# Chapter 11

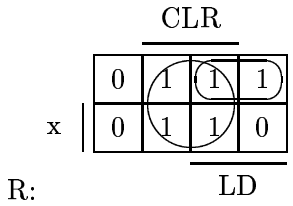
**Exercise 11.1** The register requires the load (LD), clear (CLR), and data inputs that are not available in the SR flip flop. We need to design a combinational network to generate the correct SR inputs as shown in the next table:

| $x$ | LD | CLR | S | R |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |

From the table we obtain:

$$S = x.LD.CLR'$$

R:




$$R = CLR + LD.x'$$

The circuit for the 4-bit register using SR flip flops is shown in Figure 11.1.

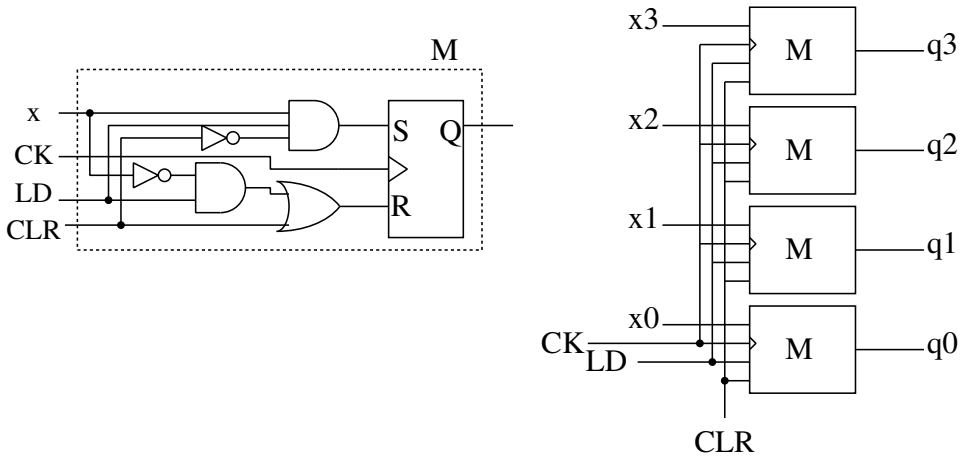



Figure 11.1: 4-bit register using SR flip flops - Exercise 11.1

**Exercise 11.3**

Bit-serial arithmetic for addition/subtraction of 16-bit operands in two's complement form. The network for this exercise is shown in Figure 11.2. The two serial inputs $a$ and $b$ are added/subtracted generating the result $s$, from least-significant to most-significant bit. When subtraction is requested, a carry in of 1 in the first clock cycle is applied to the Full-Adder, together with the complementation of all bits of $b$. When the $16^{th}$ bit of the number is being computed, the signal *last_bit* is 1. Observe that the serial hardware may compute a different number of bits only changing the counter design.

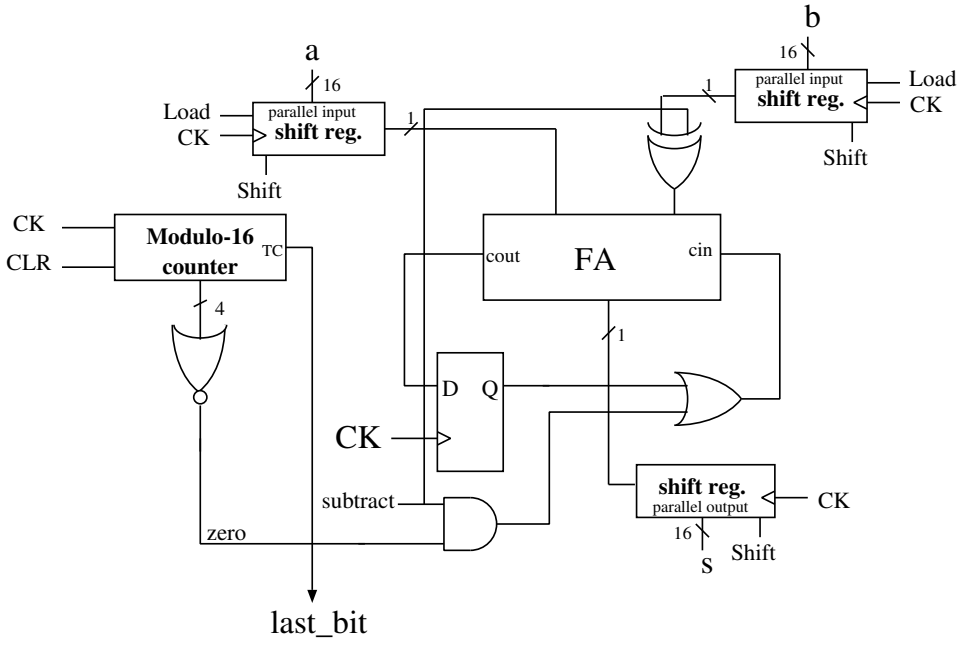



Figure 11.2: Serial addition - Exercise 11.3

| K0 | K1 | K2 | K3 | $x$ | $y$ | $S_i$ | $c_{i+1}$ | shift reg. | FF | $2^{nd}$ adder input $(K1+K2)FF$ | Internal carry (serial adder) | $z$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0000 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1000 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1100 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1110 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1111 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1111 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1111 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1111 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0111 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0011 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0001 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0000 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0000 | 0 | 0 | 0 | 0 |

Table 11.1: Circuit behavior - Exercise 11.5

**Exercise 11.5**: (a) Consider the information provided in Table 11.1, which presents the network operation for the inputs $x = 01101001$ and $y = 00010110$:

Observe that a new digit begins when $K_0 = 1$. There is a delay of 4 cycles between the input digits and the output digit. The result of the operation is 85 in decimal. It is correct since $x = 69$ and $y = 16$.

(b) The output of the first serial adder is the radix-2 sum of the $x$ and $y$ bit strings. Consider a group of four bits corresponding to one decimal digit of $x$ and $y$. These bits are applied at the input of the adder from least significant to most significant at clock cycles when $K_0$, $K_1$, $K_2$, and $K_3$ are active. Consequently, at the time $K_3$ is active, the sum is $(c_4, s_3, s_2, s_1, s_0)$, as indicated below:

$$\begin{array}{ccccc} & x_3 & x_2 & x_1 & x_0 \\ + & y_3 & y_2 & y_1 & y_0 \\ \hline c_4 & s_3 & s_2 & s_1 & s_0 \end{array}$$

Bits $(s_2, s_1, s_0)$ are inside the shifter, $s_3$ and $c_4$ are outputs of the first adder. Since the result should be in BCD (radix-10) it is necessary to correct it whenever it is larger than 9. That is:

$$z = s \bmod 10 = \begin{cases} s & \textbf{if } s \leq 9 \\ s - 10 & \textbf{if } s \geq 10 \end{cases} \tag{11.1}$$

where $s = 16c_4 + \sum_{i=0}^{3} s_i 2^i$. Consequently, it is necessary to detect whether the addition of 2 decimal digits is greater than 9 and then subtract 10 form the result.

