Chapter 10

Exercise 10.1 Comparison of FA implementations

(a) using a two-level network (Fig. 10.3(a) of the textbook) the critical path for the circuit has 3 gates. The path connects the input x_i (or y_i or c_i) and output z_i. The propagation delay of this path is obtained as:

$$\begin{aligned}
T_{pHL}(x_i \rightarrow z_i) &= t_{pHL}(NOT) + t_{pLH}(NAND-3) + t_{pHL}(NAND-4)\\
T_{pHL}(x_i \rightarrow z_i) &= 0.05 + 0.017 \times 2 + 0.07 + 0.038 \times 1 + 0.12 + 0.051L\\
T_{pHL}(x_i \rightarrow z_i) &= 0.312 + 0.051L\\
T_{pLH}(x_i \rightarrow z_i) &= t_{pLH}(NOT) + t_{pHL}(NAND-3) + t_{pLH}(NAND-4)\\
T_{pLH}(x_i \rightarrow z_i) &= 0.02 + 0.038 \times 2 + 0.09 + 0.039 \times 1 + 0.10 + 0.037L\\
T_{pLH}(x_i \rightarrow z_i) &= 0.325 + 0.037L
\end{aligned}$$

However, the delay of the path connecting the carry input to the carry output is also important for carry ripple adders. This path has a worst case delay:

$$\begin{aligned}
T_{pHL}(c_i \rightarrow c_{i+1}) &= t_{pLH}(NAND-2) + t_{pHL}(NAND-3)\\
T_{pHL}(c_i \rightarrow c_{i+1}) &= 0.05 + 0.038 + 0.09 + 0.039L\\
T_{pHL}(c_i \rightarrow c_{i+1}) &= 0.18 + 0.039L\\
T_{pLH}(c_i \rightarrow c_{i+1}) &= t_{pHL}(NAND-2) + t_{pLH}(NAND-3)\\
T_{pLH}(c_i \rightarrow c_{i+1}) &= 0.08 + 0.027 + 0.07 + 0.038L\\
T_{pLH}(c_i \rightarrow c_{i+1}) &= 0.18 + 0.038L
\end{aligned}$$

The number of equivalent gates is:

gate type	number of gates	equivalent gates
NOT	3	3× 1 = 3
NAND-3	5	5× 2=10
NAND-2	3	3× 1=3
NAND-4	1	1× 2 = 2
Total		18

(b) using HA (Fig. 10.3(b) of the textbook)

The critical path for this circuit has 3 gates: XOR-2, AND-2 and OR-2. The path connects the input x_i (or y_i) and output c_{i+1}. The propagation delay of this path is obtained as:

$$\begin{aligned}
T_{pHL}(x_i \rightarrow c_{i+1}) &= t_{pHL}(XOR-2) + t_{pHL}(AND-2) + t_{pHL}(OR-2)\\
T_{pHL}(x_i \rightarrow c_{i+1}) &= 0.3 + 0.021 \times 3 + 0.16 + 0.017 + 0.2 + 0.019L\\
T_{pHL}(x_i \rightarrow c_{i+1}) &= 0.74 + 0.019L\\
T_{pLH}(x_i \rightarrow c_{i+1}) &= t_{pLH}(XOR-2) + t_{pLH}(AND-2) + t_{pLH}(OR-2)\\
T_{pLH}(x_i \rightarrow c_{i+1}) &= 0.3 + 0.036 \times 3 + 0.15 + 0.037 + 0.12 + 0.037L\\
T_{pLH}(x_i \rightarrow c_{i+1}) &= 0.72 + 0.037L
\end{aligned}$$

The number of equivalent gates is:

gate type	number of gates	equivalent gates
XOR-2	2	2× 3 = 6
AND-2	2	2× 2=4
OR-2	1	1× 2=2
Total		12

(c) using XOR and NAND gates (Fig. 10.3(c) of the textbook)

