CHAPTER 8

Shift Registers

CHAPTER OUTLINE

CHAPTER OBJECTIVES

- Identify the basic forms of data movement in shift registers
- Explain how serial in/serial out, serial in/parallel out, parallel in/serial out, and parallel in/parallel out shift registers operate
- Describe how a bidirectional shift register operates
- Determine the sequence of a Johnson counter
- Set up a ring counter to produce a specified sequence
- Construct a ring counter from a shift register
- Use a shift register as a time-delay device
- Use a shift register to implement a serial-to-parallel data converter
- Implement a basic shift-register-controlled keyboard encoder
- Interpret ANSI/IEEE Standard 91-1984 shift register symbols with dependency notation
- Use shift registers in a system application

KEY TERMS

Key terms are in order of appearance in the chapter.

- Register
- Stage
- Load
- Bidirectional

VISIT THE WEBSITE

Study aids for this chapter are available at http://www.pearsonglobaleditions.com/floyd

INTRODUCTION

Shift registers are a type of sequential logic circuit used primarily for the storage of digital data and typically do not possess a characteristic internal sequence of states. There are exceptions, however, and these are covered in Section 8–4.

In this chapter, the basic types of shift registers are studied and several applications are presented. Also, a troubleshooting method is introduced.

8–1 Shift Register Operations

Shift registers consist of arrangements of flip-flops and are important in applications involving the storage and transfer of data in a digital system. A register has no specified sequence of states, except in certain very specialized applications. A register, in general, is used solely for storing and shifting data (1s and 0s) entered into it from an external source and typically possesses no characteristic internal sequence of states.

After completing this section, you should be able to

- Explain how a flip-flop stores a data bit
- Define the storage capacity of a shift register
- Describe the shift capability of a register

A register can consist of one or more flip-flops used to store and shift data.

A **register** is a digital circuit with two basic functions: data storage and data movement. The storage capability of a register makes it an important type of memory device. Figure 8–1 illustrates the concept of storing a 1 or a 0 in a D flip-flop. A 1 is applied to the data input as shown, and a clock pulse is applied that stores the 1 by *setting* the flip-flop. When the 1 on the input is removed, the flip-flop remains in the SET state, thereby storing the 1. A similar procedure applies to the storage of a 0 by *resetting* the flip-flop, as also illustrated in Figure 8–1.

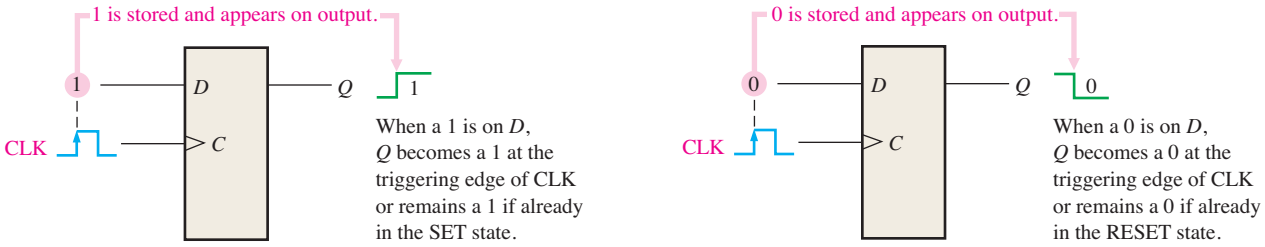


FIGURE 8–1 The flip-flop as a storage element.

The *storage capacity* of a register is the total number of bits (1s and 0s) of digital data it can retain. Each **stage** (flip-flop) in a shift register represents one bit of storage capacity; therefore, the number of stages in a register determines its storage capacity.

The *shift capability* of a register permits the movement of data from stage to stage within the register or into or out of the register upon application of clock pulses. Figure 8–2

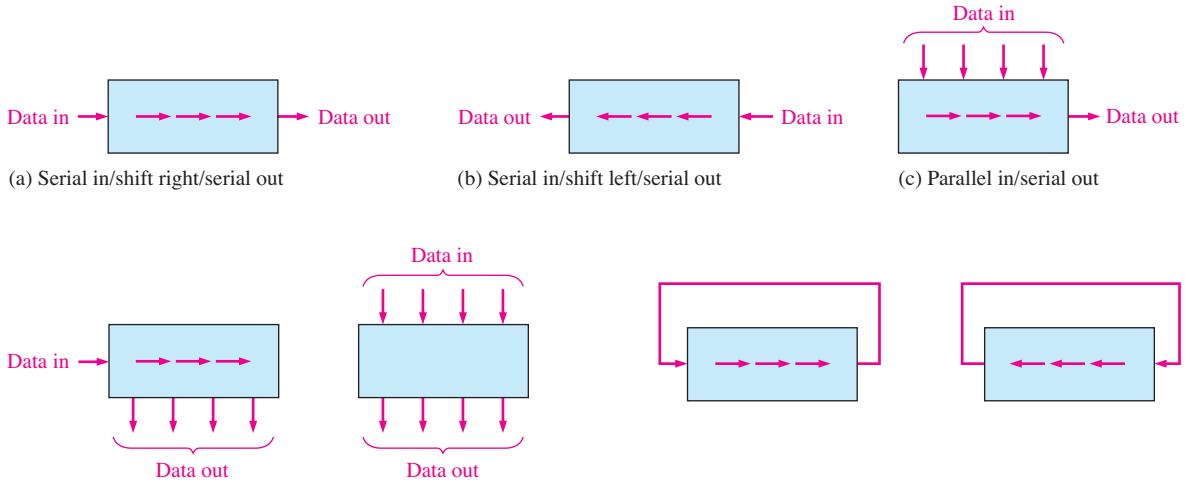


FIGURE 8–2 Basic data movement in shift registers. (Four bits are used for illustration. The bits move in the direction of the arrows.)

illustrates the types of data movement in shift registers. The block represents any arbitrary 4-bit register, and the arrows indicate the direction of data movement.

SECTION 8–1 CHECKUP

Answers are at the end of the chapter.

1. What determines the storage capacity of a shift register?
2. What two principal functions are performed by a shift register?

8–2 Types of Shift Register Data I/Os

In this section, four types of shift registers based on data input and output (inputs/outputs) are discussed: serial in/serial out, serial in/parallel out, parallel in/serial out, and parallel in/parallel out.

After completing this section, you should be able to

- Describe the operation of four types of shift registers
- Explain how data bits are entered into a shift register
- Describe how data bits are shifted through a register
- Explain how data bits are taken out of a shift register
- Develop and analyze timing diagrams for shift registers

Serial In/Serial Out Shift Registers

The serial in/serial out shift register accepts data serially—that is, one bit at a time on a single line. It produces the stored information on its output also in serial form. Let's first look at the serial entry of data into a typical shift register. Figure 8–3 shows a 4-bit device implemented with D flip-flops. With four stages, this register can store up to four bits of data.

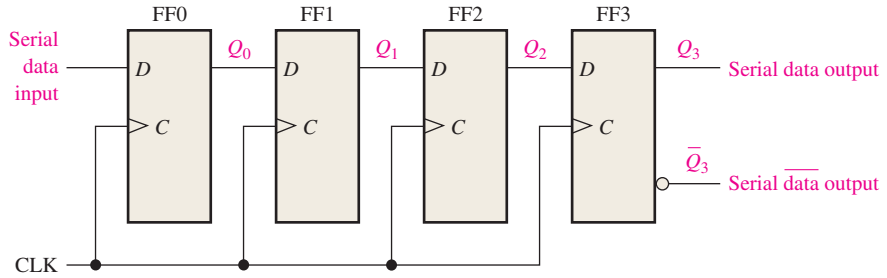


FIGURE 8–3 Serial in/serial out shift register.

InfoNote

Frequently, it is necessary to *clear* an internal register in a processor. For example, a register may be cleared prior to an arithmetic or other operation. One way that registers in a processor are cleared is using software to subtract the contents of the register from itself. The result, of course, will always be zero. For example, a processor instruction that performs this operation is SUB AL,AL. with this instruction, the register named AL is cleared.

Table 8–1 shows the entry of the four bits 1010 into the register in Figure 8–3, beginning with the least significant bit. The register is initially clear. The 0 is put onto the data input line, making $D = 0$ for FF0. When the first clock pulse is applied, FF0 is reset, thus storing the 0.

TABLE 8–1

Shifting a 4-bit code into the shift register in Figure 8–3. Data bits are indicated by a beige screen.

CLK	FF0 (Q_0)	FF1 (Q_1)	FF2 (Q_2)	FF3 (Q_3)
Initial	0	0	0	0
1	**0**	0	0	0
2	**1**	**0**	0	0
3	**0**	**1**	**0**	0
4	**1**	**0**	**1**	**0**

Next the second bit, which is a 1, is applied to the data input, making $D = 1$ for FF0 and $D = 0$ for FF1 because the D input of FF1 is connected to the Q_0 output. When the second clock pulse occurs, the 1 on the data input is shifted into FF0, causing FF0 to set; and the 0 that was in FF0 is shifted into FF1.

The third bit, a 0, is now put onto the data-input line, and a clock pulse is applied. The 0 is entered into FF0, the 1 stored in FF0 is shifted into FF1, and the 0 stored in FF1 is shifted into FF2.

For serial data, one bit at a time is transferred.

The last bit, a 1, is now applied to the data input, and a clock pulse is applied. This time the 1 is entered into FF0, the 0 stored in FF0 is shifted into FF1, the 1 stored in FF1 is shifted into FF2, and the 0 stored in FF2 is shifted into FF3. This completes the serial entry of the four bits into the shift register, where they can be stored for any length of time as long as the flip-flops have dc power.

If you want to get the data out of the register, the bits must be shifted out serially to the Q_3 output, as Table 8–2 illustrates. After CLK4 in the data-entry operation just described, the LSB, 0, appears on the Q_3 output. When clock pulse CLK5 is applied, the second bit appears on the Q_3 output. Clock pulse CLK6 shifts the third bit to the output, and CLK7 shifts the fourth bit to the output. While the original four bits are being shifted out, more bits can be shifted in. All zeros are shown being shifted in, after CLK8.

TABLE 8–2

Shifting a 4-bit code out of the shift register in Figure 8–3. Data bits are indicated by a beige screen.

CLK	FF0 (Q_0)	FF1 (Q_1)	FF2 (Q_2)	FF3 (Q_3)
Initial	**1**	**0**	**1**	**0**
5	0	**1**	**0**	**1**
6	0	0	**1**	**0**
7	0	0	0	**1**
8	0	0	0	0

EXAMPLE 8–1

Show the states of the 5-bit register in Figure 8–4(a) for the specified data input and clock waveforms. Assume that the register is initially cleared (all 0s).

Solution

The first data bit (1) is entered into the register on the first clock pulse and then shifted from left to right as the remaining bits are entered and shifted. The register contains $Q_4Q_3Q_2Q_1Q_0 = 11010$ after five clock pulses. See Figure 8–4(b).

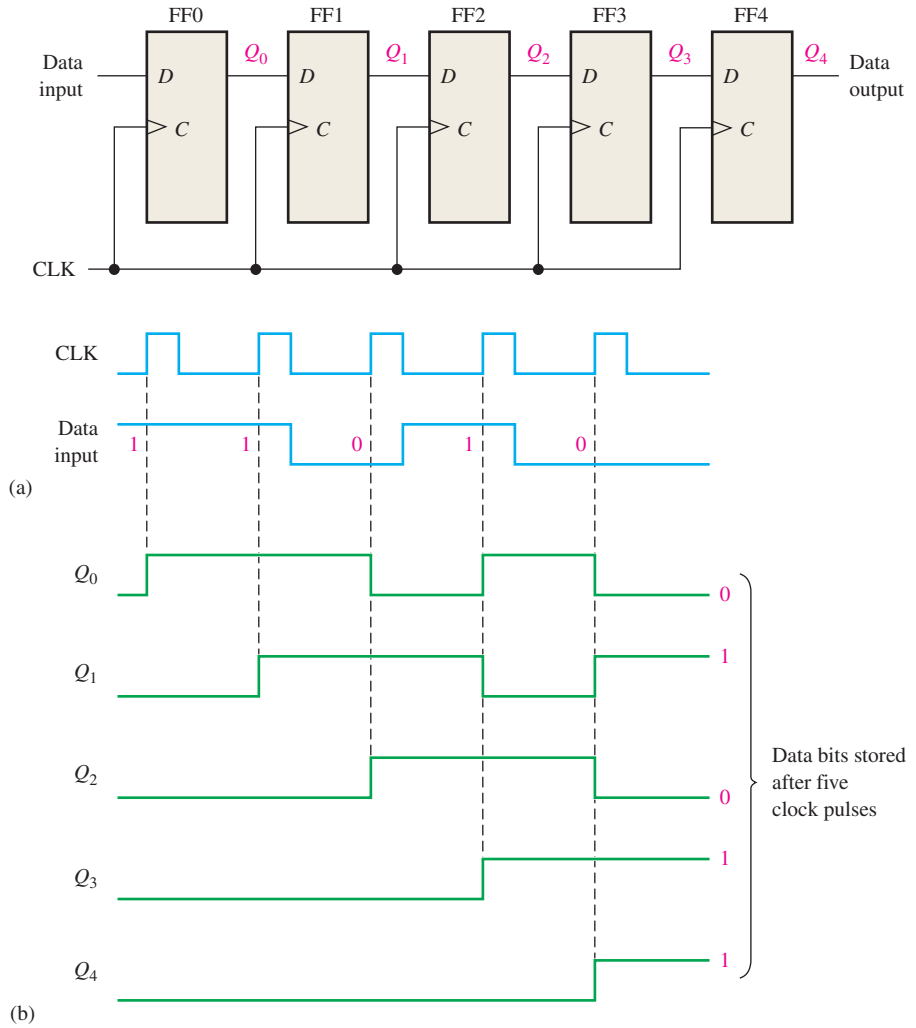


FIGURE 8–4 Open file F08-04 to verify operation. *A Multisim tutorial is available on the website.*

MultiSim

Related Problem*

Show the states of the register if the data input is inverted. The register is initially cleared.

