فصل هفتم

حافظه و منطق برنامه‌پذیر

حافظه وسیله‌ای برای ذخیره‌ی اطلاعات دودویی است (اطلاعاتی که قرار است بعداً پردازش شوند، اطلاعاتی که حاصل انجام یک پردازش هستند، اطلاعات وارد شده از وسیله‌ی ورودی، و اطلاعاتی که قرار است برای وسیله‌ی خروجی ارسال شوند).

دو فرآیند کاری مرتبط با حافظه: خواندن و نوشتن.

دو نوع حافظه: «حافظه با دسترسی تصادفی، RAM» و «حافظه‌ی فقط خواندنی، ROM». در RAM هم می‌توان نوشت و هم از آن خواند. اما در ROM تنها عمل خواندن قابل انجام است؛ اطلاعات ذخیره شده در ROM قبلاً طی مکانسیمی سخت‌افزاری در ROM ذخیره شده (به این فرآیند، «برنامه‌ریزی» گفته می‌شود) که پس از قرار گرفتن در یک سیستم دیجیتال، فقط قابل خواندن است.

حافظه‌ی ROM نوعی وسیله‌ی منطقی برنامه‌پذیر یا PLD[1] است. برخی PLDهای معروف عبارتند از:

- آرایه‌ی منطقی برنامه‌پذیر (PLA[2])
- منطق آرایه‌ای برنامه‌پذیر (PAL[3])
- آرایه‌ی گیتیِ برنامه‌پذیر در محل (FPGA[4])

یک وسیله/افزاره‌ی منطقیِ برنامه‌پذیر در واقع، یک مدار مجتمع شامل تعدادی گیت منطقی (شاید صدها میلیون گیت) است که از طریق مسیرهایی الکترونیکی (شاید صدها هزار مسیر) به هم متصل شده و این مسیرها مانند فیوز عمل می‌کنند. در ابتدای کار تمام فیوزها دست‌نخورده هستند. برنامه‌ریزی افزاره به معنای سوزاندن آن دسته از فیوزهایی است که باید برای رسیدن به یک آرایش خاص از تابع منطقی مورد نظرمان، سوزانده شوند.

برای نمایش فشرده و مختصر دیاگرام منطقیِ داخلیِ یک PLD از نمادهای خاصی مانند نماد قسمت (ب) از شکل زیر که مربوط به یک گیت OR است، استفاده می‌کنیم؛ این شکل، یک گیت OR چند ورودی را در دو حالت معمولی و حالت آرایه‌ای نشان می‌دهد.

[1] Programmable Logic Device
[2] Programmable Logic Array
[3] Programmable Array Logic
[4] Field Programmable Gate Array

(a) Conventional symbol (b) Array logic symbol

FIGURE 7.1
Conventional and array logic diagrams for OR gate

حافظه‌ی RAM

علت نامگذاری «حافظه با دسترسی تصادفی» این است که زمان لازم برای دسترسی به هر مکان یا خانه از این نوع حافظه یکسان است؛ این ویژگی در مقابل برخی انواع حافظه مانند نوار مغناطیسی است که زمان لازم برای دسترسی/بازیابی اطلاعات ذخیره شده در آن بستگی به مکان داده دارد.

دیاگرام بلوکی واحد حافظه‌ی RAM به صورت شکل زیر است.

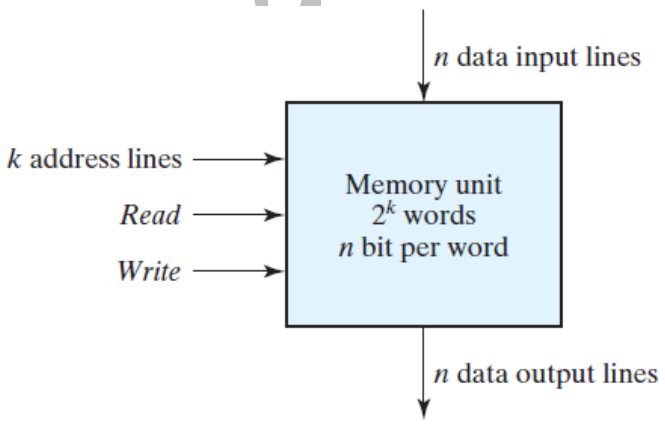


FIGURE 7.2
Block diagram of a memory unit

البته غیر از پایه‌های مشخص شده در دیاگرام فوق، ممکن است پایه‌های دیگری مانند فعال‌ساز حافظه (Enable) نیز وجود داشته باشد. معمولاً از ورودیِ فعال‌ساز برای انتخاب تراشه‌ی حافظه از بین چند تراشه‌ی موجود در سیستم دیجیتال استفاده می‌شود.

معمولاً پایه‌های خواندن و نوشتن با یکدیگر در یک پایه ادغام می‌شوند؛ بر حسب مقدار این پایه، یکی از دو عمل خواندن یا نوشتن انجام می‌شود. جدول زیر نقش این پایه را به همراه پایه‌ی فعال‌ساز نشان می‌دهد:

Table 7.1
Control Inputs to Memory Chip

Memory Enable	Read/Write	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

مثالی از یک نمونه حافظه:

Memory address		Memory content
Binary	Decimal	
0000000000	0	1011010101011101
0000000001	1	1010101110001001
0000000010	2	0000110101000110
⋮	⋮	⋮
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1101111000100101

FIGURE 7.3
Contents of a 1024 × 16 memory

شکل موج‌های زمان‌بندی[1]

پارامتر «زمان دستیابی[2]» یک حافظه به مدت زمانی گفته می‌شود که برای انتخاب یک خانه حافظه و خواندن آن لازم است.
پارامتر «زمان سیکل[3]» یک حافظه به مدت زمانی گفته می‌شود که برای تکمیل عملیات نوشتن یک مقدار در یک خانه حافظه لازم است.

عملکرد یک واحد حافظه توسط یک وسیله‌ی خارجی مانند واحد پردازش مرکزی (CPU) کنترل می‌شود. واحد CPU باید سیگنال‌های کنترل حافظه را به گونه‌ای تولید کند که عملیات کلاک‌دهی شده‌ی داخلیِ آن با عملیات خواندن و نوشتن از/در حافظه همزمان (یا سنکرون) شود. برای مثال اگر فرکانسِ کلاکِ CPU برابر 50 MHz باشد (پس دوره‌ی تناوب یا پریود کلاک برابر 20 ns باشد) و فرض کنیم زمان دست‌یابی و زمان سیکل حافظه‌ای که CPU آن را کنترل می‌کند، حداکثر

[1] Timing waveforms
[2] Access time
[3] Cycle time

50 ns باشد، در این صورت لازم است CPU برای انجام هر یک بار دسترسی به حافظه، حداقل دو و نیم یا سه سیکل کلاک زمان صرف کند.

زمان‌بندی حافظه در شکل زیر برای یک CPU با فرکانسِ کلاک 50 MHz و حافظه‌ای با ماکزیمم زمانِ سیکل 50 ns، نشان داده شده است.

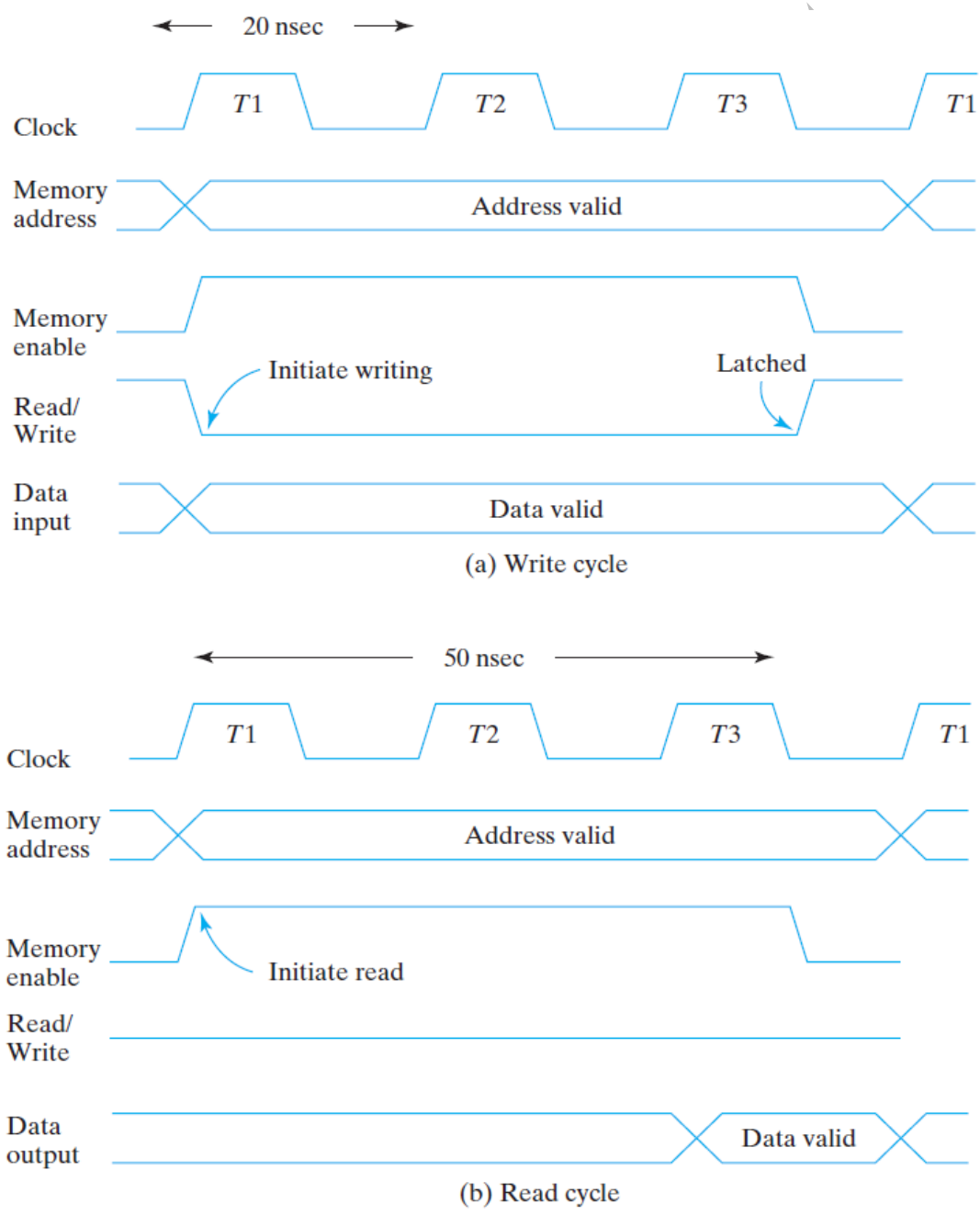


FIGURE 7.4
Memory cycle timing waveforms

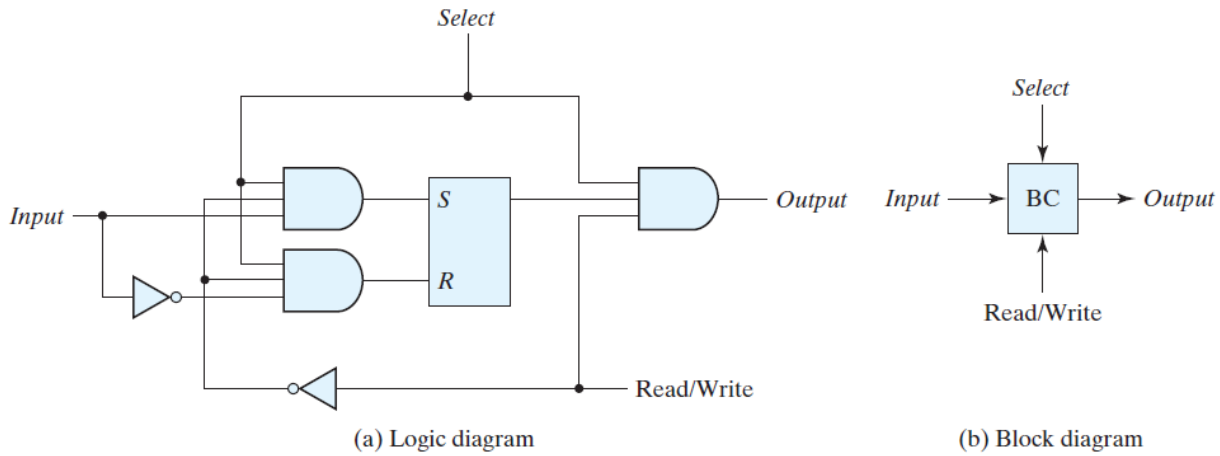
شکل (الف) مربوط به عملیات نوشتن در حافظه است که ملاحظه می‌شود نیاز به سه سیکلِ ۲۰ نانوثانیه‌ایِ T_1، T_2، و T_3 جهت تکمیل شدن دارد. ابتدا CPU باید در آغاز سیکلِ T_1 آدرس و داده‌ی معتبر را برای حافظه فراهم کند (دو خط متقاطع در خطوط آدرس و داده نشان دهنده‌ی امکان تغییر مقدار را نشان می‌دهند). «پس از تثبیت سیگنال‌های آدرس و داده»، دو سیگنال کنترلیِ فعال‌ساز و نوشتن/خواندن فعال می‌شوند تا بدین ترتیب اثر مخرب روی مقادیر دیگر خانه‌های حافظه نداشته باشند. دو سیگنال کنترلی باید حداقل به مدت ۵۰ نانوثانیه معتبر و باثبات باقی بمانند و سپس غیرفعال شوند. سیگنال‌های داده و آدرس باید برای مدت کوتاهی پس از غیرفعال شدن این دو سیگنال کنترلی،بدون تغییر و معتبر باقی بمانند. در پایان سومین سیکل کلاک، عمل نوشتن در حافظه تکمیل شده و هم‌اکنون CPU می‌تواند در سیکل T_1 بعدی دوباره به حافظه دسترسی داشته باشد.

شکل (ب) مربوط به عملیات خواندن از حافظه است. پس از معتبر و باثبات شدن آدرسِ داده، دو سیگنال کنترلی فعال می‌شوند. حافظه باید کلمه‌ی مشخص شده توسط آدرس را در مدت زمانی کمتر از ۵۰ نانوثانیه از لحظه‌ی فعال شدن روی خط داده‌ی خروجیِ خود قرار دهد. حال CPU در گذرِ منفیِ سیکلِ T_3 می‌تواند داده‌ی موجود در خط خروجیِ حافظه را به یکی از ثبات‌های داخلیِ خود منتقل کند.

دیکدکردن حافظه

ساختار درونی

ساختار داخلیِ یک حافظه‌ی RAM به ابعاد m کلمه در n بیت شامل m×n سلول ذخیره‌ساز دوودیی (BC) به همراه مدارات دیکدکردن لازم برای انتخاب هر کلمه است. سلول ذخیره‌ساز دودویی یک بلوک ساختاریِ پایه در یک واحد حافظه محسوب می‌شود. مدار معادل یک سلول دودویی که قادر به ذخیره‌ی یک بیت از اطلاعات است، در شکل زیر نشان داده شده است. بخشِ ذخیره‌سازیِ سلول با یک لچ SR به همراه گیت‌های لازم برای تشکیل یک لچ D مدل شده است. مدارِ واقعیِ یک سلول شامل چهار تا شش عدد ترانزیستور است اما برای درک عملکرد، بهتر است آن را با سمبل‌های منطقی مدل نماییم.



