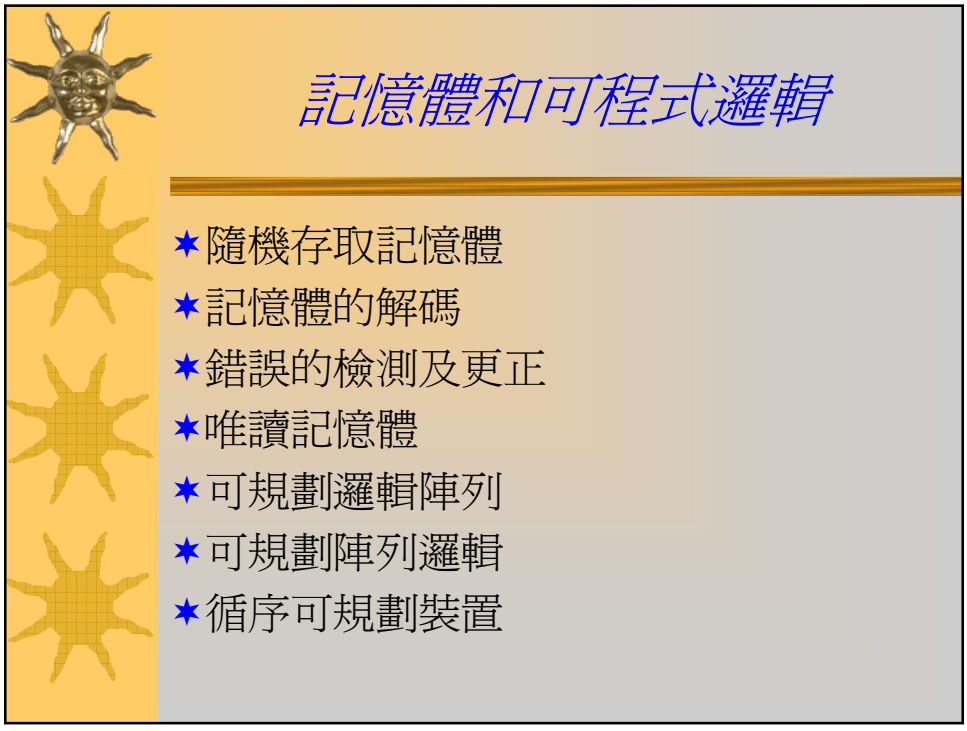

第七章
記憶體和可程式邏輯


記憶體和可程式邏輯
隨機存取記憶體
記憶體的解碼
錯誤的檢測及更正
唯讀記憶體
可規劃邏輯陣列
可規劃陣列邏輯
循序可規劃裝置

記憶體

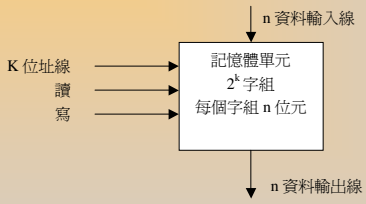
- 何謂記憶體
  - 儲存格的集合，用來儲存大量二元資訊的一種裝置。
- 記憶體的種類
  - 隨機存取記憶體(random access memory ；RAM) 。
  - 唯讀記憶體(read-only memory； ROM) 。
- 記憶體寫入(write)操作
  - 儲存一個新的資訊至記憶體內的過程。
- 記憶體讀取(read)操作
  - 將儲存在記憶體內的資訊轉移出記憶體的過程。

隨機存取記憶體

- 何謂隨機存取記憶體
  - 由任意位置做轉移資訊進出的時間都是相同的。
- 記憶體儲存單位–位元組（byte)
  - 位元組（byte)：8位元。
  - 字組長度：8位元的整數倍。
  - 字組：可爲數字、指令、字符。
- 記憶體單元容量–儲存的位元組總數。

記憶體單元的方塊圖

- 記憶體單元方塊圖



k位址線：選取記憶體中的某個字組

讀、寫：指定資料轉移方向

n資料輸入線：提供資訊儲存於記憶體內

n資料輸出線：由記憶體提供資訊出來

記憶體容量

- 範圍
  - 2^{10}~2^{32}個字組
- 字組單位
  - K(千)=2^{10}、 M(百萬) =2^{20}、 G(十億) =2^{30} 。
  - 64K=2^{16} 、2M=2^{21} 、4G=2^{32} 。
- 記憶體1K x 16
  - 有10個位元的位址，以16位元爲一單位來操作。
- 記憶體的位址位元數目
  - 視記憶體字組總數而定，$2k \geq m$ ， 與每一字組內的位元數目無關。

寫入與讀取操作

- 寫入：字組儲存於記憶體
  - 將所需字組的二進位位址轉移至位址線。
  - 將要儲存的資料位元轉移至資料輸入線。
  - 啓動寫入輸入。
- 讀取：儲存在記憶體內的字組轉移出來
  - 將所需字組的二進位位址轉移至位址線。
  - 啓動讀取輸入。

記憶晶片的控制輸入

- 記憶體致能不動作
  - 記憶體晶片不被啓動，因此不執行任何動作。
- 記憶體致能動作
  - 讀取/寫入可決定所執行的操作。

記憶體致能	讀取/寫入	記憶體操作
0	x	不動作
1	0	寫入至選取字組
1	1	從選取字組讀取

用 HDL描述記憶體

- 記憶體的宣告
  - 用一個二維陣列且用reg這個保留字，
    - 陣列中第一個數代表字組中的位元數
    - 陣列中第二個數代表記憶體內的總字組數
    - 如 reg [15:0] memmord [0:1023];
- 讀取操作指令
  - DataOut ← Mem[Address]
- 寫入操作指令
  - Mem[Address] ← DataIn

