



組合邏輯電路設計(二)

MSI、LSI 與 VLSI 執行

4-1 摘要

本單元將依序討論一些常用的標準MSI電路結構與如何利用這些電路執行相關的交換函數。

常用的MSI電路為：

1. 算術運算電路
2. 多工器(multiplexer, Mux)與解多工器(demultiplexer, deMux)
3. 解碼器(decoder)(與編碼器(encoder))
4. 比較器(comparator)

其次討論僅讀記憶器與LSI / VLSI中的PLD元件基本結構與應用。

4-2 算術運算電路設計

加法運算為數位系統中最基本的算術運算。若一個數位系統的硬體能執行兩個二進制數目的加法,則其它幾種算術運算即可利用此加法運算硬體來執行:減法可由加法電路將減數的2補數加到被減數上完成;乘法由連續執行加法運算而得;除法則由連續地由被除數減去除數完成。

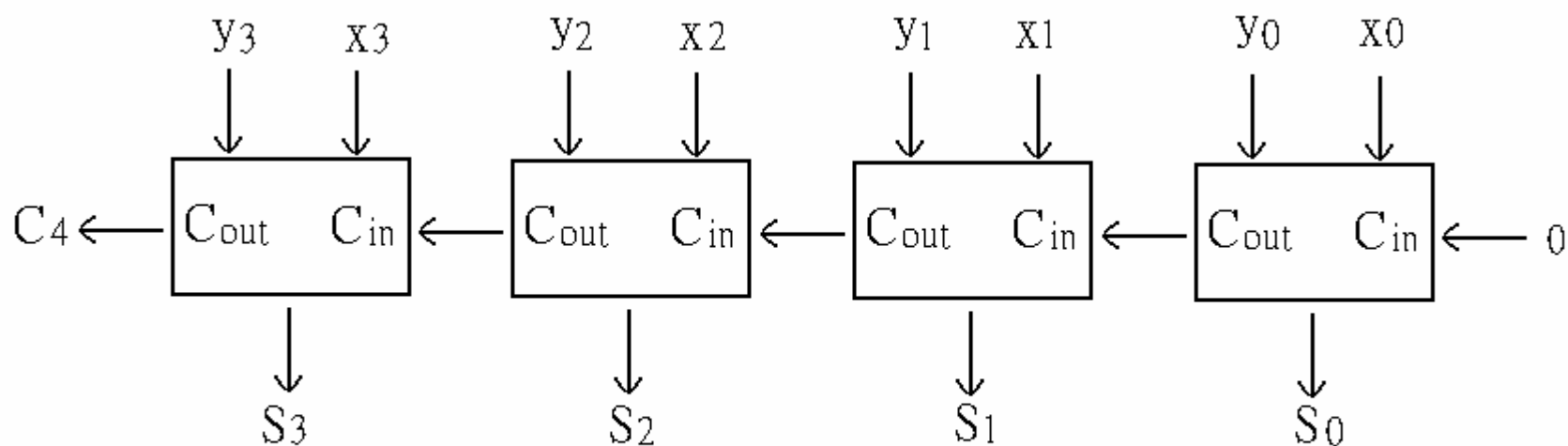
1. 並聯加法器

在實用上, 通常需要一次執行 n 個 ($n=4,8,16$ 等) 位元的加法運算, 如由前單元方式進行, 則對於 4 位元的加法器而言, 其真值表共有 2^9 次方(加數與被加數各為 4 位元及一個進位輸入) = 512 個組合, 設計程序相當複雜。

因此, n 位元(並聯)加法器通常由單位元全加器串接而成。

例題：利用全加器電路設計一個4位元並聯加法器

解：將較小有數位元的全加器的進位輸出(C_{out})串接至次一較大有效數位元的全加器的進位輸入(C_{in})即可。



4位元並聯加法器

2. 前瞻進位加法器

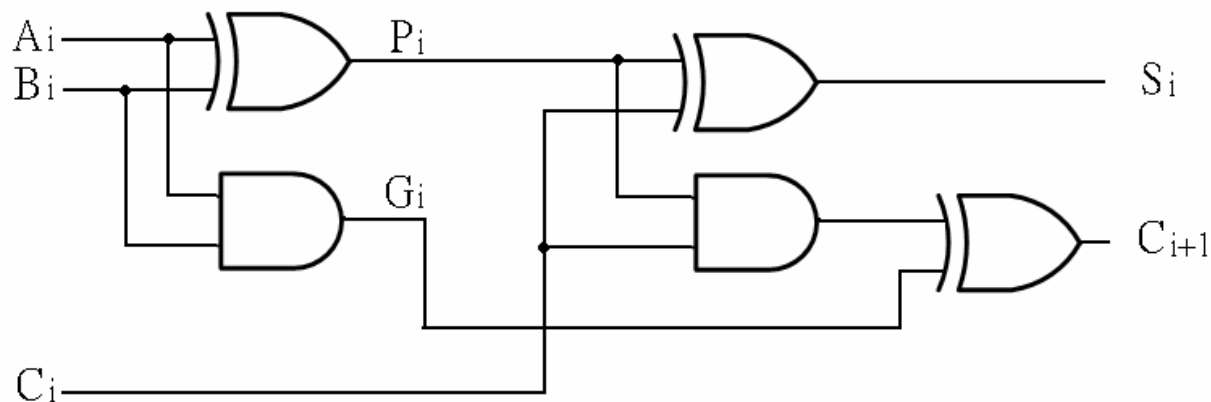
並聯加法器雖可一次執行 4 個位元的加法運算，然而若考慮到進位的傳播延遲，該電路仍然一次只能執行一個位元的加法運算。

解決的方法是經由前瞻進位(look-ahead carry)電路，產生每一級所需的進位輸入。

以上單元所設計之全加器而論

$$S = x \oplus y \oplus z$$

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



若我們定義二個新的二進變數如下：

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

則輸出和及進位可表為下式：

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

G_i 稱為進位產生(Carry generator)，當 A_i 與 B_i 均為 1 時， G_i 為 1 而與進位輸入無關。

P_i 稱為進位傳輸(Carry propagate)，它與 C_i 跟 C_{i+1} 間的進位傳輸有關。

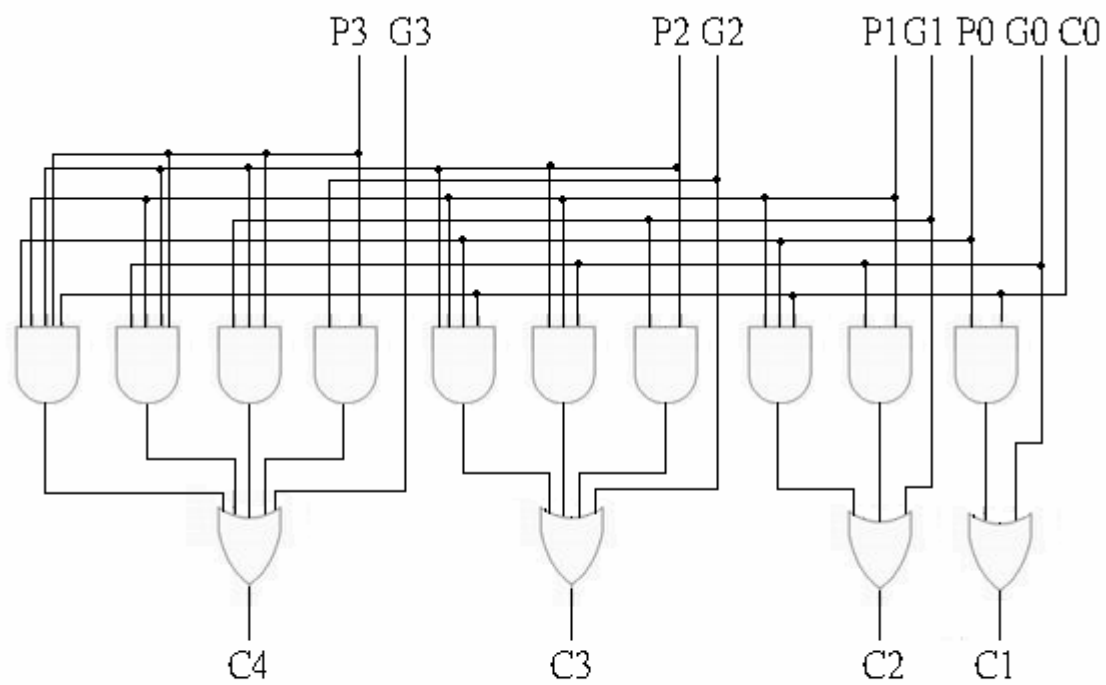
如果將每級的進位函數用前級進位取代則可得

$$C_1 = G_0 + P_0 C_0$$


$$\begin{aligned} C_2 &= G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0) \\ &= G_1 + P_1 G_0 + P_1 P_0 C_0 \end{aligned}$$

$$\begin{aligned} C_3 &= G_2 + P_2 C_2 = G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_0) \\ &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0 \end{aligned}$$

$$\begin{aligned} C_4 &= G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 \\ &\quad + P_3 P_2 P_1 P_0 C_0 \end{aligned}$$

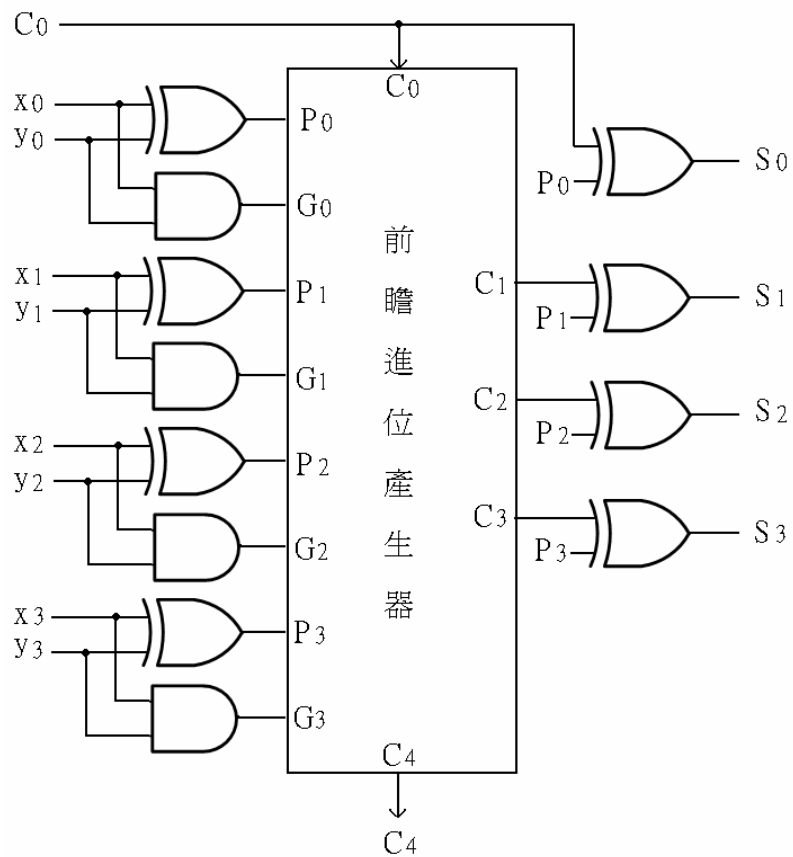


4位元前瞻進位產生器

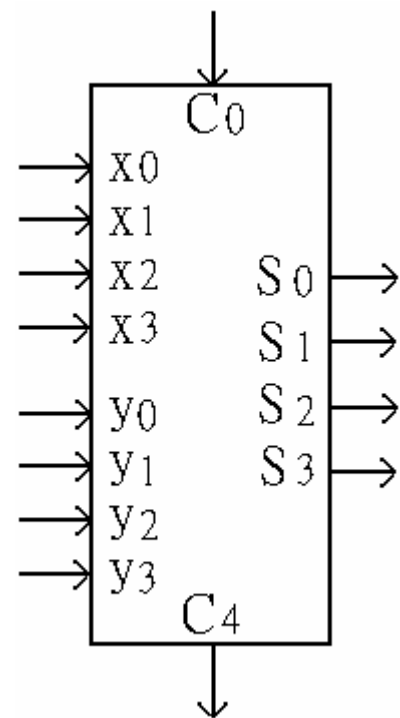


因此，所有的進位輸出皆在兩個邏輯間的延遲後得到而與級數的多寡無關。

上述電路稱為前瞻進位產生器(look-ahead carry generator)。利用此電路，可以設計一個 4 位元前瞻進位加法器(look-ahead adder)。



<電路>



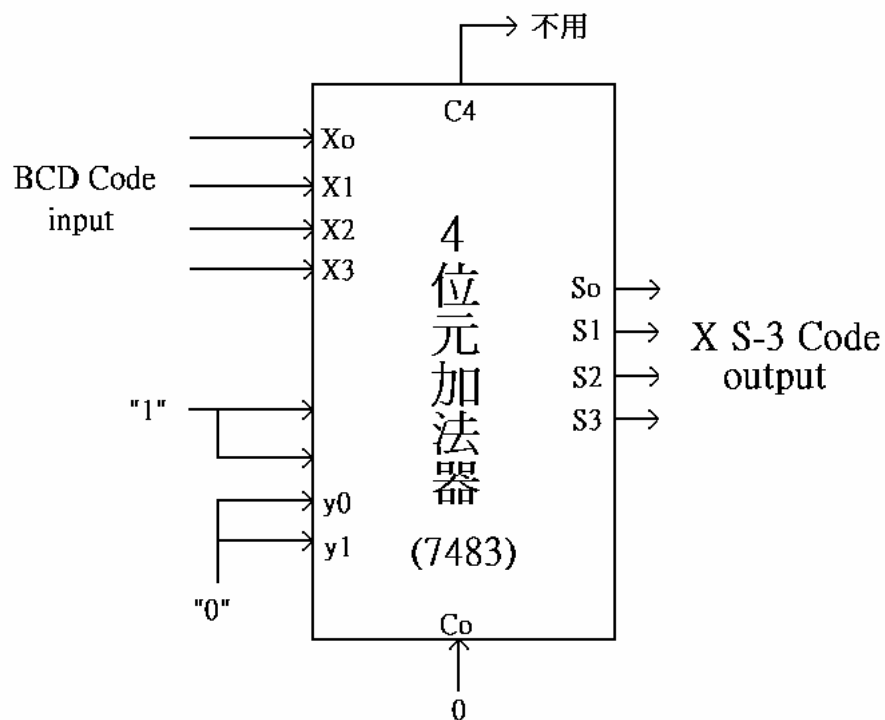
<邏輯符號>

4 位元前瞻進位加法器

※典型的4位元前瞻進位加法器為7483

應用例題：利用 4 位元並聯加法器設計一個BCD碼對加三碼的轉換電路

解：加三碼 = BCD碼 + 0011

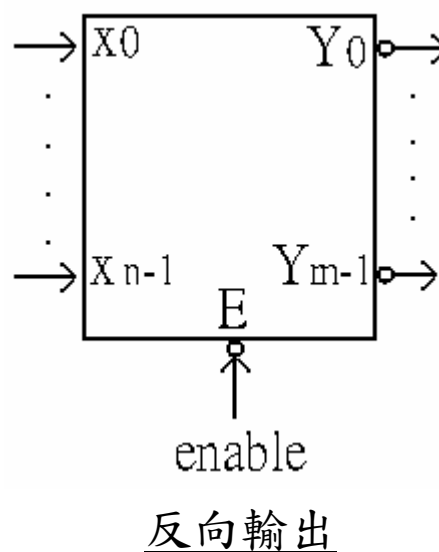
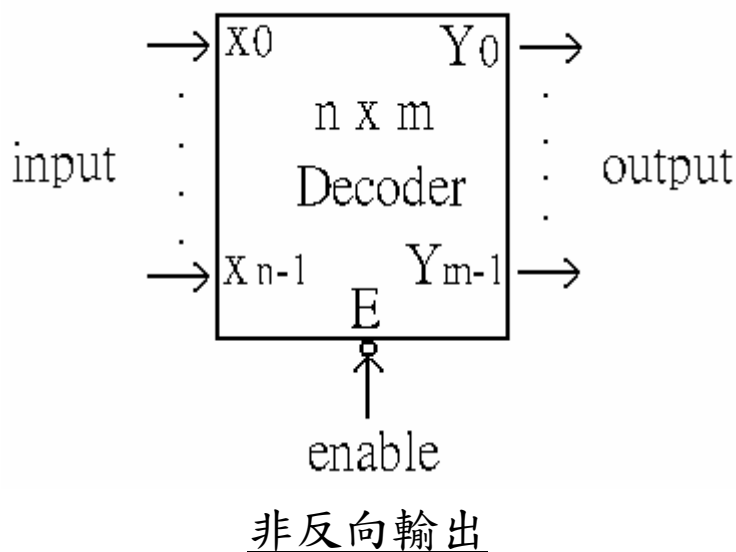