Moreover, the carry to be used for the next digit is

$$C_{out} = \begin{cases} 1 & \textbf{if } s \geq 10 \\ 0 & \textbf{if } s \leq 9 \end{cases} \tag{11.2}$$

which can be written as

$$C_{out} = \begin{cases} 1 & \textbf{if } (s \geq 16) \textbf{ or } \ (10 \leq s < 16) \\ 0 & \textbf{otherwise} \end{cases} \tag{11.3}$$

As BCD digits come in groups of 4 bits, the detection of greater or equal to 10 condition as presented in Equation 11.3 corresponds to the following expression:

$$G = c_4 + s_3(s_2 + s_1)$$

This detection is implemented in the combinational network connected to the outputs of the first adder and the shift register. The value of $G$ is stored in the FF only when $K_3 = 1$. In any other cycle the value in the FF doesn't change (function performed by the FF and MUX circuit).

For the subtraction of 10 (when the addition of the two BCD digits are larger than 9), since binary adders add modulo-16 instead, the following equation must be considered:

$$s - 10 = (s + 6) \bmod 16$$

This addition of 6 is performed using the second binary adder. Since the binary representation of 6 is 0110, it's necessary to add 1 in two consecutive cycles. The least significant bit of the first addition reaches the second adder when $K_0$ is active, if the addition of 6 is necessary, the input of the second adder must have a 1 during the cycles when $K_1$ or $K_2$ are active.

Note that when the least significant bits of another pair of BCD digits come into the first adder, the least significant bit of the addition of the previous pair is at the input of the second serial adder.

The analysis finishes when we consider the carry bits of the BCD addition. When the sum value of the digits of $x$ and $y$ is a value greater than 15, the carry bit is stored in the first adder, to be used with the next BCD digits. When the value is in the range 10 to 15, the correction step (addition of 6) executed in the second adder generates a carry which is stored in the second adder for the next sum value which is the contents of the shifter.

The timing diagram is shown in Figure 11.3.

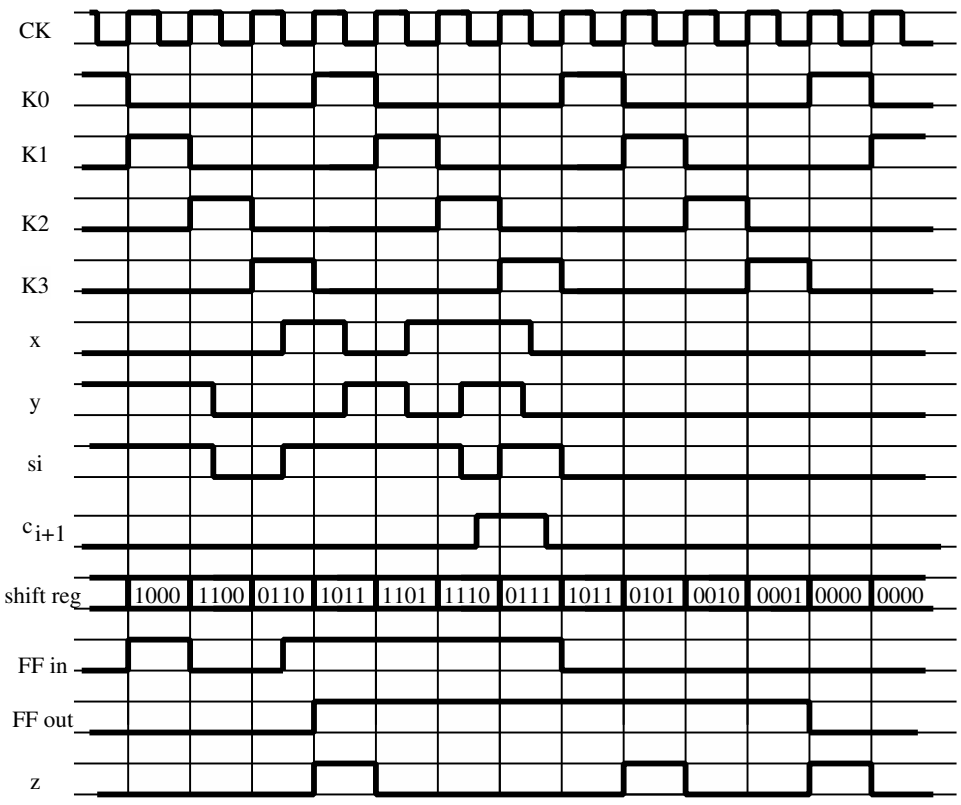



Figure 11.3: Timing diagram for Exercise 11.5

**Exercise 11.7:**

(a) The implementation of the serial change of sign module (complementer) for a 32-bit number in two's complement system is shown in Figure 11.4. After the clear signal the counter is in state 0 and the NOR gate connected to its output forces a 1 input into the half-adder (HA). The other input of the HA is connected to the complemented serial output of the shift register. The bits are shifted out least-significant bit first and stored back into the same register. The carry bit of each bit addition is delayed and inserted back into the HA, as done in serial addition.

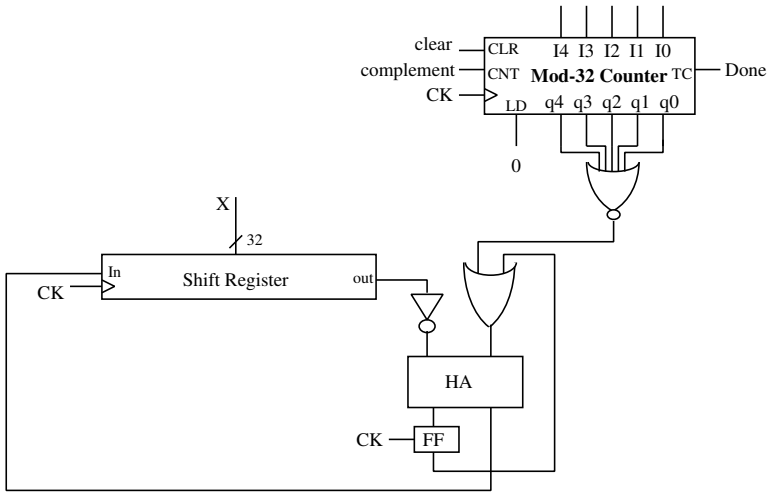



Figure 11.4: Serial change of sign (two's complement system) - Exercise 11.7

(b) When the adder receives two bits each clock cycle, it is necessary to use two shift registers. One stores the bits of $X$ in even positions, $X_{even} = (x_{30}, x_{28}, \ldots, x_2, x_0)$. The other stores the bits in odd positions, $X_{odd} = (x_{31}, x_{29}, \ldots, x_3, x_1)$. The output is stored back into the registers with the same organization. Two HAs with extra logic are used to increment $X'$. Since 2 bits per clock cycle are processed, a modulo-16 counter is enough. This design is shown in Figure 11.5.

