فصل چهارم

منطق ترکیبی

دیاگرام بلوکی یک مدار ترکیبی (Combinational)

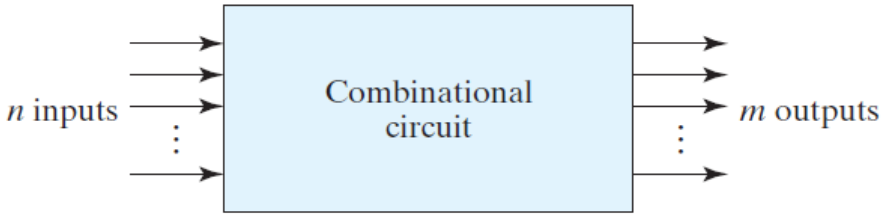


FIGURE 4.1
Block diagram of combinational circuit

در یک مدار ترکیبی، خروجی در هر لحظه فقط به مقادیر ورودی در همان لحظه بستگی دارد؛ بنابراین، یک راه برای توصیف یک مدار ترکیبی، **استفاده از جدول صحت** (Truth table) است. همچنین، چون خروجی دیگر به مقادیر قبلیِ خود وابسته نیست پس یک **مدار ترکیبی فاقد عناصر حافظه یا مسیر(های) فیدبکی** (از خروجی به ورودی) است.

تحلیل مدار ترکیبی

منظور از تحلیل یک مدار ترکیبی چیست؟ یعنی این که تابعی که مدار آن را پیاده‌سازی می‌کند، معین نماییم. برای این کار می‌توانیم تابع بولی خروجی مدار را بر حسب ورودی‌ها به دست آورده و حتی جدول صحت مدار را بنویسیم.

مثال:

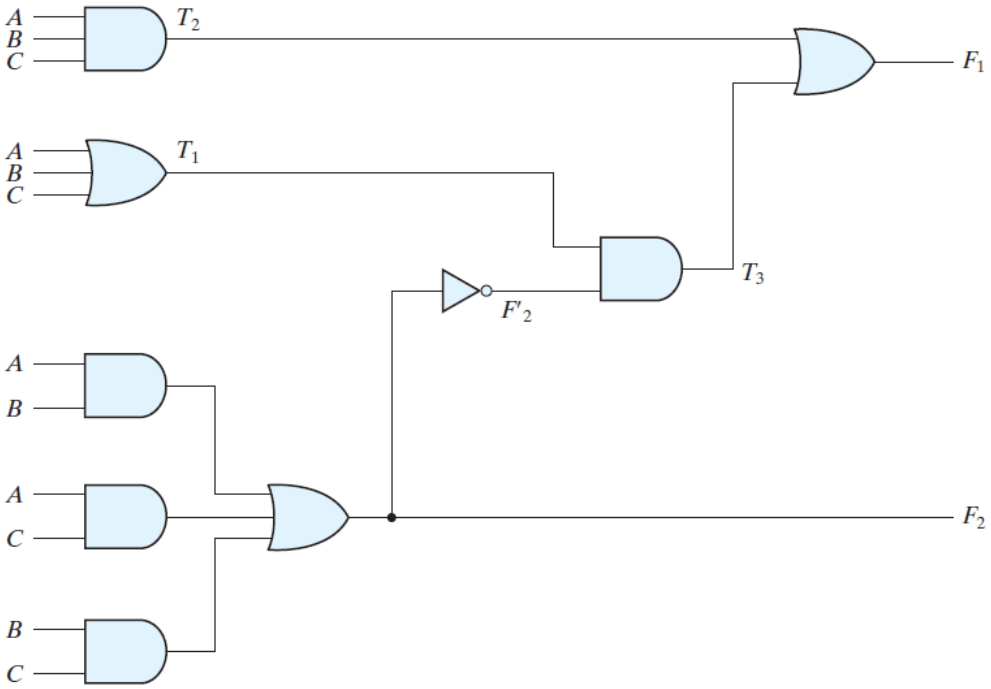


FIGURE 4.2
Logic diagram for analysis example

به دست آوردن تابع بولی خروجی‌های مدار:

$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

حال:

$$T_3 = F'_2 T_1$$
$$F_1 = T_3 + T_2$$

و بنابراین:

$$\begin{aligned} F_1 = T_3 + T_2 &= F'_2 T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$

بنابراین، توانستیم خروجی‌های F_1 و F_2 را بر حسب ورودی‌های A و B و C به دست آوریم.
حالا نوشتن جدول صحت مدار:

Table 4.1
Truth Table for the Logic Diagram of Fig. 4.2

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

طراحی مدار ترکیبی:

مراحل طراحی:

۱- تعیین تعداد ورودی‌ها و نامگذاری آنها،

۲- نوشتن جدول صحت،

۳- ساده‌سازی توابع بولی خروجی(های) مدار،

۴- ترسیم نمودار منطقی مدار[1].

مثال: طراحی مدار «مبدل کد BCD به کد مازاد ـ ۳»

۱ ـ تعداد ورودی‌ها و خروجی‌ها به ترتیب ۴ و ۴ است (چرا؟).

۲ ـ جدول صحت

[1] Logic diagram

Table 4.2
Truth Table for Code Conversion Example

Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

۳ـ ساده‌سازی

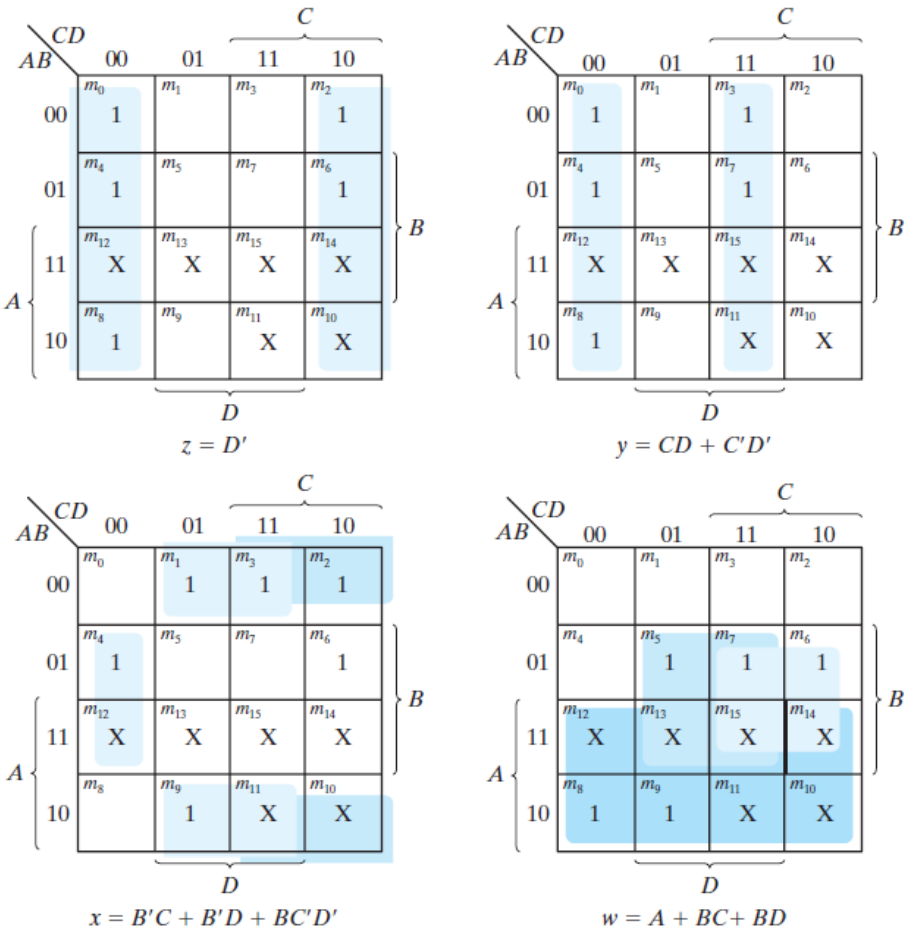


FIGURE 4.3
Maps for BCD-to-excess-3 code converter

$$z = D'$$
$$y = CD + C'D' = CD + (C + D)'$$
$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$
$$= B'(C + D) + B(C + D)'$$
$$w = A + BC + BD = A + B(C + D)$$

