Chapter 12

Exercise 12.1:

The controller has 6 states, and we are going to use the one-flip-flop per state appoach for the design. The inputs are:

Input	condition
GO	
A	$DIST > 10$
B	$(DIST \leq 10)$ and $(COUNT = 3)$
C	$(DIST \leq 10)$ and $(COUNT \neq 3)$

Calling the inputs of the flip flops for each state as NS_i, we generate the following expressions:

$$\begin{aligned} NS_0 &= S_0.GO' \\ NS_1 &= S_0.GO + S_3 \\ NS_2 &= S_1.C \\ NS_3 &= S_2 + S_4 \\ NS_4 &= S_1.A \\ NS_5 &= S_1.B + S_5 \end{aligned}$$

These expressions are implemented in a PSA as shown in Figure **??**.

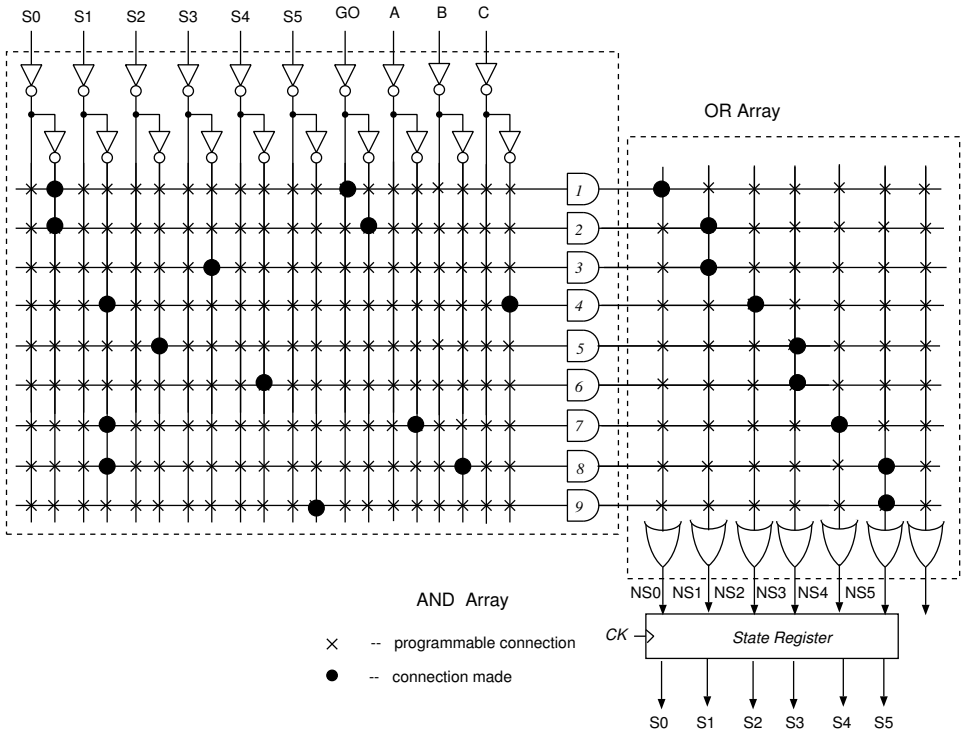

Figure 12.1: PSA implementation of a controller - Exercise 12.1

Exercise 12.3:

The number of address lines should be 4, since a BCD digit is the single input. The maximum value to be stored in the ROM is $9^2 = 81$, which requires $\lceil log_2 81 \rceil = 7$ bits to be represented in binary. Thus, at least a 16×7 ROM is required to implement the function.

Exercise 12.5:

For this implementation we need a $2^{12} \times 6$ ROM. Twelve address lines receive the input bits from a, b, and $c \in \{0, 1, ..., 15\}$. Six bits are required to represent $0 \leq s \leq 45$. The block diagram of the component is shown in Figure **??**.

