第4章

布林代數與邏輯電路

滄海書局
Tsang Hai Book Publishing Co.

4.1 布林代數

■ 什麼是代數 (algebra) 呢？代數是研究「數」和「數量」之間的關係和結構的一種數學。代數可以用來了解數字做運算時，會產生怎麼樣的結果。例如：n 是輸入值，m 是輸出值，$m = n^2 + 2n + 1$ 這個代數運算式可以表示出 n 與 m 之間的相互關係。

■ 布林代數 (Boolean algebra) 是一位名叫 George Boole 的數學家為了研究邏輯思考，所發展出來的代數系統。他將邏輯思考中的「真」(true) 與「假」(false)，以數字 “1” 和 “0” 來表示，並以二元變數來代表事件的真與假。例如：以二元變數 x 代表 A > B 這件敘述是否正確。如果 A 真的大於 B (即 A > B 這個條件真實發生時)，x = 1。如果 A 小於或等於 B (即 A > B 這個條件沒有發生，或稱這個條件為假時)，x = 0。x 為 1 或 0 會反應出 A > B 這件事是否發生。布林代數在整個邏輯理論的研究發展上，奠定了十分重要的基礎。

■ 布林代數提出不久後，有另一位學者 Claude E. Shannon 提出了「交換代數」(switching algebra)，用以描述開關元件的串、並聯與邏輯理論上 AND、OR 的相關特性。使用交換代數來表示開關電路，重新定義了布林代數在開關電路設計上的廣泛應用。

4.1.1 定義

- 和一般的數學代數一樣，想要了解布林代數，需要知道布林代數中元素 (element) 與運算子 (operator) 的定義。布林代數包含 {1, 0} 兩個元素，以及 “+”、“ · ”、“ ′ ” 三個運算子。其中，“ · ” 與 “+” 兩個二元運算子 (binary operator) 的運算原則，與先前的 and 與 or 邏輯閘相同；“ ′ ” 運算子的運算原則，與 not 邏輯閘相同。圖 4.1、4.2 與 4.3 分別是 and、or 與 not 閘的表示。

and (·)：

輸入		輸出
a	*b*	$F = a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

圖 4.1 and 閘的表示

or (+)：

輸入		輸出
a	*b*	$F = a + b$
0	0	0
0	1	1
1	0	1
1	1	1

圖 4.2 or 閘的表示

not (′)：

輸入	輸出
a	$F = a'$
0	1
1	0

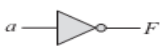


圖 4.3 not 閘的表示

4.1.2 公理

- 布林代數運算就如同一般的數學運算，需要定義一些基本的公理，才能進一步推導出其他的定理。

- 公理 1：封閉性 (closure)
  {0, 1}元素集合中的任一個元素，經過各種不同的二元邏輯運算後，得出的結果一定為 0 或 1，都屬於 {0, 1} 這個元素集合，故具備封閉性。

- 公理 2：交換律 (commutative law)
  二元運算子的左、右兩個變數，可以互相交換，而不影響其運算結果。
  (a) $a + b = b + a$
  $\Rightarrow 0 + 0 = 0 + 0;\ 0 + 1 = 1 + 0;\ 1 + 0 = 0 + 1;\ 1 + 1 = 1 + 1$
  (b) $a \cdot b = b \cdot a$
  $\Rightarrow 0 \cdot 0 = 0 \cdot 0;\ 0 \cdot 1 = 1 \cdot 0;\ 1 \cdot 0 = 0 \cdot 1;\ 1 \cdot 1 = 1 \cdot 1$

- 公理 3：結合律 **(associate law)**
  變數在同一種運算中可以相互結合而不影響其結果。換句話說，
  $(a + b) + c = a + (b + c)$ 且 $(a \cdot b) \cdot c = a \cdot (b \cdot c)$。
  例如：當 $a = 1, b = 1$ 且 $c = 0$，則
  (a) $(a + b) + c = (1 + 1) + 0 = 1 + 0 = 1$
  $a + (b + c) = 1 + (1 + 0) = 1 + 1 = 1$
  $\Rightarrow (a + b) + c = a + (b + c)$
  (b) $(a \cdot b) \cdot c = (1 \cdot 1) \cdot 0 = 1 \cdot 0 = 0$
  $a \cdot (b \cdot c) = 1 \cdot (1 \cdot 0) = 1 \cdot 0 = 0$
  $\Rightarrow (a \cdot b) \cdot c = a \cdot (b \cdot c)$
- 公理 4：分配律 **(distributive law)**

某一種運算子對另一種運算子具有分配性質的運算，而不影響其結果。
(a) "$\cdot$" 對 "+" 可進行分配運算，故 $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
此式可使用表 4.1 的真值表證明得出。

表 4.1 $a \cdot (b + c)$ 與 $(a \cdot b) + (a \cdot c)$ 的真值表

a	b	c	$b + c$	$a \cdot (b + c)$	$a \cdot b$	$a \cdot c$	$(a \cdot b) + (a \cdot c)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

觀察有陰影那兩欄，很顯然地，每一種輸入組合，其相對應輸出都一樣，
故 a · (b + c) = (a · b) + (a · c) 得證。
(b) "+" 對 "·" 可進行分配運算，故 a + (b · c) = (a + b) · (a + c)
此式可使用表 4.2 的真值表證明得出。
觀察有陰影那兩欄，每一種輸入組合，其相對應輸出都相同，
故 a + (b · c) = (a + b) · (a + c) 得證。
在一般的數學中，加法無法對乘法進行分配運算。但是在布林代數中，"+" 對 "·" 可進行分配運算，所以 $a + (b \cdot c) = (a + b) \cdot (a + c)$ 成立。

表 4.2 $a + (b \cdot c)$ 與 $(a + b) \cdot (a + c)$ 的真值表

a	b	c	$b \cdot c$	$a + (b \cdot c)$	$a + b$	$a + c$	$(a + b) \cdot (a + c)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

假設 ***X*** 和 ***Y*** 是兩種布林代數表示式，若對所有可能的輸入排列組合，***X*** 與 ***Y*** 的輸出值都相同，則我們稱此兩種表示 **(representation)** 是「同等的」、「全等的」或「等效的」**(equivalent)**。如上述的例子，$X = a + (b \cdot c)$ 且 $Y = (a + b) \cdot (a + c)$。因為 **3** 個輸入 (***a***、***b*** 與 ***c***) 總共可產生 **8** 種可能的輸入組合，觀察真值表陰影那兩欄，得知 **8** 種可能輸入中的任何一種，其 ***X*** 與 ***Y*** 的輸出值都相同，故我們可說 ***X*** 同等於 ***Y*** 或是 $a + (b \cdot c)$ 同等於 $(a + b) \cdot (a + c)$。同理可證，$a \cdot (b + c)$ 同等於 $(a \cdot b) + (a \cdot c)$。

■ 公理 5：單位元素 (identity element)

(a) “ · ” 運算有一單位元素 “1”，使得任何變數與其進行 “ · ” 運算之後，結果不變。

$a \cdot 1 = 1 \cdot a = a$

(b) “+” 運算有一單位元素 “0”，使得任何變數與其進行 “+” 運算之後，結果不變。

$a + 0 = 0 + a = a$

■ 公理 6：反元素 (inverse)

a 是 {0, 1} 元素集合中的一個元素，則必存在一個也屬於 {0, 1} 集合的元素 a′ (又稱為 a 之補數)，使得

(a) $a + a' = 1$

(b) $a \cdot a' = 0$

4.1.3 定理

有了定義及公理，我們可以得到布林代數的定理如下：

定理 1：

(a) $a + a = a$

證明如下：

$a + a = (a + a) \cdot 1$	變數 · 1 結果不變 (單位元素)
$= (a + a) \cdot (a + a')$	$1 = (a + a')$ (反元素)
$= a + (a \cdot a')$	對 $(a + a) \cdot (a + a')$ 提出 a + (分配律)
$= a + 0$	$a \cdot a' = 0$ (反元素)
$= a$	變數 + 0 結果不變 (單位元素)

(b) $a \cdot a = a$

證明如下：

$a \cdot a = (a \cdot a) + 0$	變數 + 0 結果不變 (單位元素)
$= (a \cdot a) + (a \cdot a')$	$0 = a \cdot a'$ (反元素)
$= a \cdot (a + a')$	對 $(a \cdot a) + (a \cdot a')$ 提出 a · (分配律)
$= a \cdot 1$	$(a + a') = 1$ (反元素)
$= a$	變數 · 1 結果不變 (單位元素)

定理 2：

(a) $a + 1 = 1$

證明如下：

$a + 1 = 1 \cdot (a + 1)$	$1 \cdot$ 任何變數結果不變 (單位元素)
$= (a + a') \cdot (a + 1)$	$1 = (a + a')$ (反元素)
$= a + (a' \cdot 1)$	對 $(a + a') \cdot (a + 1)$ 提出 $a +$ (分配律)
$= a + a'$	變數 $\cdot 1$ 結果不變 (單位元素)
$= 1$	$(a + a') = 1$ (反元素)

(b) $a \cdot 0 = 0$

證明如下：

$a \cdot 0 = 0 + (a \cdot 0)$	$0 +$ 任何變數結果不變 (單位元素)
$= (a \cdot a') + (a \cdot 0)$	$0 = a \cdot a'$ (反元素)
$= a \cdot (a' + 0)$	對 $(a \cdot a') + (a \cdot 0)$ 提出 $a \cdot$ (分配律)
$= a \cdot a'$	變數 $+ 0$ 結果不變 (單位元素)
$= 0$	$a \cdot a' = 0$ (反元素)

定理 3：

(a) $a + ab = a$

證明如下：

$a + ab = (a \cdot 1) + a \cdot b$	變數 $\cdot 1$ 結果不變 (單位元素)
$= a \cdot (1 + b)$	對 $(a \cdot 1) + a \cdot b$ 提出 $a \cdot$ (分配律)
$= a \cdot (b + 1)$	b 與 1 互換 (交換律)
$= a \cdot 1$	變數 $+ 1$ 結果為 1 (定理 2(a))
$= a$	變數 $\cdot 1$ 結果不變 (單位元素)

(b) $a(a + b) = a$

證明如下：

$a(a + b) = a \cdot a + a \cdot b$	$a \cdot$ 對 $a + b$ 分配 (分配律)
$= a + a \cdot b$	$a \cdot a = a$ (定理 1(b))
$= (a \cdot 1) + (a \cdot b)$	$a = a \cdot 1$ (單位元素)
$= a \cdot (1 + b)$	對 $(a \cdot 1) + (a \cdot b)$ 提出 $a \cdot$ (分配律)
$= a \cdot (b + 1)$	b 與 1 互換 (交換律)
$= a \cdot 1$	變數 $+ 1$ 結果為 1 (定理 2(a))
$= a$	變數 $\cdot 1$ 結果不變 (單位元素)

定理 4：迪摩根定理 (DeMorgan's theorems)

(a) $(a + b)' = a' \cdot b'$

(b) $(a \cdot b)' = a' + b'$

由邏輯學家和數學家迪摩根所提出的迪摩根定理，在布林代數的應用上十分重要，是使用率很高，非常重要的定理。迪摩根定理的主要意義為：

- 兩值先做 "+"，再反相 (not) 的結果，同等於兩值分別反相再做 "·"。
- 兩值先做 "·"，再反相 (not) 的結果，同等於兩值分別反相再做 "+"。

使用真值表來證明迪摩根定理是較為簡潔的方式，證明如下：

表 4.3 $(a + b)'$ 與 $a' \cdot b'$ 的真值表

a	b	$a + b$	$(a + b)'$	a'	b'	$a' \cdot b'$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

表 4.4 $(a \cdot b)'$ 與 $a' + b'$ 的真值表

a	b	$a \cdot b$	$(a \cdot b)'$	a'	b'	$a' + b'$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

(a) $(a + b)' = a' \cdot b'$ 可使用表 4.3 證明。

觀察有陰影那兩欄，每一種輸入組合，其相對應輸出都相同，故 $(a + b)' = a' \cdot b'$，即 $(a + b)'$ 同等於 $a' \cdot b'$。

(b) $(a \cdot b)' = a' + b'$ 可使用表 4.4 證明。

觀察有陰影那兩欄，每一種輸入組合，其相對應輸出都相同，故 $(a \cdot b)' = a' + b'$，即 $(a \cdot b)'$ 同等於 $a' + b'$。

4.1.4 對偶性 (duality)

從上述的公理、定理中可觀察出，每項公理、定理均包含 (a)、(b) 兩項，即每一個 “+” 的運算定律，必定有相對應的 “ · ” 運算。因為值域集合中只有 0 與 1，而且 0 與 1 正好是 “+” 與 “ · ” 之單位元素，因此在定理的 (a)、(b) 兩項中，只要將 “+” 與 “ · ” 及 “0” 與 “1” 互相調換，即成為對方的定理，此一性質稱為「對偶性質」。在任何的布林函數中，均可應用對偶性質，將其運算符號與單位元素互換，其運算式子仍可成立。將上述的公理與定理整理如表 4.5 所示。

表 4.5 布林定理的對偶性

	(a)	(b)
交換律	$a+b=b+a$	$a \cdot b=b \cdot a$
結合律	$(a+b)+c=a+(b+c)$	$(a \cdot b) \cdot c=a \cdot(b \cdot c)$
分配律	$a \cdot(b+c)=(a \cdot b)+(a \cdot c)$	$a+(b \cdot c)=(a+b) \cdot(a+c)$
單位元素	$a \cdot 1=a$	$a+0=a$
定理 1	$a+a=a$	$a \cdot a=a$
定理 2	$a+1=1$	$a \cdot 0=0$
定理 3	$a+ab=a$	$a(a+b)=a$
定理 4：迪摩根定理	$(a+b)'=a' \cdot b'$	$(a \cdot b)'=a'+b'$

4.1.5 運算優先順序 (operator precedence)

欲求布林代數之運算值時，其邏輯運算順序如下：

(1) 括號 “()”

(2) not “ ′ ”

(3) and “ · ”

(4) or “+”

如果運算式中有括號，則括號內之運算優先執行。例如：表示式為 $(A+B)' \cdot C$，先求括號內之 $A+B$ 值，再求其 not (′) 運算，最後才計算 and (·)。假設表示式為 $A+B' \cdot C$，我們需先求 B 的 not 得到 B'，再求 and 運算得出 $B' \cdot C$ 的結果，最後再執行 or 運算來算出 A 與 $B' \cdot C$ 執行 + 的結果。顯然地，運算順序與一般代數的先乘 (·) 後加 (+)，及括號內之運算優先執行的方式是完全相同的。

4.2 布林函數

表 4.6 布林函數 F 的真值表

a	b	$F = a + b$
0	0	0
0	1	1
1	0	1
1	1	1

正如一般的數學代數運算，布林代數也有函數的表示，而布林函數 **(Boolean functions)** 可用來定義數位系統的輸入訊號與輸出訊號之間的關係。通常我們使用布林代數運算式 (Boolean algebraic expression) 來描述布林函數，例如：布林函數 $F(a, b) = a + b$，代表 F 是變數 a 與 b 的函數，$a + b$ 是布林代數運算式 (一種用來描述二元變數與邏輯運算子相互關係的表示式)。其中，“+” 代表邏輯 or 運算，輸入二元變數 a 與 b 的值，只有 “0” 或 “1” 兩種可能；而布林函數 F 也一樣，只有 “0” 或 “1” 兩個可能值。

一個布林函數除了可以使用布林代數運算式來表示之外，也可以使用真值表的方式來表示。在上述布林函數 $F(a, b) = a + b$ 這個例子，因為輸入 a 與 b 為二元變數，所以可以得到 $2^n = 4$ 種組合 (輸入變數有 a、b 兩個，故 $n = 2$)，其真值表如表 4.6 所示。

將 a 與 b 的每一種輸入組合情況，執行 $a + b$ 的運算。當 $a = 0$ 且 $b = 0$，$F = 0 + 0$，所以 F 為 0；當 $a = 0$ 且 $b = 1$，$F = 0 + 1$，所以 F 為 1；當 $a = 1$ 且 $b = 0$，$F = 1 + 0$，所以 F 為 1；當 $a = 1$ 且 $b = 1$，$F = 1 + 1$，所以 F 為 1。根據這四個情況，可得出表 4.6 的真值表。