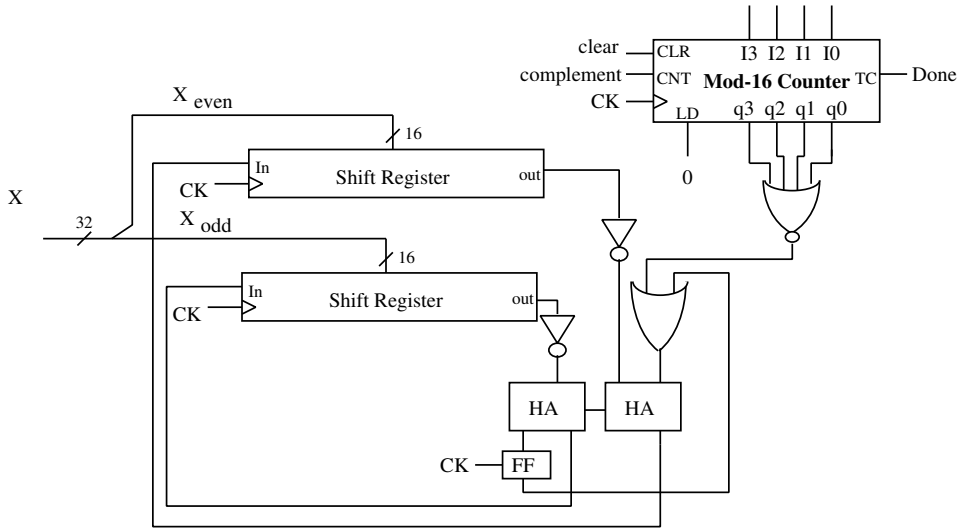



Figure 11.5: Serial complementer (two's complement system) - Exercise 11.7

**Exercise 11.9:** The design is presented in Figure 11.6. A modulo-m divider is implemented by a counter that loads the value $16 - m$ every time TC is 1. In two's complement system, the computation of $16 - m$ results in $-m$. The value of $-m$ is obtained from $m$ by bit complementation (*Compl* module) and addition of 1. Another option is to use $15 - m$ as a loading value and load the counter with this value when it reaches the state 14. Such operation corresponds to the one's complement of $m$ and it is obtained by simple bit complementation (no incrementer).

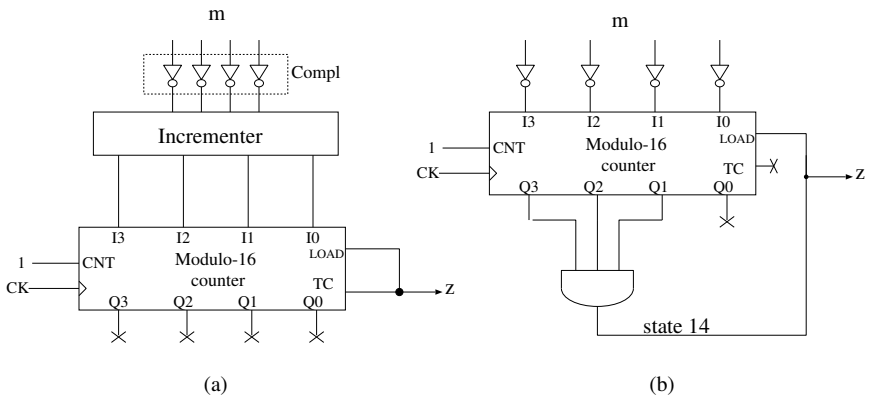



Figure 11.6: Programmable modulo-m divider - Exercise 11.9

**Exercise 11.10:** The solution for this exercise is shown in Figure 11.7. The network computes the next state adding one to the present state value when the value is less than 9, or adding 7 to the present state when it reaches 9 (forcing the counter to go back to state 0). Using high-level specification we have:

$$s(t) = \begin{cases} s(t)+1 & \text{if } s(t) < 9 \\ (s(t)+7) \bmod 16 = 0 & \text{if } s(t) = 9 \end{cases}$$

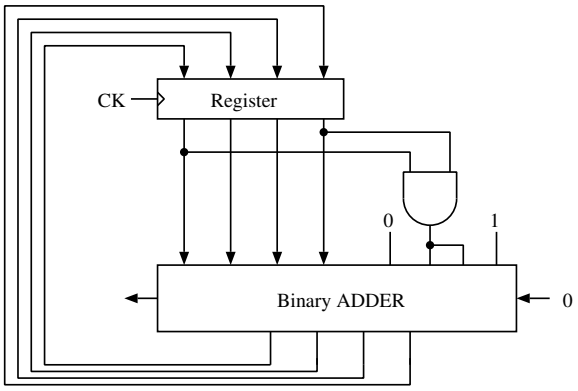
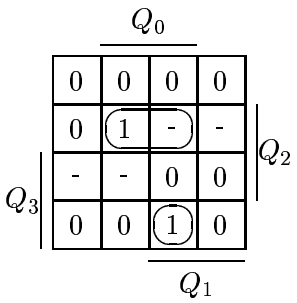



Figure 11.7: BCD counter - Exercise 11.10

**Exercise 11.11:** Using a modulo-16 binary counter with parallel inputs

(c) The state diagram is shown in Figure 11.8, with notes associated to the transitions for which the counter must be loaded. From the diagram we obtain the condition for loading on a Kmap as follows:




| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | - | - |
| - | - | 0 | 0 |
| 0 | 0 | 1 | 0 |

that results in the minimal expression:

$$LOAD = (Q_3 Q_2' Q_1 Q_0 + Q_3' Q_2 Q_0)x$$

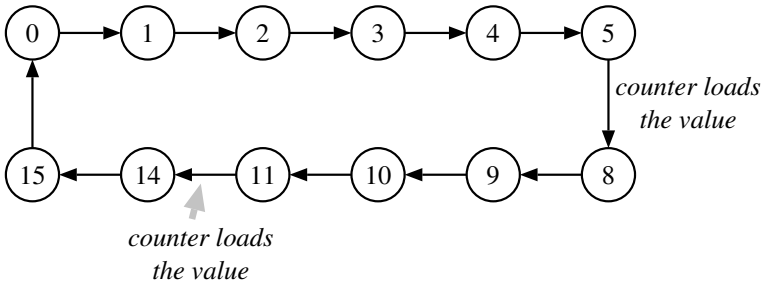



Figure 11.8: State diagram for Exercise 11.11(c)

We use the fact that LOAD overides CNT input to define:

$$CNT = x$$

Considering the inputs $I_i$ as d.c.s for the cases when $LOAD = 0$ and the appropriate next state value for when $LOAD = 1$, and using Kmaps we obtain the expressions:

$$\begin{aligned} I_3 &= 1 \\ I_2 &= I_1 = Q_1 \\ I_0 &= 0 \end{aligned}$$

The network for this system is shown in Figure 11.9.