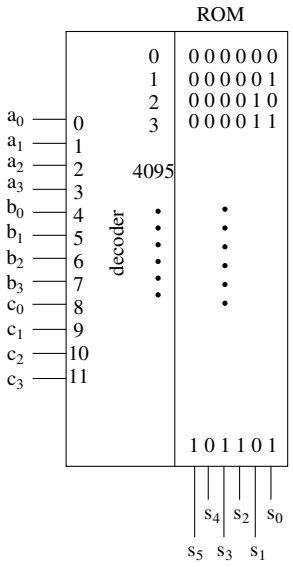

Figure 12.2: ROM implementation - 3-input 4-bit adder - Exercise 12.5

Exercise 12.7:

Part (a) - using one decoder and ROM modules of eight 4-bit words. Figure **??**

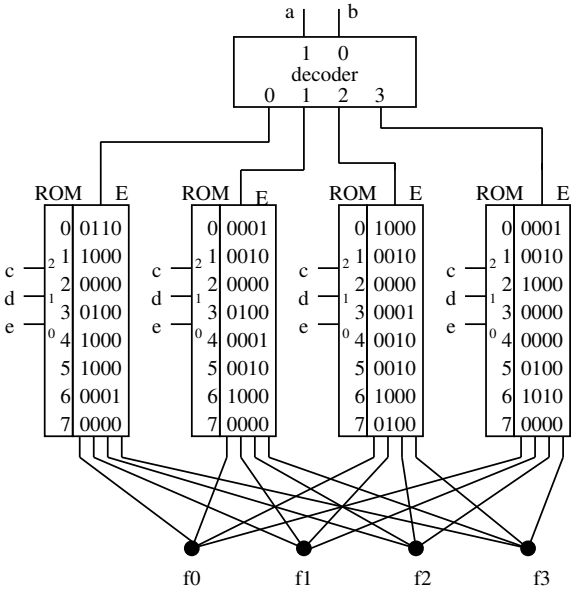

Figure 12.3: Implementation of switching function using one decoder and ROM- Ex. 12.7

$$COST = 4 \times (ROM) + DEC = 4 \times (ROM) + 4 \times (\text{AND-2}) + 2 \times (\text{NOT})$$

$$\#interconnections = 13$$

$$delay = \delta(ROM) + \delta(\text{AND-2}) + \delta(\text{NOT})$$

where δ represents the component delay.

Part (b) - using ROM modules and a multiplexer. Figure **??**.

$$COST = 4 \times (ROM) + MUX = 4 \times (ROM) + 16 \times (\text{AND-3}) + 4 \times (\text{OR-4}) + 2 \times (\text{NOT})$$

$$\#interconnections = 29$$

$$delay = \delta(ROM) + \delta(MUX) = \delta(ROM) + \delta(\text{AND-3}) + \delta(\text{OR-4})$$

Thus, the first design is better than the second one in all aspects.

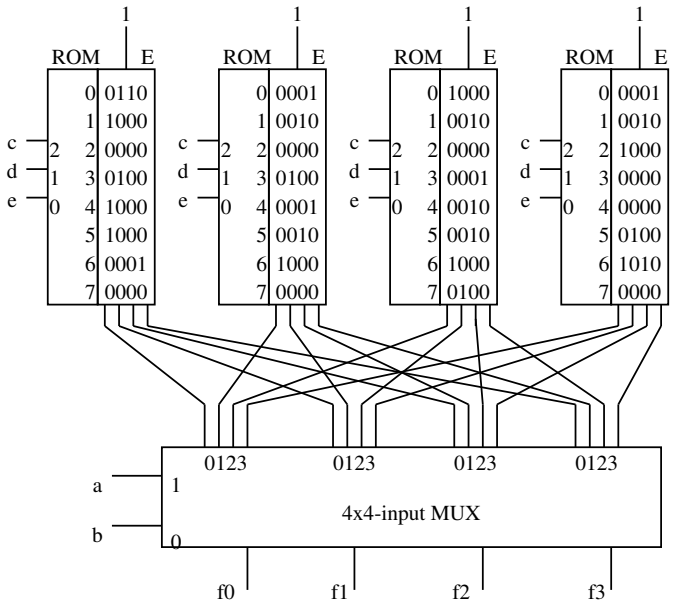

Figure 12.4: Implementation of switching function using ROM and MUX - Ex. 12.7

Exercise 12.9:

For the ROM implementation of the function we need 2^6 words of 2 bits. Thus, a complete switching function table should be implemented, with 64 entries.

