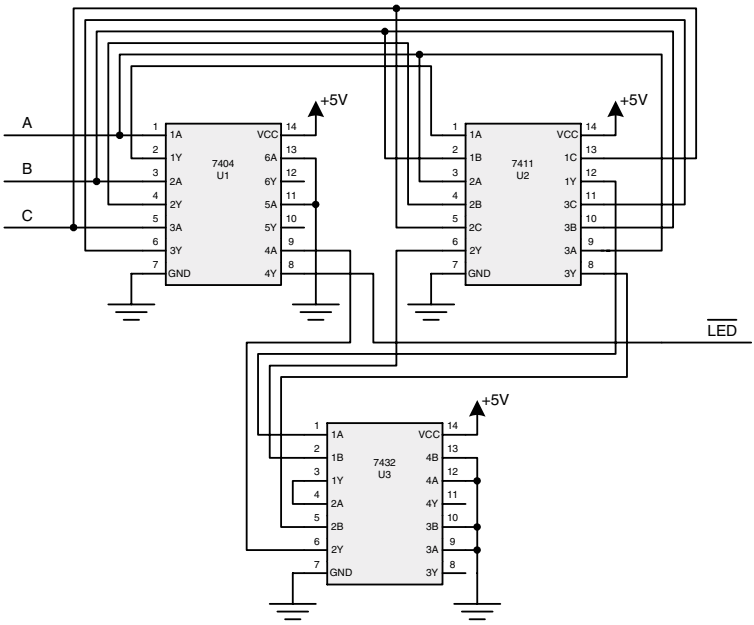

FIGURE 2.13 LED driver logic using 74111 with fewer ICs.

detected, the last bit is found by knowing that there are eight bits in a byte. During periods of inactivity, an idle communications interface is indicated by a persistent logic 0. When the transmitter is given a byte to send, it first drives a logic-1 start bit and then sends eight data bits. Each bit is sent in its own clock cycle. Therefore, nine clock cycles are required to transfer each byte. The serial interface is composed of two signals, *clock* and *serial data*, and functions as shown in Fig. 2.14.

The eight data bits are sent from least-significant bit, bit 0, to most-significant bit, bit 7, following the start bit. Following the transmission of bit 7, it is possible to immediately begin a new byte by inserting a new start bit. This timing diagram does not show a new start bit directly following bit 7. The corresponding output of the receiver is shown in Fig. 2.15. Here, *data out* is the eight-bit quan-

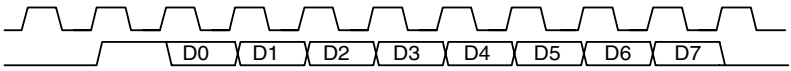

FIGURE 2.14 Serial interface bit timing.

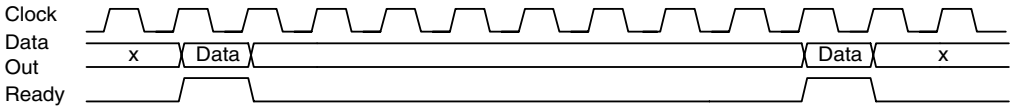

FIGURE 2.15 Serial receive output timing.

tity that has been reconstructed from the serialized bit stream of Fig. 2.14. *Ready* indicates when data out is valid and is active-high.

All that is required of this receiver is to assemble the eight data bits in their proper order and then generate a *ready* signal. This ready signal lasts only one cycle, and any downstream logic waiting for the newly arrived byte must process it immediately. In a real system, a register might exist to capture the received byte when ready goes active. This register would then pass the byte to the appropriate destination. This output timing shows two bytes transmitted back to back. They are separated by nine cycles, because each byte requires an additional start bit for framing.

In contemplating the design of the receive portion of the serial controller, the need for a serial-in/parallel-out shift register becomes apparent to assemble the individual bits into a whole byte. Additionally, some control logic is necessary to recognize the start bit, wait eight clocks to assemble the incoming byte, and then generate a ready signal. This receiver has two basic states, or modes, of operation: idle and receiving. When idling, no start bit has yet been detected, so there is no useful work to be done. When receiving, a start bit has been observed, incoming bits are shifted into the shift register, and then a ready signal is generated. As soon as the ready signal is generated, the receiver state may return to idle or remain in receiving if a new start bit is detected. Because there are two basic control logic states, the state can be stored in a single flip-flop, forming a two-state *finite state machine* (FSM). An FSM is formed by one or more *state flops* with accompanying logic to generate a new state for the next clock cycle based on the current cycle's state. The state is represented by the combined value of the state flops. An FSM with two state flops can represent four unique states. Each state can represent a particular step in an algorithm. The accompanying *state logic* controls the FSM by determining when it is time to transition to a new piece of the algorithm—a new state.

In the serial receive state machine, transitioning from idle to receiving can be done according to the serial data input, which is 0 when inactive and 1 when indicating a start bit. Transitioning back to idle must somehow be done nine cycles later. A counter could be used but would require some logic to sense a particular count value. Instead, a second shift register can be used to delay the start bit by nine cycles. When the start bit emerges from the last output bit in the shift register, the state machine can return to the idle state. Consider the logic in Fig. 2.16. The arrow-shaped boxes indicate connection points, or ports, of the circuit.

Under an idle condition, the input to the shift register is zero until the start bit appears at the data input, *din*. Nine cycles later, the ready bit emerges from the shift register. As soon as the start bit is observed, the state machine transitions to the receiving state, changing the *idle* input to 0, effectively masking further input to the shift register. This masking prevents nonzero data bits from entering the *ready* delay logic and causing false results.

Delaying the start bit by nine cycles solves one problem but creates another. The transition of the state machine back to idle is triggered by the emergence of *ready* from the shift register. Therefore, this transition will actually occur *ten* cycles after the start bit, because the state flop, like all D flip-flops, requires a single cycle of latency to propagate its input to its output. This additional cycle will prevent the control logic from detecting a new start bit immediately following the last data bit of the byte currently in progress. A solution is to design *ready* with its nine-cycle delay and *ready_next* with an eight-cycle delay by tapping off one stage earlier in the shift register. In doing so, the state

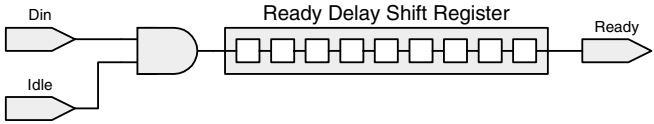

FIGURE 2.16 Serial receive ready delay.