HDL範例 7-1

例題

```
//Read and write operations of memory.
//Memory size is 64 words of 4 bits each.
module memory (Enable, ReadWrite, Address, DataIn , DataOut) ;
      input Enable, ReadWrite ;
      input [3:0]   DataIn ;
      input [5:0]   Address ;
      output [3:0]   , DataOut ;
      reg [3:0]   DataOut ;
      reg [3:0]   Mem [0:63] ;      //64 x 4 memory
      always @   (Enable or ReadWrite)
          if   (Enable)
          if   (ReadWrite)
          DataOut = Mem [Address] ;      //read
          else
          Mem [Address] = DataIn ;       //Write
          else   DataOut = 4'bz ;          // High impedance state
endmodule
```

時序圖

- 存取時間(access time)
  - 選取一個字組然後讀取它所花的時間
- 週期時間(cycle time)
  - 完成寫入操作所發的時間
- 存取時間、週期時間
  - 等於數個時脈週期
- 例題：圖7－4記憶體週期時序圖

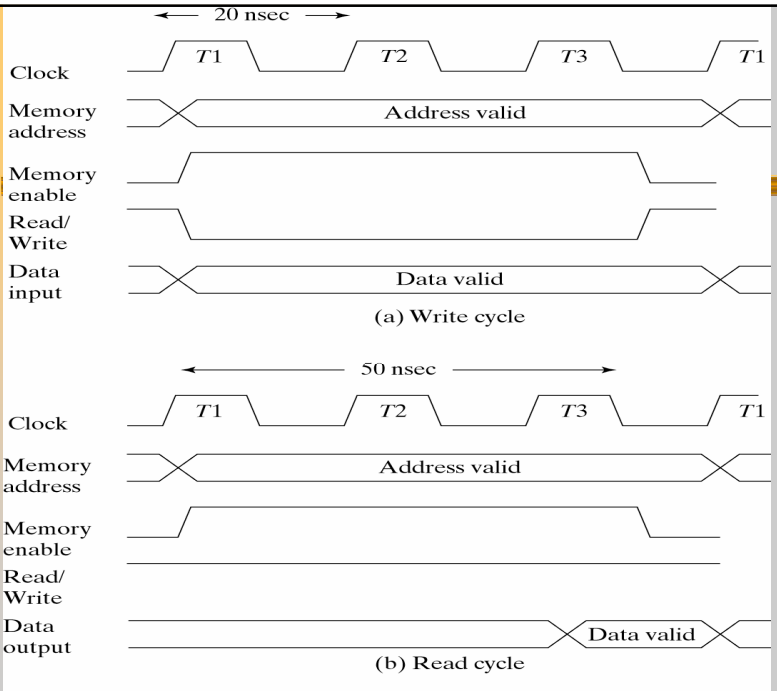


Fig. 7-4 Memory Cycle Timing Waveforms

記憶體的種類

- 記憶體存取模式
  - 隨機存取記憶體-揮發性記憶體
    - 不管存取位置爲何，存取時間相同
    - 靜態RAM(SRAM) -包含內部閂鎖以儲存二進位資訊 ，靜態RAM則較易使用且其讀寫週期較短。
    - 動態RAM(D RAM) -電荷加在電容器上的形式來儲存二進位資訊，功率消耗較少，且在單一記憶體晶片內提供較大的儲存容量
  - 循序存取記憶體-非揮發性記憶體
    - 存取時間視字組相對於讀取磁頭的位置，存取時間不同

記憶體的解碼

- 隨機存取記憶體的內部構造及說明其解碼器的操作。
- 解碼器
  - 選取由輸入位址所指定的記憶體字組。
- 一個安排二維陣列技巧的解碼器可展示出更有效率的解碼構造的例子，它被使用在大容量的記憶體 。
- 例子 -通常被應用在DRAM積體電路中的位址多工 。