As the NAND gates have smaller propagation delays than XOR gates, the critical path for this circuit is through the 2 XOR gates. The path connects the input x_i (or y_i) and output z_i. Observe that when the first XOR gate is connected to the input with higher load factor of the second XOR gate, the delay of the second XOR gate is reduced. So, two cases must be considered. The propagation delay of this path is obtained as:

$$\begin{aligned}
T_{pLH}(x_i \rightarrow z_i) &= t_{pLH}(XOR-2) + t_{pLH}(XOR-2) \\
T_{pLH}(x_i \rightarrow z_i) &= max(0.3 + 0.036 \times 3 + 0.16 + 0.036L, 0.3 + 0.036 \times 2.1 + 0.3 + 0.036L) \\
T_{pLH}(x_i \rightarrow z_i) &= max(0.568 + 0.036L, 0.6756 + 0.036L) \\
T_{pLH}(x_i \rightarrow z_i) &= 0.68 + 0.036L \\
T_{pHL}(x_i \rightarrow z_i) &= t_{pLH}(XOR-2) + t_{pHL}(XOR-2) \\
T_{pHL}(x_i \rightarrow z_i) &= max(0.3 + 0.036 \times 3 + 0.15 + 0.020L, 0.3 + 0.036 \times 2.1 + 0.3 + 0.021L) \\
T_{pHL}(x_i \rightarrow z_i) &= max(0.558 + 0.02L, 0.6756 + 0.021L) \\
T_{pHL}(x_i \rightarrow z_i) &= 0.68 + 0.021L
\end{aligned}$$

The carry output propagation delay is:

$$\begin{aligned}
T_{pLH}(c_i \rightarrow c_{i+1}) &= t_{pHL}(NAND-2) + t_{pLH}(NAND-2) \\
T_{pLH}(c_i \rightarrow c_{i+1}) &= 0.08 + 0.027 + 0.05 + 0.038 \times L \\
T_{pLH}(c_i \rightarrow c_{i+1}) &= 0.16 + 0.038L \\
T_{pHL}(c_i \rightarrow c_{i+1}) &= t_{pLH}(NAND-2) + t_{pHL}(NAND-2) \\
T_{pHL}(c_i \rightarrow c_{i+1}) &= 0.05 + 0.038 + 0.08 + 0.027L \\
T_{pHL}(c_i \rightarrow c_{i+1}) &= 0.17 + 0.027L
\end{aligned}$$

The number of equivalent gates is:

gate type	number of gates	equivalent gates
XOR-2	2	2× 3 = 6
AND-2	3	3× 1=3
Total		9

This last implementation is the one with the least number of gates. It also has less propagation delay for the carry than case (a).

Exercise 10.3 The BCD to Excess-3 converter using 4-bit binary adder is shown in Figure **??**.

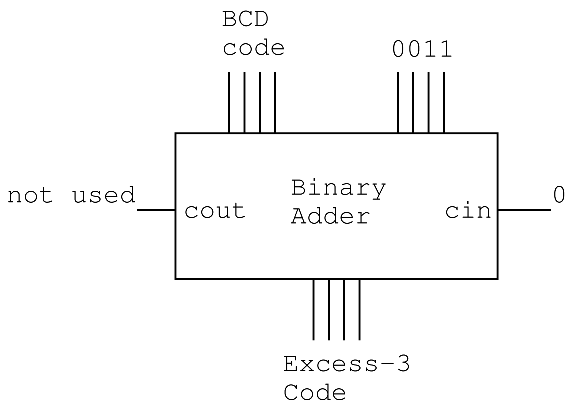


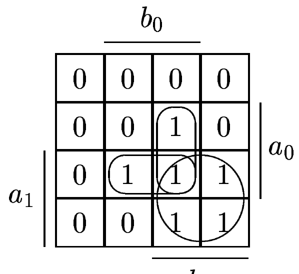
Figure 10.1: BCD to Excess-3 converter - Exercise 10.3

Exercise 10.7
Input: $A, B \in \{0,1,2,3\}$ and $C_{in} \in \{0,1\}$
Output: $Z \in \{0,1,2,3\}$ and $C_{out} \in \{0,1\}$
Function:

$$\begin{aligned} Z &= (A+B+C_{in}) \bmod 4 \\ C_{out} &= \begin{cases} 1 & \text{if } (A+B+C_{in}) \geq 4 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

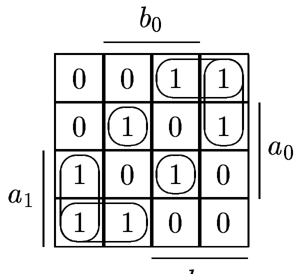
A function table is shown in Table **??**.

