Chapter 3

Examples of Solved Problems for Chapter 3, 5, 6, 7, and 8

This document presents some typical problems that the student may encounter, and shows how such problems can be solved. Note that the numbering of examples below is taken from the 2nd edition of the book *Fundamentals of Digital Logic with VHDL Design.* Since not all of these examples are relevant to ECE241, the numbering of examples, and some figure numbers, are not always sequential in this document.

Example 3.9

Problem: We introduced standard cell technology in section 3.7. In this technology, circuits are built by interconnecting building-block cells that implement simple functions, like basic logic gates. A commonly used type of standard cell are the and-or-invert (AOI) cells, which can be efficiently built as CMOS complex gates. Consider the AOI cell shown in Figure 3.70. This cell implements the function $f = \overline{x_1x_2 + x_3x_4 + x_5}$. Derive the CMOS complex gate that implements this cell.

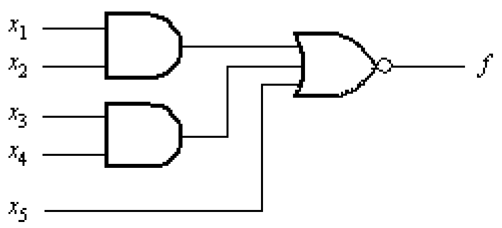


Figure 3.70. The AOI cell for Example 3.9.

Solution: Applying Demorgan's theorem in two steps gives

$$\begin{aligned} f &= \overline{x_1x_2} \cdot \overline{x_3x_4} \cdot \overline{x_5} \\ &= (\overline{x}_1 + \overline{x}_2) \cdot (\overline{x}_3 + \overline{x}_4) \cdot \overline{x}_5 \end{aligned}$$

Since all input variables are complemented in this expression, we can directly derive the pull-up network as having parallel-connected PMOS transistors controlled by x_1 and x_2, in series with parallel-connected transistors controlled by x_3 and x_4, in series with a transistor controlled by x_5. This circuit, along with the corresponding pull-down network, is shown in Figure 3.71.

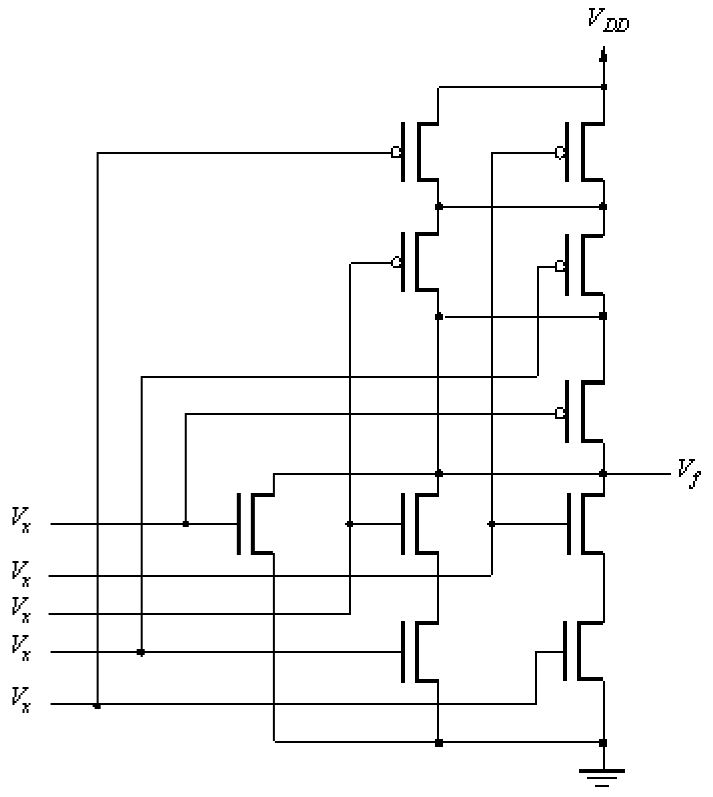


Figure 3.71. Circuit for Example 3.9.

Example 3.10

Problem: For the CMOS complex gate in Figure 3.71, determine the sizes of transistors that should be used such that the speed performance of this gate is similar to that of an inverter.
Solution: Recall from section 3.8.5 that a transistor with length L and width W has a drive strength proportional to the ratio W/L. Also recall that when transistors are connected in parallel their widths are effectively added, leading to an increase in drive strength. Similarly, when transistors are connected in series, their lengths are added, leading to a decrease in drive strength. Let us assume that all NMOS and PMOS transistors have the same length, $L_n = L_p = L$. For the pull-down network in Figure 3.71, the worst-case path involves just a single NMOS transistor. Thus, we can make the length, L_n, of each NMOS transistor the same size as in the inverter. For the pull-up network, the worst-case path involves three transistors in series. Since, as we said in section 3.8.1, PMOS transistors have about half the drive strength of NMOS transistors, we should make the effective size of the three PMOS transistors in series about twice that of an NMOS transistor.

Therefore,

$$L_p = L_n \times 3 \times 2 = 6L_n$$

Example 5.7

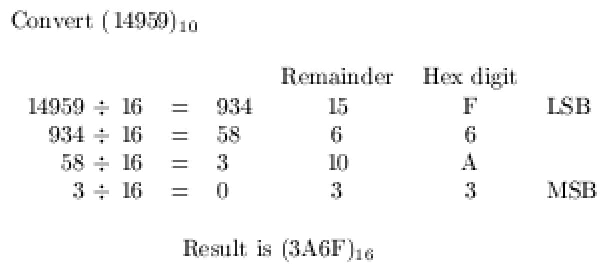
Problem: Convert the decimal number 14959 into a hexadecimal number.
Solution: An integer is converted into the hexadecimal representation by successive divisions by 16, such that in each step the remainder is a hex digit. To see why this is true, consider a four-digit number $H = h_3h_2h_1h_0$. Its value is

$$V = h_3 \times 16^3 + h_2 \times 16^2 + h_1 \times 16 + h_0$$

If we divide this by 16, we obtain

$$\frac{V}{16} = h_3 \times 16^2 + h_2 \times 16 + h_1 + \frac{h_0}{16}$$

Thus, the remainder gives h_0. Figure 5.40 shows the steps needed to perform the conversion $(14959)_{10} = (3A6F)_2$.