*Answers are at the end of the chapter.

A traditional logic block symbol for an 8-bit serial in/serial out shift register is shown in Figure 8–5. The "SRG 8" designation indicates a shift register (SRG) with an 8-bit capacity.

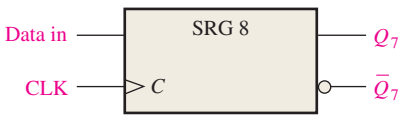


FIGURE 8–5 Logic symbol for an 8-bit serial in/serial out shift register.

Serial In/Parallel Out Shift Registers

Data bits are entered serially (least-significant bit first) into a serial in/parallel out shift register in the same manner as in serial in/serial out registers. The difference is the way in which the data bits are taken out of the register; in the parallel output register, the output of each stage is available. Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously, rather than on a bit-by-bit basis as with the serial output. Figure 8–6 shows a 4-bit serial in/parallel out shift register and its logic block symbol.

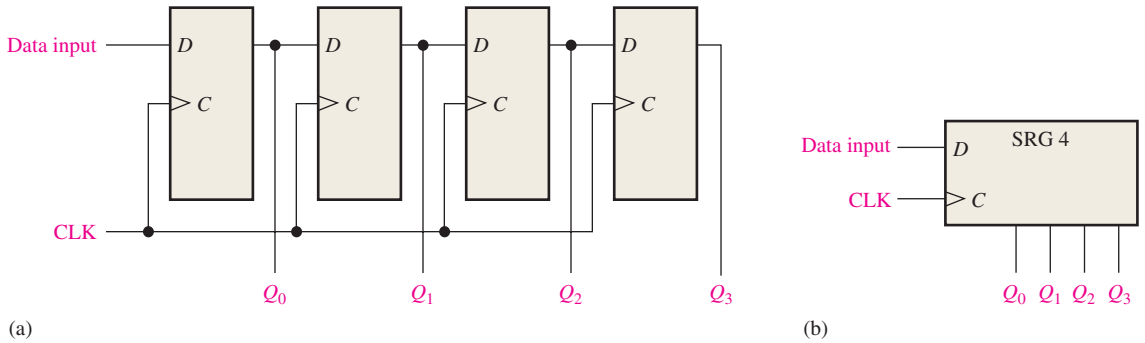


FIGURE 8–6 A serial in/parallel out shift register.

EXAMPLE 8–2

Show the states of the 4-bit register (SRG 4) for the data input and clock waveforms in Figure 8–7(a). The register initially contains all 1s.

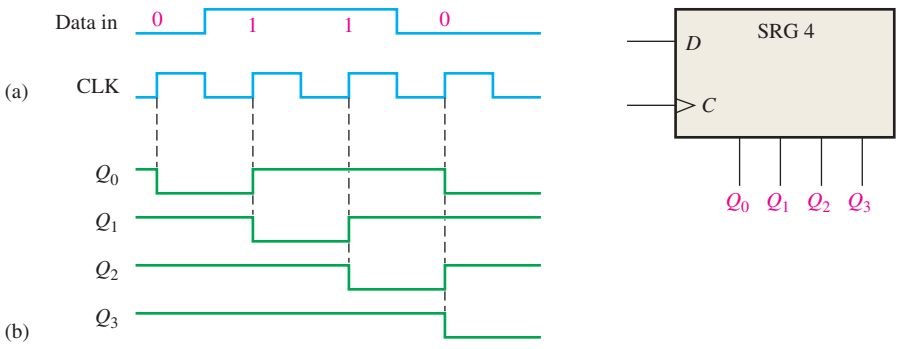


FIGURE 8–7

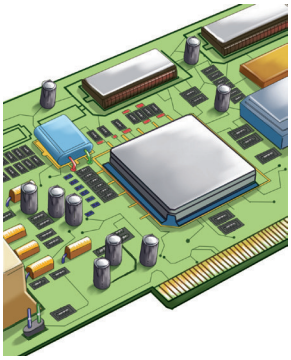
Solution

The register contains 0110 after four clock pulses. See Figure 8–7(b).

Related Problem

If the data input remains 0 after the fourth clock pulse, what is the state of the register after three additional clock pulses?

IMPLEMENTATION: 8-BIT SERIAL IN/PARALLEL OUT SHIFT REGISTER



Fixed-Function Device The 74HC164 is an example of a fixed-function IC shift register having serial in/parallel out operation. The logic block symbol is shown in Figure 8–8. This device has two gated serial inputs, *A* and *B*, and an asynchronous clear ($\overline{CLR}$) input that is active-LOW. The parallel outputs are Q_0 through Q_7.

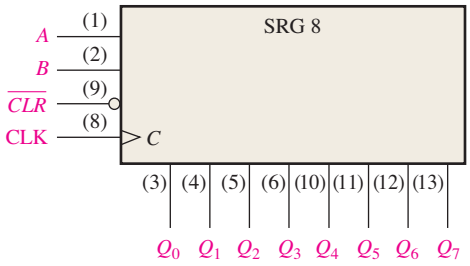


FIGURE 8–8 The 74HC164 8-bit serial in/parallel out shift register.

A sample timing diagram for the 74HC164 is shown in Figure 8–9. Notice that the serial input data on input *A* are shifted into and through the register after input *B* goes HIGH.

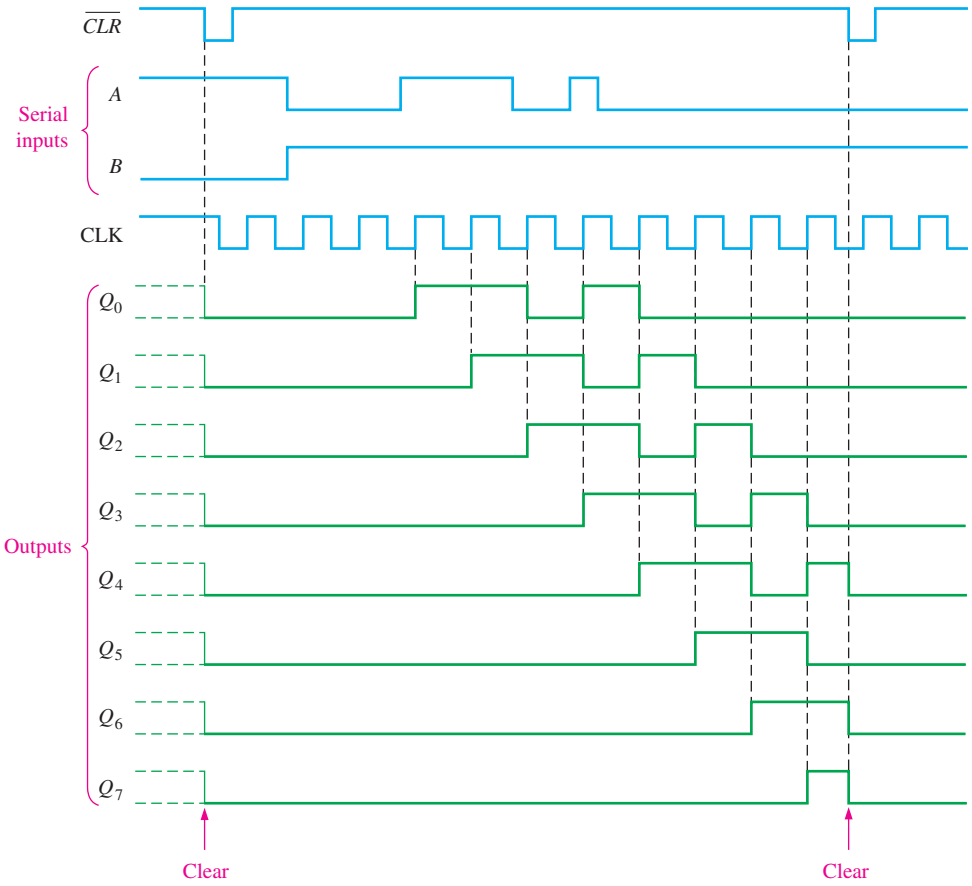


FIGURE 8–9 Sample timing diagram for a 74HC164 shift register.

Programmable Logic Device (PLD) The 8-bit serial in/parallel out shift register can be described using VHDL and implemented as hardware in a PLD. The program code is as follows. (Blue comments are not part of the program.)

```
library ieee;
use ieee.std_logic_1164.all;
entity SerInParOutShift is
   port (D0, Clock, Clr: in std_logic; Q0, Q1, Q2, Q3,
   Q4, Q5, Q6, Q7: inout std_logic);
end entity SerInParOutShift;
architecture LogicOperation of SerInParOutShift is
component dffl is
   port (D, Clock: in std_logic; Q: inout std_logic);
end component dff1;
begin
      FF0: dff1 port map(D=>D0 and Clr, Clock=>Clock, Q=>Q0);
      FF1: dff1 port map(D=>Q0 and Clr, Clock=>Clock, Q=>Q1);
      FF2: dff1 port map(D=>Q1 and Clr, Clock=>Clock, Q=>Q2);
      FF3: dff1 port map(D=>Q2 and Clr, Clock=>Clock, Q=>Q3);
      FF4: dff1 port map(D=>Q3 and Clr, Clock=>Clock, Q=>Q4);
      FF5: dff1 port map(D=>Q4 and Clr, Clock=>Clock, Q=>Q5);
      FF6: dff1 port map(D=>Q5 and Clr, Clock=>Clock, Q=>Q6);
      FF7: dff1 port map(D=>Q6 and Clr, Clock=>Clock, Q=>Q7);
   end architecture LogicOperation;
```

D0: Data input
Clock: System clock
Clr: Clear
Q0–Q7: Register outputs

D flip-flop with preset and clear inputs was described in Chapter 7 and is used as a component.

Instantiations describe how the flip-flops are connected to form the register.

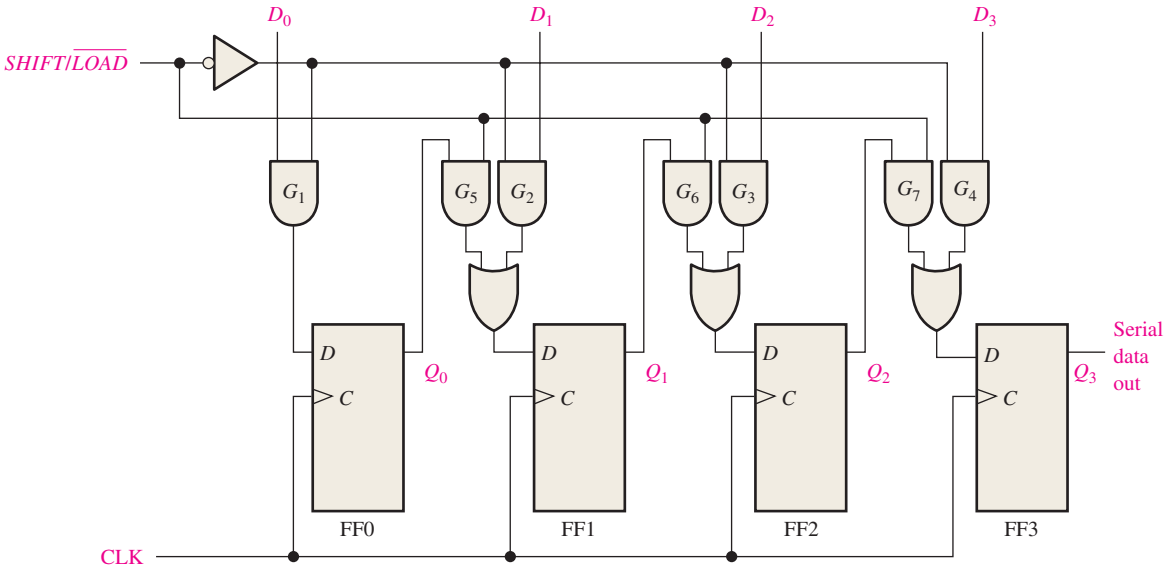
Parallel In/Serial Out Shift Registers

For a register with parallel data inputs, the bits are entered simultaneously into their respective stages on parallel lines rather than on a bit-by-bit basis on one line as with serial data inputs. The serial output is the same as in serial in/serial out shift registers, once the data are completely stored in the register.

