output may not transition until the next clock cycle. Whether the input signal meets or misses the first flop's timing window, an extra cycle of latency is added by the second synchronizing flop.

For the synchronizing circuit to function properly, the asynchronous input must remain active for a sufficient time to be detected by the destination clock domain. If the destination clock has a 20-ns period, pulses shorter than 20 ns will have a low probability of being detected. It is best to guarantee that the pulse is active for several destination clock periods. When the two clock frequencies in a clock domain crossing are known, it is possible to determine a safe minimum pulse width. There are also situations in which one or both clock frequencies are unknown or variable. A circuit may need to run at a range of frequencies. Microprocessors are a good example of this, because they are designed to operate over a range of frequencies determined by an engineer for a specific project.

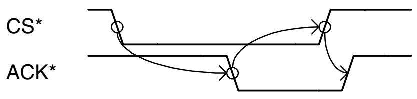

FIGURE 10.13 Four-corner microprocessor bus handshake.

In situations in which the clock frequencies are variable, asynchronous interfaces must be self-regulating to guarantee minimum pulse widths such that each clock domain can properly detect signals asserted by the other domain. *Four-corner handshaking* is a popular way to implement a self-regulated control interface. Four-corner handshaking establishes a rule that signals are not deasserted until a corresponding signal from the other entity is observed to transition, indicating that the locally generated signal has been properly detected by the remote entity. In the case of a request/acknowledge microprocessor interface, chip select is not deasserted until acknowledge is asserted, and acknowledge is not deasserted until chip select is deasserted as shown in Fig. 10.13.

Once causal control signals are properly synchronized across a clock domain boundary, arbitrary data paths can be synchronized as well. A register or data bus cannot simply be passed through a two-stage synchronizer, because there is no way of knowing if all the members of that bus passed through the flops uniformly. Even if the delays of each signal are closely matched, individual flops have minute physical differences that can cause their metastable characteristics to differ. Unlike a single control signal with just two states, an N-bit bus has 2^N states. If the control signal changes one cycle earlier or later, there is no misinterpretation of its activity. The latter case can lead to some bits transitioning before others, resulting in random values appearing at the synchronizer output for one or two clock cycles.

A basic method for synchronizing a data path is to associate it with a control signal that is passed through a synchronizer. As shown in Fig. 10.14, the data source drives a data valid signal at the same time as the data and holds the two entities stable for a minimum duration (achieved by either calculation or four-corner handshaking). The destination uses a two-flop synchronizer to move the control signal to its clock domain. The synchronization process has an inherent latency during which it is guaranteed that the data path reaches the destination logic. When the destination detects the properly synchronized control signal, it samples the data path, which by now has been stable at the destination's inputs for well over a full clock cycle and thereby meets the input flops' setup time. Metastability of the data path is avoided by an inherent synchronization time delay during which the data remains static. This scheme relies on the assumption of relatively uniform propagation delay between the data valid signal and the data path, an assumption that is achievable in most circumstances.

Synchronizing a data path with discrete control signals is a straightforward approach requiring little overhead. Its disadvantage is that multiple clock cycles are necessary to move a single unit of data. When higher transfer efficiency is necessary, a FIFO is the means of synchronizing data paths across clock domain boundaries. The FIFO must operate on two separate clocks, unlike a conventional synchronous FIFO with just a single clock. A FIFO enables data to be continuously written on one clock and read on another, subject to limitations imposed by data rate mismatches at the write

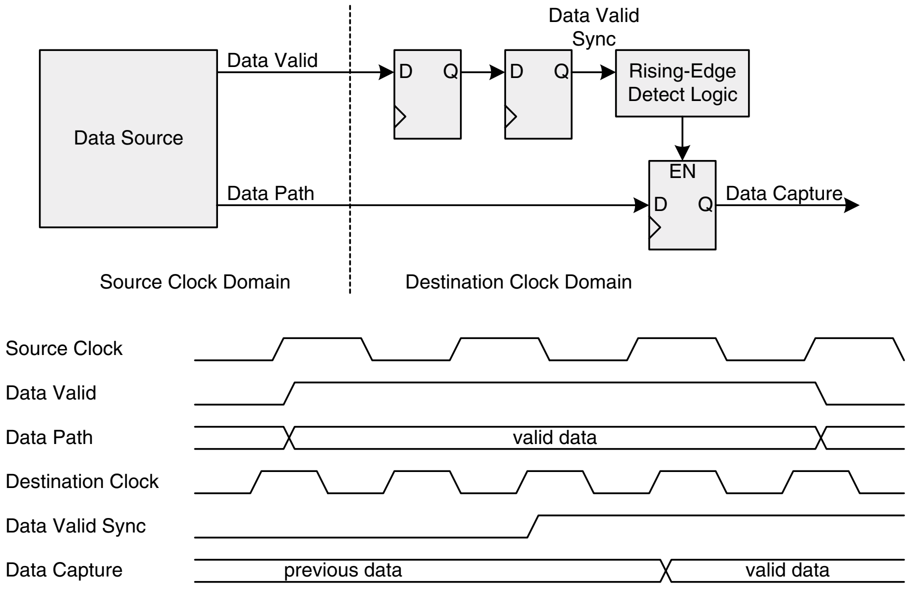

FIGURE 10.14 Data path synchronization.

and read ports. If a FIFO is used to carry 8-bit data from a 100-MHz clock to a 50-MHz clock, it is clear that the overall data rate cannot exceed the slower read rate, 50 MBps, without overflowing the FIFO and losing data. If the situation is reversed, there is still a 50-MB overall maximum data rate, and the 100-MHz read logic must inherently handle gaps in its data path, which has a 100-MB bandwidth. As long as the read and write data rates are matched over time, the dual-clock FIFO will never overflow or underflow and provides an efficient synchronization mechanism.

10.4 FINITE STATE MACHINES

Finite state machines (FSMs) are powerful design elements used to implement algorithms in hardware. A loose analogy is that an FSM is to logic design what programming is to software design. An FSM consists of a *state vector*, a set of registers, with associated logic that advances the state each clock cycle depending on external input and the current state. In this respect, an FSM is analogous to a set of software instructions that are sequenced via a microprocessor's program counter. Each state can be designed to take a different arbitrary action and branch to any other state in the same conceptual way that software branches to different program sections as its algorithm dictates. If a problem can be decomposed into a set of deterministic logical steps, that algorithm can be implemented with an FSM.

A counter is a simple example of an FSM, though its actions are very limited. Each state simply advances to the next state on each clock cycle. There is no conditional branching in a typical counter. FSMs are often represented graphically before being committed to RTL. Figure 10.15 shows a bubble diagram representation of a two-bit counter. Each state is represented by its own bubble, and arcs show the conditions that cause one state to lead to other states. An unlabeled arc is taken to mean un-