The Kmaps for the cases $c_{in} = 0$ and $c_{in} = 1$ are shown next:
$c_{in} = 0$



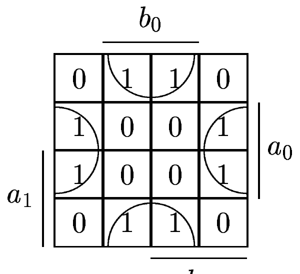
0	0	0	0
0	0	1	0
0	1	1	1
0	0	1	1

C_{out}:



0	0	1	1
0	1	0	1
1	0	1	0
1	1	0	0

z_1:



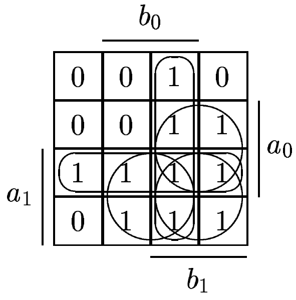
0	1	1	0
1	0	0	1
1	0	0	1
0	1	1	0

z_0:

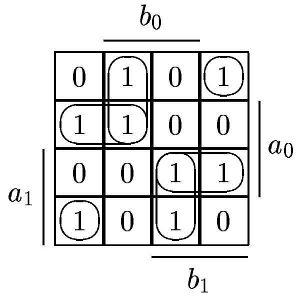
	Inputs					Outputs		
i	C_{in}	a_1	$a0$	b_1	b_0	C_{out}	z_1	z_0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	1
2	0	0	0	1	0	0	1	0
3	0	0	0	1	1	0	1	1
4	0	0	1	0	0	0	0	1
5	0	0	1	0	1	0	1	0
6	0	0	1	1	0	0	1	1
7	0	0	1	1	1	1	0	0
8	0	1	0	0	0	0	1	0
9	0	1	0	0	1	0	1	1
10	0	1	0	1	0	1	0	0
11	0	1	0	1	1	1	0	1
12	0	1	1	0	0	0	1	1
13	0	1	1	0	1	1	0	0
14	0	1	1	1	0	1	0	1
15	0	1	1	1	1	1	1	0
16	1	0	0	0	0	0	0	1
17	1	0	0	0	1	0	1	0
18	1	0	0	1	0	0	1	1
19	1	0	0	1	1	1	0	0
20	1	0	1	0	0	0	1	0
21	1	0	1	0	1	0	1	1
22	1	0	1	1	0	1	0	0
23	1	0	1	1	1	1	0	1
24	1	1	0	0	0	0	1	1
25	1	1	0	0	1	1	0	0
26	1	1	0	1	0	1	0	1
27	1	1	0	1	1	1	1	0
28	1	1	1	0	0	1	0	0
29	1	1	1	0	1	1	0	1
30	1	1	1	1	0	1	1	0
31	1	1	1	1	1	1	1	1

Table 10.1: Function table of a 2-bit adder

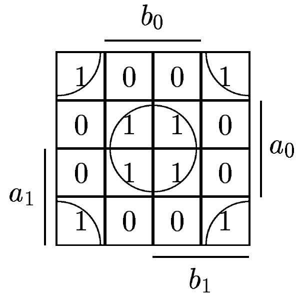
$c_{in} = 1$



C_{out}:



z_1:



z_0:

The following logic expressions are obtained from the K-maps.

$$\begin{aligned} C_{out} &= (a_1b_1 + a_1a_0b_0 + a_0b_1b_0)c_{in}' + (a_1b_1 + a_0b_1 + a_1b_0 + b_1b_0 + a_1a_0)c_{in} \\ z_1 &= (b_1b_0'a_1' + a_1'a_0'b_1 + a_1a_0b_1b_0 + a_1'a_0b_1'b_0 + a_1a_0'b_1' + a_1b_1'b_0')c_{in}' \\ &\quad +(a_1'a_0'b_1b_0' + a_1a_0'b_1'b_0' + a_1b_1b_0 + a_1a_0b_1 + a_1'a_0b_1' + a_1'b_1'b_0)c_{in} \\ z_0 &= (a_0'b_0 + a_0b_0')c_{in}' + (a_0b_0 + a_0'b_0')c_{in} \end{aligned}$$