儲存格

★儲存一位元資訊的二進位儲存格

– 二進位儲存格必須非常的小，內部閂鎖可儲存一個位元。

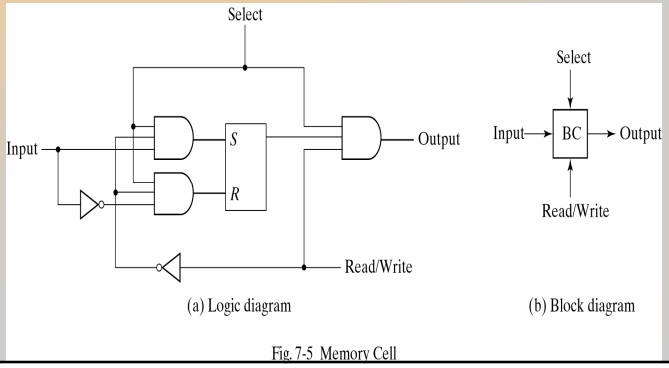


Fig. 7-5 Memory Cell

4X4 RAM方塊圖

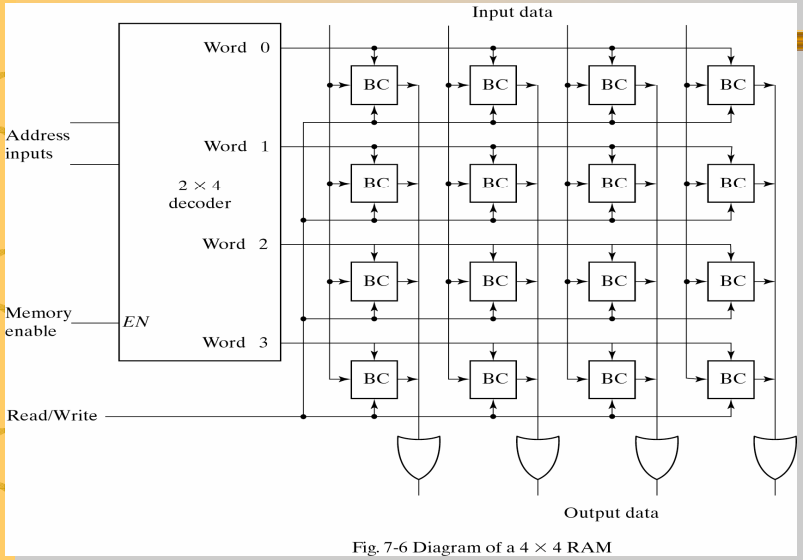


Fig. 7-6 Diagram of a 4 × 4 RAM

4X4 RAM內部構造

- 2x4解碼器
  - 選取此四個字組的其中，一個用記憶體啓動輸入來啓動。
- 讀取操作期間
  - 所選取字組的四個位元即經由OR閘通至輸出端 。
- 在寫入操作期間
  - 出現在輸入線上的可用資料被轉移到所選取字組的四個二進位儲存格 。

二維解碼

- 一個陣列內安排記憶體儲存格盡可能的讓它接近正方形。在此架構下，一個具有k條輸入的解碼器就可以被兩個只有k/2條輸入的解碼器所取代，其中一個解碼器是用來做二維矩陣架構中列的選取，而另一個則是做行的選取。
- 1k-字組記憶體使用二維選取方法的例子

二維選取方法的例子

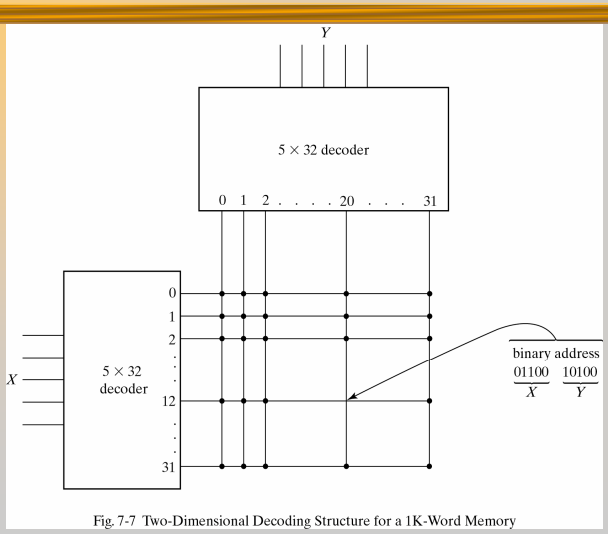


Fig. 7-7 Two-Dimensional Decoding Structure for a 1K-Word Memory

二維選取方法的例子

- 使用單獨一個10x1024的解碼器。
  - 單獨的解碼器，我們可能需要1024 個AND閘，且每個AND閘需有10條輸入線。
- 用兩個5x32的解碼器
  - 只需要64個AND閘，且每個AND閘只有5條輸入。
- 在記憶體陣列的每一個字組是由X線的其中一條與Y線的其中一條同時來選取的。每個交叉點代表一個字組而每個字組內可有任意個位元。

SRAM、DRAM記憶體 比較

- SRAM記憶體
- DRAM記憶體
  - DRAM的單位密度是SRAM的四倍，這允許將四倍的記憶體容量放置在一個已知大小晶片上。而儲存在DRAM的每個位元其價格爲SRAM的1/3～1/4左右。我們了解到另一個較省錢的原因是因爲DRAM的功率消耗較SRAM來得低。這些優點，使得在製造大記憶體時都喜歡用DRAM技術來完成。因爲它們的大容量，解碼DRAM位址的解碼就使用二維陣列的方式，通常較大的記憶體就有多個陣列。爲了減少IC包裝的接腳數，設計者就利用位址多工方法，藉由一組位址輸入接腳來容納位址元件。

位址多工方塊圖

- 64K-字組的記憶體

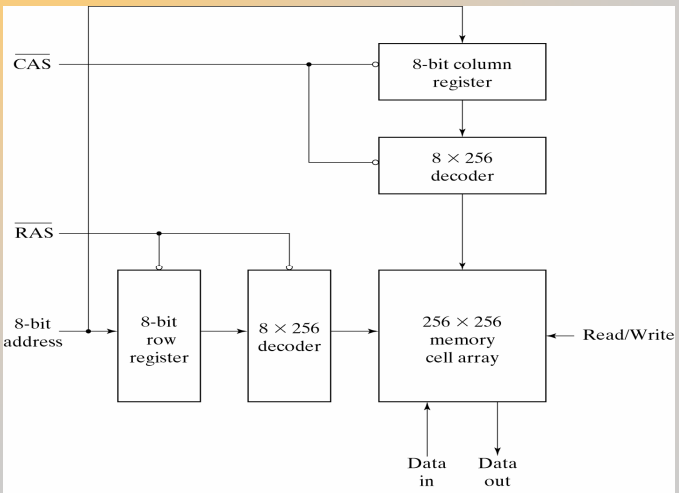


Fig. 7-8 Address Multiplexing for a 64K DRAM

64K-字組位址多工

- 位址多工
  - 有一個8-位元的位址輸入及兩個位址致能。此位址致能包括啓動代表列與行暫存器的位址。兩階段的使用RAS與CAS設定可以將16-位元的位址應用到DRAM。這兩部分的位址各自在它們的暫存器中，而解碼器須將它們解碼以便選取相對應於列與行位址的一個儲存格，然後在此儲存格中執行讀取或寫入的操作。