使用SSI執行，需要好幾個SSI電路，而用MSI執行時，則只需要一個電路。

4-3 解碼器(decoders)

解碼器是一個具有 n 個輸入而最多有 2^n 個輸出端的組合邏輯電路。

典型的 $n \times m$ (或稱 n 對 m)解碼器方塊圖如下



* 有些電路沒有致能控制線; 有些電路為高電位啟動。

*由圖中知它具有 n 條輸入線與 m 條輸出線，而且 $m < 2^n$ 。當 n 個位元的輸入組合皆使用時， $m = 2^n$ 稱為完全解碼；當 n 個位元的輸入組合有部分未使用(即不在意組合)時，稱為未完全解碼。

例題1：設計一具有低電位啟動致能控制的 2×4 非反相輸出解碼器電路。

解：真值表：

E	x_1	x_0	Y_0	Y_1	Y_2	Y_3
1	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1

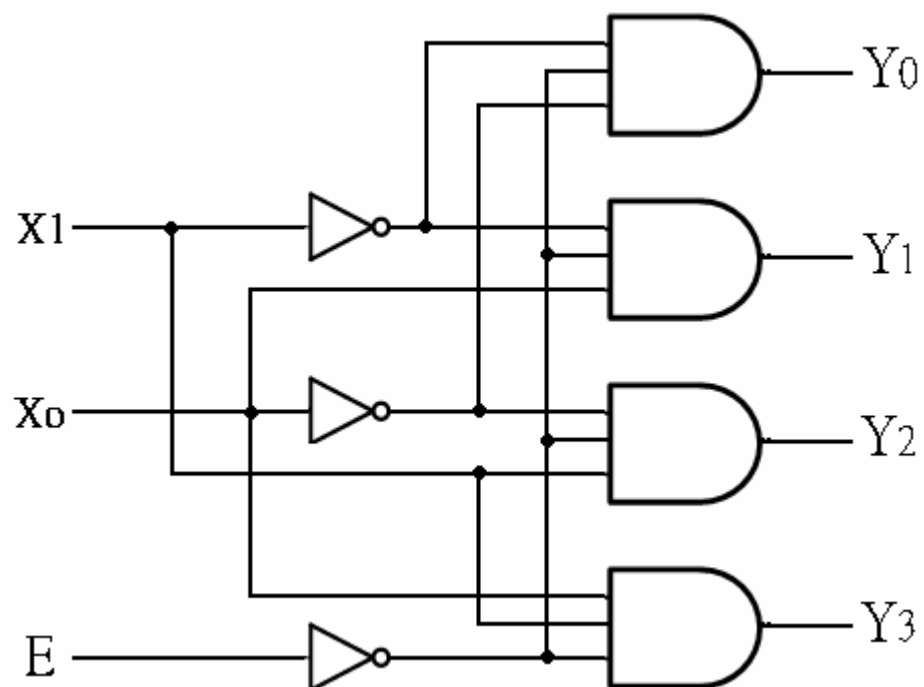
$$Y_0 = E' x_1' x_0'$$

$$Y_1 = E' x_1' x_0$$

$$Y_2 = E' x_1 x_0'$$

$$Y_3 = E' x_1 x_0$$

邏輯電路圖：



解：真值表：

[illegible]

$$\begin{array}{lcl}
 Y_0' = X_4' X_3' X_2' X_1' & \therefore Y_0 = (X_4' X_3' X_2' X_1')' \\
 Y_1' = X_4' X_3' X_2' X_1 & \therefore Y_1 = (X_4' X_3' X_2' X_1')' \\
 \quad \quad \quad / & & / \\
 \quad \quad \quad / & & / \\
 Y_9' = X_8 X_3' X_2' X_1 & \therefore Y_9 = (X_9 X_3' X_2' X_1)'
 \end{array}$$

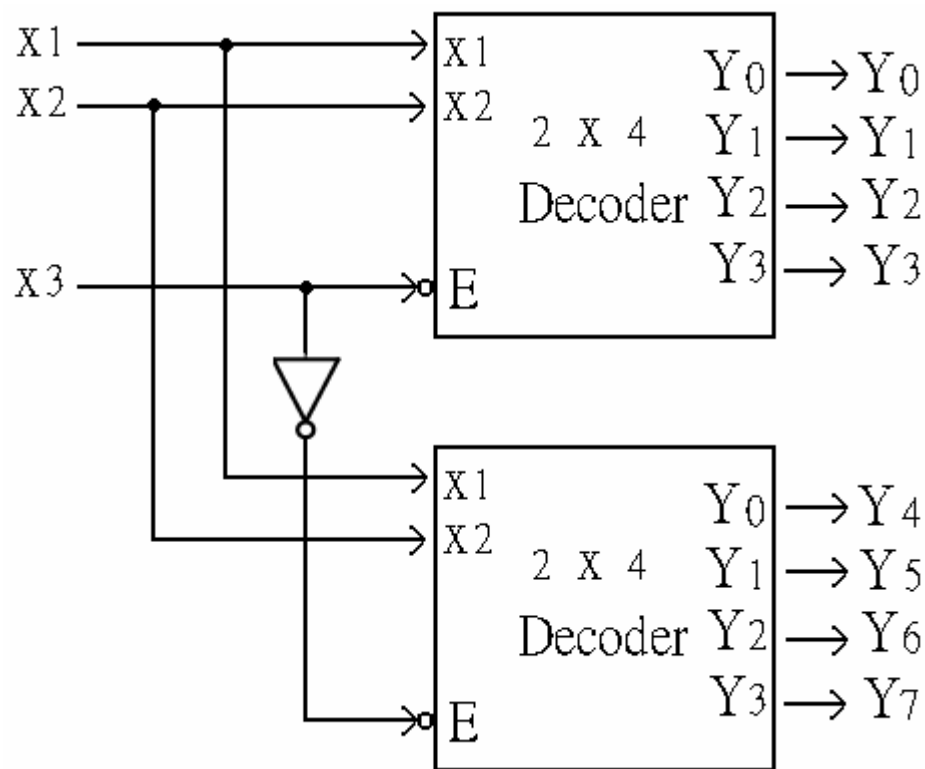
由例1、例2知在 $m < 2^n$ 的情況, Decoder只產生 n 個變數中的前面 m 個最小項；在 $m = 2^n$ 時，則產生所有最小項。

1. 解碼器的擴充

在實用上，常將多個解碼器組合以形成一個較大的解碼器。在這種組合中，使用的解碼器必須為完全解碼的電路(即 $m=2^n$)而且具有致能控制輸入。

例題：利用兩個 2×4 解碼器(具有致能控制輸入)組
成一個 3×8 解碼器電路。

解：



2.執行交換函數

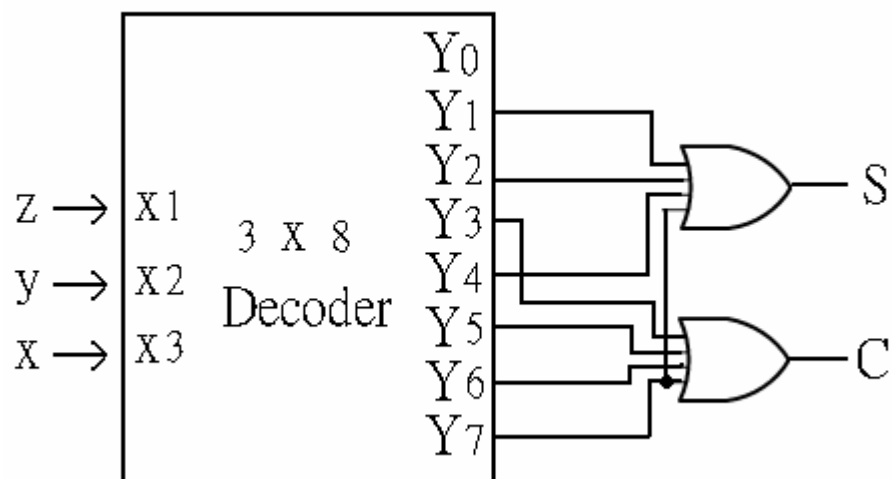
所有 n 個輸入與 m 個輸出的任何組合邏輯均可利用一個 n 對 2^n 線解碼器與 m 個 OR Gate 來完成。

例題：利用一個解碼器與二個 OR 閘製作一全加器
解：

$$S(X, Y, Z) = \sum (1, 2, 4, 7)$$

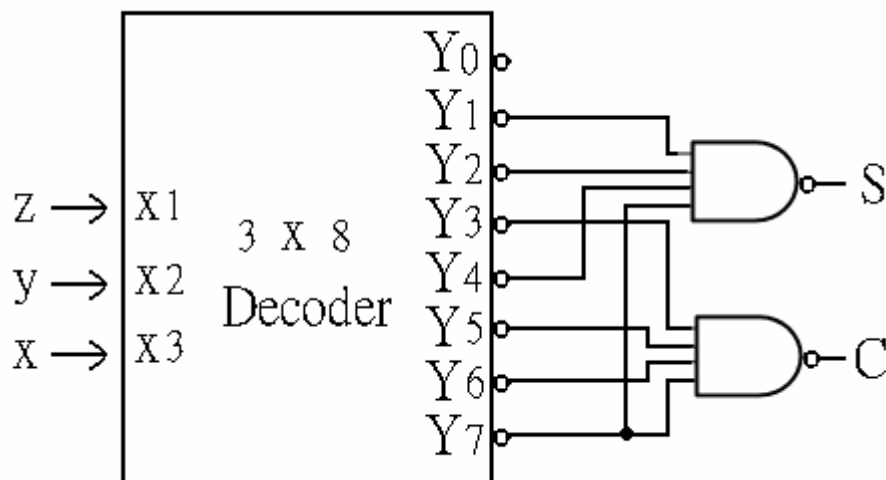
$$C(X, Y, Z) = \sum (3, 5, 6, 7)$$

x	y	z	s	c
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



討論：

(1) 若解碼器的輸出為反相輸出，則依據DeMorgan定理，外加的 OR 閘應改為NAND閘。



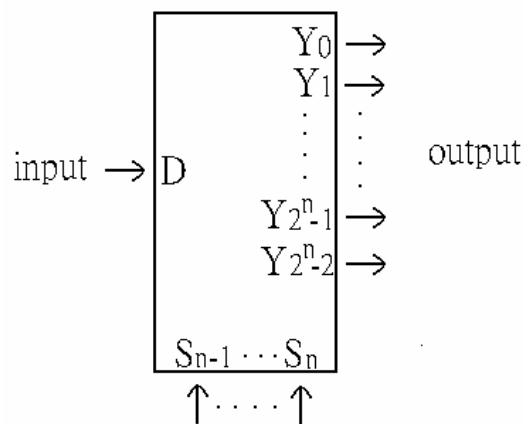
$$S = (Y_1' \cdot Y_2' \cdot Y_4' \cdot Y_7')'$$
$$= Y_1 + Y_2 + Y_4 + Y_7$$

(2) 一個函數如果有許多最小項的和，則此OR閘有很多輸入線。然而一個函數若有 k 個最小項，其補數 F' 有 $2^n - k$ 個最小項，如果 $k > 2^n/2$ ，則 $2^n/2 - k$ 比 k 小，此時利用 F' 來製作線路，最小項數目比較少。此種情形利用 NOR Gate 將 F' 的最小項相加，則可得到 F 而輸入線較少。

4-4 解多工器(demultiplexer)

解多工器 (簡稱 DeMUX) 是一個具有從單一輸入線接收資訊，並將之傳送到 2^n 個可能輸出中的一個組合邏輯電路。輸出線的指定由 n 條選擇線(或稱為位址線)決定。

典型的 1×2^n (或稱1對 2^n) 解多工器方塊圖如下：

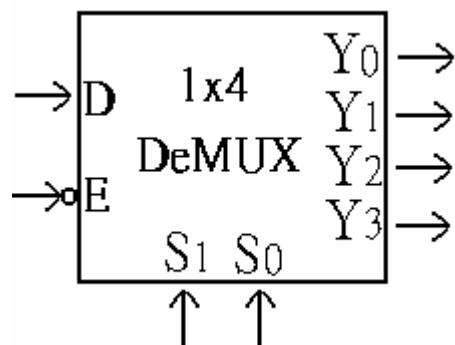


比較前典型的 $n \times m$ (或稱 n 對 m) 解碼器方塊圖知:
具有致能控制輸入的解碼器, 若將其致能控制線當
做資料輸入端, 則可以當作解多工器效用; 解多工器
若將其資料輸入端當作致能控制線, 則其等效電路
為一個解碼器。因此在 MSI 電路中, 解碼器與解多
工器通常為同一個電路。

<例題>: 設計一個具有致能控制（低電位啟動）的
 1×4 解多工器。

解:

真值表:



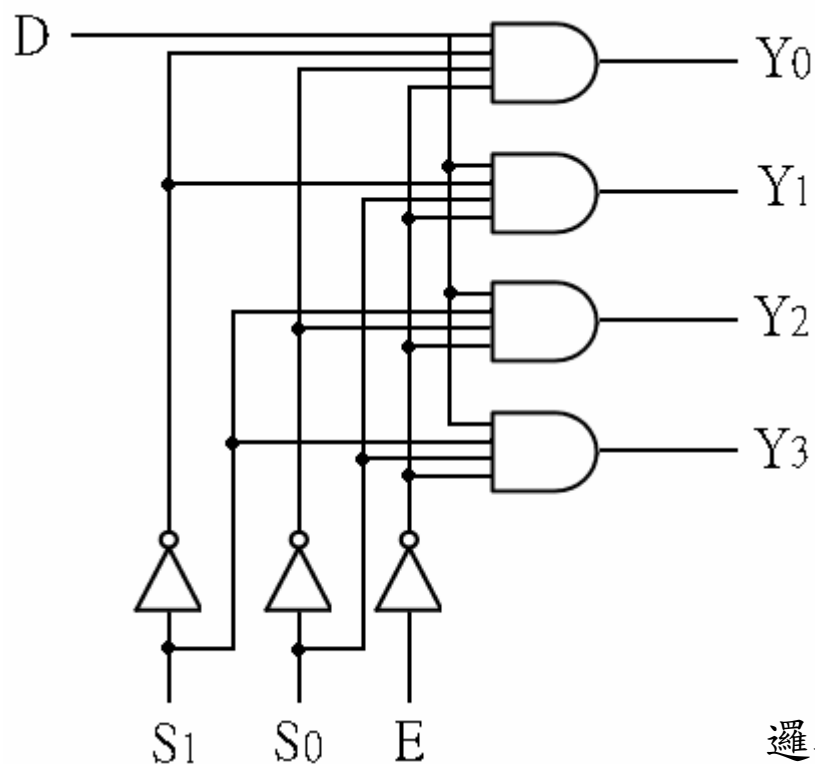
方塊圖

E	S1	S0	Y0	Y1	Y2	Y3
1	0	0	0	0	0	0
0	0	0	D	0	0	0
0	0	1	0	D	0	0
0	1	0	0	0	D	0
0	1	1	0	0	0	D

利用變數引入圖化簡得：

$$Y_0 = E' S_1' S_0' D \quad Y_1 = E' S_1' S_0 D$$

$$Y_2 = E' S_1 S_0' D \quad Y_3 = E' S_1 S_0 D$$



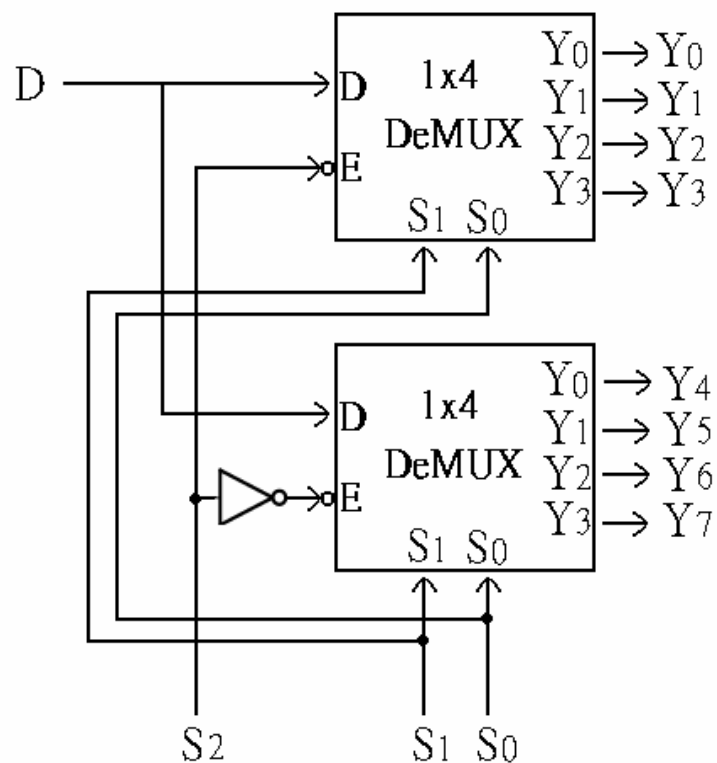
邏輯電路

1. 解多工器的擴充

在實際應用中,也常將多個解多工器(具有致能控制)串接以形成較多輸出端的解多工器樹 (demultiplexer tree)