總而言之，一個布林函數有兩種表示方式：(1) 布林代數運算式；(2) 真值表。下面的範例 4.1 與 4.2 說明兩者之間如何互相轉換。

範例 4.1

已知一個布林函數 F(a, b, c) = a · b + a · c + a · b · c，試畫出其真值表。

作法 1：

3 個輸入變數有 8 種輸入組合。將每一組輸入組合代入布林運算式中以求出 F。

$a \cdot b$　$a \cdot c$　$a \cdot b \cdot c$

(1) 當 $a = 0, b = 0$ 且 $c = 0$ 時，$F = (0 \cdot 0) + (0 \cdot 0) + (0 \cdot 0 \cdot 0) = 0$。
(2) 當 $a = 0, b = 0$ 且 $c = 1$ 時，$F = (0 \cdot 0) + (0 \cdot 1) + (0 \cdot 0 \cdot 1) = 0$。
(3) 當 $a = 0, b = 1$ 且 $c = 0$ 時，$F = (0 \cdot 1) + (0 \cdot 0) + (0 \cdot 1 \cdot 0) = 0$。
(4) 當 $a = 0, b = 1$ 且 $c = 1$ 時，$F = (0 \cdot 1) + (0 \cdot 1) + (0 \cdot 1 \cdot 1) = 0$。
(5) 當 $a = 1, b = 0$ 且 $c = 0$ 時，$F = (1 \cdot 0) + (1 \cdot 0) + (1 \cdot 0 \cdot 0) = 0$。
(6) 當 $a = 1, b = 0$ 且 $c = 1$ 時，$F = (1 \cdot 0) + (1 \cdot 1) + (1 \cdot 0 \cdot 1) = 1$。
(7) 當 $a = 1, b = 1$ 且 $c = 0$ 時，$F = (1 \cdot 1) + (1 \cdot 0) + (1 \cdot 1 \cdot 0) = 1$。
(8) 當 $a = 1, b = 1$ 且 $c = 1$ 時，$F = (1 \cdot 1) + (1 \cdot 1) + (1 \cdot 1 \cdot 1) = 1$。

上述 8 列中的每一列，對應到真值表中的一列 (某一特定的 a、b、c 輸入及其相對應的 F 輸出)，所以可畫出如下的真值表：

a	b	c	F	
0	0	0	0	(1)
0	0	1	0	(2)
0	1	0	0	(3)
0	1	1	0	(4)
1	0	0	0	(5)
1	0	1	1	(6)
1	1	0	1	(7)
1	1	1	1	(8)

作法 2：

可依下列步驟，一步一步畫出其真值表。

3 個輸入變數有 8 種輸入組合，故首先標示 8 列的輸入組合，從 abc = 000 到 abc = 111 (下表的前三欄)。

接著，參考第一、二欄 a 與 b 的值，and (且) 在一起，可決定出 ab 的值，將不同列的 ab 值標示於下表的第四欄。參考第一、三欄 a 與 c 的值，and 在一起，決定出 ac 的值，將不同列的 ac 值標示於下表的第五欄。參考第一、二、三欄 a、b 與 c 的值，and 在一起，決定出 abc 的值，將不同列的 abc 值標示於下表的第六欄。

因為 F(a, b, c) = ab + ac + abc，所以將第四、五、六欄的值，or(或) 在一起可得到 F。

a	*b*	*c*	*ab*	*ac*	*abc*	*F = ab + ac + abc*
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	0	0	0
1	0	0	0	0	0	0
1	0	1	0	1	0	1
1	1	0	1	0	0	1
1	1	1	1	1	1	1

作法 3：

事實上，還有第三種方法可以推導出真值表。首先觀察 F(a, b, c) = ab + ac + abc，此式把 ab、ac、abc 三個項 or 在一起，故只要任何一個項為 1，F 也一定為 1。

第一項 (term) 是 ab，所以我們知道當 a = 1 且 b = 1 時，無論 c = 0 或 1 都會讓 F 等於 1，所以真值表中輸入組合為 (a = 1 且 b = 1 且 c = 0) 與 (a = 1 且 b = 1 且 c = 1) 那兩列的輸出應該都為 1。

第二項是 ac，所以我們知道當 a = 1 且 c = 1 時，無論 b = 0 或 1 都會讓 F 等於 1，所以真值表中輸入組合為 (a = 1 且 b = 0 且 c = 1) 與 (a = 1 且 b = 1 且 c = 1) 那兩列的輸出應該都為 1。

第三項是 abc，所以我們知道當 a = 1 且 b = 1 且 c = 1 時，F 等於 1，所以真值表中輸入組合為 (a = 1 且 b = 1 且 c = 1) 那一列的輸出應該為 1。總和後可知，真值表在 (*a* = 1 且 *b* = 0 且 *c* = 1)、(*a* = 1 且 *b* = 1 且 *c* = 0)、(*a* = 1 且 *b* = 1 且 *c* = 1) 這三種輸入組合下的輸出為 1，其他輸入組合情況的輸出都為 0。根據上述的結論，我們可以畫出符合 *F*(*a*, *b*, *c*) = *ab* + *ac* + *abc* 的真值表。

範例 **4.2**
已知一個布林函數的真值表如下所示，試寫出該函數的布林代數
運算式。作法：

a	*b*	*c*	*F*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

觀察真值表，可得知：
(1) 當 a = 0, b = 0 且 c = 1 時，F = 1。
(2) 當 a = 1, b = 1 且 c = 1 時，F = 1。
(3) 其他輸入條件時，F 皆為 0。

根據情況 (1)，我們可推導出 a′ · b′ · c 此項為 1，可以確保 F 得到 1。
根據情況 (2)，我們可推導出 a · b · c 此項為 1，可以確保 F 得到 1。
故 F 的布林代數運算式可以寫成：
F(a, b, c) = a′b′c + abc，
當 a = 0,b = 0 且 c = 1 時，a′b′c 這一項為 1；或者是當 a = 1, b = 1 且 c = 1 時，abc 項為 1，兩種情況的任一種發生，都會使 F 得到 1。

4.2.1 正規表示

布林函數運算式中若將所有積項 (也就是 and term) 以「和」運算 (“+”，or) 來組合，則所得出的式子稱之為「積項的和」表示式 (sum of products，簡稱 SOP)。如下所示，兩式都是 SOP 的形式，將數個積項 or “+” 在一起。
E(a, b, c) = ab + ac′ + abc (含 3 個積項： ab、ac′、abc)
F(a, b, c) = ab′ + ac + a′b′c′ (含 3 個積項)
除了「積項的和」表示法之外，也可將所有和項 (即 or term) 以「積」運算 (“ · ”，and) 來組合，稱之為「和項的積」表示式 (product of sums，簡稱 POS)。如下所示，兩式都是 POS 的形式，將數個和項 and “ · ” 在一起。
G(a, b, c) = (b + c′) · (a + b + c) (含 2 個和項：b + c′、a + b + c)
H(a, b, c) = (a + b′) · (a + c) · (a′ + b′ + c′) (含 3 個和項)

SOP 與 POS 這兩種形式的表示法，稱為布林函數的標準表示法。
一個布林代數運算式，若其式子完全符合「積項的和」或是「和項的積」，則稱為「標準型布林代數表示式」；否則，稱之為「非標準型布林代數表示式」。例如：函數 $F = (ab + bc)(a + c)$，它不是標準型布林代數表示式。但可利用分配律 (如下式)，將 F 函數整理成為 $F = ab + abc + bc$，使其符合積項的和 (SOP) 型式，而成為標準型布林代數表示式。

$F(a, b, c) = (ab + bc)(a + c)$
$= (ab + bc) \cdot (a + c)$
$= ((ab + bc) \cdot a) + ((ab + bc) \cdot c)$
$= (aab + abc) + (abc + bcc)$
$= ab + abc + bc$ ($aa = a$ 且 $cc = c$) → 將三個積項 or 在一起

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)。一個布林代數運算式的和項如果包含所有的輸入變數，此和項則稱為「最大項」(maxterm)，或標準和項 (standard sum term)。表 4.7 為三個輸入變數之所有「標準積項」與「標準和項」的組合。其中 m0、m1、…、m7 代表三個變數之標準積項的項號，而 M0、M1、…、M7 則代表三個變數之標準和項的項號。很明顯地，Mi = mi′，例如：m4 = ab′c′，則 m4′ = (ab′c′)′ = a′ + b + c (迪摩根定理)，等於 M4。

表 4.7　三變數 (a、b、c) 形成之標準積項與標準和項

a	b	c	標準積項 (minterm)		標準和項 (maxterm)	
0	0	0	$a'b'c'$	m_0	$a + b + c$	M_0
0	0	1	$a'b'c$	m_1	$a + b + c'$	M_1
0	1	0	$a'bc'$	m_2	$a + b' + c$	M_2
0	1	1	$a'bc$	m_3	$a + b' + c'$	M_3
1	0	0	$ab'c'$	m_4	$a' + b + c$	M_4
1	0	1	$ab'c$	m_5	$a' + b + c'$	M_5
1	1	0	abc'	m_6	$a' + b' + c$	M_6
1	1	1	abc	m_7	$a' + b' + c'$	M_7

一個積項的和 SOP 標準型布林代數表示式中，若全部的項都是「標準積項」，則稱之為「正規型表示」(canonical form)。例如：三變數布林函數 F(a, b, c) = a + bc + a′bc′ 是由三個積項所組合而成，式子中除了 a′bc′ 項為「標準積項」之外，其餘的 a 與 bc 都只是一般積項，故 F 稱為「積項的和」的標準型布林代數表示式，但不是「積項的和」的正規型布林代數表示式。而三變數布林函數 G(a, b, c) = abc + ab′c + a′bc′ + a′bc 是由四個積項所組合而成，因為每個積項都是「標準積項」，故 G 不僅是「積項的和」的標準型布林代數表示式，更是「積項的和」的正規型布林代數表示式。

同理，一個和項的積 POS 標準型布林代數表示，若其所有的項都是「標準和項」，則稱為「正規型表示」。例如：三變數布林函數 $F(a, b, c) = (a)(b + c)(a' + b + c')$ 是由三個和項所組合而成，式子中除了 $a' + b + c'$ 項為「標準和項」之外，其餘的 a 與 $b + c$ 都只是一般和項，故 F 稱為「和項的積」的標準型布林代數表示式，但不是「和項的積」的正規型布林代數表示式。而三變數布林函數 $G(a, b, c) = (a + b + c)(a + b' + c)(a' + b + c')(a' + b + c)$ 是由四個和項所組合而成，因為每個和項都是「標準和項」，故 G 不僅是「和項的積」的標準型布林代數表示式，更是「和項的積」的正規型布林代數表示式。

符合「正規型表示」的布林函數，由於所使用的項都是標準項，所以可用如表 4.7 所示的項號來表示。對於「積項的和」之正規型函數，可用 “Σ(…)” 來表示 (括號內為其項號)，而「和項的積」之正規型函數，可用 “Π(…)” 來表示 (括號內為其項號)。例如：

(1) F1(a, b, c) = a′b′c + ab′c′ + abc。

參考表 4.7，F1 也可以表示成 F1(a, b, c) = m1 + m4 + m7 = Σ(1, 4,7)。換句話說，F1(a, b, c) = Σ(1, 4, 7) 與 F1(a, b, c) = a′b′c + ab′c′+ abc 這兩種表示式，代表行為完全相同的布林函數，兩者「同等」。

(2) F2(a, b, c) = (a + b + c)(a + b′ + c)(a + b′ + c′)

參考表 4.7，F2 也可以表示成 F2(a, b, c) = M0 · M2 · M3 = Π (0, 2,3)。F2(a, b, c) = Π(0, 2, 3) 與 F2(a, b, c) = (a + b + c)(a + b′ + c)(a+ b′ + c′) 這兩種表示式，代表行為完全相同的布林函數，兩者「同等」。

由於任何一個布林函數都可以使用真值表來表示，也因此一定能使用正規型的 SOP 來表示該函數。說明如下：真值表的每一列都對應一個標準積項，把那些讓輸出為 1 的標準積項 or “+” 起來，就可得出函數的正規型 SOP 表示。

範例 **4.3**

已知一個布林函數的真值表如下所示，試用正規型的 SOP 來表示該函數。作法：

觀察真值表，我們知道

(1) a = 0 且 b = 0 且 c = 1 時，F = 1。

(2) a = 0 且 b = 1 且 c = 1 時，F = 1。

(3) a = 1 且 b = 1 且 c = 1 時，F = 1。

(4) 其他輸入情況，F = 0。

根據表 4.7，我們知道條件 (1)、(2)、(3) 的變數值分別代表標準積項 m1、m3、m7，故函數可以表示成 F = m1 + m3 + m7 (把那些讓輸出為 1 的標準積項 or 起來) 或是寫成 F = a′b′c + a′bc + abc = Σ(1, 3, 7) 的正規型 SOP 表示。

a	*b*	*c*	*F*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

對一個正規型 SOP 表示的布林函數而言，該函數之補函數的各個項會等於原函數標準積項之外的其餘項。說明如下：由於正規型 SOP 函數乃是將真值表中標示為 “1” 的標準積項 or 在一起，所以其補函數便是將標示為 “0” 的標準積項 or 在一起。例如：布林函數 $F(a, b, c)$ = Σ(1, 4, 5, 6) 的真值表如表 4.8 所示。

表 4.8 $F(a, b, c) = \sum(1, 4, 5, 6)$ 的真值表

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

F(a, b, c) = Σ(1, 4, 5, 6) = m1 + m4 + m5 + m6。從真值表中得知補函數 F′ 是標示為 0 之標準積項的和，所以補函數 F′ 的正規型 SOP 表示可寫成F′(a, b, c) = Σ(0, 2, 3, 7) = m0 + m2 + m3 + m7如此一來，F′ 的補函數等於 F，表示成

(F′)′ = F = (m0 + m2 + m3 + m7)′

= (m0)′ · (m2)′ · (m3)′ · (m7)′ (迪摩根定理)

= M0 · M2 · M3 · M7 (Mi = mi′)

= Π(0, 2, 3, 7)

可得下列結論，

F = Σ(1, 4, 5, 6) = (F′)′ = Π(0, 2, 3, 7)

即 F(a, b, c) = m1 + m4 + m5 + m6 = M0 · M2 · M3 · M7。

根據上面的例子，若要將一個「積項的和」正規型表示，轉換為「和項的積」的表示，則只需將 Σ 與 Π 互換，且將括號內之項換成原式所缺之項即可。例如上述之函數 F 為 Σ(1, 4, 5, 6)，其積項為 1、4、5、6，所缺之項應為 0、2、3、7，因此其「和項之積」為 Π(0, 2, 3,7)。同理可證，G = Σ(1, 6, 7) = Π(0, 2, 3, 4, 5)；H = Σ(0, 1, 2, 5, 6) = Π(3,4, 7)。相對地，若要將一個「和項的積」正規型函數，轉換為「積項的和」的表示，則只需將 Π 與 Σ 互換，且將括號內之項換成原式所缺之項即可。$G(a, b, c)$ = Π(1, 3, 5) = Σ(0, 2, 4, 6, 7)；H = Π(2, 4, 6, 7) = Σ(0, 1,3, 5)。

範例 4.4
已知布林函數 F = ab + ac′ + a′bc, G = a′c + b′c + abc，請分別用正規型的 SOP 與 POS 來表示函數 F 與 G。
作法：
a · b a · c′ a′ · b · c
(1) 當 a = 0, b = 0 且 c = 0 時，F = (0 · 0) + (0 · 1) + (1 · 0 · 0) = 0。同理，G = 0。
(2) 當 a = 0, b = 0 且 c = 1 時，F = (0 · 0) + (0 · 0) + (1 · 0 · 1) = 0。同理，G = 1。
(3) 當 a = 0, b = 1 且 c = 0 時，F = (0 · 1) + (0 · 1) + (1 · 1 · 0) = 0。同理，G = 0。
(4) 當 a = 0, b = 1 且 c = 1 時，F = (0 · 1) + (0 · 0) + (1 · 1 · 1) = 1。同理，G = 1。
(5) 當 a = 1, b = 0 且 c = 0 時，F = (1 · 0) + (1 · 1) + (0 · 0 · 0) = 1。同理，G = 0。
(6) 當 a = 1, b = 0 且 c = 1 時，F = (1 · 0) + (1 · 0) + (0 · 0 · 1) = 0。同理，G = 1。
(7) 當 a = 1, b = 1 且 c = 0 時，F = (1 · 1) + (1 · 1) + (0 · 1 · 0) = 1。同理，G = 0。
(8) 當 a = 1, b = 1 且 c = 1 時，F = (1 · 1) + (1 · 0) + (0 · 1 · 1) = 1。同理，G = 1。

a	*b*	*c*	*F*	*G*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

根據表 4.7，我們知道 F = m3 + m4 + m6 + m7 (把輸出為 1 的標準積項or起來)，也可寫成 F = a′bc + ab′c′ + abc′ + abc = Σ(3, 4, 6, 7) 的正規型SOP 表示。同理，G = m1 + m3 + m5 + m7，也可寫成 G = a′b′c + a′bc +ab′c + abc = Σ(1, 3, 5, 7) 的正規型 SOP 表示。

因為 F 的補函數 F′ 可用真值表中 F 那欄標示為 0 之標準積項的和來表示，所以 F′ 的正規型 SOP 可寫成F′(a, b, c) = Σ(0, 1, 2, 5) = m0 + m1 + m2 + m5

