第5章

組合電路

滄海書局
Tsang Hai Book Publishing Co.

■5.1 組合邏輯
■5.2 常見的組合電路
■5.3 加法器和減法器
  ■5.3.1 半加法器
  ■5.3.2 全加法器
  ■5.3.3 漣波進位加法器
  ■5.3.4 進位前看加法器
  ■5.3.5 半減法器
  ■5.3.6 全減法器
  ■5.3.7 加減法器
■5.4 比較器
■5.5 多工器
■5.6 解多工器
■5.7 編碼器
■5.8 解碼器

5.1 組合邏輯

組合邏輯電路 **(combinational logic circuit)** 是邏輯電路的一種，它是由輸入變數、輸出變數和許多邏輯閘組合而成，由於不包含任何記憶元件 (memory device) 與反饋線 (feedback line)，因此輸出訊號值只和目前的輸入訊號值有關。圖 5.1 是組合電路的概念示意圖，它包含一組 *m* 位元的輸入與一組 *n* 位元的輸出。我們可以依據輸入和輸出間的邏輯關係與時間延遲的需求，設計出組合電路內部的邏輯閘結構及閘與閘之間的連結方式。

與「組合邏輯電路」相對的另一種邏輯電路則是「循序邏輯電路」(sequential logic circuit)。由於循序電路使用閂鎖器 (latch) 或正反器 (flip-flop) 來暫存資料或狀態，它的輸出結果除了和目前的輸入值有關之外，也與先前的輸入值、輸出值都有關係。圖 5.2 是循序邏輯電路的示意圖，其詳細功能則在第 6 章才介紹。

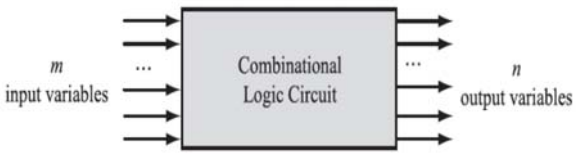


圖 5.1 組合電路的概念示意圖

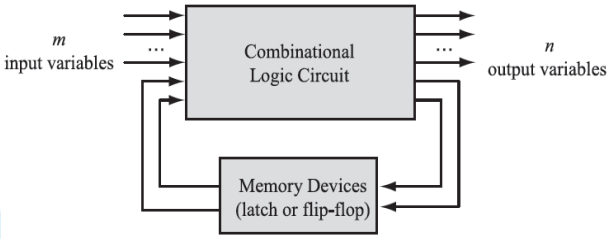


圖 5.2 循序電路的概念示意圖

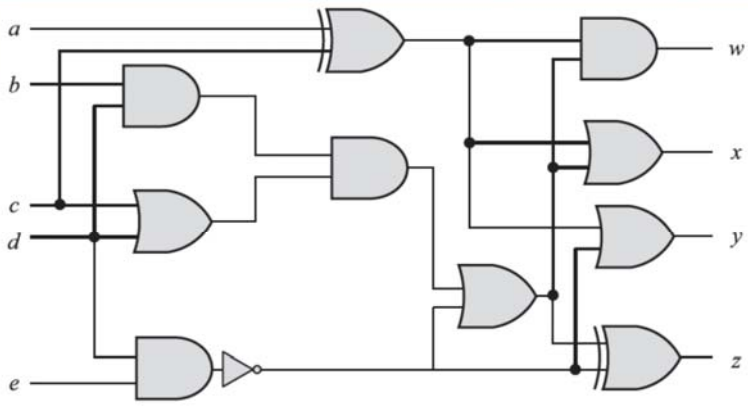


圖 5.3 一個 5 位元輸入 4 位元輸出組合電路的邏輯電路圖

假設某一個組合電路的內部詳細邏輯電路圖如圖 5.3 所示，包含 5位元的輸入與 4 位元的輸出。仔細觀察此電路結構，可以得出下列結論：(1) 組合邏輯僅僅由許多不同種類的邏輯閘連接組成，訊號是由某一邊傳遞到另一邊 (無任何反饋線)；(2) 它沒有記憶單元與反饋線，所以無任何暫存資料或狀態的功能；(3) 它任一時刻的輸出值，僅僅與目前的 (present) 輸入值有關，而與先前的 (previous) 輸入值完全無關。若考慮邏輯閘的傳遞延遲時間，我們可得下列結論：「當某一特定輸入訊號進入組合電路並經過一段傳遞延遲時間後，最後可以在輸出端得到針對此輸入訊號產生的穩定輸出訊號。」

很顯然地，第 4 章介紹卡諾圖化簡時所提及的電路，都不包含記憶元件與反饋線，故那些電路都是歸類成組合電路。

5.2 常見的組合電路

要設計一個組合電路，最標準的作法是使用在 4.9 節曾介紹過的六個設計步驟：

1. 詳細了解欲設計電路的功能與輸入、輸出之間的關係。
2. 根據輸入與輸出之間的關係，列出「真值表」。
3. 根據真值表，畫出其卡諾圖 (每一個輸出變數畫出一個卡諾圖)。
4. 進行卡諾圖化簡，得出「積項的和」或「和項的積」的最簡布林代數表示式。
5. 以 and、or 或 not 邏輯閘將最簡代數表示式轉換成電路。
6. 將電路依需要轉換成全部以 nand 閘或 nor 閘實現的電路。

範例 5.1
有一種和 BCD 類似的十進碼，稱之為「2-4-2-1」碼 (其中每個數字代表相對應位置的權重)，這種碼的編碼方式與 BCD 碼的關係如下表 (BCD 碼只有從 0 到 9) 請設計一個轉換器組合電路，將輸入的BCD 碼轉換成相對應的 2-4-2-1 碼。

十進數	BCD 碼	2-4-2-1 碼
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1
4	0 1 0 0	0 1 0 0
5	0 1 0 1	1 0 1 1
6	0 1 1 0	1 1 0 0
7	0 1 1 1	1 1 0 1
8	1 0 0 0	1 1 1 0
9	1 0 0 1	1 1 1 1
10 (不使用)	1 0 1 0	0 1 0 1
11 (不使用)	1 0 1 1	0 1 1 0
12 (不使用)	1 1 0 0	0 1 1 1
13 (不使用)	1 1 0 1	1 0 0 0
14 (不使用)	1 1 1 0	1 0 0 1
15 (不使用)	1 1 1 1	1 0 1 0

作法：
假設四個輸入變數標示為 a、b、c、d，四個輸出變數標示為 w、x、y、z，則此電路的方塊圖如下所示：

我們希望設計一個能將輸入 BCD 碼 (abcd) 轉換成相對應 2-4-2-1 碼 (wxyz) 的轉換電路。根據題目所提供的轉換表，可得出下列真值表

a	*b*	*c*	*d*	*w*	*x*	*y*	*z*	
0	0	0	0	0	0	0	0	m_0
0	0	0	1	0	0	0	1	m_1
0	0	1	0	0	0	1	0	m_2
0	0	1	1	0	0	1	1	m_3
0	1	0	0	0	1	0	0	m_4
0	1	0	1	1	0	1	1	m_5
0	1	1	0	1	1	0	0	m_6
0	1	1	1	1	1	0	1	m_7
1	0	0	0	1	1	1	0	m_8
1	0	0	1	1	1	1	1	m_9
1	0	1	0	*X*	*X*	*X*	*X*	m_{10}
1	0	1	1	*X*	*X*	*X*	*X*	m_{11}
1	1	0	0	*X*	*X*	*X*	*X*	m_{12}
1	1	0	1	*X*	*X*	*X*	*X*	m_{13}
1	1	1	0	*X*	*X*	*X*	*X*	m_{14}
1	1	1	1	*X*	*X*	*X*	*X*	m_{15}

畫出相對應的卡諾圖如下：

w

ab \ cd	00	01	11	10
00	0 m_0	0 m_1	0 m_3	0 m_2
01	0 m_4	1 m_5	1 m_7	1 m_6
11	X m_{12}	X m_{13}	X m_{15}	X m_{14}
10	1 m_8	1 m_9	X m_{11}	X m_{10}

x

ab \ cd	00	01	11	10
00	0 m_0	0 m_1	0 m_3	0 m_2
01	1 m_4	0 m_5	1 m_7	1 m_6
11	X m_{12}	X m_{13}	X m_{15}	X m_{14}
10	1 m_8	1 m_9	X m_{11}	X m_{10}

y

ab \ cd	00	01	11	10
00	0 m_0	0 m_1	1 m_3	1 m_2
01	0 m_4	1 m_5	0 m_7	0 m_6
11	X m_{12}	X m_{13}	X m_{15}	X m_{14}
10	1 m_8	1 m_9	X m_{11}	X m_{10}

z

ab \ cd	00	01	11	10
00	0 m_0	1 m_1	1 m_3	0 m_2
01	0 m_4	1 m_5	1 m_7	0 m_6
11	X m_{12}	X m_{13}	X m_{15}	X m_{14}
10	0 m_8	1 m_9	X m_{11}	X m_{10}

進行化簡如下：

w

ab \ cd	00	01	11	10
00				
01		1	1	1
11	X	X	X	X
10	1	1	X	X

x

ab \ cd	00	01	11	10
00				
01	1		1	1
11	X	X	X	X
10	1	1	X	X

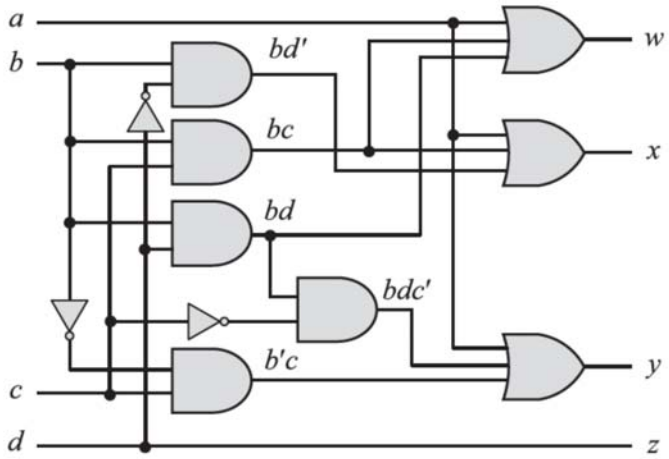
y

ab \ cd	00	01	11	10
00			1	1
01		1		
11	X	X	X	X
10	1	1	X	X

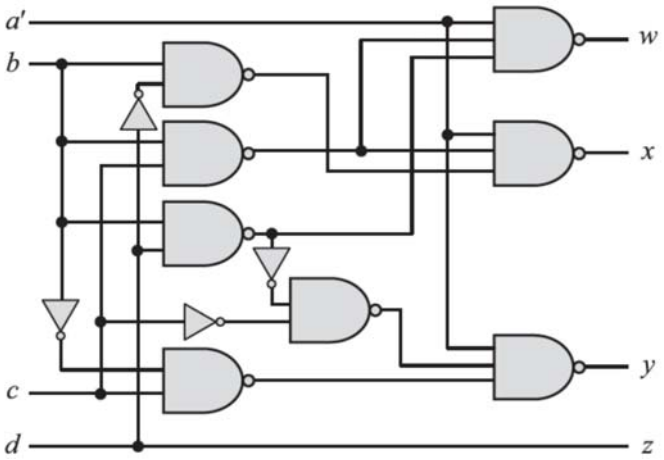
z

ab \ cd	00	01	11	10
00		1	1	
01		1	1	
11	X	X	X	X
10		1	X	X

故 $w = a + bd + bc$、$x = a + bc + bd'$、$y = a + d'c + bc'd$、$z = d$。
其邏輯電路圖如下圖，



其 nand 閘實現電路為



很明顯地，這兩個實現電路的訊號是由左邊輸入傳遞到右邊輸出，因為沒有記憶單元與反饋線，所以是組合電路。

本章主要希望應用前面介紹的知識和技巧，設計一些較常使用的組合電路，如：數學運算常用到的加法器 (adders)、減法器 (substractors)，或是比較器 (comparators)、解碼器 (decoders)、編碼器 (encoders) 和多工器 (multiplexers) 等。這些常見的組合電路，在現今的數位 IC cellbased設計中，通常是事先被設計好並當成一個個標準元件來應用的。藉由 EDA 軟體工具 (electronic design automation tool) 的協助，可在電路設計時，指定套用這些「標準元件」。例如：可選用適合自己設計需求的多位元「加法器」。通常在這些標準元件庫裡，會有各種不同特性的元件可供選擇，例如加法器，我們就有至少兩種選擇：可指定一種稱為 ripple carryadder (RCA) 的加法器電路，或是另一種稱為 carry look-ahead adder(CLA) 的加法器電路。前者會使用較小的面積但速度較慢，而後者則會有較快的速度但成本較高。使用這些組合電路時，若能了解其優缺點並依需要選擇，可使設計出來的電路在速度與面積上更符合需求。