۴ـ ترسیم نمودار منطقی

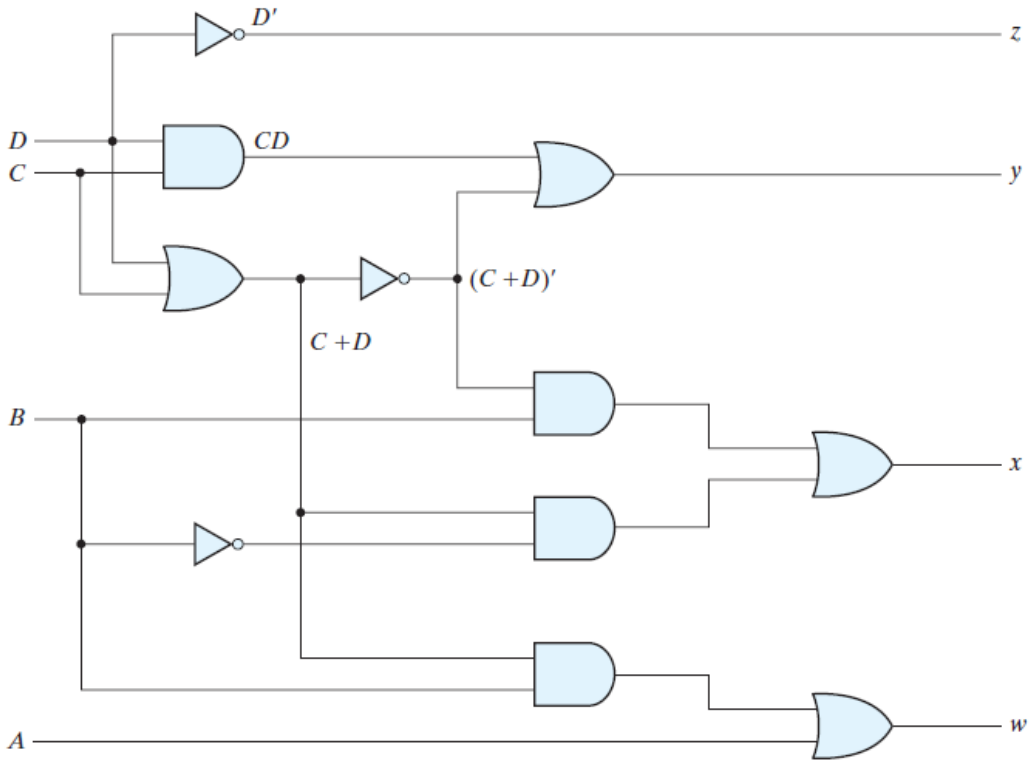


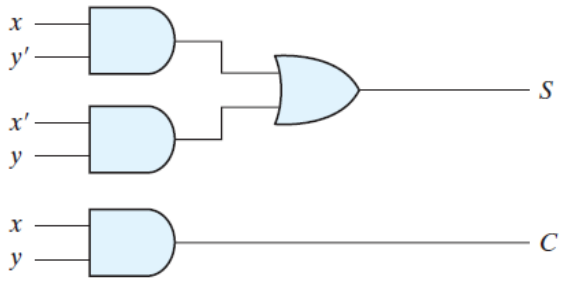
FIGURE 4.4
Logic diagram for BCD-to-excess-3 code converter

مثال: نیم‌جمع‌کننده (Half Adder)

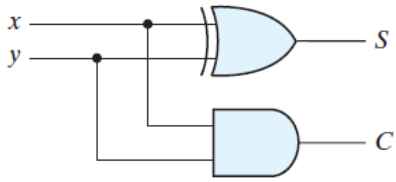
Table 4.3
Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = x'y + xy'$$
$$C = xy$$



(a) $S = xy' + x'y$
$C = xy$



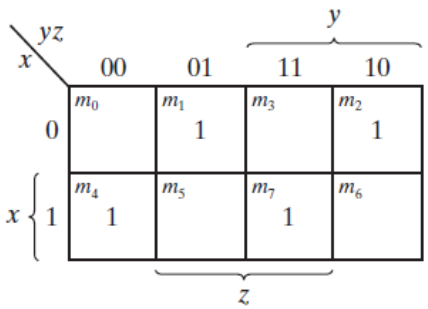
(b) $S = x \oplus y$
$C = xy$

FIGURE 4.5
Implementation of half adder

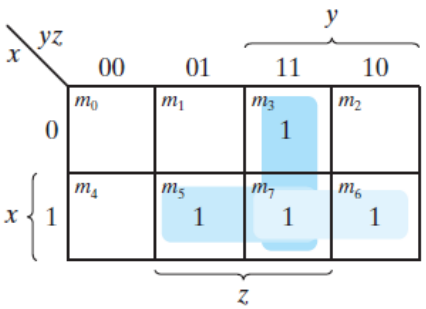
مثال: تمام جمع‌کننده (Full Adder)

Table 4.4
Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



(a) $S = x'y'z + x'yz' + xy'z' + xyz$



(b) $C = xy + xz + yz$

FIGURE 4.6
K-Maps for full adder

$$S = x'y'z + x'yz' + xy'z' + xyz$$
$$C = xy + xz + yz$$

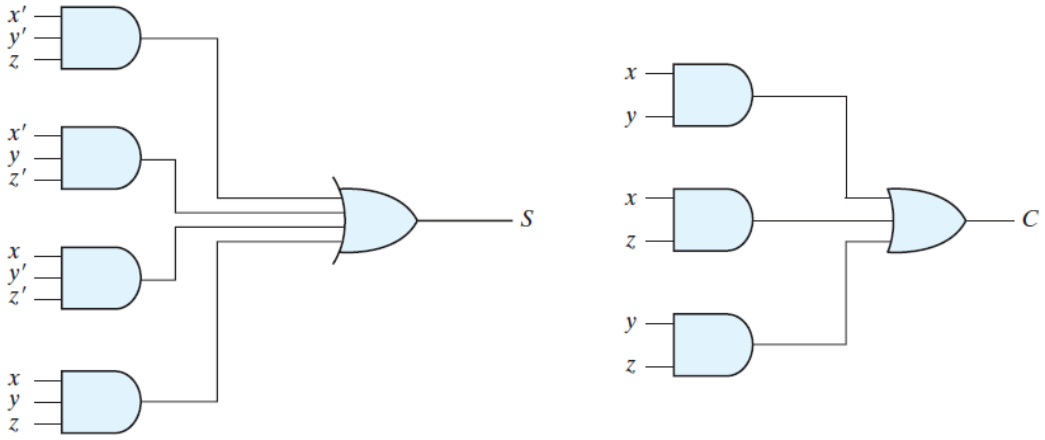


FIGURE 4.7
Implementation of full adder in sum-of-products form

یک تمام جمع‌کننده را می‌توان به کمک دو نیم‌جمع‌کننده و یک گیت OR پیاده‌سازی کرد. زیرا:
دلیل اول: دلیل منطقی (برای جمع سه بیت می‌توان آنها را دو به دو با هم جمع کرد)
دلیل دوم:

$$
\begin{aligned}
S &= z \oplus (x \oplus y) \\
&= z'(xy' + x'y) + z(xy' + x'y)' \\
&= z'(xy' + x'y) + z(xy + x'y') \\
&= xy'z' + x'yz' + xyz + x'y'z \\
C &= z(xy' + x'y) + xy = xy'z + x'yz + xy
\end{aligned}
$$

پس:

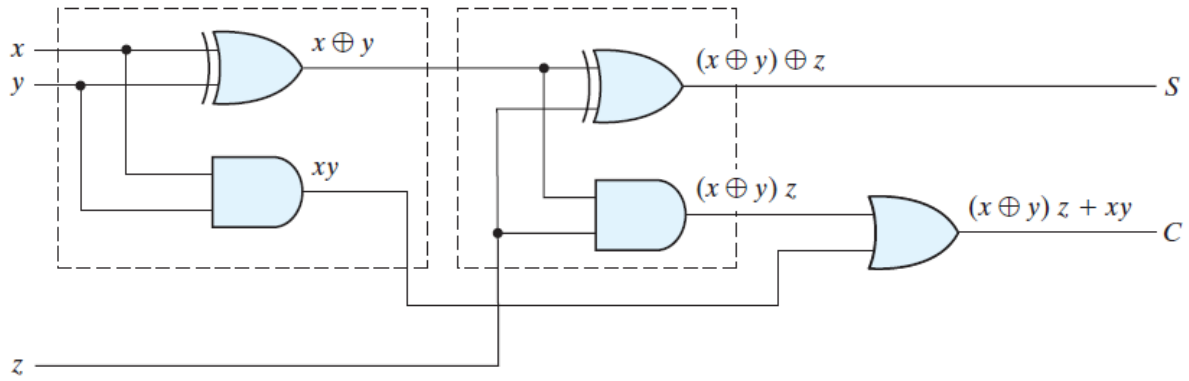


FIGURE 4.8
Implementation of full adder with two half adders and an OR gate

جمع‌کننده‌ی دودویی:
از ترکیب تعداد n تمام‌جمع‌کننده می‌توان یک جمع‌کننده‌ی n ـ بیتی ساخت. مثلاً برای حالت n=4:

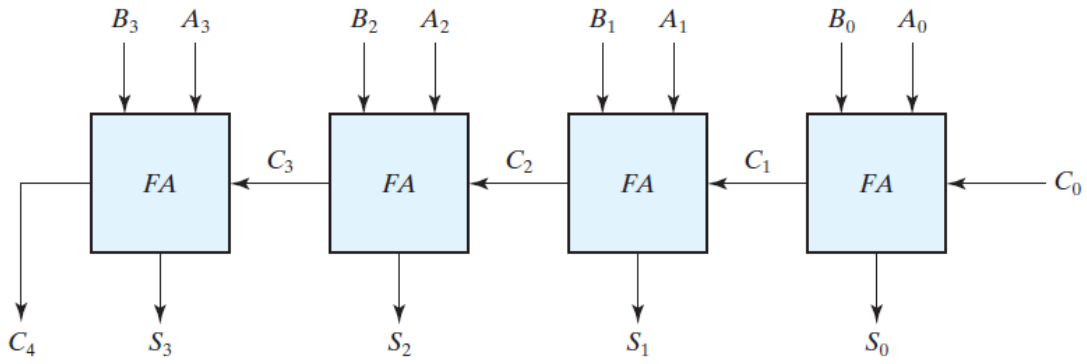


FIGURE 4.9
Four-bit adder

جمع‌کننده با پیش‌بین رقم نقلی (Adder with Carry Lookahead)

زمان انتشار کل در جمع‌کننده‌ی دودویی که اخیراً دیدیم برابر است با زمان تاخیر انتشار یک تمام‌جمع‌کننده ضرب در تعداد طبقات آن جمع‌کننده‌ی دودویی.

تاخیر انتشار رقم نقلی یکی از عوامل محدودساز سرعت مدار جمع‌کننده‌ی دودویی است؛ بنابراین در این جا به دنبال محاسبه‌ی سریع رقم‌های نقلی میانی برای رسیدن به جواب نهایی مدار جمع‌کننده‌ی دودویی هستیم. به عبارت دیگر، لازم نیست هر طبقه تمام‌جمع‌کننده منتظر کامل شدن فرآیند جمع در طبقه‌ی قبلی باشد؛ بلکه به دنبال طراحی مداری هستیم که همین که ورودی‌های داده (A_iها و B_iها) آماده شدند، مستقیماً به دنبال محاسبه‌ی رقم‌های نقلی باشد تا زمان تاخیر را کاهش دهد.

مدار یک تمام جمع‌کننده و نام‌گذاری سیگنالهای مورد نیاز در ادامه‌ی کار:

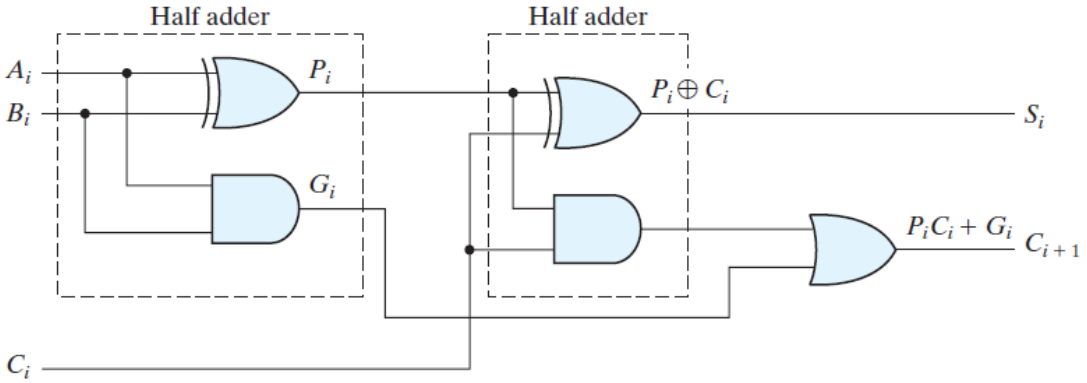


FIGURE 4.10
Full adder with *P* and *G* shown

ورودی‌های این مدار:

$$P_i = A_i \oplus B_i$$
$$G_i = A_iB_i$$

خروجی‌های این مدار:

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

به G_i «مولد نقلی[1]» گفته می‌شود. این سیگنال هرگاه A_i و B_i هر دو برابر با ۱ باشند، مقدار نقلیِ ۱ تولید می‌کند و این تولید مستقل از C_i (رقم نقلی وارد شده به طبقه‌ی i ـ اُم) است. به P_i «انتشار نقلی[2]» گفته می‌شود زیرا جمله‌ای است که معین می‌کند آیا رقم نقلیِ مرحله‌ی i ـ اُم به مرحله‌ی بعدیِ i+1 انتشار پیدا کند یا خیر (یعنی ۱ شدنِ رقم نقلی وارد شده به طبقه‌ی i آیا موجب ۱ شدنِ رقم نقلیِ وارد شده به طبقه‌ی i+1 بشود؟).

نوشتن عبارات مستقیم جهت محاسبه‌ی رقم‌های نقلیِ میانی:

$$C_0 = \text{input carry}$$
$$C_1 = G_0 + P_0 C_0$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 = P_2 P_1 P_0 C_0$$

حالا مدار مولد رقم نقلیِ پیش‌بین:

[1] Carry Generate

[2] Carry Propagate

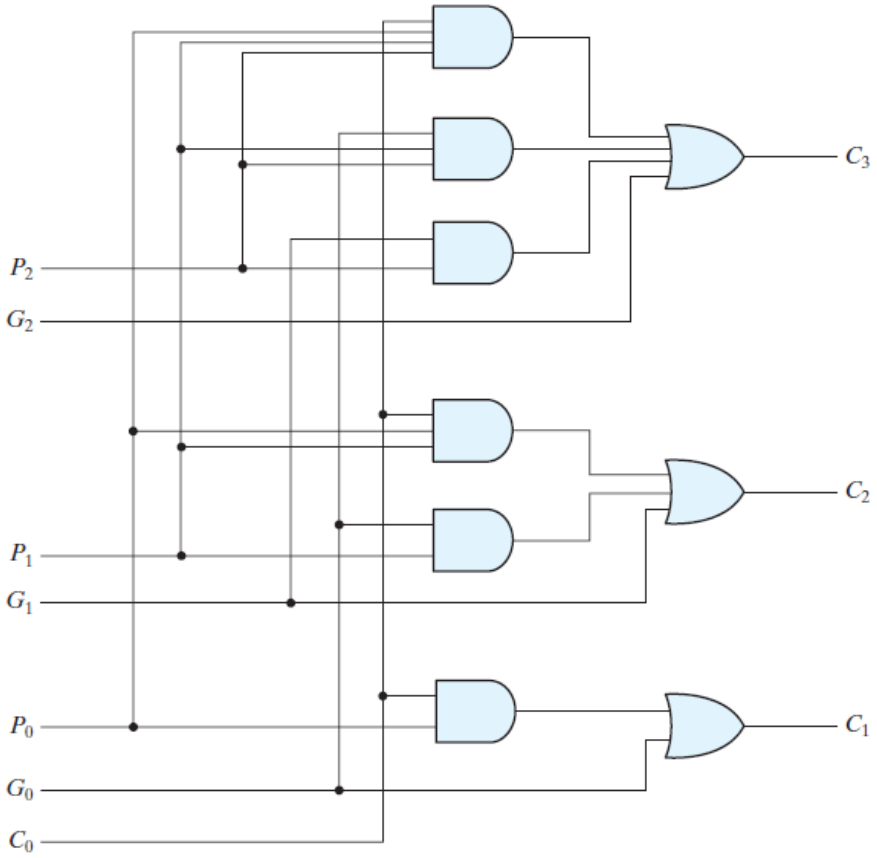


FIGURE 4.11
Logic diagram of carry lookahead generator

مدار نهایی:

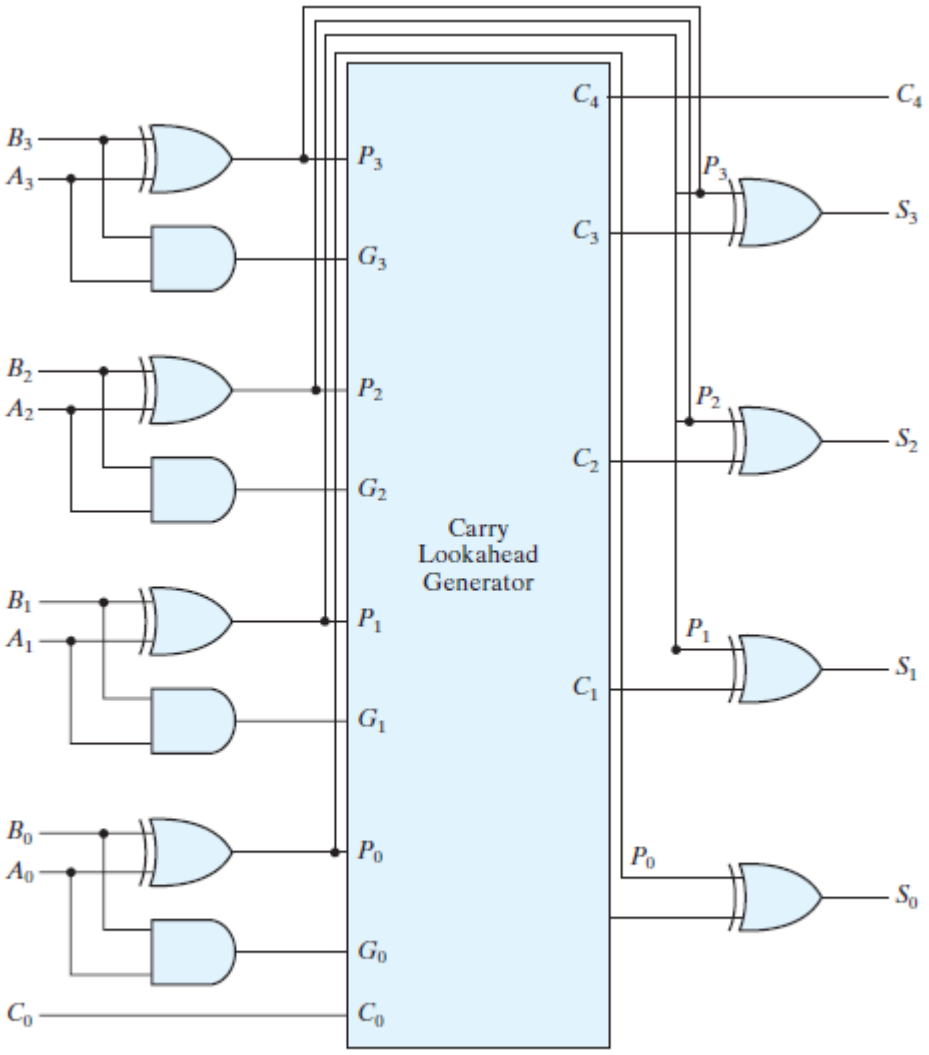


FIGURE 4.12
Four-bit adder with carry lookahead

مدار جمع و تفریق دودیی

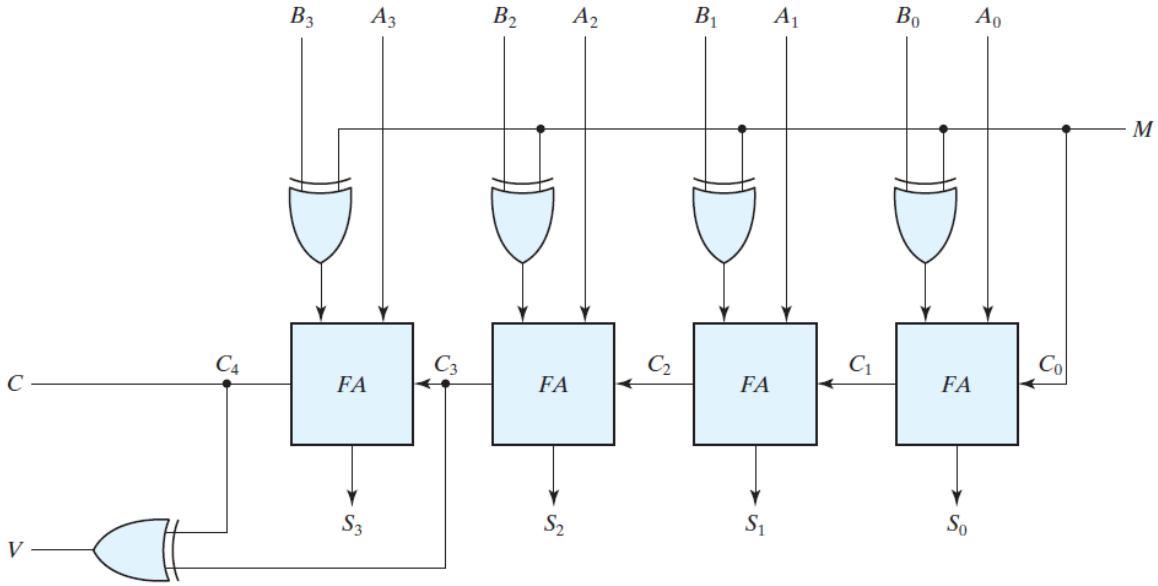


FIGURE 4.13
Four-bit adder–subtractor (with overflow detection)

روش تولید پرچم سرریز V در مدار فوق: هرگاه رقم نقلی وارد شده به بیت علامت با رقم نقلی خارج شده از این بیت یکی باشد، سرریز رخ نداده است وگرنه سرریز رخ داده و نتیجه‌ی چهار بیتی S_0 تا S_3 به تنهایی معتبر نبوده و به بیت پنجمِ C نیاز داریم.