Convert $(14959)_{10}$

			Remainder	Hex digit	
$14959 \div 16$	=	934	15	F	LSB
$934 \div 16$	=	58	6	6	
$58 \div 16$	=	3	10	A	
$3 \div 16$	=	0	3	3	MSB

Result is $(3A6F)_{16}$

Figure 5.40. Conversion from decimal to hexadecimal.

Example 5.10

Problem: In computer computations it is often necessary to compare numbers. Two four-bit signed numbers, $X = x_3x_2x_1x_0$ and $Y = y_3y_2y_1y_0$, can be compared by using the subtractor circuit in Figure 5.43, which performs the operation $X - Y$. The three outputs denote the following:

- $Z = 1$ if the result is 0; otherwise $Z = 0$
- $N = 1$ if the result is negative; otherwise $N = 0$
- $V = 1$ if arithmetic overflow occurs; otherwise $V = 0$

Show how Z, N, and V can be used to determine the cases $X = Y$, $X < Y$, $X \leq Y$, $X > Y$, and $X \geq Y$.
Solution: Consider first the case $X < Y$, where the following possibilities may arise:

- If X and Y have the same sign there will be no overflow, hence $V = 0$. Then for both positive and negative X and Y the difference will be negative ($N = 1$).

- If X is negative and Y is positive, the difference will be negative ($N = 1$) if there is no overflow ($V = 0$); but the result will be positive ($N = 0$) if there is overflow ($V = 1$).

Therefore, if $X < Y$ then $N \oplus V = 1$.

The case $X = Y$ is detected by $Z = 1$. Then, $X \leq Y$ is detected by $Z + (N \oplus V) = 1$. The last two cases are just simple inverses: $X > Y$ if $\overline{Z + (N \oplus V)} = 1$ and $X \geq Y$ if $\overline{N \oplus V} = 1$.

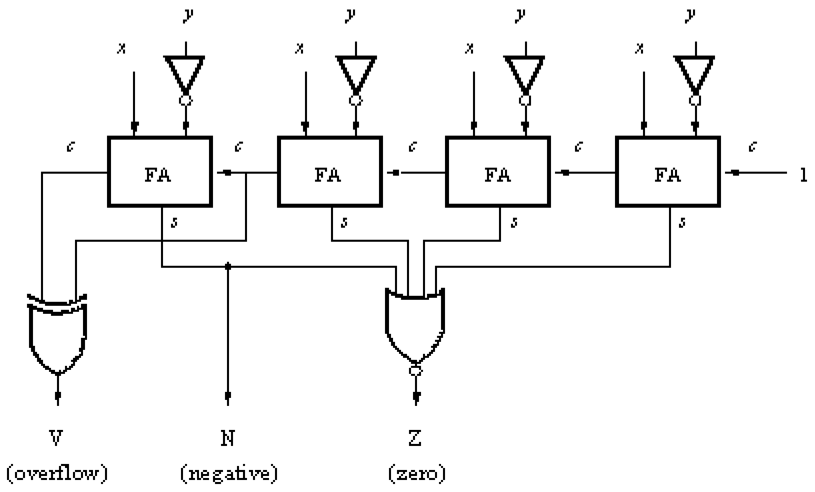


Figure 5.43. A comparator circuit.

Example 6.25

Problem: Implement the function $f(w_1, w_2, w_3) = \sum m(0, 1, 3, 4, 6, 7)$ by using a 3-to-8 binary decoder and an OR gate.

Solution: The decoder generates a separate output for each minterm of the required function. These outputs are then combined in the OR gate, giving the circuit in Figure 6.50.

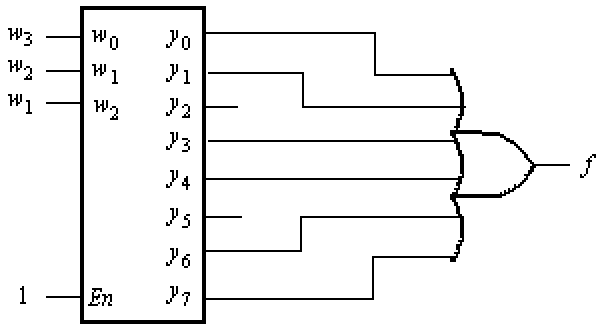


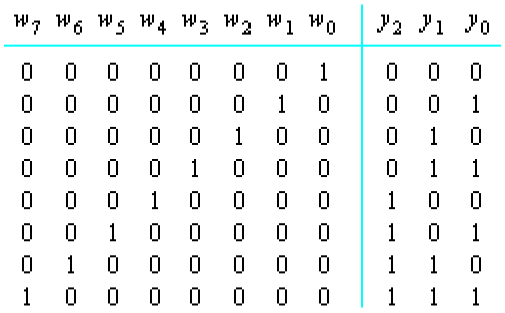
Figure 6.50. Circuit for Example 6.25.

Example 6.26

Problem: Derive a circuit that implements an 8-to-3 binary encoder.