5.3 加法器和減法器

在本節，我們將介紹執行算數運算加法和減法的組合電路。我們把實現加法運算的組合電路稱為加法器，而實現減法運算的組合電路則稱之為減法器。

首先，我們觀察一個位元的加法運算。兩個一位元的加數與被加數，相加後有四種可能：0 + 0 = 0、0 + 1 = 1、1 + 0 = 1、1 + 1 = 10。

前三種相加可能產生的結果用一位元表示即可；但是第四種相加，需要用兩個位元來表示。此兩位元的較高位元，通常稱為進位位元 (carry bit)；而較低位元，則稱為兩數的總和位元 (sum bit)。從上面分析可得知，實現一位元加法器時，一定要慮到進位輸出位元，故一位元加法器電路應至少包含有二個一位元的輸入和二個一位元的輸出 (一位元進位輸出與一位元總和值)。

相對地，若是一位元的減法運算，則要考慮到下述的四種相減可能：0 − 0 = 0、0 − 1 = 11、1 − 0 = 1、1 − 1 = 0。第二種相減的結果會產生負值，所以需要用兩個位元來表示，其中較高位元表示借位輸出，較低位元表示兩值相減的結果，其餘三種相減情況則用一個位元表示即可。因為考慮借位輸出位元，故實現一位元減法器電路時，也應包含有二個一位元的輸入和二個一位元的輸出 (一位元借位輸出與一位元相減結果值)。

接下來，我們將分別介紹半加法器、全加法器、漣波進位加法器、進位前看加法器、半減法器、全減法器等電路。

5.3.1 半加法器

加法器顧名思義就是可以針對輸入訊號做加法運算的電路。一個一位元加法器，如果輸入的加數、被加數都是一位元，輸出包含一位元進位與一位元總和值，則此「一位元加法器」又稱為「半加法器」(half adder, H.A.)。
半加法器的行為可以描述如下：

當輸入的加數與被加數同時為 “1” 時，則輸出的和為 “0” 且進位輸出為 “1”。

當輸入的加數與被加數同時為 “0” 時，則輸出的和為 “0” 且進位輸出為 “0”。

當輸入的加數與被加數兩者之中只有一個為 “1” 時，則輸出的和為 “1” 且進位輸出為 “0”。

圖 5.4 半加法器方塊圖

輸入		輸出	
A	*B*	*C*	*S*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

圖 5.5 半加法器真值表

如果用 A 及 B 分別代表被加數與加數，C 與 S 代表進位輸出與總和，則半加法器的方塊圖如圖 5.4 所示，其中 A、B、C、S 都是一位元值。根據上述半加法器的行為，我們可寫出半加法器的真值表，如圖 5.5 所示。利用真值表，我們可以寫出半加法器之最簡積項的和布林代數表示式：

$S = A'B + AB' = A \oplus B$

$C = AB$

若使用 or 閘和 and 閘實現，則半加法器的電路圖如圖 5.6 所示。
若使用 xor 閘和 and 閘實現，則如圖 5.7 所示。

$$S = A'B + AB' = A \oplus B$$
$$C = AB$$

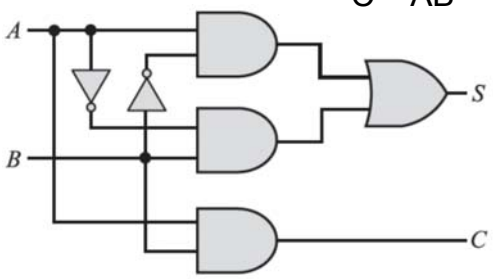


圖 5.6 半加法器電路圖-1

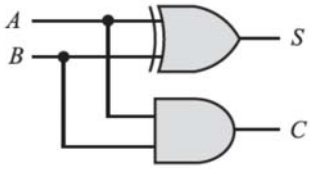


圖 5.7 半加法器電路圖-2

5.3.2 全加法器

前一小節介紹的一位元加法器 (半加法器)，其輸入的加數與被加數都是一位元。但是當輸入的加數與被加數是一個位元以上時，會需要一個多位元的加法器。最常見的多位元加法器有兩種：漣波進位加法器 (ripple carry adder) 和進位前看加法器 (carry look-ahead adder)，我們將分別在 5.3.3 節與 5.3.4 節介紹。

圖 5.8 一位元半加法器的運算

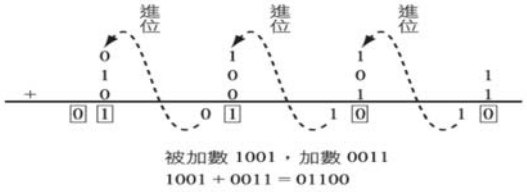


圖 5.9 四位元加法運算

一位元半加法器的運算如圖 5.8 所示。很明顯地，當進行多位元加法運算時，半加法器會不敷使用，因為第 n 位元的輸出進位，會變成第 n + 1 位元的輸入進位值，如圖 5.9 所示。如此一來，在每一個位元我們需要另一種加法器，能將被加數、加數與前一個位元來的進位相加。此時加法運算有八種可能：0 + 0 + 0 = 0、0 + 0 + 1 = 1、0 + 1 +0 = 1、0 + 1 + 1 = 10、1 + 0 + 0 = 1、1 + 0 + 1 = 10、1 + 1 + 0 = 10、1 +1 + 1 = 11。這一種位元加法器，不僅要考慮到進位輸出位元也要考慮到進位輸入位元，通常稱之為「全加法器」，它包含三個一位元的輸入 (被加數、加數與前一個位元來的進位) 和兩個一位元的輸出。

全加法器 (full adder, F.A.) 的行為可以描述如下：

當輸入的加數、被加數和進位輸入三者同時為 “1” 時，則輸出的和為 “1” 且進位輸出為 “1”。

當輸入的加數、被加數和進位輸入三者中有二個同時為 **“1”** 時，則輸出的和為 **“0”** 且進位輸出為 **“1”**。

當輸入的加數、被加數和進位輸入三者中只有一個為 **“1”** 時，則輸出的和為 **“1”** 且進位輸出為 **“0”**。

當輸入的加數、被加數和進位輸入三者同時為 **“0”** 時，則輸出的和為 **“0”** 且進位輸出為 **“0”**。

如果使用 *A*、*B*、*C*in 分別代表被加數、加數與進位輸入，*C*out 與 *S* 代表進位輸出與和，則全加法器的方塊圖如圖 5.10 所示，其中 *A*、*B*、*C*in、*C*out、*S* 都是一位元值。

依據上述全加法器的行為，我們可寫出全加法器的真值表，如圖 5.11 所示。

利用真值表可進行卡諾圖化簡，如圖 5.12 所示。

所以，我們可以寫出全加法器最簡之積項的和布林代數表示式：

$S = AB'Cin' + A'B'Cin + ABCin + A'BCin'$

$Cout = AB + ACin + Bcin$

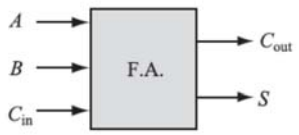


圖 5.10 全加法器方塊圖

輸入			輸出	
A	*B*	C_{in}	C_{out}	*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

圖 5.11 全加法器真值表

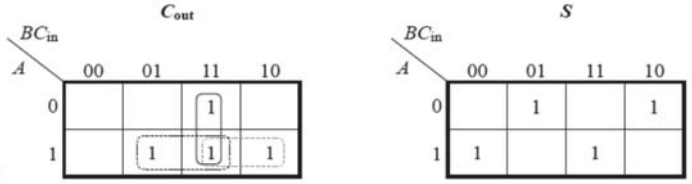


圖 5.12 全加法器卡諾圖

可得到使用 or 閘和 and 閘實現的全加法器電路圖，如圖 5.13 所示。

$S = AB'Cin' + A'B'Cin + ABCin + A'BCin'$
$Cout = AB + ACin + Bcin$

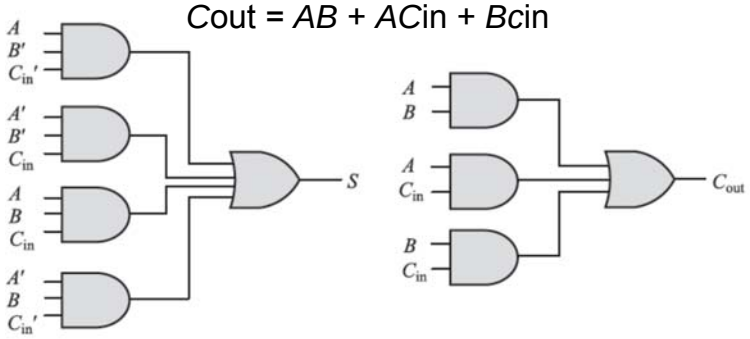


圖 5.13 全加法器電路圖

我們也可以推導出以 xor 實作全加法器的結果：
$S = AB'Cin' + A'B'Cin + ABCin + A'BCin'$
$= Cin'(AB' + A'B) + Cin(A'B' + AB)$
$= Cin'(A \oplus B) + Cin(A \oplus B)'$
$= Cin \oplus (A \oplus B)$
$Cout = AB'Cin + A'BCin + ABCin + ABCin'$
$= Cin(AB' + A'B) + AB(Cin + Cin')$
$= Cin(A \oplus B) + AB$
可使用 xor 閘、and 閘和 or 閘實現全加法器，如圖 5.14 所示。

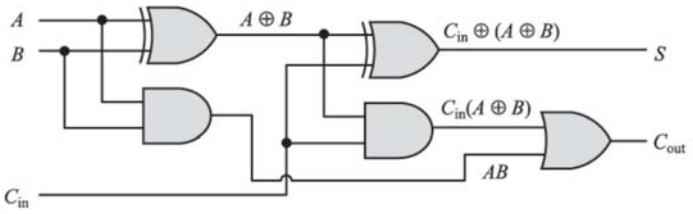


圖 5.14 全加法器電路圖

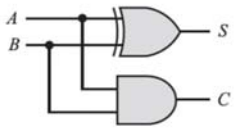


圖 5.7 半加法器電路圖-2

比較全加法器 (圖 5.14) 與半加法器 (5.3.1 節圖 5.7) 的電路圖，可以知道，如果半加法器已經設計實現完成，則全加法器可以使用兩個半加法器與一個 or 閘連接在一起來快速實現，如圖 5.15 所示。

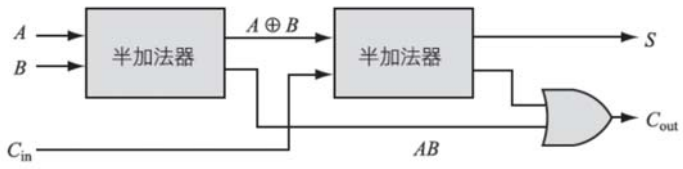
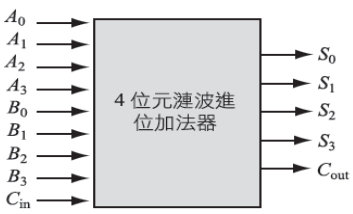


圖 5.15　全加法器電路圖

5.3.3 漣波進位加法器

要執行 n 位元的加法運算，我們可以串聯 n 個全加法器來實作出「漣波進位傳遞加法器」(ripple carry adder, R.C.A.)──這是實現多位元加法器最簡單的方法。當輸入 n 位元的被加數與 n 位元的加數，以及一個最低位元進位輸入時，漣波進位傳遞加法器會輸出 n 位元的數值和與最高位元的進位輸出。

圖 5.16 是 4 位元的漣波進位傳遞加法器的方塊圖，其中輸入的被加數和加數，分別使用 A0～A3 和 B0～B3 表示，最低位元進位輸入用Cin 表示，輸出數值和用 S0～S3 表示，最高位元進位輸出用 Cout 表示。4 位元加法器是一個組合電路，如果我們直接使用前面提及的組合電路設計六個步驟來設計此加法器，會造成很大的困擾。為什麼呢？因為 4 位元加法器的輸入變數共有 9 個 (4 個被加數、4 個加數、1個最低位元進位)，這代表我們需建構一個 $2^9 = 512$ 列的真值表，而且需進行 9 個變數的卡諾圖化簡。