(a) Logic diagram (b) Block diagram

FIGURE 7.5
Memory cell

ساختار منطقیِ یک RAM کوچک در شکل زیر نشان داده شده است. این حافظه شامل چهار کلمه‌ی چهار بیتی است؛ بنابراین، شامل ۱۶ سلول دودویی و دو خط آدرس است. این دو خط آدرس وارد یک دیکدر 2×4 شده تا یکی از چهار کلمه را انتخاب کند. حافظه‌ای با تعداد 2^k کلمه‌ی n بیتی نیاز به k خط آدرس دارد تا وارد یک دیکدر $k\times2^k$ شوند. هر یک از خروجی‌های دیکدر یک کلمه‌ی n بیتی را برای نوشتن یا خواندن انتخاب می‌کنند.

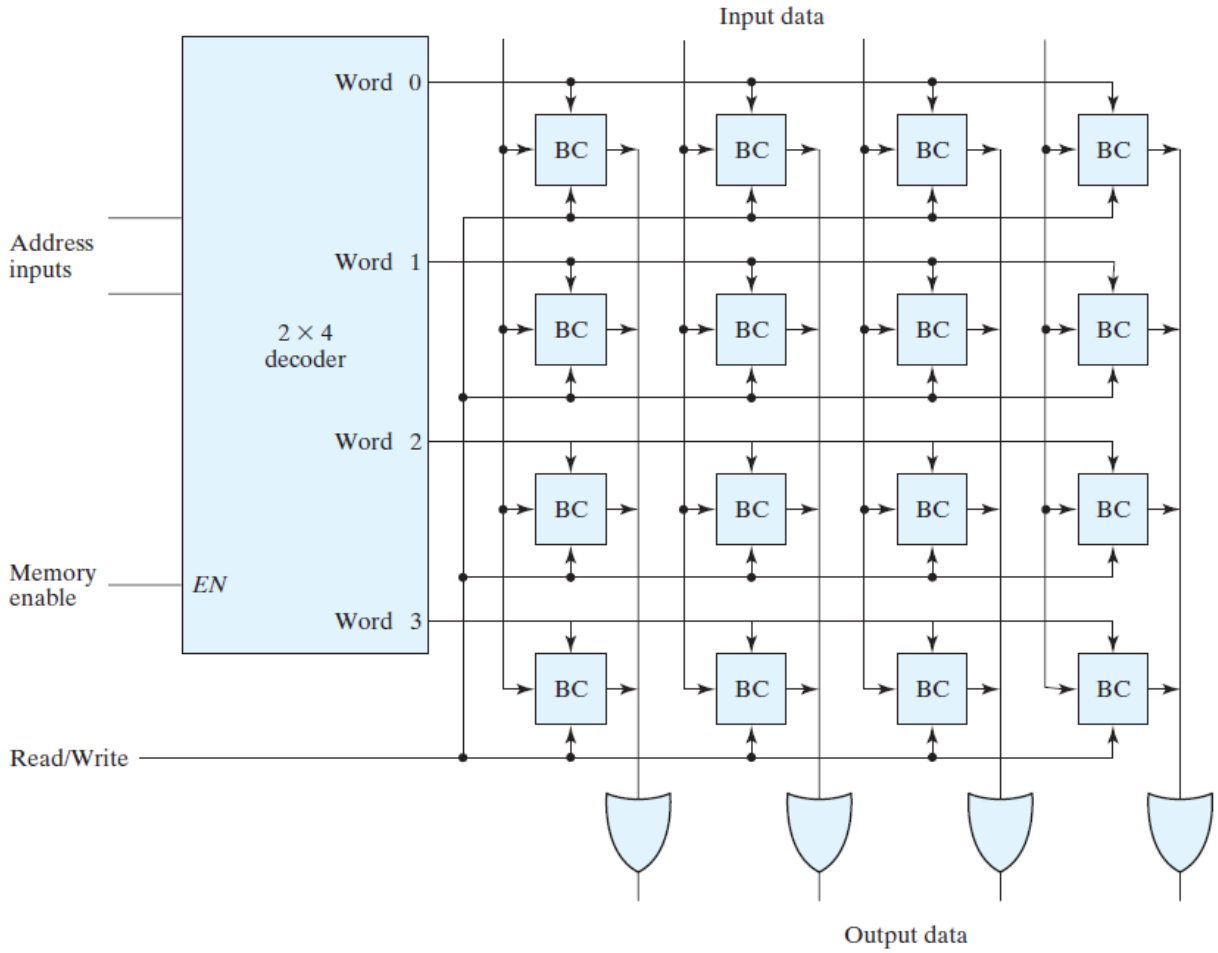


FIGURE 7.6
Diagram of a 4 × 4 RAM

دیکد کردن متقارن[1]

یک دیکدر با k ورودی و 2^k خروجی به 2^k گیت AND و k ورودی در هر گیت نیاز دارد. با انتخاب طرح دوبعدی و استفاده از دو دیکدر می‌توان تعداد کل گیت‌ها و نیز تعداد ورودی‌های گیت‌ها را کاهش داد. ایده‌ی اولیه در دیکدکردن دوبعدی، آرایش سلول‌های حافظه در یک آرایه است که تا حد ممکن نزدیک به یک مربع باشد. در این آرایش، دو دیکدر با k/2 ورودی، به جای دیکدر k ورودی، به کار می‌رود. یک دیکدر انتخاب سطر و یک دیکدر انتخاب ستون را در آرایش ماتریسی دوبعدی به عهده دارد.

الگوی انتخاب دوبعدی برای یک حافظه‌ی 1K کلمه‌ای در شکل زیر نشان داده شده است. به جای استفاده از یک دیکدر 10×1024، دو دیکدر 5×32 به کار رفته است. هنگام استفاده از دو دیکدر، ۶۴ گیت AND و با ۵ ورودی در هرکدام لازم است.

[1] Coincident decoding

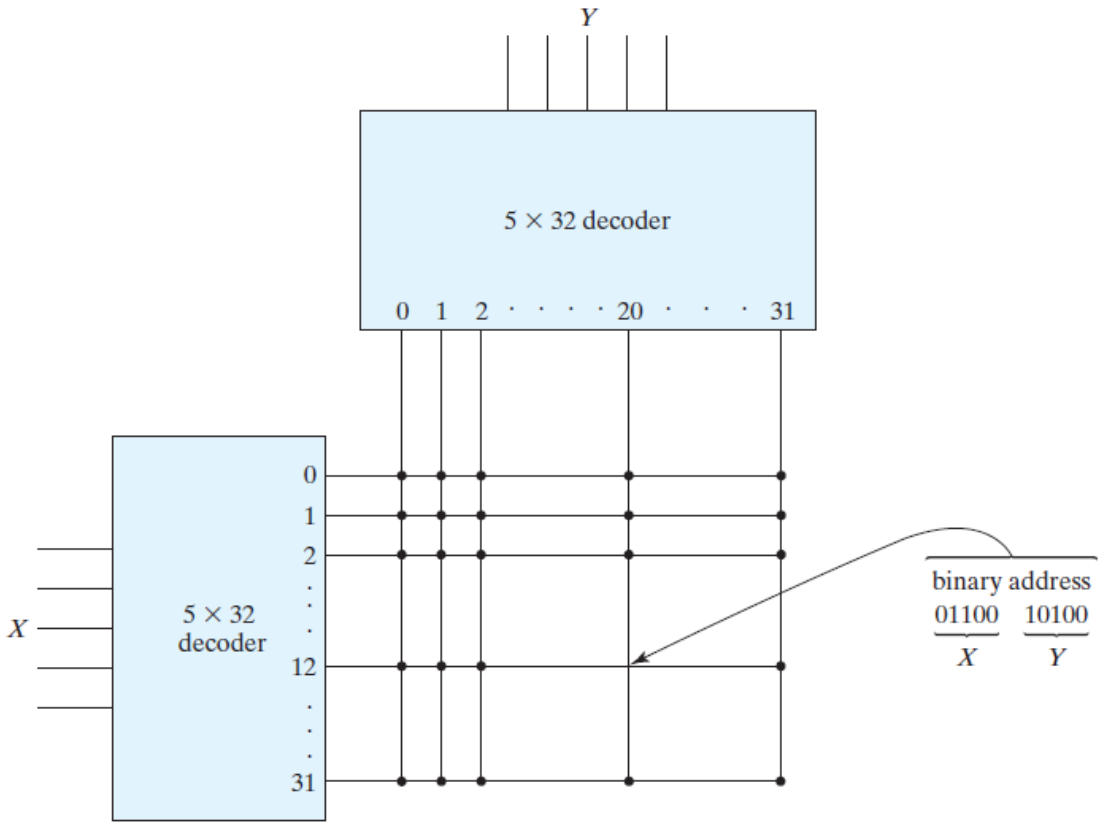


FIGURE 7.7
Two-dimensional decoding structure for a 1K-word memory

مالتی‌پلکس کردن آدرس

سلول حافظه در SRAM شامل شش ترانزیستور و سلول حافظه در DRAM شامل تنها یک ترانزیستور MOS و یک خازن است. در DRAMها به دلیل سادگی ساختار، چگالی سلول‌ها چهار برابر چگالی SRAM است؛ این ظرفیت باعث می‌شود تا ظرفیت آن نسبت به حافظه‌ی SRAM چهاربرابر گردد. به دلیل ظرفیت زیاد، دیکدکردن آدرس به صورت آرایه‌ی دو بعدی بوده و حافظه‌های بزرگتر حتی چندین آرایه دارند. برای کاهش تعداد پایه‌های[1] بسته‌ی تراشه، طراحان از مالتی‌پلکس کردن آدرس استفاده می‌کنند. در آرایه‌ی دوبعدی، آدرس به دو بخش (سطر و ستون) تقسیم می‌شود؛ حال، در زمان‌های مختلف ابتدا بخش سطر و سپس بخش ستون اعمال می‌شود.

برای تشریح ایده‌ی مالتی پلکس کردن از یک حافظه‌ی ۶۴ کیلوکلمه‌ای مطابق با شکل بعدی استفاده می‌کنیم. حافظه متشکل از آرایه‌ی دوبعدیِ سلول‌ها با ۲۵۶ سطر و ۲۵۶ ستون است که جمعاً $2^8 \times 2^8 = 2^{16} = 64K$ خواهد بود. تنها یک خط ورود داده، یک خط خروج داده، و یک کنترل خواندن/نوشتن وجود دارد. همچنین یک خط آدرس ۸ بیتی ورودی و دو سیگنال آگاه‌گر آدرس (Address Strobe) وجود دارد. آگاه‌گرهای آدرس برای فعال کردن ثبات‌های مربوط به سطر و ستون تعبیه شده‌اند. آگاه‌گر آدرس سطر (RAS) ثبات ۸ بیتیِ سطر و آگاه‌گر آدرس ستون (CAS) ثبات ۸ بیتیِ ستون را فعال می‌کند.

[1] Pin

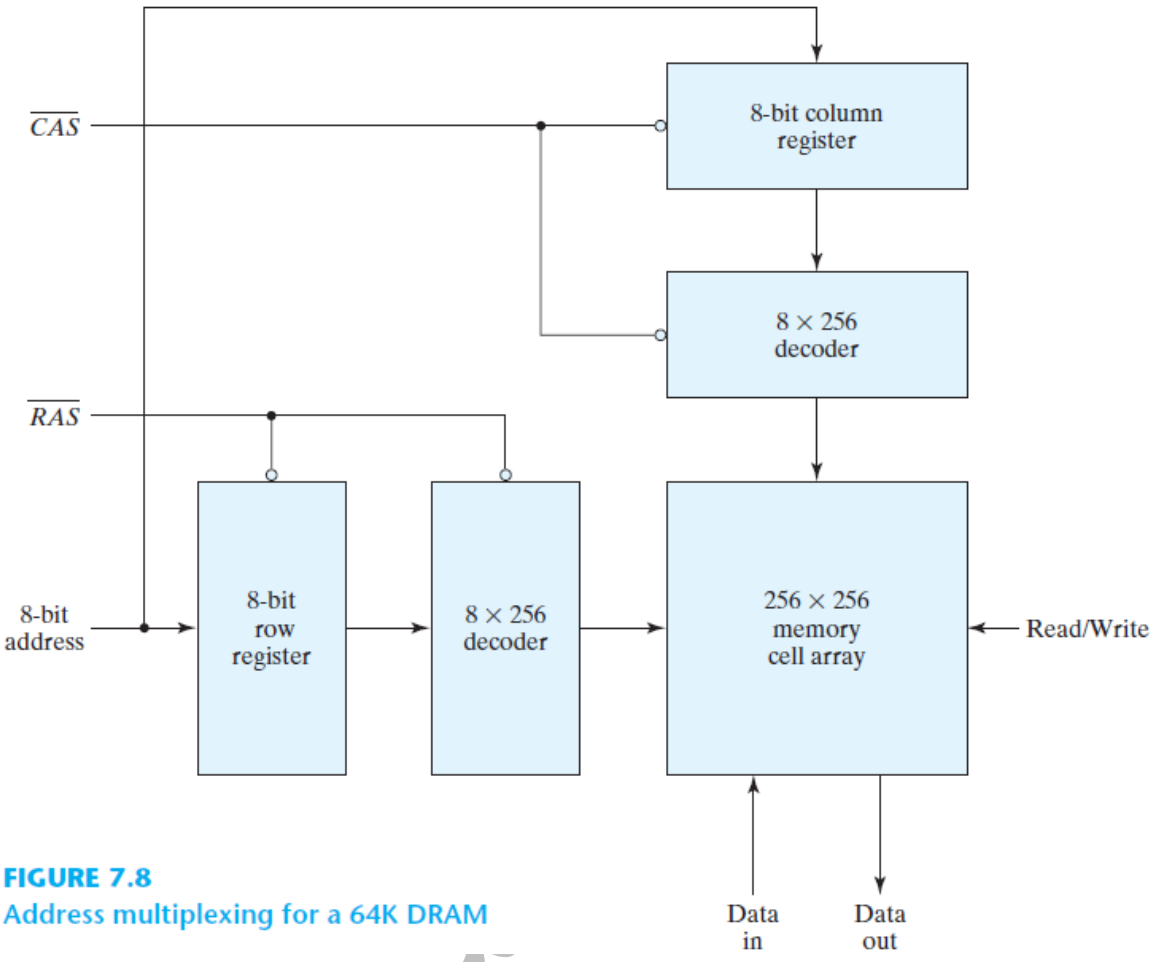


FIGURE 7.8
Address multiplexing for a 64K DRAM

۱۶ بیت آدرس با استفاده از سیگنال‌های RAS و CAS طی دو مرحله به DRAM اعمال می‌شود. ابتدا هر دو سیگنال آگاه‌گر در حالت 1 هستند. آدرس ۸ بیتیِ سطر به خط آدرس اعمال شده و سپس RAS=0 می‌شود تا موجب شود آدرس در ثبات سطر بارگذاری شود. سیگنال RAS هم‌زمان دیکدر سطر را نیز فعال می‌کند؛ در نتیجه، آدرس سطر دیکد شده و یکی از سطرهای آرایه انتخاب می‌شود. پس از گذشت زمانی برابر با زمان نشست[1] مربوط به انتخاب سطر، سیگنال RAS به سطح 1 بازمی‌گردد. حال آدرس ۸ بیتی ستون به خط آدرس اعمال شده و سیگنال CAS به 0 برده می‌شود تا آدرس مزبور در ثبات آدرس ستون ذخیره شده و همزمان، دیکدر آدرس ستون نیز آدرس مزبور را دیکد کرده و یکی از ستون‌های آرایه انتخاب می‌شود. در این لحظه از زمان، دو بخش آدرسِ ۱۶ بیتی در ثبات‌های خود بوده و دیکدرها نیز آن را دیکد کرده و یکی از سلول‌ها را انتخاب کرده‌اند و می‌توان عمل خواندن یا نوشتن را روی آن سلول اجرا کرد. قبل از اجرای عمل جدید روی سلول‌های حافظه باید CAS را به 1 بازگرداند.