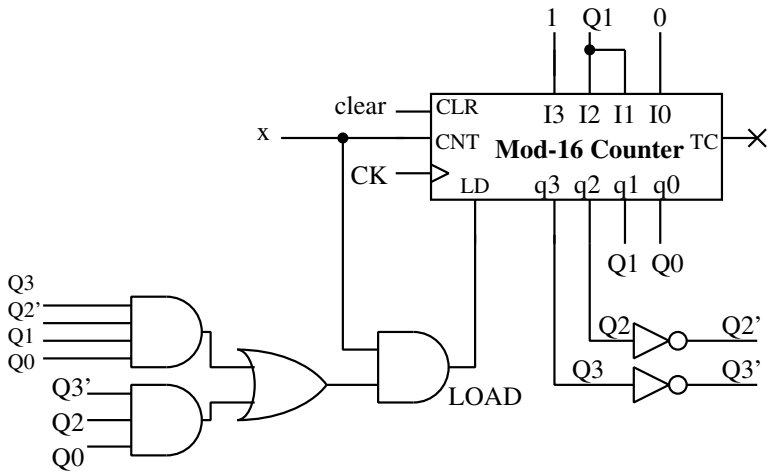



Figure 11.9: Network for Exercise 11.11(c)

**Exercise 11.13:**

Assuming that the module-16 counter has only up-counting feature, the down-counting must be implemented by loading the next state and a zero must be loaded when up-counting and the counter state (*count*) is 10.

The LOAD signal is expressed as:

$$LOAD = \begin{cases} 1 & \text{if } (countup \text{ and } count = 10) \text{ or } countdown \\ 0 & \text{otherwise} \end{cases}$$

The value to be loaded is:

$$i = \begin{cases} i-1 & \text{if } countdown \text{ and } count \neq 0 \\ (10)_{10} & \text{if } countdown \text{ and } count = 0 \\ 0 & \text{if } countup \text{ and } count = (10)_{10} \\ \text{don't care} & otherwise \end{cases}$$

Representing the parallel input $i$ by a vector $I = (I_3, I_2, I_1, I_0)$ and the *count* state by a vector $Q = (Q_3, Q_2, Q_1, Q_0)$, the following expression is obtained:

$$LOAD = countup.Q_3.Q_1 + countdown$$

Lets first consider the values to be loaded when counting down.

| $Q = (Q_3, Q_2, Q_1, Q_0)$ | $I = (I_3, I_2, I_1, I_0)$ |
|---|---|
| 0000 | 1010 |
| 0001 | 0000 |
| 0010 | 0001 |
| 0011 | 0010 |
| 0100 | 0011 |
| 0101 | 0100 |
| 0110 | 0101 |
| 0111 | 0110 |
| 1000 | 0111 |
| 1001 | 1000 |
| 1010 | 1001 |
| 1011 | - - - - |
| 1100 | - - - - |
| 1101 | - - - - |
| 1110 | - - - - |
| 1111 | - - - - |

$I_3$:

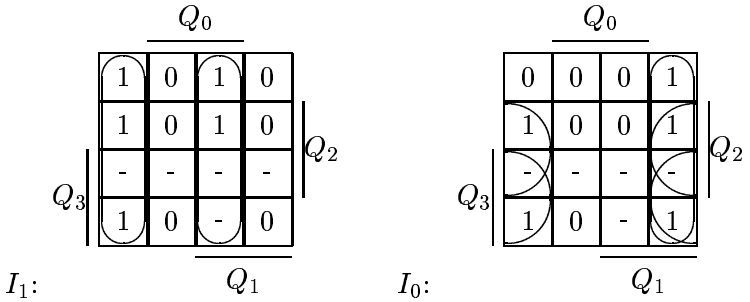
| $Q_3$ \ $Q_0Q_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 1 | 0 | 0 | 0 |
| 01 | 0 | 0 | 0 | 0 |
| 11 | - | - | - | - |
| 10 | 0 | 1 | - | 1 |

$I_2$:

| $Q_3$ \ $Q_0Q_1$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 0 | 1 | 1 | 1 |
| 11 | - | - | - | - |
| 10 | 1 | 0 | - | 0 |

$$I_3 = (Q_3'Q_2'Q_1'Q_0' + Q_0Q_3 + Q_1Q_3).countdown$$

$$I_2 = (Q_2Q_0 + Q_2Q_1 + Q_3Q_1'Q_0').countdown$$




$$I_1 = (Q_1'Q_0' + Q_1Q_0).countdown$$

$$I_0 = (Q_2Q_0' + Q_3Q_0' + Q_1Q_0').countdown$$

Since when $countup = 1$ we must have $countdown = 0$, the value loaded when counting up (at state $count = 10$) is $I = 0$, which is correct.

The network that implements the counter is shown in Figure 11.10. Black boxes were used to represent the gate networks that implement $I_3, I_2, I_1$, and $I_0$.

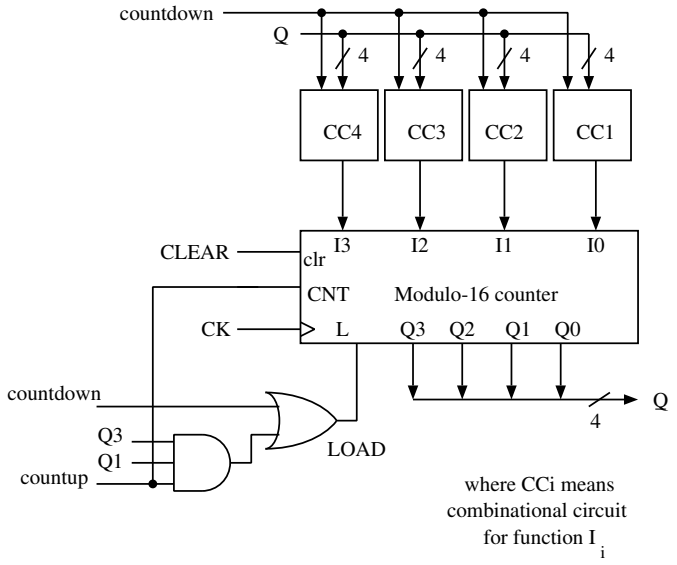



Figure 11.10: Modulo-11 up/down counter - Exercise 11.13

**Exercise 11.15:**

There is a mistake on Figure 11.32 of the textbook. The two counters should be configured as shown in Figure 11.11.

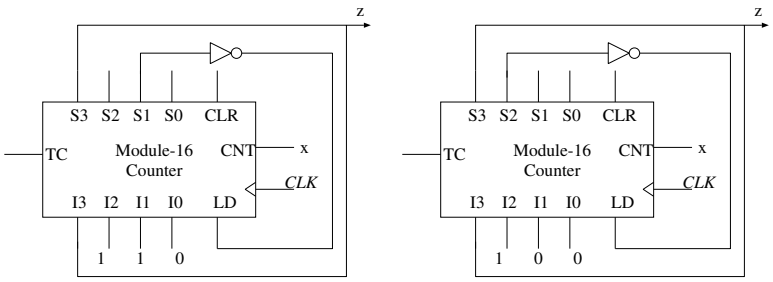



Figure 11.11: Frequency dividers for Exercise 11.15

The first counter has a load condition given as

$$LD = S_1'$$

for which the value loaded is $I = (S_3, 1, 1, 0)$, and $z = S_3$. Looking at all possible present states we are able to obtain the following state-transition and output table:

| PS | NS | output ($z$) |
|---|---|---|
| 0000 | 0110 | 0 |
| 0110 | 0111 | 0 |
| 0111 | 1000 | 0 |
| 1000 | 1110 | 1 |
| 1110 | 1111 | 1 |
| 1111 | 0000 | 1 |

This counter implements a modulo-6 frequency divider, with 50%-duty-cycle frequency.