جمع‌کننده‌ی BCD

الف: مدار

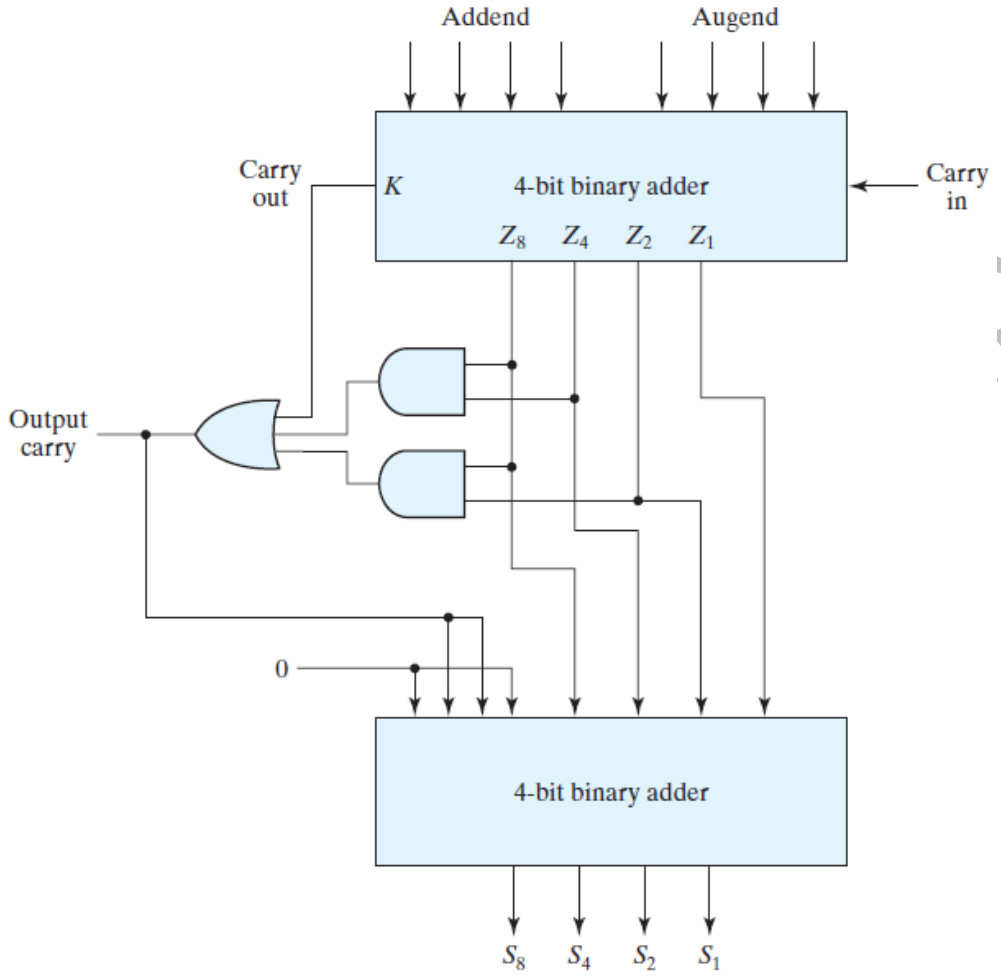


FIGURE 4.14
Block diagram of a BCD adder

ب: جدول عملکرد

Table 4.5
Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

توضیح: با توجه به جدول عملکرد فوق، زمانی باید نتیجه‌ی جمع دودویی دو رقم BCD تصحیح (یعنی با ۶ جمع) شود که یا K=1 شود، یا Z_8 و Z_4 هر دو ۱ شوند، یا Z_8 و Z_2 هر دو ۱ شوند.

ضرب کننده‌ی دو عدد ۲ ـ بیتی:

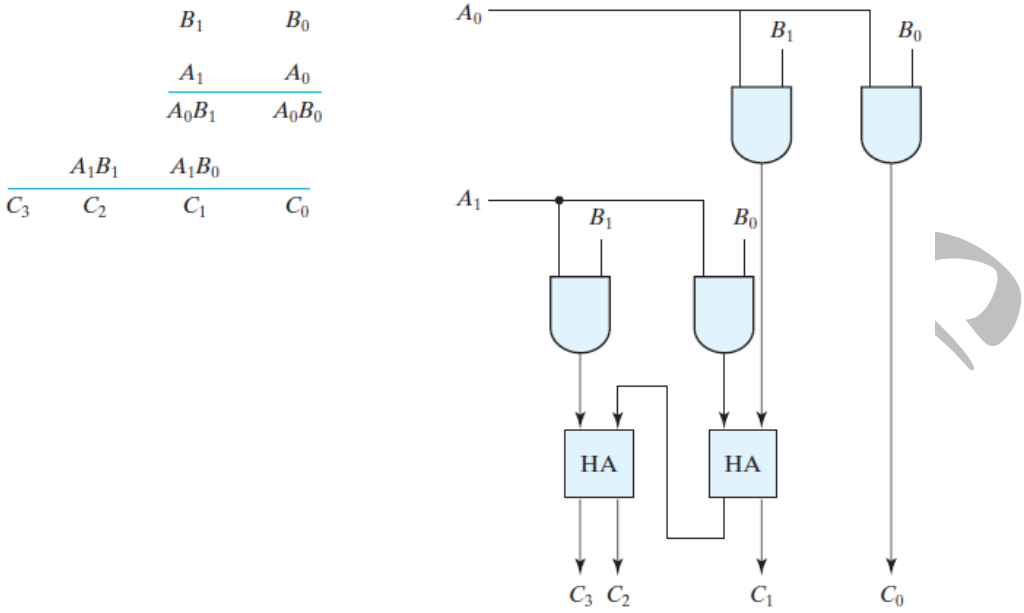


FIGURE 4.15
Two-bit by two-bit binary multiplier

ضرب‌کننده عدد ۳ـ بیتی در عدد ۴ـ بیتی:

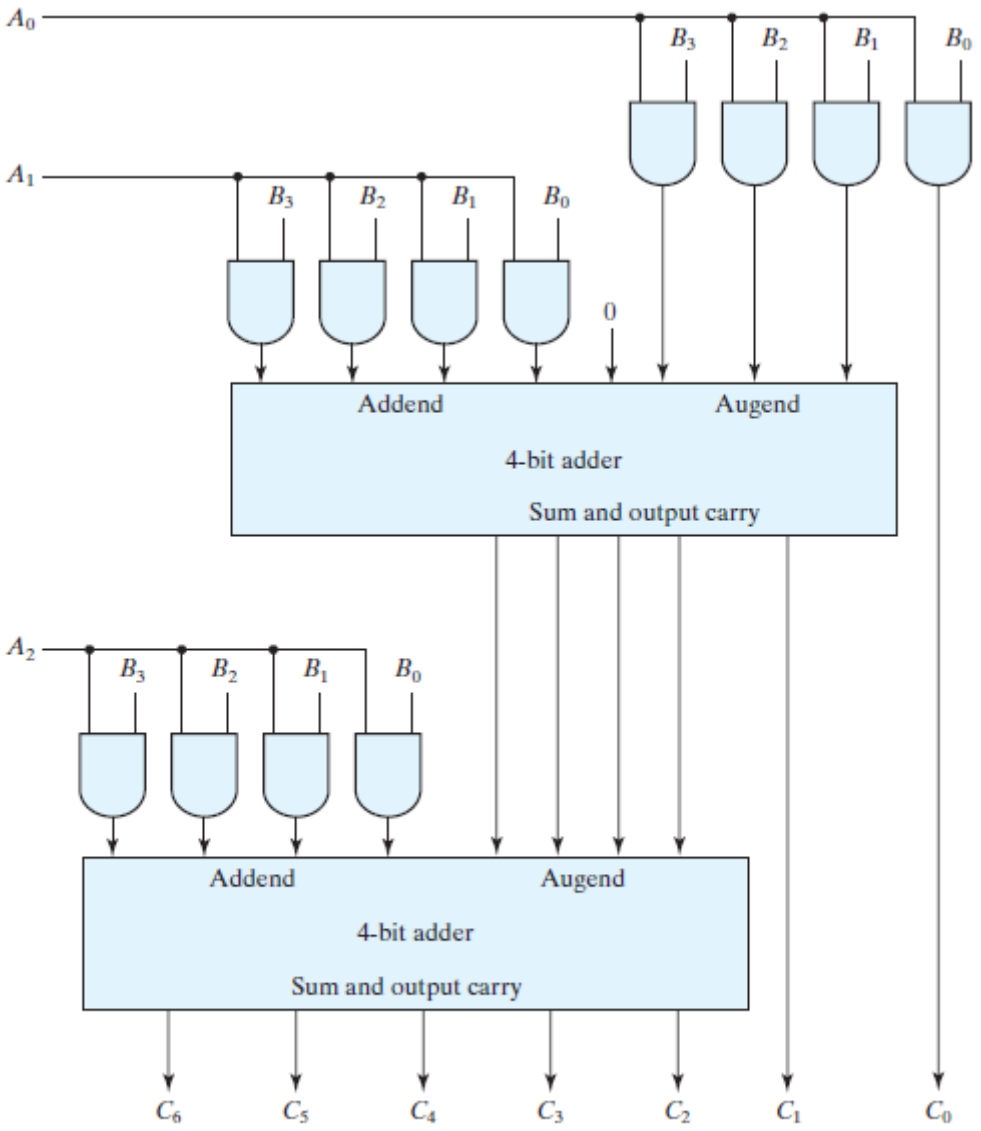


FIGURE 4.16
Four-bit by three-bit binary multiplier

رمزگشا یا دیکدر (Decoder)

مداری است که دارای n ورودی و 2^n خروجی است طوری که در هر لحظه فقط و فقط یکی از خروجی‌ها فعال (Active) و بقیه غیرفعال هستند. منظور از فعال بودن، می‌تواند ۱ شدن (در این صورت «فعال ـ بالا»[1] نامیده می‌شود) یا صفر شدن (در این صورت «فعال ـ پایین»[2] نامیده می‌شود) باشد.