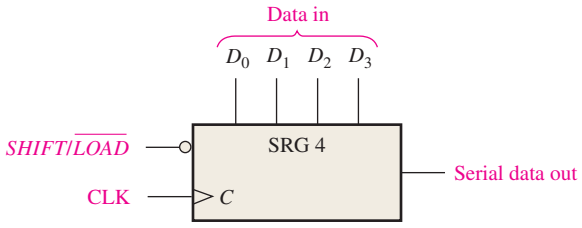
For parallel data, multiple bits are transferred at one time.

Figure 8–10 illustrates a 4-bit parallel in/serial out shift register and a typical logic symbol. There are four data-input lines, D_0, D_1, D_2, and D_3, and a $SHIFT/\overline{LOAD}$ input, which allows four bits of data to **load** in parallel into the register. When $SHIFT/\overline{LOAD}$ is LOW, gates G_1 through G_4 are enabled, allowing each data bit to be applied to the D input of its respective flip-flop. When a clock pulse is applied, the flip-flops with $D = 1$ will set and those with $D = 0$ will reset, thereby storing all four bits simultaneously.

When $SHIFT/\overline{LOAD}$ is HIGH, gates G_1 through G_4 are disabled and gates G_5 through G_7 are enabled, allowing the data bits to shift right from one stage to the next. The OR gates allow either the normal shifting operation or the parallel data-entry operation, depending on which AND gates are enabled by the level on the $SHIFT/\overline{LOAD}$ input. Notice that FF0 has a single AND to disable the parallel input, D_0. It does not require an AND/OR arrangement because there is no serial data in.



(a) Logic diagram



(b) Logic symbol

FIGURE 8–10 A 4-bit parallel in/serial out shift register. Open file F08-10 to verify operation.

MultiSim

EXAMPLE 8–3

Show the data-output waveform for a 4-bit register with the parallel input data and the clock and *SHIFT*/$\overline{LOAD}$ waveforms given in Figure 8–11(a). Refer to Figure 8–10(a) for the logic diagram.

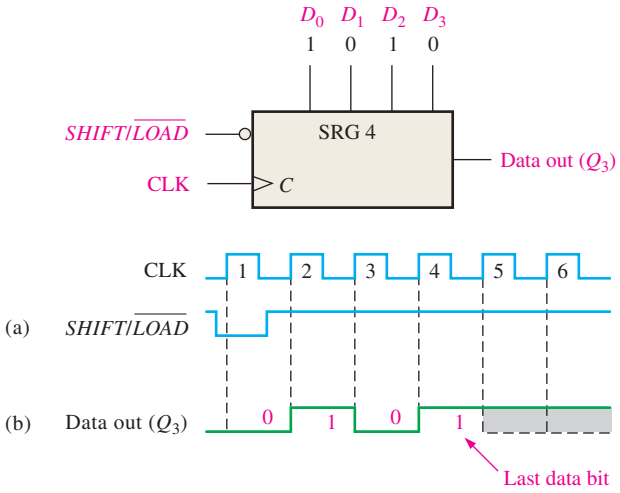


FIGURE 8–11

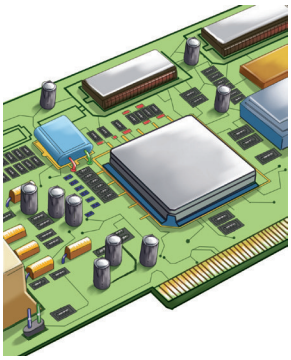
Solution

On clock pulse 1, the parallel data ($D_0D_1D_2D_3 = 1010$) are loaded into the register, making Q_3 a 0. On clock pulse 2 the 1 from Q_2 is shifted onto Q_3; on clock pulse 3 the 0 is shifted onto Q_3; on clock pulse 4 the last data bit (1) is shifted onto Q_3; and on clock pulse 5, all data bits have been shifted out, and only 1s remain in the register (assuming the D_0 input remains a 1). See Figure 8–11(b).

Related Problem

Show the data-output waveform for the clock and *SHIFT*/$\overline{LOAD}$ inputs shown in Figure 8–11(a) if the parallel data are $D_0D_1D_2D_3 = 0101$.

IMPLEMENTATION: 8-BIT PARALLEL LOAD SHIFT REGISTER



Fixed-Function Device The 74HC165 is an example of a fixed-function IC shift register that has a parallel in/serial out operation (it can also be operated as serial in/serial out). Figure 8–12 shows a typical logic block symbol. A LOW on the *SHIFT*/$\overline{LOAD}$ input (*SH*/$\overline{LD}$) enables asynchronous parallel loading. Data can be entered serially on the *SER* input. Also, the clock can be inhibited anytime with a HIGH on the *CLK INH* input. The serial data outputs of the register are Q_7 and its complement $\overline{Q}_7$. This implementation is different from the synchronous method of parallel loading previously discussed, demonstrating that there are usually several ways to accomplish the same function.

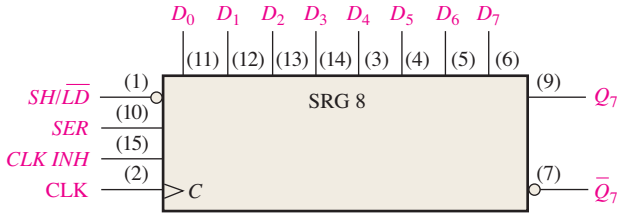


FIGURE 8–12 The 74HC165 8-bit parallel load shift register.

Figure 8–13 is a timing diagram showing an example of the operation of a 74HC165 shift register.

Programmable Logic Device (PLD) The 8-bit parallel load shift register is a parallel in/serial out device and can be implemented in a PLD with the following VHDL code:

```
library ieee:
use ieee.std_logic_1164.all;

entity ParSerShift is
    port (D0, D1, D2, D3, D4, D5, D6, D7, SHLD, Clock:
          in std_logic; Q, QNot: inout std_logic);
end entity ParSerShift;

architecture LogicOperation of ParSerShift is
    signal S1, S2, S3, S4, S5, S6, S7,
           Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7: std_logic;
function ShiftLoad (A,B,C: in std_logic)return std_logic is
begin
    return ((A and B) or (not B and C));
end function ShiftLoad;
```

D0-D7: Parallel input
SHLD: Shift Load input
Clock: System clock
Q: Serial output
QNot: Inverted serial output

S1-S7: Shift load signals from function ShiftLoad
Q0-Q7: Intermediate variables for flip-flop stages

Function ShiftLoad provides the AND-OR function shown in Figure 8–10 to allow the parallel load of data or data shift from one flip-flop stage to the next.

```
component dff1 is
port (D, Clock: in std_logic;
      Q: inout std_logic);
end component dff1;
```

D flip-flop component used as storage for shift register

```
begin
   SL1:S1 <=ShiftLoad(Q0, SHLD, D1);
   SL2:S2 <=ShiftLoad(Q1, SHLD, D2);
   SL3:S3 <=ShiftLoad(Q2, SHLD, D3);
   SL4:S4 <=ShiftLoad(Q3, SHLD, D4);
   SL5:S5 <=ShiftLoad(Q4, SHLD, D5);
   SL6:S6 <=ShiftLoad(Q5, SHLD, D6);
   SL7:S7 <=ShiftLoad(Q6, SHLD, D7);
   FF0: dff1 port map(D=>D0 and not SHLD, Clock=>Clock, Q=>Q0);
   FF1: dff1 port map(D=>S1, Clock=>Clock, Q=>Q1);
   FF2: dff1 port map(D=>S2, Clock=>Clock, Q=>Q2);
   FF3: dff1 port map(D=>S3, Clock=>Clock, Q=>Q3);
   FF4: dff1 port map(D=>S4, Clock=>Clock, Q=>Q4);
   FF5: dff1 port map(D=>S5, Clock=>Clock, Q=>Q5);
   FF6: dff1 port map(D=>S6, Clock=>Clock, Q=>Q6);
   FF7: dff1 port map(D=>S7, Clock=>Clock, Q=>Q);
   QNot <=not Q;
end architecture LogicOperation;
```

ShiftLoad instances SL1–SL7 allow eight bits of data to load into flip-flop stages FF0–FF7 or to shift through the register providing the parallel load serial out function.

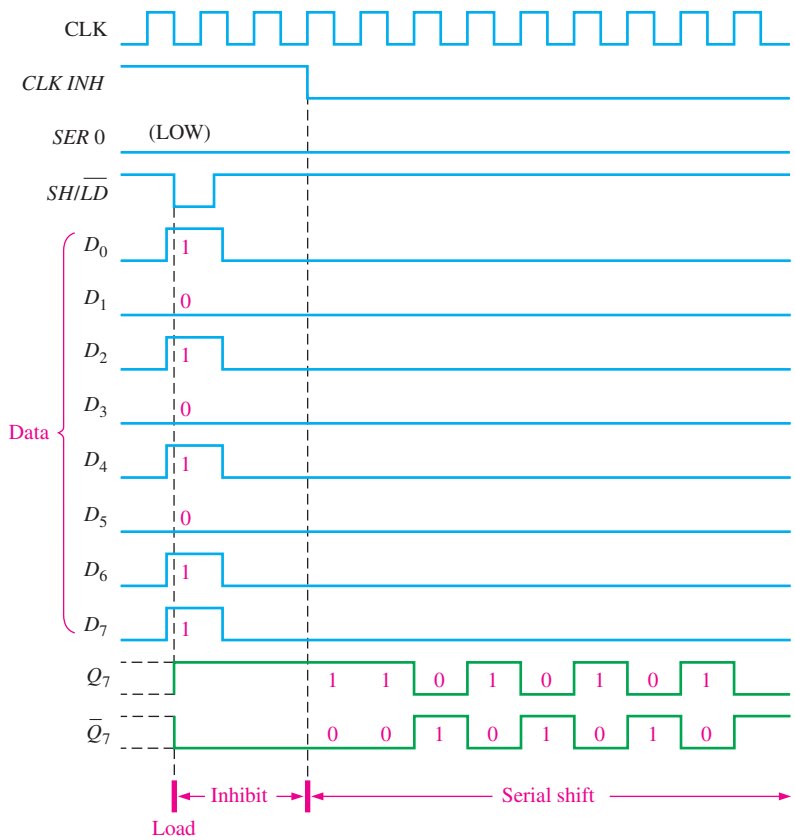


FIGURE 8–13 Sample timing diagram for a 74HC165 shift register.

Parallel In/Parallel Out Shift Registers

Parallel entry and parallel output of data have been discussed. The parallel in/parallel out register employs both methods. Immediately following the simultaneous entry of all data bits, the bits appear on the parallel outputs. Figure 8–14 shows a parallel in/parallel out shift register.

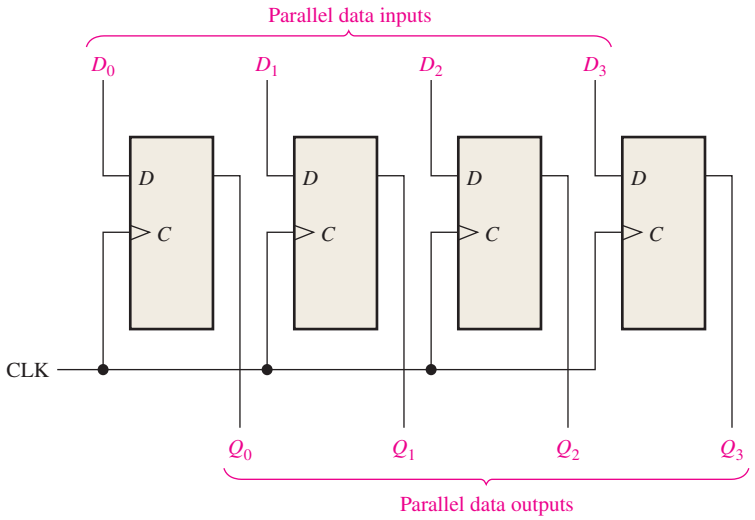
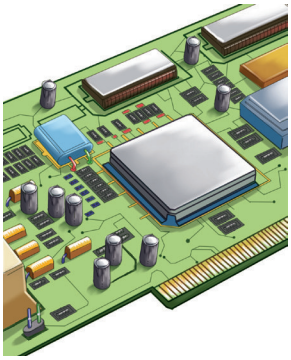


FIGURE 8–14 A parallel in/parallel out register.