錯誤的檢測及更正

- 錯誤檢測方法
  - 是用同位位元(見3-8節)，同位位元與記憶體內的資料字組一起產生與儲存。
- 錯誤更正碼
  - 可以產生多重同位位元檢查元，這些與資料字組一起儲存在記憶體內，每個檢查位元都是資料字組內的一組位元的同位元，當字組從記憶體位置讀取回來時，相關的同位位元也同時從記憶體讀取，同時與已讀取資料所產生的一組新檢查位元做比較，若這些檢查位元是正確的，那表示沒有錯誤發生，若這些檢查位元與所儲存的同位位元不相符，它們可產生一個唯一的圖型，稱之爲多義元(syndrome)，可用來辨識位元的錯誤。

漢明碼

- 隨機存取記憶體中最常用的一種錯誤更正碼 。
- 漢明碼中，是將k個同位位元加至n個位元的資料字組，形成一個n+k個位元的新字組。
- 若是2的冪次方編號，則保留給同位位元，其餘的位元則是資料位元。
- 此碼可被用在任何長度的字組 。
- 漢明碼有k個檢查字元及n的資料位元，總共有n+k個字元。它們之間的關係如下：

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

以k表示，解出n

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

此關係式可計算出，與k個檢查位元配合所使用之資料位元數的公式。

漢明碼例子

- 資料字組的8個位元爲11000100
- 位元位置：

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

P_1 = 位元(3,5,7,9,11)的XOR = 1⊕1⊕0⊕0⊕0 = 0

P_2 = 位元(3,6,7,10,11)的XOR = 1⊕0⊕0⊕1⊕0 = 0

P_4 = 位元(5,6,7,12)的XOR = 1⊕0⊕0⊕0 = 1

P_8 = 位元(9,10,11,12)的XOR = 0⊕1⊕0⊕0 = 1

- 8位元資料字組與4個同位位元一起儲存在記憶體內而成爲一個12位元的合成字組。

檢查位元之計算

★當這12個位元從記憶體讀取出來時，它們再一次檢查可能的錯誤，此同位位元依包括同位位元的相同位元組合來檢查，4個檢查位元的計算如下：

C_1 = 位元(1,3,5,7,9,11)的XO

C_2 = 位元(2,3,6,7,10,11)的XOR

C_4 = 位元(4,5,6,7,12)的XOR

C_8 = 位元(8,9,10,11,12)的XOR

結果$C = C_8C_4C_2C_1$ = 0 0 0 0，這表示無錯誤發生。

K個檢查位元所使用之資料位元數

檢查位元數 k	遮料位元範圍 m
3	2-4
4	5-11
5	12-26
6	27-57
7	58-120

單-錯誤更正　雙-錯誤檢測

- 漢明碼僅能檢測及更正單一錯誤，多重錯誤則無法檢測。
- 藉由加入另外的同位位元至編碼的字組，漢明碼可以用來更正單一錯誤及檢測出雙錯誤。
- 先前12位元已編碼的字組變爲001110010100P_{13}，其中的P_{13}是從其他12個位元做互斥或所得到的。
- 字組從記憶體讀取出來時，檢查位元要被計算，同時同位位元也包含在此13個位元。

偶同位例子

- 若P=0，此同位位元是正確的(偶同位)。
- 但若P=1時，則此13個位元上的同位不正確(奇同位)那麼可能發生下列四種狀況：
  - 若C = 0且P=0，無錯誤發生。
  - 若C≠0且P=1 ，發生單一錯誤,且可被更正。
  - 若C≠0 且P=0，發生雙錯誤，可被檢測但無法更正。
  - 若C = 0 且P=1，有一錯誤發生在P_{13}位元。
- 這種方式可以檢測出多於兩個的錯誤，但並不保證可以檢測出

唯讀記憶體的方塊圖

- ROM本身是一個可以永久儲存二進位資訊的記憶體裝置。
- ROM的方塊圖如圖7-9所示，它包含k個輸入與n個輸出。

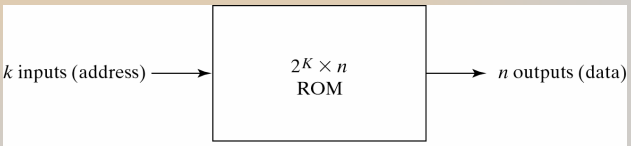


Fig. 7-9 ROM Block Diagram

唯讀記憶體

- 輸入提供了記憶體的位址，而輸出則是提供儲存字組的資料位元,它是由address來選擇,至於ROM中的所包含的字組數目則是由k條位址輸入線,所定義出2 ^{k}不同的位元
- 數目,也來代表2^{k}個不同的字組,注意,ROM並沒有資料輸入,因爲它並沒有寫的動作,積體電路ROM晶片,有一個或是更多的致能輸入且有時還有三態的輸出來簡化大的ROM陣列結構.