[1] Settling time

کد همینگ برای تشخیص و تصحیح خطا

در هنگام ذخیره و بازیابی اطلاعات در یک حافظه ممکن است خطایی روی مقدار بیت‌های داده رخ دهد؛ لذا یک واحد حافظه باید قابلیت اطمینان داشته باشد. برای افزایش قابلیت اطمینان یک واحد حافظه می‌توان از کدهای تشخیص و تصحیح خطا استفاده کرد. کد همینگ یکی از همین کدگذاری‌ها است که مبتنی بر تولید بیت توازن کار می‌کند. برای مثال، در هنگام ذخیره‌ی بیت‌های داده، تعدادی بیت توازن نیز تولید و در کنار این بیت‌ها ذخیره می‌شوند. هر بیت توازن متناظر با گروه خاصی از بیت‌های داده تولید شده است. حال در هنگام خواندن داده‌ها، بیت‌های توازن نیز خوانده شده و با مجموعه‌ی جدیدی از بیت‌های تست تولید شده از روی داده‌های خوانده شده مقایسه می‌شوند. اگر بیت‌های تست صحیح باشند، خطایی رخ نداده است وگرنه، کد خاصی به نام نشانه (یا سندروم[1]) تولید می‌شود که محل وقوع بیت خطادار را نشان می‌دهد و با معکوس‌سازی آن بیت، می‌توان خطا را برطرف کرد.

کد همینگ تنها قادر به تشخیص یک بیت خطا بوده و خطاهای چندتایی قابل تشخیص نیستند.

برای یک داده‌ی n بیتی، تعداد بیت‌های لازم کد همینگ (k) از جدول زیر قابل محاسبه است:

Table 7.2
Range of Data Bits for k Check Bits

Number of Check Bits, *k*	Range of Data Bits, *n*
3	2–4
4	5–11
5	12–26
6	27–57
7	58–120

برای مثال، اگر داده‌ی ۸ بیتیِ 11000100 را در نظر بگیریم، تعداد k=4 بیت برای کد همینگ باید در نظر بگیریم. این بیت‌ها را باید در میان بیت‌های داده قرار دهیم. بنابراین، تعداد کل بیت‌ها برابر ۱۲=۸+۴ می‌شود. اگر موقعیت بیت‌ها را از ۱ تا ۱۲ شماره‌گذاری کنیم، بیت‌های توازن باید در موقعیت‌هایی قرار داده شوند که توان صحیحی از ۲ باشند. بنابراین ۴ بیت کد همینگ (که می‌توانیم آنها را با P_i بنامیم،) باید در موقعیت‌های ۱، ۲، ۴، و ۸ قرار داده شوند و بنابراین، این بیت‌ها را با علائم P_1، P_2، P_4، و P_8 نامگذاری می‌کنیم.

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12
	P_1	P_2	1	P_4	1	0	0	P_8	0	1	0	0

[1] Syndrome

برای تعیین مقدار بیت P_1 به این صورت عمل می‌کنیم که شماره‌ی این بیت را به صورت چهار بیتی می‌نویسیم (بنابراین: 1 = 0001). می‌بینیم که فقط اولین بیت از سمت چپ، یعنی بیتِ LSB، مقدار ۱ دارد. حال، بیت P_1 مسئول برقراری توازن زوج در بین تمام بیت‌هایی است که بیت کم‌ارزش (LSB) در نمایش دودوییِ شماره‌ی مکان آنها برابر 1 باشد؛ یعنی موقعیت‌های زیر:

1 = 0001, 3 = 0011, 5=0101, 7=0111, 9=1001, 11=1011

اگر به طور مشابه عمل کنیم، می‌بینیم که بیت P_2 مسئول بیت‌های با شماره‌ی مکانیِ ۲، ۳، ۶، ۷، ۱۰، و ۱۱، بیت P_4 مسئول بیت‌های با شماره‌ی مکانیِ ۴ تا ۷، و ۱۲، و بالاخره بیتِ P_8 نیز مسئول بیت‌های با شماره‌ی مکانیِ ۸ تا ۱۲ است. مقدار هر یک از بیت‌های P_i به گونه‌ای تعیین می‌شود که حاصل XOR کردن بیت‌های متناظر با این بیت (از جمله خود P_i) برابر با صفر بشود؛ به بیان دیگر مقدار P_i برابر است با حاصلِ XOR بیت‌های متناظر با P_i (به جز خودِ P_i).

بنابراین داریم:

$P_1 = XOR(1, 1, 0, 0, 0) = 0$
$P_2 = XOR(1, 0, 0, 1, 0) = 0$
$P_4 = XOR(1, 0, 0, 0) = 1$
$P_8 = XOR(0, 1, 0, 0) = 1$

بنابراین، داده‌ی ۱۲ بیتی که در حافظه ذخیره می‌شود عبارت است از:

	0	0	1	1	1	0	0	1	0	1	0	0
Bit position:	1	2	3	4	5	6	7	8	9	10	11	12

حال اگر این ۱۲ بیت از حافظه بازیابی شوند، برای بررسی وقوع خطا و تصحیح آن، ابتدا تعدادی بیت وارسی (یا بیتِ چک[1]) مشابه با همان فرآیندی که برای تولید بیت‌های P_i گفته شد، تولید می‌شود. این بیت‌های وارسی را C_i می‌نامیم. نحوه‌ی شماره‌گذاریِ اندیسِ i و نحوه‌ی محاسبه‌ی C_i مشابه با محاسبه‌ی P_i است. اگر خطایی رخ نداده باشد، تمام C_iها مقدار صفر خواهند داشت؛ اما اگر در یکی از بیت‌ها خطایی رخ داده باشد، حداقل یکی از C_iها صفر نبوده و در این صورت معادل دهدهیِ عدد دودیِ زیر (که همان نشانه یا سندروم است،) موقعیت آن بیتی که دچار خطا شده است را نشان می‌دهد:

$C = C_8C_4C_2C_1$

برای مثال، اگر خطایی در موقعیت ۱ یا ۵ رخ دهد:

[1] Check bit

Bit position:	1	2	3	4	5	6	7	8	9	10	11	12	
	0	0	1	1	1	0	0	1	0	1	0	0	No error
	1	0	1	1	1	0	0	1	0	1	0	0	Error in bit 1
	0	0	1	1	0	0	0	1	0	1	0	0	Error in bit 5

می‌توان دید:

	C_8	C_4	C_2	C_1
For no error:	0	0	0	0
With error in bit 1:	0	0	0	1
With error in bit 5:	0	1	0	1

کد همینگ را می‌توان برای کلمات داده‌ای با هر طول استفاده کرد. به طور کلی، این کد متشکل از k بیت وارسی و n بیت داده بوده و مجموعاً n+k بیت را شامل می‌شود. نشانه، C، متشکل از k بیت و مقدار آن بین صفر تا 2^k-1 تغییر می‌کند که مقدار صفر به معنای عدم وقوع خطا و هر مقدار دیگر (از ۱ تا 2^k-1 که تعداد آنها 2^k-1 عدد است)، محل وقوع خطا را نشان می‌دهد؛ بنابراین با توجه به این که کلاً n+k بیت داریم، مقدار 2^k-1 باید بزرگتر یا مساوی با n+k باشد:

$$2^k - 1 \geq n + k$$

و یا:

$$2^k - 1 - k \geq n$$

رابطه‌ی اخیر فرمولی برای تعیین تعداد بیت‌های داده‌ای که همراه با k بیت وارسی به کار می‌روند را نشان می‌دهد. برای مثال اگر k=3 باشد، حداکثر تعداد بیت‌های داده‌ی قابل استفاده برابر $n\leq 2^3-1-3 = 4$ بیت خواهد بود. برای k=4 داریم: $n\leq 2^4-1-4 = 11$ بیت. در این حال، تعداد بیت‌های داده ممکن است کمتر از ۱۱ بیت باشد اما حداقل باید ۵ بیت را داشته باشد وگرنه ۳ بیت وارسی کفایت کرده و نیازی به ۴ بیت وارسی نداریم. محدوده‌های مشخص شده در جدول ۷ـ۲ از روی همین قاعده محاسبه و نوشته شده‌اند.

تصحیح تک‌خطا و تشخیص جفت‌خطا[1]

برای افزایش کارایی کد همینگ، می‌توان به مجموعه‌ی بیتهای قبلی یک بیت توازن نیز افزود. این بیت توازن برای حفظ توازن زوج در بیت تمام بیت‌ها (در مثال قبلی، یعنی تمام ۱۳ بیت: ۱۲ بیت شامل داده و کد همینگ و یک بیت هم همین بیت توازن اضافه شده) به کار گرفته می‌شود. بنابراین در مثال قبلی، از ساختارِ

$$001110010100P_{13}$$

[1] Single-error correction and Double-error detection

استفاده می‌کنیم که P_{13} همان بیت توازن زوج اضافه شده است. در این مثال، مقدار P_{13} برابر با ۱ است؛ بنابراین، کلمه‌ی

0011100101001

در حافظه ذخیره می‌شود. در هنگام بازیابی این کلمه، دوباره بیت‌های وارسی (کلمه‌ی نشانه، C) و نیز بیت توازن زوج (با در نظر گرفتن تمام ۱۳ بیت)، P محاسبه شده و بر حسب مقدار C و P تصمیم‌گیری می‌شود:

If $C = 0$ and $P = 0$, no error occurred.
If $C \neq 0$ and $P = 1$, a single error occurred that can be corrected.
If $C \neq 0$ and $P = 0$, a double error occurred that is detected, but that cannot be corrected.
If $C = 0$ and $P = 1$, an error occurred in the P_{13} bit.

این روال می‌تواند بیش از دو خطا را شناسایی کند اما تصحیح آنها تضمین نمی‌شود.

حافظه‌ی فقط خواندنی (ROM)

حافظه‌ی ROM اساساً یک وسیله‌ی ذخیره‌سازی است که در آن اطلاعات دودویی به طور دائم نگهداری می‌شود. حافظه‌ی ROM فاقد ورودی داده است زیرا عمل نوشتن را انجام نمی‌دهد. تراشه‌های ROM دارای ورودی فعال‌ساز بوده و گاهی هم با خروجی سه حالته عرضه می‌شوند تا ساخت آرایه‌های بزرگتر ROM را امکان‌پذیر کنند.

نمودار بلوکی ROM:

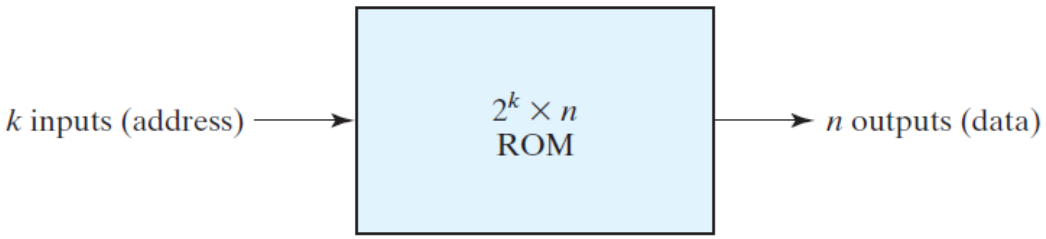


FIGURE 7.9
ROM block diagram

ساختار منطقی داخلِ یک حافظه‌ی ROM 32×8 (یعنی ۳۲ خانه‌ی حافظه که هر خانه حافظه، ۸ بیتی است) به صورت زیر است:

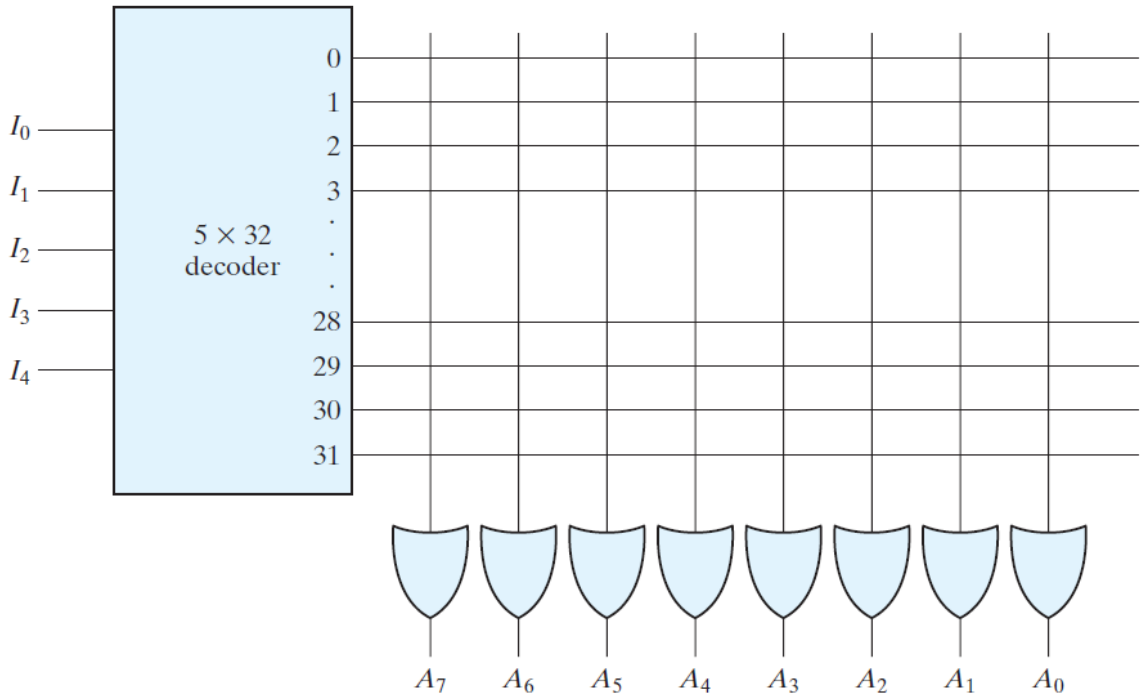


FIGURE 7.10
Internal logic of a 32 × 8 ROM

پنج خط آدرس به وسیله‌ی دیکدر 5×32 به ۳۲ خروجی مجزا دیکد می‌شوند. هر یک از خروجی‌های دیکدر یک آدرس حافظه را نشان می‌دهند. ۳۲ خروجی دیکدر به هر یک از ۸ گیت OR متصل هستند. بنابراین، هر گیت OR دارای ۳۲ ورودی است که به هر یک از خروجی‌های دیکدر متصل است. در دیاگرام منطقیِ اخیر، ۲۵۶ نقطه‌ی تقاطع[1] یا اتصال وجود دارد که قابل برنامه‌ریزی‌اند. یک اتصال قابل برنامه‌ریزی از دو خط در واقع به معنای کلیدی است که می‌تواند به هریک از دو حالت بسته (یعنی اتصال دو خط) و باز (یعنی دو خط جدا از هم) تغییر وضع دهد. برای پیاده‌سازی اتصالات تکنولوژی‌های مختلفی وجود دارد که یکی از ساده‌ترین آنها، فیوز است. فیوز در حالت معمولی دو نقطه را به هم متصل کرده است اما با اعمال یک پالس ولتاژ قوی، سوزانده شده و اتصال از بین می‌رود.