[1] Active High
[2] Active Low

مثالی از یک دیکدر ۳ـ به ـ ۸ فعال ـ بالا:

Table 4.6
Truth Table of a Three-to-Eight-Line Decoder

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

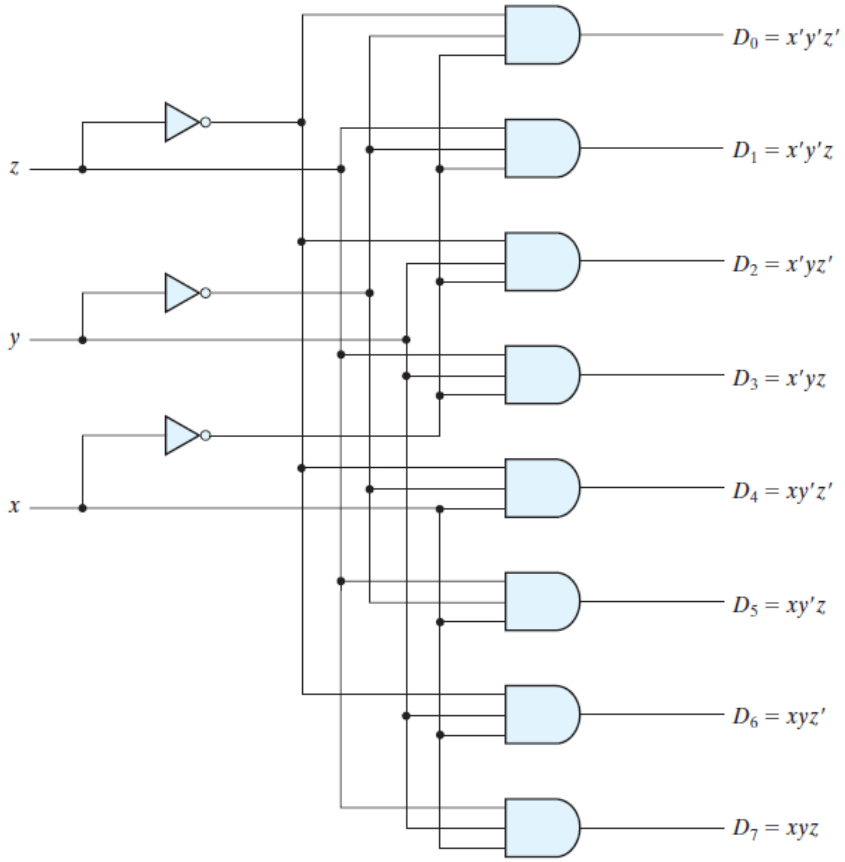
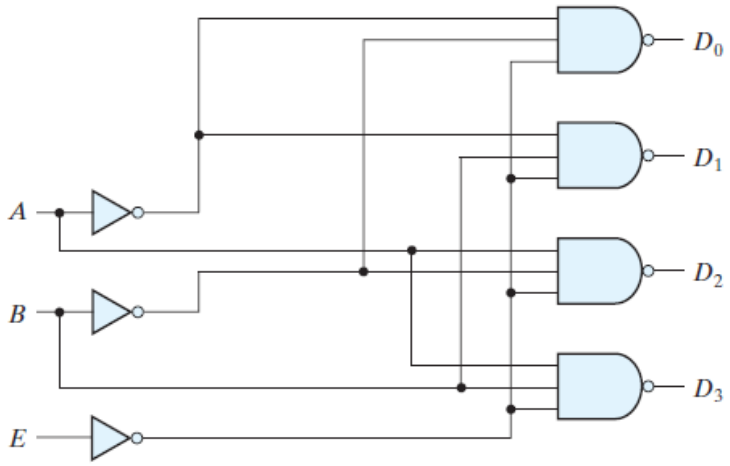


FIGURE 4.18
Three-to-eight-line decoder

دیکدر با ورودی فعال‌ساز (Enable)



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	1

(b) Truth table

FIGURE 4.19
Two-to-four-line decoder with enable input

ساخت دیکدر ابعاد بالا از روی دیکدرهای ابعاد پایین‌تر

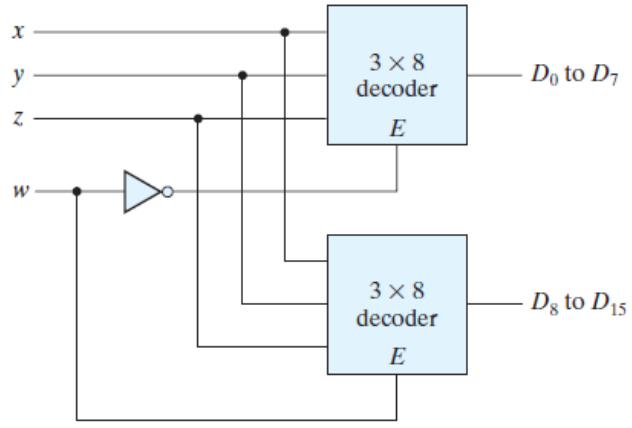


FIGURE 4.20
4 × 16 decoder constructed with two 3 × 8 decoders

پیاده‌سازی مدارات ترکیبی به کمک دیکدر

مثال: پیاده‌سازی یک تمام‌جمع‌کننده

$$S(x, y, z) = \Sigma(1, 2, 4, 7)$$
$$C(x, y, z) = \Sigma(3, 5, 6, 7)$$

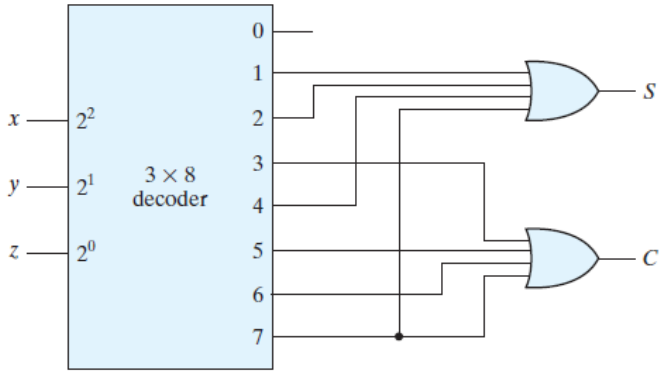


FIGURE 4.21
Implementation of a full adder with a decoder

رمزگذار یا انکودر (Encoder)

مداری است که دارای 2^n ورودی و n خروجی است. معمولاً از بین ورودی‌ها در هر لحظه فقط یکی دارای مقدار فعال (بسته به فعال‌ـ بالا یا فعال‌ـ پایین بودن) بوده و بقیه غیرفعال هستند. بسته به این که کدام ورودی مقدار فعال دارد، کد مناسب n ـ بیتی در خروجی تولید می‌شود. بنابراین، انکودر عکس عملکرد یک دیکدر را انجام می‌دهد.

نوعی انکودر به نام انکودر اولویت[1] وجود دارد که در آن، اگر بیش از یک ورودی مقدار فعال داشته باشند، یکی از آنها که اولویت بالاتری نسبت به بقیه دارد، تعیین کننده‌ی مقدار خروجی خواهد بود.