Using boolean algebra we reduce the expressions to:

$$\begin{aligned} C_{out} &= a_1b_1 + a_0b_1b_0 + a_1a_0b_0 \\ &\quad + c_{in}b_1b_0 + c_{in}a_0b_1 + c_{in}a_1a_0 + c_{in}a_1b_0 \\ z_1 &= a_1'a_0'b_1b_0' + a_1'a_0b_1'b_0 + a_1a_0b_1b_0 + a_1a_0'b_1'b_0' \\ &\quad + c_{in}a_1'b_1'b_0 + c_{in}a_1'a_0b_1' + c_{in}a_1a_0b_1 + c_{in}a_1b_1b_0 \\ &\quad + c_{in}'a_1'a_0'b_1 + c_{in}'a_1'b_1b_0' + c_{in}'a_1b_1'b_0' + c_{in}'a_1a_0'b_1' \\ z_0 &= c_{in}a_0'b_0' + c_{in}a_0b_0 + c_{in}'a_0b_0' + c_{in}'a_0'b_0 \end{aligned}$$

From the equations, we count 26 NAND gates, with a maximum fan-in of 12 inputs. We assume that NOT gates are used to obtain the complement of the input variables, so 5 NOT gates are also used in the network. The network has one level of NOT gates, and two levels of NAND gates. Internally, all NAND gates in the intermediate level are connected to only one input of the next level, all loads are 1. The loads of the input signals and their complements (generated by the NOT gates) are:

Signal	Load
a_1	11
a_1'	6
b_1	11
b_1'	6
a_0	11
a_0'	6
b_0	11
b_0'	6
c	11
c'	6

Exercise 10.11:
Fixed-point number representation: $(x_6x_5x_4.x_3x_2x_1x_0)$
(a) Sign-and-magnitude: the most significant bit corresponds to the sign

$$\begin{aligned} x_{max} &= 011.1111 \rightarrow \frac{2^6-1}{2^4} = 3.9375 \\ x_{min} &= 111.1111 \rightarrow \frac{-(2^6-1)}{2^4} = -3.9375 \end{aligned}$$

(b) Two's Complement

$$\begin{aligned} x_{max} &= 011.1111 \rightarrow \frac{2^6-1}{2^4} = 3.9375 \\ x_{min} &= 100.0000 \rightarrow \frac{-2^7+2^6}{2^4} = -4.0 \end{aligned}$$

(c) One's Complement

$$\begin{aligned} x_{max} &= 011.1111 \rightarrow \frac{2^6-1}{2^4} = 3.9375 \\ x_{min} &= 100.0000 \rightarrow \frac{-(2^7-1)+2^6}{2^4} = -3.9375 \end{aligned}$$

Exercise 10.13:

(a) to show that the addition in one's complement can be performed by two steps let us consider two numbers x and y represented by $x_R = x \bmod C$ and $y_R = y \bmod C$, where $C = 2^n - 1$. We want to obtain $s_R = w_R \bmod C$, where $w_R = x_R + y_R$ is the result of the addition, represented by the vector $\underline{w_R} = \{w_n, w_{n-1}, \ldots, w_1, w_0)$. The most significant bit of w_R is the carry out bit of the $n-$bit addition performed with x_R and y_R. Since $x_R, y_R < C$, $w_R < 2C$. We must consider the following 3 cases:

1. If $w_R < C$ then $w_R \bmod C = w_R$ and $w_n = 0$

2. If $w_R = C$ then $w_R \bmod C = 0$ and $w_n = 0$

3. If $2C > w_R > C$ then $w_R \bmod C = w_R - C = w_R - 2^n + 1$ and $w_n = 1$

Consequently, if $w_n = 0$, the result is equal to w_R, and if $w_n = 1$ the result is obtained discarding w_n (subtracting 2^n) and adding 1. Note that case (2) produces a result vector $(1, 1, \ldots, 1)$, which is correct since this is another representation of 0 in the one's complement system.

(b) To verify that the operation of the adder with the carry output connected to the carry-in is combinational we consider the following cases:

- there is a combination 00 or 11 for one particular bit position. In that case the carry chain is broken by this combination. No active loop.
- there is no combination 00 or 11, that means, all bit positions have the combination 01. In that case, there is an active loop. To have a stable result all carries have to be reset to zero before the operation starts.

Figure **??** shows examples of both cases.