<例題1>: 利用兩個具有致能控制的 1×4 解多工器，設計一個 1×8 解多工器電路。

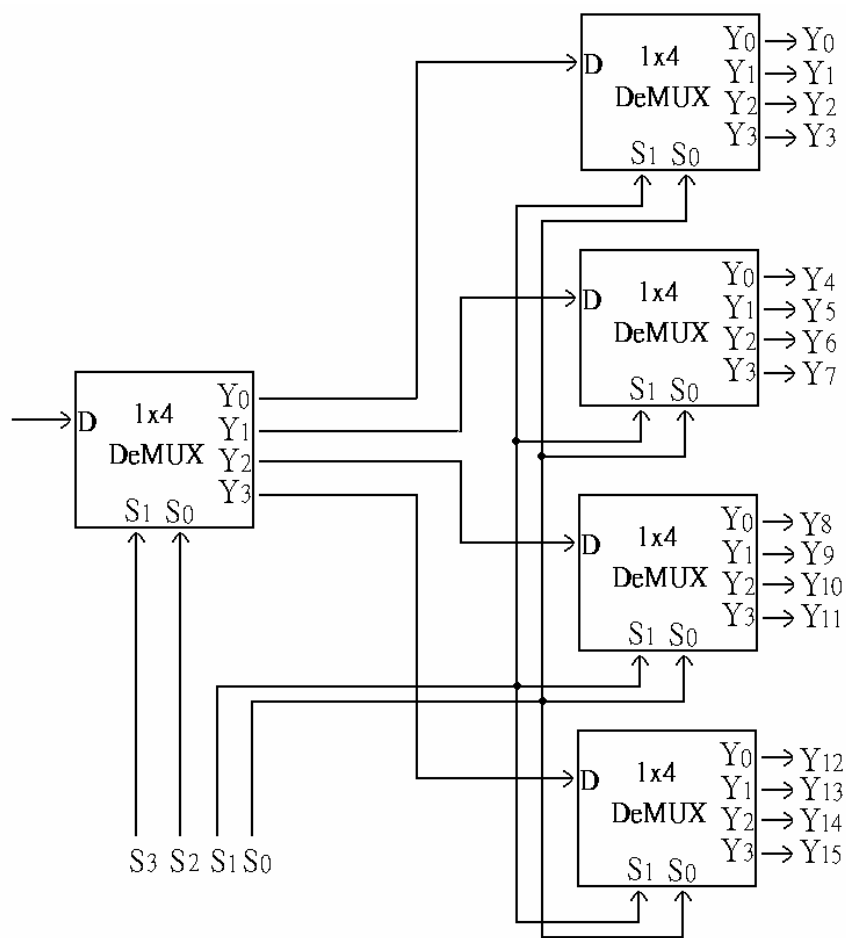
解：



<例題2> (解多工器樹)

利用五個 1×4 解多工器, 設計一個 1×16 解多工器
器電路。

解:

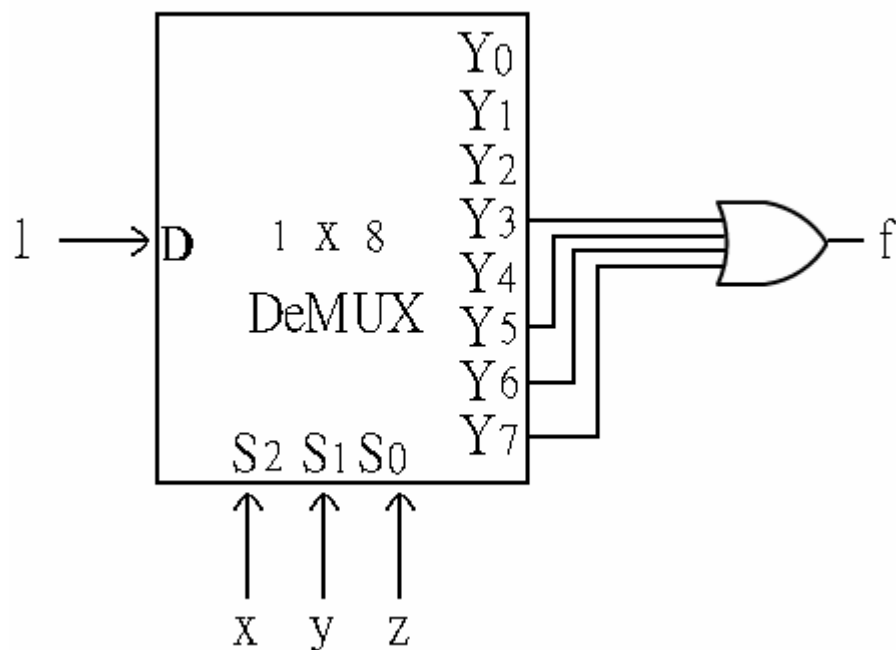


2. 執行交換函數

由前述知: 解多工器電路本身只是個乘積項(最小項)產生電路而已，因此欲執行 SOP 表示的交換函數時，必須外加 OR 閘。

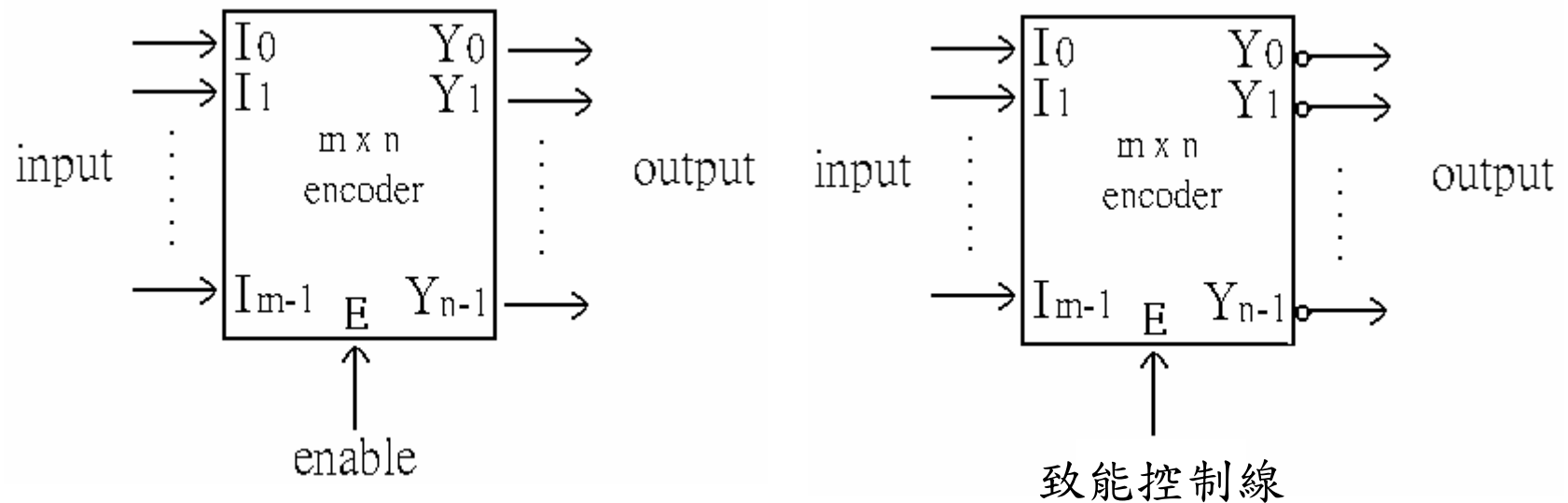
<例題> 利用一個 1×8 解多工器執行下列交換函數
 $f(x, y, z) = \sum(3, 5, 6, 7)$

解：



4-5 編碼器(encoder)

編碼器與解碼器的動作是相反的，一個編碼器具有 m 條輸入線與 n 條輸出線，而 $m \leq 2^n$ ，如下圖所示



有些電路沒有致能控制線

亦即它有 2^n (或少些) 個 input line 與 n 個 output line，且其輸出對應於 (2^n 個) m 個輸入變數代表的二進碼。

例題：設計一個 8×3 編碼器電路，假設八個輸入線
每次只有一條啟動(為1)

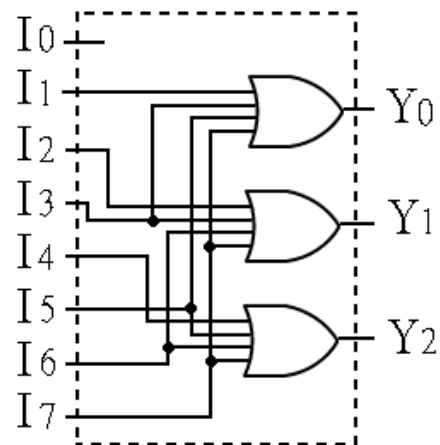
解：因為八個輸入線每次只有一條啟動(為1)，因此其
真值表只需列出八種(而非256)組合


I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	Y_2	Y_1	Y_0
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

$$Y_0 = I_1 + I_3 + I_5 + I_7$$

$$Y_1 = I_2 + I_3 + I_6 + I_7$$

$$Y_2 = I_4 + I_5 + I_6 + I_7$$



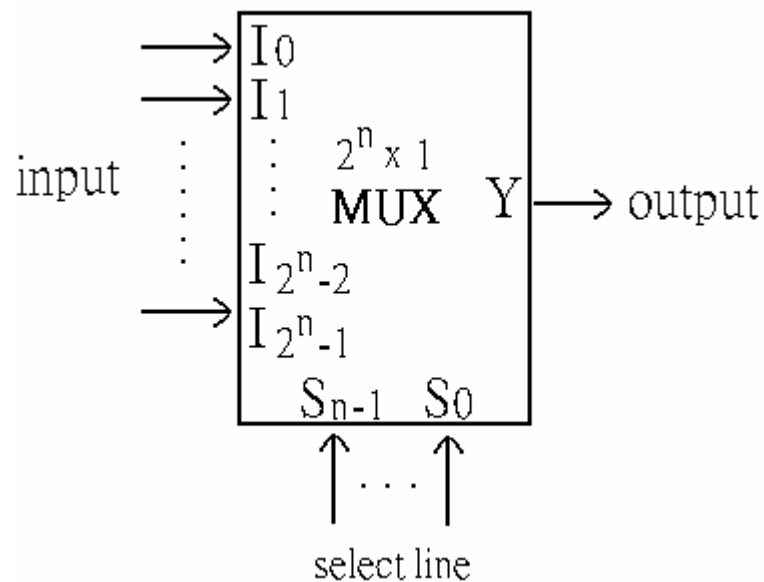


在許多實際應用中，限制每次只允許一條輸入線啟動是不可能的，這時的解決方法是採用優先權編碼器(priority encoder)。這類編碼器建立了一個輸入優先權，以確保只有最高優先權的啟動輸入線被編碼。

4-6 多工器(multiplexer)

多工器(簡稱MUX)為一組合邏輯電路，它能從多個輸入線中選取一條輸入線，並將其資訊置於單一的輸出線上。有時也稱為資料選擇線(data selector)。對於特定輸入線的指定由一群選擇線(有時稱為位址線)決定。

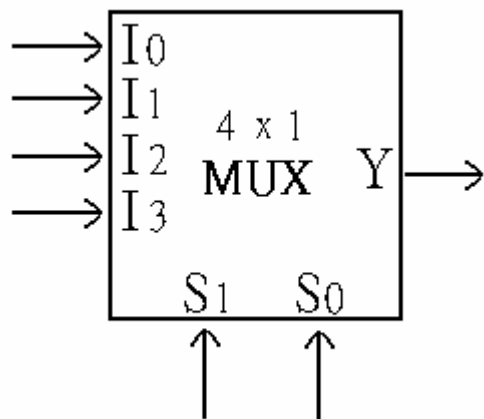
典型的 $2^n \times 1$ (或稱 2^n 對 1) 多工器方塊圖如下：



<例題>：設計一個 4 對 1 多工器電路

解：

方塊圖



真值表

S ₁	S ₀	Y
0	0	I ₀
0	1	I ₁
1	0	I ₂
1	1	I ₃

利用變數引圖化簡得

$$Y = S'_1 S'_0 I_0 + S'_1 S_0 I_1 + S_1 S'_0 I_2 + S_1 S_0 I_3$$

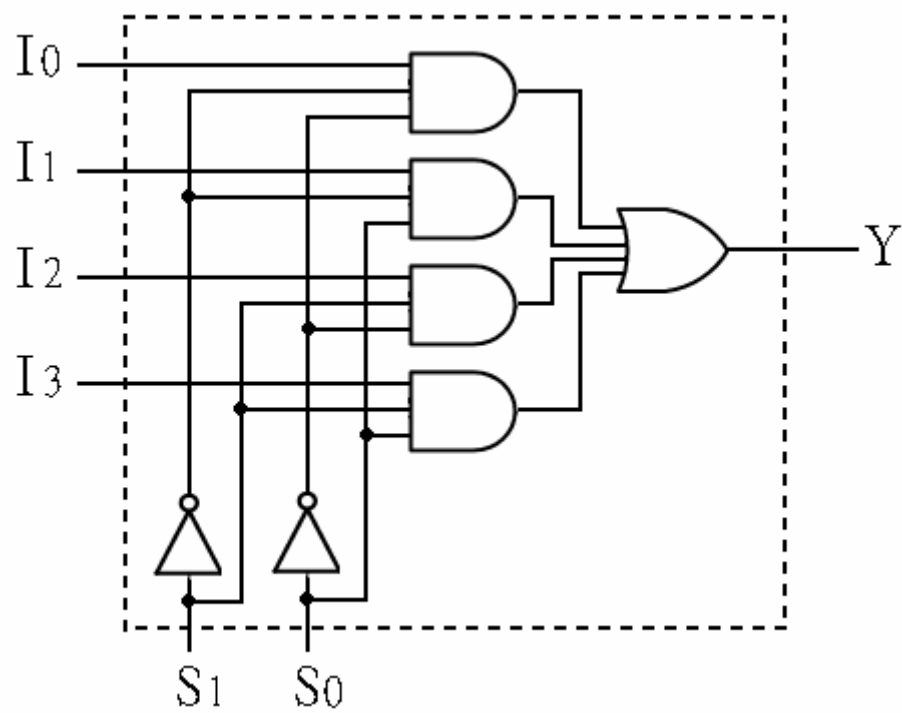
一般來說，具有 n 個控制輸入的 MUX 可以用來選擇 2^n 個資料中的任何一個，具有 n 個控制輸入和 2^n 個資料輸入的 MUX

其輸出的一般方程式

$$Y = \sum_{k=0}^{2^n-1} m_k I_k$$

m_k 是 n 個控制變數的全及項

邏輯電路：



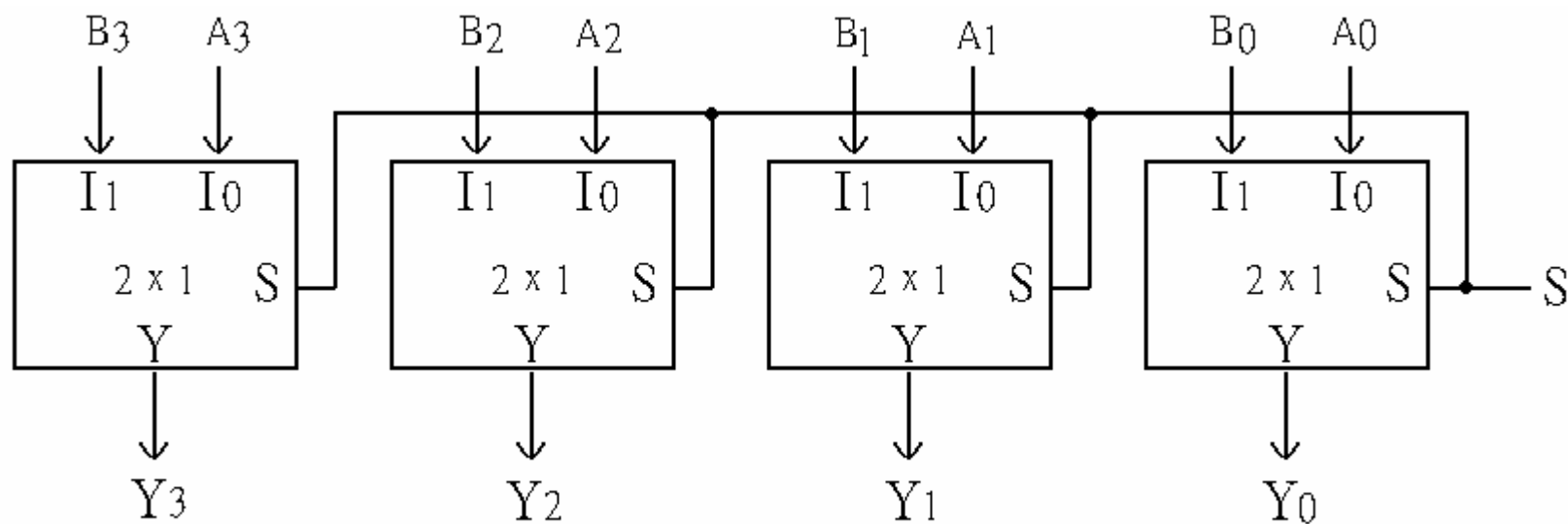


在實際應用中，通常需要將多個多工器並接使用，以構成較大的資料寬度。

例如在計算機中，資料線(或稱資料匯流排，data bus)的寬度通常為 4 條, 8 條, 16 條或32條。亦即設計數位系統時經常以 MUX 選擇要處理或選擇的資料。