Solution: The truth table for the encoder is shown in Figure 6.51. Only those rows for which a single input variable is equal to 1 are shown; the other rows can be treated as don't care cases. From the truth table it is seen that the desired circuit is defined by the equations

$$\begin{aligned} y_2 &= w_4 + w_5 + w_6 + w_7 \\ y_1 &= w_2 + w_3 + w_6 + w_7 \\ y_0 &= w_1 + w_3 + w_5 + w_7 \end{aligned}$$



w_7	w_6	w_5	w_4	w_3	w_2	w_1	w_0	y_2	y_1	y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

Figure 6.51. Truth table for a 3-to-8 binary encoder.

Example 6.27

Problem: Implement the function

$$f(w_1, w_2, w_3, w_4) = \overline{w}_1\overline{w}_2\overline{w}_4\overline{w}_5 + w_1w_2 + w_1w_3 + w_1w_4 + w_3w_4w_5$$

by using a 4-to-1 multiplexer and as few other gates as possible. Assume that only the uncomplemented inputs w_1, w_2, w_3, and w_4 are available.
Solution: Since variables w_1 and w_4 appear in more product terms in the expression for f than the other three variables, let us perform Shannon's expansion with respect to these two variables. The expansion gives

$$\begin{aligned} f &= \overline{w}_1\overline{w}_4 f_{\overline{w}_1\overline{w}_4} + \overline{w}_1 w_4 f_{\overline{w}_1 w_4} + w_1\overline{w}_4 f_{w_1\overline{w}_4} + w_1w_4 f_{w_1w_4} \\ &= \overline{w}_1\overline{w}_4(\overline{w}_2\overline{w}_5) + \overline{w}_1w_4(w_3w_5) + w_1\overline{w}_4(w_2 + w_3) + w_1w_2(1) \end{aligned}$$

We can use a NOR gate to implement $\overline{w}_2\overline{w}_5 = \overline{w_2 + w_5}$. We also need an AND gate and an OR gate. The complete circuit is presented in Figure 6.52.

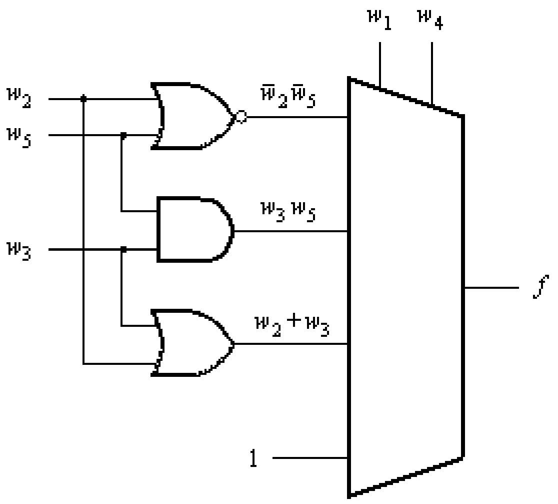


Figure 6.52. Circuit for Example 6.27.

Example 6.28

Problem: In Chapter 4 we pointed out that the rows and columns of a Karnaugh map are labeled using Gray code. This is a code in which consecutive valuations differ in one variable only. Figure 6.53 depicts the conversion between three-bit binary and Gray codes. Design a circuit that can convert a binary code into a Gray according the figure.

b_2	b_1	b_0	g_2	g_1	g_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

Figure 6.53. Binary to Gray code conversion.

Solution: From the figure it follows that

$$\begin{aligned} g_2 &= b_2 \\ g_1 &= b_1\overline{b}_2 + \overline{b}_1 b_2 \\ &= b_1 \oplus b_2 \\ g_0 &= b_0\overline{b}_1 + \overline{b}_0 b_1 \end{aligned}$$

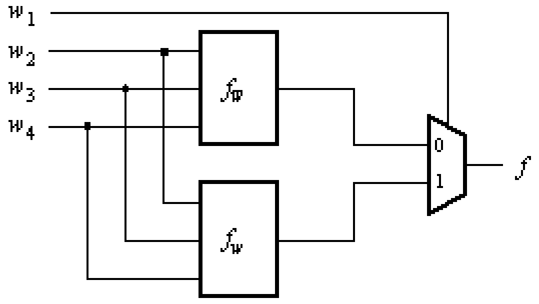
$$= \quad b_0 \oplus b_1$$

Example 6.29

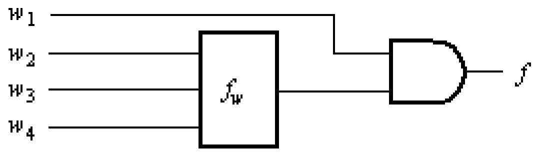
Problem: In section 6.1.2 we showed that any logic function can be decomposed using Shannon's expansion theorem. For a four-variable function, $f(w_1, \ldots, w_4)$, the expansion with respect to w_1 is

$$f(w_1, \ldots, w_4) = \overline{w}_1 f_{\overline{w}_1} + w_1 f_{w_1}$$

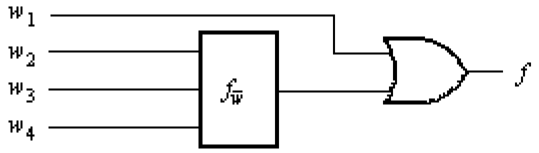
A circuit that implements this expression is given in Figure 6.54*a*.
(a) If the decomposition yields $f_{\overline{w}_1} = 0$, then the multiplexer in the figure can be replaced by a single logic gate. Show this circuit.
(b) Repeat part *a* for the case where $f_{w_1} = 1$.
Solution: The desired circuits are shown in parts (*b*) and (*c*) of Figure 6.54.