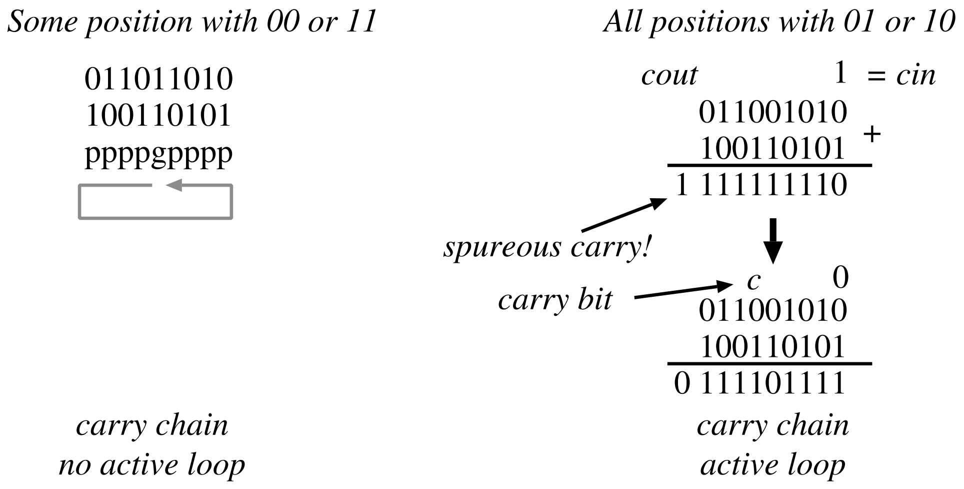


Figure 10.2: Example of carry chains for One's complement adder - Exercise 10.13 (part b)

Exercise 10.15: Consider the following table for the most significant bits of the operands a_{n-1} and b_{n-1} and the sum bit s_{n-1}, during the addition of n-bit two's complement operands a and b.:

a_{n-1}	b_{n-1}	s_{n-1}	c_n	c_{n-1}	Ovf
0	0	0	0	0	No
0	0	1	1	0	Yes
0	1	0	1	1	No
0	1	1	0	0	No
1	0	0	1	1	No
1	0	1	0	0	No
1	1	0	0	1	Yes
1	1	1	1	1	No

From the table one can see that the overflow cases are related to the cases when $c_n \oplus c_{n-1} = 1$.

For the one's complement system the test doesn't work properly in the case of adding $(-0) + (-2^{n-1} + 1)$, which corresponds to the addition of the vectors $(111\ldots11)$ and $(100\ldots00)$. If the addition is performed in two stages (no end-around-carry), the first stage will generate $c_n = 1$ and $c_{n-1} = 0$, detecting an overflow condition that does not exist.

Exercise 10.17: A 4-bit ALU for ADD and NAND operations is shown in Figure **??**. A carry-ripple adder was implemented. By the use of a multiplexer, the output of the adder, or the NAND gate is selected as circuit output.

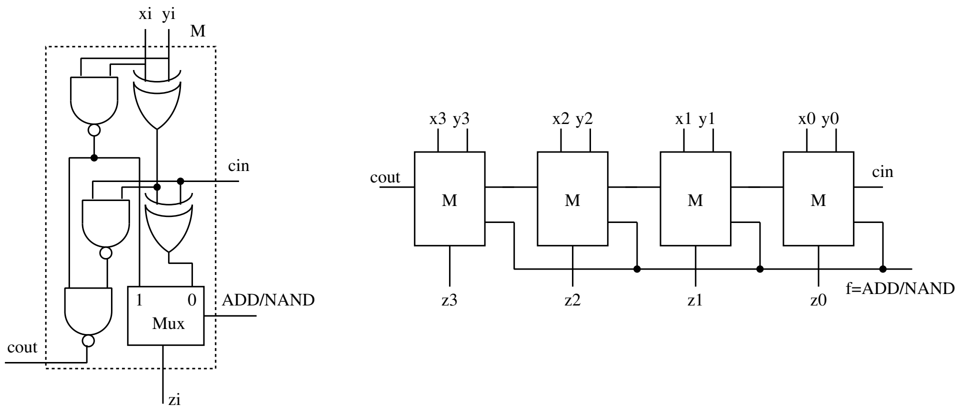


Figure 10.3: 4-bit ALU for ADD/NAND

Exercise 10.19: Figure **??** shows the iterative structure to be designed. Each cell has one bit of each of the operands (a_i and b_i), one carry-in (CIN), and one carry-out ($COUT$). These carries have three values: Equal, Greater, and Smaller. The comparator output is obtained from the carry-out of the last cell. The high-level description of the cell is:

$$COUT = \begin{cases} CIN & \text{if } a_i = b_i \\ GREATER & \text{if } ((a_i > b_i) \text{ and } CIN = EQUAL) \text{ or } (CIN = GREATER) \\ SMALLER & \text{if } ((a_i < b_i) \text{ and } CIN = EQUAL) \text{ or } (CIN = SMALLER) \end{cases}$$

with the initial condition $CIN = EQUAL$ (applied to the leftmost cell).