32X8 ROM的內部邏輯

- 五個輸入則是藉由一個5x32的解碼器將它們解碼成32個不同的輸出。每一個解碼器的輸出代表一個記憶體的位址，而這32個解碼器的輸出和8個OR閘的每一個相連接。

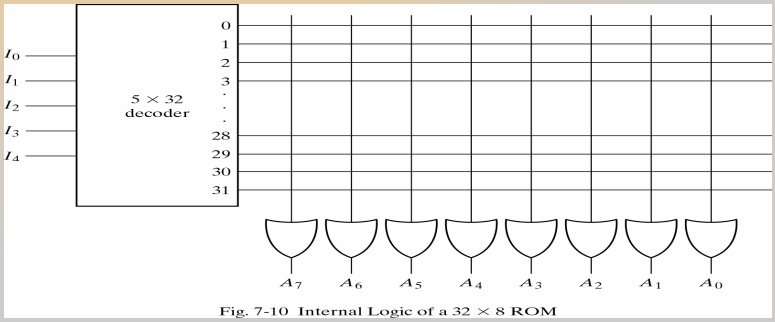


Fig. 7-10 Internal Logic of a 32 × 8 ROM

ROM的真值表

- 儲存在ROM內部的二進位可藉由真值表說明，此表可以看出在每一個位址裡的字組內容。
- 32x8 ROM的內容可以用類似於表7-3的真值表來說明

輸入					輸出							
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

依據表7-3所規劃的ROM

- 規劃此ROM的硬體程序，就是依據所給的真值表結果將其間的連接用熔絲熔斷

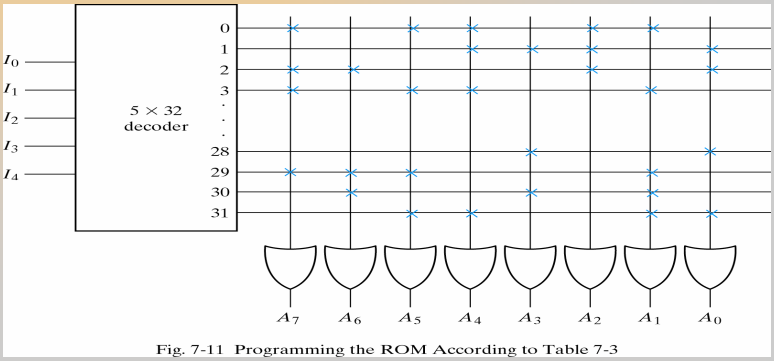


Fig. 7-11 Programming the ROM According to Table 7-3

組合邏輯電路的製作

- 基本上ROM是一個包含解碼器及OR閘這兩者在一起的單一裝置。
- ROM的輸出可以被規劃成代表在組合邏輯電路中輸出變數的布林函數，它是藉由選擇這些包含在函數裡的最小項之連接來完成。
- ROM內部的操作可以用兩種方式來解釋，第一種解釋就是包含一個儲存字組的固定模型；第二個解釋可以把它看成是，一個組合邏輯電路製作的單元。從這個觀點來看，每一個輸出端點可以分別被考慮成一個最小項之和的布林函數之輸出。
- 當一個組合邏輯電路用一個ROM來設計，它就不需要去設計邏輯電路，或是表示出此單元的內部邏輯連接。

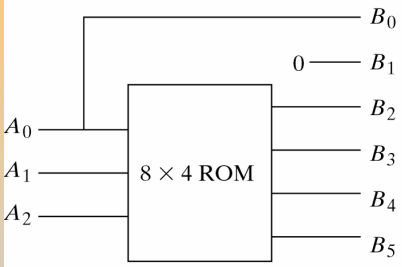
例題7-1

- 用一個ROM來設計一個組合邏輯電路。此電路接受一個3個位元的數，並且產生一個二進位輸出，此輸出值等於輸入值的平方。
  - 第一個步驟就是將此組合邏輯電路的真值表推導出來。
  - 此電路需要三個輸入及六個輸出以完成所有可能的二進位數。
  - ROM的最小尺寸必須有三個輸入，四個輸出，此三個輸出可以指定出8個字組，所以ROM的大小必須爲 8x4。

例題7-1的真值表

輸入				輸出							十進位
A_2	A_1	A_0		B_5	B_4	B_3	B_2	B_1	B_0		
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

例題7-1 ROM的完成



(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

Fig. 7-12 ROM Implementation of Example 7-1

ROM的種類

★ROM中所需的路徑可以用四種不同的方式來規劃

- 第一種稱之爲罩幕規劃(mask programming)
  - 可由半導體製造公司在該單元製造的最後過程中完成。
  - 製造ROM的程序則需要顧客填好適合他想要的ROM的真值表。
  - 此種程序是很昂貴的。因爲賣方會向顧客收取製作此特別ROM罩幕的費用，基於這種理由，若有大量製造相同ROM的組態時，罩幕規劃才會較經濟些。
- 第二種稱之爲可規劃僅讀記憶體(programmable read-only memory或PROM)的ROM則較爲經濟。
  - 這允許使用者在實驗室規劃該PROM，以得到所期望輸入位址與儲存字組之間的關係。
  - 規劃器(programmer)它是可以用來製造這種程序的設計。
  - ROM或PROM規劃的硬體程序是不可逆的。

ROM的種類

- 第三種ROM的型式爲可清除的(erasable)的PROM或是EPROM，EPROM即使先前已被規劃過了，但仍可重建其初始值。將EPROM放置在一特殊的紫外線下一段時間，短波輻射對作爲規劃接點用的內部浮接邏輯閘放電，在清除後，此ROM回到其初始狀態而可以被規劃出一些新的值。