For the PLA implementation, it is important to notice that $z_1 = 1$ when $a = b$, and $z_0 = 1$ when $a = (b - 1) \bmod 8$. The minterms used to generate z_1 are shown in the following table:

$a_2a_1a_0$	$b_2b_1b_0$	product terms to generate z_1
000	000	$a_2'a_1'a_0'b_2'b_1'b_0'$
001	001	$a_2'a_1'a_0b_2'b_1'b_0$
010	010	$a_2'a_1a_0'b_2'b_1b_0'$
011	011	$a_2'a_1a_0b_2'b_1b_0$
100	100	$a_2a_1'a_0'b_2b_1'b_0'$
101	101	$a_2a_1'a_0b_2b_1'b_0$
110	110	$a_2a_1a_0'b_2b_1b_0'$
111	111	$a_2a_1a_0b_2b_1b_0$

For PLAs what matters is the number of product terms. It is not possible to reduce the number of product terms shown in the table since both a and b change one bit from one row to another. Thus, two literals are different from one minterm to another, making impossible the combination of two minterms. The number of product terms is already minimal.

Similarly, for z_0 (the case $a = (b - 1) \bmod 8$):

$a_2a_1a_0$	$b_2b_1b_0$	product terms to generate z_0
000	111	$a_2'a_1'a_0'b_2b_1b_0$
001	000	$a_2'a_1'a_0b_2'b_1'b_0'$
010	001	$a_2'a_1a_0'b_2'b_1'b_0$
011	010	$a_2'a_1a_0b_2'b_1b_0'$
100	011	$a_2a_1'a_0'b_2'b_1b_0$
101	100	$a_2a_1'a_0b_2b_1'b_0'$
110	101	$a_2a_1a_0'b_2b_1'b_0$
111	110	$a_2a_1a_0b_2b_1b_0'$

where again the number of product terms is minimal.

Thus, the PLA would need to have 16 products, 6 inputs, and 2 outputs.

Exercise 12.11:
Size of state register and ROM for:

(a) Moore sequential system with 512 states, 3 inputs and 2 outputs. For 512 states, a minimum of 9 bits are required to represent each state. The ROM needs to have as many address lines as the number of state bits plus the number of inputs, which corresponds to a total of 12 bits. Since this is a Moore machine the total number of ROM bits is reduced by using separate ROMs to generate the next state and to generate the outputs. The ROM for the next state will have $2^{12} \times 9$ bits. The ROM to generate the output will be a $2^9 \times 2$ bits ROM. Thus, the implementation will require a $2^{12} \times 9$ ROM, a $2^9 \times 9$ ROM, a and a 9-bit register. Of course, the implementation can use only one ROM, resulting the same as for the Mealy case.

(b) For a Mealy model, the system will have only a single $2^{12} \times 11$ ROM, and a 9-bit register. This time the output values are stored with the next state bits.

(c) When the state transition depends only on one input, a multiplexer (in this case a network of multiplexer would be required, since the number of inputs is quite large). The multiplier is used to select the correct input among the possible system inputs, depending on the present state of the system. The output of the multiplexer is used as an address line for the ROM which will have a size of $2^{10} \times 9$ bits, and generates the next state bits. Since this is a Moore machine, another ROM is used to generate the outputs. Same way as done in part (a), a $2^9 \times 2$ ROM is required for the outputs. A 9-bit register is used to store the state values. A block diagram of the system is shown in Figure **??**.