The second counter has $LD = S_2'$, $z = S_3$, and $I = (S_3, 1, 0, 0)$. The state-transition and output table for this case is:

| PS | NS | $z$ |
|---|---|---|
| 0000 | 0100 | 0 |
| 0100 | 0101 | 0 |
| 0101 | 0110 | 0 |
| 0110 | 0111 | 0 |
| 0111 | 1000 | 0 |
| 1000 | 1100 | 1 |
| 1100 | 1101 | 1 |
| 1101 | 1110 | 1 |
| 1110 | 1111 | 1 |
| 1111 | 0000 | 1 |

This counter implements a modulo-10 frequency divider, with 50%-duty-cycle frequency.

The timing diagram for both cases is shown in Figure 11.12.

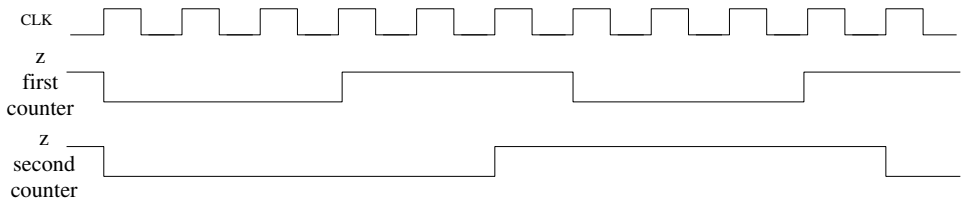



Figure 11.12: Timing digram for counters in Exercise 11.15

**Exercise 11.17:**

From the network in Figure 11.33 of the text we get the following expressions for timing behavior:

$$\begin{aligned} z(t+1) &= [z(t) + COUNT(t)] \bmod 256 \\ COUNT(t+1) &= [COUNT(t) + 1] \bmod 256 \end{aligned}$$

with the initial condition $COUNT(0) = c_0$ we get the solution for the second recurrence as:

$$COUNT(t) = (t + c_0) \bmod 256$$

The first recurrence equation is transformed to:

$$z(t+1) = [z(t) + ((t + c_0) \bmod 256)] \bmod 256 = [z(t) + t + c_0] \bmod 256$$

To find a solution for the first recurrence, let us evaluate a few terms, considering the initial condition $z(0) = z_0$:

$$\begin{aligned} z(1) &= (z_0 + 0 + c_0) \bmod 256 \\ z(2) &= ((z_0 + 0 + c_0) + 1 + c_0) \bmod 256 = (z_0 + 1 + 2c_0) \bmod 256 \end{aligned}$$

Then the general solution is:

$$z(t) = (z_0 + \sum_{i=0}^{t-1} i + tc_0) \bmod 256 = (z_0 + (t-1)t/2 + tc_0) \bmod 256$$

For $z_0 = 0$ and $c_0 = 0$ we get:

$$z(10) = (\frac{9}{2} \cdot 10) \bmod 256 = 45$$

**Exercise 11.19:**

We use up-counting for $(s(t)+1)$ mod 10 function. Since the counter is module-16, it is necessary to detect when the counter state is 9 and load a 0. Similarly, the count-down feature is used for $(s(t)-1)$ mod 8. For this case the value 7 has to be loaded whenever the counter state is 0, and the value 0 is loaded whenever the count is 9 (since $(9-1) \bmod 8 = 0$). Consequently, if LOAD overrides the count signals, we get:

$$Count-up = \begin{cases} 1 & \textbf{if } (x=1) \\ 0 & \textbf{otherwise} \end{cases} \tag{11.4}$$

$$Count-down = \begin{cases} 1 & \textbf{if } (x=2) \\ 0 & \textbf{otherwise} \end{cases} \tag{11.5}$$

$$LOAD = \begin{cases} 1 & \textbf{if } ((x=1 \ \textbf{ or } x=2) \textbf{ and } s(t)=9) \ \textbf{ or } (x=2 \textbf{ and } s(t)=0) \\ 0 & \textbf{otherwise} \end{cases} \tag{11.6}$$

The value to be loaded is

$$i = \begin{cases} 0 & \textbf{if } s(t)=9 \\ 7 & \textbf{if } s(t)=0 \\ \text{don't care} & \textbf{otherwise} \end{cases} \tag{11.7}$$

For the binary implementation it is necessary to encode $x$ into binary variables. We choose a binary enconding such that $x = 2x_1 + x_0$. This results in

$$\begin{aligned} Count-up &= x_0 \\ Count-down &= x_1 \\ LOAD &= Q_3Q_0(x_1+x_0) + x_1Q_3'Q_2'Q_1'Q_0' \\ I_3 &= 0 \\ I_2 &= I_1 = I_0 = Q_3' \end{aligned}$$

The corresponding network is presented in Figure 11.13.

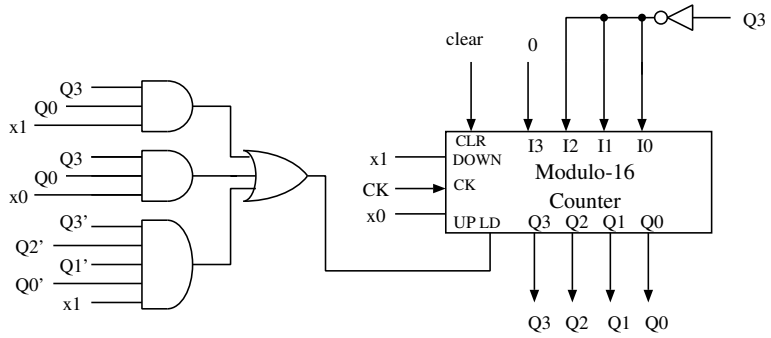



Figure 11.13: Exercise 11.19

**Exercise 11.20:**

The state diagram is shown in Figure 11.14. The state names represent even number of 1s (with an "e" subscript) and odd number of 1s (with an "o" subscript). From the diagram we obtain the following expressions for the counter control inputs:

$$\begin{aligned} CNT &= A_e x' + B_e x + C_o x + D_e x + A_o x' + B_o x + C_e x + D_o x \\ LOAD &= CNT' \end{aligned}$$

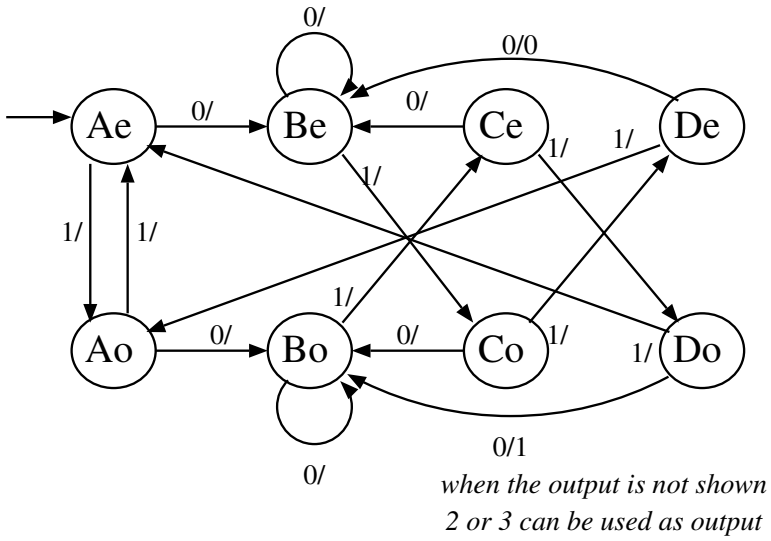



Figure 11.14: State diagram - Exercise 11.20

Consider the following state codes:

| State | code |
|---|---|
| $A_e$ | 0 |
| $B_e$ | 1 |
| $C_e$ | 6 |
| $D_e$ | 3 |
| $A_o$ | 4 |
| $B_o$ | 5 |
| $C_o$ | 2 |
| $D_o$ | 7 |