IMPLEMENTATION: 4-BIT PARALLEL-ACCESS SHIFT REGISTER



Fixed-Function Device The 74HC195 can be used for parallel in/parallel out operation. Because it also has a serial input, it can be used for serial in/serial out and serial in/parallel out operations. It can be used for parallel in/serial out operation by using Q_3 as the output. A typical logic block symbol is shown in Figure 8–15.

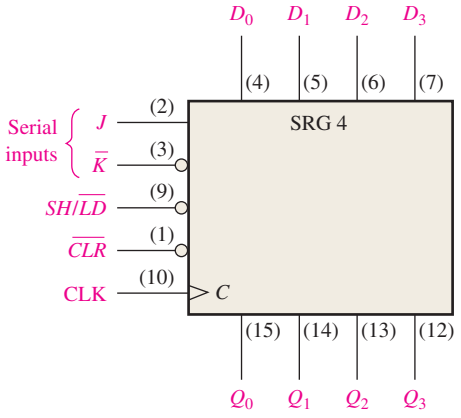


FIGURE 8–15 The 74HC195 4-bit parallel access shift register.

When the $SHIFT/\overline{LOAD}$ input ($SH/\overline{LD}$) is LOW, the data on the parallel inputs are entered synchronously on the positive transition of the clock. When ($SH/\overline{LD}$) is HIGH, stored data will shift right (Q_0 to Q_3) synchronously with the clock. Inputs J and $\overline{K}$ are the serial data inputs to the first stage of the register (Q_0); Q_3 can be used for serial output data. The active-LOW clear input is asynchronous.

The timing diagram in Figure 8–16 illustrates the operation of this register.

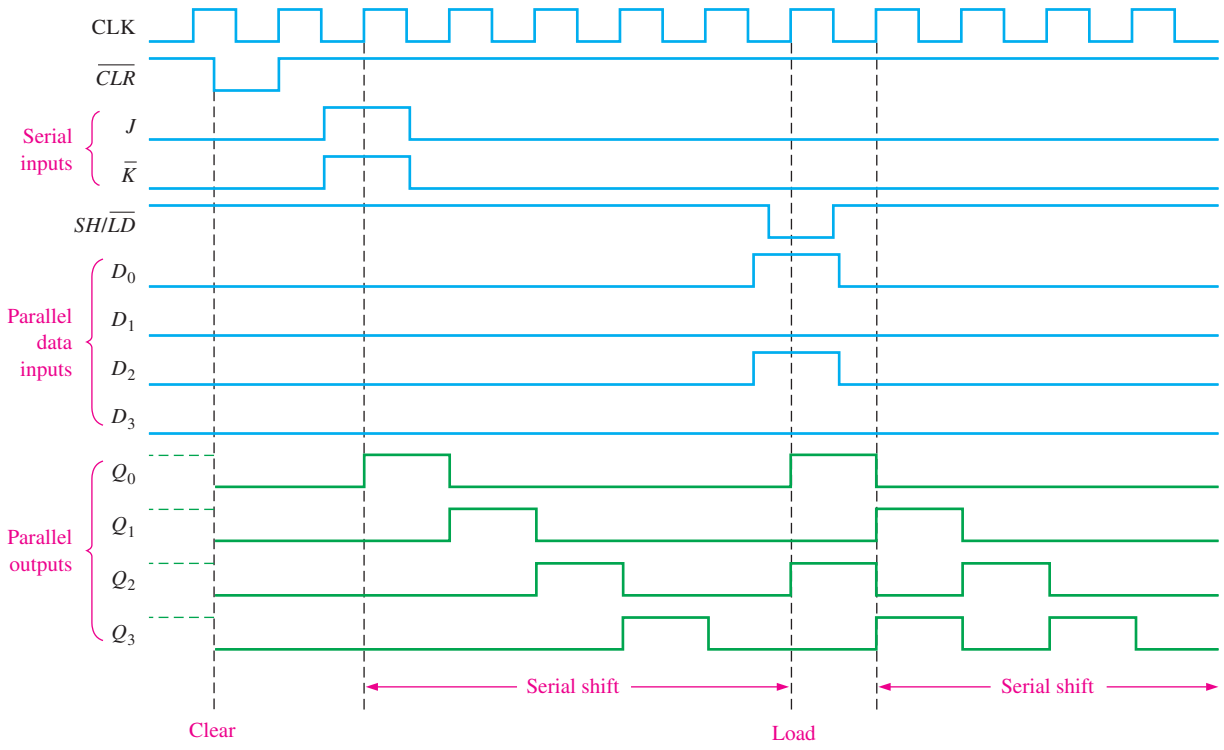


FIGURE 8–16 Sample timing diagram for a 74HC195 shift register.

Programmable Logic Device (PLD) The VHDL code for a 4-bit parallel in/parallel out shift register is as follows:

```
library ieee;
use ieee.std logic_1164.all;

entity ParInParOut is
   port (D0, D1, D2, D3, Clock: in std_logic;
         Q0, Q1, Q2, Q3: inout std_logic);
end entity ParInParOut;

architecture LogicOperation of ParInParOut is
   component dff1 is
      port (D, Clock: in std_logic;
            Q: inout std_logic);
end component dff1;

begin
   FF0: dff1 port map (D=>D0, Clock=>Clock, Q=>Q0);
   FF1: dff1 port map (D=>D1, Clock=>Clock, Q=>Q1);
   FF2: dff1 port map (D=>D2, Clock=>Clock, Q=>Q2);
   FF3: dff1 port map (D=>D3, Clock=>Clock, Q=>Q3);
end architecture LogicOperation;
```

SECTION 8–2 CHECKUP

1. Develop the logic diagram for the shift register in Figure 8–3, using J-K flip-flops to replace the D flip-flops.
2. How many clock pulses are required to enter a byte of data serially into an 8-bit shift register?
3. The bit sequence 1101 is serially entered (least-significant bit first) into a 4-bit parallel out shift register that is initially clear. What are the Q outputs after two clock pulses?
4. How can a serial in/parallel out register be used as a serial in/serial out register?
5. Explain the function of the $SHIFT/\overline{LOAD}$ input.
6. Is the parallel load operation in a 74HC165 shift register synchronous or asynchronous? What does this mean?
7. In Figure 8–14, $D_0 = 1, D_1 = 0, D_2 = 0$, and $D_3 = 1$. After three clock pulses, what are the data outputs?
8. For a 74HC195, $SH/\overline{LD} = 1, J = 1$, and $\overline{K} = 1$. What is Q_0 after one clock pulse?

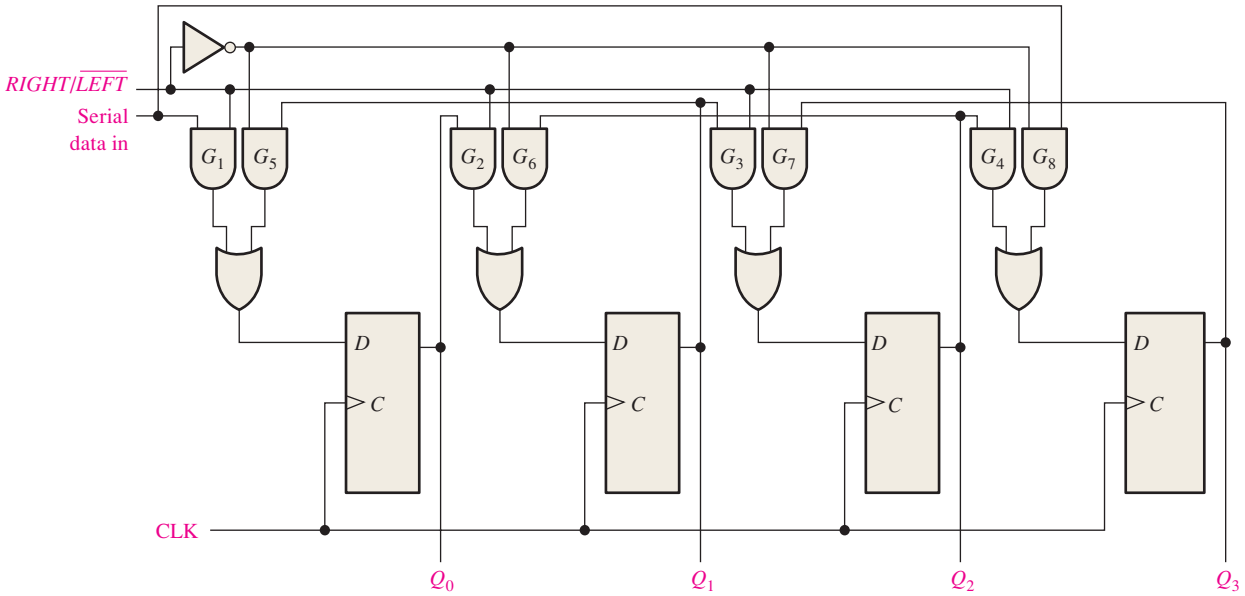
8–3 Bidirectional Shift Registers

A **bidirectional** shift register is one in which the data can be shifted either left or right. It can be implemented by using gating logic that enables the transfer of a data bit from one stage to the next stage to the right or to the left, depending on the level of a control line.

After completing this section, you should be able to

- Explain the operation of a bidirectional shift register
- Discuss the 74HC194 4-bit bidirectional universal shift register
- Develop and analyze timing diagrams for bidirectional shift registers

A 4-bit bidirectional shift register is shown in Figure 8–17. A HIGH on the $RIGHT/\overline{LEFT}$ control input allows data bits inside the register to be shifted to the right, and a LOW



MultiSim **FIGURE 8–17** Four-bit bidirectional shift register. Open file F08-17 to verify the operation.

enables data bits inside the register to be shifted to the left. An examination of the gating logic will make the operation apparent. When the *RIGHT*/$\overline{LEFT}$ control input is HIGH, gates G_1 through G_4 are enabled, and the state of the Q output of each flip-flop is passed through to the D input of the *following* flip-flop. When a clock pulse occurs, the data bits are shifted one place to the *right.* When the *RIGHT*/$\overline{LEFT}$ control input is LOW, gates G_5 through G_8 are enabled, and the Q output of each flip-flop is passed through to the D input of the *preceding* flip-flop. When a clock pulse occurs, the data bits are then shifted one place to the *left.*

EXAMPLE 8–4

Determine the state of the shift register of Figure 8–17 after each clock pulse for the given *RIGHT*/$\overline{LEFT}$ control input waveform in Figure 8–18(a). Assume that $Q_0 = 1$, $Q_1 = 1$, $Q_2 = 0$, and $Q_3 = 1$ and that the serial data-input line is LOW.

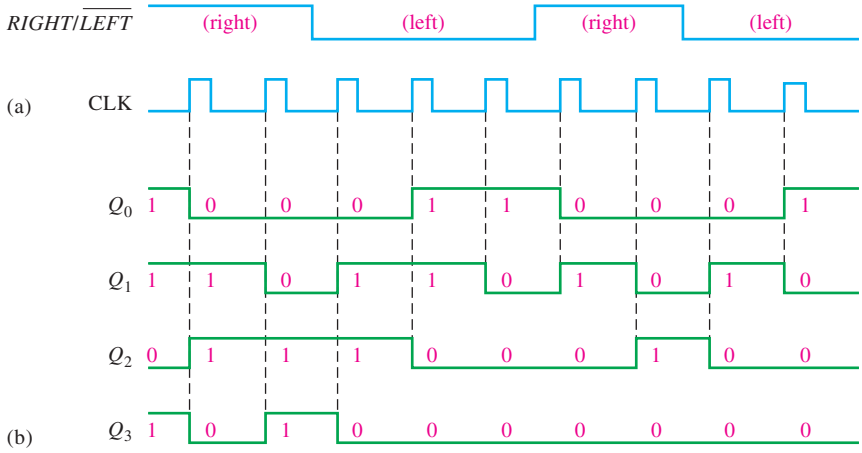


FIGURE 8–18

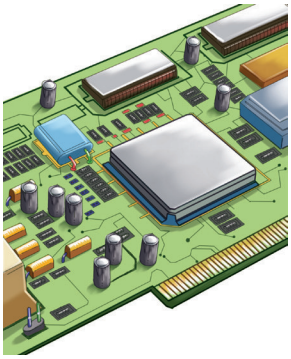
Solution

See Figure 8–18(b).