如此一來，F′ 的補函數等於(F′)′ = F = (m0 + m1 + m2 + m5)′

= (m0)′ · (m1)′ · (m2)′ · (m5)′ (迪摩根定理)

= M0 · M1 · M2 · M5

= Π(0, 1, 2, 5)

故 F = M0 · M1 · M2 · M5 = Π(0, 1, 2, 5)，也可寫成 F = (a + b + c)(a + b + c′)(a + b′ + c) (a′ + b + c′) 的正規型 POS 表示。

同理，G = M0 · M2 · M4 · M6 = Π(0, 2, 4, 6)，也可寫成 G = (a + b +c)(a + b′ + c)(a′ + b + c) (a′ + b′ + c) 的正規型 POS 表示。

4.3 布林函數的實現(implementation)

布林函數可用來定義數位系統的輸入訊號與輸出訊號之間的關係。而要將布林函數以實際的硬體電路來製作實現時，最簡單的方式就是，觀察其布林代數運算式，針對不同的邏輯運算子如：“+”、“·”、“′”等，分別使用第 3 章介紹的數位邏輯閘來一一實現。此外，所有的二元變數代表此電路的輸入，函數值為其輸出，電路則由不同的 AND、OR、NOT 等邏輯閘連接而成。

表 4.9 呈現第 3 章介紹過的兩輸入邏輯閘之相關資料，其中最後一欄列出該邏輯閘所需要之閘傳遞延遲 **(gate propagation delay)**。閘傳遞延遲會因為所使用的 IC 邏輯家族不同、製程不同、晶圓代工 cell 元件庫不同而不一樣，表 4.9 所列的是使用 CMOS .18 μm IC 製程得出之閘傳遞延遲 (時間單位為 10−9 秒)，讀者藉由參考此值可以知道反及閘與反或閘的延遲大概只有及閘與或閘的一半。表 4.10 則列出表 4.9 邏輯閘之相對應真值表。

表 4.9　兩輸入邏輯閘

名稱	圖示	布林函數	閘傳遞延遲 (.18 μm IC 製程
及閘 (and)		$F = ab$ 或 $F = a \cdot b$	0.08 ns
或閘 (or)		$F = a + b$	0.07 ns
反閘 (not)		$F = a'$	0.03 ns
互斥或閘 (xor)		$F = a \oplus b$	0.14 ns
反及閘 (nand)		$F = (ab)'$ 或 $F = (a \cdot b)'$	0.03 ns
反或閘 (nor)		$F = (a + b)'$	0.04 ns
反互斥或閘 (xnor)		$F = (a \oplus b)'$	0.15 ns
緩衝閘 (buffer)		$F = a$	0.06 ns

表 4.10　兩輸入邏輯閘之真值表

布林函數	真值表	布林函數	真值表
$F = ab$ (and)	a b F 0 0 0 0 1 0 1 0 0 1 1 1	$F = (ab)'$ (nand)	a b F 0 0 1 0 1 1 1 0 1 1 1 0
$F = a + b$ (or)	a b F 0 0 0 0 1 1 1 0 1 1 1 1	$F = (a + b)'$ (nor)	a b F 0 0 1 0 1 0 1 0 0 1 1 0
$F = a'$ (not)	a F 0 1 1 0	$F = a$ (buffer)	a F 0 0 1 1
$F = a \oplus b$ (xor)	a b F 0 0 0 0 1 1 1 0 1 1 1 0	$F = (a \oplus b)'$ (xnor)	a b F 0 0 1 0 1 0 1 0 0 1 1 1

已知某特定的布林函數 F(a, b, c) = ab + a′c，則我們可根據布林代數表示式，並依照邏輯運算順序，導出此函數的邏輯電路圖 (circuit diagram or logic diagram)，詳細的推導步驟說明如下。

首先，找出此表示式的邏輯運算順序，再依照順序執行下列步驟：

步驟一： 使用一個 not 閘將輸入 a 做運算，求得 a′。

步驟二： 使用一個 and 閘將步驟一得出的 a′ 與輸入 c 做運算，求得 a′c。

步驟三： 使用一個 and 閘將輸入 a 與 b 做運算，求得 ab。

步驟四： 使用一個 or 閘將步驟二得出的 $a'c$、步驟三得出的 ab 做運算，求得 $ab + a'c$。

圖 4.4 代表導出此邏輯電路圖的四個連續步驟，變數 a、b、c 代表此電路的輸入，函數值為其輸出，電路則由不同的 AND、OR、NOT等邏輯閘連接而成。

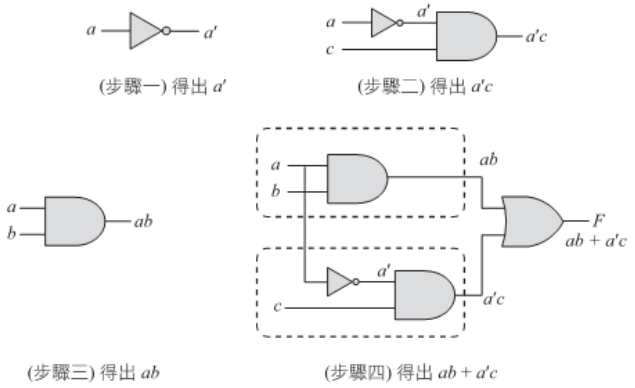


圖 4.4 導出 $F = ab + a'c$ 邏輯電路圖的四個步驟

範例 4.5

已知一個布林函數 F(a, b, c) = c(a + b) + abc，試畫出其邏輯電路圖。

作法：

首先找出此表示式的邏輯運算順序，再依序執行之。

步驟一： 使用一個 or 閘將輸入 a 與 b 做運算，求得 a + b。

步驟二： 使用一個 and 閘將步驟一得出的 (a + b) 與輸入 c 做運算，求得 c(a + b)。

步驟三： 使用一個 and 閘將輸入的 a、b、c 做運算，求得 abc。步驟四： 使用一個 or 閘將步驟二得出的 $c(a + b)$、步驟三得出的 abc 做運算，求得 $c(a + b) + abc$。

這個被推導出的邏輯電路圖 (圖 4.5)，視為設計者根據布林函數所設計出來的「數位電路」。就如同 3.3 節所述，設計完成的數位電路可使用各種不同的電路製造技術來製作或實現成為「成品」，現階段最受歡迎的就是用半導體技術，將電路以 IC 積體電路或稱晶片的方式來製作。

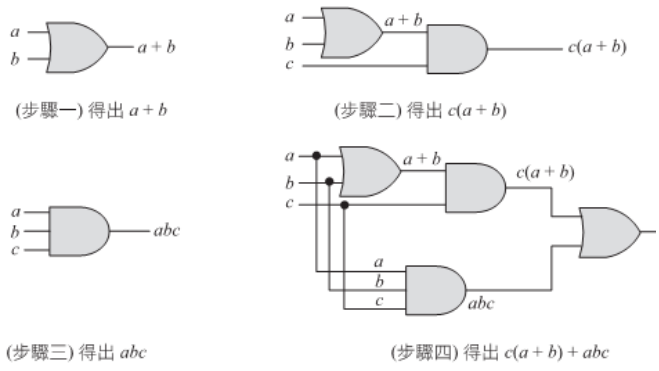


圖 4.5 導出 $F = c(a + b) + abc$ 邏輯電路圖的四個步驟

範例 4.6

已知一個布林函數 F(a, b, c) = (a + b + c)(a′ + b′ + c)(a′ + c)，試畫出其邏輯電路圖。

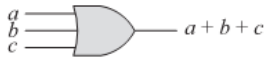
作法：

首先找出此表示式的邏輯運算順序，再依序執行之。

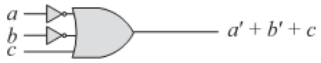
步驟一：使用一個 or 閘將輸入 a、b、c 與做運算，求得 a + b +c。步驟二：使用一個 not 閘將輸入 a 做運算，求得 a′，一個 not 閘將輸入 b 做運算，求得 b′。一個 or 閘將 a′、b′、c 做運算，求得 a′ + b′ + c。

步驟三：使用一個 or 閘將 a′ 與 c 做運算，求得 a′ + c。

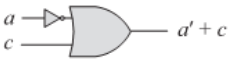
步驟四：使用一個 and 閘將步驟一得出的 a + b + c、步驟二得出的 a′ + b′ + c、步驟三得出的 a′ + c 做運算，求得 (a + b+ c)(a′ + b′ + c)(a′ + c)。



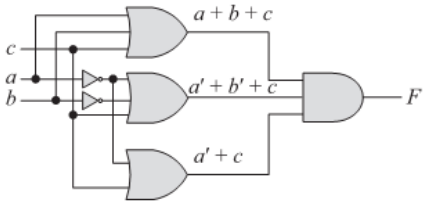
(步驟一) 得出 $a + b + c$



(步驟二) 得出 $a' + b' + c$



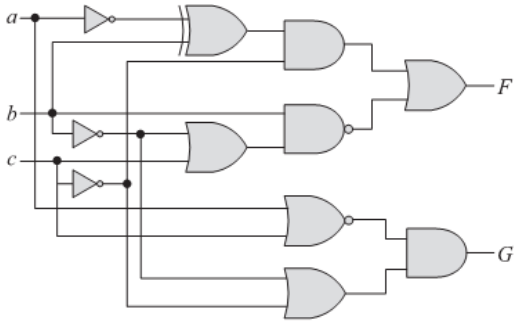
(步驟三) 得出 $a' + c$



(步驟四) 得出 $F = (a + b + c)(a' + b' + c)(a'$

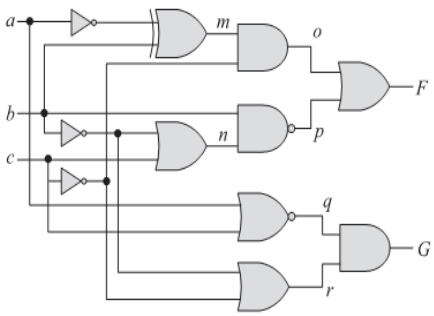
範例 4.7

已知兩布林函數 F 與 G 的邏輯電路圖如下所示，試分別寫出 F 與 G 的布林代數表示式。



作法：

先將那些電路圖上的中間值標示適當的變數符號，如下所示



依序寫出相對應的布林運算式，

$$m = (a' \oplus b)$$
$$n = b' + c$$
$$o = mc' = (a' \oplus b)c'$$
$$p = (bn)' = (b(b' + c))'$$
$$q = (a + c)'$$
$$r = b' + c'$$
$$F = o + p = (a' \oplus b)c' + (b(b' + c))'$$
$$G = qr = (a + c)'\,(b' + c')$$

範例 **4.8**

已知布林函數 $F(a, b, c) = a + b$，試畫出真值表與其邏輯電路圖。

已知布林函數 $G(a, b, c) = ac' + bc' + ac + bc$，試畫出真值表與其邏輯電路圖。請問 F 與 G 兩函數的關係為何？

作法：

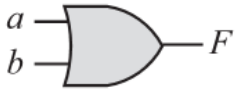
先畫出 F 的真值表，如下所示：

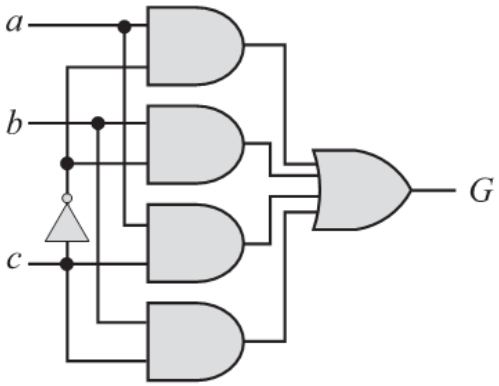
a	b	c	$F = a + b$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

- 再畫出 G 的真值表，如下所示：

a	b	c	c'	ac'	bc'	ac	bc	$G = ac' + bc' + ac + bc$
0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	1	0	1	0	1	0	0	1
0	1	1	0	0	0	0	1	1
1	0	0	1	1	0	0	0	1
1	0	1	0	0	0	1	0	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

F 與 G 的邏輯電路圖，則分別如下所示：





觀察 F 和 G 兩個布林函數的真值表，對所有的輸入組合而言，F 與 G 的輸出值都相同，則我們稱此兩個函數是「同等的」、「全等的」或「等效的」(equivalent)。既然真值表完全一樣，我們也可以視上述 F 的邏輯電路圖與 G 的邏輯電路圖，是「同等的」、「全等的」或「等效的」。假設有兩個電路，當輸入組合相同時，其輸出就會一樣，則這兩個電路可稱之為等效電路。從真值表就可證明，F 與 G 的電路會是等效電路。

4.4 布林函數的化簡 (simplification)

布林函數被用來定義數位系統輸入訊號與輸出訊號之間的關係，換句話說，布林函數可代表將設計的數位電路之全部功能。因此，若能設法將布林函數簡化到最精簡的表示，則執行此函數所需的硬體電路成本可能降到最低。前幾節提到，布林函數可用布林代數運算式來描述，因此，如果能夠設法減少布林代數運算式中，所包含「項」(term) 的總個數與每一個「項」所含的變數數目，則實現此函數行為之電路的複雜度會降低。為什麼呢？因為當電路使用較少數目的項來實現，代表所使用的邏輯閘數目會較少、電路成本較低；一個項裡面使用較少個變數來實現，則代表該邏輯閘的輸入變數數目較少、電路成本會較低 (3 個輸入 and 閘的成本高於 2 個輸入的 and 閘，閘的輸入變數愈多成本愈高)。這個減少布林函數項的數目及變數數目的過程，也可稱之為「電路的化簡」。換句話說，若能順利減少某一個布林函數所包含「項」(term) 的總數與每一個「項」的變數數目，未來將此化簡過的布林函數實現成真正硬體電路時，電路所需的成本，會低於沒有化簡過的電路。

如 4.3 節的範例 4.8 所示，雖然 $F(a, b, c) = a + b$ 與 $G(a, b, c) = ac' + bc' + ac + bc$ 兩個函數表示式不相同，但它們的真值表都一樣，所以兩者被視為等效、全等的，兩個電路是等效電路。觀察兩者的布林表示式與邏輯電路圖，很明顯地 F 的項總數較少且每一個項所含的變數也較少，故它的實現電路也較簡單。因此，如果原始布林函數表示為 G，而我們經過一連串的化簡步驟後，能將 G 簡化得出 F，則實現 F 所需的成本是遠低於 G 的，F 也可以說是 G 的簡化表示。故化簡的目的就是：將布林函數簡化到最精簡的表示式，讓此函數所需的硬體電路成本降到最低。

電路的化簡主要分為三種方式：(1) 代數化簡；(2) 圖形化簡；(3) 列表化簡，其中前兩種方式將分別在 4.4.1 與 4.4.2 節中介紹。列表化簡法雖然極具規律，但動作繁瑣且過程冗長，以手動方式來執行相當辛苦，只適合以電腦程式的方式來實現，因此本書將不介紹此化簡方法，有興趣的讀者請自行參考相關資料。

4.4.1 代數化簡

代數化簡主要是利用先前所提到的布林代數公理與定理，進行化簡動作。接下來的範例 4.9 與 4.10 可說明代數化簡的功用。範例 4.9

已知布林函數 F(a, b) = a + a′b，請進行代數化簡並說明化簡的效能。

作法：

如果不化簡，直接將原始布林代數運算式實現成邏輯電路圖，可得出圖 4.6。

需要 1 個 not 閘、1 個 and 閘、1 個 or 閘，共三個閘來實現 F(a, b)=a + a′b 函數。

當利用 4.1 節的布林代數公理與定理來進行化簡，可得

F = a + a′b = a + (a′ · b)

= (a + a′) (a + b)　　a + 對 a′b 分配 (分配律)

= 1 · (a + b)　　a + a′ = 1 (反元素)

= a + b　　1 · 任何變數其結果不變 (單位元素)

將簡化後的 F(a, b) = a + b 實現成邏輯電路圖，可得出圖 4.7。

只需要 1 個 or 閘來實現 F = a + b 函數。很顯然地，簡化後的電路成本 (邏輯閘個數) 低於原始的函數。總而言之，a + a′b 與 a + b 是同等的、等效的；即圖 4.6 與圖 4.7 兩個電路是同等的、等效的。同一組a、b 輸入，會得出相同的輸出結果，故我們選用簡化過的電路 (圖 4.7)來實現布林代數 F 以降低硬體成本。表 4.11 列出圖 4.6 與 4.7 兩個電路的真值表，從真值表也可證明兩者是同等的。