<例題>: 利用 2×1 多工器設計一個 4 位元 2×1 多工器。

解:



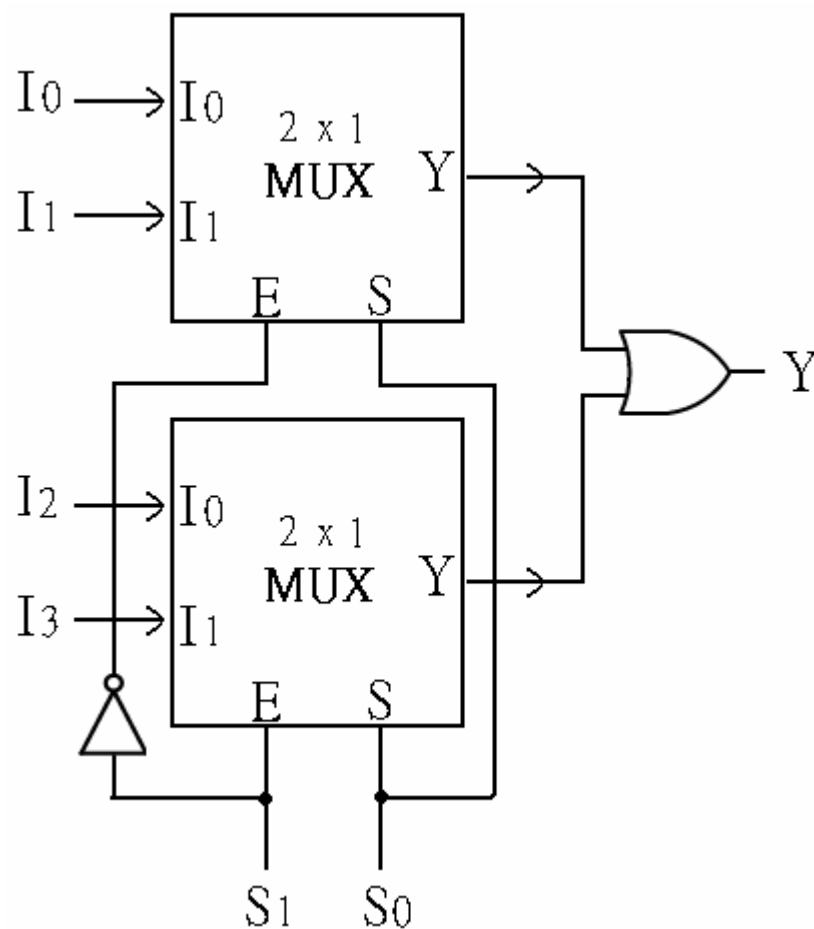
(用以選擇資料的四連多工器)

1. 多工器擴充

兩個或多個多工器通常可以組合，以形成一個具有較多輸入端的多工器，利用此種方式構成的多工器電路稱為多工器樹(multiplexer tree)。多工器樹的構成方式可以使用具有致能輸入端的多工器或沒有致能輸入端的多工器。

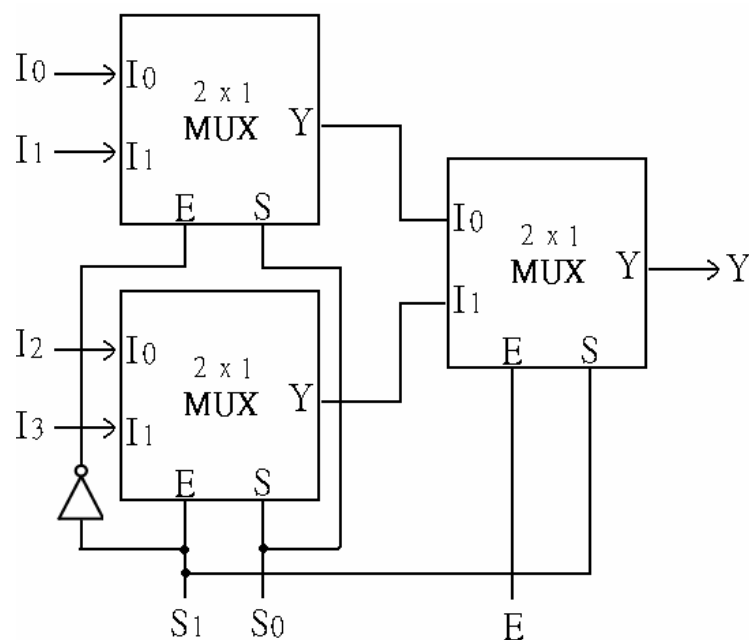
<例題> 利用兩個具有致能控制線(高電位啟動)
 2×1 多工器，設計一個 4×1 多工器。

解：



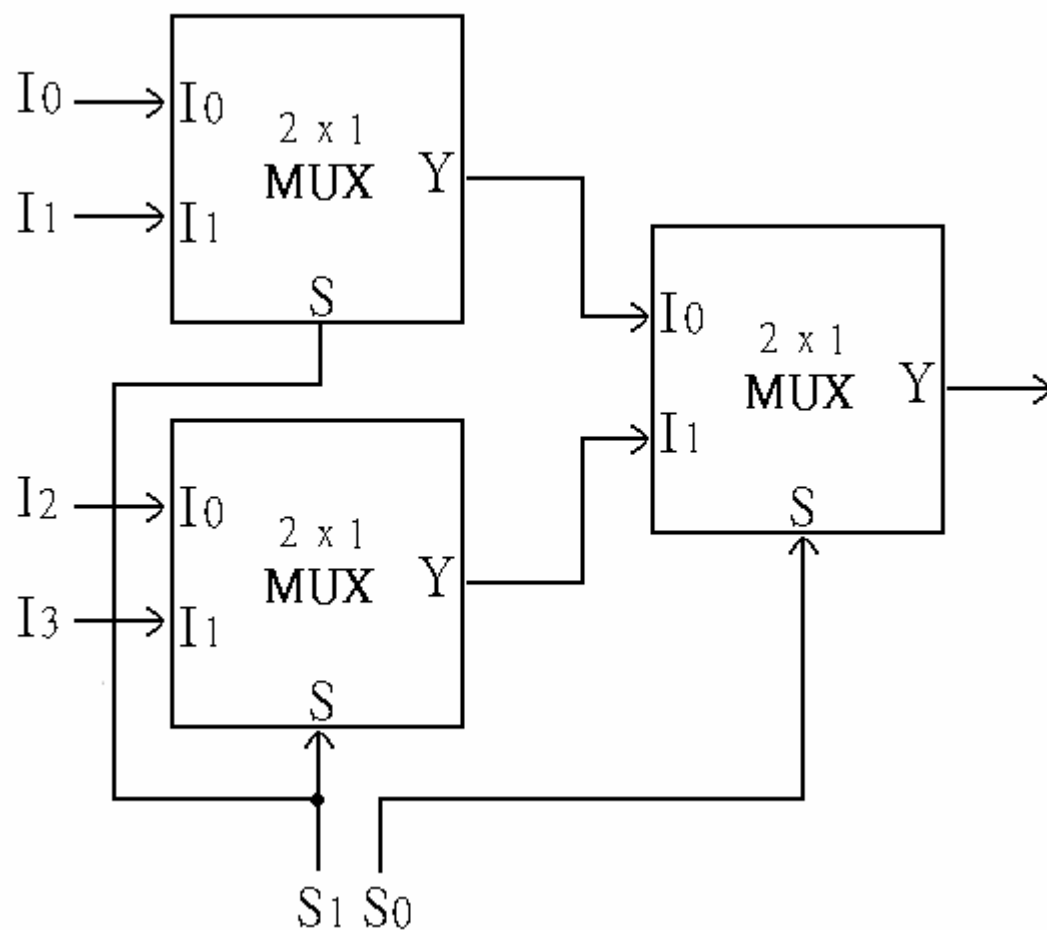
<例題> 利用一個具有致能控制的 2×1 多工器取代上題 OR Gate，使上題變成一具有致能控制之 4×1 MUX。

修改上面例題的 4×1 多工器，使其具有致能控制輸入。假設致能控制為高電位啟動，而多工器不被致能時輸出為高電位。



<例題> 利用三個不具有致能控制的 2×1 多工器，
設計一個 4×1 多工器。

解：



(pb) 利用類似的方法，可以將 3 個 4×1 多工器擴充成一個 16×1 MUX；兩個 4×1 MUX 與一個 2×1 MUX 組成一個 8×1 MUX。

2. 多工器執行任意交換函數

$2^n \times 1$ 多工器乃一通用邏輯模組(universal logic module，而稱 ULM)，即它可以執行任意的交換函數。

使用單級多工器執行一個 n 個變數的交換函數時，通常可以採用下列三種型式的多工器：

1. $2^n \times 1$ 多工器
2. $2^{n-1} \times 1$ 多工器
3. $2^{n-m} \times 1$ 多工器

然而利用多工器執行交換函數時，目前還沒有一個較簡便的方法可以求出最簡單的執行是使用多少個選擇線變數的多工器。

但對於 n 個交換變數的交換函數而言，使用 $n-1$ 個選擇線變數(即 $2^{n-1} \times 1$)的多工器，必定可以執行該函數。

換句話說，即用一個 2^n 對 1 MUX 可完成 $n+1$ 個變數的布氏函數。

執行方式: 取其中 n 個變數連接到多工器的選擇線，剩下一個變數用於多工器的輸入。

<例題> 利用多工器實現 $F(A, B, C) = A'B' + AC$

解: 3 變數 \rightarrow 可選擇 $2^{3-1} \times 1 = 4 \times 1$ 多工器

選擇 A, B 變數連接到 MUX 選擇線

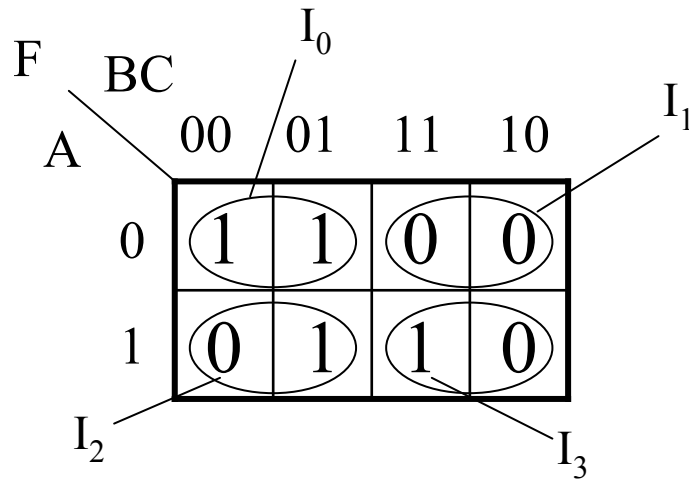
由 MUX 輸出的一般方程式

$$\begin{aligned} 1. \quad Y &= \sum_{k=0}^{2^n-1} m_k I_k \quad n: \text{控制輸入} \\ &= \sum_{k=0}^3 m_k I_k = m_0 I_0 + m_1 I_1 + m_2 I_2 + m_3 I_3 \end{aligned}$$

$$\text{又 } F = A' B' + A C$$

$$= \underbrace{A' B' C}_{m_0} + \underbrace{A' B' C'}_{I_0} + \underbrace{A B' C}_{m_2 I_2} + \underbrace{A B C}_{m_3 I_3} + m_1 \cdot 0_{I_1}$$

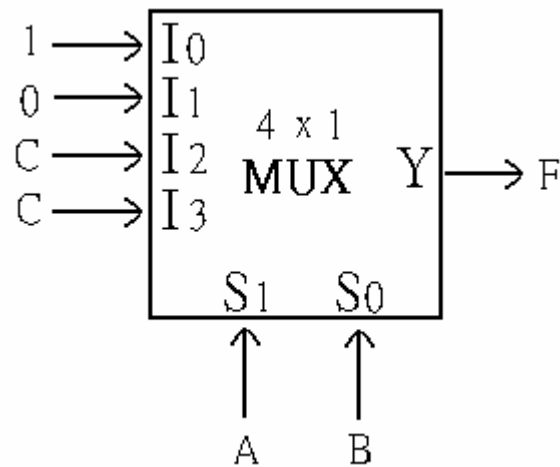
2. or



$$I_0 = 1, I_1 = 0$$

$$I_2 = C, I_3 = C$$

邏輯電路



3. 製作表

	I_0	I_1	I_2	I_3
C'	0	2	4	6
C	1	3	5	7
	1	0	C	C

$$F = A' B' + A C$$

$$= A' B' C + A' B' C' + A B' C + A B C$$

$$= \Sigma(0, 1, 5, 7)$$

(化為全及項)

<例題> 利用多工器實現以下交換函數

$$F(A, B, C, D) = \Sigma (0, 1, 3, 6, 7, 8, 11, 12, 14)$$

解： 4 變數 $\rightarrow n = 4$

1. 選 $2^4 \times 1$ MUX 亦即 $2^n \times 1$ MUX

則每一個 MUX 輸入為常數 0 或 1

2. 選 $2^{4-1} \times 1$ MUX 亦即 $2^{n-1} \times 1$ MUX = 8×1

MUX 將 A, B, C 接到 select line，D 作為 MUX input 用。

		CD			
		00	01	11	10
F AB	00	1	1	1	0
	01	0	0	1	1
	11	1	0	0	1
	10	1	0	1	0

得

$$I_0 = 1 \quad I_1 = D$$

$$I_2 = 0 \quad I_3 = 1$$

$$I_4 = D' \quad I_5 = D$$

$$I_6 = D' \quad I_7 = D'$$

不必外加邏輯閘，但須同時有補數形式與非補數形式的變數。

3. 選擇 $2^{4-2} \times 1$ MUX 亦即 2^{n-m} MUX = 4×1 MUX
將 A, B 接到 select line, C, D 作為 MUX input

F		CD			
		00	01	11	10
AB	00	1	1	1	0
	01	0	0	1	1
	11	1	0	0	1
	10	1	0	1	0

$$I_0 = C' + D$$

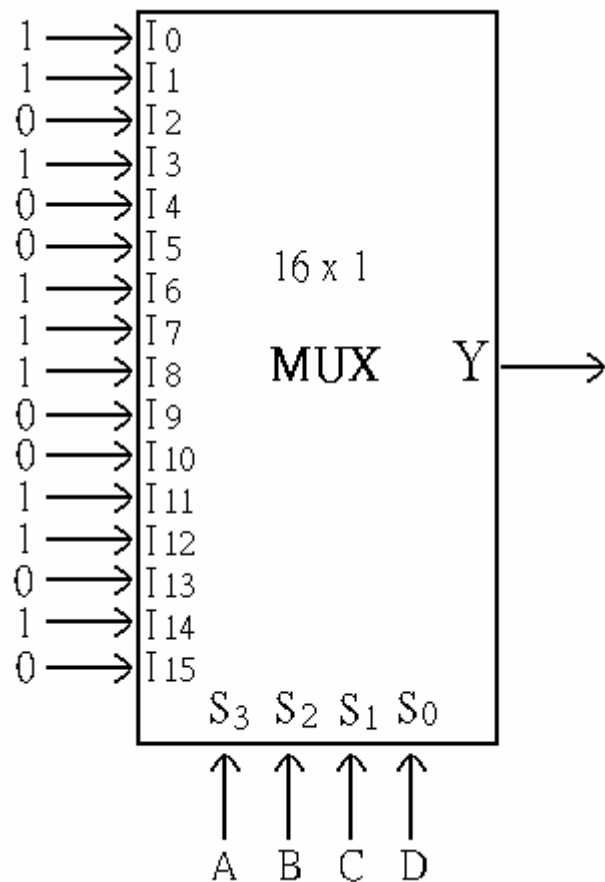
$$I_1 = C$$

$$I_3 = D'$$

$$I_2 = C \odot D = C' \oplus D$$

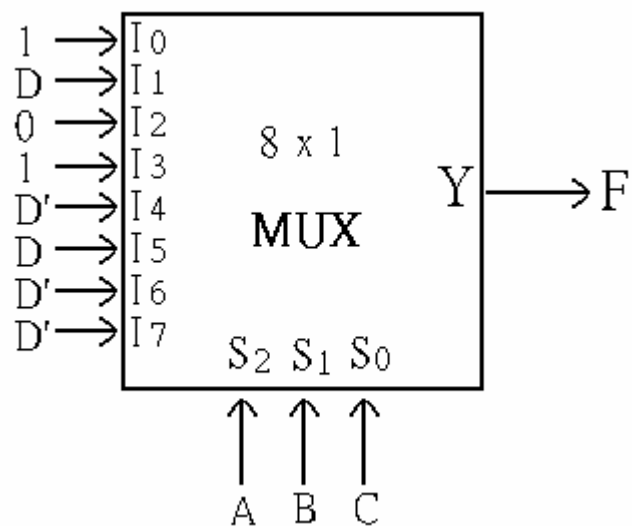
邏輯電路：

(1)