- 第四種ROM的型式稱之爲可電性清除(electrically-erasable)的PEOM(EEPROM或E^2PROM)，它的方法如前面EPROM，除了先前規劃的接點可以用電氣訊號代替紫外線來將它清除，這個好處就是清除此裝置時，不必將它從治具上移走。

PLD的組合

★ 三種主要的組合PLD的裝置。它們之間的不同爲其AND-OR陣列中的規劃接點之配置

- 可規劃的僅讀記憶體(PROM)
  - 有一固定的AND陣列構造如一解碼器以及可規劃的OR陣列。
  - 此可規劃的OR閘是用最小項之和的方式執行布林函數。
- 可規劃陣列邏輯(PAL)
  - 有一個可規劃的AND陣列以及一個固定的OR陣列，此AND閘規劃成提供布林函數積項之用，而由每一OR閘來取邏輯和。
- 可規劃規邏輯陣列(PLA)
  - 其AND及OR陣列均可以被規劃，在AND陣列中的積項可被任一OR閘所共用，以提供所需積之和的執行。
  - 具適用性

三種PLD的構造

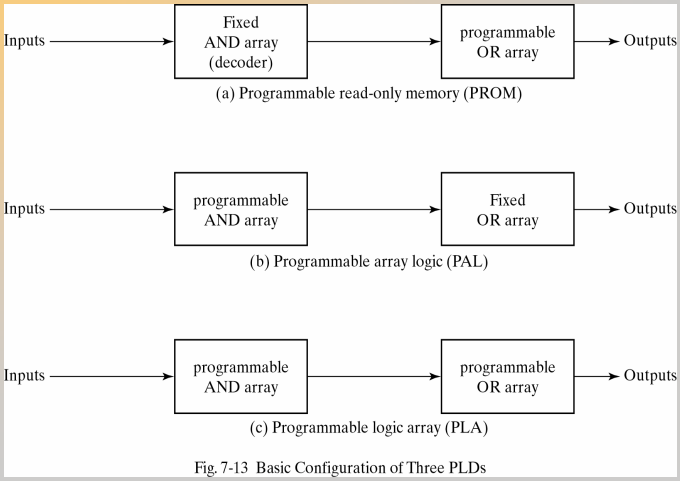


Fig. 7-13 Basic Configuration of Three PLDs

可規劃邏輯陣列

★可規劃邏輯陣列(PLA)在觀念上和PROM很相似，除了不能提供變數的全部解碼及不能產生所有的最小項外，PLA內的解碼器則被一個AND陣列所取代。此AND閘被規劃成產生輸入變數所組成的任何乘積項，然後這些乘積項被連接到OR閘以提供所需布林函數的積之和。

PLA具有3個輸入4個乘積項及2個輸出

- 執行的布林函數爲

在圖中的每一個AND閘所產生的乘積項都被列在該邏輯閘的輸出，而這些乘積項則是由那些輸入被連接到交叉點且用X做標記所決定的。而一個OR閘的輸出，則給予此邏輯做被選擇積項之和。至於它的輸出則視連接到XOR閘的一個輸入來決定，其結果可能是補數或是保留原來的值。

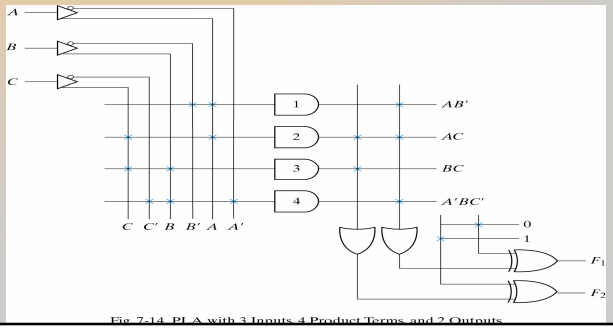


Fig. 7-14 PLA with 3 Inputs, 4 Product Terms, and 2 Outputs

規劃表

★此PLA規劃表包括三個部份

- 第一部份列了用數字表示的乘積項
- 第二部份指定在輸入與AND閘間所需的路徑
- 第三部份指定了AND及OR閘間的路徑

						輸出	
	成績項	輸入				(T)	(C)
		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	-

規劃表

- 每一個乘積項而言，它們的輸入用1，0或是一(虛線)來標記
- 規劃表中輸入與AND閘間的路徑是由標題爲"輸入"這一行所指定的
- 在AND與OR閘間的路徑是由標題爲輸出的行所指定的

PLA

- PLA的大小是由輸入數目，乘積項數目及輸出數目來指定的
- 一個典型的PLA積體電路可有16個輸入，48個乘積項，及8個輸出。
- 如，有一個具有n個輸入，k個乘積項，n個輸出的PLA。其內部邏輯包含了n個緩衝一反相，k個AND閘，m個OR閘，及m個XOR閘。
- 當利用PLA來設計一個數位系統時，不需要像圖7-14那樣畫出此單元的內部接線，所需要的就是一個PLA的規劃表。
- PLA可做罩幕規劃或現場規劃，

例題7-2

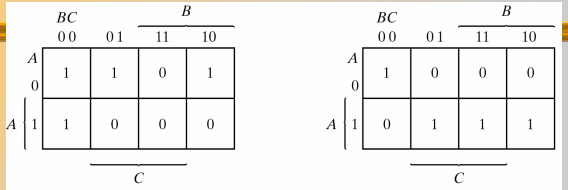
- 用PLA 來執行下列兩個布林方程式:

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

- 此二個函數可用圖7-15之卡諾圖加以簡化，這二個函數的真值與補數都用積之和的型式加以簡化，產生的最少乘積項的組合爲：

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

例題7-2之解答



$F_1 = A'B' + A'C' + B'C'$
$F_1 = (AB + AC + BC)'$

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

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

Fig. 7-15 Solution to Example 7-2

可規劃陣列邏輯

- 可規劃陣列邏輯是一個具有一固定OR陣列與一可規劃AND陣列的可規劃邏輯裝置。由於僅有AND閘可規劃，所以此種PAL較容易規劃，但不像PLA那樣的適用。
- 典型的PAL積體電路可能有8個輸入，8個輸出及8個區域。而每個區域均有一組8個寬度的AND-OR閘，至於輸出端子，有時則用三態緩衝器或是反相器來區分。
- 當用PAL設計時，布林函數必須簡化成可適合每一部份，不像PLA，一個乘積項不可能由兩個或更多的OR閘來分享。因此，每一個函數均可自行簡化，而不用考慮共用的乘積項。而每一區域內的乘積項數目是固定的，若該函數中乘積項數目過大時，那麼就可能需要兩個區域來執行一個布林函數。

PAL具有4個輸入4 個輸出及3個寬度的AND-OR結構

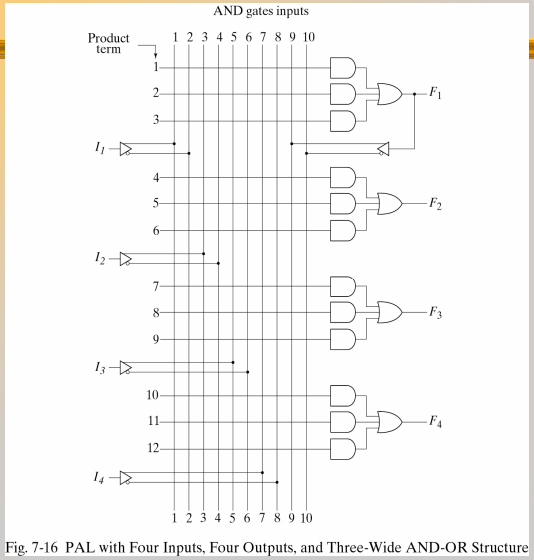


Fig. 7-16 PAL with Four Inputs, Four Outputs, and Three-Wide AND-OR Structure

用PAL設計組合邏輯電路

- 布林函數

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

- 化簡後之布林函數

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

- PAL規劃表除了AND閘需要規劃外，其餘均與PLA的規劃表相似。

PAL規劃表

成績項		AND 輸入					
		A	B	C	D	W	
1		1	1	-	-	-	$w = ABC' + A'B'CD'$
2		0	0	0	0	-	
3		-	-	-	-	-	
4		1	-	-	-	-	$x = A + BCD$
5		-	1	1	1	-	
6		-	-	-	-	-	
7		0	1	-	-	-	$y = AB' + CD + B'D'$
8		-	-	1	1	-	
9		-	0	-	0	-	
10		-	-	-	-	1	$z = w + AC'D' + A'B'C'D'$
11		0	-	0	0	-	
12		1	0	0	1	-	

規劃表所指定的PAL熔絲路徑圖

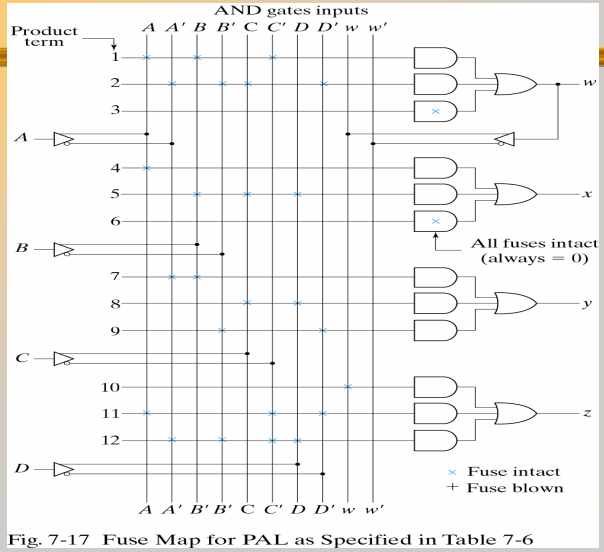


Fig. 7-17 Fuse Map for PAL as Specified in Table 7-6

循序可規劃裝置

- 數位系統常用正反器及邏輯閘來設計電路。
- PLD只包含了邏輯閘，因此當設計數位系統時，就有必要包括額外的正反器
- 循序可規劃裝置則同時包含了邏輯閘與正反器。因此，此裝置可以被規劃成執行各種不同的循序電路功能。
- 我們將它分成三個類型來描述，而不深入探討其詳細的構造。
  1. 循序(或簡單)可規劃邏輯裝置(SPLD)
  2. 複雜的可規劃邏輯裝置(CPLD)
  3. 現場可規劃邏輯陣列(FPGA)

SPLD

- SPLD在積體電路元件內除了有AND-OR陣列外，還包括了正反器，一個循序電路就如圖7-18所示。
- 一個PAL或PLA被修改就是利用包含從暫存器連接到一些正反器的變動。至於電路的輸出則可以由OR閘或正反器的輸出得到。
- 最常被用到的SPLD型態是用D型正反器和PAL組合在一起。
- SPLD的每個部份被稱做marcocell，一個marcocell就是一個電路，它包含一個積之和的組合邏輯函數及一個可自由選擇的正反器。