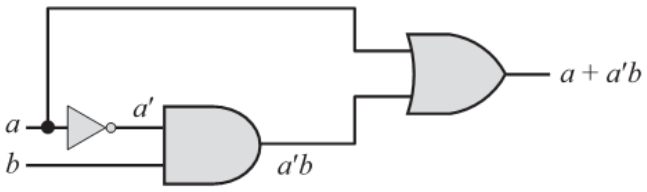


圖 4.6　$F = a + a'b$ 之邏輯電路圖

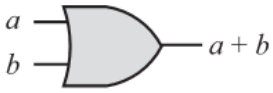


圖 4.7　$F = a + b$ 之邏輯電路圖

表 4.11　圖 4.6 與 4.7 電路的真值表

a	b	a'	$a'b$	$a + a'b$	$a + b$
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	1	0	0	1	1

範例 **4.10**

已知布林函數 $F(a, b, c) = a'b'c + a'bc + ab'$，請進行代數化簡。

作法：

利用布林代數公理與定理來進行化簡，可得

$F = a'b'c + a'bc + ab'$

$= a'cb' + a'cb + ab'$　　c、b 位置互換 (交換律)

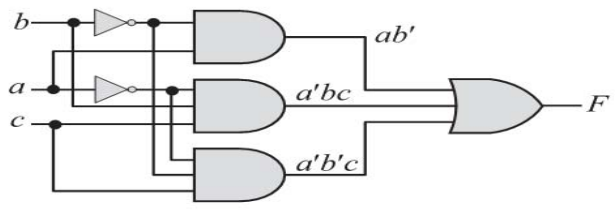
$= a'c(b' + b) + ab'$　　$a'cb' + a'cb$ 提出 $a'c$ (分配律)

$= a'c(1) + ab'$　　$b + b' = 1$ (反元素)

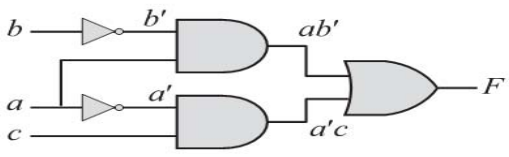
$= a'c + ab'$　　1 · 任何變數結果不變 (單位元素)

化簡前後之電路圖如下所示：

化簡前



化簡後



很顯然地，簡化後的電路成本 (邏輯閘個數) 低於原始的函數。

觀察範例 4.9，原始的布林代數表示式為 a + a′b，包含 a 與 a′b 共 2 個項。化簡後得 a + b，包含 a 與 b 共 2 個項。簡化動作雖然沒有減少所包含「項」的總個數，但是有減少「項」所含的變數數目，從 2 個變數 a′b 精簡成 1 個變數 b，其電路成本也真的會降低。

同理，觀察範例 4.10，原始的布林代數表示式為 a′b′c + a′bc + ab′，包含 a′b′c、a′bc 與 ab′ 共 3 個項。化簡後得 a′c + ab′，包含 a′c 與 ab′ 共 2 個項。簡化動作有減少所包含「項」的總個數，每一個「項」所含的變數數目也有減少，故電路成本會降低。

事實上，從上述兩個範例的推導，讀者應該可以得出這樣的結論：任何一個布林函數，應該都擁有幾個 (即一個以上) 不同的布林代數運算式。這幾個代數運算式當中，有些運算式較複雜，實現所需的電路成本較高；有些運算式較精簡，實現所需的電路成本較低。既然這些不同的電路 (代數運算式)，都是用來實現 (描述) 同一個布林函數，設計者當然要儘量挑選較為精簡的電路 (較精簡的代數運算式)，以降低製作實現所需的成本。當然，最終的目標是使用最精簡形式 (simplest form) 的運算式來實現電路，以得到成本最低的電路。

4.4.2 圖形化簡

如 4.4.1 節所述，使用代數化簡方式來簡化布林函數時，並沒有一定的規則可循，常常會因為經驗不同，而得出不一樣的結果。例如：

假設布林函數為 F(a, b, c) = abc + abc′ + ab′c，則可以化簡成 F = ab + ab′c，也可以化簡成 F = ac + abc′

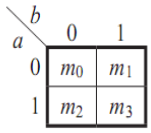
由於化簡的結果不同，其邏輯電路也不相同，故電路成本也可能會不一樣。

對於較複雜的布林函數而言，在代數化簡的過程中，即便很有經驗者也會產生不同的結果。而且因為化簡過程並沒有固定的原則可遵循，無法預知下一個步驟的可能結果，也無法確定最後得到的是不是正確答案：函數的最簡形式表示。

為了解決代數化簡的困擾，針對較複雜的布林函數，使用一種稱為「圖形化簡」的方法，是較為簡單而且直接有效的。代數化簡不容易達到很高的可靠性，但是圖形化簡則因為步驟有規律可循，有經驗者只要小心地依序執行化簡步驟，就可得出預期的結果。更重要的是，因為圖形化簡具備按部就班的步驟，所以最後可以得到該布林函數的最簡形式表示，也確保能得出最簡形式的邏輯電路圖。

a	*b*	標準積項	
0	0	$a'b'$	m_0
0	1	$a'b$	m_1
1	0	ab'	m_2
1	1	ab	m_3

(a) 標準積項



(b) 卡諾圖

圖 4.8 兩變數 (*a*、*b*) 的標準積項與卡諾圖

圖形化簡法其實是一種將真值表的圖形型式進行化簡的方法，通常也被稱為「卡諾圖化簡法」(Karnaugh map simplification) 或是「K圖化簡法」(K-map simplification)。卡諾圖被視為真值表的圖形型式，它是一個由許多正方格所構成的圖，圖中每一方格代表一個最小項或標準積項。圖 4.8 與 4.9 分別列出兩變數 (a、b) 與三變數 (a、b、c) 的標準積項與卡諾圖。

觀察圖 4.9(b)，卡諾圖規定相鄰兩個欄之間的編號只能有一個位元的改變，所以圖中的排列方式是像格雷碼的編碼順序 (00→01→11→10)，而不是一般的二進制編碼順序(00→01→10→11)。當相鄰兩個欄間的編號只有一個位元的變化時，我們就可以將任何相鄰的兩個正方格進行合併，達到化簡的目的 (減少項的總數及每一個項的變數數目)。

a	b	c	標準積項	
0	0	0	$a'b'c'$	m_0
0	0	1	$a'b'c$	m_1
0	1	0	$a'bc'$	m_2
0	1	1	$a'bc$	m_3
1	0	0	$ab'c'$	m_4
1	0	1	$ab'c$	m_5
1	1	0	abc'	m_6
1	1	1	abc	m_7

(a) 標準積項

a \ bc	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

(b) 卡諾圖

圖 4.9 三變數 (a、b、c) 的標準積項與卡諾圖

卡諾圖中的一個方格代表一個標準積項，觀察圖 4.9(b) 中任何相鄰的兩格，若它們被 or 在一起，從布林代數的原理就可以推知它們可被合併化簡。例如：m1 (a′b′c) 與 m3 (a′bc) 是相鄰的兩格，根據布林代數定理我們知道 a′b′c + a′bc = a′c(b′ + b) = a′c，故 a′b′c + a′bc 可被簡化成 a′c。m2 (a′bc′) 與 m6 (abc′) 是相鄰的兩格，我們知道 a′bc′ + abc′ = bc′(a′ + a) = bc′，故 a′bc′ + abc′ 可被簡化成 bc′。m4 (ab′c′) 與 m5 (ab′c) 是相鄰的兩格，我們知道 ab′c′ + ab′c = ab′(c′ + c) = ab′，故 ab′c′ + ab′c 可被簡化成 ab′。

此外，應該把卡諾圖想像成一個上下、左右相接之圓桶圖形，所以上下之項 (圖的最上列與最下列) 視為相鄰，左右之項 (圖的最左欄與最右欄) 亦視為相鄰。例如：圖 4.9(b) 中的 m0 與 m1 相鄰、m0 與 m4 相鄰，最重要的是 m0 與 m2 也相鄰、m4 與 m6 也相鄰。

a	b	c	標準積項	
0	0	0	$a'b'c'$	m_0
0	0	1	$a'b'c$	m_1
0	1	0	$a'bc'$	m_2
0	1	1	$a'bc$	m_3
1	0	0	$ab'c'$	m_4
1	0	1	$ab'c$	m_5
1	1	0	abc'	m_6
1	1	1	abc	m_7

(a) 標準積項

a \ bc	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

(b) 卡諾圖

圖 4.9 三變數 (a、b、c) 的標準積項與卡諾圖

4.5 積項的和之化簡

欲使用卡諾圖化簡法來求得某函數的最簡 SOP 形式，包含下述四個步驟：

步驟一： 畫出該布林函數的真值表。

步驟二： 根據真值表，畫出其卡諾圖，將輸出為 1 的標準積項那方格標示為 1，輸出為 0 的標準積項那格標示為 0。

步驟三： 將標示為 1 且相鄰的方格進行合併。合併時需注意：

a. 從相鄰方格選擇較少的方格開始進行合併，此外，愈多格合併在一起愈好 (簡化程度愈高)。合併的原則為：
相鄰 2 格合併、相鄰 4 格合併、相鄰 8 格合併，以 1→2→4→8→16 ($2n$) 格的方式合併。

b. 所有標示為 1 的標準積項方格，一定都要被包含到。

c. 有需要時，一個標示為 1 的標準積項方格可與其他方格進行多次合併 (即被包含好幾次)，以增加簡化程度。

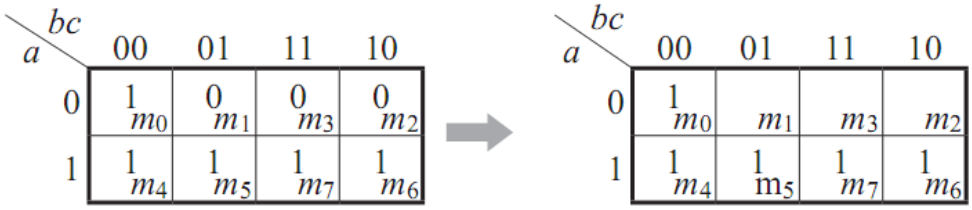
步驟四： 將卡諾圖內標示為 1 的未合併方格的代表積項與合併後的較大格之代表積項寫出來，or 在一起，可得出該函數的最簡 SOP 形式 (注意：有些題目簡化後的 **SOP** 表示會有一個以上的正確解，換句話說，可能有兩個以上的最簡 **SOP** 正確答案)。

舉例說明如下，

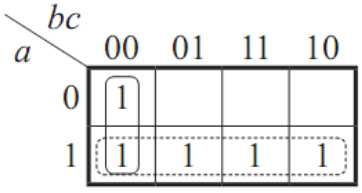
已知三變數布林函數 $F = \Sigma(0, 4, 5, 6, 7)$，使用卡諾圖化簡法來求得該函數最簡 SOP 形式。首先，執行步驟一，我們需先畫出此函數的真值表，如下所示：

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

接著執行步驟二，畫出如下卡諾圖，為了方便接下來步驟的進行，通常我們只把輸出為 1 的標準積項那方格標示為 1，其他輸出為 0 的標準積項則隱藏不標示。



針對標示為 1 的方格，執行步驟三的合併動作，從相鄰方格選擇較少的方格開始進行合併。此例中，*m*0 只有 *m*4 一個相鄰方格為 1 (一個選擇)，其他像 *m*4 則有 *m*0、*m*5、*m*6 三個相鄰的方格為 1 (三個不同選擇)，所以從 *m*0 開始進行合併。先將相鄰的 *m*0 與 *m*4 兩格合併，接著可將相鄰的 *m*4、*m*5、*m*6 與 *m*7 四格合併，如下所示：



因為合併成愈大格，其簡化程度愈高，所以我們將 $m4$ 那格與其他格合併兩次，以得出較大的合併格。如果 $m4$ 那格只使用 1 次，合併結果會較差。

最後，將合併後較大格之代表積項寫出來，or 在一起，得出答案。$m0$ 與 $m4$ 合併後，以描繪的代表積項為 $b'c'$。$m4$、$m5$、$m6$ 與 $m7$ 合併後，以描繪的代表積項為 a。故 $F = b'c' + a$，此為 F 的最簡 SOP 形式。

為了驗證上述卡諾圖化簡結果之正確性，我們可以將此範例用布林代數化簡重做一次，說明如下：

$F = \Sigma(0, 4, 5, 6, 7)$

$= m0 + m4 + m5 + m6 + m7$

$= a'b'c' + ab'c' + ab'c + abc' + abc$。

卡諾圖的化簡是將相鄰的方格合併，其中 $m4$ 有重複使用。

所以可以用布林定理 1 將 $m4$ 換為 $m4 + m4$，則函數可重寫成

$F = m0 + m4 + m5 + m6 + m7 = (m0 + m4) + (m4 + m5 + m6 + m7)$

其中

(1) $m0 + m4$

$= a'b'c' + ab'c'$ 利用分配律將 $b'c'$ 提出

$= b'c'(a' + a)$ 再利用反元素與單位元素將 $(a' + a)$ 消去

$= b'c'$

(2) $m4 + m5 + m6 + m7 = ab'c' + ab'c + abc' + abc$

$= a \cdot (b'c' + b'c + bc' + bc)$ 利用分配律將 a 提出

$= a \cdot (b' \cdot (c' + c) + b \cdot (c' + c))$ 利用分配律將 b' 與 b 提出

$= a \cdot (b' + b)$ 利用反元素與單位元素將 $(c' + c)$ 消去

$= a$ 利用反元素與單位元素將 $(b' + b)$ 消去

最後可以把函數化簡為 $F = b'c' + a$，與卡諾圖化簡法的結果相同。

從上述的卡諾圖化簡過程中，可以發現：排列卡諾圖時，讓相鄰兩欄只有一個位元改變，是為了利用分配律、單位元素與反元素等代數定理消去多餘變數，因此變數項號排列需使用格雷碼方式。另外，合併方格數目為 2^n 時，可以將 n 個變數消去。例如：合併 **m0**、**m4** 將 **a** 一個變數消去；合併 **m4**、**m5**、**m6**、**m7** 將 **b**、**c** 二個變數消去。

三變數卡諾圖的 2 格合併，總共有 12 種可能情況，圖 4.10 呈現在不同情況下的代表積項。

要寫出代表積項的口訣如下：若要合併的兩方格，某變數 x 的值一個為 0、一個為 1，則合併後此變數被消去；若要合併的兩方格，某變數 y 的值都為 0，則合併後的積項含此變數 y′；若要合併的兩方格，某變數 z 的值都為 1，則合併後的積項含此變數 z。

觀察圖 4.10 的第一種情況，要合併的兩方格 (a′b′c′ - m0 與 ab′c′ -m4)，變數 a 的值一個為 0、一個為 1，變數 b 與 c 的兩個值都為 0，故合併後變數 *a* 被消去，得代表積項 *b′c′* 項。此合併可以用布林代數化簡法證明如下：

$a'b'c' + ab'c' = b'c'(a' + a) = b'c'$。

觀察圖 4.10 的最後一種情況，要合併的兩個方格 (*ab′c′* - *m*4 與 *abc′*- *m*6)，變數 *b* 的值一個為 0、一個為 1，變數 *a* 的兩個值都為 1，變數 *c*的兩個值都為 0，故合併後變數 *b* 被消去，得代表積項 *ac′* 項。此合併可以用布林代數化簡法證明如下：

$ab'c' + abc' = ac'(b' + b) = ac'$。

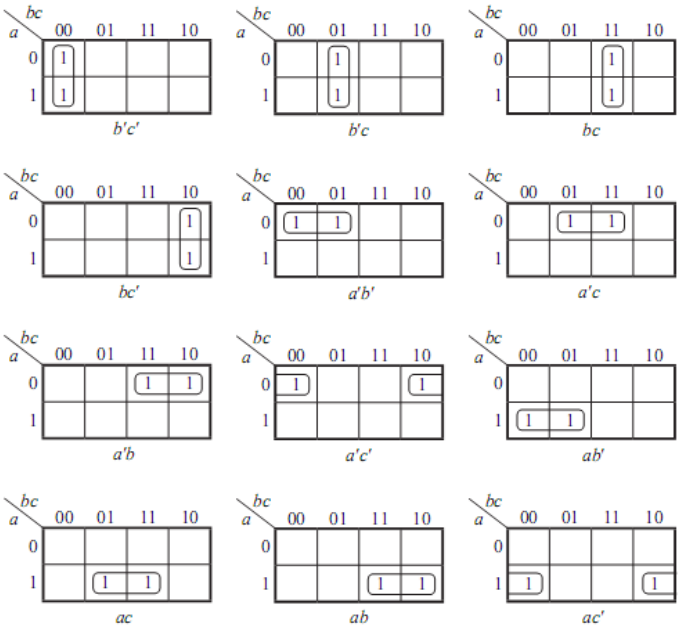


圖 4.10　三變數卡諾圖 2 格合併時，各種不同的代表積項

同理，圖 4.11 呈現三變數卡諾圖在 4 格合併時，不同情況下的代表積項。因為 4 格合併，所以整合後消去兩個變數，最後結果都只剩下單一個變數項。觀察圖 4.11 的第一種情況，要合併的四個方格(*m*0、*m*1、*m*4、*m*5)，變數 *a* 與 *c* 的值包含 0 與 1，變數 *b* 的值都為 0，故合併後變數 *a* 與 *c* 被消去，得代表積項 *b′* 項。