Related Problem

Invert the *RIGHT*/$\overline{LEFT}$ waveform, and determine the state of the shift register in Figure 8–17 after each clock pulse.

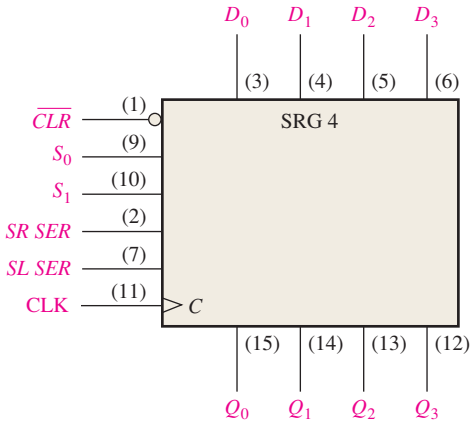
IMPLEMENTATION: 4-BIT BIDIRECTIONAL UNIVERSAL SHIFT REGISTER



Fixed-Function Device The 74HC194 is an example of a universal bidirectional shift register in integrated circuit form. A **universal shift register** has both serial and parallel input and output capability. A logic block symbol is shown in Figure 8–19, and a sample timing diagram is shown in Figure 8–20.

Parallel loading, which is synchronous with a positive transition of the clock, is accomplished by applying the four bits of data to the parallel inputs and a HIGH to the S_0 and S_1 inputs. Shift right is accomplished synchronously with the positive edge of the clock when S_0 is HIGH and S_1 is LOW. Serial data in this mode are entered at the shift-right serial input (*SR SER*). When S_0 is LOW and S_1 is HIGH, data bits shift left synchronously with the clock, and new data are entered at the shift-left serial input (*SL SER*). Input *SR SER* goes into the Q_0 stage, and *SL SER* goes into the Q_3 stage.

FIGURE 8–19 The 74HC194 4-bit bidirectional universal shift register.



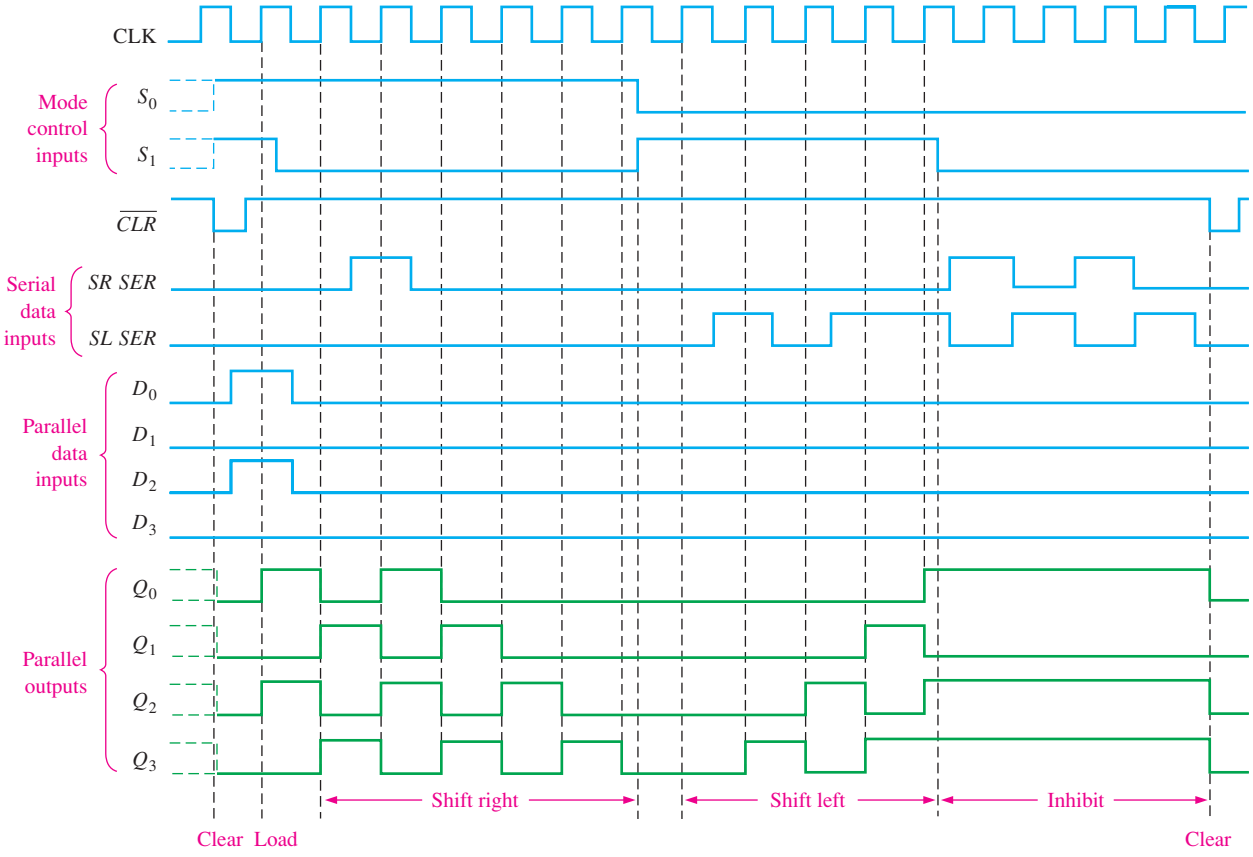


FIGURE 8–20 Sample timing diagram for a 74HC194 shift register.

Programmable Logic Device (PLD) The following code describes a 4-bit bidirectional shift register with a serial input:

```
library ieee;
use ieee.std_logic_1164.all;
entity FourBitBiDirSftReg is
port (R_L, DataIn, Clock: in std_logic;
   Q0, Q1, Q2, Q3: buffer std_logic);
end entity FourBitBiDirSftReg;
```

R_L: Right/left
DataIn: Serial input data
Clock: System clock
Q0-Q3: Register outputs

```
architecture LogicOperation of FourBitBiDirSftReg is
component dff1 is
   port(D,Clock: in std_logic; Q: out std_logic);   } D flip-flop component declaration
end component dff1;
signal D0, D1, D2, D3: std_logic;   Internal flip-flop inputs
begin
   DO <= (DataIn and R_L) or (not R_L and Q1);
   D1 <= (Q0 and R_L) or (not R_L and Q2);          } Describes the internal signals
   D2 <= (Q1 and R_L) or (not R_L and Q3);            with Boolean equations
   D3 <= (Q2 and R_L) or (not R_L and DataIn);
   FF0: dff1 port map(D => D0, Clock => Clock, Q => Q0);
   FF1: dff1 port map(D => D1, Clock => Clock, Q => Q1);   } Describes how the
   FF2: dff1 port map(D => D2, Clock => Clock, Q => Q2);     flip-flops are connected
   FF3: dff1 port map(D => D3, Clock => Clock, Q => Q3);
end architecture LogicOperation;
```

SECTION 8–3 CHECKUP

1. Assume that the 4-bit bidirectional shift register in Figure 8–17 has the following contents: $Q_0 = 1, Q_1 = 1, Q_2 = 0$, and $Q_3 = 0$. There is a 1 on the serial data-input line. If $RIGHT/\overline{LEFT}$ is HIGH for three clock pulses and LOW for two more clock pulses, what are the contents after the fifth clock pulse?

8–4 Shift Register Counters

A shift register counter is basically a shift register with the serial output connected back to the serial input to produce special sequences. These devices are often classified as counters because they exhibit a specified sequence of states. Two of the most common types of shift register counters, the Johnson counter and the ring counter, are introduced in this section.

After completing this section, you should be able to

- Discuss how a shift register counter differs from a basic shift register
- Explain the operation of a Johnson counter
- Specify a Johnson sequence for any number of bits
- Explain the operation of a ring counter and determine the sequence of any specific ring counter

The Johnson Counter

In a **Johnson counter** the complement of the output of the last flip-flop is connected back to the *D* input of the first flip-flop (it can be implemented with other types of flip-flops as well). If the counter starts at 0, this feedback arrangement produces a characteristic sequence of states, as shown in Table 8–3 for a 4-bit device and in Table 8–4 for a 5-bit device. Notice that the 4-bit sequence has a total of eight states, or bit patterns, and that the 5-bit sequence has a total of ten states. In general, a Johnson counter will produce a modulus of $2n$, where n is the number of stages in the counter.

TABLE 8–3

Four-bit Johnson sequence.

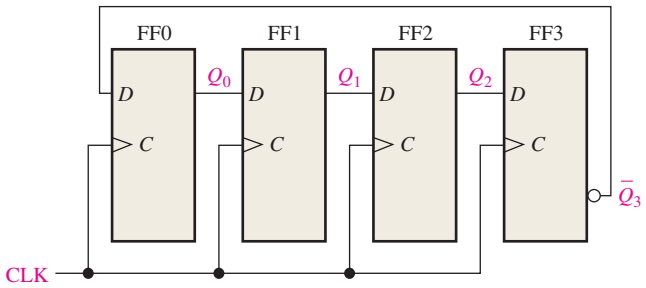
Clock Pulse	Q_0	Q_1	Q_2	Q_3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

TABLE 8–4

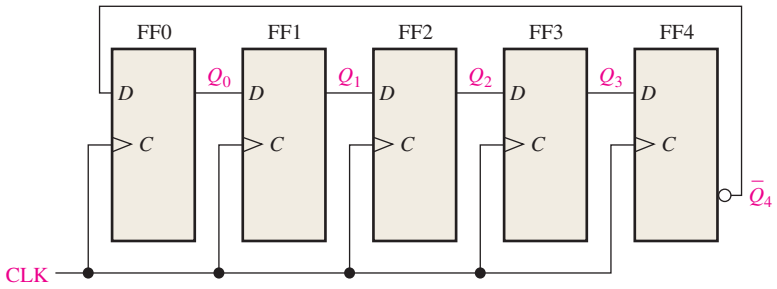
Five-bit Johnson sequence.

Clock Pulse	Q_0	Q_1	Q_2	Q_3	Q_4
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	1	1	1	1	0
5	1	1	1	1	1
6	0	1	1	1	1
7	0	0	1	1	1
8	0	0	0	1	1
9	0	0	0	0	1

The implementations of the 4-stage and 5-stage Johnson counters are shown in Figure 8–21. The implementation of a Johnson counter is very straightforward and is the same regardless of the number of stages. The Q output of each stage is connected to the D input of the next



(a) Four-bit Johnson counter



(b) Five-bit Johnson counter

FIGURE 8–21 Four-bit and 5-bit Johnson counters.

stage (assuming that *D* flip-flops are used). The single exception is that the $\overline{Q}$ output of the last stage is connected back to the *D* input of the first stage. As the sequences in Table 8–3 and 8–4 show, if the counter starts at 0, it will "fill up" with 1s from left to right, and then it will "fill up" with 0s again.

Diagrams of the timing operations of the 4-bit and 5-bit counters are shown in Figures 8–22 and 8–23, respectively.

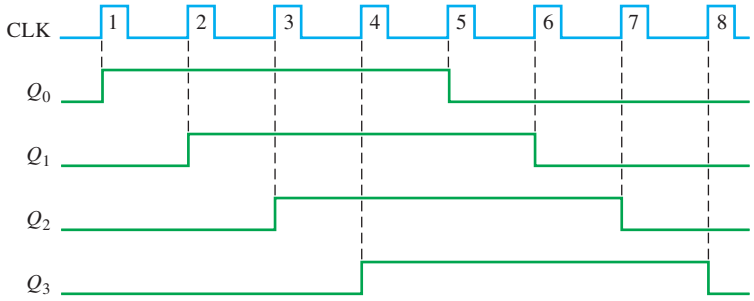


FIGURE 8–22 Timing sequence for a 4-bit Johnson counter.

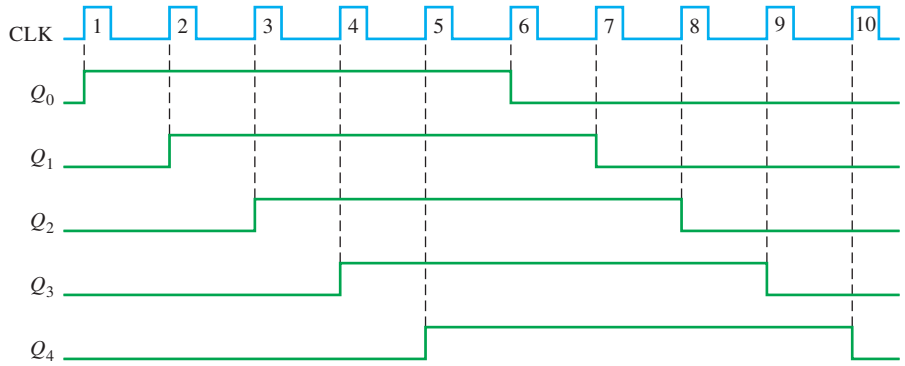


FIGURE 8–23 Timing sequence for a 5-bit Johnson counter.