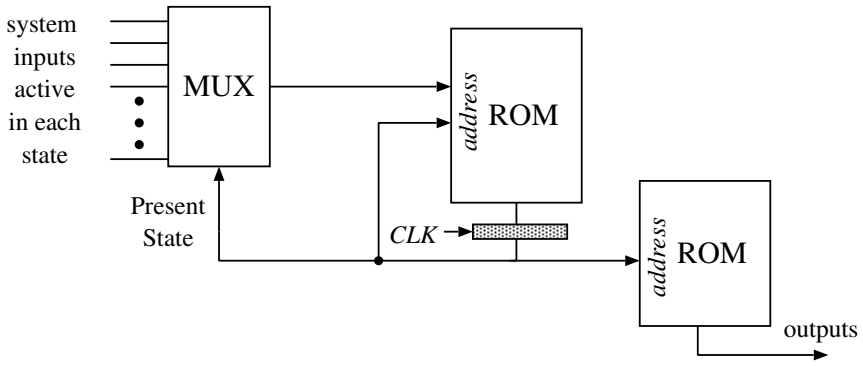

Figure 12.5: Block diagram for Exercise 12.11(c)

(d) For a Mealy system, with the same conditions of the one considered in part (c), a $2^{10} \times 11$ ROM, a network of muxes to select the input (same as in part (c)), and a 9-bit register are used. The ROM stores both the next state bits and the output values.

(e) Since the system in part (c) is implemented as a Moore machine, the output doesn't depend on the input values, and for this reason, this question doesn't make sense. If we apply this idea to the system in part (d), then the output should be generated by a separate ROM with 11 addressing lines and 2-bit words. The generation of the next state would be done by a $2^{10} \times 9$ ROM. The network of muxes to select the inputs would be used, and a 9-bit register would store the state bits.

Exercise 12.13:

Observe that each state has transitions to another state based on only one input among 3 possible inputs $\in \{a, b, c\}$. A multiplexer can be used to select the desired input, and the diagram can be modified to have only one input, let's call it x. The new transition table would be:

PS	Inputs	
	$x = 0$	$x = 1$
S0	S3/10	S1/01
S1	S1/00	S2/01
S2	S3/00	S4/01
S3	S1/00	S0/00
S4	S4/00	S3/11
	NS,$z_1 z_0$	

Assume that the state S_i is represented by the vector $i = (y_2, y_1, y_0)$. The network that implements the state diagram is given in Figure **??**.