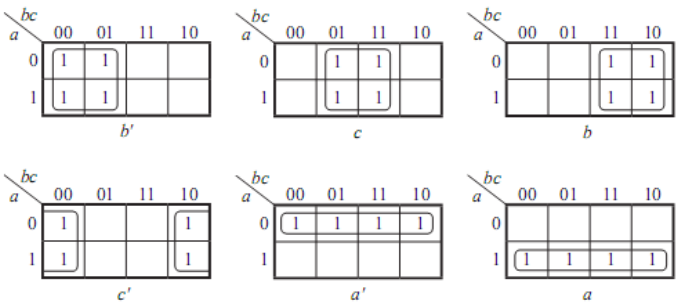


圖 4.11　三變數卡諾圖 4 格合併時，各種不同的代表積項

範例 **4.11**

已知四變數布林函數 $F(a, b, c, d) = abc'd' + a'b'c + ab'c'd + cd$，請執行卡諾圖化簡，求出該函數最簡 SOP 形式。

作法：

先畫出此函數的真值表，如下所示：

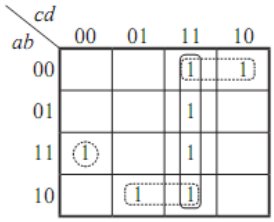
a	b	c	d	F	
0	0	0	0	0	m_0
0	0	0	1	0	m_1
0	0	1	0	1	m_2
0	0	1	1	1	m_3
0	1	0	0	0	m_4
0	1	0	1	0	m_5
0	1	1	0	0	m_6
0	1	1	1	1	m_7

a	b	c	d	F	
1	0	0	0	0	m_8
1	0	0	1	1	m_9
1	0	1	0	0	m_{10}
1	0	1	1	1	m_{11}
1	1	0	0	1	m_{12}
1	1	0	1	0	m_{13}
1	1	1	0	0	m_{14}
1	1	1	1	1	m_{15}

- 如此一來，可得出卡諾圖如下所示

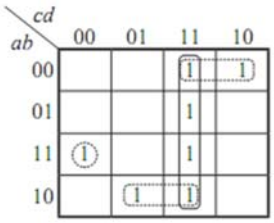
ab \ cd	00	01	11	10
00	m_0	m_1	1 m_3	1 m_2
01	m_4	m_5	1 m_7	m_6
11	1 m_{12}	m_{13}	1 m_{15}	m_{14}
10	m_8	1 m_9	1 m_{11}	m_{10}

- 選擇適當方格來合併如下圖：



針對所有標示為 1 的方格來處理，$m12$ 那格完全沒有相鄰方格無法合併，故得 $abc'd'$ 項 (沒有合併的方格項包含 4 個變數)。$m2$ 那格只有一個相鄰的方格為 1，故先處理，右上 2 格 ($m2$ 與 $m3$) 合併後得出$a'b'c$ 項 (2 個方格合併項包含 3 個變數)。$m9$ 那格也只有一個相鄰的方格為 1，故先處理，左下 2 格 ($m9$ 與 $m11$) 合併後得出 $ab'd$ 項 (2 個方格合併項包含 3 個變數)。最後，中間 4 格合併得出 cd 項 (4 個方格合併項包含 2 個變數)，故函數的最簡 SOP 表示為 $F(a, b, c, d) = abc'd' + a'b'c + ab'd + cd$。

在這個範例中原來 $F(a, b, c, d) = abc'd' + a'b'c + ab'c'd + cd$，經卡諾圖化簡後得出該函數最簡表示 $F(a, b, c, d) = abc'd' + a'b'c + ab'd + cd$，其中 $ab'c'd$ 項的變數 c' 已被簡化消去。



範例 4.12

已知三變數布林函數 F(a, b, c) = Σ(0, 2, 3, 7)，請執行卡諾圖化簡，

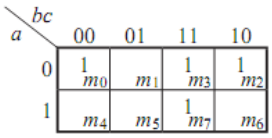
求出該函數最簡 SOP 形式。

作法：

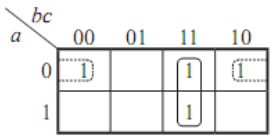
先畫出此函數的真值表，如下所示：

a	b	c	F	
0	0	0	1	m_0
0	0	1	0	m_1
0	1	0	1	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，可得出其卡諾圖如下所示



選擇適當方格來合併如下圖



$m0$ 那格只有一個相鄰的方格為 1，故先處理，與 $m2$ 合併後 得出$a'c'$ 項。$m7$ 那格也只有一個相鄰的方格為 1，先處理，與 $m3$ 合併後得出 bc 項。請注意，此時所有的 1 都已被進行合併了，所以不用再對 $m3$ 與 $m2$ 進行合併以避免造成額外項，故函數的最簡 SOP 表示為 $F(a, b, c) = a'c' + bc$。

範例 **4.13**

已知四變數布林函數 $F(a, b, c, d) = \Sigma(0, 1, 2, 4, 5, 6, 8, 10, 11)$，請執行卡諾圖化簡，求出該函數最簡 SOP 形式。

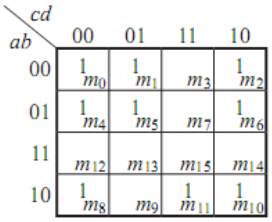
作法：

先畫出此函數的真值表，如下所示：

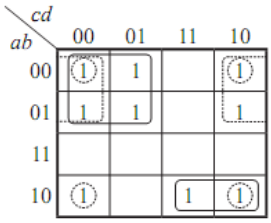
a	b	c	d	F	
0	0	0	0	1	m_0
0	0	0	1	1	m_1
0	0	1	0	1	m_2
0	0	1	1	0	m_3
0	1	0	0	1	m_4
0	1	0	1	1	m_5
0	1	1	0	1	m_6
0	1	1	1	0	m_7

a	b	c	d	F	
1	0	0	0	1	m_8
1	0	0	1	0	m_9
1	0	1	0	1	m_{10}
1	0	1	1	1	m_{11}
1	1	0	0	0	m_{12}
1	1	0	1	0	m_{13}
1	1	1	0	0	m_{14}
1	1	1	1	0	m_{15}

如此一來，可得出其卡諾圖如下所示

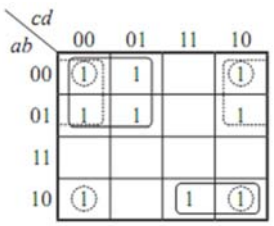


選擇適當方格來合併如下圖：



四個角落的 4 格合併後得出 $b'd'$ 項，左上 4 格合併後得出 $a'c'$ 項，左上 2 格右上 2 格共 4 格合併後得出 $a'd'$ 項，右下 2 格合併後得出 $ab'c$項，故函數的最簡 SOP 表示為 $F(a, b, c, d) = b'd' + a'c' + a'd' + ab'c$。

在這個範例中原來 $F(a, b, c, d) = \Sigma(0, 1, 2, 4, 5, 6, 8, 10, 11) = a'b'c'd' + a'b'c'd + a'b'cd' + a'bc'd' + a'bc'd + a'bcd' + ab'c'd' + ab'cd' + ab'cd$，經卡諾圖化簡後得出該函數的最簡表示 $F(a, b, c, d) = b'd' + a'c' + a'd' + ab'c$。



範例 **4.14**

已知四變數布林函數 $F(a, b, c, d) = \Sigma(0, 1, 4, 12, 13)$，請執行卡諾圖化簡，求出該函數最簡 SOP 形式。

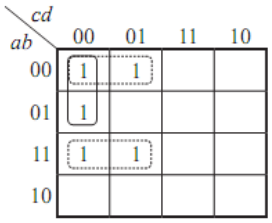
作法：

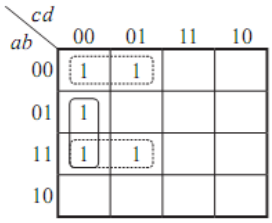
先畫出此函數的真值表，如下所示：

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

a	b	c	d	F	
1	0	0	0	0	m_8
1	0	0	1	0	m_9
1	0	1	0	0	m_{10}
1	0	1	1	0	m_{11}
1	1	0	0	1	m_{12}
1	1	0	1	1	m_{13}
1	1	1	0	0	m_{14}
1	1	1	1	0	m_{15}

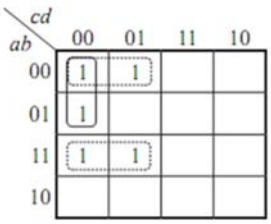
畫出其卡諾圖後，我們發現最多只有 2 相鄰方格能合併，沒有 4 格相鄰的狀況。$m1$ 那格只有一個相鄰的方格為 1，先處理，與 $m0$ 合併後得出 $a'b'c'$ 項。$m13$ 那格也只有一個相鄰的方格為 1，先處理，與 $m12$ 合併後得出 abc' 項。接下來剩下 $m4$，此時如下圖左與右，有兩種不同合併方式 (左邊 $m4$ 與 $m0$ 合併，或者如右邊 $m4$ 與 $m12$ 合併)：

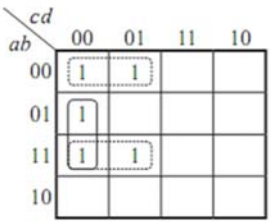




這兩種合併方式都符合卡諾圖化簡規定，結果都正確。

使用左圖合併方式可得，$F = a'b'c' + abc' + a'c'd'$。使用右圖合併方式可得，$F = a'b'c' + abc' + bc'd'$，兩者都包含 3 個積項且每個積項包含3 個變數。兩種答案都是布林函數 F 的最簡 SOP 形式，都是正確的。從這個範例中，我們可以知道，有時候使用卡諾圖來求布林函數的最簡 **SOP** 形式，會有一個以上的正確解。





五個變數的卡諾圖因為方格較多較難分析，而且由於項號的排列需保持格雷碼的排列順序不容易直接畫出，故通常用 2 個四變數的卡諾圖來組合設計，舉例說明如下：已知布林函數 $F(a, b, c, d, e) = \Sigma(0, 2, 5, 7, 8, 10, 15, 16, 18, 21, 23, 24, 26)$，先畫出其真值表。

a	b	c	d	e	F	
0	0	0	0	0	1	m_0
0	0	0	0	1	0	m_1
0	0	0	1	0	1	m_2
0	0	0	1	1	0	m_3
0	0	1	0	0	0	m_4
0	0	1	0	1	1	m_5
0	0	1	1	0	0	m_6
0	0	1	1	1	1	m_7
0	1	0	0	0	1	m_8
0	1	0	0	1	0	m_9
0	1	0	1	0	1	m_{10}
0	1	0	1	1	0	m_{11}
0	1	1	0	0	0	m_{12}
0	1	1	0	1	0	m_{13}
0	1	1	1	0	0	m_{14}
0	1	1	1	1	1	m_{15}

a	b	c	d	e	F	
1	0	0	0	0	1	m_{16}
1	0	0	0	1	0	m_{17}
1	0	0	1	0	1	m_{18}
1	0	0	1	1	0	m_{19}
1	0	1	0	0	0	m_{20}
1	0	1	0	1	1	m_{21}
1	0	1	1	0	0	m_{22}
1	0	1	1	1	1	m_{23}
1	1	0	0	0	1	m_{24}
1	1	0	0	1	0	m_{25}
1	1	0	1	0	1	m_{26}
1	1	0	1	1	0	m_{27}
1	1	1	0	0	0	m_{28}
1	1	1	0	1	0	m_{29}
1	1	1	1	0	0	m_{30}
1	1	1	1	1	0	m_{31}

將五變數卡諾圖以 a 分成 $a = 0$、$a = 1$ 兩個四變數卡諾圖，畫出如下左右兩個 4 變數卡諾圖。

$a = 0$

bc \ de	00	01	11	10
00	1 m_0	m_1	m_3	1 m_2
01	m_4	1 m_5	1 m_7	m_6
11	m_{12}	m_{13}	1 m_{15}	m_{14}
10	1 m_8	m_9	m_{11}	1 m_{10}

$a = 1$

bc \ de	00	01	11	10
00	1 m_{16}	m_{17}	m_{19}	1 m_{18}
01	m_{20}	1 m_{21}	1 m_{23}	m_{22}
11	m_{28}	m_{29}	m_{31}	m_{30}
10	1 m_{24}	m_{25}	m_{27}	1 m_{26}

對兩個 **4** 變數卡諾圖個別進行化簡，然後把兩個卡諾圖視為一上一下，相同位置也可合併，得左圖 $m7$ 與 $m15$ 可以合併得出 ***a′cde*** 項。左圖 $m5$ 與 $m7$ 兩格可以合併，右圖 $m21$ 與 $m23$ 兩格也可以合併，一上一下都在同樣位置，故合併四格得出 ***b′ce*** 項。左圖 $m0$、$m2$、$m8$、$m10$ 四格可以合併，右圖 $m16$、$m18$、$m24$、$m26$ 四格也可以合併，一上一下都在同樣位置，故合併八格，得出 ***c′e′*** 項。故最後得出 $F = a'cde + b'ce + c'e'$。

$a = 0$

bc \ de	00	01	11	10
00	(1)			(1)
01		1	1	
11			1	
10	(1)			(1)

$a = 1$

bc \ de	00	01	11	10
00	(1)			(1)
01		1	1	
11				
10	(1)			(1)

範例 **4.15**

已知五變數布林函數 $F(a, b, c, d, e) = \Sigma(1, 3, 5, 7, 8, 11, 12, 17, 19, 21, 23, 24, 27, 28)$，請執行卡諾圖化簡，求出該函數最簡 SOP 形式。

作法：

將五變數卡諾圖以 a 分成 $a = 0$、$a = 1$ 兩個四變數卡諾圖，畫出如下左右兩個 4 變數卡諾圖。

$a = 0$

bc \ de	00	01	11	10
00	m_0	1 m_1	1 m_3	m_2
01	m_4	1 m_5	1 m_7	m_6
11	1 m_{12}	m_{13}	m_{15}	m_{14}
10	1 m_8	m_9	1 m_{11}	m_{10}

$a = 1$

bc \ de	00	01	11	10
00	m_{16}	1 m_{17}	1 m_{19}	m_{18}
01	m_{20}	1 m_{21}	1 m_{23}	m_{22}
11	1 m_{28}	m_{29}	m_{31}	m_{30}
10	1 m_{24}	m_{25}	1 m_{27}	m_{26}

對兩個 **4** 變數卡諾圖個別進行化簡，然後把兩個卡諾圖視為一上一下，相同位置也可合併，得左圖 $m12$ 與 $m8$ 兩格可以合併，右圖 $m28$ 與 $m24$ 兩格也可以合併，一上一下都在同樣位置，故合併四格得出 ***bd′e′*** 項。左圖 $m11$ 與 $m3$ 兩格可以合併，右圖 $m27$ 與 $m19$ 兩格也可以合併，一上一下都在同樣位置，故合併四格得出 ***c′de*** 項。左圖 $m1$、$m3$、$m5$、$m7$ 四格可以合併，右圖 $m17$、$m19$、$m21$、$m23$ 四格也可以合併，一上一下都在同樣位置，故合併八格，得出 ***b′e*** 項。故最後得出 $F = bd'e' + c'de + b'e$。

$a = 0$

bc \ de	00	01	11	10
00		1	(1)	
01		1	1	
11	1			
10	1		(1)	

$a = 1$

bc \ de	00	01	11	10
00		1	(1)	
01		1	1	
11	1			
10	1		(1)	

4.6 和項的積之化簡

先前介紹的卡諾圖化簡方法，可以順利求出布林函數最簡的「積項的和-SOP」形式。而事實上，我們也可以使用卡諾圖化簡方法來求得布林函數最簡的「和項的積-POS」形式。接下來，我們將介紹使用卡諾圖化簡方法來求出最簡的 POS 形式的執行步驟：

步驟一： 畫出該布林函數 F 的真值表。

步驟二： 根據真值表畫出卡諾圖，將輸出為 1 的標準積項那方格標示為 1，輸出為 0 的標準積項那格標示為 0。步驟三： 將標示為 0 且相鄰的方格進行合併，求出補函數 F' 的SOP 形式。合併時需注意：

a. 從相鄰方格選擇較少的方格開始進行合併，此外愈多格合併在一起愈好 (簡化程度愈高)。合併的原則為：

相鄰 2 格合併、相鄰 4 格合併、相鄰 8 格合併，以 1→2→4→8→16 (2^n) 格的方式合併。

b. 所有標示為 0 的標準積項方格，一定都要被包含到。

c. 有需要時，一個標示為 0 的標準積項方格可與其他方格進行多次合併 (即被包含好幾次)，以增加簡化程度。

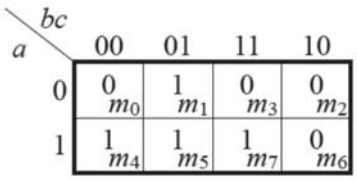
步驟四： 將卡諾圖內標示為 0 的未合併方格的代表積項與合併後的較大格之代表積項寫出來，or 在一起，可得出 F' 之最簡 SOP 形式。

