the pointers but by their relative values. An empty FIFO begins with its read and write pointers set to the same value. As entries are written, the write pointer increments. As entries are read, the read pointer increments as well. If the read pointer ever catches up to the write pointer such that the two match, the FIFO is empty again. If the read pointer fails to advance, the write pointer will eventually wrap around the end of the memory array and become equal to the read pointer. At this point, the FIFO is full and cannot accept any more data until reading resumes. Full and empty flags are generated by the FIFO to provide status to the writing and reading logic. Some FIFOs contain more detailed fullness status, such as signals that represent programmable fullness thresholds.

The interfaces of a FIFO can be asynchronous (no clock) or synchronous (with a clock). If synchronous, the two ports can be designed to operate with a common clock or different clocks. Although older asynchronous FIFOs are still manufactured, synchronous FIFOs are now more common. Synchronous FIFOs have the advantage of improved interface timing, because flops placed at a device's inputs and outputs reduce timing requirements to the familiar setup, hold, and clock-to-out specifications. Without such a registered interface, timing specifications become a function of the device's internal logic paths.

One common role that a FIFO fills is in clock domain crossing. In such an application, there is a need to communicate a series of data values from a block of logic operating on one clock to another block operating on a different clock. Exchanging data between clock domains requires special attention, because there is normally no way to perform a conventional timing analysis across two different clocks to guarantee adequate setup and hold times at the destination flops. Either an asynchronous FIFO or a dual-clock synchronous FIFO can be used to solve this problem, as shown in Fig. 4.19.

The dual-port memory at the heart of the FIFO is an asynchronous element that can be accessed by the logic operating in either clock domain. A dual-clock synchronous FIFO is designed to handle arbitrary differences in the clocks between the two halves of the device. When one or more bytes are written on clock A, the write-pointer information is carried safely across to the clock B domain within the FIFO via inter-clock domain synchronization logic. This enables the read-control interface to determine that there is data waiting to be read. Logic on clock B can read this data long after it has been safely written into the memory array and allowed to settle to a stable state.

Another common application for a FIFO is rate matching where a particular data source is bursty and the data consumer accepts data at a more regular rate. One example is a situation where a sequence of data is stored in DRAM and needs to be read out and sent over a communications interface one byte at a time. The DRAM is shared with a CPU that competes with the communications interface for memory bandwidth. It is known that DRAMs are most efficient when operated in a page-mode burst. Therefore, rather than perform a complete read-transaction each time a single byte

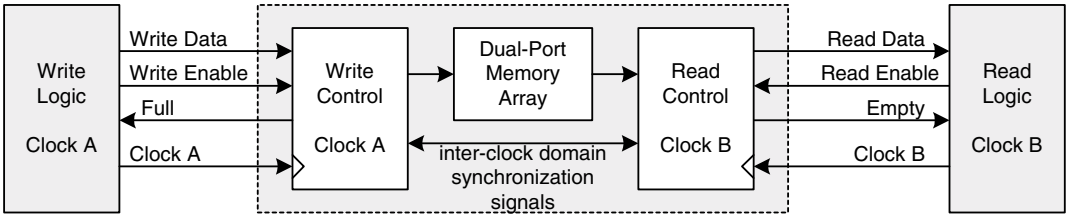

FIGURE 4.19 Clock domain crossing with synchronous FIFO.