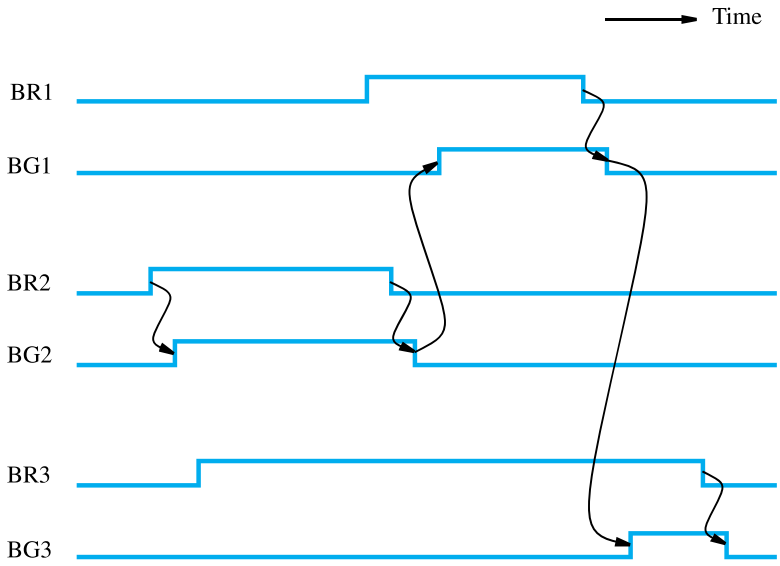

Figure 7.9 Granting use of the bus based on priorities.

the arbiter to the masters. A master requests use of the bus by activating its Bus-request line. If a single Bus-request is activated, the arbiter activates the corresponding Bus-grant. This indicates to the selected master that it may now use the bus for transferring data. When the transfer is completed, that master deactivates its Bus-request, and the arbiter deactivates its Bus-grant.

Figure 7.9 illustrates a possible sequence of events for the case of three masters. Assume that master 1 has the highest priority, followed by the others in increasing numerical order. Master 2 sends a request to use the bus first. Since there are no other requests, the arbiter grants the bus to this master by asserting BG2. When master 2 completes its data transfer operation, it releases the bus by deactivating BR2. By that time, both masters 1 and 3 have activated their request lines. Since device 1 has a higher priority, the arbiter activates BG1 after it deactivates BG2, thus granting the bus to master 1. Later, when master 1 releases the bus by deactivating BR1, the arbiter deactivates BG1 and activates BG3 to grant the bus to master 3. Note that the bus is granted to master 1 before master 3 even though master 3 activated its request line before master 1.

7.4 INTERFACE CIRCUITS

The I/O interface of a device consists of the circuitry needed to connect that device to the bus. On one side of the interface are the bus lines for address, data, and control. On the other side are the connections needed to transfer data between the interface and the I/O

device. This side is called a *port,* and it can be either a parallel or a serial port. A parallel port transfers multiple bits of data simultaneously to or from the device. A serial port sends and receives data one bit at a time. Communication with the processor is the same for both formats; the conversion from a parallel to a serial format and vice versa takes place inside the interface circuit.

Before we present specific circuit examples, let us recall the functions of an I/O interface. According to the discussion in Section 3.1, an I/O interface does the following:

1. Provides a register for temporary storage of data
2. Includes a status register containing status information that can be accessed by the processor
3. Includes a control register that holds the information governing the behavior of the interface
4. Contains address-decoding circuitry to determine when it is being addressed by the processor
5. Generates the required timing signals
6. Performs any format conversion that may be necessary to transfer data between the processor and the I/O device, such as parallel-to-serial conversion in the case of a serial port

7.4.1 PARALLEL INTERFACE

We now explain the key aspects of interface design by means of examples. First, we describe an interface circuit for an 8-bit input port that can be used for connecting a simple input device, such as a keyboard. Then, we describe an interface circuit for an 8-bit output port, which can be used with an output device such as a display. We assume that these interface circuits are connected to a 32-bit processor that uses memory-mapped I/O and the asynchronous bus protocol depicted in Figures 7.6 and 7.7.

Input Interface

Figure 7.10 shows a circuit that can be used to connect a keyboard to a processor. The registers in this circuit correspond to those given in Figure 3.3. Assume that interrupts are not used, so there is no need for a control register. There are only two registers: a data register, KBD_DATA, and a status register, KBD_STATUS. The latter contains the keyboard status flag, KIN.