The Ring Counter

A **ring counter** utilizes one flip-flop for each state in its sequence. It has the advantage that decoding gates are not required. In the case of a 10-bit ring counter, there is a unique output for each decimal digit.

A logic diagram for a 10-bit ring counter is shown in Figure 8–24. The sequence for this ring counter is given in Table 8–5. Initially, a 1 is preset into the first flip-flop, and the rest of the flip-flops are cleared. Notice that the interstage connections are the same as those for a

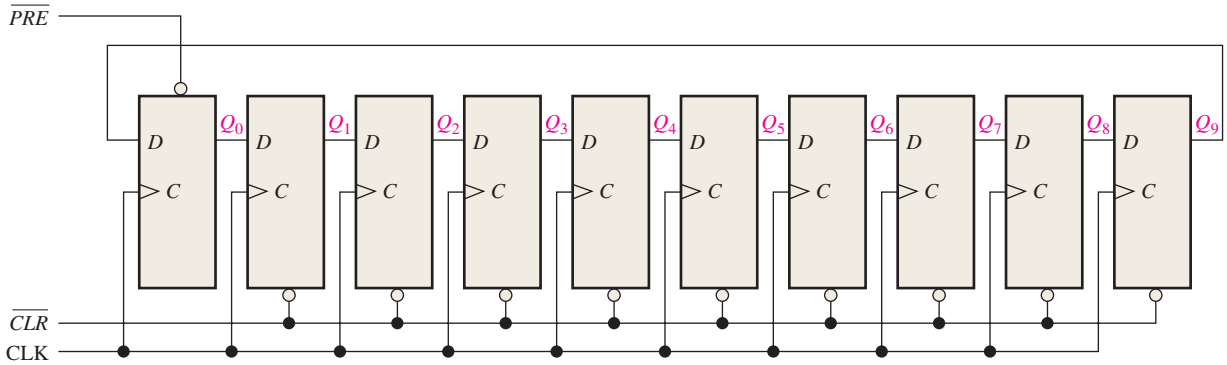


FIGURE 8–24 A 10-bit ring counter. Open file F08-24 to verify operation.

MultiSim

TABLE 8–5

Ten-bit ring counter sequence.

Clock Pulse	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Q_7	Q_8	Q_9
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	0	1	0	0
8	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1

Johnson counter, except that Q rather than $\overline{Q}$ is fed back from the last stage. The ten outputs of the counter indicate directly the decimal count of the clock pulse. For instance, a 1 on Q_0 represents a zero, a 1 on Q_1 represents a one, a 1 on Q_2 represents a two, a 1 on Q_3 represents a three, and so on. You should verify for yourself that the 1 is always retained in the counter and simply shifted "around the ring," advancing one stage for each clock pulse.

Modified sequences can be achieved by having more than a single 1 in the counter, as illustrated in Example 8–5.

EXAMPLE 8–5

If a 10-bit ring counter similar to Figure 8–24 has the initial state 1010000000, determine the waveform for each of the Q outputs.

Solution

See Figure 8–25.

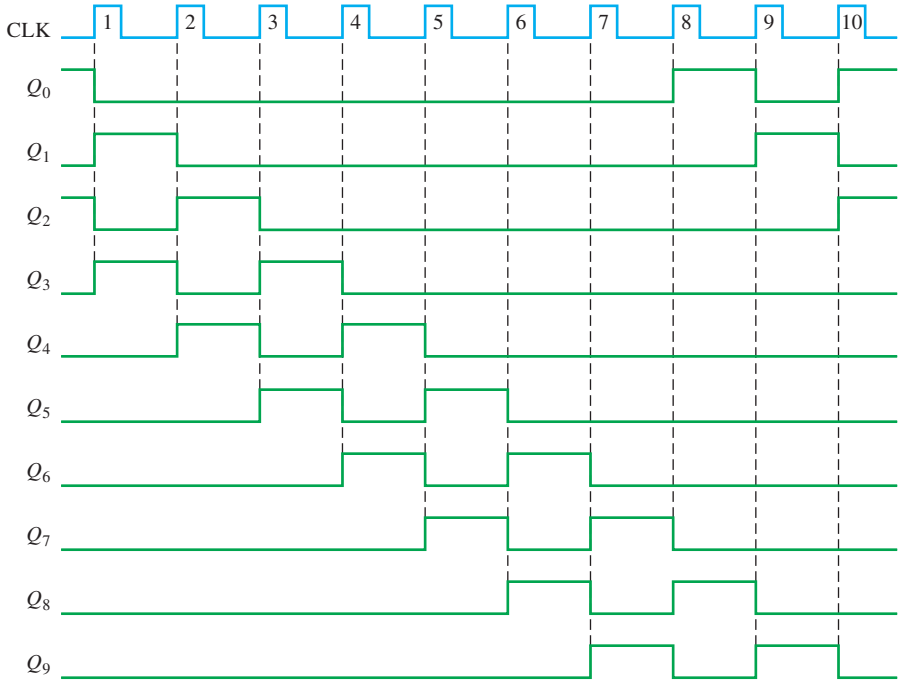


FIGURE 8–25

Related Problem

If a 10-bit ring counter has an initial state 0101001111, determine the waveform for each Q output.

SECTION 8–4 CHECKUP

1. How many states are there in an 8-bit Johnson counter sequence?
2. Write the sequence of states for a 3-bit Johnson counter starting with 000.

8–5 Shift Register Applications

Shift registers are found in many types of applications, a few of which are presented in this section.

After completing this section, you should be able to

- Use a shift register to generate a time delay
- Implement a specified ring counter sequence using a 74HC195 shift register
- Discuss how shift registers are used for serial-to-parallel conversion of data
- Define *UART*
- Explain the operation of a keyboard encoder and how registers are used in this application

Time Delay

A serial in/serial out shift register can be used to provide a time delay from input to output that is a function of both the number of stages (n) in the register and the clock frequency.

When a data pulse is applied to the serial input as shown in Figure 8–26, it enters the first stage on the triggering edge of the clock pulse. It is then shifted from stage to stage on each successive clock pulse until it appears on the serial output n clock periods later. This time-delay operation is illustrated in Figure 8–26, in which an 8-bit serial in/serial out shift register is used with a clock frequency of 1 MHz to achieve a time delay (t_d) of 8 μs (8 $\times$ 1 μs). This time can be adjusted up or down by changing the clock frequency. The time delay can also be increased by cascading shift registers and decreased by taking the outputs from successively lower stages in the register if the outputs are available, as illustrated in Example 8–6.

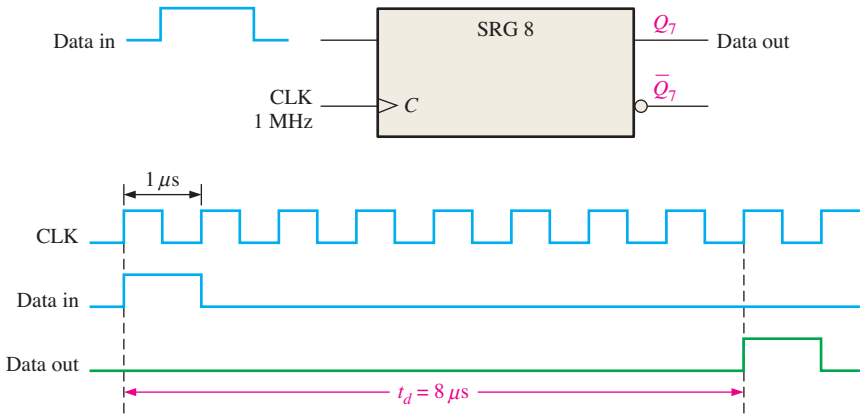


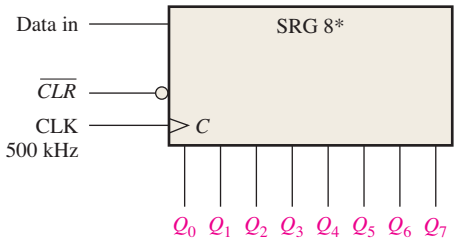
FIGURE 8–26 The shift register as a time-delay device.

InfoNote

Microprocessors have special instructions that can emulate a serial shift register. The accumulator register can shift data to the left or right. A right shift is equivalent to a divide-by-2 operation and a left shift is equivalent to a multiply-by-2 operation. Data in the accumulator can be shifted left or right with the rotate instructions; ROR is the rotate right instruction, and ROL is the rotate left instruction. Two other instructions treat the carry flag bit as an additional bit for the rotate operation. These are the RCR for rotate carry right and RCL for rotate carry left.

EXAMPLE 8–6

Determine the amount of time delay between the serial input and each output in Figure 8–27. Show a timing diagram to illustrate.



* Data shifts from Q_0 toward Q_7.

FIGURE 8–27

Solution

The clock period is 2 μs. Thus, the time delay can be increased or decreased in 2 μs increments from a minimum of 2 μs to a maximum of 16 μs, as illustrated in Figure 8–28.

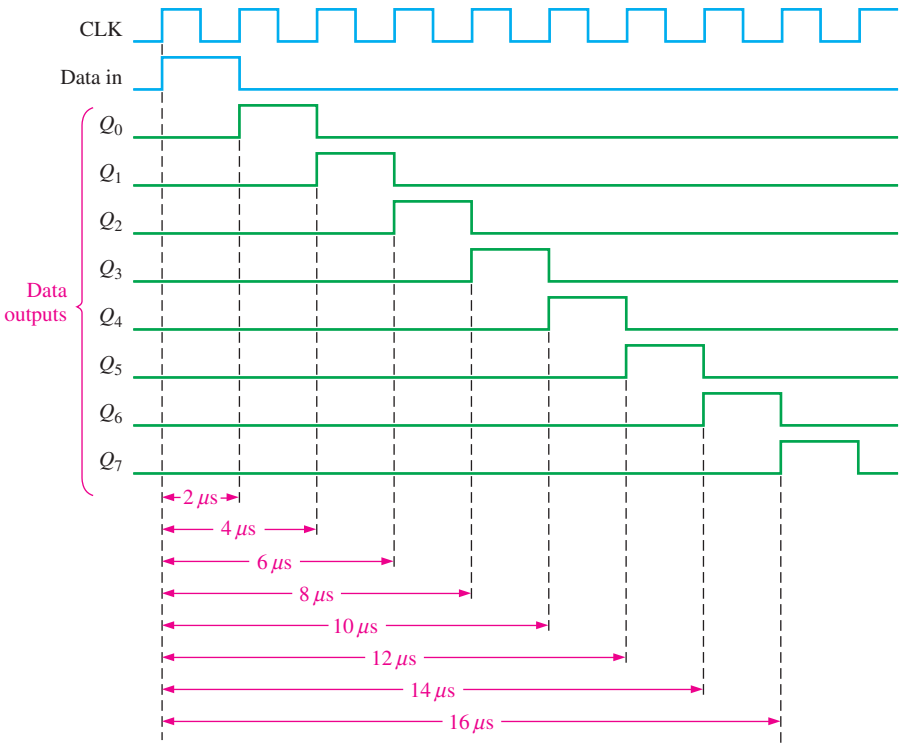
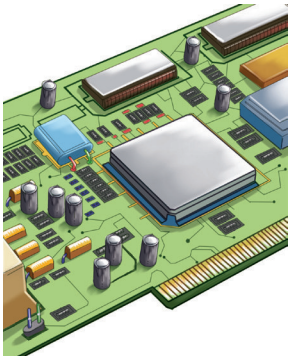


FIGURE 8–28 Timing diagram showing time delays for the register in Figure 8–27.

Related Problem

Determine the clock frequency required to obtain a time delay of 24 μs to the Q_7 output in Figure 8–27.

IMPLEMENTATION: A RING COUNTER



Fixed-Function Device If the output is connected back to the serial input, a shift register can be used as a ring counter. Figure 8–29 illustrates this application with a 74HC195 4-bit shift register.

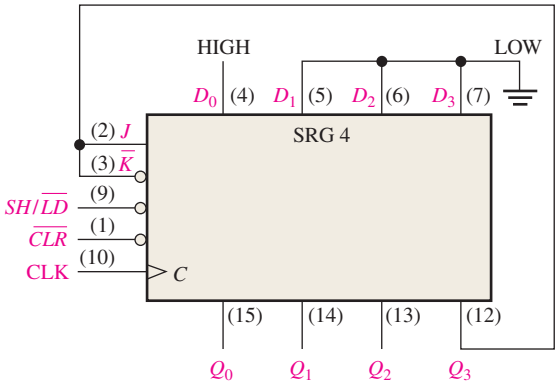


FIGURE 8–29 74HC195 connected as a ring counter.