步驟五： 利用迪摩根定理，將最簡 SOP 形式的 F' 再變補一次，可得出 F 的最簡 POS 形式。

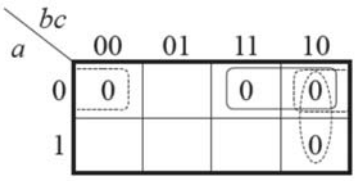
觀察上述步驟，可以知道，當要得出最簡的 POS 時，我們是先將所有標示為 0 的項進行卡諾圖化簡，如此可得出代表 F' 的最簡 SOP 形式，然後再用迪摩根定理對 F' 變補，最後即可求得 F 的最簡 POS 形式。

舉例說明如下：試求布林函數 $F(a, b, c) = \Sigma(1, 4, 5, 7)$ 的最簡 POS 形式。首先，畫出 F 的真值表，如下所示：

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



針對所有標示為 0 的項進行卡諾圖化簡，可得出 F' 的 SOP 表示，



得出 $F' = a'c' + a'b + bc'$。接著求 F' 的補函數，運算如下：

$(F')' = F = (a'c' + a'b + bc')'$

$= (a'c')' \cdot (a'b)' \cdot (bc')'$ 迪摩根定理

$= (a'' + c'')(a'' + b')(b' + c'')$ 迪摩根定理

$= (a + c)(a + b')(b' + c)$

此函數的最簡 POS 形式為

$F = (a + c)(a + b')(b' + c)$。

範例 **4.16**

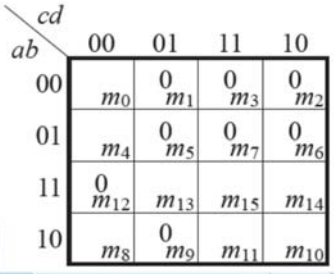
已知四變數布林函數 $F(a, b, c, d) = \Sigma(0, 4, 8, 10, 11, 13, 14, 15)$，請執行卡諾圖化簡，求出該函數最簡 POS 形式。

作法：

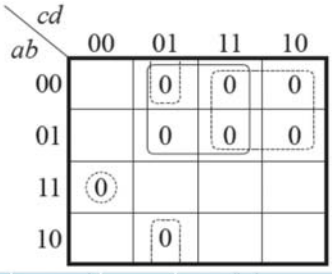
先畫出此函數的真值表，如下所示：

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

a	b	c	d	F	
1	0	0	0	1	m_8
1	0	0	1	0	m_9
1	0	1	0	1	m_{10}
1	0	1	1	1	m_{11}
1	1	0	0	0	m_{12}
1	1	0	1	1	m_{13}
1	1	1	0	1	m_{14}
1	1	1	1	1	m_{15}



畫出其卡諾圖後，針對輸出為 0 之積項執行合併：



得出 F′ = abc′d′ + b′c′d + a′d + a′c。接著求 F′ 的補函數：

(F′)′ = F = (abc′d′ + b′c′d + a′d + a′c)′

= (abc′d′)′ · (b′c′d)′ · (a′d)′ · (a′c)′ 迪摩根定理

= (a′ + b′ + c + d)(b + c + d′)(a + d′)(a + c′) 迪摩根定理

此函數的最簡 POS 形式為

F = (a′ + b′ + c + d)(b + c + d′)(a + d′)(a + c′)。

範例 **4.17**

已知四變數布林函數 $F(a, b, c, d) = \Pi(0, 1, 4, 10, 11, 12, 13, 14, 15)$，請執行卡諾圖化簡，求出該函數最簡 POS 形式。

作法：

因為 $F(a, b, c, d) = \Pi(0, 1, 4, 10, 11, 12, 13, 14, 15) = \Sigma(2, 3, 5, 6, 7, 8, 9)$，所以先畫出此函數的真值表

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

a	b	c	d	F	
1	0	0	0	1	m_8
1	0	0	1	1	m_9
1	0	1	0	0	m_{10}
1	0	1	1	0	m_{11}
1	1	0	0	0	m_{12}
1	1	0	1	0	m_{13}
1	1	1	0	0	m_{14}
1	1	1	1	0	m_{15}

畫出其卡諾圖後，針對輸出為 0 之積項執行合併 (其實就是將原先標示於 Π 內的 0、1、4、10、11、12、13、14、15 那幾項合併)：

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

ab \ cd	00	01	11	10
00	0	0		
01	0			
11	0	0	0	0
10			0	0

合併方式如圖所示，唯一沒標示合併對象的是 $m4$，因它具有兩種不同合併方式。若 $m4$ 與 $m0$ 合併，則可寫出 $F' = ab + ac + a'b'c' + a'c'd'$。

若 $m4$ 與 $m12$ 合併，則可寫出 $F' = ab + ac + a'b'c' + bc'd'$。這兩種方式都正確，最後得出

$(F')' = F = (ab + ac + a'b'c' + a'c'd')'$

$= (ab)' \cdot (ac)' \cdot (a'b'c')' \cdot (a'c'd')'$　　迪摩根定理

$= (a' + b')(a' + c')(a + b + c)(a + c + d)$　　迪摩根定理

函數的最簡 POS 形式之一為 $F = (a' + b')(a' + c')(a + b + c)(a + c + d)$。

或者是 $F' = ab + ac + a'b'c' + bc'd'$，得出

$(F')' = F = (ab + ac + a'b'c' + bc'd')'$

$= (ab)' \cdot (ac)' \cdot (a'b'c')' \cdot (bc'd')'$　　迪摩根定理

$= (a' + b')(a' + c')(a + b + c)(b' + c + d)$　　迪摩根定理

所以，另一個可能的最簡 POS 形式為

$F = (a' + b')(a' + c')(a + b + c)(b' + c + d)$。

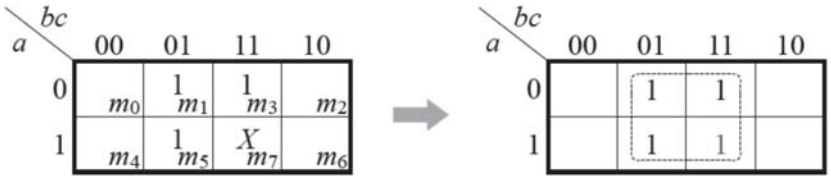
4.7 可忽略條件 (don't care condition)

在真值表或卡諾圖上標明 0 或 1，表示當輸入為該組合情況下，所得到的函數輸出值為 0 或 1。在某些特殊應用中，可能會有部分輸入組合在系統真實運作時並不會發生，既然這些輸入組合情況不會真的發生，將這些輸入組合在真值表或卡諾圖標示為輸出 0 或 1 都沒有關係，不會影響結果。這些不會發生的輸入組合又稱為可忽略項。例如：要設計一個 BCD 碼的七段顯示解碼器，假設其輸入為四位元的BCD 碼，會有十六種可能輸入組合。但因 BCD 碼只採用到前面的十種組合，所以未被採用的六種組合，對七段顯示器的解碼電路來說，可以視為「可忽略項」(don't care term)。既然這六種輸入組合情況不會真的發生，可以不必考慮這些項的 0 或 1，是否會對函數產生影響。

若將這些可忽略項妥善利用，使其在卡諾圖化簡時，依需要指定為 1或 0，有可能將函數進一步的化簡，得到更好的簡化結果。為了方便顯示，在真值表或卡諾圖上，可忽略項的輸出值被標示為 X，因為這些項不一定讓函數為 1，也不一定讓函數為 0。通常以 "d(…) = Σ(…)" 來表示所有可忽略項的和。舉例說明如下：已知布林函數 F(a, b, c) = Σ(1, 3, 5)，其中輸入組合 a = 1、b = 1 且 c = 1 不會發生，為可忽略條件。

欲求 F 的最簡 SOP 形式，方法如下：先畫出此函數的真值表，如下所示：

a	*b*	*c*	*F*	
0	0	0	0	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	1	m_5
1	1	0	0	m_6
1	1	1	*X*	m_7



所畫出卡諾圖如下左圖所示，很明顯地，若將可忽略項 *m*7 視為 1，則可合併的方格數會由 2 格變成 4 格。因此為了更好的化簡結果，我們將 *m*7 視為 1，如下圖所示。

故此函數的最簡 SOP 形式可表示為 *F*(*a*, *b*, *c*) = *c*。

範例 4.18

已知四變數布林函數 F(a, b, c, d) = Σ(0, 1, 2, 5, 8, 10) 其可忽略條件為 d(a, b, c, d) = Σ(4, 6)，請執行卡諾圖化簡，求出該函數最簡 SOP形式，並畫出其邏輯電路圖。

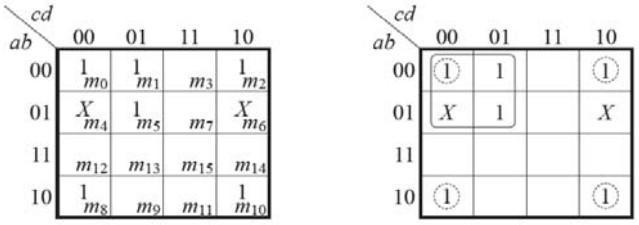
作法：

先畫出此函數的真值表，如下所示：

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

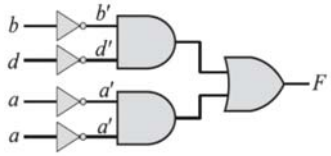
a	b	c	d	F	
1	0	0	0	1	m_8
1	0	0	1	0	m_9
1	0	1	0	1	m_{10}
1	0	1	1	0	m_{11}
1	1	0	0	0	m_{12}
1	1	0	1	0	m_{13}
1	1	1	0	0	m_{14}
1	1	1	1	0	m_{15}

如此一來，可得出其卡諾圖如下之左圖，選擇適當方格來合併成如下圖：



四個角落的 4 格 合併後得出 $b'd'$ 項。把 $m4$ 視為 1，左上 4 格 合併後得出 $a'c'$。$m6$ 不需使用 (即把它視為 0)，故最簡 SOP 表示為 $F(a, b, c, d) = b'd' + a'c'$。

其邏輯電路圖如下所示：



範例 **4.19**

已知四變數布林函數 F(a, b, c, d) = Σ(0, 3, 4, 6, 7) 其可忽略條件為d(a, b, c, d) = Σ(2, 14)，請執行卡諾圖化簡，求出該函數最簡 POS 形式，並畫出其邏輯電路圖。

作法：

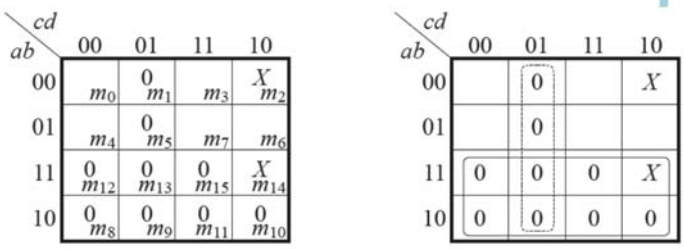
注意，此題是求出 POS。

先畫出此函數的真值表，如下所示：

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

a	b	c	d	F	
1	0	0	0	0	m_8
1	0	0	1	0	m_9
1	0	1	0	0	m_{10}
1	0	1	1	0	m_{11}
1	1	0	0	0	m_{12}
1	1	0	1	0	m_{13}
1	1	1	0	X	m_{14}
1	1	1	1	0	m_{15}

列出所有標示為 0 的方格，可得如下之左圖，選擇適當方格來合併求出 F' 如下之右圖：

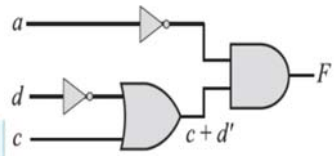


把 $m14$ 視為 0，下面 8 格合併後得出 a。$m2$ 不需使用 (即把它視為 1)，可得 $F' = a + c'd$。求其補函數可得

$(F')' = F = (a + c'd)' = a'(c + d')$

故最簡 POS 表示為 $F = a'(c + d')$。

其邏輯電路圖如下所示：



4.8 nand 閘與 nor 閘電路

- 為了設計出較低複雜度的電路，可使用前述的卡諾圖化簡法，來求出某個函數的最簡 SOP 形式或是最簡 POS 形式。
- 但是如 4.3 節的表 4.9 所示，當使用 CMOS 的 IC 製造技術來實現電路時，nand 閘與nor 閘的閘傳輸延遲只有 and 閘與 or 閘的一半，而事實上其電路實作成本也大約只有一半。
- 換句話說，和 and 閘與 or 閘比較起來，nand 閘與 nor 閘的製作成本低、執行速度也較快。因此，為了降低 IC 製造成本，增快 IC 工作速度，會將電路轉換成以 nand 或 nor 閘來實現。
- 本節主要就是介紹，如何將前述最簡的 POS 電路或 SOP 電路，轉換成等效、同等的 nand 或 nor 閘實現的電路。

4.8.1 **nand** 閘電路之實現

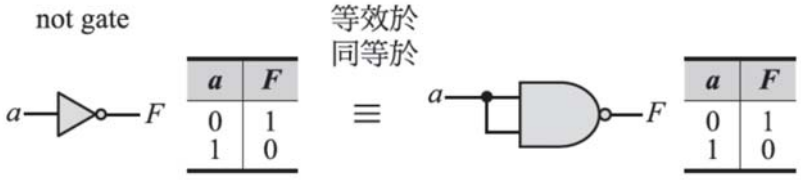
反及閘 nand 是一種所謂的通用閘 **(universal gate)**，代表任何電路系統都可以只使用反及閘，而不需用到其他的閘來實作得出。為什麼呢？因為一個電路中最基本的邏輯閘是 and、or 與 not 閘。這三種閘都可以使用 nand 來實作得知，說明如下：

反閘 (not)

反閘的圖示與真值表如下左圖所示。

若將 nand 閘的兩個輸入接在一起 (代表 nand 的兩個輸入會同時為 0 或同時為 1)，圖示與真值表如下右圖所示。

觀察下圖左右兩者的真值表一模一樣，故此兩個電路可視為是同等的、等效的。



a	F
0	1
1	0

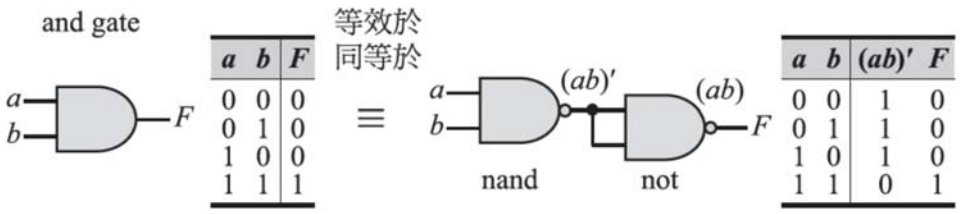
a	F
0	1
1	0

及閘 (and)

及閘的圖示與真值表如下左圖所示。

nand 的補數為 and， 所以要以 nand 取代 and 閘，只需要額外使用一個 not 閘即可達成，其圖示與真值表如下右圖所示。

觀察下圖左右兩者的真值表一模一樣，故此兩個電路可視為是同等等效的。



a	b	F
0	0	0
0	1	0
1	0	0
1	1	1

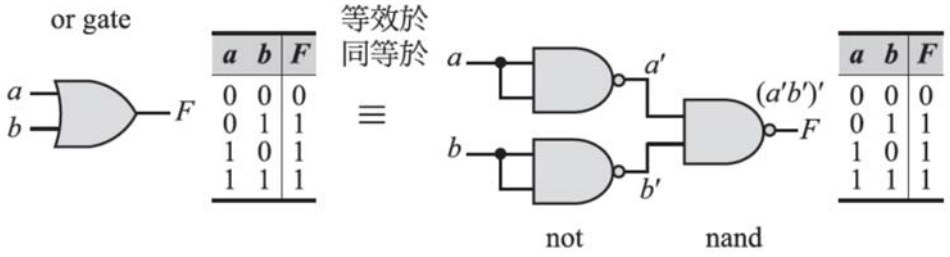
a	b	(ab)′	F
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

或閘 (or)