(a) Shannon's expansion of the function *f*.



(b) Solution for part *a*



(c) Solution for part *b*

Figure 6.54. Circuits for Example 6.29.

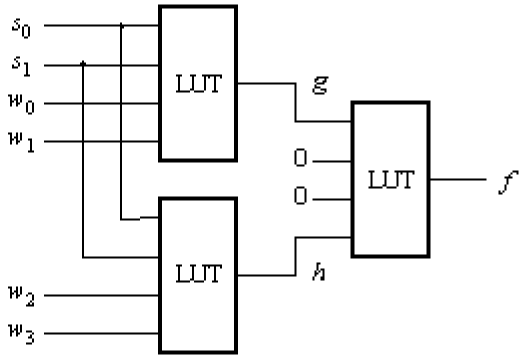
Example 6.30

Problem: In several commercial FPGAs the logic blocks are 4-LUTs. What is the minimum number of 4-LUTs needed to construct a 4-to-1 multiplexer with select inputs s_1 and s_0 and data inputs w_3, w_2, w_1, and w_0?

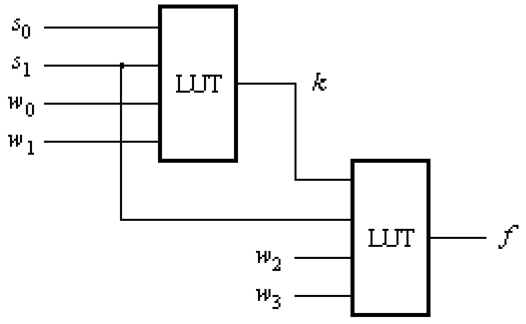
Solution: A straightforward attempt is to use directly the expression that defines the 4-to-1 multiplexer

$$f = \overline{s}_1\overline{s}_0 w_0 + \overline{s}_1 s_0 w_1 + s_1\overline{s}_0 w_2 + s_1 s_0 w_3$$

Let $g = \overline{s}_1\overline{s}_0 w_0 + \overline{s}_1 s_0 w_1$ and $h = s_1\overline{s}_0 w_2 + s_1 s_0 w_3$, so that $f = g + h$. This decomposition leads to the circuit in Figure 6.55*a*, which requires three LUTs.



(a) Using three LUTs



(b) Using two LUTs

Figure 6.55. Circuits for Example 6.30.

When designing logic circuits, one can sometimes come up with a clever idea which leads to a superior implementation. Figure 6.55*b* shows how it is possible to implement the multiplexer with just two LUTs, based on the following observation. The truth table in Figure 6.2*b* indicates that when $s_1 = 0$ the output must be either w_0 or w_1, as determined by the value of s_0. This can be generated by the first LUT. The second LUT must make the choice between w_2 and w_3 when $s_1 = 1$. But, the choice can be made only by knowing the value of s_0. Since it is impossible to have five inputs in the LUT, more information has to be passed from the first to the second LUT. Observe that when $s_1 = 1$ the output f will be equal to either w_2 or w_3, in which case it is not necessary to know the values of w_0 and w_1. Hence, in this case we can pass on the value of s_0 through the first LUT, rather than w_0 or w_1. This can be done by making the function of this LUT

$$k = \overline{s}_1\overline{s}_0w_0 + \overline{s}_1s_0w_1 + s_1s_0$$

Then, the second LUT performs the function

$$f = \overline{s}_1k + s_1\overline{k}w_3 + s_1kw_4$$

Example 6.31

Problem: In digital systems it is often necessary to have circuits that can shift the bits of a vector by one or more bit positions to the left or right. Design a circuit that can shift a four-bit vector $W = w_3w_2w_1w_0$ one bit position to the right when a control signal *Shift* is equal to 1. Let the outputs of the circuit be a four-bit vector $Y = y_3y_2y_1y_0$ and a signal k, such that if $Shift = 1$ then $y_3 = 0$, $y_2 = w_3$, $y_1 = w_2$, $y_0 = w_1$, and $k = w_0$. If $Shift = 0$ then $Y = W$ and $k = 0$.
Solution: The required circuit can be implemented with five 2-to-1 multiplexers as shown in Figure 6.56. The *Shift* signal is used as the select input to each multiplexer.

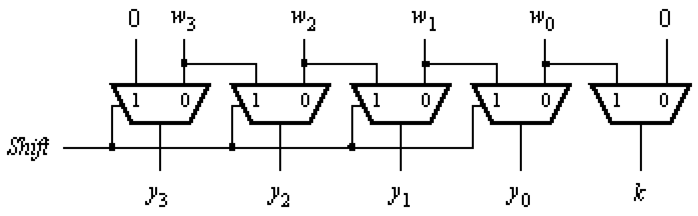


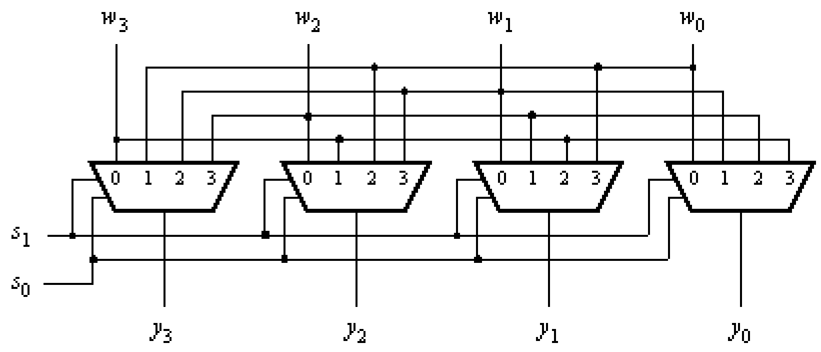
Figure 6.56. A shifter circuit.