	C_3	C_2	C_1	C_{in}
	A_3	A_2	A_1	A_0
+	B_3	B_2	B_1	B_0
C_{out}	S_3	S_2	S_1	S_0

圖 5.16　4 位元漣波進位傳遞加法器的方塊圖

事實上，5 個變數以上的化簡已經很難用手動製圖來進行，更不要說是 9 個變數。因此，對多位元加法器直接進行卡諾圖化簡實作是有困難的，最好的方式乃是利用前一小節所設計的全加法器來組合連結以得出多位元加法器電路。漣波進位傳遞加法器的概念主要是依循一般傳統二進位的加法運算，把每個位元相加後產生的進位輸出 (如圖 5.16 標示的 *C*3、*C*2、*C*1)，傳遞到下一個位元變成下一個位元的進位輸入，故漣波進位加法器可以利用串聯全加法器來實現之。觀察圖 5.16，第一個位元的進位輸入 *C*in (或表示為 *C*0) 的值為 0，而 *A*0、*B*0 的值不固定 (依真實輸入而定)，把第一個位元的三個輸入值相加後，會產生 *S*0 與 *C*1 這兩個輸出，*S*0 是第一個位元的總和值而 *C*1 為相加後的進位輸出，此進位輸出*C*1 會成為第二個位元的進位輸入。第二個位元的 *A*1、*B*1、*C*1 相加後會產生 *S*1 與 *C*2，其中 *S*1 為第二個位元的總和，*C*2 為第二個位元的進位輸出，也就是第三個位元的進位輸入，其餘以此類推。很明顯地，我們可以看到進位值就像是漣波 (ripple) 般的由上一個全加法器傳遞到下一個全加法器，愈高位元的進位輸入需要等待愈久的傳遞時間才能得出正確值，故這種加法器被稱為漣波進位 (ripple carry) 傳遞加法器。

圖 5.17 為 4 位元漣波進位傳遞加法器的內部結構。當要進行加法運算時，首先要輸入被加數 (*A*3*A*2*A*1*A*0) 與加數 (*B*3*B*2*B*1*B*0) 的值，此時最低位元進位 *C*0 為 0。當輸入值順利進入加法器後，因為第 2、3、4 位元全加法器的進位輸入 (*C*3*C*2*C*1) 都還不是正確值，所以此時這三個加法器所計算出來的位元總和 (*S*3*S*2*S*1) 與進位值 (*C*out*C*3*C*2) 都是錯誤的。唯一例外是第一位元全加法器，因為它的進位輸入值 *C*0 是正確的，因此 *S*0 與 *C*1 在延遲一小段時間後會先被正確計算出來 (延遲時間為一個全加法器內部的閘傳遞延遲時間)。等到 *C*1 被正確計算得出後，此時第二位元全加法器的三個輸入值 (*A*1、*B*1、*C*1) 都已正確，因此再延遲一小段時間後，可得到正確的 *S*1 與 *C*2。當 *C*2 被正確計算得出後，此時第三位元全加法器的三個輸入值 (*A*2、*B*2、*C*2) 都已正確，因此又再延遲一小段時間後，可得到正確的 *S*2 與 *C*3。以此類推，愈高位元的全加法器需等待愈久的延遲時間才能得出正確結果，而這個延遲時間是來自於之前的每一級進位計算所需的閘傳遞延遲時間。

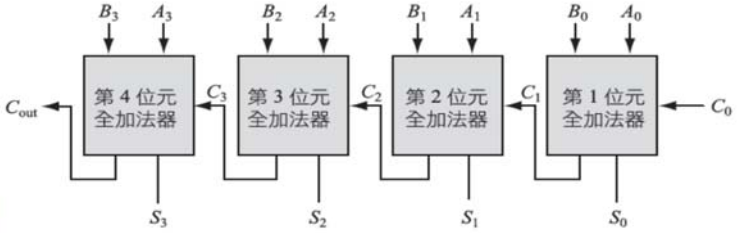


圖 5.17　4 位元漣波進位傳遞加法器內部結構圖

4 位元的漣波進位加法器需要經過四級進位延遲時間才能計算得出正確輸出，而 8 位元的漣波進位加法器則需要經過八級的進位延遲時間才能得出正確輸出。當位元數愈高，這些延遲時間可能會過長而無法接受。遇到這種情況，可以使用下一小節將介紹的進位前看加法器。進位前看的成本雖然較高，但延遲時間遠小於漣波進位加法器。

5.3.4 進位前看加法器

為了解釋進位前看的原理，我們使用 4 位元的進位前看加法器 (carry look-ahead adder, C.L.A.) 為例子來說明。首先，我們假設進位前看加法器的輸出入變數是使用和 4 位元漣波進位傳遞加法器(圖 5.16) 一樣的定義，輸入的被加數和加數分別使用 A0～A3 和 B0～B3表示，最低位元進位輸入用 Cin 表示，輸出數值和用 S0～S3 表示，最高位元進位輸出則用 Cout 表示。進位前看加法器的主要目的是讓「進位傳遞延遲時間儘量不隨著加法器位元數目的增多而增加」，換句話說，就是希望「較高位元進位輸入所需的延遲時間和較低位元進位輸入所需的延遲時間愈接近愈好」。為達到這個目標，我們可觀察圖 5.18。圖 5.18 是第 i 位元的全加法器的內部電路圖，在這個圖中我們特別把會導致進位傳遞延遲的路徑 (path) 以陰影箭號標示出來 (從 Ci 到 Ci + 1)。針對這個位元加法器，我們先定義兩個中間變數 Pi 與 Gi 如圖所示，則我們可寫出下列表示式：

Pi = Ai ⊕ Bi (1)

Gi = AiBi (2)

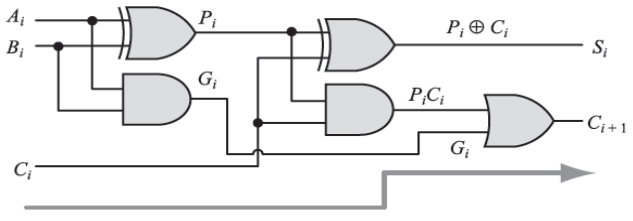


圖 5.18　第 *i* 位元全加法器內部電路圖

如此一來，此第 *i* 位元全加法器之輸入與輸出布林代數表示式為：

$Si = Pi \oplus Ci = (Ai \oplus Bi) \oplus Ci$ (3)

$Ci + 1 = PiCi + Gi = (Ai \oplus Bi)\, Ci + AiBi$ (4)

觀察計算 *Ci* + 1 的 (4) 式，我們發現 *Ci* + 1 和 *Ci* 具有類似遞迴的關係，下一級進位位元要被正確算出，需要前一級進位位元已正確得出，進位如此一級等一級會造成嚴重的進位傳遞延遲。為解決此問題，進位前看加法器乃是展開每個 *Ci* + 1 的值，讓它完全展開到只包含 *C*0 的進位輸入，如下所示：

$C1 = P0C0 + G0$ (5)

$C2 = P1C1 + G1 = P1(P0C0 + G0) + G1 = P1P0C0 + P1G0 + G1$ (6)

$C3 = P2C2 + G2 = P2(P1P0C0 + P1G0 + G1) + G2$

$= P2P1P0C0 + P2P1G0 + P2G1 + G2$ (7)

$C4 = P3C3 + G3 = P3(P2P1P0C0 + P2P1G0 + P2G1 + G2) + G3$

$= P3P2P1P0C0 + P3P2P1G0 + P3P2G1 + P3G2 + G3$ (8)

以此類推，很明顯地，第 *i* 位元的進位輸出 *Ci* + 1 可以展開成只與 *C*0 有關。雖然位元數愈高，展開後的式子愈長、愈複雜，但一定可以展開成只和 *C*0 有關係。因此，進位前看加法器的進位位元不需要一級一級地傳遞過去，故其電路運算速度比漣波進位加法器來得快。

使用上述的 (1)(2)(3) 式，我們知道 *Pi* 的值是 *Ai* 和 *Bi* 的值 xor 的結果，*Gi* 的值是 *Ai* 和 *Bi* 的值 and 的結果，*Si* 的值是 *Ci* 和 *Pi* 的值 xor的結果。因此，可以建構出如圖 5.19 的 4 位元進位前看加法器的架構圖。圖 5.19 中的進位展開電路會分別執行 (5)(6)(7)(8) 式，將左邊輸入的 *C*0、*P*0～*P*3、*G*0～*G*3 值展開計算後得出右邊的 *C*1～*C*4 值。進位展開電路共包含四個模組，如圖 5.20 所示，分別執行 (5)(6)(7)(8) 式。

由於進位前看加法器解決了進位位元的傳遞問題，故其加法運算延遲時間是低於漣波進位加法器的，但從圖 5.19 與 5.20 也可得知，進位前看加法器的電路複雜度 (即電路成本) 會高於漣波進位加法器。

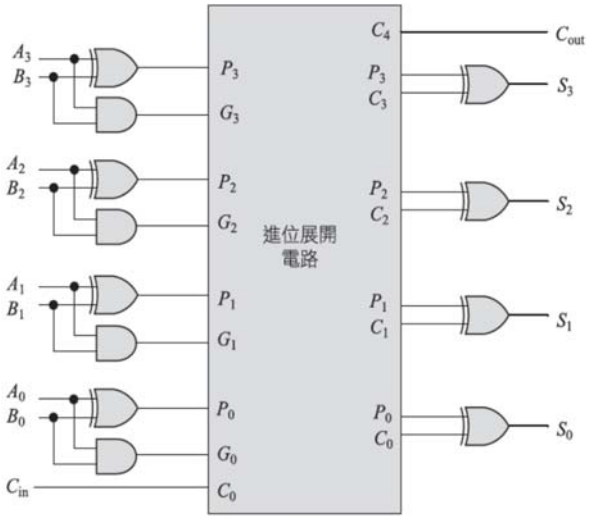


圖 5.19 4 位元進位前看加法器架構圖

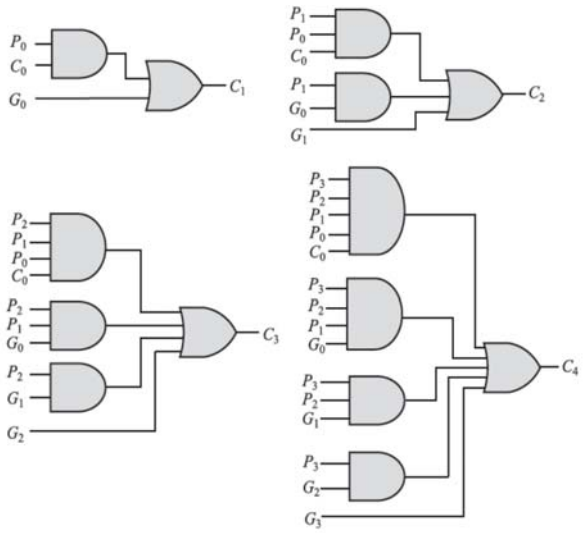


圖 5.20 進位展開電路的四個模組

5.3.5 半減法器

和加法器一樣，針對單位元減法器，我們介紹兩種基本類型：半減法器 **(half subtractor, H.S.)** 和全減法器。兩者最大的不同是全減法器有三個輸入，而半減法器只有兩個輸入，即全減法器比半減法器多了借位輸入。接下來，我們先介紹半減法器。

半加法器是當輸入兩值相加為 “2” 時，輸出進位值為 “1”；半減法器則是當被減數小於減數時，輸出借位值為 “1”，也就是說當借位輸出值為 “1” 時，會向高位元借 “1”，讓目前位元的值因此多了 “2”。其行為描述如下：

當被減數為 **“0”** 且減數也為 **“0”** 時，差值為 **“0”**，借位輸出為 **“0”**。
當被減數為 **“0”** 且減數為 **“1”** 時，差值為 **“1”**，借位輸出為 **“1”**。
當被減數為 **“1”** 且減數為 **“0”** 時，差值為 **“1”**，借位輸出為 **“0”**。
當被減數為 **“1”** 且減數也為 **“1”** 時，差值為 **“0”**，借位輸出為 **“0”**。

針對第二種情況，我們可以理解成當被減數 − 減數為負值時，我們需要設輸出借位為 “1”，而此時可視為 (2 + 被減數) − 減數，因此差值為 “1”。

如果使用 *A* 和 *B* 分別代表被減數與減數，用 *M* 和 *D* 分別代表借位輸出及相減差，則半減法器的方塊圖如圖 5.21 所示，其中每個訊號都為一位元。根據上述半減法器的行為，我們可以寫出半減法器的真值表，如圖 5.22 所示。