ذخیره‌سازی دودویی داخلی یک ROM با یک جدول صحت که محتوای کلمه‌ی متناظر با هر آدرس را نشان می‌دهد، قابل نمایش است. برای مثال، محتوای یک ROM 32×8 ممکن است به صورت جدول صحت نمایش داده شده در شکل زیر باشد.

[1] Crosspoint

Table 7.3
ROM Truth Table (Partial)

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		⋮							⋮			
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

روش سخت‌افزاریِ برنامه‌ریزیِ ROM مبتنی بر سوزاندن فیوزها بر طبق یک جدول صحت است. برای جدول صحت نمایش داده شده‌ی اخیر، سوزاندن یا برقراری اتصال فیوزها به شکل زیر خواهد بود:

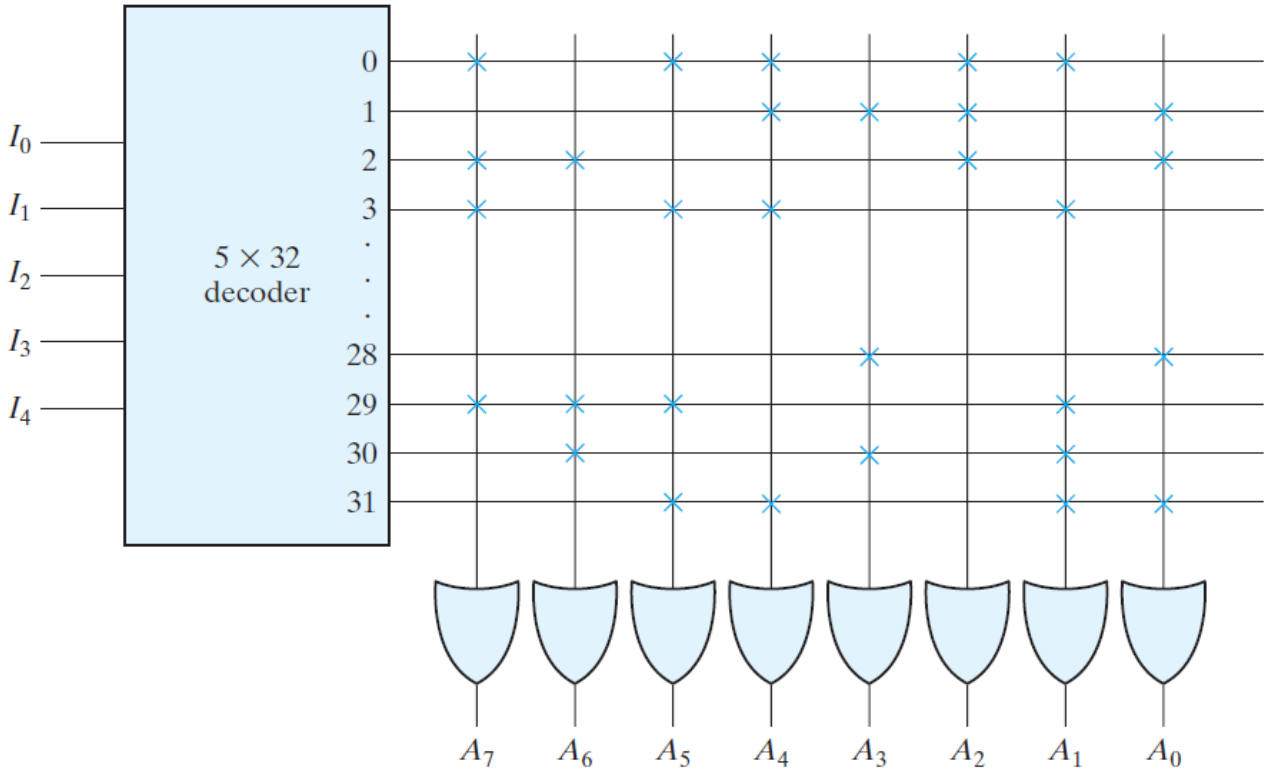


FIGURE 7.11
Programming the ROM according to Table 7.3

پیاده‌سازی مدارات ترکیبی

می‌دانیم که یک دیکدر با k متغیر ورودی قادر به تولید 2^k مینترم است. حال با OR کردن مینترم‌های مورد نظرمان از خروجیِ دیکدر می‌توانیم تابع دلخواه خود را ایجاد و تولید کنیم. با توجه به این که یک حافظه‌ی ROM دارای هر دو عنصر دیکدر و گیت OR است، پس می‌توان از آن برای پیاده‌سازی توابع بولی (و در نتیجه، مدارات ترکیبی) استفاده کرد. در مثال 32×8 ROM اخیر، هر یک از بیت‌های خروجیِ ROM را می‌توان به عنوان یک تابع بولی از ۵ متغیر ورودی در نظر گرفت. بنابراین، برای مثال، خروجیِ A_7 را می‌توان یک تابع بولی به فرم سیگمای زیر نوشت (با توجه هب جدول صحت اخیر):

$$A_7(I_4, I_3, I_2, I_1, I_0) = \Sigma(0, 2, 3, \ldots, 29)$$

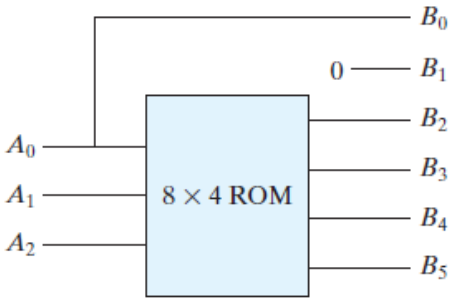
EXAMPLE 7.1

Design a combinational circuit using a ROM. The circuit accepts a three-bit number and outputs a binary number equal to the square of the input number.

The first step is to derive the truth table of the combinational circuit. In most cases, this is all that is needed. In other cases, we can use a partial truth table for the ROM by utilizing certain properties in the output variables. Table 7.4 is the truth table for the combinational circuit. Three inputs and six outputs are needed to accommodate all possible binary numbers. We note that output B_0 is always equal to input A_0, so there is no need to generate B_0 with a ROM, since it is equal to an input variable. Moreover, output B_1 is always 0, so this output is a known constant. We actually need to generate only four outputs with the ROM; the other two are readily obtained. The minimum size of ROM needed must have three inputs and four outputs. Three inputs specify eight words, so the ROM must be of size 8×4. The ROM implementation is shown in Fig. 7.12. The three inputs specify eight words of four bits each. The truth table in Fig. 7.12(b) specifies the information needed for programming the ROM. The block diagram of Fig. 7.12(a) shows the required connections of the combinational circuit.

Table 7.4
Truth Table for Circuit of Example 7.1

Inputs			Outputs						
A_2	A_1	A_0	B_5	B_4	B_3	B_2	B_1	B_0	Decimal
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49



(a) Block diagram

A_2	A_1	A_0	B_5	B_4	B_3	B_2
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	1
0	1	1	0	0	1	0
1	0	0	0	1	0	0
1	0	1	0	1	1	0
1	1	0	1	0	0	1
1	1	1	1	1	0	0

(b) ROM truth table

FIGURE 7.12
ROM implementation of Example 7.1

انواع ROM

نوع برنامه‌ریزی ماسک (mask programmable): به سفارش طراح و توسط سازنده عمل برنامه‌ریزی انجام می‌شود. این کار هزینه‌بر بوده و مقرون به صرفه نیست مگر این که در تعداد زیادی سفارش داده شود.

نوع ROM برنامه‌پذیر یا PROM[1]: در ابتدا همه‌ی فیوزها دست‌نخورده بوده و کاربر به کمک یک دستگاه پروگرامر که مبتنی بر اعمال پالس‌های ولتاژ قوی کار می‌کند، می‌تواند برخی فیوزها را سوزانده و الگوی دائمیِ خودش را برنامه‌ریزی کند. این نوع برنامه‌ریزی، برگشت‌ناپذیر بوده و اگر بخواهیم الگوی جدیدی در PROM ایجاد کنیم، باید قبلی را دور ریخته و PROM جدیدی استفاده کنیم.

نوع PROM پاک‌شونده یا EPROM[2]: این نوع را حتی پس از برنامه‌ریزی می‌توان به کمک تابش اشعه‌ی ماوراء بنفش خاصی پاک کرده و قابل برنامه‌ریزی مجدد ساخت.

نوع PROM پاک‌شونده‌ی الکتریکی یا EEPROM[3] یا E^2PROM: این نوع مشابه با نوع قبلی است با این تفاوت که عمل پاک کردن حافظه به صورت الکتریکی قابل انجام بوده و حتی نیازی به خارج کردن حافظه از داخل سوکت[4] خود نیست. حافظه‌های EEPROM دارای عمر پاک کردن بوده و حداکثر تا تعداد مشخصی می‌توان آنها را پاک و مجدداً برنامه‌ریزی کرد.

نوع حافظه‌ی فلش (Flash memory): مشابه با EEPROM است با این تفاوت که نیاز به دستگاه خاصی برای پاک کردن نداشته و مدارات لازم برای این کار به صورت داخلی در خود حافظه تعبیه شده‌اند. این نوع حافظه به طور گسترده‌ای در تکنولوژی‌های جدید مانند گوشی‌های تلفن همراه، دوربین‌های دیجیتال، تلویزیون دیجیتال، مخابرات از راه دور، ذخیره‌سازی غیرفرّار داده‌ها، و میکروکنترلرها استفاده می‌شوند. حافظه‌های فلش علاوه بر اینها، دارای مدارات دیگری هستند که امکان پاک کردن همزمان بلوک‌های با اندازه‌ی ۱۶ بایت تا ۶۴ کیلوبایت را می‌دهند. این نوع حافظه‌ها نیز مانند حافظه‌های EEPROM دارای عمر پاک کردن در حدود 10^5 دفعه هستند.

PLDهای ترکیبی[5]

PROM یک وسیله‌ی منطقیِ برنامه‌پذیر (PLD) ترکیبی به صورت مدار مجتمع است که گیت‌های آن به دو بخش آرایه‌ی AND (همان دیکدر) و آرایه‌ی OR تفکیک شده است تا فرم جمع حاصل‌ضرب‌ها را پیاده‌سازی کنند. در PROM آرایه‌ی AND ثابت اما آرایه‌ی OR برنامه‌پذیر است. از این دیدگاه، سه نوع PLD وجود دارد که در شکل زیر نمایش داده شده است.

---

[1] Programmable ROM

[2] Erasable PROM

[3] Electrically Erasable PROM

[4] Socket

[5] Combinational PLD

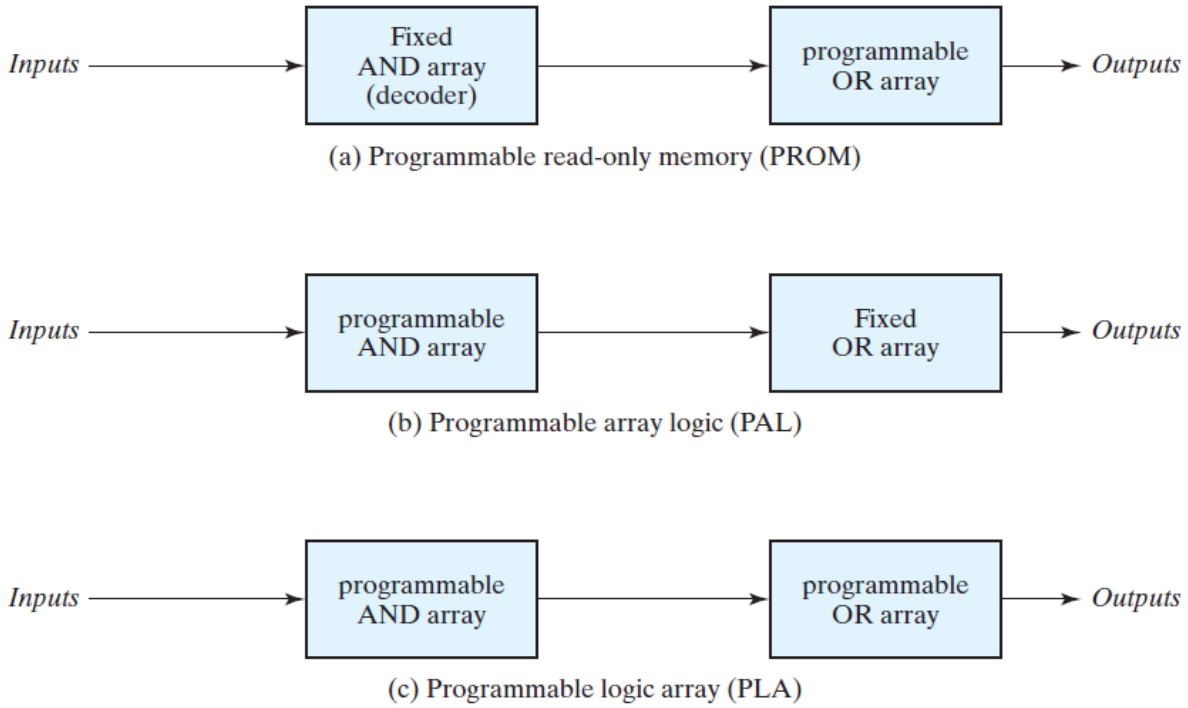


FIGURE 7.13
Basic configuration of three PLDs

در «منطق آرایه‌ی برنامه‌پذیر» یا PAL، برعکسِ PROM، آرایه‌ی AND برنامه‌پذیر و آرایه‌ی OR ثابت است. با انعطاف‌ترین PLD، «آرایه‌ی منطقی برنامه‌پذیر» یا PLA است که در آن هر دو آرایه برنامه‌پذیرند.
دو بخش بعدی به طراحی مدارات ترکیبی به کمک PLA و PAL خواهند پرداخت.