Example 6.32

Problem: The shifter circuit in Example 6.31 shifts the bits of an input vector by one bit position to the right. It fills the vacated bit on the left side with 0. A more versatile shifter circuit may be able to shift by more bit positions at a time. If the bits that are shifted out are placed into the vacated positions on the left, then the circuit effectively rotates the bits of the input vector by a specified number of bit positions. Such a circuit is often called a *barrel shifter*. Design a four-bit barrel shifter that rotates the bits by 0, 1, 2, or 3 bit positions as determined by the valuation of two control signals s_1 and s_0.
Solution: The required action is given in Figure 6.57*a*. The barrel shifter can be implemented with four 4-to-1 multiplexers as shown in Figure 6.57*b*. The control signals s_1 and s_0 are used as the select inputs to the multiplexers.

s_1	s_0	y_3	y_2	y_1	y_0
0	0	w_3	w_2	w_1	w_0
0	1	w_0	w_3	w_2	w_1
1	0	w_1	w_0	w_3	w_2
1	1	w_2	w_1	w_0	w_3

(a) Truth table

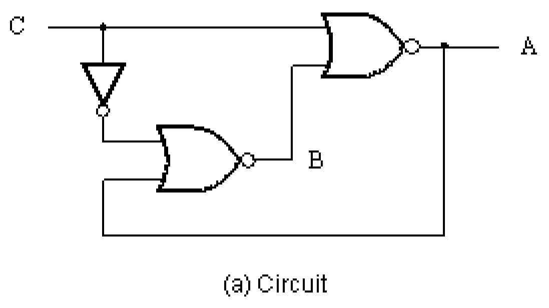


(b) Circuit

Figure 6.57. A barrel shifter circuit.

Example 7.13

Problem: Consider the circuit in Figure 7.80*a*. Assume that the input C is driven by a square wave signal with a 50% duty cycle. Draw a timing diagram that shows the waveforms at points A and B. Assume that the propagation delay through each gate is Δ seconds.
Solution: The timing diagram is shown in Figure 7.80*b*.



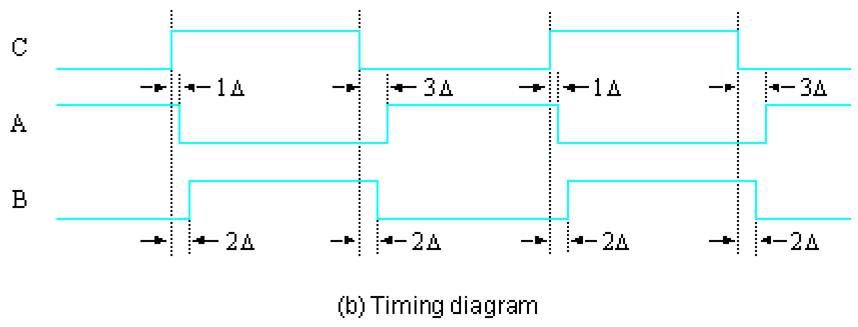


Figure 7.80. Circuit for Example 7.13.

Example 7.14

Problem: Determine the functional behavior of the circuit in Figure 7.81. Assume that input w is driven by a square wave signal.
Solution: When both flip-flops are cleared, their outputs are $Q_0 = Q_1 = 0$. After the Clear input goes high, each pulse on the w input will cause a change in the flip-flops as indicated in Figure 7.82. Note that the figure shows the state of the signals after the changes caused by the rising edge of a pulse have taken place.

In consecutive time intervals the values of Q_1Q_0 are 00, 01, 10, 00, 01, and so on. Therefore, the circuit generates the counting sequence: 0, 1, 2, 0, 1, and so on. Hence, the circuit is a modulo-3 counter.

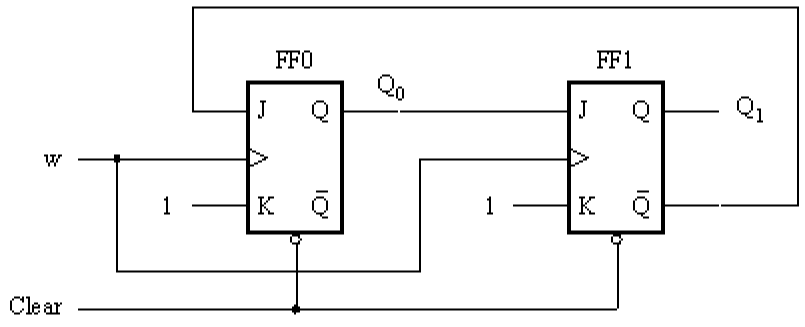


Figure 7.81. Circuit for Example 7.14.

Time interval	FF0			FF1		
	J_0	K_0	Q_0	J_1	K_1	Q_1
Clear	1	1	0	0	1	0
t_1	1	1	1	1	1	0
t_2	0	1	0	0	1	1
t_3	1	1	0	0	1	0
t_4	1	1	1	1	1	0

Figure 7.82. Summary of the behavior of the circuit in Figure 7.81.

Example 7.15

Problem: Figure 7.74 (in the text book) shows a circuit that generates four timing control signals T_0, T_1, T_2, and T_3. Design a circuit that generates six such signals, T_0 to T_5.
Solution: The scheme of Figure 7.75 (in the text book) can be extended by using a modulo-6 counter, given in Figure 7.26 (in the text book), and a decoder that produces the six timing signals. A simpler alternative is possible by using a Johnson counter. Using three D-type flip-flops in a structure depicted in Figure 7.30 (in the text book), we can generate six patterns of bits $Q_0Q_1Q_2$ as shown in Figure 7.83. Then, using only six more two-input AND gates, as shown in the figure, we can obtain the desired signals. Note that the patterns $Q_0Q_1Q_2$ equal to 010 and 101 cannot occur in the Johnson counter, so these cases are treated as don't care conditions.