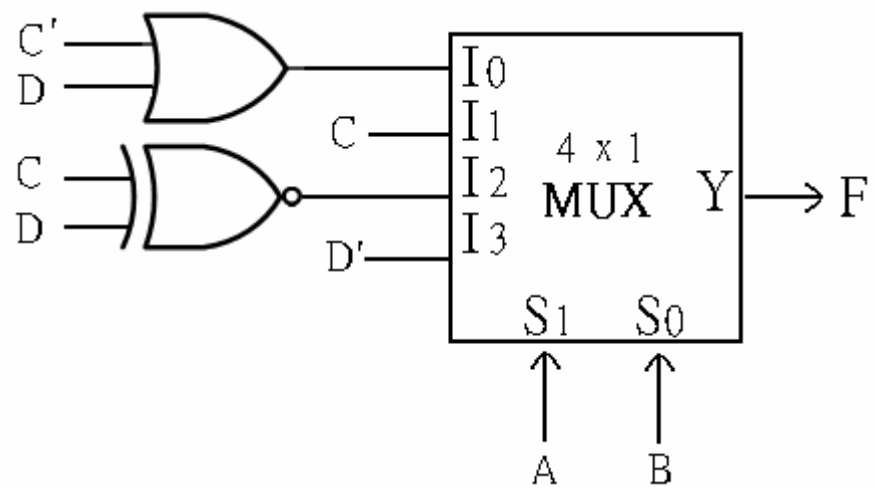
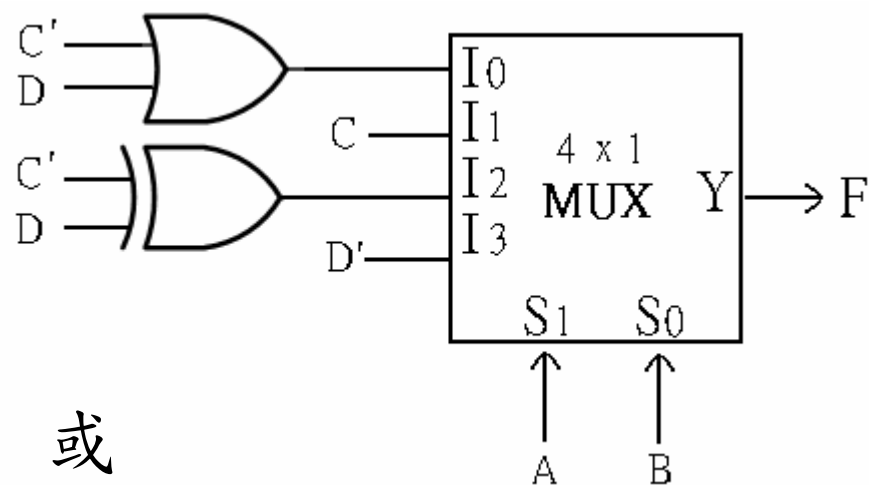



$$F = \Sigma (0,1,3,6,7,8,11,12,14)$$

(2)



(3)





(比較)討論:在多輸出函數的執行中，若使用多工器必須每一個函數一個多工器，但使用解碼器/解多工器時只需多加入 OR 閘(或 NAND 閘)而已，故對多輸出函數而言，使用 Decoder / DeMUX 執行時，通常較為經濟。

4-7 比較器(comparator)

一個 n 位元比較器為一個具有比較兩個 n 位元數目的大小，並指示出結果: $A > B$, $A = B$ 或 $A < B$ 的組合邏輯電路。

1. 單位元數值比較器

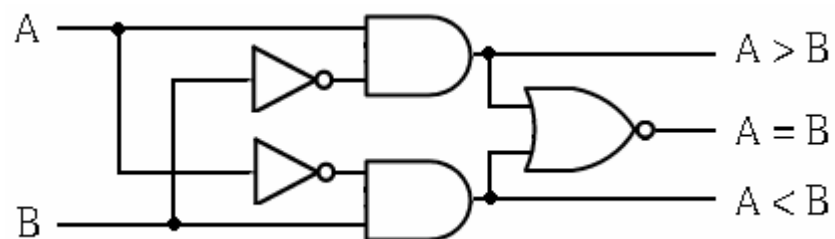
A	B	$A > B$	$A < B$	$A = B$
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

$$(A > B) = x = A B'$$

$$(A < B) = y = A' B$$

$$\begin{aligned}(A = B) = z &= A' B' + A B \\ &= A \odot B = \overline{A \oplus B}\end{aligned}$$

邏輯電路：



2. 二位元數值比較器

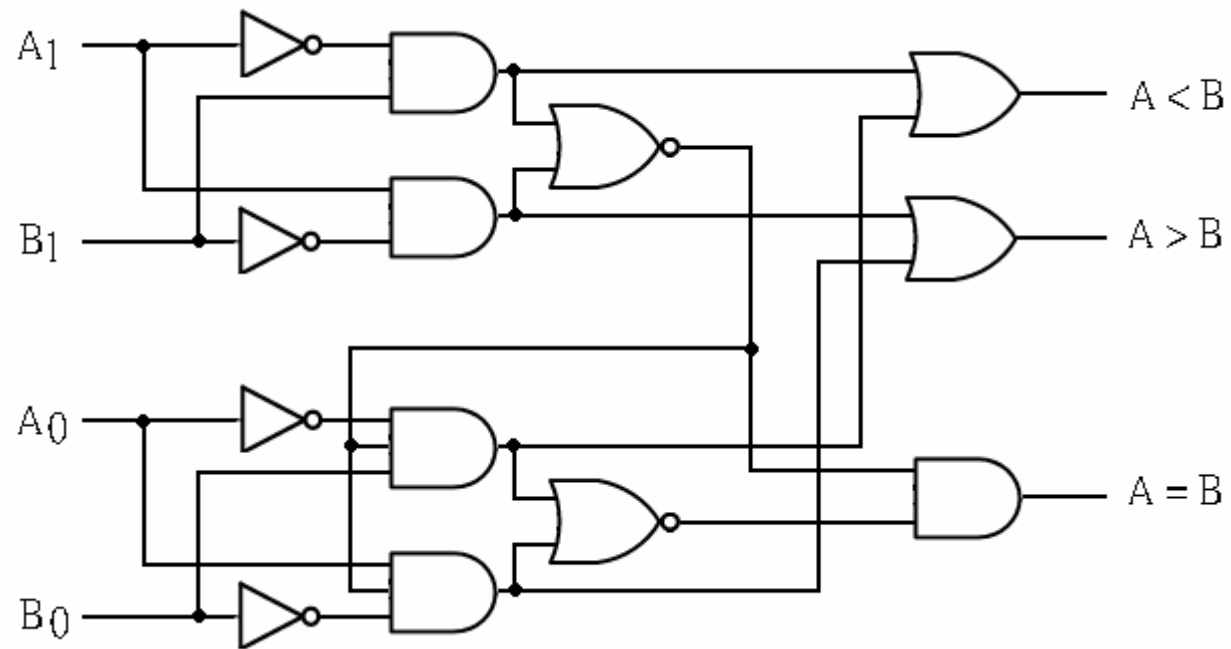
由前知 真值表大小為 2^{2^n} 種組合

則 $2^{2^n} = 2^4 = 16$

A_1	B_1	A_0	B_0	$A > B$	$A < B$	$A = B$	$A \neq B$
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	1
⋮				⋮			

2^{2^n} 種組合 \rightarrow 演算法(Algorithm)

: 為一套步驟以解決問題的程序



3. 四位元數值比較器

設A、B 二個數字各有四個位元，其係數如下

$$A = A_3 A_2 A_1 A_0$$

$$B = B_3 B_2 B_1 B_0$$

每對位元相等關係可用相當函數來表示

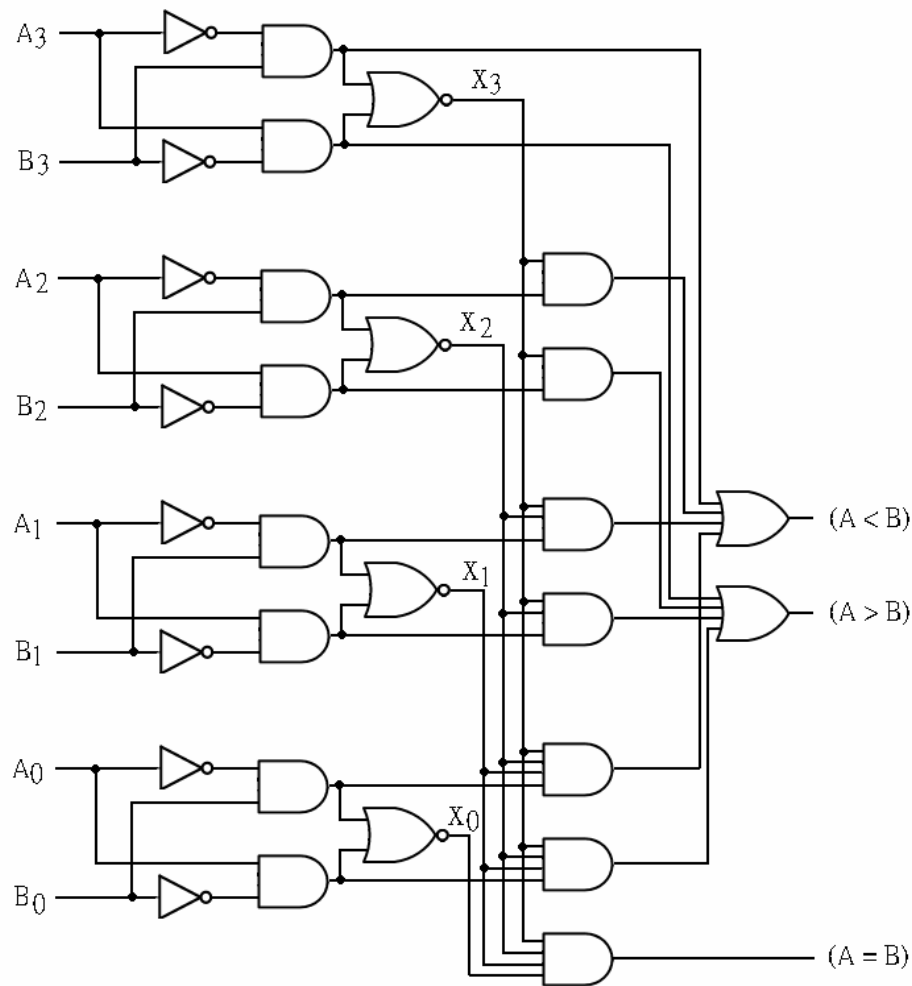
$$x_i = A_i B_i + A'_i B'_i, \quad i = 0, 1, 2, 3$$

$$(A=B) = x_3 x_2 x_1 x_0$$

$$(A>B) = A_3 B'_3 + x_3 A_2 B'_2 + x_3 x_2 A_1 B'_1 + x_3 x_2 x_1 A_0 B'_0$$

$$(A<B) = A'_3 B_3 + x_3 A'_2 B_2 + x_3 x_2 A'_1 B_1 + x_3 x_2 x_1 A'_0 B_0$$

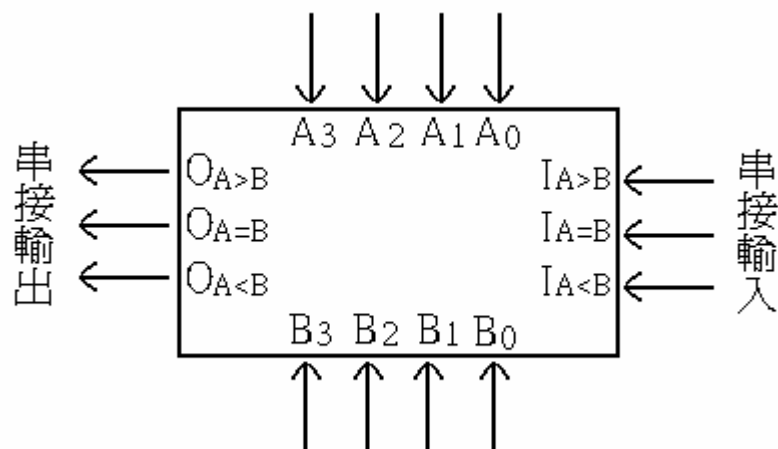
邏輯電路: 不等函數中含有許多相等函數，是以可利用許多相等函數中的線路來完成不等函數。



4 位元數值比較器

為能具有擴充能力，商用的 MSI 比較器通常還包括三個輸入端： $A < B$, $A = B$ 與 $A > B$ 。

例如 TTL 型 7485 四位元數值比較器。



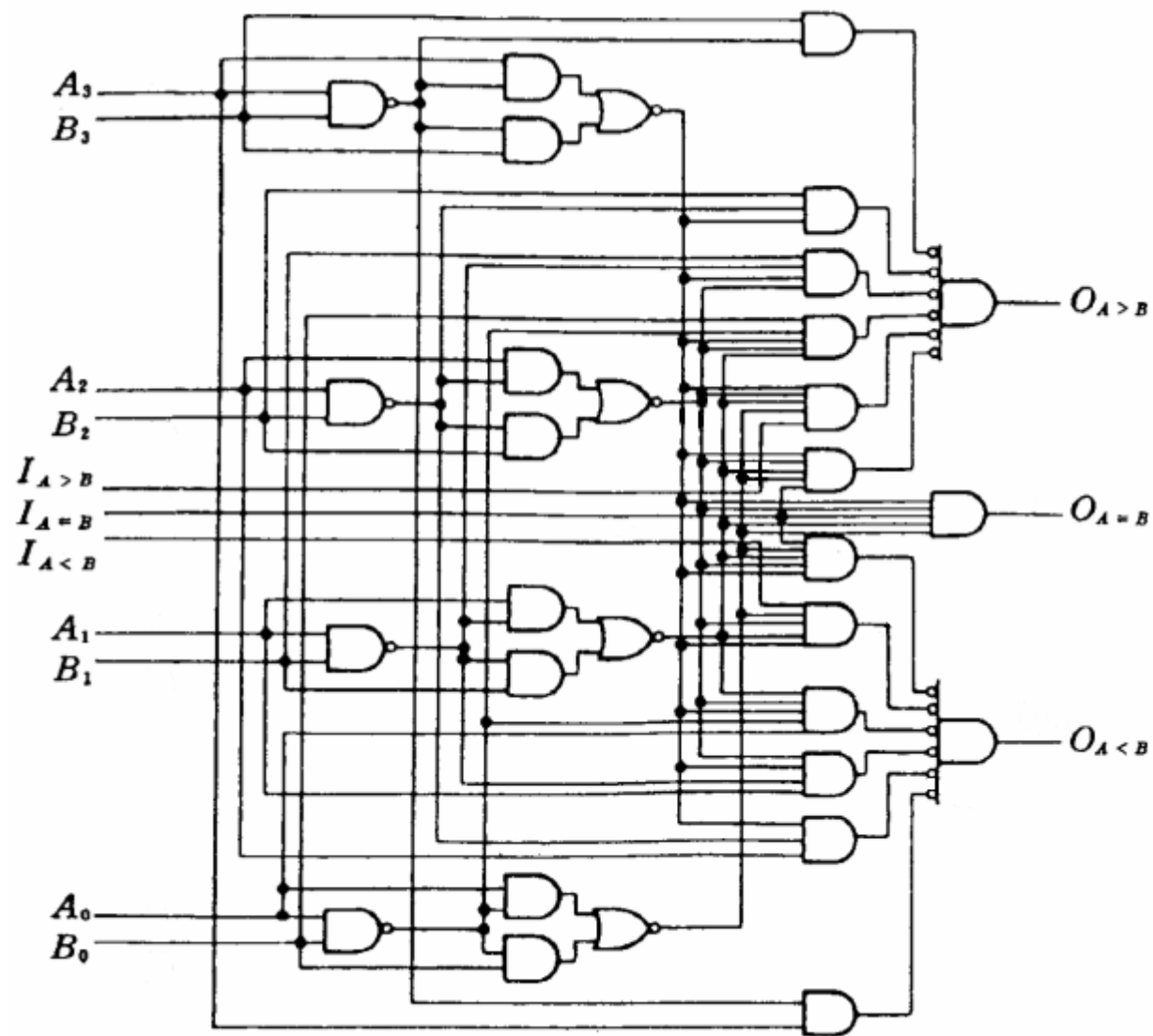
設計方法同上，只當 $A = B$ 時，多考慮 $I_{A>B}$, $I_{A=B}$ 及 $I_{A<B}$