آرایه‌ی منطقی برنامه‌پذیر (PLA)

یک نمونه PLA برنامه‌ریزی شده با سه ورودی و دو خروجی در شکل زیر نشان داده شده است. در این PLA، دو تابع بولی زیر پیاده‌سازی شده‌اند:

$$F_1 = AB' + AC + A'BC'$$
$$F_2 = (AC + BC)'$$

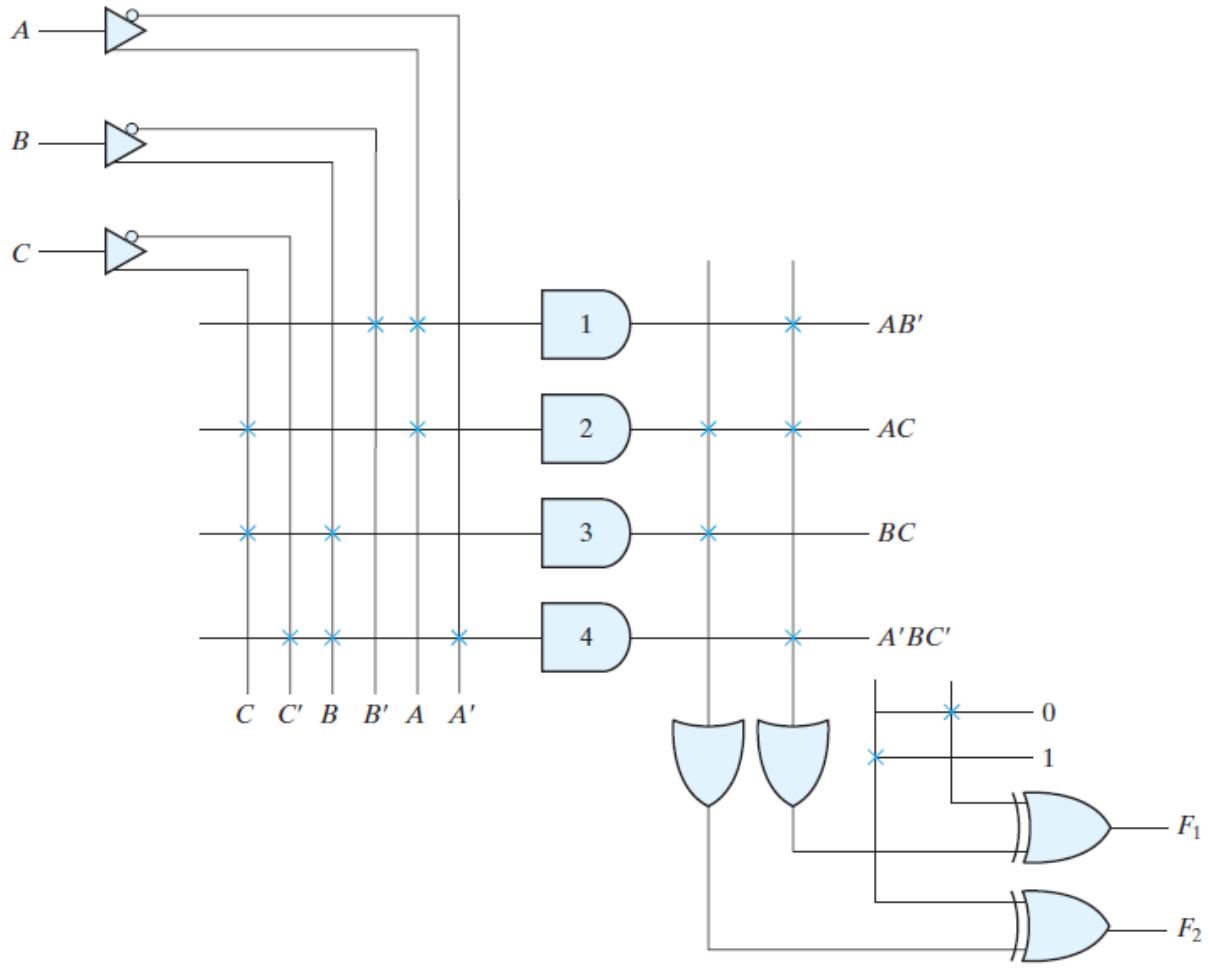


FIGURE 7.14
PLA with three inputs, four product terms, and two outputs

گیت‌های XOR برای معکوس کردن یا نکردن و در نتیجه تولید فرم SOP یا POS استفاده می‌شوند.
نقشه‌ی اتصالات PLA را می‌توان به کمک یک جدول مانند جدول زیر نمایش داد تا نیازی به ترسیم دیاگرام منطقی PLA (مانند شکل اخیر) نباشد.

Table 7.5
PLA Programming Table

		Inputs			Outputs (T)	(C)
	Product Term	A	B	C	F_1	F_2
AB'	1	1	0	—	1	—
AC	2	1	—	1	1	1
BC	3	—	1	1	—	1
$A'BC'$	4	0	1	0	1	—

دو نوع PLA وجود دارد: نوع ماسک برنامه‌پذیر[1] که مشابه با PROM ماسک برنامه‌پذیر است؛ و نوع برنامه‌پذیر در محل یا FPLA[2] که عمل برنامه‌ریزی توسط کاربر و به کمک دستگاه تجاری ویژه‌ی این کار انجام می‌شود.

با توجه به محدود بودن تعداد گیت‌ها در یک PLA، استفاده از ساده‌سازی تابع بولی به منظور کاهش تعداد جملات بسیار مهم است؛ توجه شود که تعداد لیترال‌ها مهم نیست زیرا تمام متغیرهای ورودی به هر ترتیب در دسترس‌اند. دیگر این که بهتر است هر دو نوع تابع خروجی، یعنی متمم یا غیرمتمم، باید بررسی شوند تا ببینیم کدام یک با تعداد کمتری جمله ضرب قابل بیان بوده و کدام جمله نیز در دیگر توابع مشترک است.

EXAMPLE 7.2

Implement the following two Boolean functions with a PLA:

$$F_1(A, B, C) = \Sigma(0, 1, 2, 4)$$
$$F_2(A, B, C) = \Sigma(0, 5, 6, 7)$$

The two functions are simplified in the maps of Fig. 7.15. Both the true value and the complement of the functions are simplified into sum-of-products form. The combination that gives the minimum number of product terms is

$$F_1 = (AB + AC + BC)'$$

and

$$F_2 = AB + AC + A'B'C'$$

This combination gives four distinct product terms: AB, AC, BC, and $A'B'C'$. The PLA programming table for the combination is shown in the figure. Note that output F_1 is the true output, even though a C is marked over it in the table. This is because F_1 is generated with an AND–OR circuit and is available at the output of the OR gate. The XOR gate complements the function to produce the true F_1 output.

PLA programming table

	Product term	Inputs A	B	C	Outputs (C) F_1	(T) F_2
AB	1	1	1	–	1	1
AC	2	1	–	1	1	1
BC	3	–	1	1	1	–
$A'B'C'$	4	0	0	0	–	1

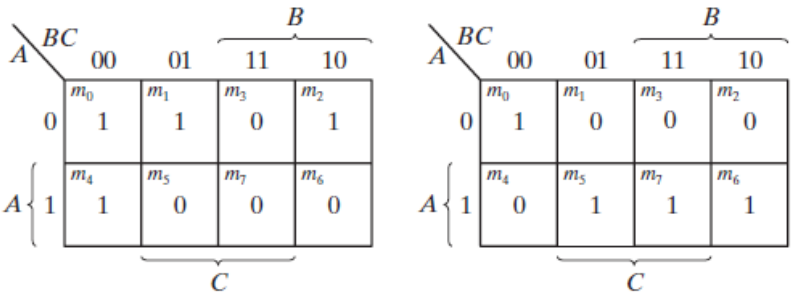


FIGURE 7.15
Solution to Example 7.2

[1] Mask programmable PLA

[2] Field-Programmable Logic Array

منطق آرایه‌ای برنامه‌پذیر (PAL)

در مقایسه با PLA، برنامه‌ریزی PAL ساده‌تر است (زیرا تنها گیت‌های AND نیاز به برنامه‌ریزی دارند) اما قابلیت انعطاف آن کمتر است.

آرایش منطقی یک نمونه PAL در شکل زیر نشان داده شده است. ملاحظه می‌شود:

- تعداد گیت‌های AND به ازاء هر تابع بولی خروجی، محدود است.
- از یکی از توابع بولیِ خروجی می‌توان به عنوان یک جمله‌ی حاصلضرب در ساخت دیگر توابع بولی استفاده کرد.
- به دلیل محدود بودن تعداد گیت‌های AND بای تا حد امکان، توابع بول را ساده‌سازی کرد.
- در این جا (برخلاف PLA) نمی‌توان یک جمله‌ی حاصلضرب را بین دو یا چند OR به اشتراک گذاشت؛ بنابراین، هر تابع بدون توجه به وجود اشتراک در جملات ضرب باید ساده شود.
- تعداد جملات ضرب در هر بخش ثابت است؛ بنابراین اگر تعداد جملات در تابع زیاد باشد، برای پیاده‌سازی آن تابع ممکن است از دو بخش استفاده شود (وجود فیدبک در شکل اخیر برای تحقق همین منظور است). در این باره به مثال بعدی توجه کنید.

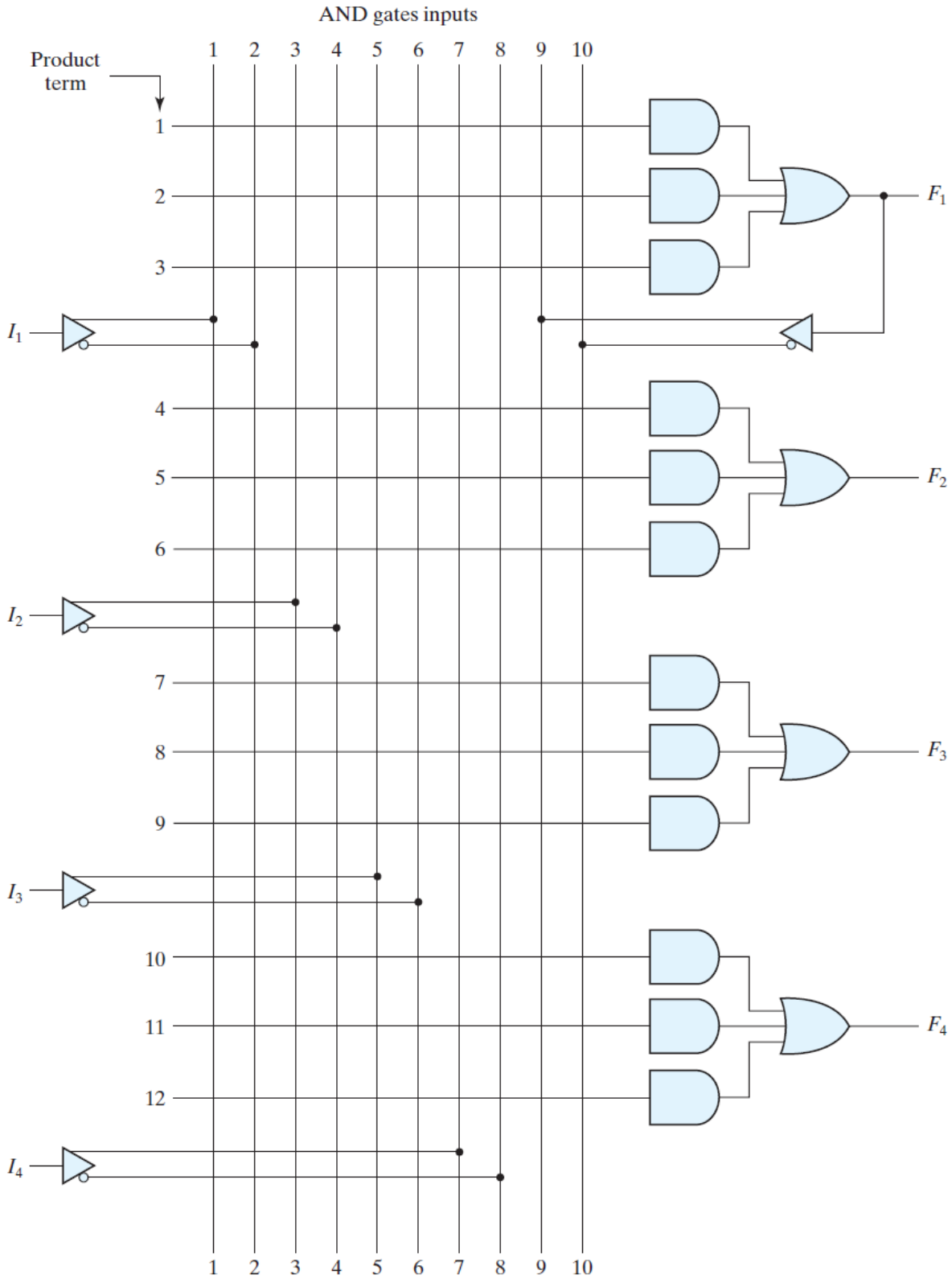


FIGURE 7.16
PAL with four inputs, four outputs, and a three-wide AND–OR structure

مثال: فرض کنید بخواهیم توابع زیر را روی PAL نشان داده شده در شکل اخیر پیاده‌سازی کنیم.

$$w(A, B, C, D) = \sum(2, 12, 13)$$
$$x(A, B, C, D) = \sum(7, 8, 9, 10, 11, 12, 13, 14, 15)$$
$$y(A, B, C, D) = \sum(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$
$$z(A, B, C, D) = \sum(1, 2, 8, 12, 13)$$

برای این منظور، ابتدا توابع را با کمک جدول کارنو ساده‌سازی می‌کنیم. در این ساده‌سازی، توابعی را که بیش از سه جمله‌ی حاصلضرب دارند، برحسب دیگر توابع بیان می‌کنیم:

$$w = ABC' + A'B'CD'$$
$$x = A + BCD$$
$$y = A'B + CD + B'D'$$
$$z = ABC' + A'B'CD' + AC'D' + A'B'C'D$$
$$= w + AC'D' + A'B'C'D$$

جدول برنامه‌ریزی PAL به صورت زیر می‌شود.

Table 7.6
PAL Programming Table

Product Term	AND Inputs A	B	C	D	w	Outputs
1	1	1	0	—	—	$w = ABC' + A'B'CD'$
2	0	0	1	0	—	
3	—	—	—	—	—	
4	1	—	—	—	—	$x = A + BCD$
5	—	1	1	1	—	
6	—	—	—	—	—	
7	0	1	—	—	—	$y = A'B + CD + B'D'$
8	—	—	1	1	—	
9	—	0	—	0	—	
10	—	—	—	—	1	$z = w + AC'D' + A'B'C'D$
11	1	—	0	0	—	
12	0	0	0	1	—	

نقشه‌ی فیوزها (دیاگرام منطقی داخلیِ PAL پس از برنامه‌ریزی) به صورت شکل زیر می‌شود. در این شکل، گیت‌های AND که داخل آنها علامت × گذاشته شده است، به این معنا است که باید آنها را بدون تغییری رها کنیم (تا در خروجی آنها صفر تولید شود).