Initially, a bit pattern of 1000 (or any other pattern) can be synchronously preset into the counter by applying the bit pattern to the parallel data inputs, taking the $SH/\overline{LD}$ input LOW, and applying a clock pulse. After this initialization, the 1 continues to circulate through the ring counter, as the timing diagram in Figure 8–30 shows.

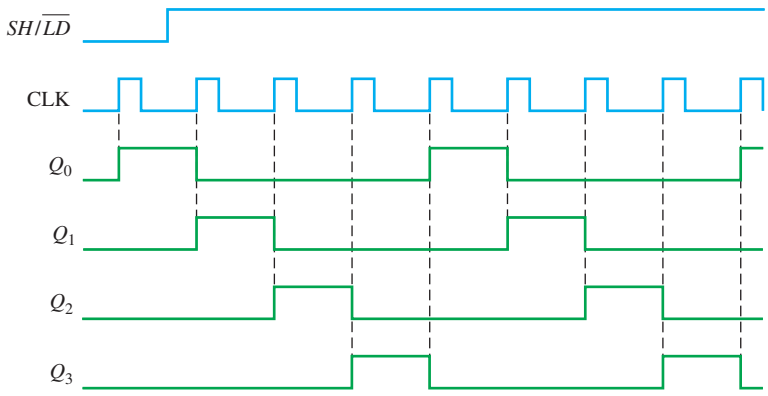


FIGURE 8–30 Timing diagram showing two complete cycles of the ring counter in Figure 8–29 when it is initially preset to 1000.

Programmable Logic Device (PLD) The VHDL code for a 4-bit ring counter using D flip-flops is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity RingCtr is
   port (I, Clr, Clock: in std_logic;
         Q0, Q1, Q2, Q3: inout std_logic);
end entity RingCtr;

architecture LogicOperation of RingCtr is
```

I: Serial input bit to clock data into the shift register
Clr: Ring counter clear input
Clock: System clock
Q0-Q3: Ring counter output stages