A typical keyboard consists of mechanical switches that are normally open. When a key is pressed, its switch closes and establishes a path for an electrical signal. This signal is detected by an encoder circuit that generates the ASCII code for the corresponding character. A difficulty with such mechanical pushbutton switches is that the contacts *bounce* when a key is pressed, resulting in the electrical connection being made then broken several times before the switch settles in the closed position. Although bouncing may last only one or two milliseconds, this is long enough for the computer to erroneously interpret a single pressing of a key as the key being pressed and released several times. The effect of bouncing can be eliminated using a simple debouncing circuit, which could be part of the keyboard hardware

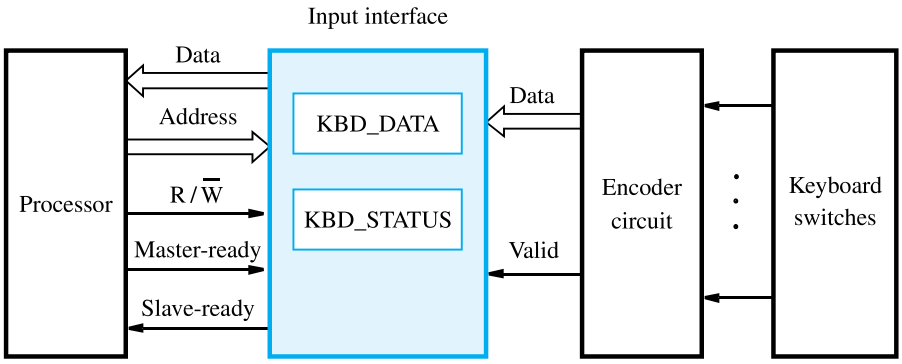

Figure 7.10 Keyboard to processor connection.

or may be incorporated in the encoder circuit. Alternatively, switch bouncing can be dealt with in software. The software detects that a key has been pressed when it observes that the keyboard status flag, KIN, has been set to 1. The I/O routine can then introduce sufficient delay before reading the contents of the input buffer, KBD_DATA, to ensure that bouncing has subsided. When debouncing is implemented in hardware, the I/O routine can read the input character as soon as it detects that KIN is equal to 1.

The output of the encoder in Figure 7.10 consists of one byte of data representing the encoded character and one control signal called Valid. When a key is pressed, the Valid signal changes from 0 to 1, causing the ASCII code of the corresponding character to be loaded into the KBD_DATA register and the status flag KIN to be set to 1. The status flag is cleared to 0 when the processor reads the contents of the KBD_DATA register. The interface circuit is shown connected to an asynchronous bus on which transfers are controlled by the handshake signals Master-ready and Slave-ready, as in Figure 7.6. The bus has one other control line, $R/\overline{W}$, which indicates a Read operation when equal to 1.

Figure 7.11 shows a possible circuit for the input interface. There are two addressable locations in this interface, KBD_DATA and KBD_STATUS. They occupy adjacent word locations in the address space, as in Figure 3.3. Only one bit, b_1, in the status register actually contains useful information. This is the keyboard status flag, KIN. When the status register is read by the processor, all other bit locations appear as containing zeros.

When the processor requests a Read operation, it places the address of the appropriate register on the address lines of the bus. The address decoder in the interface circuit examines bits A_{31-3}, and asserts its output, My-address, when one of the two registers KBD_DATA or KBD_STATUS is being addressed. Bit A_2 determines which of the two registers is involved. Hence, a multiplexer is used to select the register to be connected to the bus based on address bit A_2. The two least-significant address bits, A_1 and A_0, are not used, because we have assumed that all addresses are word-aligned.

The output of the multiplexer is connected to the data lines of the bus through a set of tri-state gates. The interface circuit turns the tri-state gates on only when the three signals Master-ready, My_address, and $R/\overline{W}$ are all equal to 1, indicating a Read operation. The