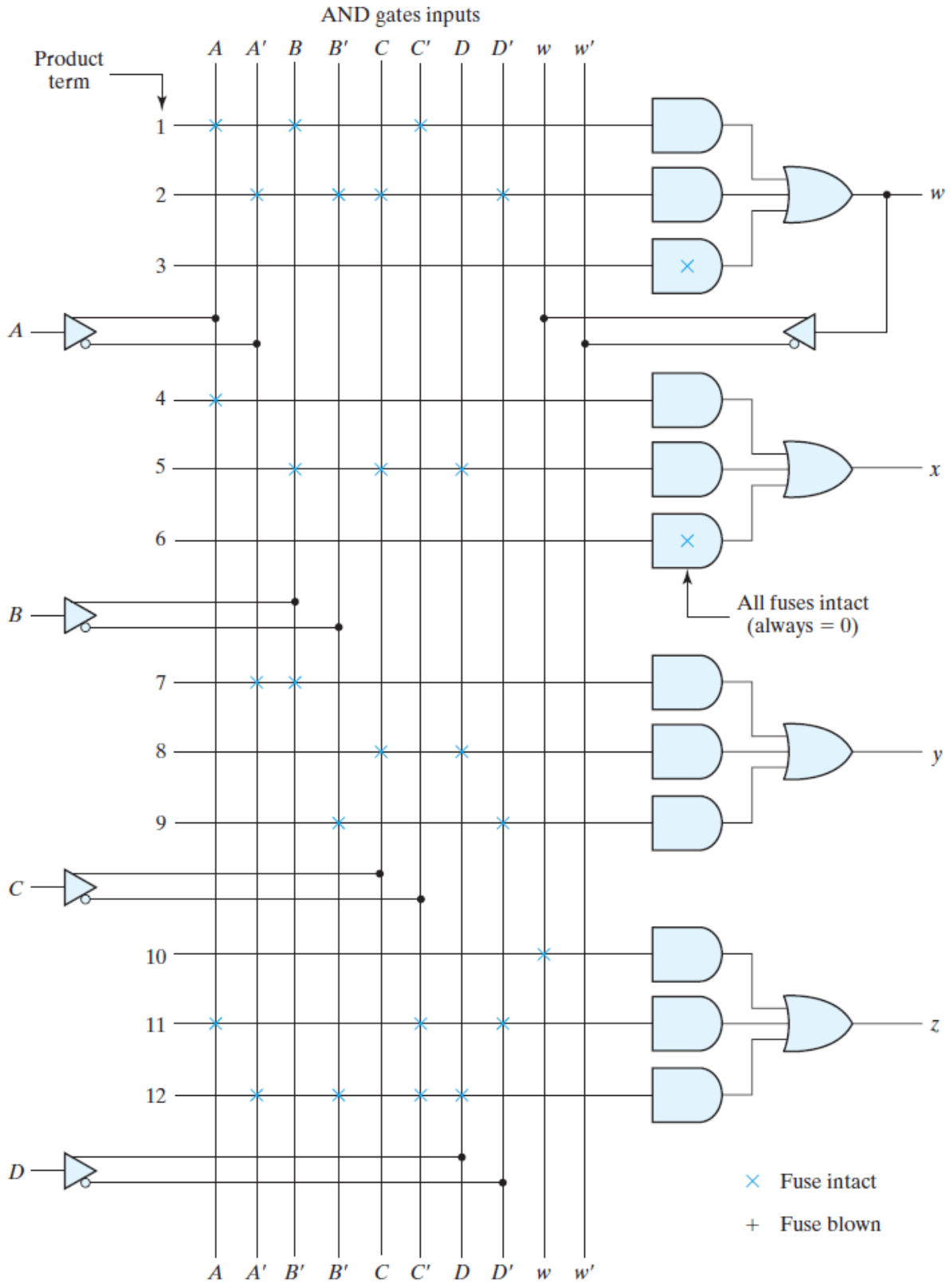


FIGURE 7.17
Fuse map for PAL as specified in Table 7.6

وسایل برنامه‌پذیر ترتیبی

سیستم‌های دیجیتال با استفاده از فلیپ‌فلاپ‌ها و گیت‌ها طراحی می‌شوند. PLDهایی که تاکنون دیدیم تنها ترکیبی بوده و از گیت‌ها ساخته می‌شدند. در مقابل، وسایل برنامه‌پذیر ترتیبی (SPLD[1]) شامل هر دو نوع مدارات گیتی و فلیپ فلاپ بوده و بنابراین، قابل استفاده جهت پیاده‌سازی انواع توابع ترتیبی هستند.

چندین نوع SPLD تجاری وجود دارد که سه نوع اصلی آنها عبارتند از:

- وسیله‌ی منطقیِ برنامه‌پذیر ترتیبی یا ساده (SPLD)
- وسیله‌ی منطقیِ برنامه‌پذیر پیچیده (CPLD[2])
- آرایه‌ی گیتیِ برنامه‌پذیر در محل (FPGA[3])

PLD ترتیبی را گاهاً PLD ساده نیز می‌گویند تا آن را از نوع پیچیده تفکیک کنند.

SPLD علاوه بر آرایه‌ی AND-OR شامل فلیپ فلاپ نیز می‌باشد. بنابراین، طرح ساختار داخلی آن نتیجه‌ی اصلاح ساختار داخلیِ یک PLA یا PAL و به صورت زیر است:

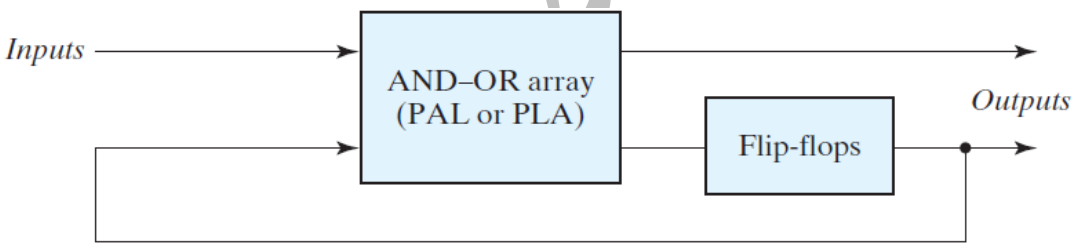


FIGURE 7.18
Sequential programmable logic device

خروجی‌های مدار را از گیت‌های OR یا از خروجی‌های فلیپ فلاپ‌ها می‌توان گرفت. امکان استفاده از خروجی‌های فلیپ‌فلاپ‌ها در ساخت جملات حاصضربی نیز وجود دارد. هر بخش از یک SPLD را یک «ماکروسل[4]» می‌گویند. ماکروسل مداری است که شامل تابع ترکیبیِ جمع حاصل‌ضرب‌ها و یک فلیپ فلاپ اختیاری است.

نمودار منطقیِ یک ماکروسل پایه در شکل زیر نشان داده شده است.

[1] Sequential (or Simple) PLD
[2] Complex PLD
[3] Field Programmable Gate Array
[4] Macrocell

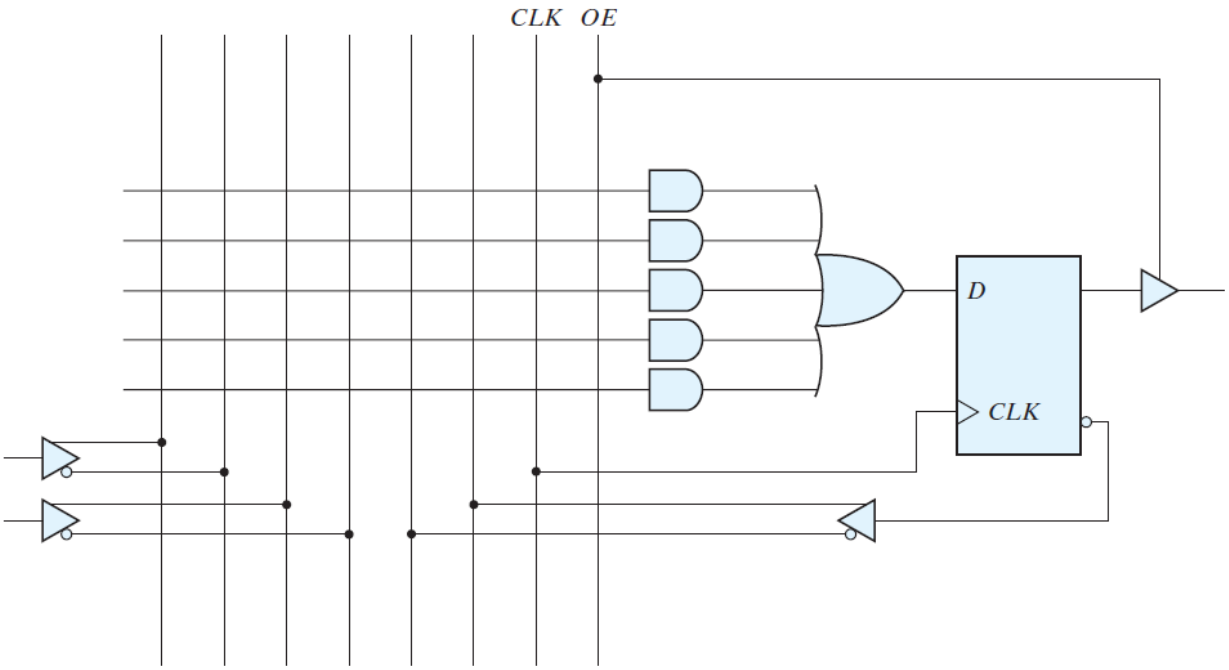


FIGURE 7.19
Basic macrocell logic

آرایه‌ی AND-OR مشابه با آن چه در مورد یک PAL دیدیم، است. خروجی توسط یک فلیپ فلاپ D حساس به لبه درایو/راه‌اندازی می‌شود. این فلیپ فلاپ به یک کلاک مشترک متصل شده است و حالت آن در هر لبه‌ی کلاک تغییر می‌کند. خروجیِ فلیپ فلاپ به یک بافر (یا وارونگرِ) سه‌حالته متصل شده که توسط سیگنال فعال‌ساز خروجی (OE) کنترل می‌شود. خروجی فلیپ فلاپ به یکی از ورودی‌های گیت‌های AND برنامه‌پذیر برای تهیه‌ی حالت فعلیِ مدار ترتیبی پسخورد (فیدبک) شده است. SPLDها نوعاً دارای ۸ الی ۱۰ ماکروسل در هر بسته‌ی آی‌سی هستند. همه‌ی فلیپ‌فلاپ‌ها به یک ورودی کلاک مشترک متصل‌اند و تمام بافرهای سه‌حالته توسط ورودیِ OE کنترل می‌شوند.

یک ماکروسل، علاوه بر امکان برنامه‌ریزی آرایه‌ی AND، دارای امکانات برنامه‌ریزی دیگری است که متداولترین این امکانات عبارتند از: امکان استفاده یا نادیده‌گرفتن فلیپ‌فلاپ، انتخاب لبه‌ی کلاک، انتخاب (از بینِ) پاک‌کردن و پیش‌تنظیم برای ثبات، و امکان خروجیِ صحیح یا متمم آن. با کمک مالتی‌پلکسرها و برنامه‌ریزی ورودی‌های انتخاب آنها می توان برای انتخاب دو یا چهار مسیر مجزا استفاده کرد.

یک وسیله‌ی منطقیِ برنامه‌پذیر پیچیده (CPLD) مجموعه‌ای از PLDهای منفرد روی یک مدار مجتمع است. با بهره‌گیری از یک ساختار اتصالات برنامه‌پذیر[1]، امکان ارتباط PLDها با یکدیگر، درست همانند ارتباط درونیِ هر PLD، فراهم آمده است. آرایش کلیِ یک CPLD در شکل زیر نشان داده شده است.

[1] Programmable interconnect structure

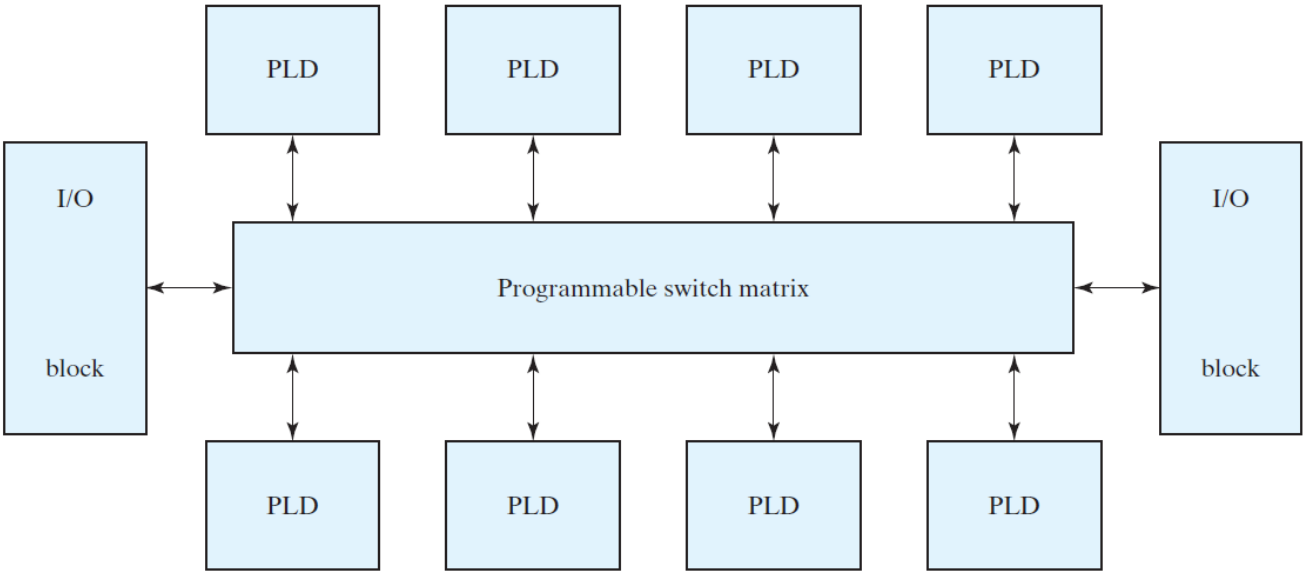


FIGURE 7.20
General CPLD configuration

در این طرح، یک ماتریس سوئیچ برنامه‌پذیر نقش همان ساختار اتصالات برنامه‌پذیر اشاره شده را برعهده دارد. در این طرح هر PLD نوعاً شامل ۸ الی ۱۶ ماکروسل است.

بلوک‌های ورودی/خروجی (I/O) ارتباط با پایه‌های آی سی را برقرار می‌کنند. هر پایه‌ی متعلق به بلوک I/O با یک بافر سه‌حالته درایو شده و می‌تواند به عنوان ورودی یا خروجی برنامه‌ریزی شود.

ماتریس سوئیچ ورودی‌ها را از بلوک I/O دریافت کرده و آن را به سوی ماکروسل خاصی هدایت می‌کند. به طور مشابه، خروجی‌های مورد نظر از یک ماکروسل قابل انتقال و هدایت به پایه‌های خروجی از بلوک I/O هستند.

ماکروسل‌های داخلیِ هر PLD به طور کامل با هم در ارتباط و اتصال هستند. اگر یک ماکروسل شامل جملات ضرب استفاده‌نشده‌ای باشد، می‌توان از آنها در ماکروسل‌های مجاور استفاده کرد.

برای یک CPLD معماری یا طرح‌های مختلفی می‌تواند وجود داشته باشد؛ مهمترین زمینه‌های تفاوت بین این معماری‌ها عبارتند از:

- PLDهای داخلی (که گاهاً به آنها بلوک‌های تابعی[1] نیز گفته می‌شود)،
- نوع ماکروسل،
- بلوک‌های I/O،
- ساختار اتصالات داخلیِ برنامه‌پذیر.