此即 $I_{A>B} = 1$ 時 $O_{A>B} = 1$, $I_{A=B} = 1$ 時 $O_{A=B} = 1$, $I_{A<B} = 1$ 時 $O_{A<B} = 1$ 。

真值表

資料輸入				串級輸入			串級輸出		
$A_3 \cdot B_3$	$A_2 \cdot B_2$	$A_1 \cdot B_1$	$A_0 \cdot B_0$	$I_{A>B}$	$I_{A=B}$	$I_{A<B}$	$O_{A>B}$	$O_{A=B}$	$O_{A<B}$
$A_3 > B_3$	Φ	Φ	Φ	Φ	Φ	Φ	1	0	0
$A_3 < B_3$	Φ	Φ	Φ	Φ	Φ	Φ	0	0	1
$A_3 = B_3$	$A_2 > B_2$	Φ	Φ	Φ	Φ	Φ	1	0	0
$A_3 = B_3$	$A_2 < B_2$	Φ	Φ	Φ	Φ	Φ	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	Φ	Φ	Φ	Φ	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	Φ	Φ	Φ	Φ	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	Φ	Φ	Φ	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	Φ	Φ	Φ	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	1	0	0	1	0	0
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	0	1	0	0	1
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	0	1	0	0	1	0

邏輯電路：



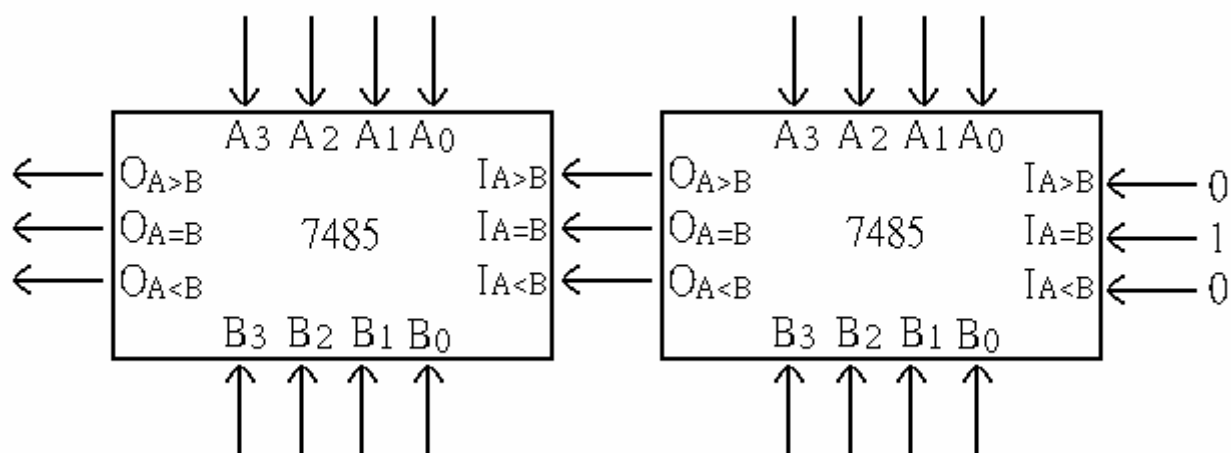
$$\begin{aligned}
 O'_{A>B} = & A'_3 B_3 + (A_3 \odot B_3) A'_2 B_2 + (A_3 \odot B_3) (A_2 \odot B_2) \\
 & A'_1 B_1 + (A_3 \odot B_3) (A_2 \odot B_2) (A_1 \odot B_1) A'_0 B_0 + \\
 & (A_3 \odot B_3) (A_2 \odot B_2) (A_1 \odot B_1) (A_0 \odot B_0) I_{A<B} + \\
 & (A_3 \odot B_3) (A_2 \odot B_2) (A_1 \odot B_1) (A_1 \odot B_0) I_{A=B}
 \end{aligned}$$

$$\begin{aligned}
 O'_{A<B} = & A_3 B'_3 + (A_3 \odot B_3) A_2 B'_2 + (A_3 \odot B_3) (A_2 \odot B_2) \\
 & A_1 B'_1 + (A_3 \odot B_3) (A_2 \odot B_2) (A_1 \odot B_1) A_0 B'_0 + \\
 & (A_3 \odot B_3) (A_2 \odot B_2) (A_1 \odot B_1) (A_0 \odot B_0) I_{A>B} + \\
 & (A_3 \odot B_3) (A_2 \odot B_2) (A_1 \odot B_1) (A_1 \odot B_0) I_{A=B}
 \end{aligned}$$

$$O_{A=B} = (A_3 \odot B_3) (A_2 \odot B_2) (A_1 \odot B_1) (A_0 \odot B_0) I_{A=B}$$

<例題> 試利用兩個 7485 4 位元比較器設計一個 8 位元比較器。

解：



4-8 可規劃邏輯元件(PLD)

可規劃邏輯元件(PLD)層 ASIC 一種，以 IC 包裝密度而言，PLD 元件為 LSI 或 VLSI。目前常用的幾種主要 PLD 元件為：

1. ROM (僅讀記憶體 read-only memory)
2. PLA (可規劃邏輯陣列, programmable logic array)
3. PAL (可規劃陣列邏輯, programmable array logic)

這些元件基本結構均為兩層 AND-OR 電路，因此可以執行任何交換函數。其間主要差異在於 AND 閘陣列與 OR 閘陣列的規劃性。

1. 基本觀念:

在邏輯觀點上，PLD 元件為一個具有熔絲(fuse)連接的邏輯閘陣列元件。

在電路設計上，它可以取代許多個由 SSI 或 MSI 等元件組成的電路模組。

因而可減少外部連接線數目、PC 板面積、與連接器數目, 結果使得硬體成本大大降低。

PLD 原理與應用

- PLD (Programmable Logic Device)可程式邏輯元件。
- PLD 能把幾個或幾十個 TTL 的元件功能容納進入。與 PLD 元件的功能大小有關。
- 降低成本、面積縮小，大大提高了穩定度。
- 設計過程: 選擇 PLD 元件，方程式輸入、編譯、模擬、燒錄，實驗到驗證結果。(以上為PAL, GAL方式)
- 編譯器: AMD/MMI 所發展之 PALASM 軟體。只要熟習一種 PLD 編譯器語法設計，當學習其他的發展軟體也能很快加以使用。

- PALASM 提供一套語法結構稱為狀態機器(state machine)語法，僅需要狀態圖或狀態表就可從事程式設計。使得在設計序向邏輯電路時更能得心應手。
- PLD 不但可取代多個系統的數位 IC，使電路板的體積大為偏小，而且電路具有保密性，產品的開發時間大為偏短，因此極具經濟性及競爭力。

- PLD 依製造廠商之不同及特性之差異而分別被命名為：

PAL (Programmable Array Logic)是PLD始祖，只能被燒錄一次。(雙極性元件)

EPLD (Erasable Programmable Logic Device)

採用CMOS EPROM的技術製造而成，可用紫外光照射清除，可重複使用。

GAL (Generic Array Logic)

採用EEPROM (Electrical Erasable Programmable ROM) (CMOS) 技術製造而成，使用更為方便，可重複燒錄100次以上，且資料可以在 GAL 內部保存20年以上。

PEEL (Programmable Electrically Erasable Logic)

也是採用CMOS EEPROM的技術製造而成，可重複燒錄1000次以上，燒入之資料可保存10年以上。

- EE 即時可重新規劃，清除和規劃只需要數秒時間。
- 市面上所設計的數位邏輯 PLD 訓練器，其設計方式是採用 PAL 的設計方式，再轉換燒到 GAL。
- PIC 16C5X 系列 CMOS 單晶片微電腦：Watch Dog Timer、RISC 電路，靜止模式 $< 10\mu\text{A}$ ，尤其保密鎖 (security fuse) 保障了所有設計人的心血。
- PLD 可程式邏輯元件，主要由 AND、OR、NOT、FF 及 FUSE 等組合而成。
- 利用 PLD 內部一部分 Fuse 燒斷，可設計所需元件。
- 只要懂得布林代數的邏輯設計概念，即可做 PLD 的電路設計。

- 
- 幾個常用PEEL元件:
PEEL18CV8, PEEL22CV10, PEEL22CV10Z
 - PLSI: Programmable Large Scale Integration
 - FPGA: Field programmable Gate Array

2. PLD 的規劃方式

PLD元件的規劃方式有：罩網規劃(mark programming) 與現場規劃(field programmable)。

罩網規劃IC (mark programmable IC)需要使用者提供規劃圖型(program 或 interconnection pattern)予IC製造商。一旦製造完成，IC 功能即告固定。

現場規劃可規劃IC (field programmable IC)，則由使用者，自行利用規劃器(programmable equipment)規劃所需規格。有些(IC)可以清除再重新規劃，有些則不能。

現場規劃通常使用在數位系統的初級階段，以獲較佳彈性；罩網可規劃則使用在系統設計完成後的大量出產，以降低成本。

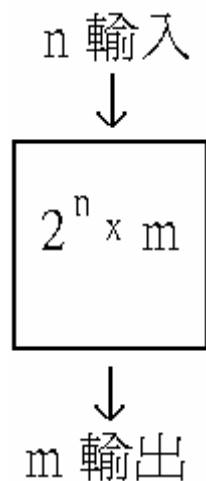
3. PLD 元件的規劃方式與對應元件表

<div>規劃方式</div> <div>元件類型</div>	單網規劃	現場規劃
ROM	ROM (Read Only Memory)	{ PROM (programmable ROM)(只能規劃一次) { EPROM (erasable prom) 可清除可規劃(紫外線) EEPROM (electrical EPROM)(電壓清除方式)
PLA	PLA	FPLA (Field PLA) 只能規劃一次
PAL	HAL	1.(只能清除一次)PAL 2. 可清除可規劃 EPLD* (erasable PLD) (紫外線) GAL* (電壓)(EEPLD)

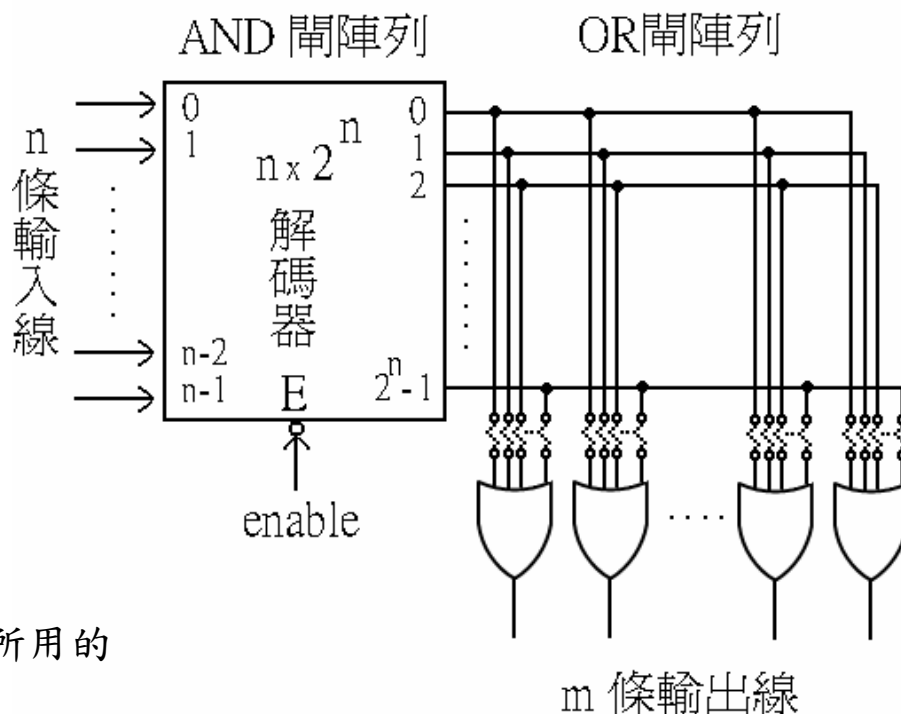
4. 僅讀記憶體(read-only memory, ROM)

一個 $2^n \times m$ 的 ROM 為一個 LSI 或 VLSI 元件，它包括一個 $n \times 2^n$ 解碼器與 m 個具有 2^n 個輸入的可規劃 OR 閘陣列。


$2^n \times m$ ROM 方塊圖



$2^n \times m$ ROM 的邏輯結構



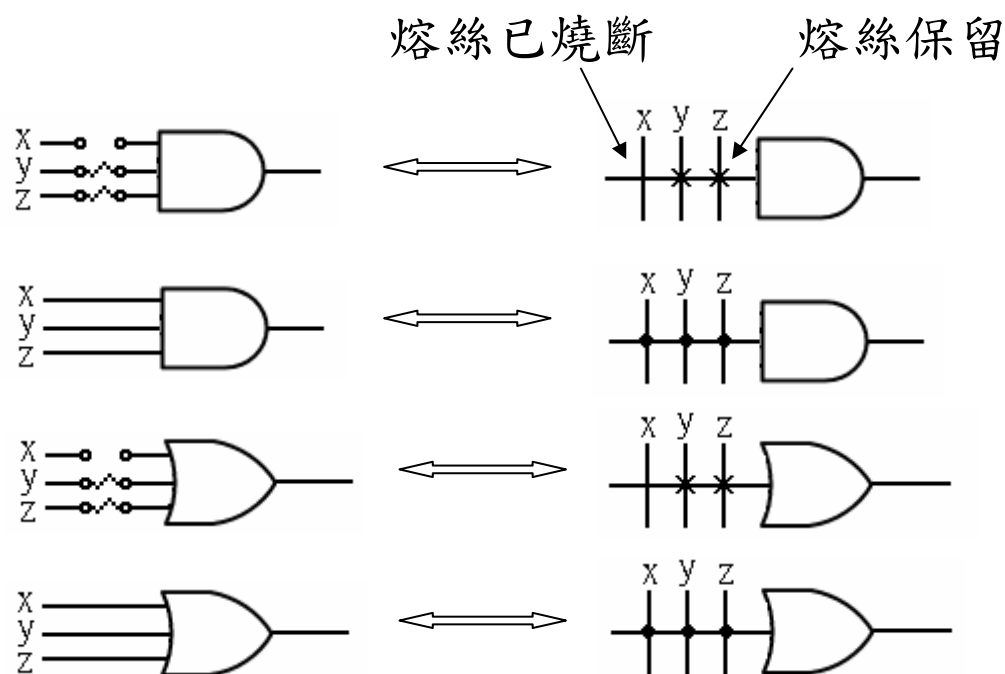
圖中熔絲在實際元件中，依所用的製造技術而有不同。



在現場可規劃元件中，若以雙極性的製造技術則為鎳鉻熔絲(nickel-chromium fuse)；若以 MOS 的製造技術，則為浮動閘極(floating gate) FET。