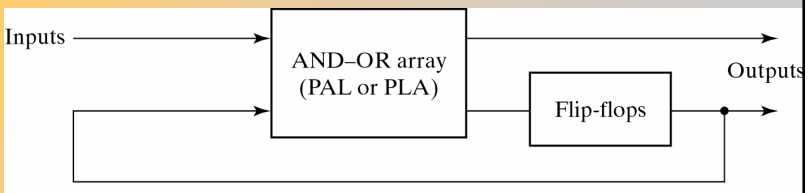


Fig. 7-18 Sequential Programmable Logic Device

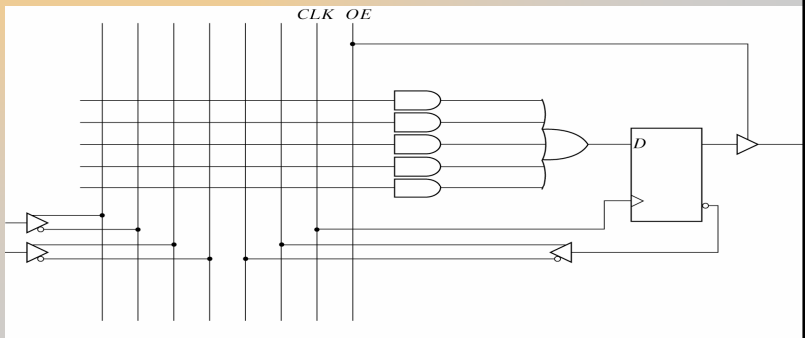


Fig. 7-19 Basic Macrocell Logic

現場可規劃邏輯序列(FPLS)

- 第一種可規劃裝置被發展出來，主要是爲了要製作循序電路
- 典型的FPLS則是由一個PLA及幾個正反器的輸出所組成的。而這些正反器適合被規劃成JK型或D型正反器的操作。
- FPLS並不是很成功的。

macrocell

- 一個典型的SPLD IC，它的包裝裡就有8個到10個macrocell，所有正反器都被連接到共同時脈輸入，同時所有三態緩衝器都由OE輸入來控制。
- macrocell可以有其他規劃特色，典型的規劃是可以自由選擇的。包括使用或不使用此正反器的能力，選擇時脈邊緣的極性，暫存器的清除與設定的選擇，以及輸出是真或假的選擇等。

CPLD

- 一個CPLD是收集一些個別的PLD把它們放在一個單獨的積體電路上。
- 一個普通的CPLD結構如圖7-20所示。它透過一個可規劃的開關矩陣將多重的PLD交互連接。輸出/輸入方塊提供到IC接腳的連接。每支I/O接腳則是由三態緩衝器推導出來的，它可以被規劃成輸入或是輸出，而開關矩陣接受從I/O方塊來的輸入，且直接將它傳送至個別的macrocell。

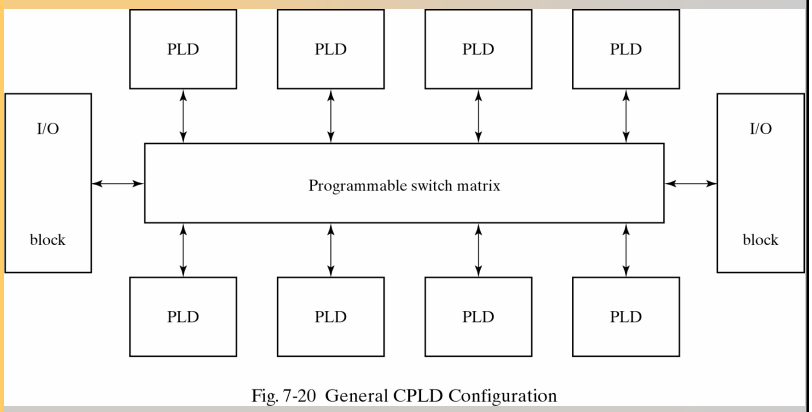


Fig. 7-20 General CPLD Configuration

FPGA

- 現場可規劃邏輯陣列(FPGA)是一個VLSI電路
- VLSI設計中，最基本的元件就是邏輯陣列
- 典型的FPGA包含上百個或上千個邏輯方塊陣列，且由可規劃的輸入及輸出方塊所圍繞，且使用可規劃交互連接將它們連接在一起。
- 典型的FPGA邏輯方塊包含查看表格、多工器、邏輯閘及正反器。
  - 此查看表格是儲存在SRAM內的真值表，且提供此邏輯方塊一個組合邏輯電路函數。

- 使用RAM來代替ROM的好處就是可以將規劃好的真值表寫入記憶體。而缺點就是記憶體容易變的。且當電源重新開始時，則查看表格的內容必須重新載入，而程式可由主電腦或另一個PROM下載。程式則一直被保留，直到FPGA重新被規劃或是電源被關掉。每次電源被打開，這裝置就必須被重新規劃。在程式中藉由使用不同的邏輯製作，來重新規劃FPGA，此法可被用在不同的應用中
- 用PLD，CPLD或是FPGA來設計電路則需要額外的電腦輔助設計工具(CAD)來製造合成程序。可利用的工具如用產生圖形輸入包裝及硬體描述語言(HDL)，例如ABEL，VHDL及VERILOG。合成工具則產生架構及連接邏輯方塊以符合用HDL描述的高階設計。