利用真值表，我們可以寫出半減法器的最簡積項的和布林代數式：

$D = A'B + AB' = A \oplus B$

$M = A'B$

故半減法器的電路圖如圖 5.23 所示。

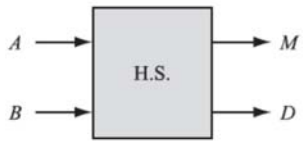


圖 5.21 半減法器方塊圖

輸入		輸出	
A	*B*	*M*	*D*
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

圖 5.22 半減法器真值表

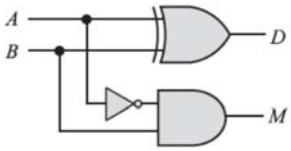


圖 5.23 半減法器電路圖

5.3.6 全減法器

全減法器 **(full subtractor, F.S.)** 的行為可以描述如下：

當被減數為 **“0”**、減數為 **“0”** 時且借位輸入為 **“0”** 時，差值為 **“0”** 且借位輸出為 **“0”**。

當被減數為 **“0”**、減數為 **“0”** 時且借位輸入為 **“1”** 時，差值為 **“1”** 且借位輸出為 **“1”**。

當被減數為 **“0”**、減數為 **“1”** 時且借位輸入為 **“0”** 時，差值為 **“1”** 且借位輸出為 **“1”**。

當被減數為 **“0”**、減數為 **“1”** 時且借位輸入為 **“1”** 時，差值為 **“0”** 且借位輸出為 **“1”**。

當被減數為 **“1”**、減數為 **“0”** 時且借位輸入為 **“0”** 時，差值為 **“1”** 且借位輸出為 **“0”**。

當被減數為 **“1”**、減數為 **“0”** 時且借位輸入為 **“1”** 時，差值為 **“0”** 且借位輸出為 **“0”**。

當被減數為 **“1”**、減數為 **“1”** 時且借位輸入為 **“0”** 時，差值為 **“0”** 且借位輸出為 **“0”**。

當被減數為 **“1”**、減數為 **“1”** 時且借位輸入為 **“1”** 時，差值為 **“1”** 且借位輸出為 **“1”**。

以第二種情況為例，當被減數為0、減數也為0且借位輸入為1時，因為 (被減數 − 減數 − 借位輸入) 的值為負數，我們需要設輸出借位為1，而此時可視為 ((2 + 被減數) − 減數 − 借位輸入)，因此差值為 “1” (2 + 0 − 0 − 1 = 1)。

假設我們用 *A*、*B* 和 *N* 分別代表被減數、減數與借位輸入，用 *M*和 *D* 分別代表借位輸出及相減差，根據以上的行為描述，我們可以寫出「全減法器」的真值表，如圖 5.24 所示。利用真值表可進行卡諾圖化簡，如圖 5.25 所示。

輸入			輸出	
A	*B*	*N*	*M*	*D*
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

圖 5.24　全減法器之真值表

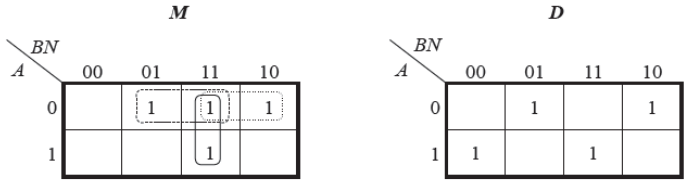


圖 5.25　全減法器之卡諾圖

可寫出全減法器最簡之積項的和代數表示式

$D = AB'N' + A'B'N + ABN + A'BN'$

$M = A'N + BN + A'B$

可得到使用 or 閘和 and 閘實現的全減法器電路圖，如圖 5.26 所示。我們也可以推導出以 xor 實作全減法器的電路如下：

D = AB′N′ + A′B′N + ABN + A′BN′
= (AB′ + A′B)N′ + (A′B′ + AB)N
= (A ⊕ B)N′ + (A ⊕ B)′N
= (A ⊕ B) ⊕ N
= A ⊕ B ⊕ N
M = A′B′N + A′BN + ABN + A′BN′
= A′(B′N + BN′) + (A′ + A)BN
= A′(B ⊕ N) + BN

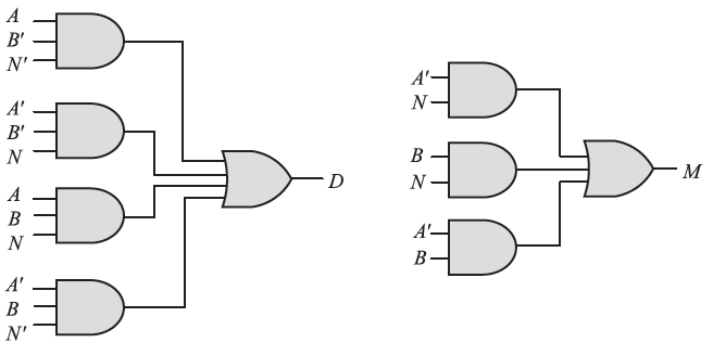


圖 5.26　全減法器電路圖

可使用 xor 閘、and 閘和 or 閘實現全減法器，如圖 5.27 所示。

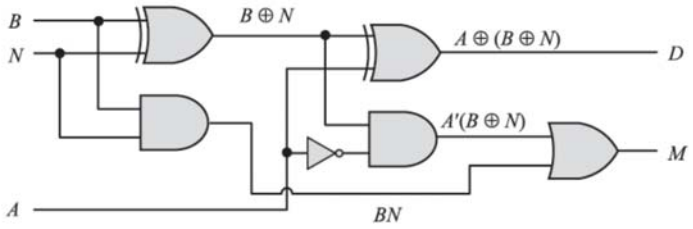


圖 5.27　全減法器電路圖

使用二進制加法運算來完成。例如：A − B = A + (−B)，執行方式是將減數轉換成其相對應的 2 補數，然後將被減數加上減數的 2 補數即可得出答案。如此一來，我們很容易就可以使用 n 個全加法器來設計一個 n 位元加減法器。假設輸入為 A 與 B，當執行算術加法運算 A + B 時，將 A 加上 B 得解；當執行算術減法運算 A − B 時，將 A 加上 B 的 2 補數得解。

圖 5.28 是一個四位元加減法器的電路，其中 *A* 代表被加數或被減數，*B* 代表加數或減數。電路中，四個全加法器以漣波進位的方式串接一起，*B* 則先經過一個 xor 閘才進入全加法器。透過控制信號 control 的控制，此電路可以順利執行 *A* 加上 *B* 或是 *A* 加上 *B* 的 2 補數，詳細過程說明於後。

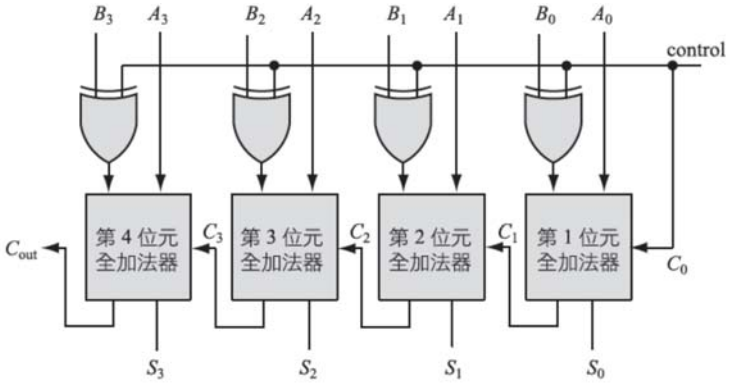


圖 5.28　四位元加減法器電路

兩輸入的 xor 閘可當作數位電路中的「可控制開關」，如圖 5.29 所示，假設 *C* 為控制訊號，則 *C* 的值可決定輸入訊號 *I* 與輸出訊號 *Y* 之間的關係。當 *C* 固定為 0 時，若 *I* = 0，*Y* = 0；若 *I* = 1，*Y* = 1，故 *C* 為 0 時，*Y* = *I*。當 *C* 固定為 1 時，若 *I* = 0，*Y* = 1；若 *I* = 1，*Y* = 0，故 *C* 為 1 時，*Y* = *I*′。

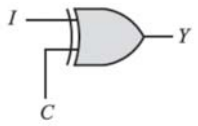


圖 5.29　xor 閘可控制開關

利用上述 xor 閘的特質，觀察圖 5.28，把 control 當成 xor 閘的控制訊號，我們可知：當 control = 0 時，輸入 4 個全加法器的值是 *A* 及 *B*，且最低位元進位輸入 *C*0 為 0，因此 4 個全加法器會計算 *A* + *B* + 0 = *A* + *B*。當 control = 1 時，輸入 4 個全加法器的值則是 *A* 及 *B*′，且最低位元進位輸入 *C*0 為 1，因此 4 個全加法器會計算 *A* + *B*′ + 1 (因為 *B*′ 是 *B* 的 1 補數，再加上 1 可得 *B* 的 2 補數，故 *A* + (*B*′ + 1) = *A* + (*B* 的 2 補數)，就等於計算 *A* − *B*。因此，這個電路藉由 control 訊號的設定，可執行加法或者是減法動作。

5.4 比較器

在組合邏輯電路設計中，有時需要使用比較器 **(comparator)** 來決定兩個數值的大小關係。兩個輸入數值 *A* 和 *B* 之間的大小關係有三種可能：*A* 大於 *B* (greater than)、*A* 等於 *B* (equal to) 或 *A* 小於 *B* (less than) 三種。

假設我們使用 *gt*、*lt* 和 *eq* 分別代表大於輸出值、小於輸出值與相等輸出值。如此一來，比較器的行為可以描述如下：

如果 ***A*** 大於 ***B*** 時，大於輸出值 ***gt*** 為 **1**，否則 ***gt*** 為 **0**。
如果 ***A*** 小於 ***B*** 時，小於輸出值 ***lt*** 為 **1**，否則 ***lt*** 為 **0**。
如果 ***A*** 等於 ***B*** 時，相等輸出值 ***eq*** 為 **1**，否則 ***eq*** 為 **0**。

圖 5.30 是四位元比較器的方塊圖，其中要比較的兩個輸入值分別用 *A*0～*A*3 和 *B*0～*B*3 表示，而 *gt*、*lt* 和 *eq* 則分別代表大於輸出值、小於輸出值與相等輸出值。

如果我們直接使用前面提及的組合電路設計六個步驟來設計比較器電路，會造成很大的困擾。為什麼呢？因為四位元比較器的輸入變數共有 8 個 (*A*0～*A*3 和 *B*0～*B*3)，這表示我們需畫出一個 $2^8 = 256$ 列的真值表，且需進行 8 個變數的卡諾圖化簡，很顯然地，這很難以人工手動方式來進行。換言之，對多位元比較器直接進行卡諾圖化簡實作是有困難的，最好的方式乃是利用一些特殊的技術，或是考量比較器本身的特質，為其量身打造一個專屬電路較為實際。

為設計一個合適的比較器，我們分別考慮相等、大於與小於三種情形，藉由思考比較器功能的特殊性，在不使用卡諾圖的情況下設計該電路。

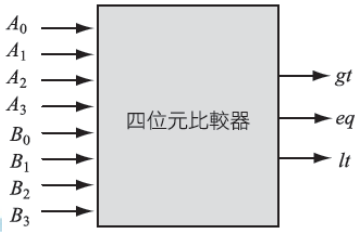


圖 5.30　四位元比較器方塊圖

相等

我們該如何實作比較器電路呢？可以使用一個簡易的方式，若兩值從高位元到低位元的每一個位元都是相同的，我們就可以說這兩個數值相等。而要比較某個位元是否相等，最簡單的想法就是兩輸入位元都「同時為 1」或是「同時為 0」。因此我們可以將位元相等的概念以下列的布林函數表示：

$Xi = AiBi + Ai'Bi'$ $i = 0, 1, 2, 3$

其中，*Ai* 和 *Bi* 分別表示 *A* 和 *B* 的第 *i* 個位元。觀察上式，當 *Ai* 和 *Bi* 同時為 1 時，*AiBi* 這一項為 1，故 $Xi = 1$；當 *Ai* 和 *Bi* 同時為 0 時，*Ai'Bi'* 這一項為 1，故 $Xi = 1$。因此我們知道，當 *A* 和 *B* 的第 *i* 個位元相等時，*Xi* 值為 1，只要 *A* 和 *B* 的第 *i* 個位元不相等，*Xi* 值為 0。

有了位元相等的布林函數，我們可以使用下列的布林函數 *eq* 來表示四位元的 *A* 及 *B* 值是完全相等的：

$eq = X0\ X1\ X2\ X3$

$= (A0B0 + A0'B0')(A1B1 + A1'B1')(A2B2 + A2'B2')(A3B3 + A3'B3')$

當 $X0 = 1$、$X1 = 1$、$X2 = 1$ 且 $X3 = 1$ 時，四個位元的相等同時成立，故 *A* 等於 *B* 成立 (因為 *X*0 *X*1 *X*2 *X*3 是 and 在一起)。上面式子的每個括號，分別表示從最低到最高位元是否相等，當這四個括號結果同時成立，則 *eq* 的值為 1，表示 *A* 值和 *B* 值相等，否則為 0，表示 *A* 值和 *B* 值不相等。用 and 閘和 or 閘實作電路如圖 5.31 所示。