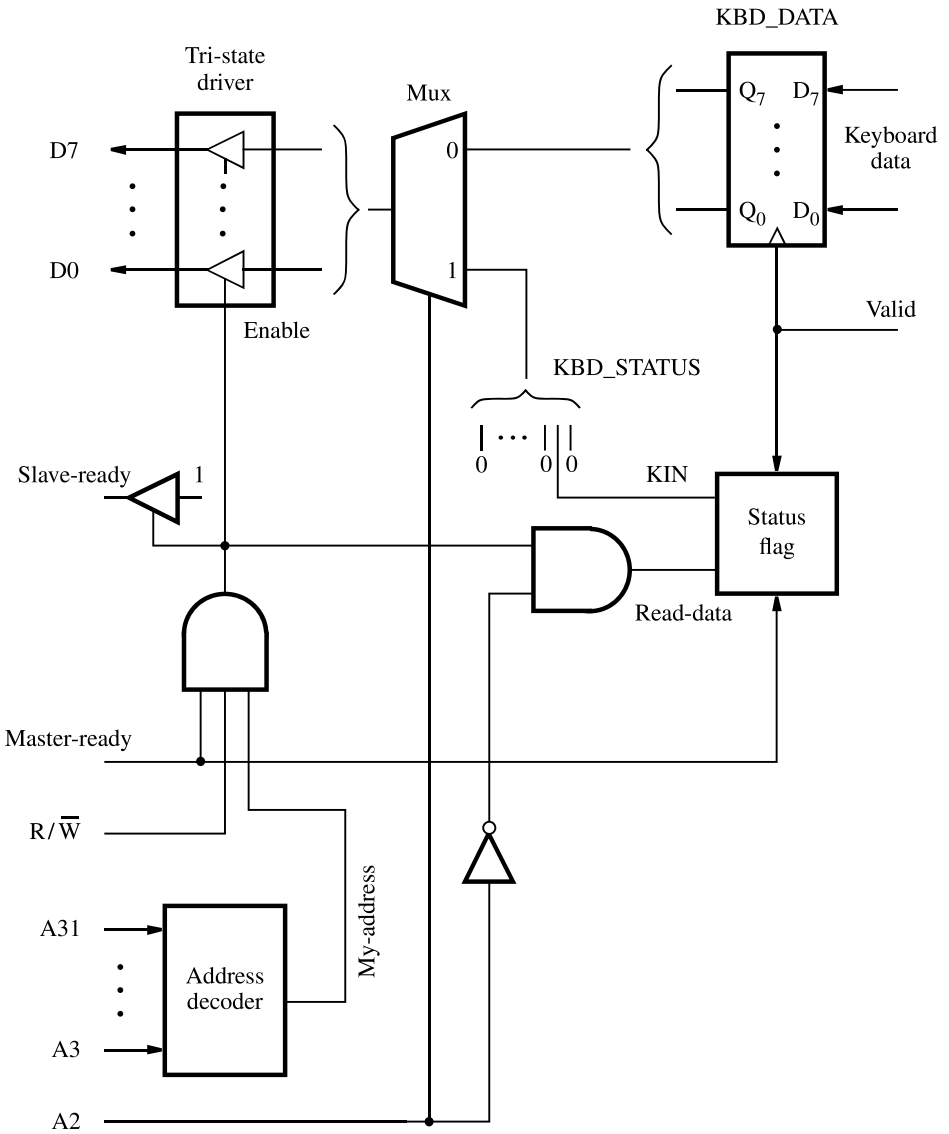

Figure 7.11 An input interface circuit.

Slave-ready signal is asserted at the same time, to inform the processor that the requested data or status information has been placed on the data lines. When address bit A_2 is equal to 0, Read-data is also asserted. This signal is used to reset the KIN flag.

A possible implementation of the status flag circuit is given in Figure 7.12. The KIN flag is the output of a NOR latch connected as shown. A flip-flop is set to 1 by the rising edge on the Valid signal line. This event changes the state of the NOR latch to set KIN to

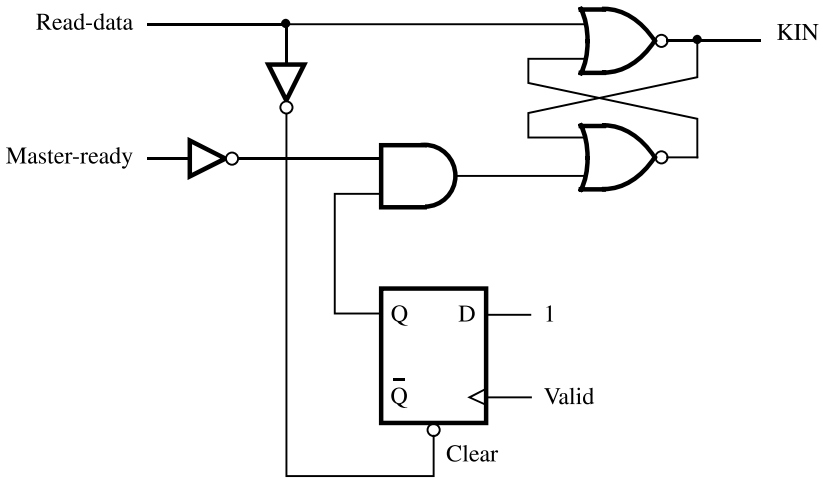

Figure 7.12 Circuit for the status flag block in Figure 7.11.