```
component dff1 is
   port (D, Clock, Pre, Clr: in std_logic;
      Q: inout std_logic);
end component dff1;
begin
   FF0: dff1 port map(D=> Q3, Clock=>Clock, Q=>Q0, Pre=> not I, Clr=>'1');
   FF1: dff1 port map(D=> Q0, Clock=>Clock, Q=>Q1, Pre=>'1', Clr=>not Clr);
   FF2: dff1 port map(D=> Q1, Clock=>Clock, Q=>Q2, Pre=>'1', Clr=>not Clr);
   FF3: dff1 port map(D=> Q2, Clock=>Clock, Q=>Q3, Pre=>'1', Clr=>not Clr);
end architecture LogicOperation;
```

D flip-flop component used as storage for shift register

FF0-FF3 flip-flop instantiations show how flip-flops are connected and represent one flip-flop for each state in the ring counter sequence. FF0 Pre input acts as a serial input when I is high. FF1-FF3 Clr input clears flip-flop stages when Clr is low.

Serial-to-Parallel Data Converter

Serial data transmission from one digital system to another is commonly used to reduce the number of wires in the transmission line. For example, eight bits can be sent serially over one wire, but it takes eight wires to send the same data in parallel.

Serial data transmission is widely used by peripherals to pass data back and forth to a computer. For example, USB (universal serial bus) is used to connect keyboards printers, scanners, and more to the computer. All computers process data in parallel form, thus the requirement for serial-to-parallel conversion. A simplified serial-to-parallel data converter, in which two types of shift registers are used, is shown in Figure 8–31.

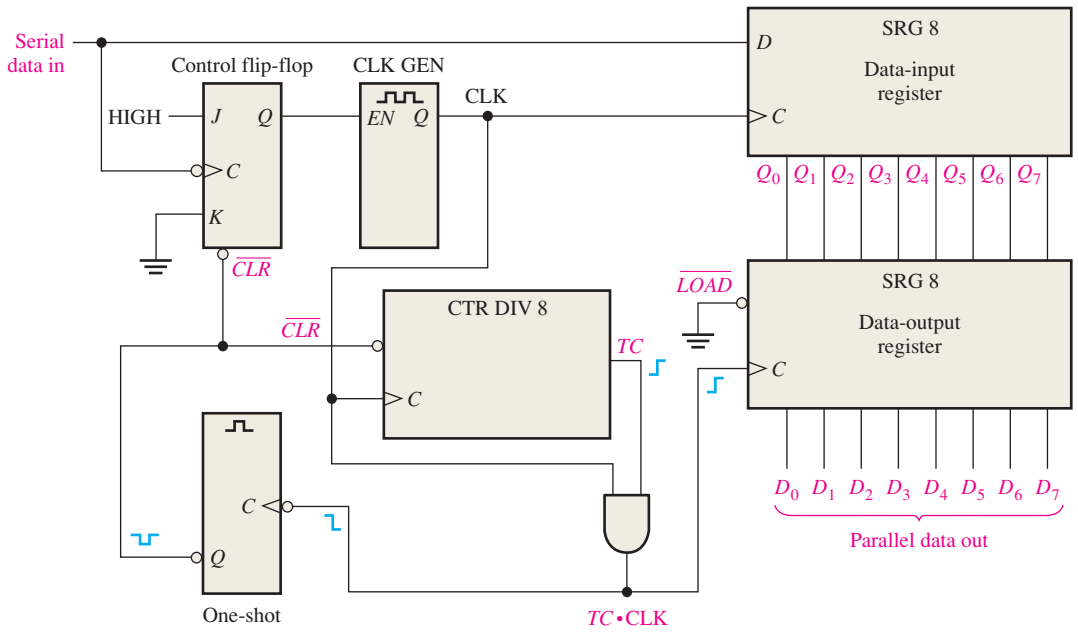


FIGURE 8–31 Simplified logic diagram of a serial-to-parallel converter.

To illustrate the operation of this serial-to-parallel converter, the serial data format shown in Figure 8–32 is used. It consists of eleven bits. The first bit (start bit) is always 0 and always begins with a HIGH-to-LOW transition. The next eight bits (D_7 through D_0) are the data bits (one of the bits can be parity), and the last one or two bits (stop bits) are always 1s. When no data are being sent, there is a continuous HIGH on the serial data line.

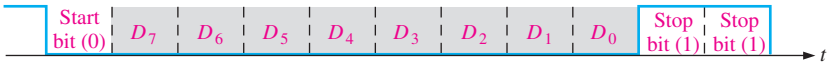


FIGURE 8–32 Serial data format.

The HIGH-to-LOW transition of the start bit sets the control flip-flop, which enables the clock generator. After a fixed delay time, the clock generator begins producing a pulse waveform, which is applied to the data-input register and to the divide-by-8 counter. The clock has a frequency precisely equal to that of the incoming serial data, and the first clock pulse after the start bit occurs during the first data bit.

The timing diagram in Figure 8–33 illustrates the following basic operation: The eight data bits (D_7 through D_0) are serially shifted into the data-input register. Shortly after the

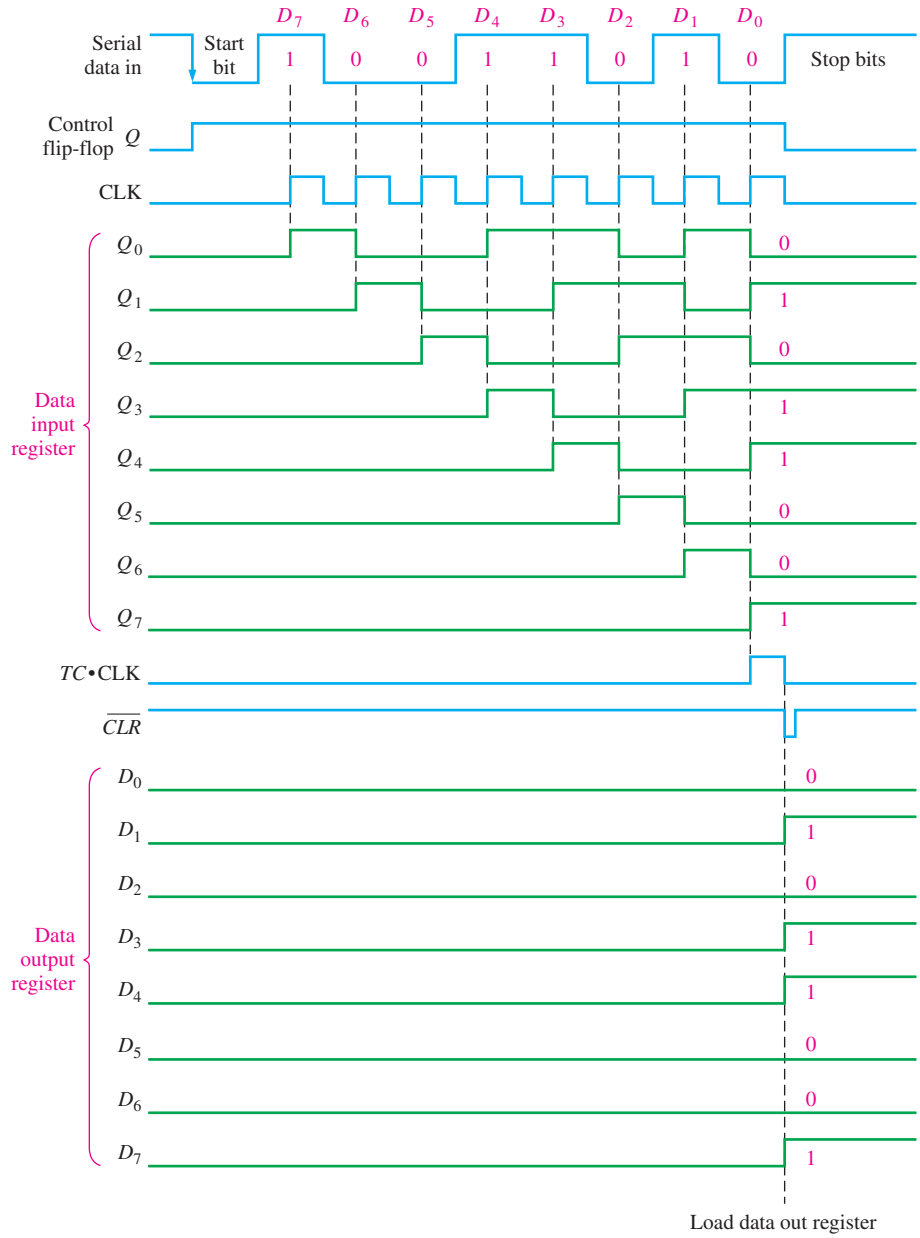


FIGURE 8–33 Timing diagram illustrating the operation of the serial-to-parallel data converter in Figure 8–31.

eighth clock pulse, the terminal count (*TC*) goes from LOW to HIGH, indicating the counter is at the last state. This rising edge is ANDed with the clock pulse, which is still HIGH, producing a rising edge at $TC \cdot \text{CLK}$. This parallel loads the eight data bits from the data-input shift register to the data-output register. A short time later, the clock pulse goes LOW and this HIGH-to-LOW transition triggers the one-shot, which produces a short-duration pulse to clear the counter and reset the control flip-flop and thus disable the clock generator. The system is now ready for the next group of eleven bits, and it waits for the next HIGH-to-LOW transition at the beginning of the start bit.

By reversing the process just stated, parallel-to-serial data conversion can be accomplished. Since the serial data format must be produced, start and stop bits must be added to the sequence.

Universal Asynchronous Receiver Transmitter (UART)

As mentioned, computers and microprocessor-based systems often send and receive data in a parallel format. Frequently, these systems must communicate with external devices that send and/or receive serial data. An interfacing device used to accomplish these conversions is the UART (Universal Asynchronous Receiver Transmitter). Figure 8–34 illustrates the UART in a general microprocessor-based system application.

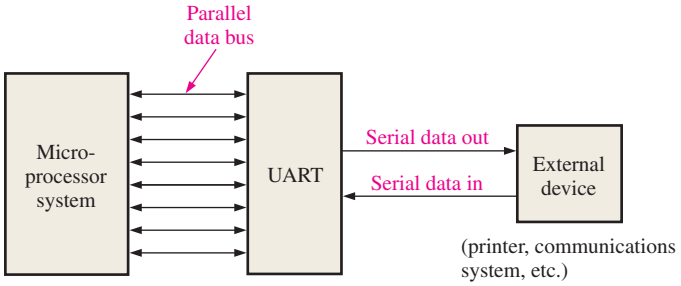


FIGURE 8–34 UART interface.

A UART includes both serial-to-parallel and parallel-to-serial conversion, as shown in the block diagram in Figure 8–35. The data bus is basically a set of parallel conductors along which data move between the UART and the microprocessor system. Buffers interface the data registers with the data bus.

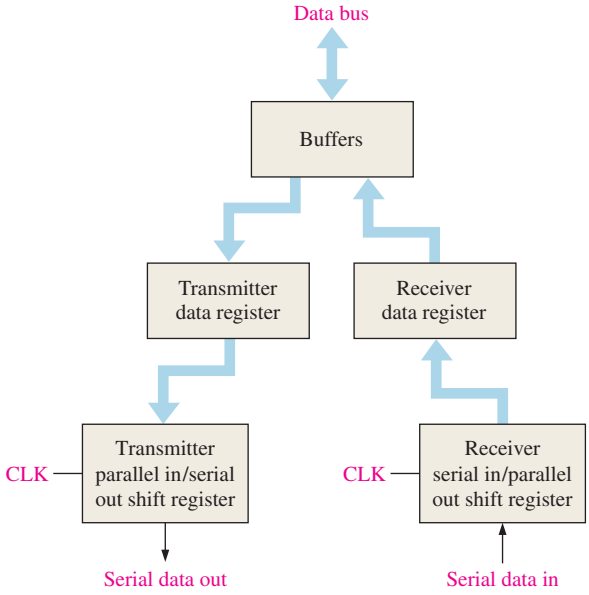


FIGURE 8–35 Basic UART block diagram.

The UART receives data in serial format, converts the data to parallel format, and places them on the data bus. The UART also accepts parallel data from the data bus, converts the data to serial format, and transmits them to an external device.

Keyboard Encoder

The keyboard encoder is a good example of the application of a shift register used as a ring counter in conjunction with other devices. Recall that a simplified computer keyboard encoder without data storage was presented in Chapter 6.

Figure 8–36 shows a simplified keyboard encoder for encoding a key closure in a 64-key matrix organized in eight rows and eight columns. Two parallel in/parallel out 4-bit shift

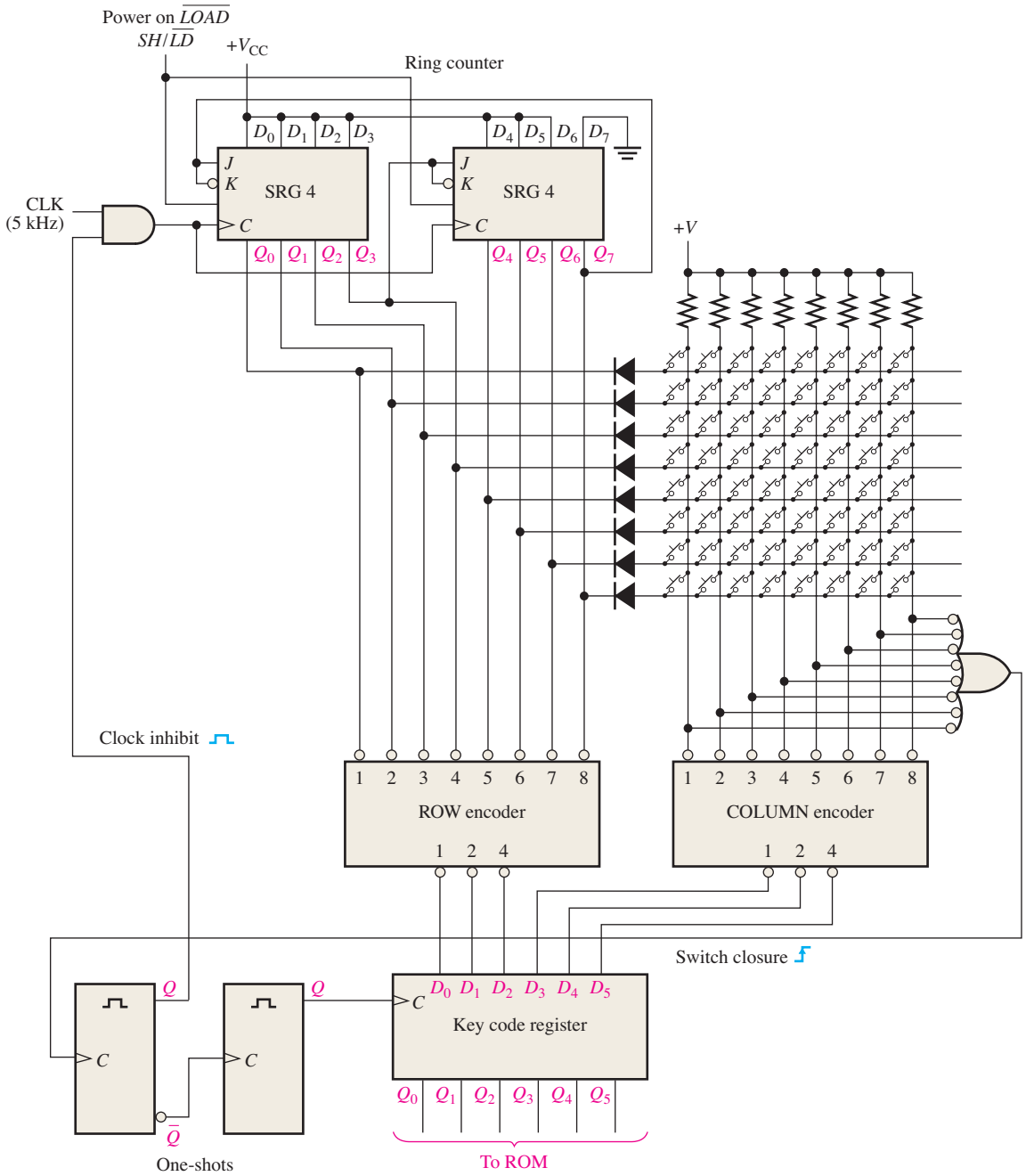


FIGURE 8–36 Simplified keyboard encoding circuit.

registers are connected as an 8-bit ring counter with a fixed bit pattern of seven 1s and one 0 preset into it when the power is turned on. Two priority encoders (introduced in Chapter 6) are used as eight-line-to-three-line encoders (9 input HIGH, 8 output unused) to encode the ROW and COLUMN lines of the keyboard matrix. A parallel in/parallel out register (key code) stores the ROW/COLUMN code from the priority encoders.

The basic operation of the keyboard encoder in Figure 8–36 is as follows: The ring counter "scans" the rows for a key closure as the clock signal shifts the 0 around the counter at a 5 kHz rate. The 0 (LOW) is sequentially applied to each ROW line, while all other ROW lines are HIGH. All the ROW lines are connected to the ROW encoder inputs, so the 3-bit output of the ROW encoder at any time is the binary representation of the ROW line that is LOW. When there is a key closure, one COLUMN line is connected to one ROW line. When the ROW line is taken LOW by the ring counter, that particular COLUMN line is also pulled LOW. The COLUMN encoder produces a binary output corresponding to the COLUMN in which the key is closed. The 3-bit ROW code plus the 3-bit COLUMN code uniquely identifies the key that is closed. This 6-bit code is applied to the inputs of the key code register. When a key is closed, the two one-shots produce a delayed clock pulse to parallel-load the 6-bit code into the key code register. This delay allows the contact bounce to die out. Also, the first one-shot output inhibits the ring counter to prevent it from scanning while the data are being loaded into the key code register.

The 6-bit code in the key code register is now applied to a ROM (read-only memory) to be converted to an appropriate alphanumeric code that identifies the keyboard character. ROMs are studied in Chapter 11.

SECTION 8–5 CHECKUP

1. In the keyboard encoder, how many times per second does the ring counter scan the keyboard?
2. What is the 6-bit ROW/COLUMN code (key code) for the top row and the left-most column in the keyboard encoder?
3. What is the purpose of the diodes in the keyboard encoder? What is the purpose of the resistors?

8–6 Logic Symbols with Dependency Notation

Two examples of ANSI/IEEE Standard 91-1984 symbols with dependency notation for shift registers are presented. Two specific IC shift registers are used as examples.

After completing this section, you should be able to

- Understand and interpret the logic symbols with dependency notation for the 74HC164 and the 74HC194 shift registers

The logic symbol for a 74HC164 8-bit serial in/parallel out shift register is shown in Figure 8–37. The common control inputs are shown on the notched block. The clear ($\overline{CLR}$) input is indicated by an R (for RESET) inside the block. Since there is no dependency prefix to link R with the clock (C1), the clear function is asynchronous. The right arrow symbol after C1 indicates data flow from Q_0 to Q_7. The A and B inputs are ANDed, as indicated by the embedded AND symbol, to provide the synchronous data input, $1D$, to the first stage (Q_0). Note the dependency of D on C, as indicated by the 1 suffix on C and the 1 prefix on D.

Figure 8–38 is the logic symbol for the 74HC194 4-bit bidirectional universal shift register. Starting at the top left side of the control block, note that the $\overline{CLR}$ input is active-LOW and is asynchronous (no prefix link with C). Inputs S_0 and S_1 are mode inputs that

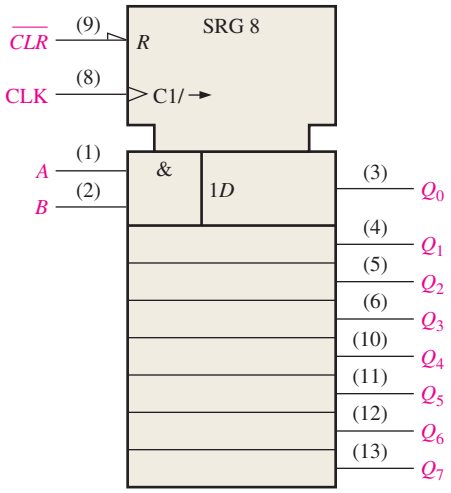


FIGURE 8–37 Logic symbol for the 74HC164.

determine the *shift-right, shift-left,* and *parallel load* modes of operation, as indicated by the $\frac{0}{3}$ dependency designation following the *M*. The $\frac{0}{3}$ represents the binary states of 0, 1, 2, and 3 on the S_0 and S_1 inputs. When one of these digits is used as a prefix for another input, a dependency is established. The $1\rightarrow/2\leftarrow$ symbol on the clock input indicates the following: $1\rightarrow$ indicates that a right shift (Q_0 toward Q_3) occurs when the mode inputs (S_0, S_1) are in the binary 1 state ($S_0 = 1$, $S_1 = 0$), $2\leftarrow$ indicates that a left shift (Q_3 toward Q_0) occurs when the mode inputs are in the binary 2 state ($S_0 = 0$, $S_1 = 1$). The shift-right serial input (*SR SER*) is both mode-dependent and clock-dependent, as indicated by 1, 4*D*. The parallel inputs (D_0, D_1, D_2, and D_3) are all mode-dependent (prefix 3 indicates parallel load mode) and clock-dependent, as indicated by 3, 4*D*. The shift-left serial input (*SL SER*) is both mode-dependent and clock-dependent, as indicated by 2, 4*D*.

The four modes for the 74HC194 are summarized as follows:

Do nothing:	$S_0 = 0, S_1 = 0$	(mode 0)
Shift right:	$S_0 = 1, S_1 = 0$	(mode 1, as in 1, 4*D*)
Shift left:	$S_0 = 0, S_1 = 1$	(mode 2, as in 2, 4*D*)
Parallel load:	$S_0 = 1, S_1 = 1$	(mode 3, as in 3, 4*D*)

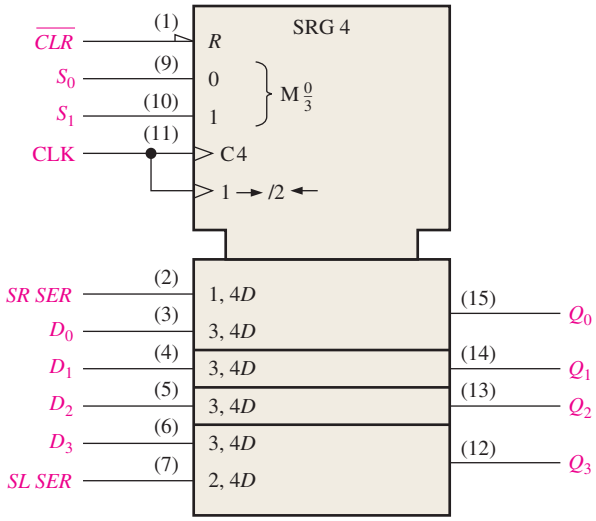


FIGURE 8–38 Logic symbol for the 74HC194.

SECTION 8–6 CHECKUP

1. In Figure 8–38, are there any inputs that are dependent on the mode inputs being in the 0 state?
2. Is the parallel load synchronous with the clock?