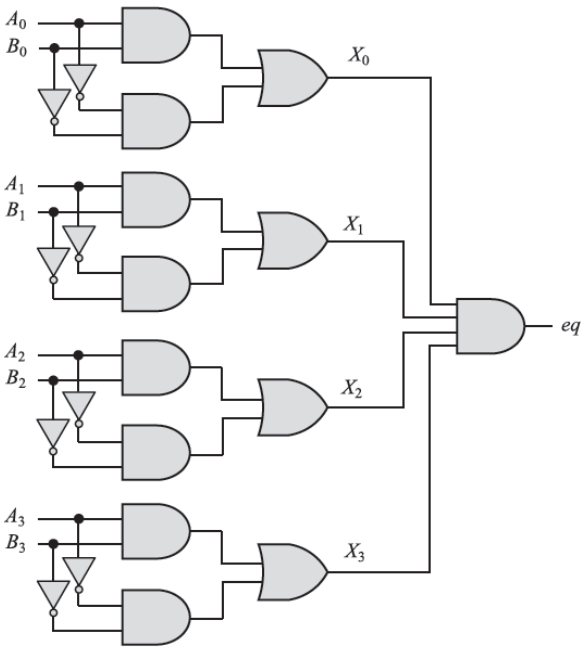


圖 5.31　比較器相等部分的電路圖

大於

應用前述比較位元相等的作法，我們可以設計「大於」的電路。
要判斷 *A* 是否大於 *B*，我們觀察其行為如下：
　當 ***A3*** 為 **1** 且 ***B3*** 為 **0**，則表示 ***A > B***。
　當 ***A3*** 和 ***B3*** 相等、***A2*** 為 **1** 且 ***B2*** 為 **0** 同時成立時，則表示 ***A > B***。
　當 A3 和 B3 相等、A2 和 B2 相等、A1 為 1 且 B1 為 0 同時成立時，則表示 A > B。
　當 A3 和 B3 相等、A2 和 B2 相等、A1 和 B1 相等、A0 為 1 且 B0 為 0 同時成立時，則表示 A > B。
上述四個情況中的任一個成立，表示 A > B，否則表示 A > B 不成立。
要決定 A 是否大於 B，我們從最高位元開始檢查。當 A3 = 1 且 B3 = 0，我們知道 A 大於 B。當 A3 與 B3 相等 (即 X3 = 1 時)，A2 = 1 且 B2 = 0，則 A 大於 B，其餘依此類推。因此，要決定 A 是否大於 B，可以使用下列的布林函數 gt 來表示

$$gt = A3B3' + X3A2B2' + X3X2A1B1' + X3X2X1A0B0'$$

上式的每一項代表每一個 *A* > *B* 成立的情況，四個情況中的任一個成立 (因為 or 在一起)，表示 *A* > *B*，否則表示 *A* > *B* 不成立。
可用邏輯閘實作如圖 5.32 所示，其中 *X*1、*X*2 與 *X*3 的實現電路如圖 5.31 所示。

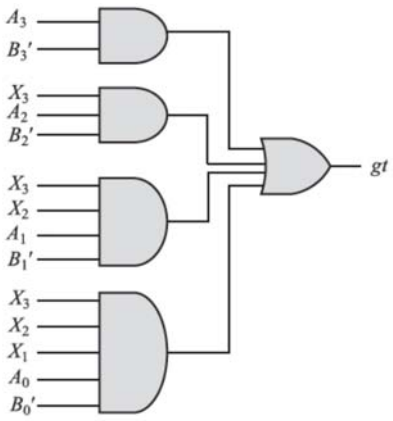


圖 5.32　比較器大於部分的電路圖

小於

同理，應用前述比較位元相等的作法，我們可以設計「小於」的電路。要判斷 *A* 是否小於 *B*，我們觀察其行為如下：

當 ***A*3** 為 **0** 且 ***B*3** 為 **1**，則表示 ***A* < *B***。

當 ***A*3** 和 ***B*3** 相等、***A*2** 為 **0** 且 ***B*2** 為 **1** 同時成立時，則表示 ***A* < *B***。

當 ***A*3** 和 ***B*3** 相等、***A*2** 和 ***B*2** 相等、***A*1** 為 **0** 且 ***B*1** 為 **1** 同時成立時，則表示 ***A* < *B***。

當 ***A*3** 和 ***B*3** 相等、***A*2** 和 ***B*2** 相等、***A*1** 和 ***B*1** 相等、***A*0** 為 **0** 且 ***B*0** 為 **1** 同時成立時，則表示 ***A* < *B***。

上述四個情況中的任一個成立，表示 ***A* < *B***，否則表示 ***A* < *B*** 不成立。

要決定 *A* 是否小於 *B*，我們從最高位元開始檢查。當 *A*3 = 0 且 *B*3 = 1，我們知道 *A* 小於 *B*。當 *A*3 與 *B*3 相等 (即 *X*3 = 1 時)，*A*2 = 0 且 *B*2 = 1，則 *A* 小於 *B*，其餘依此類推。因此，要決定 *A* 是否小於 *B*，可以使用以下的布林函數 *lt* 來表示

$$lt = A3'B3 + X3A2'B2 + X3X2A1'B1 + X3X2X1A0'B0$$

上式的每一項代表每一個 *A* < *B* 成立的情況，四個情況中的任一個成立 (因為 or 在一起)，表示 *A* < *B*，其他情況則表示 *A* < *B* 不成立。可用邏輯閘實作如圖 5.33，其中 *X*1、*X*2 與 *X*3 的實現電路如圖 5.31 所示。

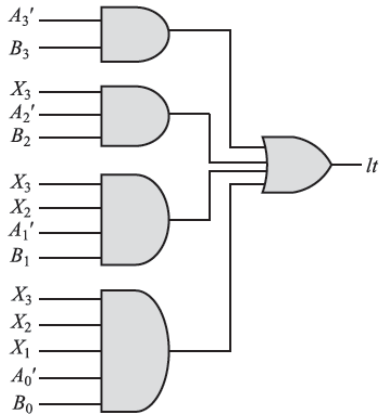


圖 5.33　比較器小於部分的電路圖

若將比較器大於、小於、相等的三個電路整合一起，由於有一些可重複共用的部分，因此電路會比直接將圖 **5.31**、**5.32**、**5.33** 三個電路相加來得精簡，整合後的比較器電路圖如圖 **5.34** 所示。

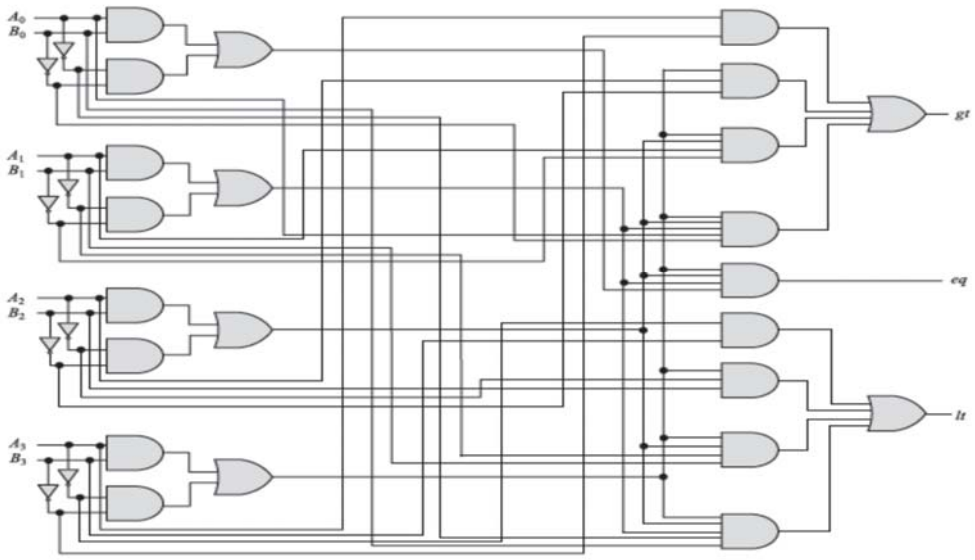


圖 5.34 完整比較器電路圖

5.5 多工器

多工器 **(multiplexer)** 是一個組合電路，能讓我們從眾多輸入訊號線中選擇要將哪一條輸入訊號線資料送到輸出端，故有時候也稱為資料選擇器 **(data selector)**。例如：有一多工器具備兩個輸入、一個輸出 (又稱為 2 對 1 多工器)，藉由一位元的選擇線，可以決定哪一個輸入訊號值會傳送到輸出端。一般來說，當有 2^n 條輸入訊號線時，需要使用至少 n 條選擇線以決定將哪一條輸入訊號線的資料傳送到輸出訊號端。以下我們將分別介紹一位元的 2 對 1 多工器、4 對 1 多工器和一個多位元 2 對 1 多工器。

2 對 1 多工器

假設使用 *A* 和 *B* 代表多工器的輸入訊號線，用 *S* 表示多工器的選擇訊號線，*O* 表示多工器的輸出訊號線，則 **2** 對 **1** 多工器 **(2-to-1 multiplexer)** 的方塊圖如圖 5.35 所示，通常為一個梯型圖示並在方塊內標示 MUX。

2-to-1 多工器行為描述如下：

當輸入選擇線 ***S*** 為 **0** 時，輸出 ***A*** 訊號線的資料 **(**即 ***O* = *A*)**。

當輸入選擇線 ***S*** 為 **1** 時，輸出 ***B*** 訊號線的資料 **(**即 ***O* = *B*)**。

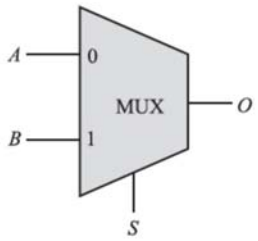


圖 5.35　2-to-1 多工器方塊圖

換句話說，當 *S* 為 0 時，輸出訊號 *O* 的值等於輸入訊號 *A* 的值；當 *S* 為 1 時，輸出訊號 *O* 的值等於輸入訊號 *B* 的值。詳細的真值表如下所示：

S	*A*	*B*	*O*
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

為了方便檢視多工器，我們通常用下列的功能表來描述多工器而不使用完整的真值表：

S	*O*
0	*A*
1	*B*

這個功能表中呈現，當 S 為 0 時，輸出 O 等於輸入訊號 A 的值；當 S 為 1 時，輸出 O 等於輸入訊號 B 的值。換句話說，當選擇線 S 為 0 時，A 訊號線的資料是可以「通過」(pass) 多工器並且輸出的，但 B 訊號線的資料卻是被「阻隔」(block) 的；當選擇線 S 為 1 時，B 訊號線的資料是可以「通過」多工器並且輸出的，但 A 訊號線的資料卻是被「阻隔」的。

在 3.2.1 節 and 閘的介紹中曾提及，and 閘可當作數位電路中的「可控制開關」(見圖 3.6)。將兩輸入 and 閘其中的一個輸入端連接至控制訊號，則控制訊號的值可決定另一個輸入端的訊號是否能傳送到 and 閘的輸出端。利用上述 and 閘的控制特性，我們可以設計出 2-to-1 多工器的電路，如圖 5.36 所示。

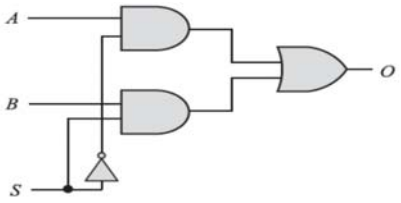


圖 5.36　2-to-1 多工器電路圖

根據圖 5.36，當選擇訊號 *S* 為 0 時，上面 and 閘的控制端輸入值為 1，因此輸入 *A* 訊號可以傳送通過 and 閘到達最後級的 or 閘，而此時下面 and 閘的控制端輸入值為 0，輸入 *B* 訊號無法傳送通過 and 閘，因此最後 or 閘的輸出乃是由上面的 and 閘的輸出 (即是 *A*) 所決定，故 *O* = *A*。

同理，當選擇訊號 *S* 為 1 時，上面 and 閘的控制端輸入值為 0，因此輸入 *A* 訊號無法傳送通過 and 閘，而此時下面 and 閘的控制端輸入值為 1，輸入 *B* 訊號可以傳送通過 and 閘到達最後級的 or 閘，因此最後 or 閘的輸出乃是由下面的 and 閘的輸出 (即是 *B*) 所決定，故 *O* = *B*。

4 對 1 多工器

假設使用 *A*、*B*、*C* 和 *D* 代表多工器的輸入訊號線，用 *S*1 與 *S*0 表示多工器的兩條選擇訊號線，*O* 表示多工器的輸出訊號線，則 **4** 對 **1** 多工器 **(4-to-1 multiplexer)** 的方塊圖就如圖 5.37 所示。

4-to-1 多工器行為描述如下：

當輸入選擇線 ***S1*** 為 **0** 且 ***S0*** 為 **0** 時，輸出 ***A*** 訊號線的資料 (即 ***O =A***)。
當輸入選擇線 ***S1*** 為 **0** 且 ***S0*** 為 **1** 時，輸出 ***B*** 訊號線的資料 (即 ***O =B***)。
當輸入選擇線 ***S1*** 為 **1** 且 ***S0*** 為 **0** 時，輸出 ***C*** 訊號線的資料 (即 ***O =C***)。
當輸入選擇線 ***S1*** 為 **1** 且 ***S0*** 為 **1** 時，輸出 ***D*** 訊號線的資料 (即 ***O =D***)。