1, but only when Master-ready is low. The reason for this additional condition is to ensure that KIN does not change state while being read by the processor. Both the flip-flop and the latch are reset to 0 when Read-data becomes equal to 1, indicating that KBD_DATA is being read.

The circuits shown in Figures 7.11 and 7.12 are intended to illustrate the various functions that an interface circuit needs to perform. A designer using modern computer-aided design tools would specify these functions using a hardware description language such as VHDL or Verilog. The resulting circuits would depend on the technology used and may or may not be the same as the circuits shown in these figures.

Output Interface

Let us now consider the output interface shown in Figure 7.13, which can be used to connect an output device such as a display. We have assumed that the display uses two handshake signals, New-data and Ready, in a manner similar to the handshake between the bus signals Master-ready and Slave-ready. When the display is ready to accept a character, it asserts its Ready signal, which causes the DOUT flag in the DISP_STATUS register to be set to 1. When the I/O routine checks DOUT and finds it equal to 1, it sends a character to DISP_DATA. This clears the DOUT flag to 0 and sets the New-data signal to 1. In response, the display returns Ready to 0 and accepts and displays the character in DISP_DATA. When it is ready to receive another character, it asserts Ready again, and the cycle repeats.

Figure 7.14 shows an implementation of this interface. Its operation is similar to that of the input interface of Figure 7.11, except that it responds to both Read and Write operations. A Write operation in which $A_2 = 0$ loads a byte of data into register DISP_DATA. A Read operation in which $A_2 = 1$ reads the contents of the status register DISP_STATUS. In this case, only the DOUT flag, which is bit b_2 of the status register, is sent by the interface. The remaining bits of DISP_STATUS are not used. The state of the status flag is determined

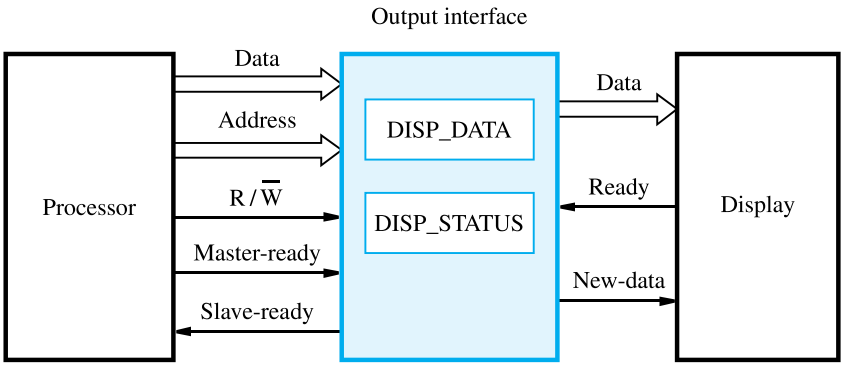

Figure 7.13 Display to processor connection.

by the handshake control circuit. A state diagram describing the behavior of this circuit is given as Example 7.4 at the end of the chapter.

7.4.2 SERIAL INTERFACE

A serial interface is used to connect the processor to I/O devices that transmit data one bit at a time. Data are transferred in a bit-serial fashion on the device side and in a bit-parallel fashion on the processor side. The transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability. A block diagram of a typical serial interface is shown in Figure 7.15. The input shift register accepts bit-serial input from the I/O device. When all 8 bits of data have been received, the contents of this shift register are loaded in parallel into the DATAIN register. Similarly, output data in the DATAOUT register are transferred to the output shift register, from which the bits are shifted out and sent to the I/O device.

The part of the interface that deals with the bus is the same as in the parallel interface described earlier. Two status flags, which we will refer to as SIN and SOUT, are maintained by the Status and control block. The SIN flag is set to 1 when new data are loaded into DATAIN from the shift register, and cleared to 0 when these data are read by the processor. The SOUT flag indicates whether the DATAOUT register is available. It is cleared to 0 when the processor writes new data into DATAOUT and set to 1 when data are transferred from DATAOUT to the output shift register.

The double buffering used in the input and output paths in Figure 7.15 is important. It is possible to implement DATAIN and DATAOUT themselves as shift registers, thus obviating the need for separate shift registers. However, this would impose awkward restrictions on the operation of the I/O device. After receiving one character from the serial line, the interface would not be able to start receiving the next character until the processor reads the contents of DATAIN. Thus, a pause would be needed between two characters to give the processor time to read the input data. With double buffering, the transfer of the second character can begin as soon as the first character is loaded from the shift register into the