بهترین راه برای شناسایی یک وسیله‌ی خاص، مطالعه‌ی مقالات منتشر شده توسط سازنده‌ی آن وسیله است.

[1] Functional blocks

آرایه‌ی گیتیِ برنامه‌پذیر در محل (FPGA) یک مدار VLSI است که قابل برنامه‌ریزی در محل کاربر است. یک افزاره‌ی FPGA معمولاً شامل میلیون‌ها بلوک منطقی است که به وسیله‌ی بلووک‌های ورودی‌ـخروجی محصور شده و از طریق اتصالات برنامه‌پذیر قابل اتصال و ارتباط با یکدیگر هستند. در این گروه از افزاره‌ها، انواع بسیار متنوعی از پیکره‌بندی (آرایش)های داخلی وجود داشته و کارایی هر افزاره به مدارات موجود در بلوک‌های منطقی و نیز به راندمان اتصالات برنامه‌ریزی شده بستگی دارد.

یک بلوک منطقی در FPGA معمولاً شامل تعدادی جدول جستجو (LUT[1])، مالتی‌پلکسر، گیت و فلیپ‌فلاپ است. یک جدول جستجو در واقع، یک جدول درستی است که در یک SRAM ذخیره شده و توابع مدار ترکیبی را برای بلوک منطقی فراهم می‌کند. این توابع از جدول درستی ذخیره شده در SRAM و مشابه با روش پیاده‌سازی مدارهای ترکیبی با ROM، تحقق می‌یابند. برای مثال، یک RAM با اندازه‌ی 16×2 می‌تواند جدول درستی یک مدار ترکیبی با چهار ورودی و دو خروجی را ذخیره نماید. بخش منطق ترکیبی همراه با تعدادی مالتی‌پلکسر برنامه‌پذیر جهت ایجاد معادلات ورودی مورد نظر برای فلیپ‌فلاپ و نیز تامین خروجیِ مورد نظر برای بلوک منطقی استفاده می‌شود.

مزیت استفاده از RAM به جای ROM در ذخیره‌ی جدول درستی این است که جدول درستی از طریق نوشتن در حافظه قابل برنامه‌ریزی است. عیب آن نیز این است که این نوع حافظه، فرّار بوده و هنگام قطع برق باید محتوای جداول جستجو را مجدداً بارگذاری کرد. برنامه‌ی لازم (جهت بارگذاری مجدد) را می‌توان به کمک یک کامپیوتر مرکزی یا یک PROM موجود روی بُرد بارگذاری کرد. پس از بارگذاری، این برنامه تا زمانی که برنامه‌ی جدیدی بخواهد روی FPGA بارگذاری مجدد شود یا این که برق SRAM قطع شود، پایدار و باقی خواهد ماند. قابلیت برنامه‌ریزی مجددِ FPGA امکان عملی شدن و تحققِ کاربردهای زیادی را فراهم می سازد.

FPGAهای شرکت Xilinx

اولین FPGAهای تجاری جهان را شرکت Xilinx (بخوانید «زایلینکس») در سال ۱۹۸۵ و تحت عنوان «نسل XC2000» به بازار عرضه کرد. پس از آن، نسل‌های XC3000 و XC4000 عرضه شدند که مبنای خانواده‌های معروف و شناخته شده‌ی Spartan و Virtex قرار گرفتند. تفاوت بین هر کدام از این نسل‌ها و خانواده‌ها، بهبودی بود که در زمینه‌هایی مانند چگالی، کارایی، مصرف توان، سطوح ولتاژ، تعداد پایه‌ها (یا«پین‌ها[2]»)، و عملکرد رخ می‌داد. برای مثال، خانواده‌ی Spartan در ابتدای کار با ظرفیت حداکثر ۴۰ کیلو گیت شروع کرد اما امروزه خانواده‌ی Spartan-6 به راحتی شامل ۱۵۰ هزار سلول منطقی به همراهِ ۴/۸ مگابیت حافظه‌ی RAM است.

[1] Lookup table

[2] Pin

معماری پایه در Xilinx

معماری پایه در خانواده‌ی Spartan و خانواده‌های قبل‌تر، مطابق با شکل زیر، شامل آرایه‌ای از بلوک‌های منطقیِ قابل پیکره‌بندی (CLB[1])، انواع منابع/امکاناتِ مسیردهی[2] محلی و سرتاسری[3]، انواع بلوک‌های ورودی‌ـ‌خروجی (IOB[4])، بافرهای ورودی‌ـ‌خروجی قابل برنامه‌ریزی[5]، و حافظه‌ی پیکره‌بندیِ مبتنی بر SRAM است.

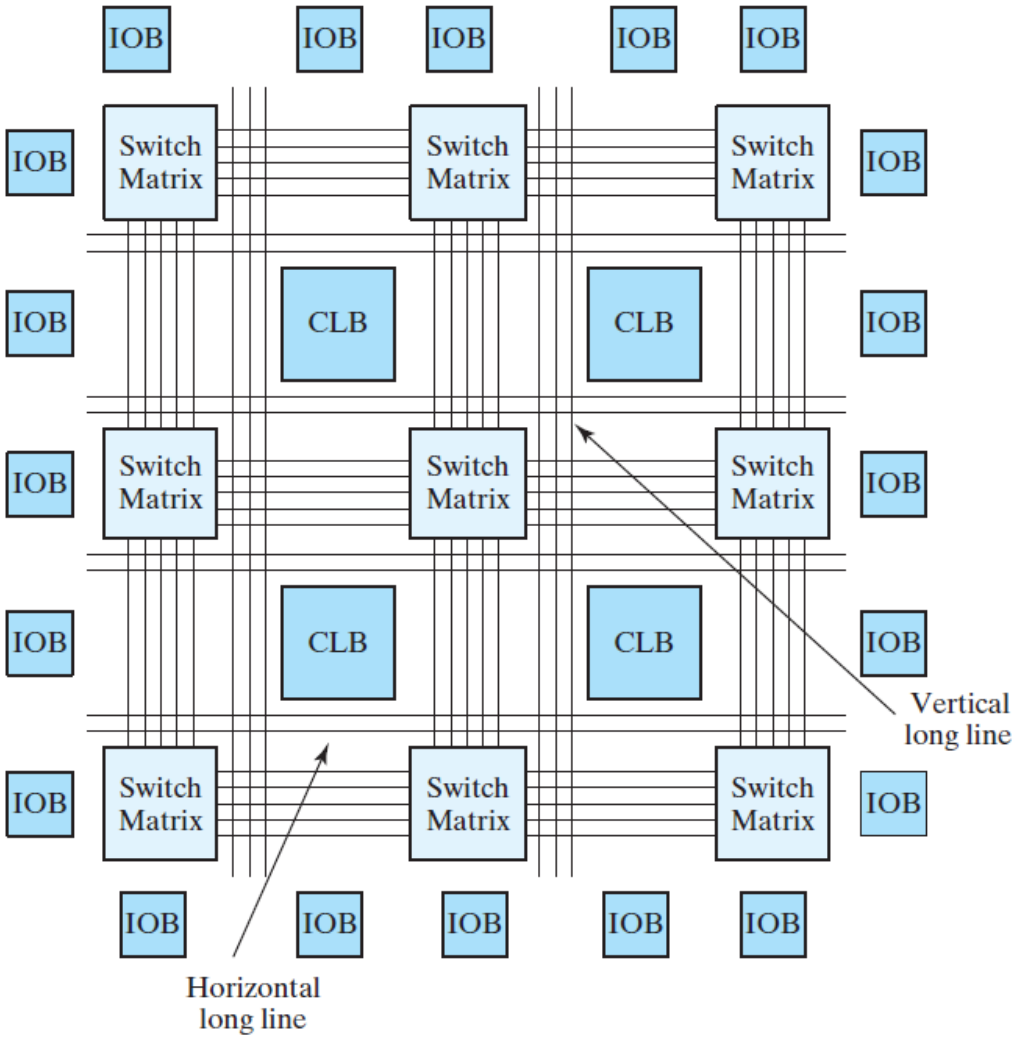


FIGURE 7.21
Basic architecture of Xilinx Spartan and predecessor devices

[1] Configurable Logic Block
[2] Routing resources
[3] Global
[4] Input-Output Blocks
[5] Programmable I/O buffers

بلوک منطقیِ قابل پیکره‌بندی (CLB)

هر CLB، مطابق با شکل زیر، متشکل از یک جدول جستجوی قابل برنامه‌ریزی، تعدادی مالتی پلکسر، تعدادی ثبات، و تعدادی مسیر برای سیگنال‌های کنترلی است.

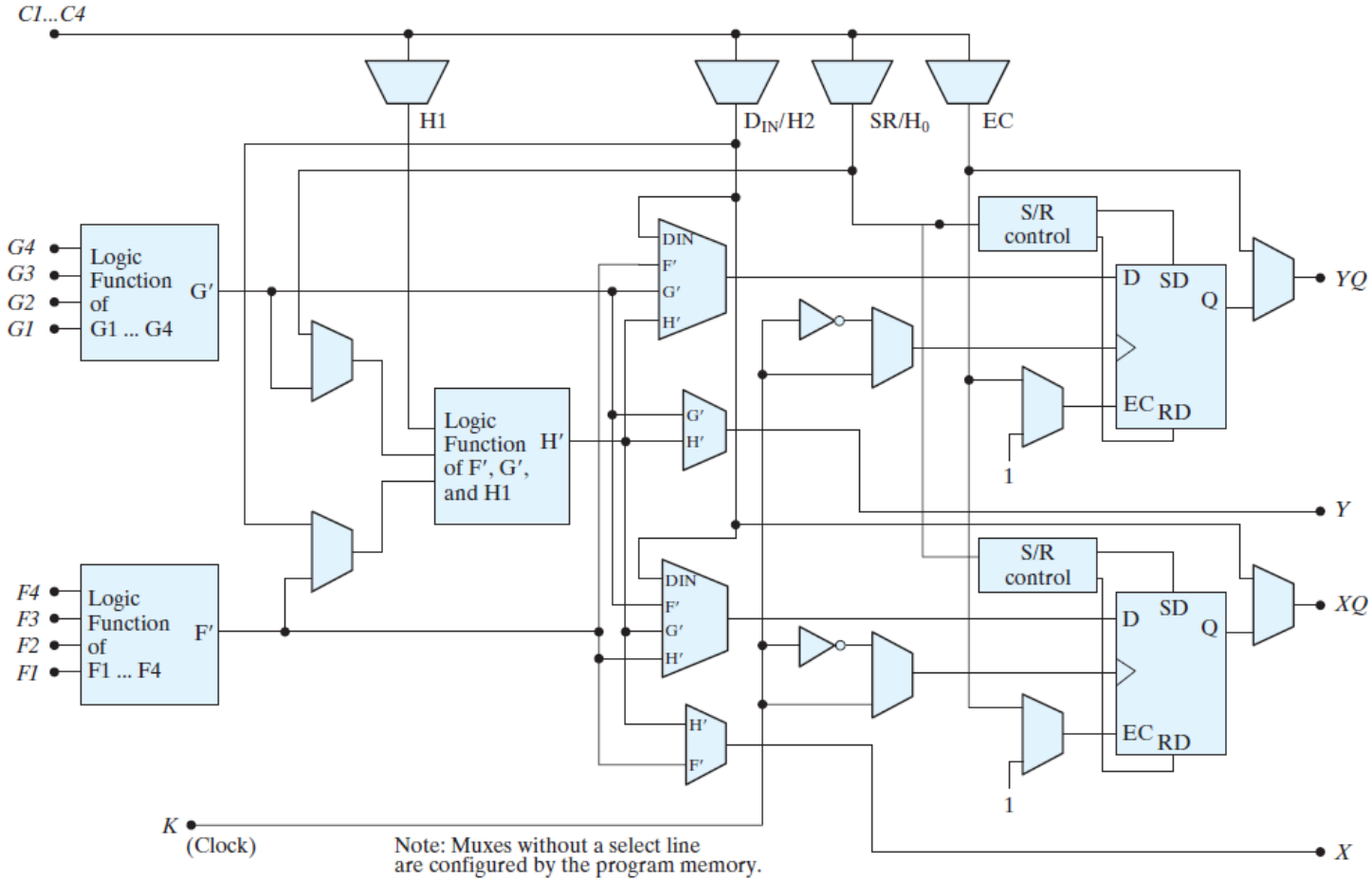


FIGURE 7.22
CLB architecture

دو مولد تابعِ (F و G) مبتنی بر جدول جستجو، قادر به تولید هر تابع دلخواهِ چهار ورودی هستند؛ تابع مولدِ H نیز قادر است هر تابع بولیِ سه‌متغیره‌ی دلخواه را تولید کند. بلوک تابعی H می‌تواند ورودی‌هایش را از جداول جستجوی F و G یا از ورودی‌های بیرونی دریافت کند. سه مولد تابع مزبور را می‌توان برای تولید موارد زیر برنامه‌ریزی کرد:

(۱) سه تابع متفاوت از سه مجموعه‌ی مستقل از ورودی‌ها (دو تا دارای چهار ورودی و یکی دارای سه ورودی ـ یک تابع باید داخلِ CLB رجیستر شود)،

(۲) یک تابع دلخواه پنج متغیره،

(۳) یک تابع دلخواه چهارمتغیره همراه با چند تابع شش‌متغیره،

(۴) چند تابع نُه متغیره.

هر CLB دارای دو وسیله‌ی ذخیره‌سازی است که می‌توانند به صورت فلیپ فلاپ‌های حساس به لبه با کلاک مشترک، یا، در نسل XC4000X، می‌توانند به صورت فلیپ فلاپ یا لچ شفاف با کلاک مشترک (قابل برنامه‌ریزی برای لبه بوده و جداگانه وارون شدنی‌اند) و فعال‌ساز مشترک برنامه‌ریزی شوند. عناصر ذخیره‌ساز می‌توانند ورودی‌شان را از مولدهای تابع یا از ورودیِ D_{in} بگیرند. عناصر دیگر می‌توانند یک ورودی خارجی از ورودیِ H1 دریافت کنند. مولدهای تابع علاوه بر این قادرند دو خروجی (X و Y) را مستقیماً و مستقل از خروجی‌های عناصر ذخیره‌ساز درایو کنند. همه‌ی این خروجی‌ها قابل اتصال به شبکه‌ی اتصال[1] هستند. عناصر ذخیره‌ساز به وسیله‌ی یک ست/ریست سرتاسری در زمان روشن شدن راه‌اندازی و درایو می‌شوند. این ست/ریست سرتاسری به گونه‌ای برنامه‌ریزی می‌شود که منطبق بر نیازمندیِ برنامه‌ریزیِ کنترل S/R محلی مربوط به هر عنصر ذخیره‌ساز شود.

RAM توزیع شده[2]

سه مولد تابع موجود در یک CLB می‌توان به عنوان یک حافظه‌ی RAM دو درگاهیِ[3] 16×2 یا یک حافظه‌ی RAM یک‌درگاهیِ 32×1 استفاده کرد. افزاره‌های XC4000 دارای RAM بلوکی نیستند اما در این افزاره‌ها، گروهی از CLBها می‌توانند آرایه‌ای از حافظه ایجاد کنند. افزاره‌های متعلق به خانواده‌ی Spartan علاوه بر RAM توزیع شده، دارای RAM بلوکی نیز می‌باشند.