Clock cycle	Q_0	Q_1	Q_2	Control signal
0	0	0	0	$T_0 = \overline{Q}_0\overline{Q}_2$
1	1	0	0	$T_1 = Q_0\overline{Q}_1$
2	1	1	0	$T_2 = Q_1\overline{Q}_2$
3	1	1	1	$T_3 = Q_0Q_2$
4	0	1	1	$T_4 = \overline{Q}_0Q_1$
5	0	0	1	$T_5 = \overline{Q}_1Q_2$

Figure 7.83. Timing signals for Example 7.15.

Example 7.16

Problem: Design a circuit that can be used to control a vending machine. The circuit has five inputs: Q (quarter), D (dime), N (nickel), *Coin*, and *Resetn*. When a coin is deposited in the machine, a coin-sensing mechanism generates a pulse on the appropriate input (Q, D, or N). To signify the occurrence of the event, the mechanism also generates a pulse on the line *Coin*. The circuit is reset by using the *Resetn* signal (active low). When at least 30 cents has been deposited, the circuit activates its output, Z. No change is given if the amount exceeds 30 cents.

Design the required circuit by using the following components: a six-bit adder, a six-bit register, and any number of AND, OR, and NOT gates.

Solution: Figure 7.84 gives a possible circuit. The value of each coin is represented by a corresponding five-bit number. It is added to the current total, which is held in register S. The required output is

$$Z = s_5 + s_4 s_3 s_2 s_1$$

The register is clocked by the negative edge of the *Coin* signal. This allows for a propagation delay through the adder, and ensures that a correct sum will be placed into the register.

In Chapter 9 we will show how this type of control circuit can be designed using a more structured approach.

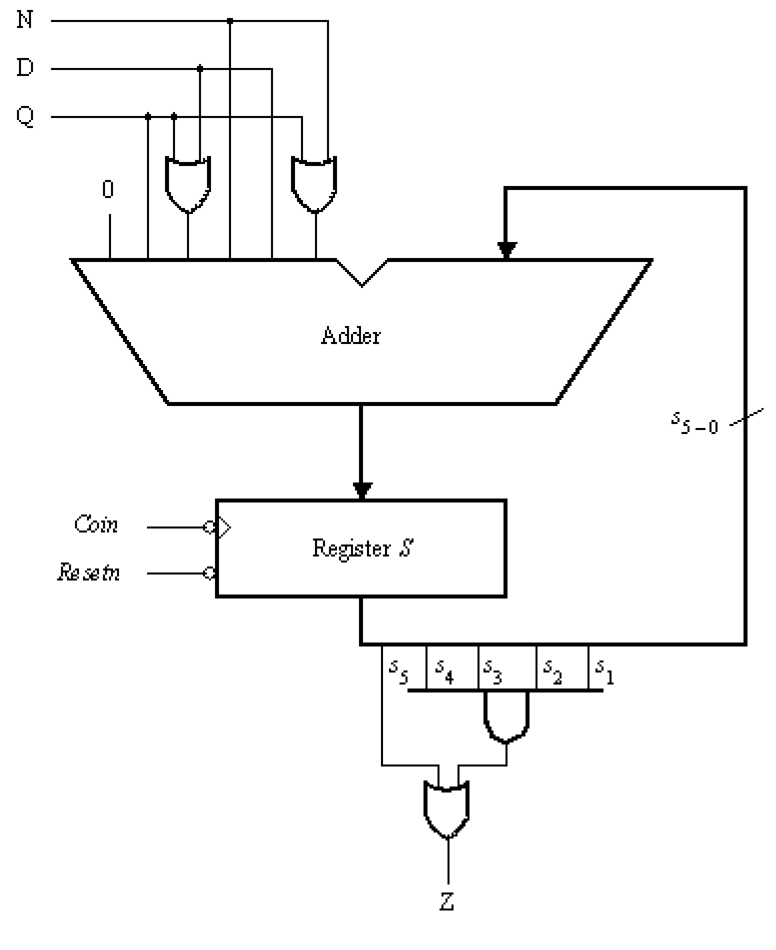


Figure 7.84. Circuit for Example 7.16.

Example 8.11

Problem: Design an FSM that has an input w and an output z. The machine is a sequence detector that produces $z = 1$ when the previous two values of w were 00 or 11; otherwise $z = 0$.

Solution: Section 8.1 presents the design of a sequence detector that detects the occurrence of consecutive 1s. Using the same approach, the desired FSM can be specified using the state diagram in Figure 8.91. State C denotes the occurrence of two or more 0s, and state E denotes two or more 1s. The corresponding state table is shown in Figure 8.92.

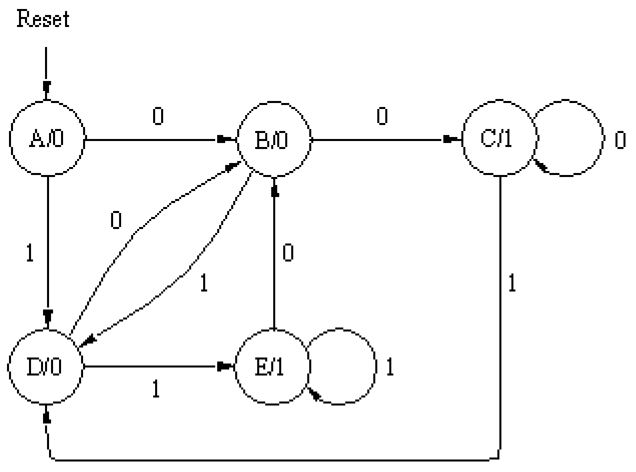


Figure 8.91. State diagram for Example 8.11.