The expressions for the parallel inputs, outputs and CNT are obtained from Kmaps:

$$\begin{aligned} I_2 &= Q_2 Q_0 + Q_2' Q_0' = (Q_2 \oplus Q_0)' \\ I_1 &= 0 \\ I_0 &= Q_1 + Q_0 \\ z_0 &= Q_2 \\ z_1 &= x + Q_1' + Q_0' \\ CNT &= xQ_0 + xQ_1 + x'Q_1'Q_0' \end{aligned}$$

The network for this system is shown in Figure 11.15.

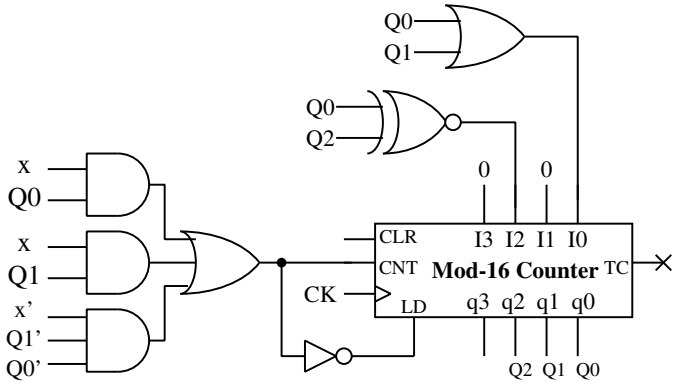



Figure 11.15: Network for Exercise 11.20

**Exercise 11.23:** Using two modulo-16 binary counters we implement cascade and parallel counters for the cases:

(a) a cascade implementation of a modulo-23 counter is shown in Figure 11.16. The state 22, $Q = 10110$ is detected and the counter is loaded with the value 0.

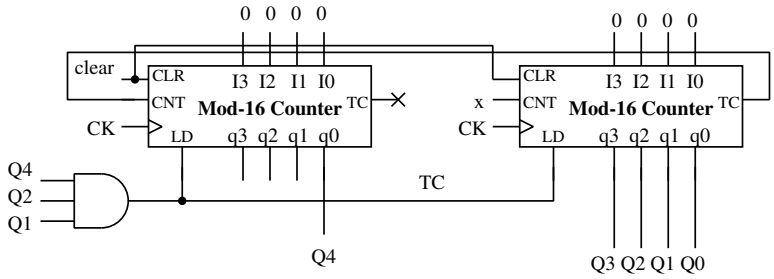



Figure 11.16: Cascade implementation of modulo-23 counter - Exercise 11.23(a)

The parallel implementation of the same counter uses a modulo-8 and a modulo-3 counter that together form a modulo-24 parallel counter. The counting sequence, considering $\underline{T} = (T_1 T_0)$ as the output of the modulo-3 counter and $\underline{E} = (E_2 E_1 E_0)$ as the output of the modulo-8 counter is given by the following table:

| state | $T_1 T_0$ | $E_2 E_1 E_0$ |
|---|---|---|
| 0 | 00 | 000 |
| 1 | 01 | 001 |
| 2 | 10 | 010 |
| 3 | 00 | 011 |
| 4 | 01 | 100 |
| 5 | 10 | 101 |
| 6 | 00 | 110 |
| 7 | 01 | 111 |
| 8 | 10 | 000 |
| 9 | 00 | 001 |
| 10 | 01 | 010 |
| 11 | 10 | 011 |
| 12 | 00 | 100 |
| 13 | 01 | 101 |
| 14 | 10 | 110 |
| 15 | 00 | 111 |
| 16 | 01 | 000 |
| 17 | 10 | 001 |
| 18 | 00 | 010 |
| 19 | 01 | 011 |
| 20 | 10 | 100 |
| 21 | 00 | 101 |
| 22 | 01 | 110 |
| 23 | 10 | 111 |

The state code is given by the two counters' output, $(T, E)$. For simplicity we represent the state code using decimal numbers. To modify this modulo-24 counter to a modulo-23 counter, state

22 is detected, which is represented by the code $(1, 6)$, and the state 0, coded as $(0, 0)$ is loaded, as shown in Figure 11.17.

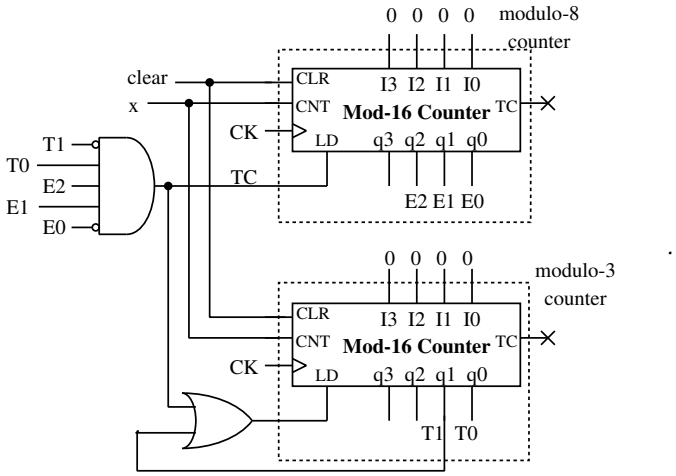



Figure 11.17: Parallel implementation of modulo-23 counter - Exercise 11.23(a)

(b) an 11-to-29 counter. A total of 19 states are needed. The cascade implementation of this counter is shown in Figure 11.18. Observe that state 29 (11101) is detected and generates a load signal. The value loaded corresponds to state 11 (1011).