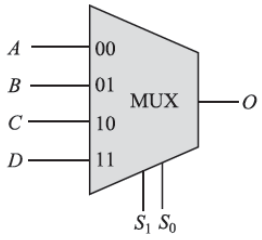


圖 5.37　4-to-1 多工器方塊圖

換句話說，我們可以得出下面的 4-to-1 多工器的功能表：

S_1	S_0	O
0	0	A
0	1	B
1	0	C
1	1	D

藉由 *S*1 與 *S*0 兩條選擇線的控制，我們可以從四個輸入訊號中挑選一個並將其輸出。同樣地利用 and 閘的開關控制特性，我們可以設計出 4-to-1 多工器的電路，如圖 5.38。

觀察圖 5.38，四個輸入 *A*、*B*、*C*、*D* 分別為圖中四個 and 閘之三輸入中的一個，選擇訊號線 *S*1*S*0 則經過反相器而產生四種組合，分別為：*S*1′*S*0′、*S*1′*S*0、*S*1*S*0′、*S*1*S*0，這四組選擇控制分別接到四個 and 閘之三輸入中的另兩個輸入。透過輸入不同的選擇訊號，電路可以確保每次只有一個 and 閘是允許資料「通過」的，其他的三個 and 閘的資料則是被「阻隔」的。當選擇訊號線 *S*1*S*0 為 00 時，最上方的 and 閘之資料是可「通過」的，其餘 and 閘的資料則是被「阻隔」的，故輸入訊號 *A* 會透過最後級的 or 閘傳到輸出 *O*。當選擇訊號線 *S*1*S*0 為 01 時，第二個 and 閘的資料是可「通過」的，其餘 and 閘的資料則是被「阻隔」的，故輸入訊號 *B* 會傳到輸出 *O*。當選擇訊號線 *S*1*S*0 為 10 時，第三個 and 閘的資料是可「通過」的，其餘 and 閘的資料則是被「阻隔」的，故輸入訊號 *C* 會傳到輸出 *O*。當選擇訊號線 *S*1*S*0 為 11 時，最下面 and 閘的資料是可「通過」的，其餘 and 閘的資料則是被「阻隔」的，故輸入訊號 *D* 會傳到輸出 *O*。

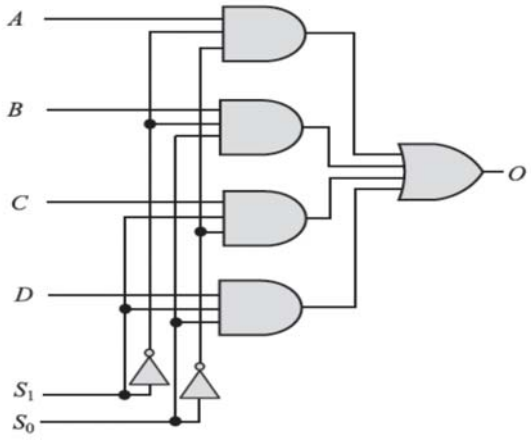


圖 5.38　4-to-1 多工器電路圖

2 對 1 致能多工器

假設使用 *A* 和 *B* 代表多工器的輸入訊號線，用 *S* 表示多工器的選擇訊號線，*en* 代表致能輸入訊號，*O* 表示多工器的輸出訊號線，則 **2** 對 **1** 致能多工器 **(2-to-1 multiplexer with enable signal)** 的方塊圖就如圖 5.39 所示。

2-to-1 致能多工器行為描述如下：

當致能輸入 ***en*** 為 **0** 時，不論選擇線 ***S*** 為何，輸出值為 **0** (即 ***O* = 0**)。

當致能輸入 ***en*** 為 **1** 時，輸入選擇線 ***S*** 為 **0** 時，輸出 ***A*** 訊號線的資料 (即 ***O* = *A***)。

當致能輸入 ***en*** 為 **1** 時，輸入選擇線 ***S*** 為 **1** 時，輸出 ***B*** 訊號線的資料 (即 ***O* = *B***)。

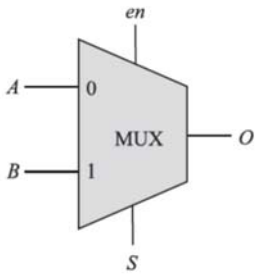


圖 5.39 2-to-1 致能多工器方塊圖

換句話說，當致能訊號 *en* 為 0 時，多工器不動作，輸出恆為 0。當致能訊號 *en* 為 1 時，多工器會啟動，若 *S* 為 0 時，輸出訊號 *O* 的值等於輸入訊號 *A* 的值；當 *S* 為 1 時，輸出訊號 *O* 的值等於輸入訊號 *B* 的值。功能表如下所示：

en	*S*	*O*
0	*X*	0
1	0	*A*
1	1	*B*

其中 *X* 為 don't care，代表不管 *S* 為 0 或 1 都一樣。當致能輸入 *en* 為 0 時，不管輸入選擇值 *S* 為何，此電路是被禁能 **(disable)** 的。在此禁能的意思是指輸出訊號全部都是 0 與輸入訊號完全無關。當致能輸入 *en* 為 1 時，才致能這個多工器 (啟動多工器) 會從兩條輸入訊號線中選擇一條，將其資料傳送到輸出訊號端。

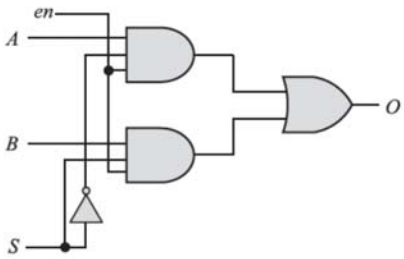


圖 5.40　2-to-1 致能多工器電路圖

在上例中，致能訊號 *en* 為 1 時，啟動多工器，故可稱為正邏輯致能。事實上，也可以使用 0 來代表電路的致能，此時稱為負邏輯致能，要使用正邏輯致能或負邏輯致能則完全視電路的實際需求而定。我們可以設計出 2-to-1 致能多工器的電路，如圖 5.40 所示。當致能訊號 *en* 為 0 時，上下兩個 and 閘都不動作，無法將資料傳到輸出端，輸出為 0。當致能訊號 *en* 為 1，若選擇訊號 *S* 為 0，上面的 and 閘動作 (下面的 and 閘因控制輸入訊號為 0，故不動作)，輸入 *A* 訊號可以傳送到輸出；若選擇訊號 *S* 為 1，下面的 and 閘動作 (上面的 and 閘因控制輸入訊號為 0，故不動作)，輸入 *B* 訊號可以傳送到輸出。

四位元 2 對 1 致能多工器

前面介紹的多工器之輸入訊號都是一位元，接下來我們介紹多位元輸入訊號的情況。假設使用 *A*0～*A*3 和 *B*0～*B*3 表示多工器的兩個四位元輸入訊號線，*S* 表示多工器的選擇訊號線，*en* 表示多工器的致能訊號線，而 *O*0～*O*3 表示多工器的輸出訊號線，則四位元 **2** 對 **1** 致能多工器 **(quadruple 2-to-1 multiplexer)** 方塊圖如圖 5.41 所示。

四位元 2-to-1 致能多工器行為可描述如下：

當致能輸入 ***en*** 為 **0** 時，不論選擇線 ***S*** 為何，輸出線 ***O*0～*O*3** 全為 **0**。

當致能輸入 ***en*** 為 **1** 時，輸入選擇線 ***S*** 為 **0** 時，***O*3*O*2*O*1*O*0 = *A*3*A*2*A*1*A*0**。

當致能輸入 ***en*** 為 **1** 時，輸入選擇線 ***S*** 為 **1** 時，***O*3*O*2*O*1*O*0 = *B*3*B*2*B*1*B*0**。

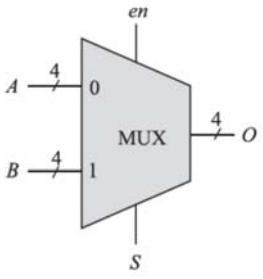


圖 5.41　四位元 2-to-1 致能多工器方塊圖

功能表描述如下：

en	S	$O_3O_2O_1O_0$
0	X	0000
1	0	$A_3A_2A_1A_0$
1	1	$B_3B_2B_1B_0$

我們可以設計出四位元 2-to-1 致能多工器的電路，如圖 5.42 所示。當致能訊號 en 為 0 時，八個 and 閘都不動作，無法將資料傳到輸出端，輸出 4 位元皆為 0。當致能訊號 en 為 1，若選擇訊號 S 為 0，上面的四個 and 閘動作 (下面的四個 and 閘因控制輸入訊號為 0，故不動作)，輸入 A3A2A1A0 訊號可以傳送到輸出；若選擇訊號 S 為 1，下面的四個 and 閘動作 (上面的四個 and 閘因控制輸入訊號為 0，故不動作)，輸入 B3B2B1B0 訊號可以傳送到輸出。

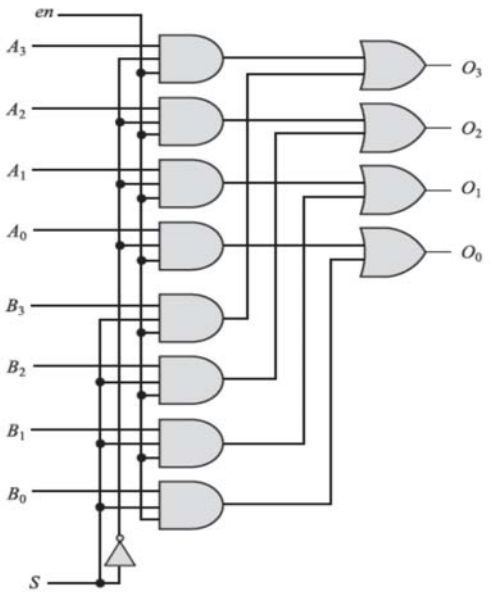


圖 5.42　四位元 2-to-1 致能多工器電路圖

5.6 解多工器

解多工器 **(demultiplexer)** 的功能和多工器剛好相反，它可以將單一輸入訊號傳送到多條輸出線中的一條，換句話說，解多工器用來選擇哪一條輸出訊號線可以接受目前的輸入訊號，因此又稱為資料分配器 (data distributor)。接下來我們介紹 1 對 4 解多工器。

1 對 4 解多工器

假設使用 in 代表輸入訊號，S1 與 S0 代表解多工器的選擇訊號線，A、B、C、D 則分別代表輸出訊號線，則 1 對 4 解多工器 (1-to-4 **demultiplexer)** 的方塊圖如圖 5.43 所示，選擇訊號線的值會決定四條輸出訊號線中的哪一條可以接受目前的輸入訊號。

1-to-4 解多工器行為描述如下：

當輸入選擇線 ***S1*** 為 **0** 且 ***S0*** 為 **0** 時，輸入訊號值會傳送到輸出訊號線 ***A*** **(**即 ***A = in*****)**。

當輸入選擇線 ***S1*** 為 **0** 且 ***S0*** 為 **1** 時，輸入訊號值會傳送到輸出訊號線 ***B*** **(**即 ***B = in*****)**。

當輸入選擇線 ***S1*** 為 **1** 且 ***S0*** 為 **0** 時，輸入訊號值會傳送到輸出訊號線 ***C*** **(**即 ***C = in*****)**。

當輸入選擇線 ***S1*** 為 **1** 且 ***S0*** 為 **1** 時，輸入訊號值會傳送到輸出訊號線 ***D*** **(**即 ***D = in*****)**。

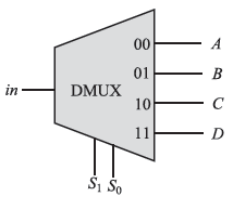


圖 5.43 1-to-4 解多工器方塊圖

換句話說，我們可以得出 1-to-4 解多工器的功能表如下：

S_1	S_0	A	B	C	D
0	0	*in*	0	0	0
0	1	0	*in*	0	0
1	0	0	0	*in*	0
1	1	0	0	0	*in*

藉由 *S*1 與 *S*0 兩條選擇線的控制，我們可以將輸入訊號傳送到四個輸出訊號中的一個。利用 and 閘的開關控制特性，我們可以設計出1-to-4 解多工器的電路，如圖 5.44。輸入 *in* 訊號分別為圖中四個 and 閘之三輸入中的一個，選擇訊號線 *S*1*S*0 則經過反相器而產生四種組合，分別為：*S*1′*S*0′、*S*1′*S*0、*S*1*S*0′、*S*1*S*0，這四組選擇控制分別接到四個 and 閘之三輸入中的另兩個輸入。透過輸入不同的選擇訊號，電路可以確保每次只有一個 and 閘是允許資料「通過」的，其他的三個 and 閘則因為控制訊號包含 0 值，故其資料是被「阻隔」的。

當選擇訊號線 *S*1*S*0 為 00 時，最上方的 and 閘之資料是可「通過」的，其餘 and 閘的資料則是被「阻隔」的，故輸入訊號 *in* 傳到輸出 *A*。當選擇訊號線 *S*1*S*0 為 01 時，第二個 and 閘的資料是可「通過」的，其餘 and 閘的資料則是被「阻隔」的，故輸入訊號 *in* 會傳到輸出 *B*。當選擇訊號線 *S*1*S*0 為 10 時，第三個 and 閘的資料是可「通過」的，其餘 and 閘的資料則是被「阻隔」的，故輸入訊號 *in* 會傳到輸出 *C*。當選擇訊號線 *S*1*S*0 為 11 時，最下面 and 閘的資料是可「通過」的，其餘 and 閘的資料則是被「阻隔」的，故輸入訊號 *in* 會傳到輸出 *D*。