或閘的圖示與真值表如下左圖所示。

根據迪摩根定理我們知道，$(a'b')' = a'' + b'' = a + b$，所以要以 nand 取代 or 閘，可使用如下右圖的電路。

觀察下圖左右兩者的真值表一模一樣，故此兩個電路可視為是同等等效的。

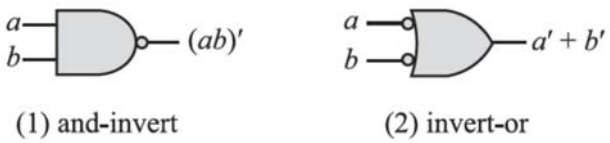


a	b	F
0	0	0
0	1	1
1	0	1
1	1	1

a	b	F
0	0	0
0	1	1
1	0	1
1	1	1

此外，為了進行 nand 閘的轉換，需重新定義二個等效圖示符號，如下圖所示。

這兩個圖示都代表 nand 閘，因為根據迪摩根定理，我們可知 $(ab)' = a' + b'$，故左右兩圖示可視為同等或等效。



(1) and-invert　　(2) invert-or

and-invert 是一般 nand 閘所常用的圖示，將 and 的輸出取補數變成nand。另一種 invert-or 是先將輸入取補數，再進入 or 閘 (符合迪摩根定理對 nand 運算的定義)，圖中的小圓代表取補數 (not)。無論是 andinvert或 invert-or 二者都可以代表 nand 閘。

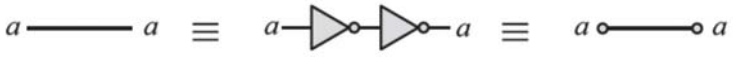
假設要將一個積項的和 SOP 形式表示的布林函數電路，轉換成全部用 nand 閘實現的電路以達到降低成本、增快速度的目的，其轉換步驟如下所示：

(1) 畫出 SOP (two-level and-or) 形式的電路圖。

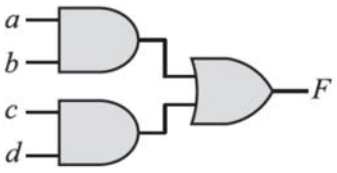
(2) and 閘與 or 閘之間，利用自補定理在同一條線兩端取補數。

(3) 分別將 and-invert 與 invert-or 轉換為 nand 閘。

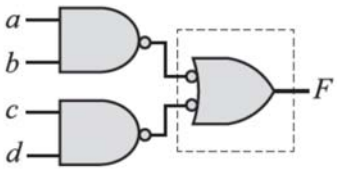
所謂自補定理乃是指：在一條線上，如果兩邊都取補數 (not 兩次)，結果不變，如下圖所示 (小圓代表 not)：



接下來，我們舉例說明，將一個 SOP 形式的布林函數電路 (下圖(1))，應用自補定理轉成下圖 (2)，最後全部轉換成以 nand 閘 (下圖 (3))，實現的步驟：

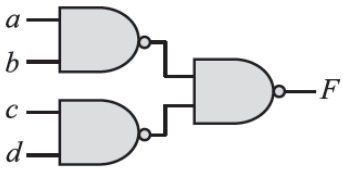


(1) SOP 電路



(2) 應用自補原理結果

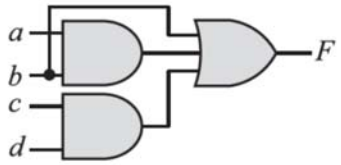
圖 (1)(2)(3) 三個電路都是等效的。



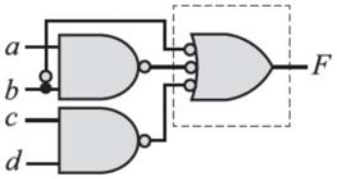
(3) nand 閘實現電路

如果 SOP 形式的電路如下圖 (1)，應用自補定理轉成下圖 (2)，最後則可能有如下圖 (3) 與 (4) 兩種表示方式：

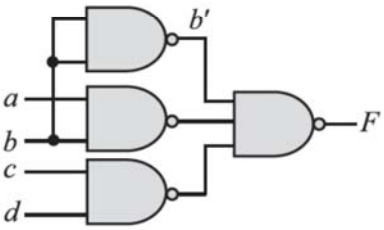
有時候為了方便表示，在畫電路圖時，除了原始輸入 a、b、c、d 外，a′、b′、c′、d′ 等四個輸入補數也可以直接使用，故圖 (4) 省略那一個將 b 變補數的反閘，不需在圖上標示。



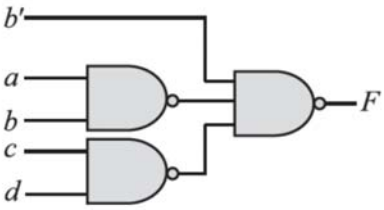
(1) SOP 電路



(2) 應用自補原理結果

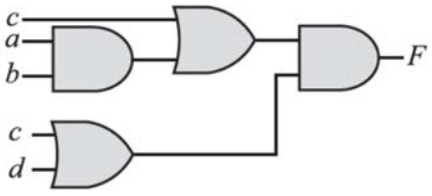


(3) nand 閘實現電路-1

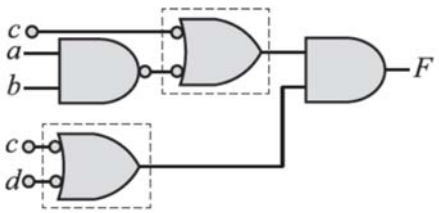


(4) nand 閘實現電路-2

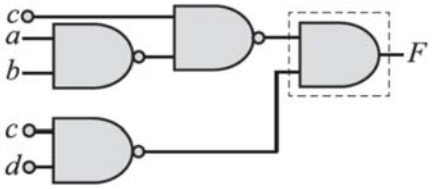
- 布林函數的標準形式可能為二階 SOP 或 POS 電路，但有時在數位設計上會使用三階以上的電路，其電路都是由 and、or、not 組合而成。
- 多層級 (multi-level) 的 and-or 形式，也可使用相同步驟轉換成以nand 閘實現。
- 三層的電路，執行動作如下圖步驟 (1)(2)(3)(4) 所示：



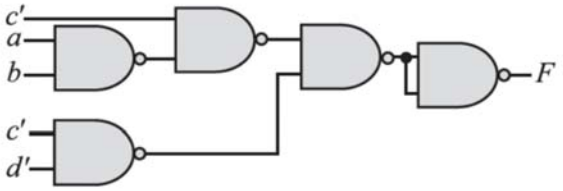
(1) 三層 and-or 電路



(2) 應用自補原理結果

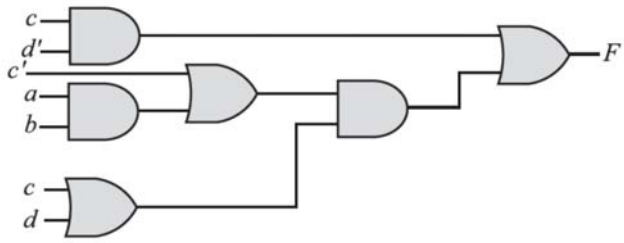


(3) 中間過程電路

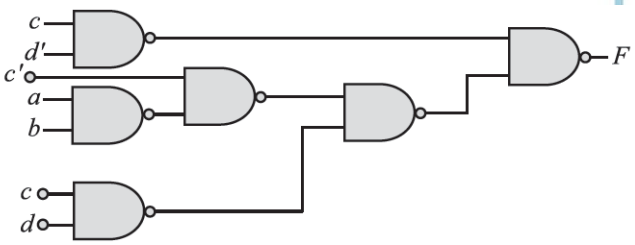


(4) nand 閘實現電路

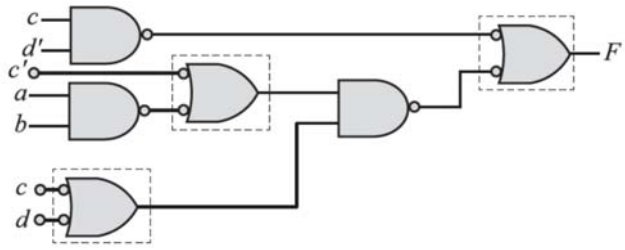
四層 and-or 的電路，執行動作如下圖步驟 (1)(2)(3)(4) 所示：



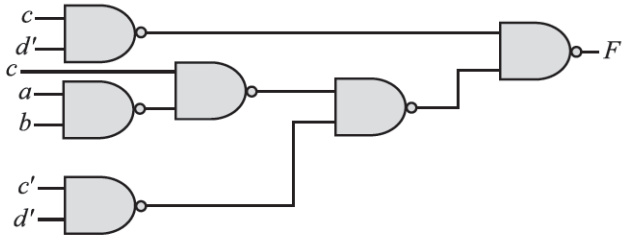
(1) 四層 and-or 電路



(3) 中間過程電路



(2) 應用自補原理結果



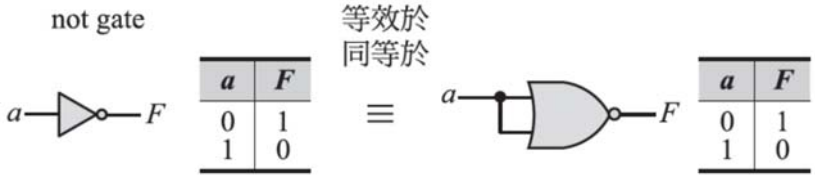
(4) nand 閘實現電路

4.8.2 nor 閘電路之實現

和前面提及的 nand 閘一樣，反或閘 nor 也是一種所謂的通用閘(universal gate)，任何電路系統都可以只使用反或閘，而不需用到其他閘來實作得出。我們首先證明電路中最基本的邏輯閘 and、or 與 not 閘等，都可以使用 nor 來實作得知，說明如下：

反閘 (not)

反閘的圖示與真值表如下左圖所示。
若將 nor 閘的兩個輸入接在一起 (代表 nor 的兩個輸入會同時為 0 或同時為 1)，圖示與真值表如下右圖所示。
觀察下圖左右兩者的真值表一模一樣，故此兩個電路可視為是同等的、等效的。

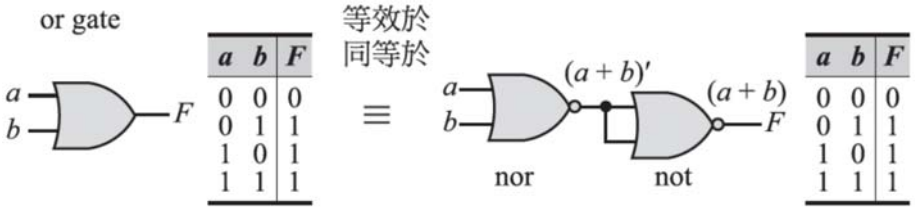


a	F
0	1
1	0

a	F
0	1
1	0

或閘 (or)

或閘的圖示與真值表如下左圖所示。
nor 的補數為 or，所以要以nor 取代 or 閘，只需要額外使用一個 not 閘即可達成，其圖示與真值表如下右圖所示。
觀察下圖左右兩者的真值表一模一樣，故此兩個電路可視為是同等等效的。

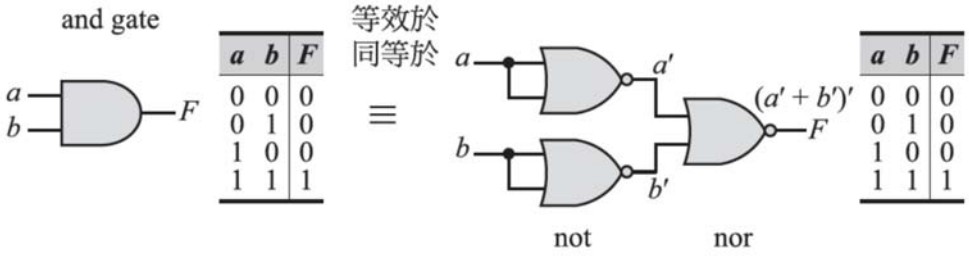


a	b	F
0	0	0
0	1	1
1	0	1
1	1	1

a	b	F
0	0	0
0	1	1
1	0	1
1	1	1

及閘 (and)

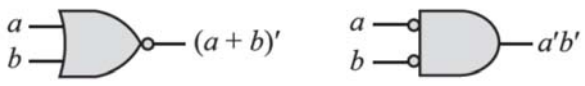
及閘的圖示與真值表如下左圖所示。
根據迪摩根定理我們知道，$(a' + b')' = a'' \cdot b'' = ab$，所以要以 nor 取代 and 閘，可使用如下右圖的電路。
觀察下圖左右兩者的真值表一模一樣，故此兩個電路可視為是同等等效的。



a	b	F
0	0	0
0	1	0
1	0	0
1	1	1

a	b	F
0	0	0
0	1	0
1	0	0
1	1	1

此外，為了進行 nor 閘的轉換，需重新定義二個等效圖示符號，如下圖所示。這兩個圖示都代表 nor 閘，根據迪摩根定理我們可知，$(a + b)' = a'b'$，故左右兩圖示可視為同等或等效。



(1) or-invert　　(2) invert-and

or-invert 是一般 nor 閘所常用的圖示，將 or 的輸出取補數變成nor。另一種 invert-and 是先將輸入取補數，再進入 and 閘 (符合迪摩根定理對 nor 運算的定義)。無論是 or-invert 或 invert-and 二者都可以代表nor 閘。

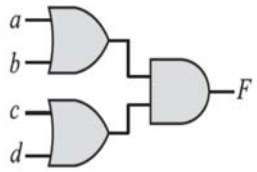
如前所述，一個積項的和 SOP 形式表示的布林函數電路，可轉換成全部以 nand 閘實現的電路。同理，一個和項的積 POS 形式表示的林函數電路，可轉換成全部以 nor 閘實現的電路以達到降低成本、增快速度的目的，其轉換步驟如下所示：

(1) 畫出 POS (two-level or-and) 形式的電路圖。

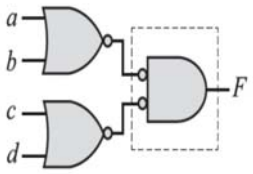
(2) or 閘與 and 閘之間，利用自補定理在同一條線兩端取補數。

(3) 分別將 or-invert 與 invert-and 轉換為 nor 閘。

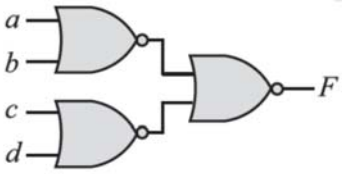
接下來，我們舉例說明，將一個 POS 形式的布林函數電路 (下圖(1))，應用自補定理轉成下圖 (2)，最後全部轉換成以 nor 閘 (下圖(3))實現的步驟：



(1) POS 電路



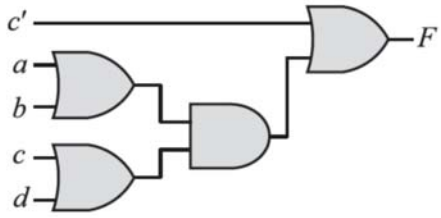
(2) 應用自補原理結果



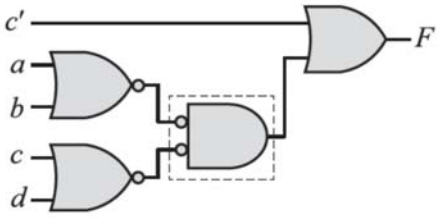
(3) nor 閘實現電路

圖 (1)(2)(3) 三個電路是等效的。

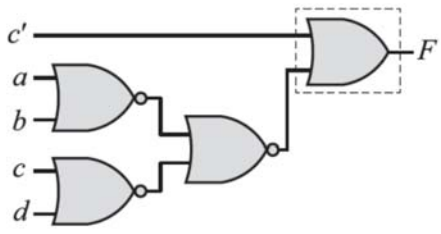
多層級 (multi-level) 的 or-and 形式，也可使用相同步驟轉換成以 nor 閘實現。三層的電路，執行動作如下圖步驟 (1)(2)(3)(4) 所示：



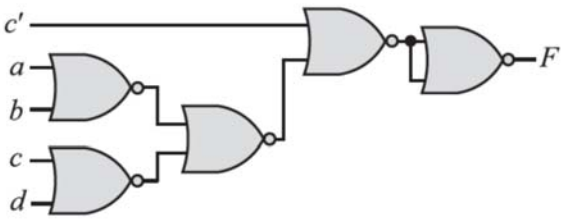
(1) 三層 or-and 電路



(2) 應用自補原理結果

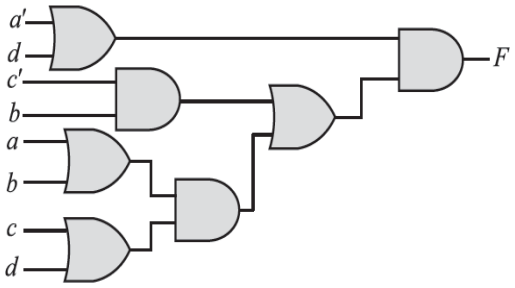


(3) 中間過程電路

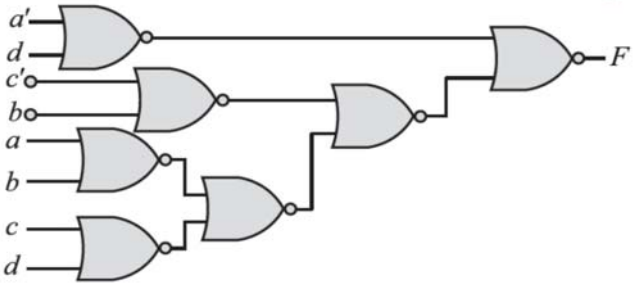


(4) nand 閘實現電路

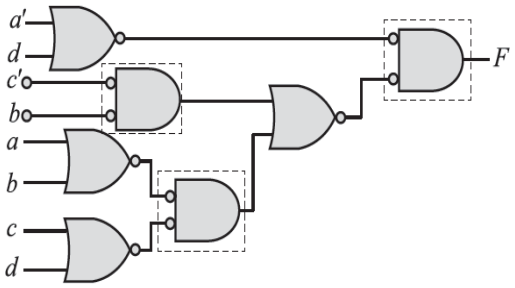
四層 or-and 的電路，執行動作如下圖步驟 (1)(2)(3)(4) 所示：