مثال: یک انکودر معمولی ۸ـ به‌ـ۳

Table 4.7
Truth Table of an Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

بنابراین، این انکودر را به کمک سه گیت OR (چهار ورودی) می‌توان ساخت.

[1] Priority Encoder

مثال: انکودر اولویت ۴ـ به ـ ۲

Table 4.8
Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

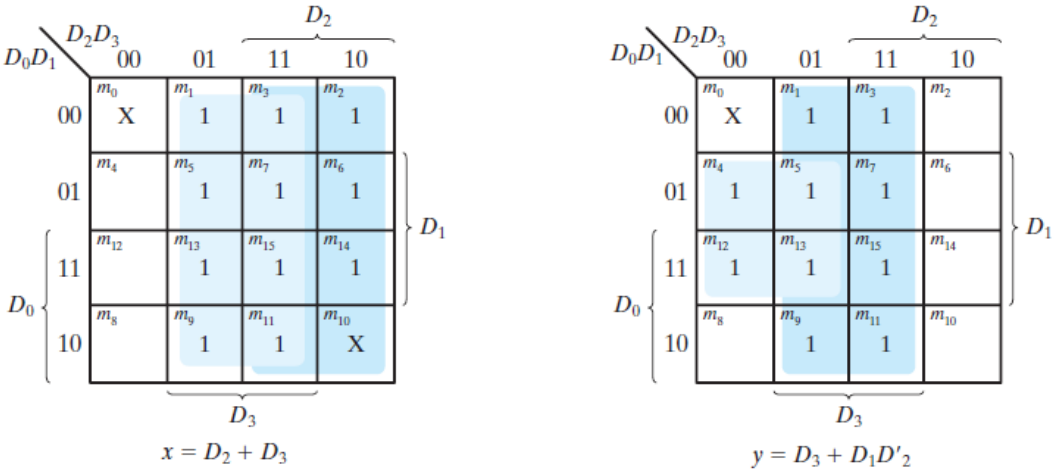


FIGURE 4.22
Maps for a priority encoder

$$x = D_2 + D_3$$
$$y = D_3 + D_1 D'_2$$
$$V = D_0 + D_1 + D_2 + D_3$$

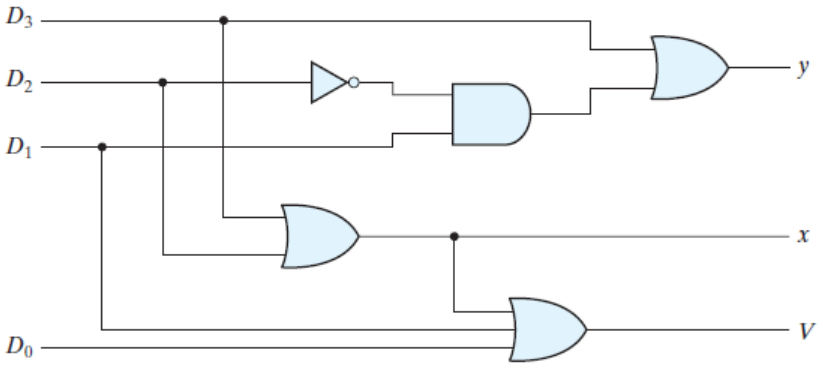
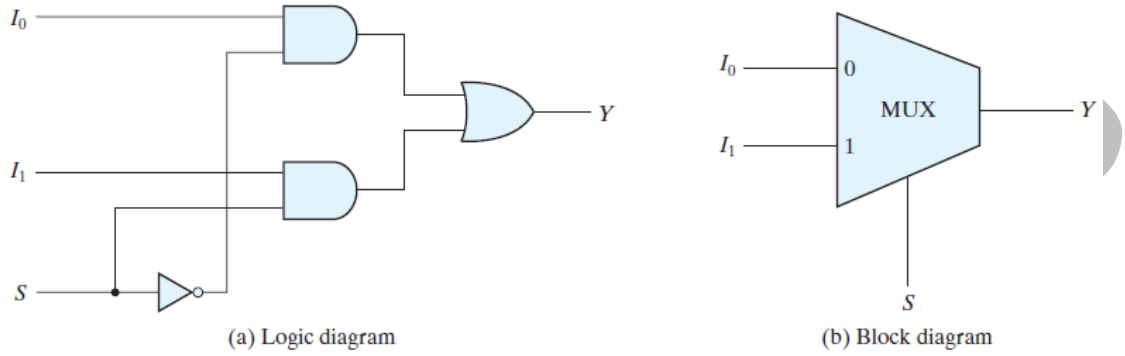


FIGURE 4.23
Four-input priority encoder

مالتی پلکسر (Multiplexer):

یک مالتی پلکسر 2^nـ به ـ ۱ مداری است که شامل 2^n ورودی داده، n ورودی انتخاب/کنترل، و یک خروجی (داده) است.

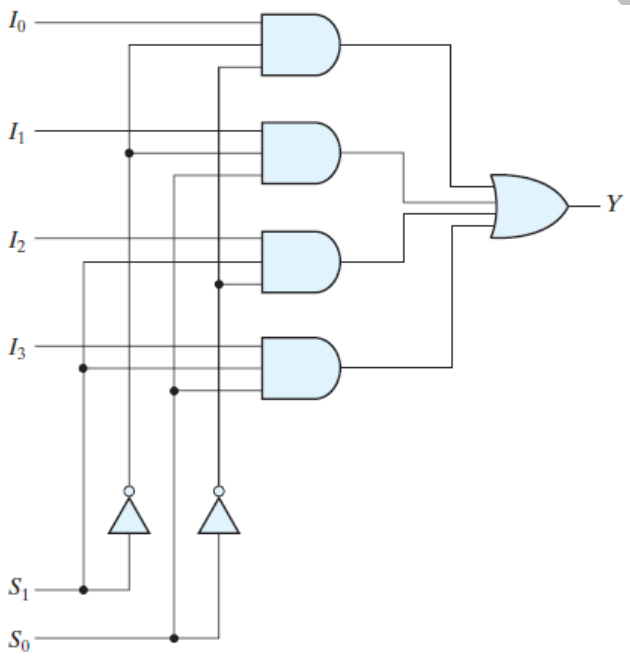
مثال: مالتی پلکسر ۲ ـ به ـ ۱



(a) Logic diagram (b) Block diagram

FIGURE 4.24
Two-to-one-line multiplexer

مثال: مالتی پلکسر ۴ ـ به ـ ۱



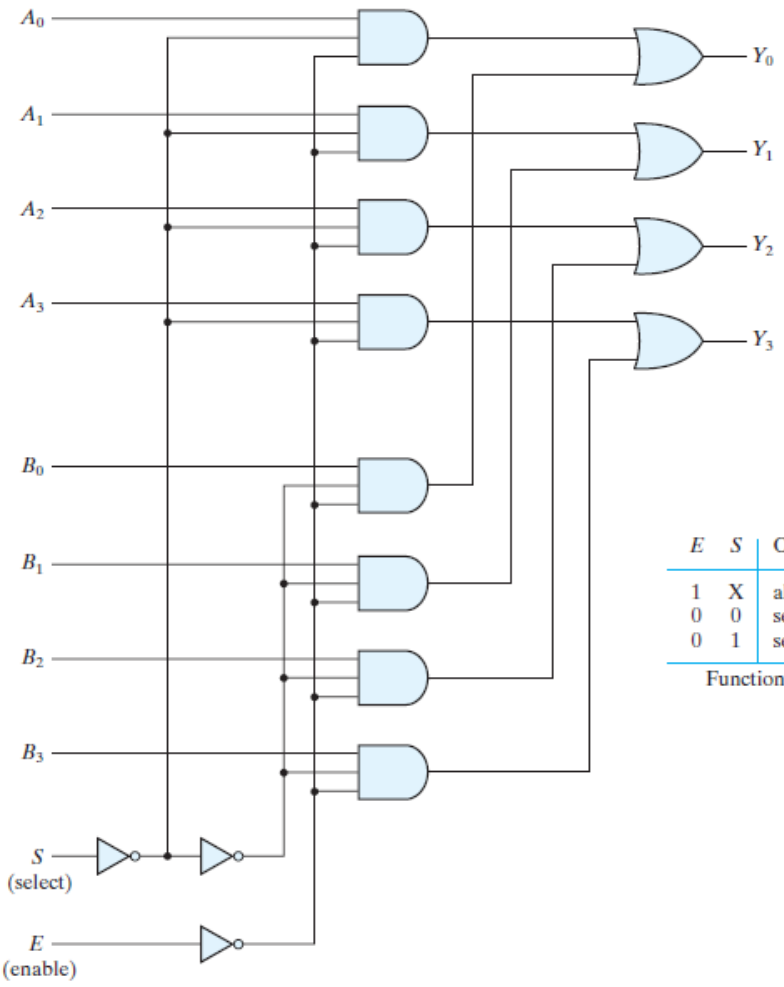
(a) Logic diagram

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

FIGURE 4.25
Four-to-one-line multiplexer

با ترکیب چند مالتی پلکسر ساده‌ی تک بیتی می‌توان یک مالتی پلکسر چندتایی ساخت. برای مثال، یک مالتی پلکسر ۲ ـ به ـ ۱ چهارتایی مانند شکل زیر است.