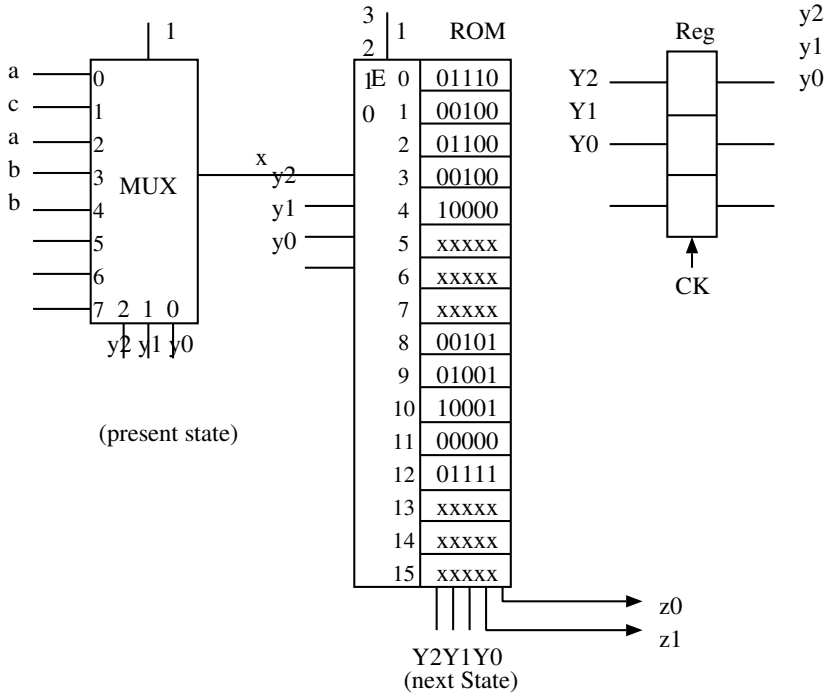

Figure 12.6: Network for Exercise 12.13

Exercise 12.15:

For this design we use a ROM that receives as input lines the values of the present state, and 2 input bits. In fact, there are 3 conditions that are used as inputs, but making use the the multiplexer we are able to select among two inputs: GO and $DIST > 10$. The third input is inserted directly as an address signal to the ROM. The ROM contains the information on the next state and the outputs ($CLEAR$,$CHECK$,$TURNLEFT90$,$COUNTUP$,$MOVE$,$STOP$). Thus 9 bits are required per ROM word. The design is shown in Figure **??**.

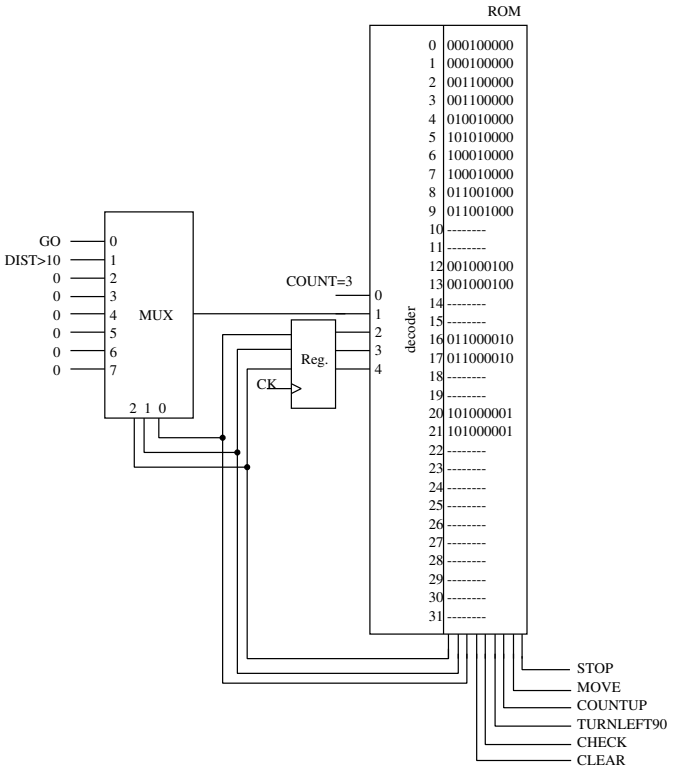

Figure 12.7: Circuit for Exercise 12.15

Exercise 12.17:

The ROM implementation of the system is shown in Figure **??**. The complete contents of the ROM that generates (z_{22}, z_{21}, z_{20}) (using decimal notation) is given in the next table:

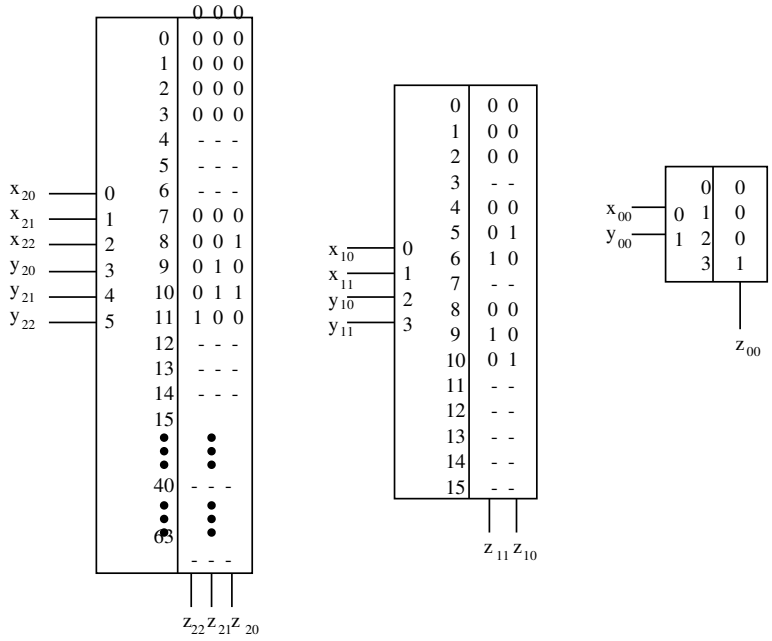

Figure 12.8: ROM implementation of Exercise 12.17

y_2	x_2	z_2	y_2	x_2	z_2	y_2	x_2	z_2	y_2	x_2	z_2
0	0	0	2	0	0	4	0	0	6	0	dc
0	1	0	2	1	2	4	1	4	6	1	dc
0	2	0	2	2	4	4	2	3	6	2	dc
0	3	0	2	3	1	4	3	2	6	3	dc
0	4	0	2	4	3	4	4	1	6	4	dc
0	5	dc	2	5	dc	4	5	dc	6	5	dc
0	6	dc	2	6	dc	4	6	dc	6	6	dc
0	7	dc	2	7	dc	4	7	dc	6	7	dc
1	0	0	3	0	0	5	0	dc	7	0	dc
1	1	1	3	1	3	5	1	dc	7	1	dc
1	2	2	3	2	1	5	2	dc	7	2	dc
1	3	3	3	3	4	5	3	dc	7	3	dc
1	4	4	3	4	2	5	4	dc	7	4	dc
1	5	dc	3	5	dc	5	5	dc	7	5	dc
1	6	dc	3	6	dc	5	6	dc	7	6	dc
1	7	dc	3	7	dc	5	7	dc	7	7	dc

The PLA implementation of the system needs the representation of the output functions as a sum of product terms. The switching expressions in sum-of-products form for the various outputs

are:

$$\begin{aligned}
z_{22}(y_{22}, y_{21}, y_{20}, x_{22}, x_{21}, x_{20}) &= m_2(12, 18, 27, 33) \\
z_{21}(y_{22}, y_{21}, y_{20}, x_{22}, x_{21}, x_{20}) &= m_2(10, 11, 17, 20, 25, 28, 34, 35) \\
z_{20}(y_{22}, y_{21}, y_{20}, x_{22}, x_{21}, x_{20}) &= m_2(9, 11, 19, 20, 25, 26, 34, 36) \\
z_{11}(y_{11}, y_{10}, x_{11}, x_{10}) &= m_1(6, 9) \\
z_{10}(y_{11}, y_{10}, x_{11}, x_{10}) &= m_1(5, 10) \\
z_{00}(y_{00}, x_{00}) &= x_{00}y_{00} = m_0(3)
\end{aligned}$$

Since the outputs z_{2i} depend on the same set of inputs, they should be mapped to the same PLA. If more inputs and products are available in the component, functions z_{1i} and z_{00} could also be implemented in the same PLA. A total of 21 products were listed in the expressions above, however, it is possible to reduce this number of products to 20. Using *ESPRESSO* we were able to reduce the number of products to generate z_2 from 16 to 15 (the others cannot be reduced). One possible solution is to combine $m_2(9)$ and $m_2(25)$ on the generation of z_{20} and obtain the single product term $y'_{22}y_{20}x'_{22}x'_{21}x_{20}$. ESPRESSO also reduces the number of literals in each product term, however, this feature is not important for PLA implementation.