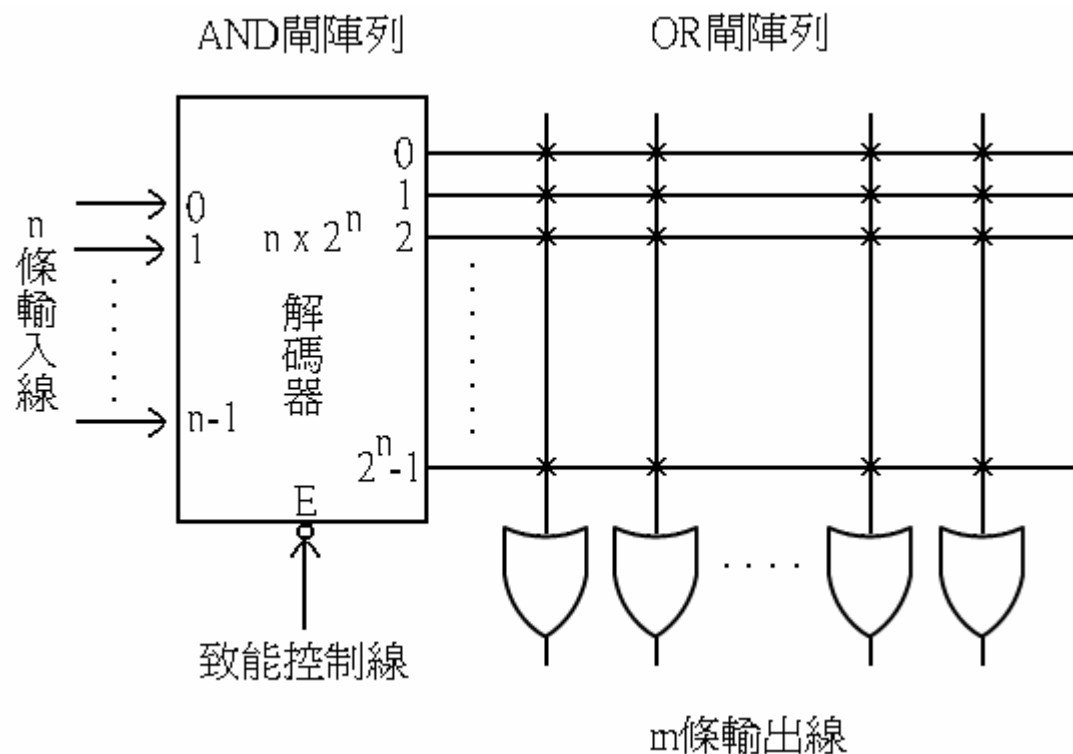
在罩網可規劃元件中，則直接以電晶體(雙極性元件)或 FET(MOS 元件)，代表 fuse 的存在可否。

為了討論與實際上設計或應用的方便，有關 PLD 元件的資料通常使用如下所列的簡便符號 (shorthand notation)。



註：有 X 符號表示該位置有熔絲(fuse)可以規劃。

是以 $2^n \times m$ ROM 的基礎結構又可表示成



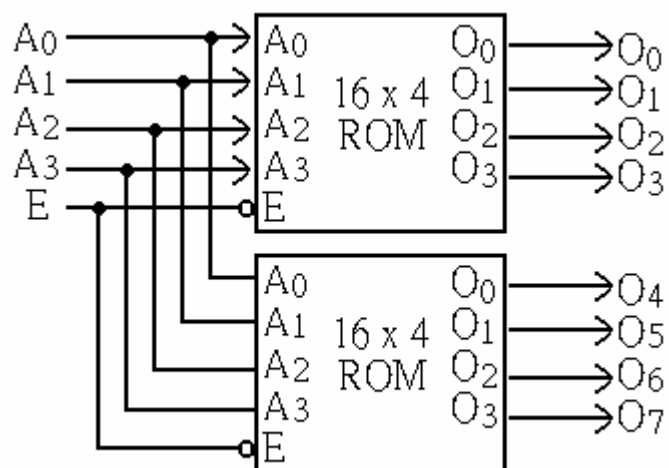
ROM 的擴充

ROM 的擴充方式分成語句長度與語句數目兩種。

（擴充語句長度係增加語句的位元數，擴充語句數目係增加語句的數目數）

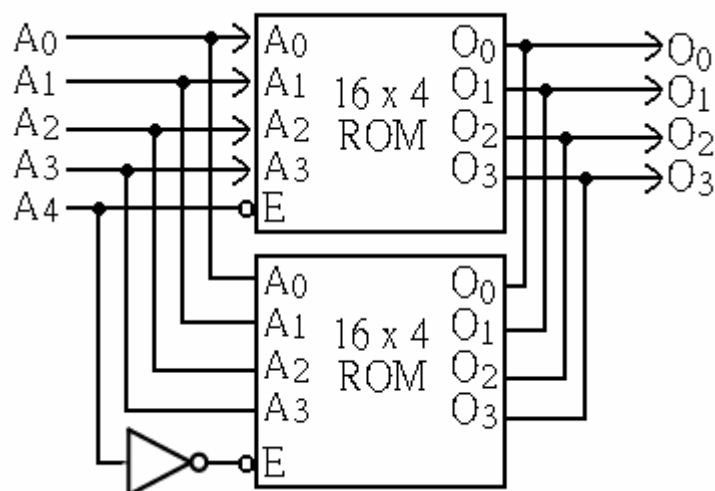
<例題> (語句長度擴充)利用兩個 16×4 ROM 設計一個 16×8 的ROM

解：



<例題> (語句數目擴充)利用兩個 16×4 ROM 設計
一個 32×4 的ROM

解：



執行交換函數

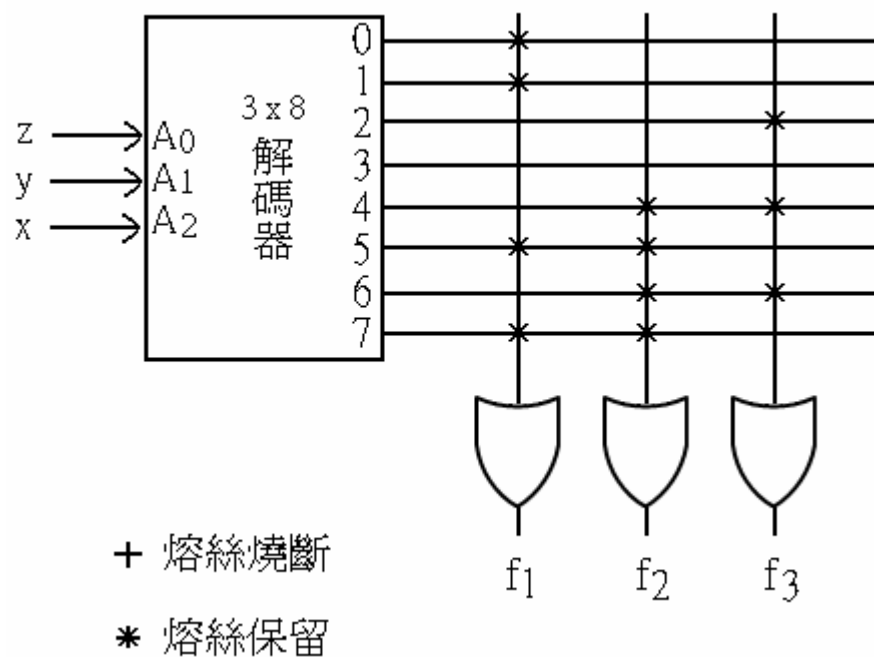
由前之 ROM 的基本結構為 AND 與 OR 兩個邏輯閘陣列，同時其 AND 閘陣列可以產生所有輸入變數的最小項，即它為一個標準 SOP 型式的多輸出函數兩層電路。因此，ROM 可以執行任意的交換函數。

<例題>：利用一個 $2^3 \times 3$ 的 ROM 執行下列交換函數：

$$f_1(x, y, z) = \Sigma(0, 1, 5, 7)$$

$$f_2(x, y, z) = \Sigma(4, 5, 6, 7)$$

$$f_3(x, y, z) = \Sigma(2, 4, 6)$$



- (1) 其實在利用 ROM 設計一個線路時，不需要像上例將內部熔絲的連接繪出，此處只為了說明方便才全部繪出。設計者只需指定 ROM 的編號及 ROM 的真值表。因真值表已提供 ROM 程式規劃的全部資訊。
- (2) 一般而言，利用 ROM 執行 m 個 n 個變數的交換函數時，所需的 ROM 容量為 $2^n \times m$ 個位元。但有時候，若仔細觀察欲執行的交換函數，往往可以消去一些不必要的輸入變數或輸出變數，進而減少 ROM 的容量。

<例題> 利用 ROM 設計一組合線路，此線路接收一 3 位元的數字，產生一個等於輸入數字平方的二進位數字。

解：

輸入			輸出					
x	y	z	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	0	0	1

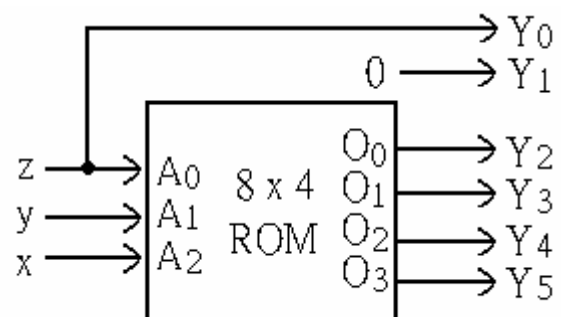
仔細觀察真值表知

$Y_0 = z$

Y_1 皆為 0

實際上以 ROM 執行的部分

邏輯電路方塊圖



(3) 另一種可節省 ROM 容量的方法為採用多級 ROM 電路。

<例題> 利用多級 ROM 電路執行下列交換函數

$$f_1(u, v, w, x, y, z) = \Sigma(4, 5, 15, 29, 42, 45, 47, 53, 58, 61)$$

$$f_2(u, v, w, x, y, z) = \Sigma(5, 20, 41, 42, 47, 61)$$

$$f_3(u, v, w, x, y, z) = \Sigma(4, 15, 29, 42, 47, 53, 63)$$

$$f_4(u, v, w, x, y, z) = \Sigma(4, 5, 15, 29, 42)$$

解：為方便起見，真值表只列出有被 $f_1 \sim f_4$ 包含的最小項。若直接以 ROM 執行，須要的 ROM 容量為 $2^6 \times 4 = 256$ 位元。

真值表

十進制	輸入						輸出			
	u	v	w	x	y	z	f ₁	f ₂	f ₃	f ₄
4	0	0	0	1	0	0	1	0	1	1
5	0	0	0	1	0	1	1	1	0	1
15	0	0	1	1	1	1	1	0	1	1
20	0	1	0	1	0	0	0	1	0	0
29	0	1	1	1	0	1	1	0	1	1
41	1	0	1	0	0	1	0	1	0	0
42	1	0	1	0	1	0	1	1	1	1
45	1	0	1	1	0	1	1	0	0	0
47	1	0	1	1	1	1	1	1	1	0
53	1	1	0	1	0	1	1	0	1	0
58	1	1	1	0	1	0	1	0	0	0
61	1	1	1	1	0	1	1	1	0	0
63	1	1	1	1	1	1	0	0	1	0
其 它 組 合							0	0	0	0

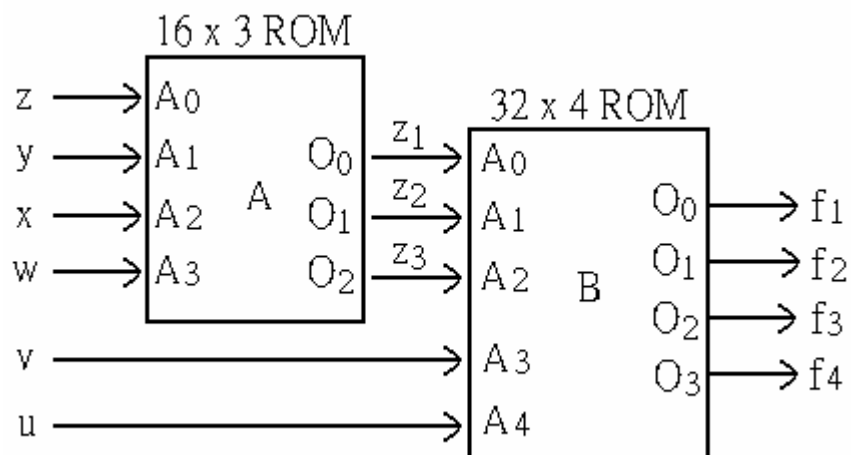
共用項與編碼(ROM A的真值表)

輸入				編碼輸出		
w	x	y	z	z ₃	z ₂	z ₁
0	1	0	0	0	0	0
0	1	0	1	0	0	1
1	0	0	1	0	1	0
1	0	1	0	0	1	1
1	1	0	1	1	0	0
1	1	1	1	1	0	1
其它組合				1	1	1

ROM B 的真值表

u	v	z ₃	z ₂	z ₁	f ₁	f ₂	f ₃	f ₄
0	0	0	0	0	1	0	1	1
0	0	0	0	1	1	1	0	1
0	0	1	0	1	1	0	1	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	1	1
1	0	0	1	0	0	1	0	0
1	0	0	1	1	1	1	1	1
1	0	1	0	0	1	0	0	0
1	0	1	0	1	1	1	1	0
1	1	0	0	1	1	0	1	0
1	1	0	1	1	1	0	0	0
1	1	1	0	0	1	1	0	0
1	1	1	0	1	0	0	1	0
Φ	Φ	1	1	1	0	0	0	0
其 它 組 合					0	0	0	0

邏輯電路



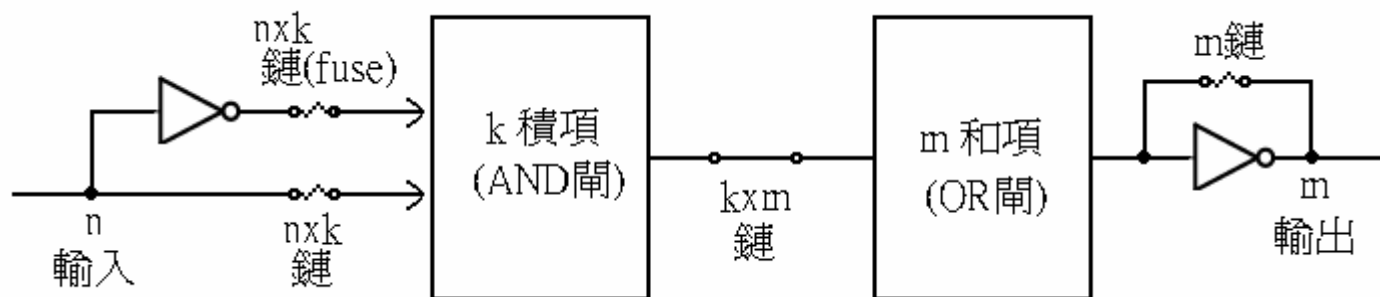
5. 程式邏輯陣列(programmable logic array, PLA)

組合線路可能有不採條件，當用 ROM 製作時，不採條件變成一定不會發生的位址輸入，而形成浪費。因此在不採條件過多時，使用程式邏輯陣列(簡稱PLA)較經濟。

在觀念上 PLA 類似 ROM，但是 PLA 並不提供變數的全部解碼，不產生 ROM 中的所有最小項。ROM 內的解碼器在此處換成 AND 陣列。

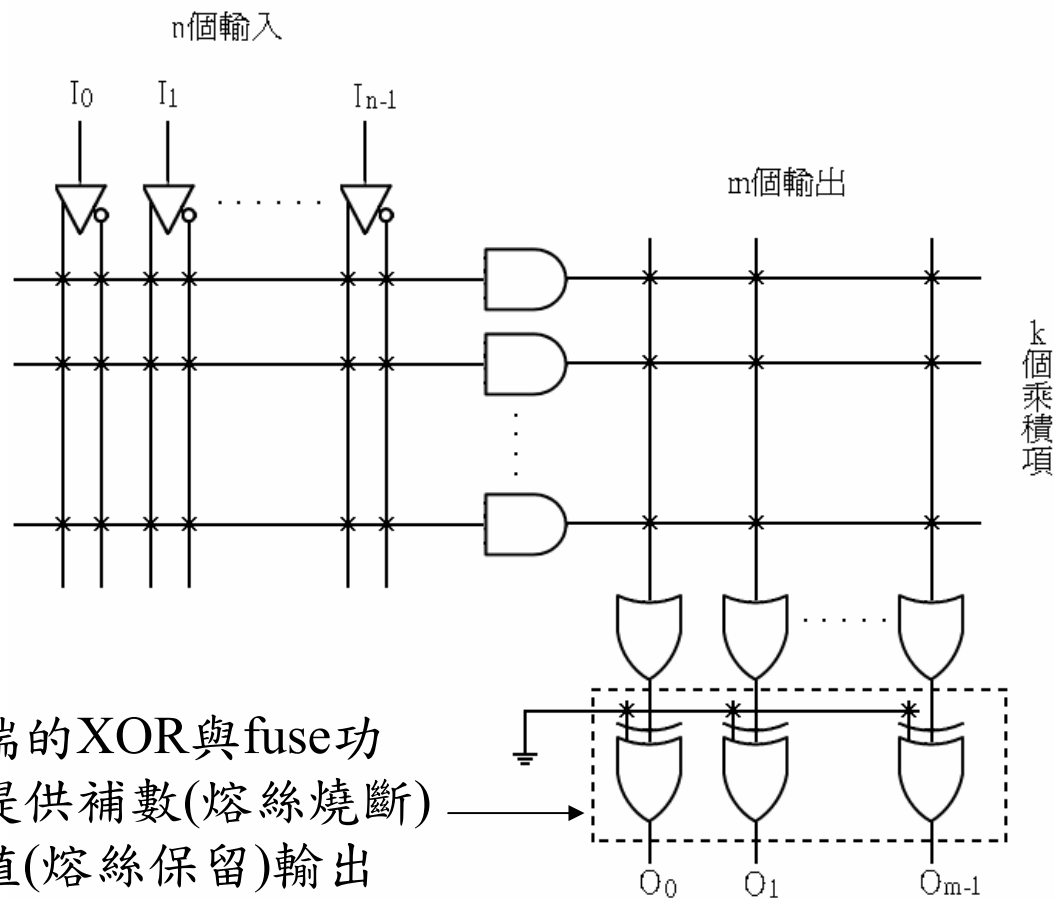
典型 PLA 的大小以輸入數目(n)，積項數目(k)及輸出數目(和項數目等於輸出數目 m)來指定。