E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Function table

FIGURE 4.26
Quadruple two-to-one-line multiplexer

پیاده‌سازی توابع بولی به کمک مالتی‌پلکسر

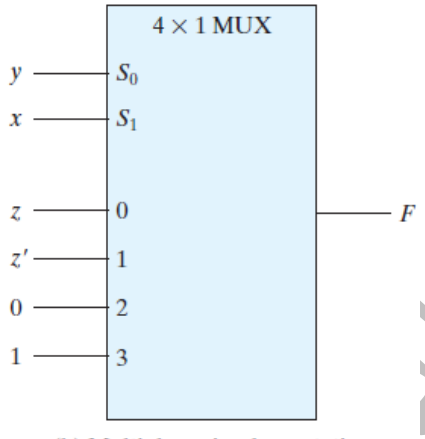
برای پیاده‌سازی یک تابع بولی n ـ متغیره از یک مالتی پلکسر با n-1 ورودی انتخاب استفاده می‌کنیم. ابتدا n-1 متغیر به n-1 ورودی انتخاب متصل می‌شوند. تنها متغیر باقیمانده‌ی تابع برای ورودی‌های داده مورد استفاده قرار می‌گیرد. اگر این متغیر باقیمانده را z بنامیم، هر یک از 2^{n-1} ورودیِ داده‌ی مالتی پلکسر را به یکی از مقادیر z، z’، 0، و یا 1 متصل می‌کنیم.

مثال ۱ : پیاده‌سازی تابع بولی زیر را در نظر بگیرید.

$$F(x, y, z) = \Sigma(1, 2, 6, 7)$$

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

FIGURE 4.27
Implementing a Boolean function with a multiplexer

مثال ۲: تابع بولی زیر را در نظر بگیرید.

$$F(A, B, C, D) = \Sigma(1, 3, 4, 11, 12, 13, 14, 15)$$

A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

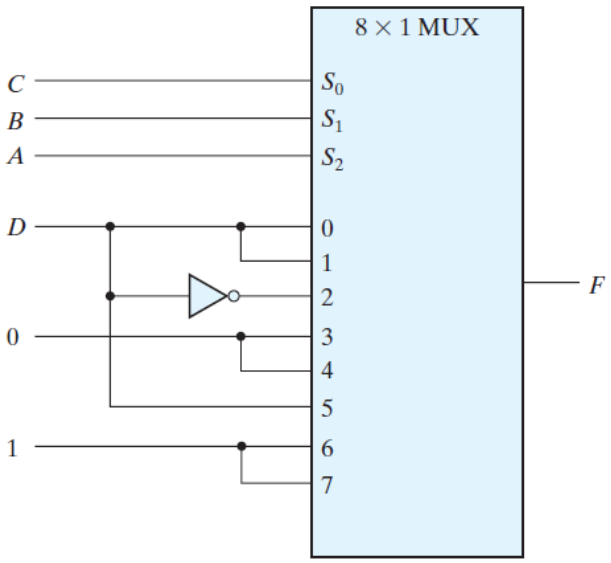


FIGURE 4.28
Implementing a four-input function with a multiplexer

بافر سه‌حالته

بافری است که خروجی آن، غیر از دو حالت صفر و ۱ ، دارای حالت/قابلیت امپدانس بالا (High Impedance) نیز می‌باشد.

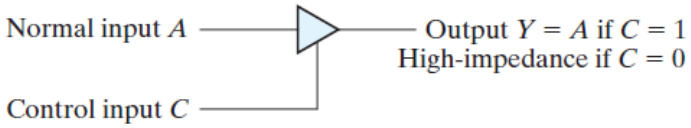
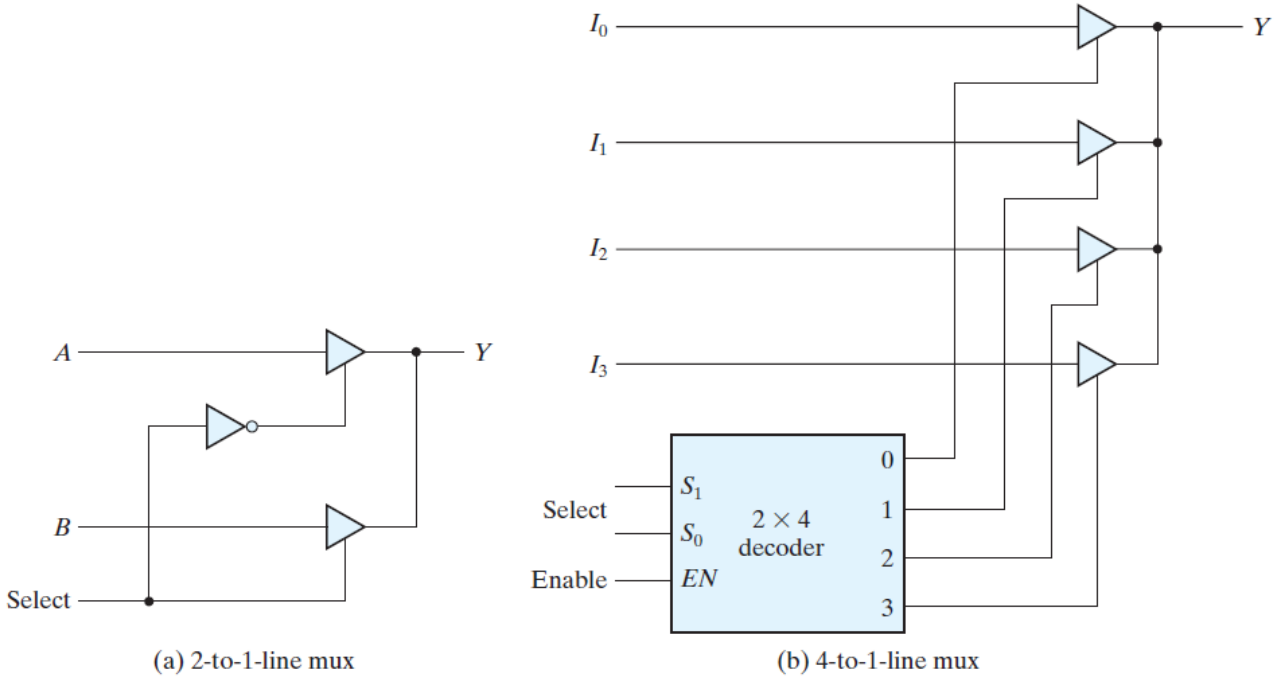


FIGURE 4.29
Graphic symbol for a three-state buffer

ساخت برخی گیت‌های منطقی با کمک بافر سه‌حالته

برای مثال ساخت یک مالتی پلکسر ۲ ـ به ـ ۱ و یک مالتی پلکسر ۴ ـ به ـ ۱ :



(a) 2-to-1-line mux (b) 4-to-1-line mux

FIGURE 4.30
Multiplexers with three-state gates

براي سلامتي رهبر انقلاب و تعجيل در ظهور حضرت ولي عصر (عج)

صلوات