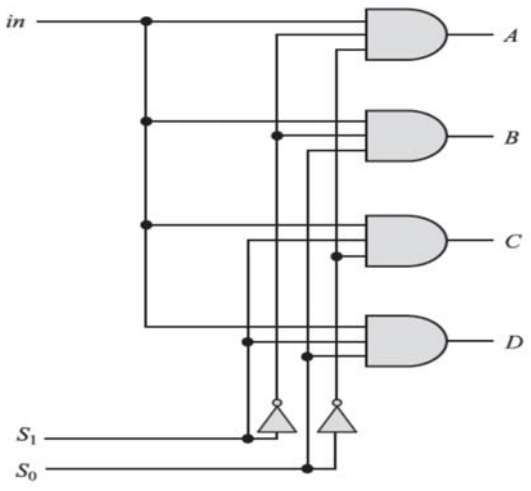


圖 5.44　1-to-4 解多工器電路圖

同理，我們也可以設計一個具致能訊號的解多工器，使用如下的功能表：

en	S_1	S_0	*A*	*B*	*C*	*D*
0	*X*	*X*	0	0	0	0
1	0	0	*in*	0	0	0
1	0	1	0	*in*	0	0
1	1	0	0	0	*in*	0
1	1	1	0	0	0	*in*

當致能訊號 *en* 為 0 時，四個輸出都為 0，無法將輸入資料 *in* 傳到任一個輸出端。當致能訊號 *en* 為 1，透過選擇訊號 *S*1*S*0 的值，可以將輸入資料 *in* 送到所選擇的那條輸出訊號線上，其電路圖如圖 5.45 所示。

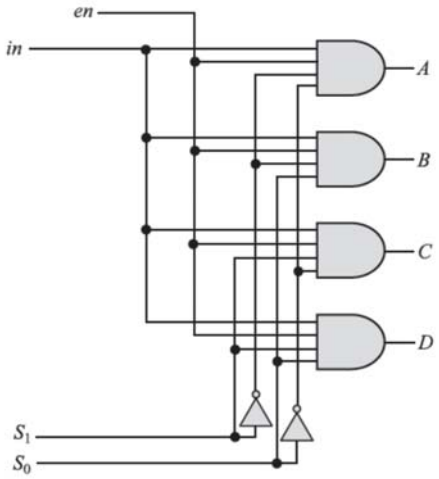


圖 5.45 1-to-4 致能解多工器電路圖

5.7 編碼器

編碼器 **(encoder)** 是一個能對輸入訊號值進行二進制編碼的組合電路。它可以將 2^n 個輸入訊號轉換成 *n* 位元的輸出訊號，假設有 *m* 個輸入與 *n* 個輸出，則稱為 *m* 對 *n* 編碼器。

例如，有一個包含 4 條輸入線、2 條輸出線的 **4** 對 **2** 編碼器 **(4-to-2 encoder)**，則此電路可以將輸入訊號值進行如下的二進制編碼：

當第一條輸入線為 **1**，其他三條輸入線為 **0** 時，輸出編碼值 **00**。
當第二條輸入線為 **1**，其他三條輸入線為 **0** 時，輸出編碼值 **01**。
當第三條輸入線為 **1**，其他三條輸入線為 **0** 時，輸出編碼值 **10**。
當第四條輸入線為 **1**，其他三條輸入線為 **0** 時，輸出編碼值 **11**。

很顯然地，四種組合可以用兩個位元來編碼。此外，我們必須確保一次只有一條輸入線為 **1**，才能正確編碼。接下來，我們將分別介紹，8 對 3 的編碼器與 4 對 2 的優先權編碼器。

8 對 3 編碼器

假設使用 *I*0～*I*7 來分別表示編碼器的八條輸入訊號線，用 *O*2、*O*1、*O*0 分別表示編碼器的三條輸出訊號線，則此 **8** 對 **3** 編碼器 **(8-to-3 encoder)** 的方塊圖如圖 5.46 所示。

其行為描述如下：

當只有 ***I*0** 為 **"1"**，其餘七條輸入線為 **"0"** 時，則輸出值 ***O*2*O*1*O*0** 為 **"000"**。

當只有 ***I*1** 為 **"1"**，其餘七條輸入線為 **"0"** 時，則輸出值 ***O*2*O*1*O*0** 為 **"001"**。

當只有 ***I*2** 為 **"1"**，其餘七條輸入線為 **"0"** 時，則輸出值 ***O*2*O*1*O*0** 為 **"010"**。

當只有 ***I*3** 為 **"1"**，其餘七條輸入線為 **"0"** 時，則輸出值 ***O*2*O*1*O*0** 為 **"011"**。

當只有 ***I*4** 為 **"1"**，其餘七條輸入線為 **"0"** 時，則輸出值 ***O*2*O*1*O*0** 為 **"100"**。

當只有 ***I*5** 為 **"1"**，其餘七條輸入線為 **"0"** 時，則輸出值 ***O*2*O*1*O*0** 為 **"101"**。

當只有 ***I*6** 為 **"1"**，其餘七條輸入線為 **"0"** 時，則輸出值 ***O*2*O*1*O*0** 為 **"110"**。

當只有 ***I*7** 為 **"1"**，其餘七條輸入線為 **"0"** 時，則輸出值 ***O*2*O*1*O*0** 為 **"111"**。

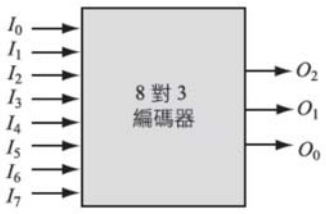


圖 5.46 8-to-3 編碼器的方塊圖

根據 8 對 3 編碼器的行為，我們可以寫出如下的真值表：

輸入								輸出		
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	O_2	O_1	O_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

8 個輸入變數應該會有 256 種輸入排列組合，真值表應該包含 256 列，在這邊為什麼我們只列出其中的 8 列呢？原因是為了讓編碼器能正常編碼，我們需要確保 **8** 個輸入變數 **(**即 **8** 條輸入線**)** 每一次只能有一個為 **1**，其他皆須為 **0**，才能順利編碼，故正常的輸入情況只有上面真值表所列出的八種而已。

觀察真值表，我們知道：

當 ***I*1**、***I*3**、***I*5** 或 ***I*7** 四個輸入中任一個為 **1** 時，***O*0** 為 **1**。

當 ***I*2**、***I*3**、***I*6** 或 ***I*7** 四個輸入中任一個為 **1** 時，***O*1** 為 **1**。

當 ***I*4**、***I*5**、***I*6** 或 ***I*7** 四個輸入中任一個為 **1** 時，***O*2** 為 **1**。

故可以寫出編碼器的布林表示式如下：

$O0 = I1 + I3 + I5 + I7$

$O1 = I2 + I3 + I6 + I7$

$O2 = I4 + I5 + I6 + I7$

可得到 8 對 3 編碼器的電路圖如圖 5.47 所示。
觀察圖 5.47 我們知道，若能確保每次只有一條輸入線為 1，則此電路會正常動作，例如：當 *I*3 為 1，其他輸入線都為 0 時，根據電路會輸出 011；當 *I*6 為 1，其他輸入線都為 0 時，根據電路會輸出 110。
但是若有超過一條的輸入線同時為 1，則輸出會有錯誤，例如： 當 *I*3與 *I*6 同時為 1，其他輸入線都為 0 時，根據電路會輸出 111；當 *I*1 與 *I*4同時為 1，其他輸入線都為 0 時，根據電路會輸出 101，這些都屬於錯誤編碼狀況。
如果輸入在同一時間有超過一條線為 1 時，我們仍希望電路正常動作，則可以使用接下來要介紹的優先權編碼器來實現。優先權編碼器的工作方式如下：當 *I*1 和 *I*4 同時為 1 時，選擇權重較高的線 *I*4 來編碼，故編碼成 100；或是當 *I*3 和 *I*5 同時為 1 時，選擇權重較高的線 *I*5來編碼，故編碼成 101。

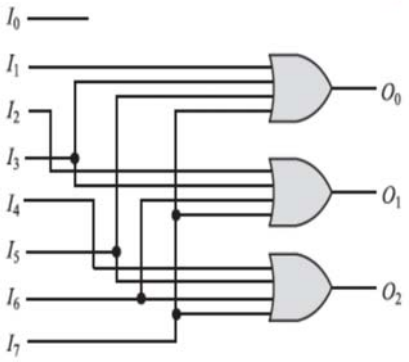


圖 5.47 8-to-3 編碼器的電路圖

優先權編碼器

優先權編碼器 **(priority encoder)** 是一個具有優先權的編碼組合電路，這個「優先權」的意思就是指：當輸入訊號線有兩條以上同時為 1時，我們只會根據權重較高 (即位元較高) 的那條輸入為 1 的線來進行編碼。接下來，我們描述一個 4 輸入 3 輸出的優先權編碼器 (包含：4條輸入線、2 條編碼輸出線、1 條輸出合法標示訊號線) 之行為，要注意的是，下列五個行為描述是具有先後順序而且是互斥的：

1. 當 **4** 條輸入線的值皆為 **0** 時，我們忽略編碼輸出線的值，並且讓輸出合法標示線的值為 **0 (**代表此輸出不合法**)**。
2. 當最高位元輸入訊號線為 **1** 且其他較低位元輸入訊號線為任意值時，則編碼輸出線輸出 **11** 且輸出合法標示線的值為 **1 (**代表此輸出合法**)**。
3. 當次高位元輸入訊號線為 **1** 且其他較低位元輸入訊號線為任意值時，則編碼輸出線輸出 **10** 且輸出合法標示線的值為 **1 (**代表此輸出合法**)**。
4. 當第三高位元輸入訊號線為 **1** 且其他較低位元輸入訊號線為任意值時，則編碼輸出線輸出 **01** 且輸出合法標示線的值為 **1 (**代表此輸出合法**)**。
5. 當最低位元輸入訊號線為 **1** 時，則編碼輸出線輸出 **00** 且輸出合法標示線的值為 **1 (**代表此輸出合法**)**。
如果分別用 I_0～I_3 代表輸入訊號，O_1 代表高位元的輸出編碼訊號，O_0代表低位元的輸出編碼訊號，V 代表輸出訊號的合法性，此優先權編碼器的方塊圖如圖 5.48 所示。

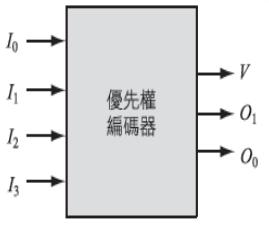


圖 5.48 4 輸入優先權編碼器的方塊圖

根據行為描述可得到真值表如下：

I_3	I_2	I_1	I_0	V	O_1	O_0
0	0	0	0	0	X	X
1	X	X	X	1	1	1
0	1	X	X	1	1	0
0	0	1	X	1	0	1
0	0	0	1	1	0	0

可畫出其卡諾圖並進行化簡如下：

O_0

I_3I_2 \ I_1I_0	00	01	11	10
00	X		1	1
01				
11	1	1	1	1
10	1	1	1	1

O_1

I_3I_2 \ I_1I_0	00	01	11	10
00	X			
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

V

I_3I_2 \ I_1I_0	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

我們可以得到以下的式子：

$O_0 = I_1I_2' + I_3$

$O_1 = I_2 + I_3$

$V = I_0 + I_1 + I_2 + I_3$

故可得到 4 輸入優先權編碼器的電路圖，如圖 5.49 所示。

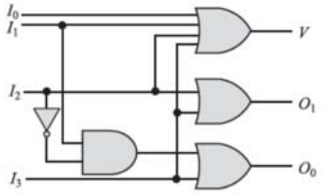


圖 5.49 4 輸入優先權編碼器的電路圖

在此特別說明一下，在資料壓縮領域，有時候會把執行資料壓縮的壓縮器也稱為編碼器，此時壓縮編碼器的動作可能十分複雜。在通訊領域，編碼器則可能是被應用來執行雜訊消除、錯誤偵測與更正的電路。本章所提及的編碼器則只是一個功能相當基本的編碼器，只負責將資料編碼成二進位輸出，沒有其他特殊功能。

5.8 解碼器

解碼器 **(decoder)** 會執行和編碼器完全相反的動作，它會對輸入的二進制碼，進行解碼的動作。解碼器可以將 k 個輸入訊號轉換成 2^k 條輸出訊號，假設有 m 個輸入與 n 個輸出，則稱為 m 對 n 解碼器。例如，有一個包含 2 條輸入線、4 條輸出線的 2 對 4 解碼器 (2-to-4 decoder)，此電路可以將輸入的二進制碼進行如下的解碼動作：