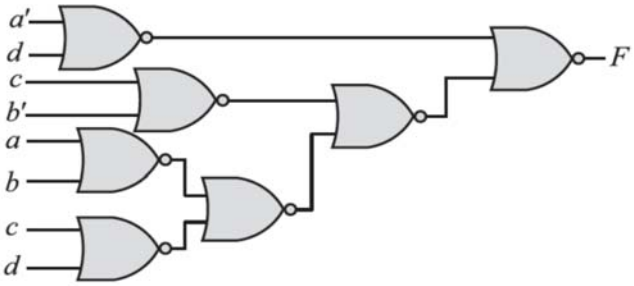
(1) 四層 or-and 電路



(3) 中間過程電路



(2) 應用自補原理結果



(4) nand 閘實現電路

4.9 邏輯電路之設計與實現

綜合本章前面幾節的介紹，邏輯電路的設計實現主要包含下列六個步驟：

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。

舉例說明如下：已知有一布林函數為 F(a, b, c, d) = Σ(0, 2, 3, 7, 8, 9,10, 11, 14, 15)，要如何設計出該函數的邏輯電路呢？

首先，因為這個題目有提供布林函數表示式，所以輸入與輸出之間的關係很明確，故可以直接畫出其真值表，如下所示：

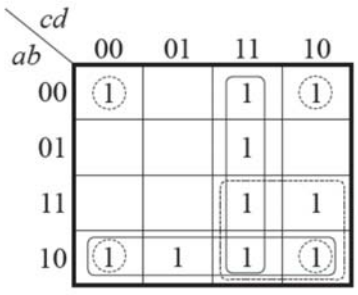
a	b	c	d	F	
0	0	0	0	1	m_0
0	0	0	1	0	m_1
0	0	1	0	1	m_2
0	0	1	1	1	m_3
0	1	0	0	0	m_4
0	1	0	1	0	m_5
0	1	1	0	0	m_6
0	1	1	1	1	m_7

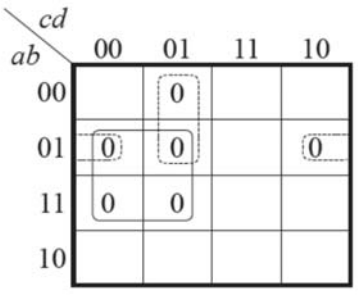
a	b	c	d	F	
1	0	0	0	1	m_8
1	0	0	1	1	m_9
1	0	1	0	1	m_{10}
1	0	1	1	1	m_{11}
1	1	0	0	0	m_{12}
1	1	0	1	0	m_{13}
1	1	1	0	1	m_{14}
1	1	1	1	1	m_{15}

可得如下卡諾圖：

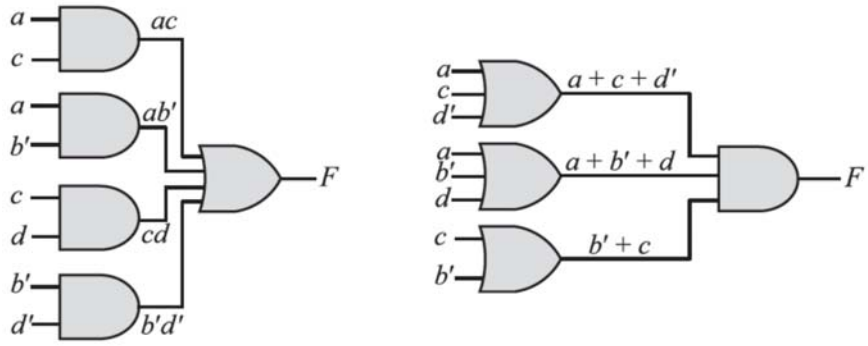
ab \ cd	00	01	11	10
00	1 m_0	0 m_1	1 m_3	1 m_2
01	0 m_4	0 m_5	1 m_7	0 m_6
11	0 m_{12}	0 m_{13}	1 m_{15}	1 m_{14}
10	1 m_8	1 m_9	1 m_{11}	1 m_{10}

進行卡諾圖化簡，下左圖是針對積項的和化簡，下右圖是針對和項的積化簡

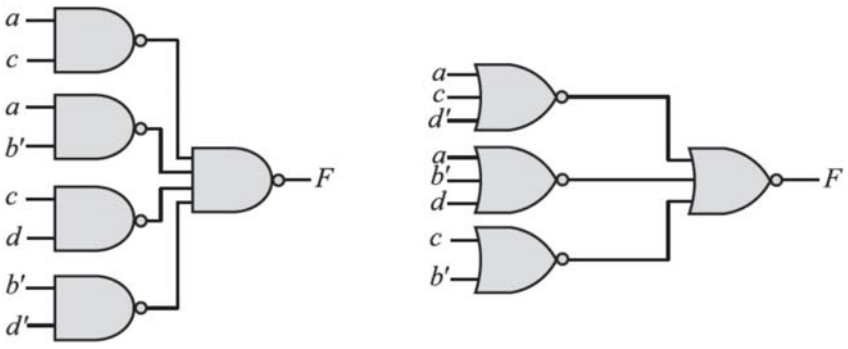




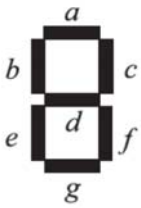
針對 SOP 形式化簡後可得，F = b′d′ + cd + ac + ab′，
針對 POS 形式化簡後可得，F′ = bc′ + a′c′d + a′bd′，
故 (F′)′ = F = (bc′ + a′c′d + a′bd′)′ = (bc′)′(a′c′d)′(a′bd′)′
= (b′ + c)(a + c +d′)(a + b′ + d)。
下圖左是 SOP 形式的邏輯電路圖，下圖右則是 POS 形式的邏輯電路圖。



為降低硬體實作成本，在 SOP 形式中的 and 閘與 or 閘之間，利用自補定理在同一條線兩端取補數，可得下圖左的 nand 閘實現電路。在POS 形式中的 or 閘與 and 閘之間，利用自補定理在同一條線兩端取補數，可得下圖右的 nor 閘實現電路，



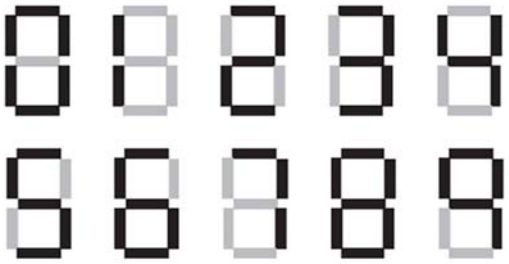
順利得出布林函數 F 的 nand 閘實現電路與 nor 閘實現電路。



範例 4.21

BCD (binary coded decimal) 是一種表示十進數的二元數值系統。BCD 系統使用四個位元的二進碼來表示 0～9 等 10 個十進數，因為四位元的二進數從 0000～1111 共有 16 個，而單位數的十進數從 0～9只有 10 個，所以最後 6 個二進數 (1010、1011、1100、1101、1110 與1111) 在 BCD 系統中不具任何意義。

如左圖所示的七段顯示器是一種很常見的輸出顯示裝置，它包含標示為 a、b、c、d、e、f、g，七個不同的 LED 燈。如果訊號為 1時，LED 發光，訊號為 0 時，LED 熄滅。下圖列出當要顯示數字 0～9 時，所需要的設定方式：



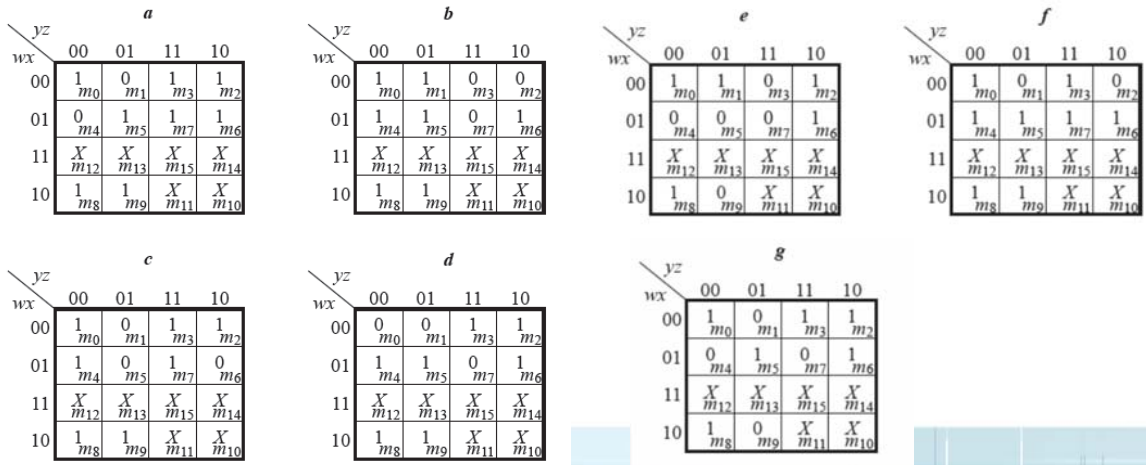
換句話說，要顯示 0 時，需讓 a、b、c、e、f、g 都為 1，只有 d 為 0；要顯示 1 時，讓 b 與 e 為 1，其他值都為 0；要顯示 7 時，讓 a、c、f為 1，其他值都為 0。試設計一個 BCD 碼對七段顯示器的解碼電路，每輸入一個 4 位元的 BCD 碼時，可藉由七段顯示裝置呈現出相對應的數字。此外，整個解碼器電路請全部使用 nand 閘來實現。

作法：

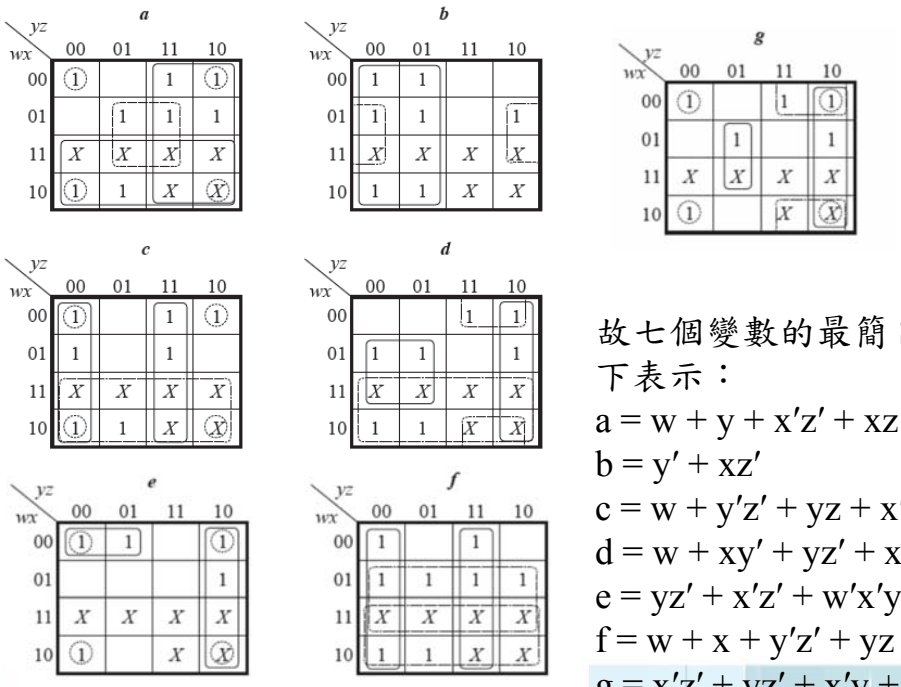
首先，觀察此題目，輸入包含四個變數用以表示 BCD 碼的 0000～1001，輸出則包含七個變數，一個變數控制七段顯示器中一個 LED 燈的亮或暗。假設四個輸入變數標示為 w、x、y、z，七個輸出變數則分別標示為 a、b、c、d、e、f、g，其中輸出變數的標示如題目的附圖所示，可畫出如下四輸入、七輸出的真值表：

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

當輸入 wxyz = 0000 時，為了要顯示 0，輸出 a、b、c、e、f、g 都需為1，只有 d 為 0；當輸入 wxyz = 0011 時，為了要顯示 3，輸出 a、c、d、f、g 都需為 1，只有 b 與 e 為 0；當輸入 wxyz = 0110 時，為了要顯示 6，輸出 a、b、d、e、f、g 都需為 1，只有 c 為 0。因為輸入為 BCD碼，所以 wxyz = 1010、1011、1100、1101、1110、1111 等這六種輸入狀況不可能發生，視為可忽略項，在真值表輸出處標示為 X。針對每一個輸出變數畫出其卡諾圖：



進行卡諾圖化簡



故七個變數的最簡 SOP 形式可寫成如下表示：

a = w + y + x′z′ + xz

b = y′ + xz′

c = w + y′z′ + yz + x′z′

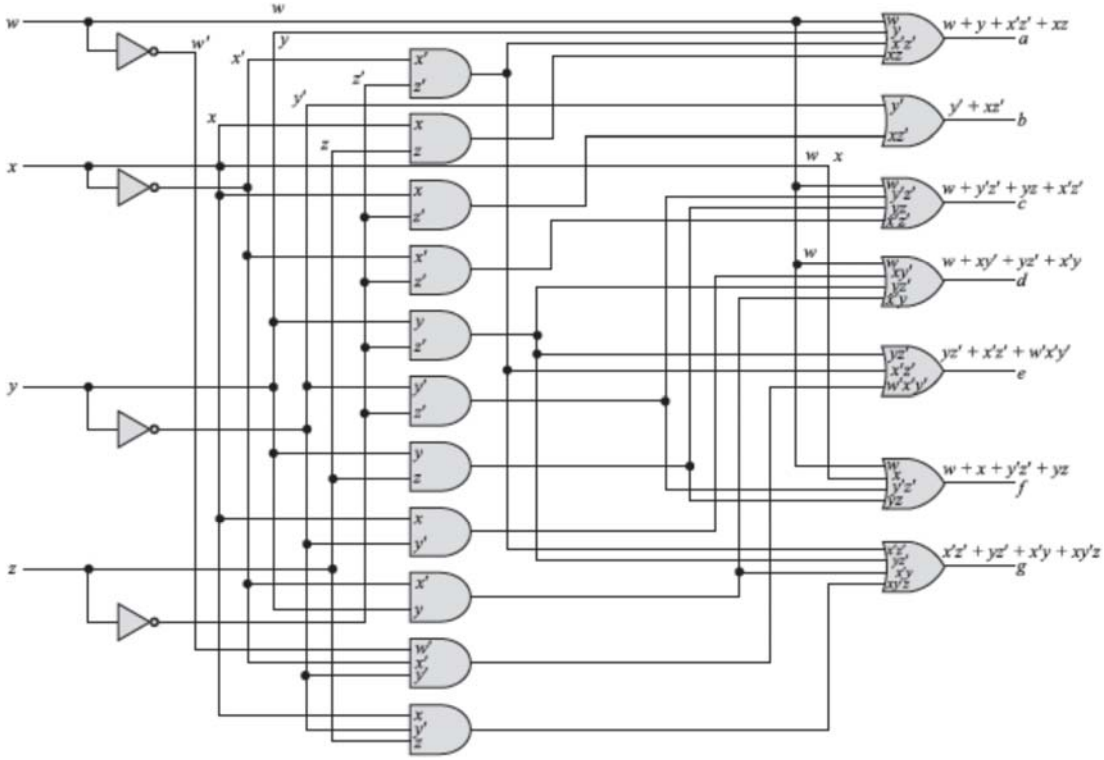
d = w + xy′ + yz′ + x′y

e = yz′ + x′z′ + w′x′y′

f = w + x + y′z′ + yz

g = x′z′ + yz′ + x′y + xy′z

可得出如下的邏輯電路圖



最後得出如下的 nand 閘實現電路圖。