We can try to reduce the number of states by using the partitioning minimization procedure in section 8.6, which gives the following partitions

$$\begin{aligned} P_1 &= (ABCDE) \\ P_2 &= (ABD)(CE) \\ P_3 &= (A)(B)(C)(D)(E) \end{aligned}$$

Since all five states are needed, we have to use three flip-flops.

Present state	Next state		Output
	$w=0$	$w=1$	z
A	B	D	0
B	C	D	0
C	C	D	1
D	B	E	0
E	B	E	1

Figure 8.92. State table for the FSM in Figure 8.91.

A straightforward state assignment leads to the state-assigned table in Figure 8.93. The codes $y_3y_2y_1 = 101, 110, 111$ can be treated as don't-care conditions. Then the next-state expressions are

$$\begin{aligned} Y_1 &= w\overline{y}_1\overline{y}_3 + w\overline{y}_2\overline{y}_3 + \overline{w}y_1y_2 + \overline{w}\overline{y}_1\overline{y}_2 \\ Y_2 &= y_1\overline{y}_2 + \overline{y}_1y_2 + w\overline{y}_2\overline{y}_3 \\ Y_3 &= wy_3 + wy_1y_2 \end{aligned}$$

The output expression is

$$z = y_3 + \overline{y}_1 y_2$$

	Present state	Next state		Output
		$w = 0$	$w = 1$	
	$y_3 y_2 y_1$	$Y_3 Y_2 Y_1$	$Y_3 Y_2 Y_1$	z
A	000	001	011	0
B	001	010	011	0
C	010	010	011	1
D	011	001	100	0
E	100	001	100	1

Figure 8.93. State-assigned table for the FSM in Figure 8.92.

These expressions seem to be unnecessarily complex, suggesting that we may attempt to find a better state assignment. Observe that state A is reached only when the machine is reset by means of the *Reset* input. So, it may be advantageous to assign the four codes in which $y_3 = 1$ to the states B, C, D, and E. The result is the state-assigned table in Figure 8.94. From it, the next-state and output expressions are

$$\begin{aligned} Y_1 &= wy_2 + \overline{w}y_3\overline{y}_2 \\ Y_2 &= w \\ Y_3 &= 1 \\ z &= y_1 \end{aligned}$$

This is a much better solution.

	Present state	Next state		Output
		$w = 0$	$w = 1$	
	$y_3 y_2 y_1$	$Y_3 Y_2 Y_1$	$Y_3 Y_2 Y_1$	z
A	000	100	110	0
B	100	101	110	0
C	101	101	110	1
D	110	100	111	0
E	111	100	111	1

Figure 8.94. An improved state assignment for the FSM in Figure 8.92.

Example 8.12

Problem: Implement the sequence detector of Example 8.11 by using two FSMs. One FSM detects the occurrence of consecutive 1s, while the other detects consecutive 0s.

Solution: A good realization of the FSM that detects consecutive 1s is given in Figures 8.16 and 8.17. The next-state and output expressions are

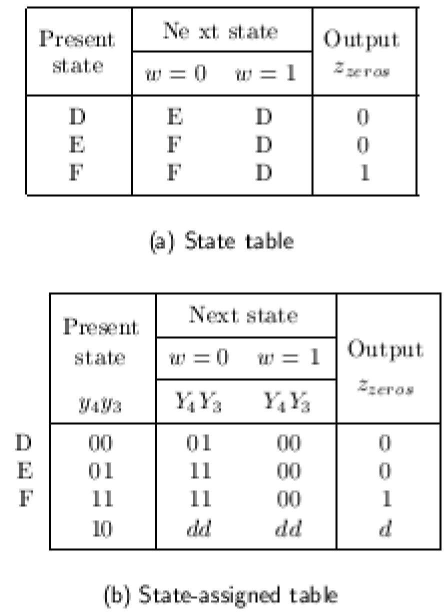
$$\begin{aligned} Y_1 &= w \\ Y_2 &= wy_1 \\ z_{ones} &= y_2 \end{aligned}$$

A similar FSM that detects consecutive 0s is defined in Figure 8.95. Its expressions are

$$\begin{aligned} Y_3 &= \overline{w} \\ Y_4 &= \overline{w}y_3 \\ z_{zeros} &= y_4 \end{aligned}$$

The output of the combined circuit is

$$z = z_{ones} + z_{zeros}$$



Present state	Next state		Output
	$w = 0$	$w = 1$	z_{zeros}
D	E	D	0
E	F	D	0
F	F	D	1

(a) State table

	Present state	Next state		Output
		$w = 0$	$w = 1$	z_{zeros}
	y_4y_3	Y_4Y_3	Y_4Y_3	
D	00	01	00	0
E	01	11	00	0
F	11	11	00	1
	10	*dd*	*dd*	*d*

(b) State-assigned table

Figure 8.95. FSM that detects a sequence of two zeros.

Example 8.13

Problem: Derive a Mealy-type FSM that can act as a sequence detector described in Example 8.11.