當兩條輸入線輸入 **00** 時，只有第一條輸出線輸出 **1**，其他三條輸出線為 **0**。

當兩條輸入線輸入 **01** 時，只有第二條輸出線輸出 **1**，其他三條輸出線為 **0**。

當兩條輸入線輸入 **10** 時，只有第三條輸出線輸出 **1**，其他三條輸出線為 **0**。

當兩條輸入線輸入 **11** 時，只有第四條輸出線輸出 **1**，其他三條輸出線為 **0**。

很顯然地，兩個位元具備四種組合，因此每次的輸入，只會讓一條輸出線輸出 1，其他輸出線皆為 0。

接下來我們將介紹兩個不同的解碼器，分別是 2 對 4 致能解碼器與 3 對 8 解碼器。

2 對 4 致能解碼器

假設我們用 I_1、I_0 和 en 分別代表解碼器的高、低位元輸入和致能輸入，用 O_3～O_0 分別代表解碼器從高位元到低位元的解碼輸出，則一個 **2** 對 **4** 致能解碼器 **(2-to-4 decoder with enable signal)** 如圖 5.50 所示。2 對 4 致能解碼器行為描述如下：

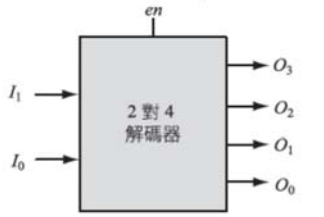


圖 5.50 2-to-4 致能解碼器方塊圖

當致能訊號為 **0** 時，輸出全為 **0 ($O_3O_2O_1O_0$ = 0000)**。

當致能訊號為 **1** 且輸入值 I_1I_0 為 **00** 時，最低位元輸出為 **1**，其餘輸出為 **0 ($O_3O_2O_1O_0$ = 0001)**。

當致能訊號為 **1** 且輸入值 I_1I_0 為 **01** 時，則第二位元輸出為 **1**，其餘輸出為 **0 ($O_3O_2O_1O_0$ = 0010)**。

當致能訊號為 **1** 且輸入值 I_1I_0 為 **10** 時，則第三位元輸出為 **1**，其餘輸出為 **0 ($O_3O_2O_1O_0$ = 0100)**。

當致能訊號為 **1** 且輸入值 I_1I_0 為 **11** 時，則最高位元輸出為 **1**，其餘輸出為 **0 ($O_3O_2O_1O_0$ = 1000)**。

當致能輸入 *en* 為 0 時，不管輸入值為何，此電路是被禁能的，輸出訊號全部都是 0。當致能輸入 *en* 為 1 時，才致能這個解碼器 (啟動解碼器)，此時只會有一條輸出線的值為 1，其他輸出線的輸出皆為 0。
由以上的行為描述，我們可以列出如下的功能表：

輸入			輸出			
en	I_1	I_0	O_3	O_2	O_1	O_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

觀察真值表，我們可以寫出相對應的布林函數：
$O_0 = (enI_1'I_0')$
$O_1 = (enI_1'I_0)$
$O_2 = (enI_1I_0')$
$O_3 = (enI_1I_0)$
根據以上的布林函數，我們可以實現這個 2 對 4 致能解碼器電路，如圖 5.51 所示。

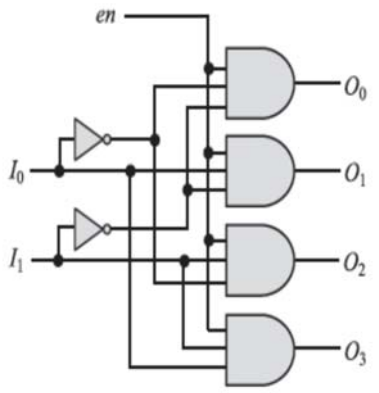


圖 5.51 2-to-4 致能解碼器電路圖

同理，我們也可以設計一個具負邏輯致能訊號的 2 對 4 解碼器，
使用如下的功能表：

輸入			輸出			
en	I_1	I_0	O_3	O_2	O_1	O_0
1	X	X	0	0	0	0
0	0	0	0	0	0	1
0	0	1	0	0	1	0
0	1	0	0	1	0	0
0	1	1	1	0	0	0

當致能輸入 *en* 為 1 時，不管輸入值為何，此電路是被禁能的，輸出訊號全部都是 0。當致能輸入 *en* 為 0 時，才致能這個解碼器 (啟動解碼器)，此時只會有一條輸出線的值為 1，其他輸出線的輸出皆為 0。觀察真值表，我們可以寫出相對應的布林函數：
$O_0 = (en'I_1'I_0')$
$O_1 = (en'I_1'I_0)$
$O_2 = (en'I_1I_0')$
$O_3 = (en'I_1I_0)$
根據以上的布林函數，我們可以實現這個 2 對 4 負邏輯致能解碼器電路，如圖 5.52 所示。

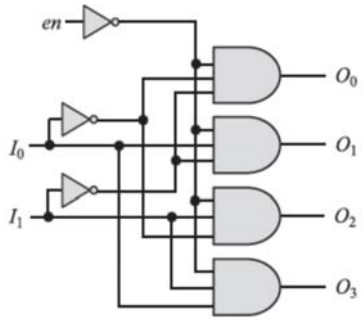


圖 5.52 2-to-4 負邏輯致能解碼器電路圖

3 對 8 解碼器

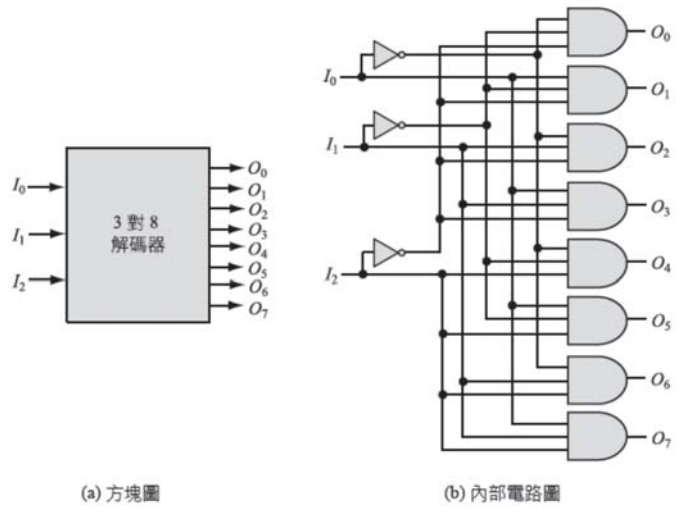
假設我們用 I_2、I_1、I_0 分別代表解碼器的三位元輸入，用 O_7～O_0 分別代表解碼器從高位元到低位元的解碼輸出，則一個 **3** 對 **8** 解碼器 **(3-to-8 decoder)** 的真值表如下所示：

輸入			輸出							
I_2	I_1	I_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

根據真值表，我們可列出輸出訊號與輸入訊號的布林代數表示式為：

$O_0 = (I_2'I_1'I_0')$
$O_1 = (I_2'I_1'I_0)$
$O_2 = (I_2'I_1I_0')$
$O_3 = (I_2'I_1I_0)$
$O_4 = (I_2I_1'I_0')$
$O_5 = (I_2I_1'I_0)$
$O_6 = (I_2I_1I_0')$
$O_7 = (I_2I_1I_0)$

如此一來，可畫出 3 對 8 解碼器的方塊圖與內部電路圖如圖 5.53 所示。



(a) 方塊圖　　(b) 內部電路圖

圖 5.53　3-to-8 解碼器

事實上，除了使用上述正常的方式來設計之外，一個 3 對 8 解碼器也可以使用兩個有致能訊號的 2 對 4 解碼器來連接而成。其連接方式如圖 5.54 所示，輸入最高位元 I_2 和 I_2' 分別當做低四位元和高四位元的 2 對 4 解碼器的致能控制訊號。當 I_2 為 0 時，上面的 2 對 4 解碼器被啟動且下面的解碼器被禁能；當 I_2 為 1 時，上面的 2 對 4 解碼器被禁能且下面的解碼器被啟動，因此藉由這兩個解碼器分別動作，剛好執行 3 對 8 解碼器的所有行為。

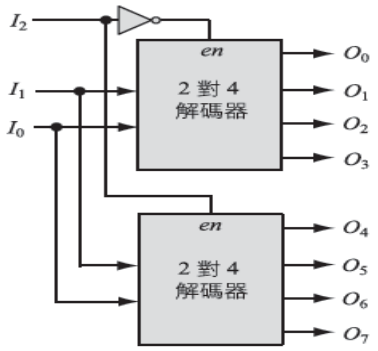


圖 5.54　以 2-to-4 實現 3-to-8 解碼器圖

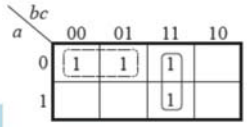
事實上，解碼器還有一個很特殊的應用。任何一個組合電路如果不要求一定要以最簡化 **(**即成本最低**)** 的方式來實現，則一定可以使用解碼器加上適當的 **or** 閘來實現。

在第 4 章曾提及，一個布林代數運算式的積項如果包含所有的輸入變數，則此積項稱之為「最小項」 (minterm)，或標準積項 (standard product term)。任何一個布林函數都可以只使用標準積項來表示，雖然用標準積項描述的布林表示式可能不是最簡化的形式，但只使用標準積項一定可以描述任何一個電路。觀察 3 對 8 解碼器的每一行，可以發現 8 條輸出線的每一條剛好對應到一個三輸入變數標準積項中的一個，因此任何三輸入變數的組合電路都可以使用一個 3 對 8 解碼器加上 or 閘來實現，以下舉兩個範例來說明。

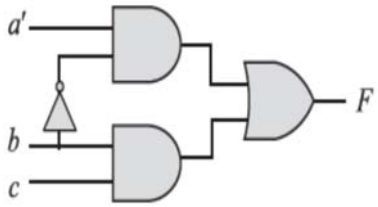
範例 **5.2**
已知三變數布林函數 $F(a, b, c) = \Sigma(0, 1, 3, 7)$，**(a)** 請執行卡諾圖化簡，實現該電路。
(b) 請使用 3 對 8 解碼器，實現該電路。
作法：
(a) 先畫出此函數的真值表，如下所示：

a	*b*	*c*	*F*	
0	0	0	1	m_0
0	0	1	1	m_1
0	1	0	0	m_2
0	1	1	1	m_3
1	0	0	0	m_4
1	0	1	0	m_5
1	1	0	0	m_6
1	1	1	1	m_7

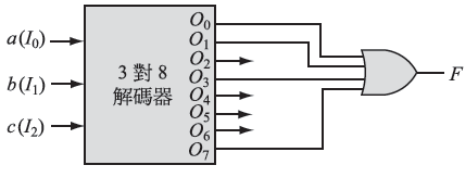
如此一來，可得出其卡諾圖並合併如下：



故函數的最簡 SOP 表示為 $F(a, b, c) = a'b' + bc$。
其實現電路為



(b) 使用 3 對 8 解碼器來實現，則將解碼器解出的 O_0、O_1、O_3 與 O_7 執行 or 運算即得答案，如下圖 (其中 3 對 8 解碼器的內部電路如圖 5.53)



很明顯地，使用卡諾圖化簡可得此函數的最簡化實現電路，相對地，**3** 對 **8** 解碼器也一定可以實現此函數，只是並不是最簡化 **(**成本最低**)** 的電路。任何組合電路都可以用解碼器搭配 **or** 閘來實現。

範例 **5.3**

請使用適當的解碼器電路來實現範例 5.1 的 BCD 碼對 2-4-2-1 碼的轉換電路。

作法：

我們知道使用卡諾圖來化簡可以得到最簡化的實現方式，詳細步驟可參考範例 5.1。

要使用解碼器來實現則需要先有真值表，此題目的真值表如下所示：

十進數	BCD 碼	2-4-2-1 碼
	a b c d	*w x y z*
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1
4	0 1 0 0	0 1 0 0
5	0 1 0 1	1 0 1 1
6	0 1 1 0	1 1 0 0
7	0 1 1 1	1 1 0 1
8	1 0 0 0	1 1 1 0
9	1 0 0 1	1 1 1 1
10 (不使用)	1 0 1 0	0 1 0 1
11 (不使用)	1 0 1 1	0 1 1 0
12 (不使用)	1 1 0 0	0 1 1 1
13 (不使用)	1 1 0 1	1 0 0 0
14 (不使用)	1 1 1 0	1 0 0 1
15 (不使用)	1 1 1 1	1 0 1 0

得知

$w(a, b, c, d) = \Sigma(5, 6, 7, 8, 9)$

$x(a, b, c, d) = \Sigma(4, 6, 7, 8, 9)$

$y(a, b, c, d) = \Sigma(2, 3, 5, 8, 9)$

$z(a, b, c, d) = \Sigma(1, 3, 5, 7, 9)$

故可得下列電路實現圖：