منابع اتصال[4]

شبکه‌ای از ماتریس‌های سوئیچی[5] (یعنی ماتریسی از سوئیچ‌ها) به منظور تحقق بخشی از معماریِ CLB که مربوط به اتصال همه‌منظوره جهت تامین انشعابات و مسیردهی در سرتاسر افزاره است، استفاده شده‌اند. سه نوع اتصال همه‌منظوره وجود دارد: خطوط تک‌ـطول[6]، خطوط جفت‌ـطول[7]، و خطوط طولانی. شبکه‌ای از خطوط تک‌طول افقی و عمودی آرایه‌ای از جعبه سوئیچ ها را به هم متصل می‌کنند؛ هر جعبه سوئیچ در داخل خود تعداد کاهش‌یافته‌ای از اتصالات بین مسیرهای سیگنال را فراهم می‌کند. هر CLB دارای یک جفت بافر سه‌حالته است که می توانند سیگنال را به داخل نزدیک‌ترین خطوط افقی در بالا یا پایینِ CLB درایو کنند.

[1] Interconnect network
[2] Distributed RAM
[3] Dual-port RAM
[4] Interconnect resources
[5] Switch matrices
[6] Single-length lines
[7] Double-length lines

بلوک ورودی ـ خروجی (IOB)

هر پایه (پینِ) قابل برنامه‌ریزیِ I/O دارای یک بلوک IOB قابل برنامه‌ریزی برای خود است که این بلوک دارای بافرهایی به منظور تطابق و سازگاری با استانداردهای سطح سیگنال TTL و CMOS است. شکل زیر طرح ساده شده‌ای از یک IOB قابل برنامه‌ریزی را نشان می‌دهد. از این بلوک می‌توان به عنوان یک ورودی، یا یک خروجی، و یا یک درگاه دوجهته (پورت) استفاده کرد.

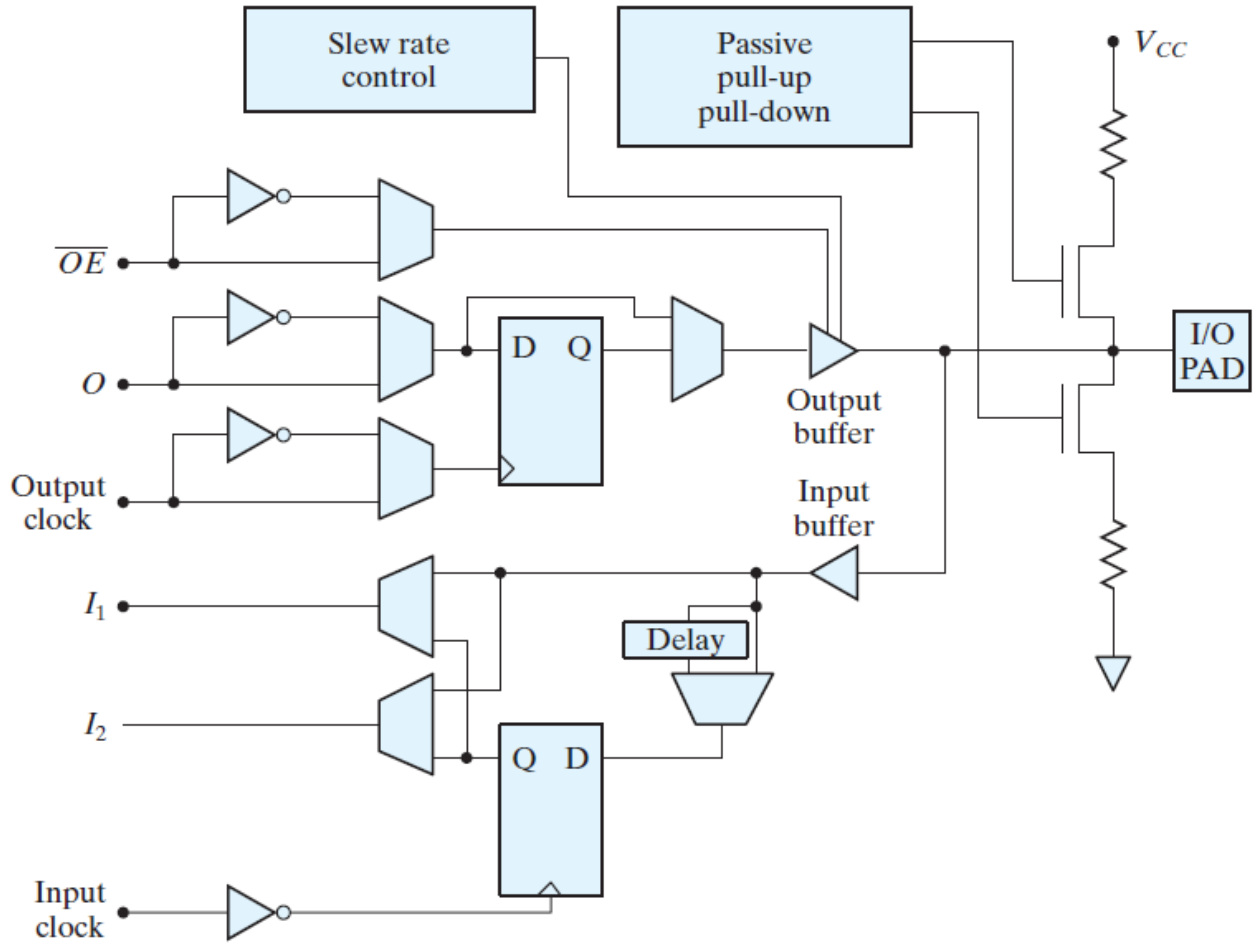


FIGURE 7.25
XC4000 series IOB

هر IOB که به عنوان یک ورودی پیکره‌بندی (آرایش) شود دارای امکان ورودی مستقیم، از لچ، و یا از ثبات است. در آرایش خروجی، IOB دارای دو حالت خروجیِ مستقیم و از طریقِ ثبات است. بافر خروجیِ یک IOB دارای امکان کنترل تغییر جهت (Slew) و انحراف (Skew) است. ثباتهای موجود در مسیرهای ورودی و خروجی یک IOB، توسط کلاک‌ها مجزا و قابل وارون‌شدن درایو می‌شوند. یک ست/ریست سرتاسری نیز موجود است.

عناصر تاخیر داخلی تاثیر حاصل از عبور سیگنال کلاک از یک بافر سرتاسری را قبل از رسیدن به یک بلوک IOB جبران می‌کنند. این تدبیر و استراتژی، نیاز به لزوم رعایت نگهداریِ (Hold) داده روی یک پین خارجیِ افزاره حذف می‌کند.

خروجیِ سه‌حالته از یک IOB، بافر خروجی را در حالت امپدانس ـ بالا قرار می‌دهد. خروجی و فعال‌ساز خروجی را می‌توان معکوس کرد. میزان تغییر جهت بافر خروجی قابل کنترل است تا بدین ترتیب بتوان حالت گذر را روی گذرگاهِ تغذیه، زمانی که سیگنال های غیربحرانی (نه چندان مهم) در حال سوئیچ کردن هستند، حداقل کرد. پینِ IOB را می‌توان به صورت بالاکش (pull-up) یا پایین‌کش (pull-down) برنامه‌ریزی کرد تا از مصرفِ بی‌جهت تغذیه و نیز نویز جلوگیری کرد.

موارد ارتقاء

تراشه‌های Spartan قادر به گنجاندن هسته‌های نرم‌افزاری در خود بوده و انواع حافظه‌ی RAM درون‌تراشه‌ای[1] موجود در آنها (SelectRAM) اعم از حافظه‌ی توزیع شده، حافظه‌ی دودرگاهی، و حافظه‌ی سنکرون را می‌توان برای پیاده‌سازی فایلهای ثباتیِ FIFO، شیفت‌رجیسترها، و حافظه‌های موقت به کار برد. این بلوک‌های حافظه‌ای را با هر عرض و اندازه‌ی دلخواهی می‌توان با یکدیگر کسکود کرده و در هر جای دلخواهی از افزاره قرار داد؛ گرچه، این کار باعث می‌شود دسترسی به CLBها برای پیاده‌سازی مدارات منطقی محدود شود. شکل زیر ساختار یک RAM درون‌تراشه‌ای را که از طریق برنامه‌ریزی یک جدول جستجو به منظور پیاده‌سازی حافظه‌ی RAM تک‌درگاهی با قابلیت نوشتن همزمان و خواندن غیرهمزمان (آسنکرون) ایجاد شده است، نشان می‌دهد. هر CLB را می‌توان با برنامه‌ریزی به یک حافظه‌ی RAM 16×2 یا 32×1 تبدیل کرد.

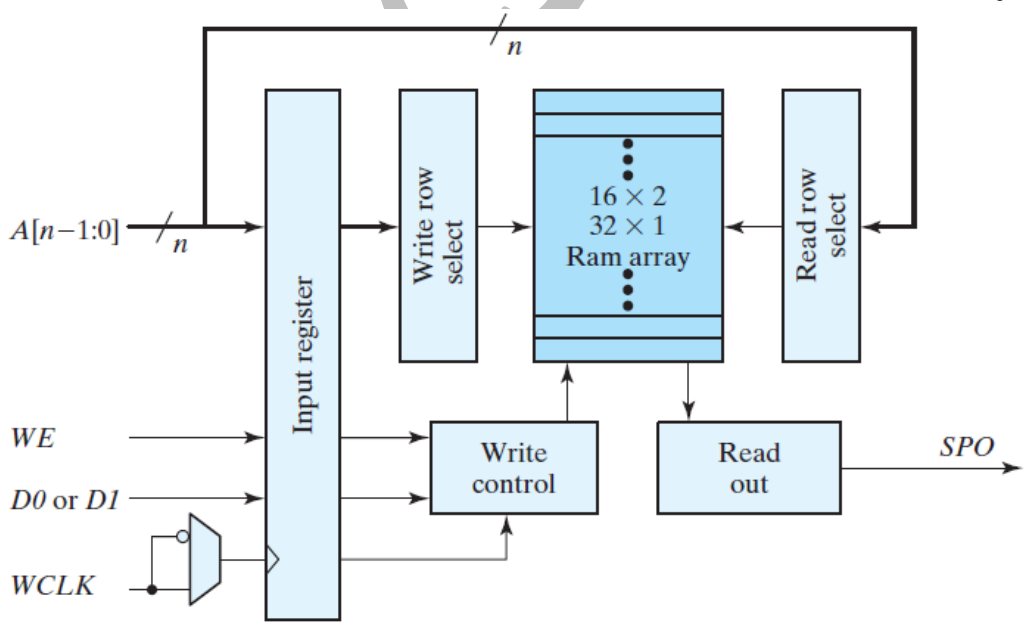


FIGURE 7.26
Distributed RAM cell formed from a lookup table

[1] On-chip

برای ساخت RAMهای دودرگاهی در افزاره‌های Spartan می‌توان از ساختار نشان داده شده در شکل زیر بهره گرفت؛ این طرح دارای یک پورت نوشتن مشترک و دو پورت خواندن غیرهمزمان (آسنکرون) است. یک CLB قادر به تشکیل یک حافظه با اندازه‌ی حداکثر 16×1 است.

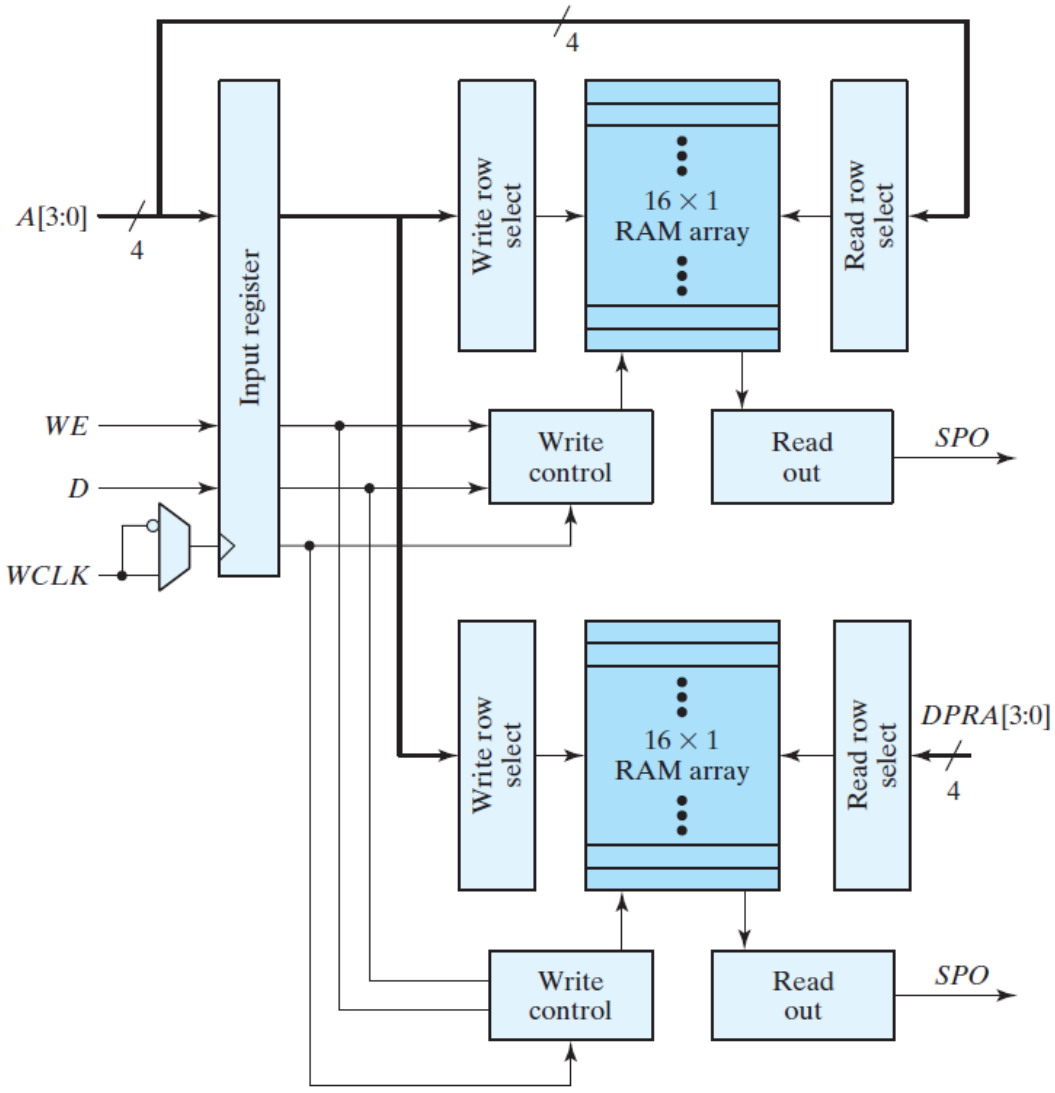


FIGURE 7.27
Spartan dual-port RAM

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

صلوات