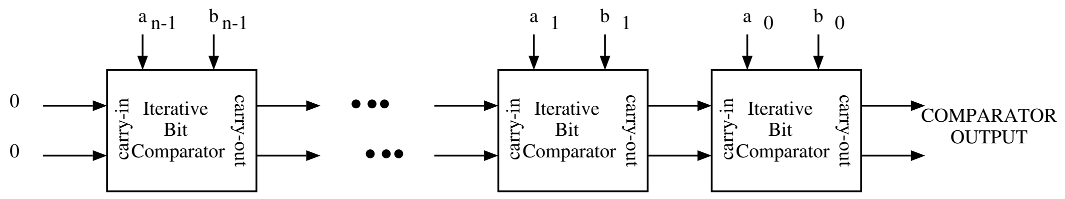


Figure 10.4: Iterative network to compare two numbers - Exercise 10.19

Considering the following encoding:

Condition	Code
EQUAL	00
SMALLER	01
GREATER	10

the implementation of each module is done as a combinational circuit that has the following switching function table:

$a_ib_iCIN_1CIN_0$	$COUT_1COUT_0$
0000	00
0001	01
0010	10
0011	–
0100	01
0101	01
0110	10
0111	–
1000	10
1001	01
1010	10
1011	–
1100	–
1101	–
1110	–
1111	–

Observe that the code 11 is a don't care condition.
Based on K-maps for each $COUT_1$ and $COUT_0$ we get the expressions:

$$\begin{aligned} COUT_1 &= CIN_1 + a_i CIN_0' \\ COUT_0 &= CIN_0 + b_i CIN_1' \end{aligned}$$

These expressions are easily implemented by gate networks.

Exercise 10.23: The values of the outputs of each module in the array multiplier is shown in Figure **??**.