Solution: A state diagram for the desired FSM is depicted in Figure 8.96. The corresponding state table is presented in Figure 8.97. Two flip-flops are needed to implement this FSM. A state-assigned table is given in Figure 8.98, which leads to the next-state and output expressions

$$\begin{aligned} Y_1 &= 1 \\ Y_2 &= w \\ z &= \overline{w} y_1 \overline{y}_2 + w y_2 \end{aligned}$$

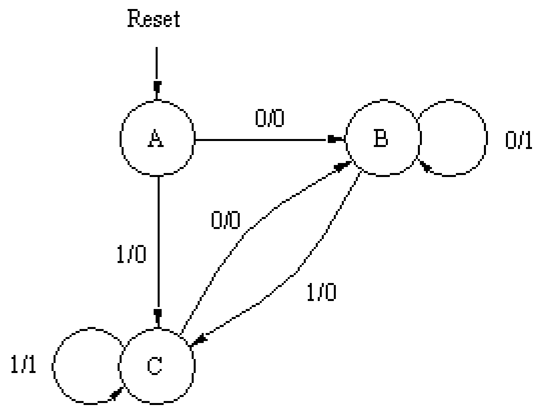


Figure 8.96. State diagram for Example 8.13.

Present state	Next state		Output z	
	$w=0$	$w=1$	$w=0$	$w=1$
A	B	C	0	0
B	B	C	1	0
C	B	C	0	1

Figure 8.97. State table for the FSM in Figure 8.96.

	Present state	Next state		Output	
		$w=0$	$w=1$	$w=0$	$w=1$
	y_2y_1	Y_2Y_1	Y_2Y_1	z	z
A	00	01	11	0	0
B	01	01	11	1	0
C	11	01	11	0	1

Figure 8.98. State-assigned table for the FSM in Figure 8.97.

Example 8.17

Problem: In computer systems it is often desirable to transmit data serially, namely, one bit at a time, to save on the cost of interconnecting cables. This means that parallel data at one end must be transmitted serially, and at the other end the received serial data has to be turned back into parallel form. Suppose that we wish to transmit ASCII characters in this manner. As explained in section 5.8, the standard ASCII code uses seven bits to define each character. Usually, a character occupies one byte, in which case the eighth bit can either be set to 0 or it can be used to indicate the parity of the other bits to ensure a more reliable transmission.

Parallel-to-serial conversion can be done by means of a shift register. Assume that a circuit accepts parallel data, $B = b_7, b_6, \ldots, b_0$, representing ASCII characters. Assume also that bit b_7 is set to 0. The circuit is supposed to generate a parity bit, p, and send it instead of b_7 as a part of the serial transfer. Figure 8.102 gives a possible circuit. An FSM is used to generate the parity bit, which is included in the output stream by using a multiplexer. A three-bit counter is used to determine when the p bit is transmitted, which happens when the count reaches 7. Design the desired FSM.

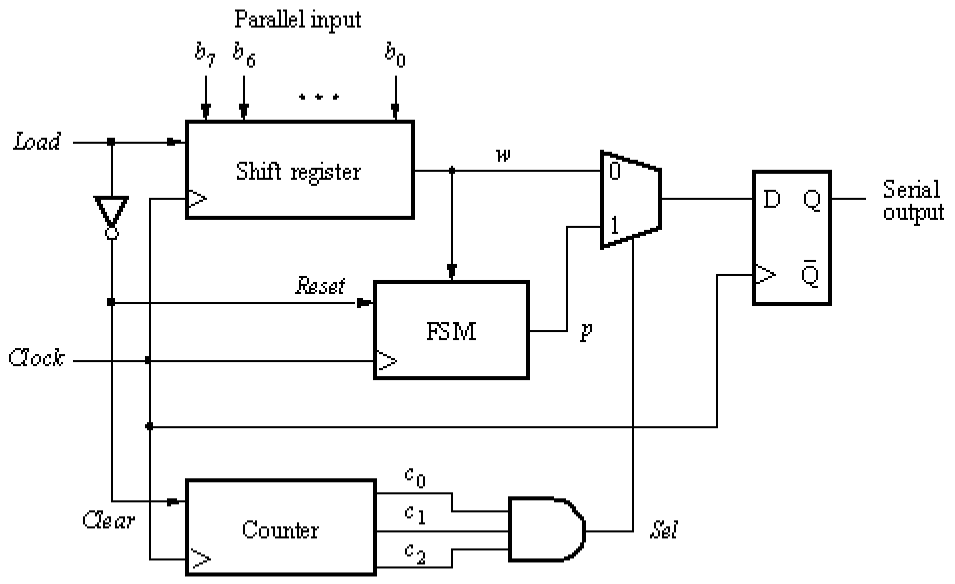


Figure 8.102. Parallel-to-serial converter.

Solution: As the bits are shifted out of the shift register, the FSM examines the bits and keeps track of whether there has been an even or odd number of 1s. It sets p to 1 if there is odd parity. Hence, the FSM must have two states. Figure 8.103 presents the state table, the state-assigned table, and the resulting circuit. The next state expression is

$$Y = \overline{w}y + w\overline{y}$$

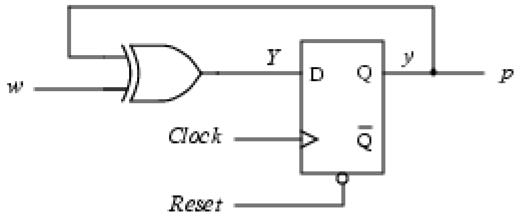
The output p is just equal to y.

Present state	Next state		Output
	$w = 0$	$w = 1$	p
S_{even}	S_{even}	S_{odd}	0
S_{odd}	S_{odd}	S_{even}	1

(a) State table

Present state	Next state		Output
	$w = 0$	$w = 1$	p
y	Y	Y	
0	0	1	0
1	1	0	1

(b) State-assigned table



(c) Circuit

Figure 8.103. FSM for parity generation.