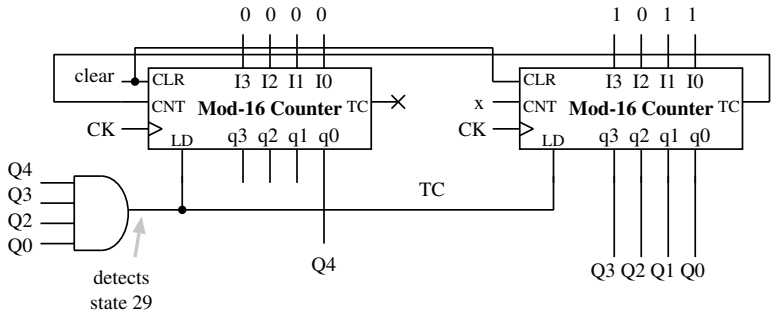



Figure 11.18: Cascade implementation of a 11-to-29 counter - Exercise 11.23(b)

The parallel implementation of this counter uses a modulo-30 parallel counter that is obtained combining a modulo-5 and a modulo-6 counter. The state sequence, assuming that the state is composed by $(E, T)$, where $\underline{E} = (E_2E_1E_0)$ is the state of the modulo-5 counter and $\underline{T} = (T_2T_1T_0)$ is the state code of the modulo-6 counter, is presented (in decimal) in the following table:

| state | $E$ | $T$ | state | $E$ | $T$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 15 | 0 | 3 |
| 1 | 1 | 1 | 16 | 1 | 4 |
| 2 | 2 | 2 | 17 | 2 | 5 |
| 3 | 3 | 3 | 18 | 3 | 0 |
| 4 | 4 | 4 | 19 | 4 | 1 |
| 5 | 0 | 5 | 20 | 0 | 2 |
| 6 | 1 | 0 | 21 | 1 | 3 |
| 7 | 2 | 1 | 22 | 2 | 4 |
| 8 | 3 | 2 | 23 | 3 | 5 |
| 9 | 4 | 3 | 24 | 4 | 0 |
| 10 | 0 | 4 | 25 | 0 | 1 |
| 11 | 1 | 5 | 26 | 1 | 2 |
| 12 | 2 | 0 | 27 | 2 | 3 |
| 13 | 3 | 1 | 28 | 3 | 4 |
| 14 | 4 | 2 | 29 | 4 | 5 |

Thus, to obtain the 11-to-29 counter we need to detect state 29 $(4,5)$ and load state 11 $(1,5)$. The network is shown in Figure 11.19.

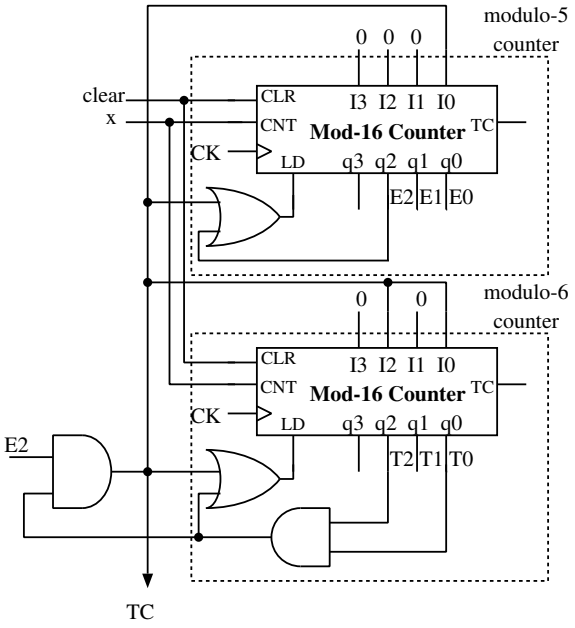



Figure 11.19: Parallel implementation of a 11-to-29 counter - Exercise 11.23(b)

(c) a frequency divider by 27

For the cascade implementation, the state (11010) is detected and the value 0 is loaded as the next state. The same load signal (TC) is the output of the frequency divider. The network is shown in Figure 11.20.

The parallel implementation makes use of a modulo-4 and a modulo-7 counter to obtain a modulo-28 counter. The state code is represented by two components $(E,T)$, where $\underline{E} = (E_1E_0)$ is the state code for the modulo-4 counter and $\underline{T} = (T_2T_1T_0)$ is the state code of the modulo-7 counter. When state 26 $(2,5)$ is reached, both counters are loaded with the initial state 0. The

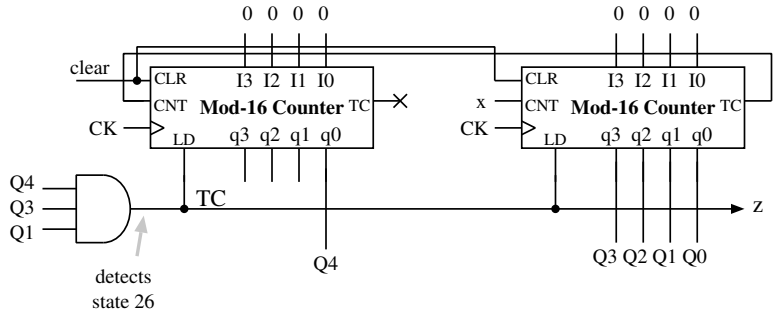



Figure 11.20: Cascade implementation of a frequency divider by 27 - Exercise 11.23(c)

network shown in Figure 11.21 shows the parallel implementation of this frequency divider. The circuit output corresponds to the load signal (TC).

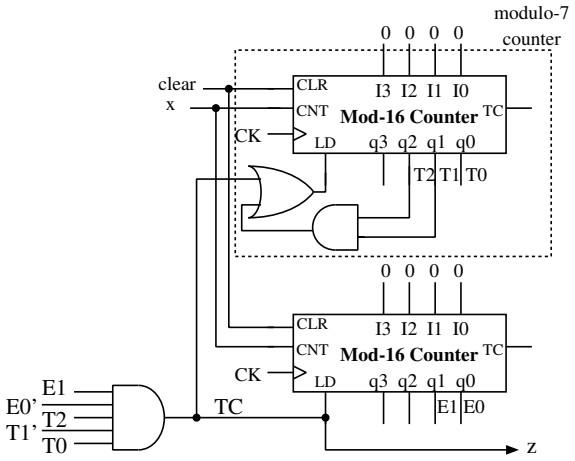



Figure 11.21: Parallel implementation of a frequency divider by 27 - Exercise 11.23(c)

**Exercise 11.25:**

The counter shown in Figure 11.37 of the text is a mod-24 parallel counter composed of a modulo-3 counter and a modulo-8 counter (twisted-tail counter). It has the following sequence of state values:

| modulo-3 $Q_1Q_0$ | modulo-8 $q_0q_1q_2q_3$ |
|---|---|
| 00 | 0000 |
| 01 | 1000 |
| 10 | 1100 |
| 00 | 1110 |
| 01 | 1111 |
| 10 | 0111 |
| 00 | 0011 |
| 01 | 0001 |
| 10 | 0000 |
| 00 | 1000 |
| 01 | 1100 |
| 10 | 1110 |
| 00 | 1111 |
| 01 | 0111 |
| 10 | 0011 |
| 00 | 0001 |
| 01 | 0000 |
| 10 | 1000 |
| 00 | 1100 |
| 01 | 1110 |
| 10 | 1111 |
| 00 | 0111 |
| 01 | 0011 |
| 10 | 0001 |

Observe that the output $z$ is 1 when $Q_1Q_0q_0q_1q_2q_3 = 100001$, that corresponds to the expression $z = Q_1Q_0'q_2'q_3$, and is equivalent to a Terminal Count (TC) signal in a binary counter.