典型的 $n \times k \times m$ PLA 電路方塊圖



熔絲數目為 $2n \times k + k \times m + m$

典型 $n \times k \times m$ PLA 內部結構電路圖：



執行交換函數

在利用 PLA 元件執行交換函數時，首先利用多輸出交換函數的化簡求得多輸出函數的最簡式後，再由這些最簡式建立一個指定 PLA 熔絲規劃的規劃表(programming table)。

為了求得最少項數，函數 F 與其補數 F' 均應予簡化以確定那一個具有較少的積項，再以較少的形式來製作。

<例題> 利用三個輸入，四個積項，及二個輸出的
PLA 製作下列函數：

$$F_1(A, B, C) = \Sigma(3, 5, 6, 7)$$

$$F_2(A, B, C) = \Sigma(0, 2, 4, 7)$$

解：原形式與補數形式均化簡如下

F ₁	A	BC			
		00	01	11	10
0				1	
1			1	1	1

$$F_1 = AB + AC + BC$$

F ₂	A	BC			
		00	01	11	10
0	1				1
1	1			1	

$$F_2 = A'C' + ABC + B'C'$$

		BC			
		00	01	11	10
A	0	Φ	Φ		Φ
	1	Φ			

$$F'_1 = B'C' + A'B' + A'C'$$

		BC			
		00	01	11	10
A	0		Φ	Φ	
	1		Φ		Φ

$$F'_2 = B'C + A'C + ABC'$$

選出最少積項數目的函數為：

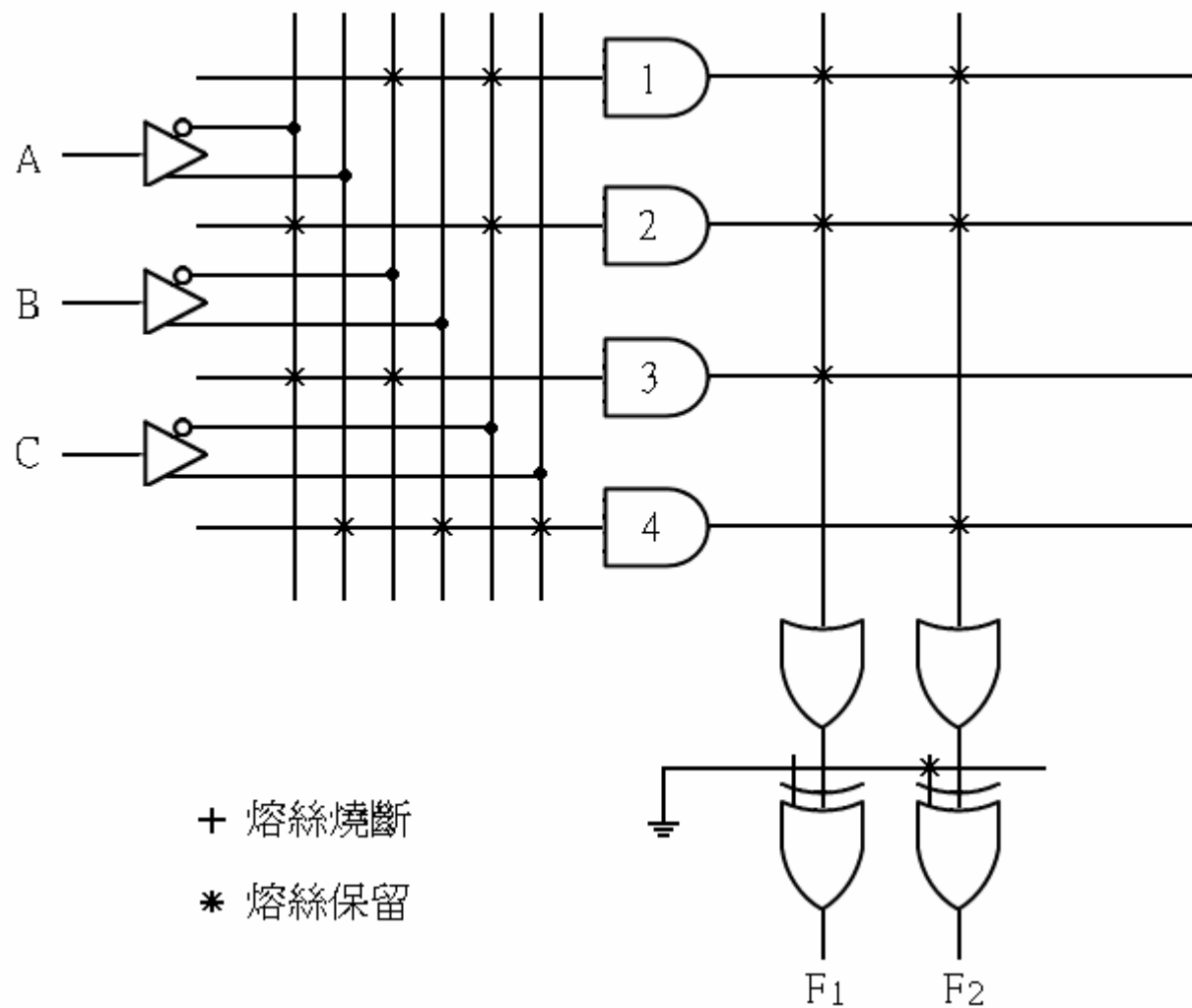
$$F'_1 = B'C' + A'B' + A'C'$$


$$F_2 = A'C' + ABC + B'C'$$

PLA 熔絲規劃表

	積項	輸入			輸出	
		A	B	C	F ₁	F ₂
B'C'	1	—	0	0	1	1
A'C'	2	0	—	0	1	1
A'B'	3	0	0	—	1	—
ABC	4	1	1	1	—	1
					C	T
					T/C	

PLA 執行交換函數





用 PLA 設計一個數位系統時，不需要像上例繪出內部的詳細連接，只要做出 PLA 熔絲規劃表。從表中即可編排 PLA 的適當路徑。

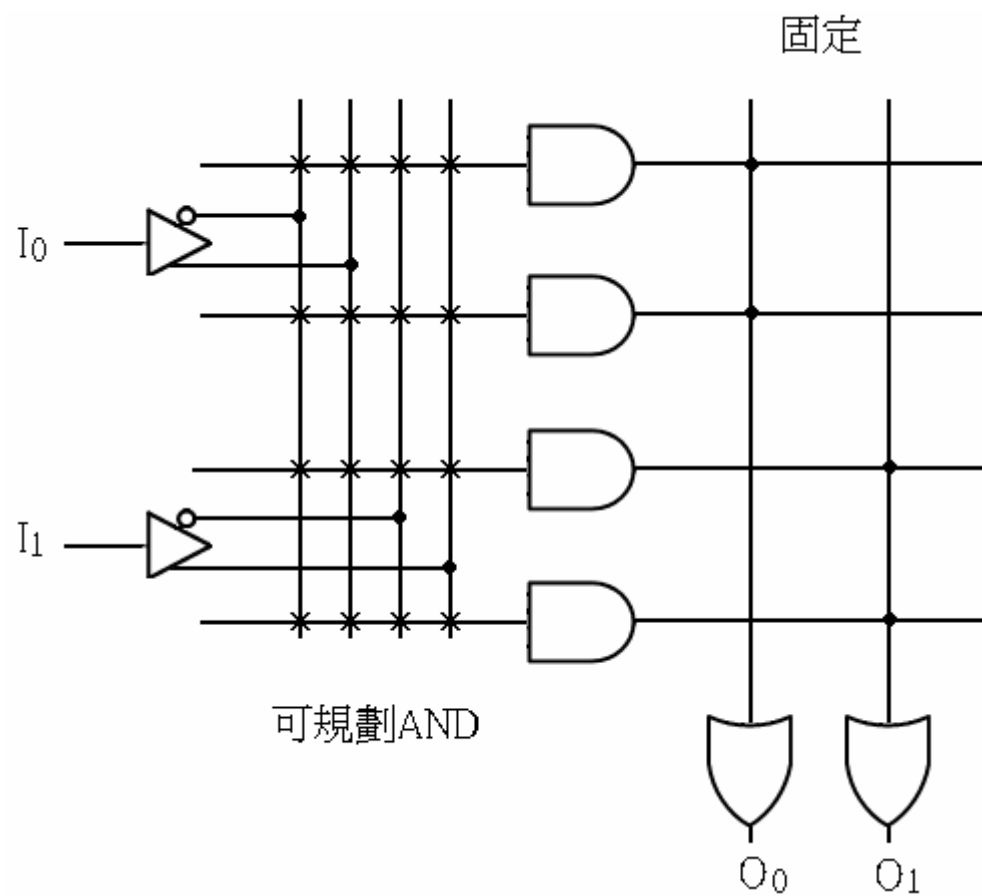
本例只是用於說明而已，典型的商用 PLA 超過 10 個輸入與大約 50 個積項，要簡化這麼多的變數函數，以列表法或利用計算機幫助才能完成。

6. 可程式規劃的陣列邏輯(programmable array logic, PAL)

PAL 為 PLA 的一種(特例)，其特性為只有 AND 閘陣列可以規劃而 OR 閘陣列固定，因此 PAL 較便宜且較易使用但較沒有彈性。其基本結構與 PLA 相同。(有些 PAL 具有 D 型 FFs 輸出電路，因此可以執行序向電路)

在設計多輸出交換函數電路時，若使用 PLA 則可以有多個交換函數共用一個乘積項，但若使用 PAL 則因其 OR 閘陣列不能規劃，因而無法共用乘積項。因此，使用 PAL 元件執行多輸出交換函數時，只須使用單一輸出交換函數化簡後一一執行即可，並不需考慮多輸出交換函數化簡之共用項問題。

PAL 邏輯結構圖



<例題> 利用 PAL 執行下列多輸出交換函數

$$f_1(w, x, y, z) = \Sigma(2, 12, 13)$$

$$f_2(w, x, y, z) = \Sigma(7, 8, 9, 10, 11, 12, 13, 14, 15)$$

$$f_3(w, x, y, z) = \Sigma(0, 2, 3, 4, 5, 6, 7, 8, 10, 11, 15)$$

$$f_4(w, x, y, z) = \Sigma(1, 2, 8, 12, 13)$$

解：利用單一輸出函數化簡程序得 $f_1 \sim f_4$ 的最簡式分別為：

$$f_1 = wxy' + w'x'yz'$$

$$f_2 = w + xyz$$

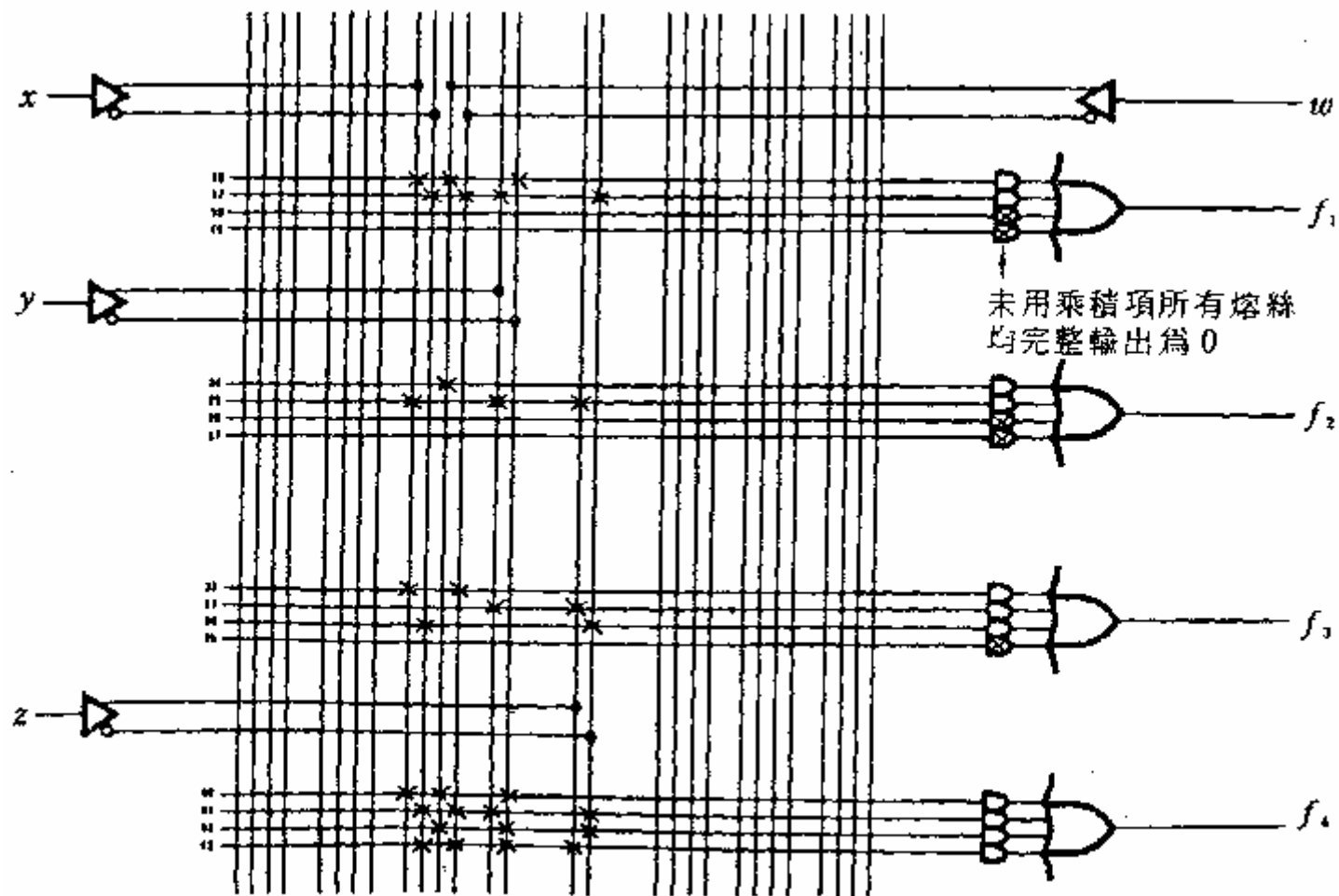
$$f_3 = w'x + yz + x'z'$$

$$f_4 = wxy' + w'x'yz' + wy'z' + w'x'y'z$$

PAL 規劃表

乘積項		AND 輸入	輸出
		w x y z	
wxy'	16	1 1 0 —	f ₁
w'x'yz'	17	0 0 1 0	
w	24	1 — — —	f ₂
xyz	25	— 1 1 1	
w'x	32	0 1 — —	f ₃
Yz	33	— — 1 1	
x'z'	34	— 0 — 0	
wxy'	40	1 1 0 —	f ₄
w'x'yz'	41	0 0 1 0	
wy'z'	42	1 — 0 0	
w'x'y'z	43	0 0 0 1	

邏輯電路



7. 結論

- (1) 三種 PLD 元件的基本差異在於 AND 閘陣列與 OR 閘陣列的規劃性。在 ROM 中，AND 閘陣列產生了所有輸入變數的最小項，因此不需要規劃，而 OR 閘陣列則可規劃的，以執行交換函數；在 PLA 中，AND 閘與 OR 閘兩個陣列均為可規劃的；在 PAL 中，AND 閘陣列可以規劃而 OR 閘陣列則規定不能規劃。

(2) 交換函數的執行方式有很多，簡單的函數可由 SSI 邏輯閘或 MSI 多工器與解碼器(解多工器)完成，較複雜的函數則使用 ROM、PAL 或 PLA 等元件來執行較為經濟，同時速度也較快。

又輸入變數少的時候，ROM 一般比 PLA 經濟，反之 PLA 通常能夠比 ROM 更經濟的解決問題。