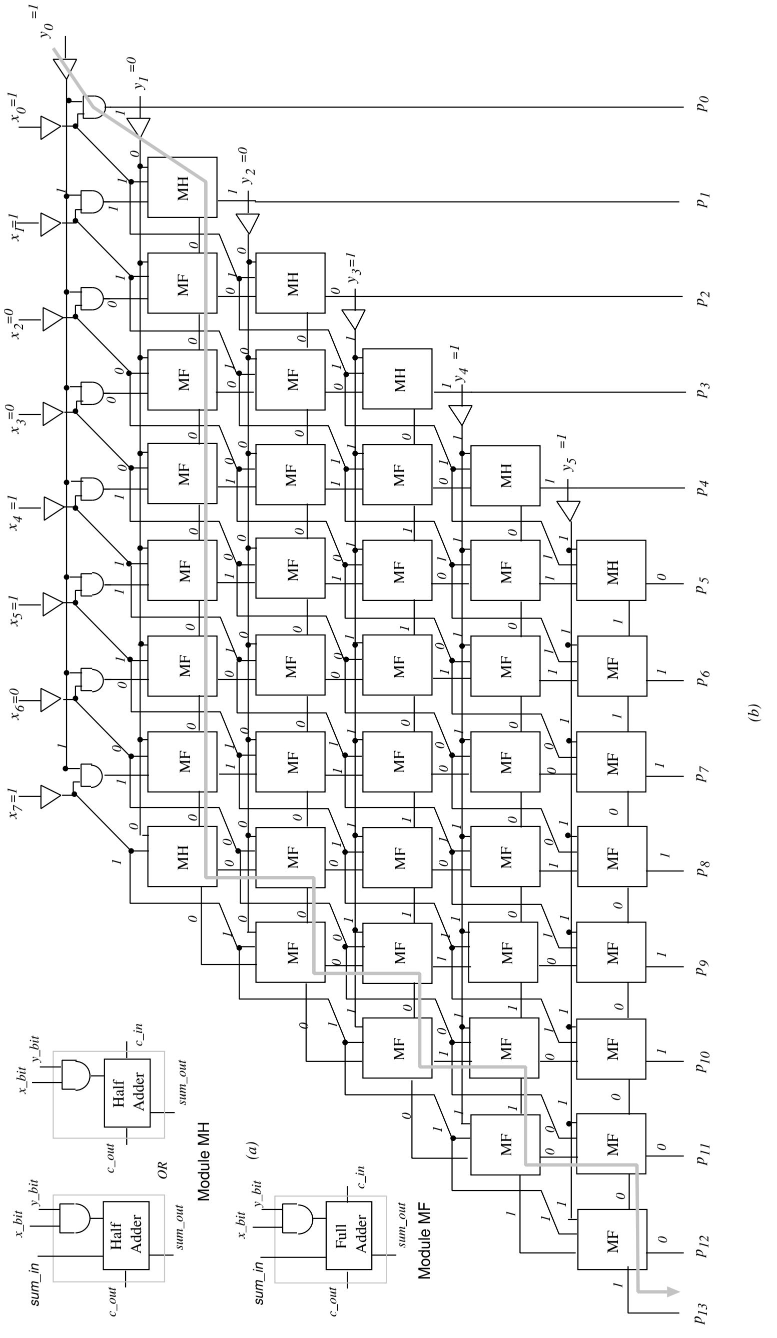


Figure 10.5: Exercise 10.23

Exercise 10.24 The implementation of the 8×4-bit multiplier is shown in Figure **??**.

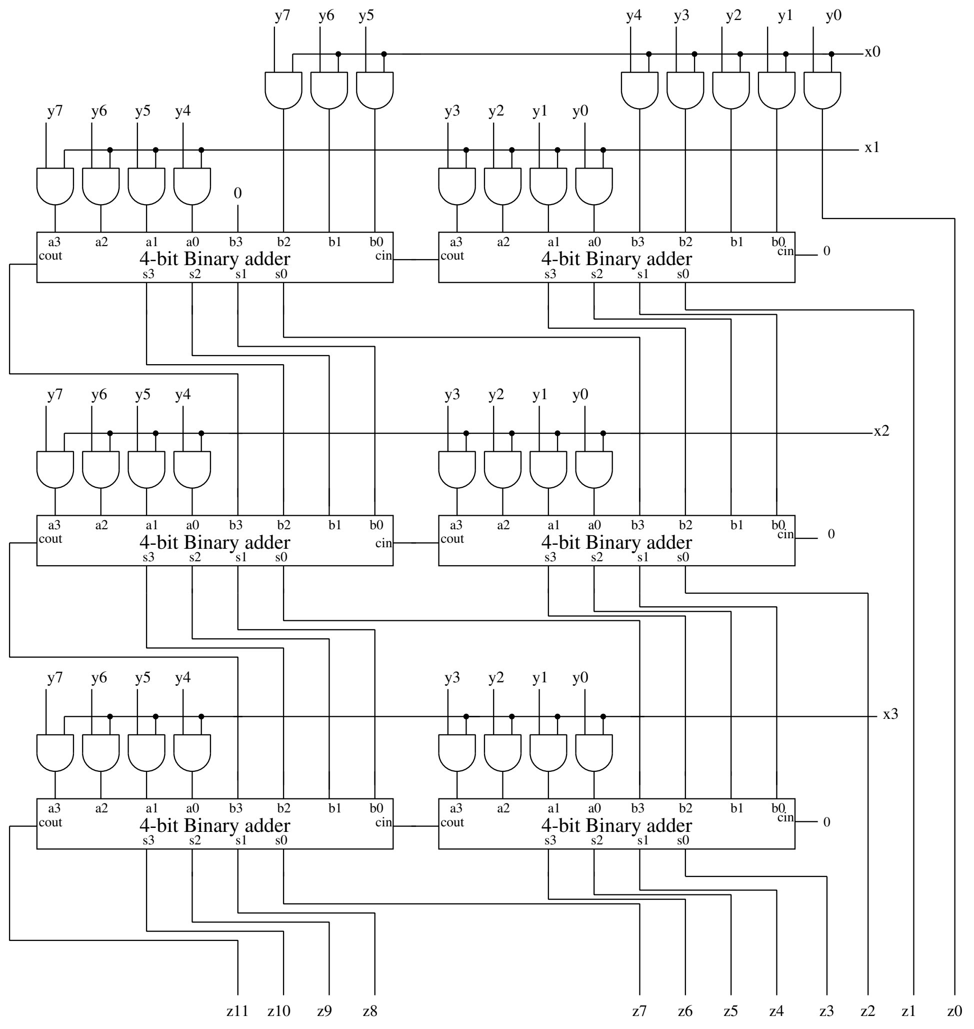


Figure 10.6: 8×4-bit multiplier - Exercise 10.24