(a) a binary modulo-24 autonomous counter using T flip flops and gates is shown in Figure 11.22. State 23 (10111) is detected by a network that implements the expression $S_{23} = Q_4Q_2Q_1Q_0$. Since the next state from (10111) must be (00000), we need $T_0 = T_1 = T_2 = T_4 = 1$ and $T_3 = 0$. The values of $T_0 = T_1 = T_2 = 0$ are already generated when the present state is (10111). The values $T_4 = 1$ and $T_3 = 0$ must be forced when $S_{23} = 1$. The implementation has fewer flip flops, more gates and more connections than the implementation in Figure 11.37 of the text.

(b) a twisted-tail modulo-24 autonomous counter using D flip flops and gates is shown in Figure 11.23. Twelve flip flops are needed, instead of the six flip-flops used in Figure 11.37 of the text. However, no gates are required and besides the clock line, all interconnections can be made very short. The TC condition is generated when the present state is (000000000001), and that is the only state when we have a 0 preceding a 1 at the rightmost flip flop.

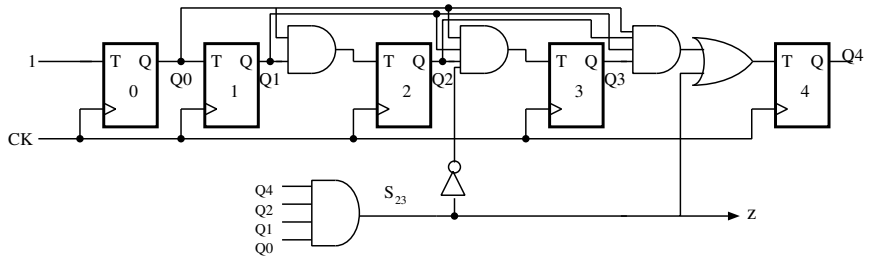



Figure 11.22: Modulo-24 autonomous counter - Exercise 11.25(a)

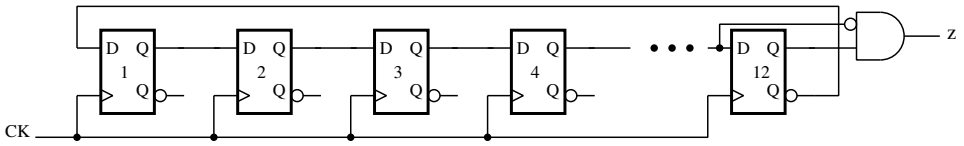



Figure 11.23: Twisted-tail modulo-24 autonomous counter - Exercise 11.25(b)

**Exercise 11.27:** The state diagram for the electronic lock is shown in Figure 11.24. Such a state diagram is adequately implemented using a modulo-16 counter and a 16-input multiplexer. In order to generate the required outputs, another counter is used to count the number of times the buttons were pressed. This counter generates a signal $END$ when the keys were pressed 12 times. We assume that the outputs of the push buttons that generate the inputs are 1s for only one clock

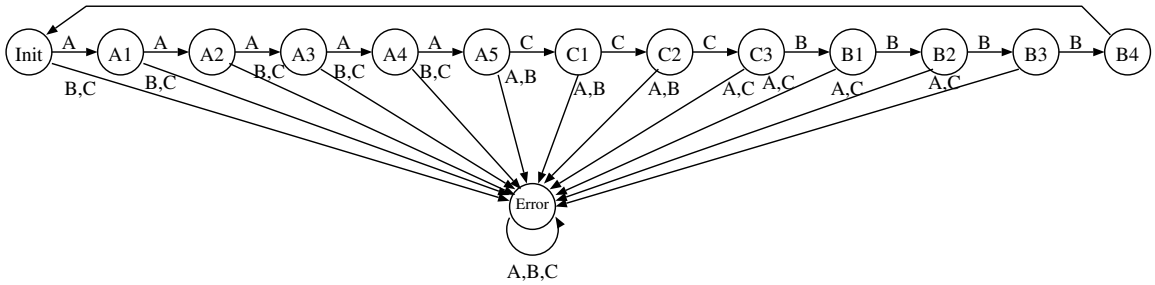



Figure 11.24: State Diagram for the electronic lock - Exercise 11.27

period, each time the user press them. A network for this task is shown in Figure 11.25 together with the network that implements the lock controller. When a correct sequence is inserted (state is B4) the output $z = 1$ is generated, and the controller goes back to the initial state. When a wrong sequence is inserted the $RED$ output is activated by the condition $B4' \cdot END$ ($END = 1$ when 12 characters were inserted). The output $GREEN$ is 1 when the controller is at state $INIT$. The $YELLOW$ is 1 while the sequence is being inserted, that means, the sequence started to be inserted and the $RED$ or $z$ output were not generated yet. When $z = 0$ the top counter goes to state 15 ($ERROR$), and the bottom one stays at state 12. Only a reset signal $R$ will remove the counters from these states, if the wrong sequence is inserted. When $z = 1$ both counters load the value 0 and they are ready to start another operation.

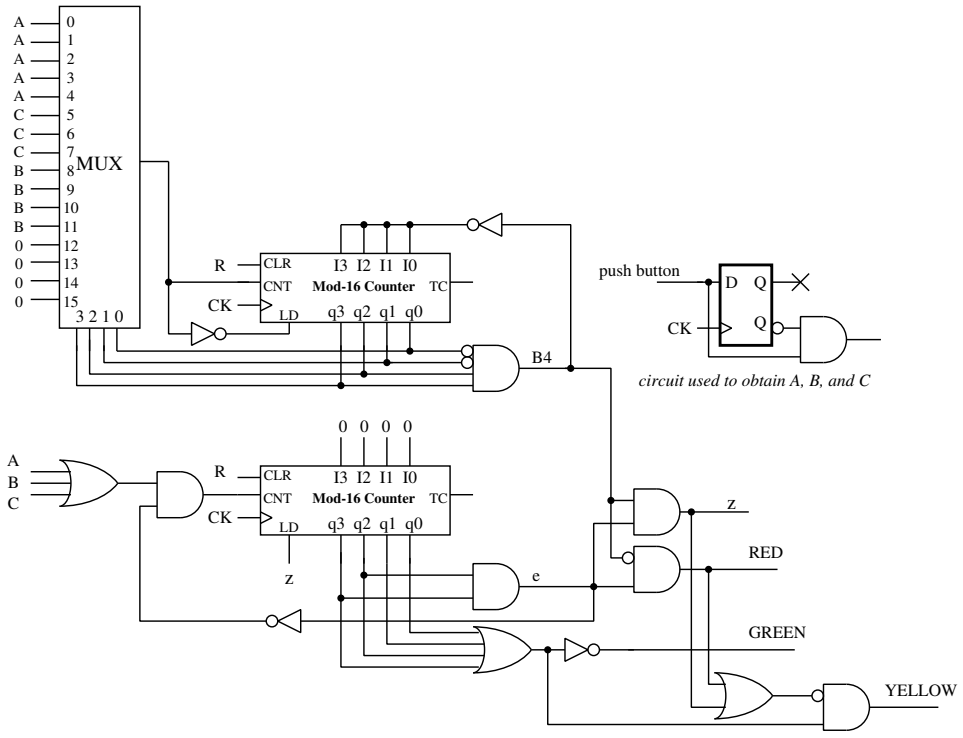



Figure 11.25: Sequential Lock network - Exercise 11.27