第3章

數位邏輯元件

滄海書局
Tsang Hai Book Publishing Co.

- 3.1 數位系統
- 3.2 基本數位邏輯元件
  - 3.2.1 及閘
  - 3.2.2 或閘
  - 3.2.3 反閘
  - 3.2.4 互斥或閘
  - 3.2.5 反及閘
  - 3.2.6 反或閘
  - 3.2.7 反互斥或閘
  - 3.2.8 緩衝閘
- 3.3 數位系統之設計與實作

Copyright©滄海書局

3.1 數位系統

前面曾提及，由於硬體電路實作的限制，數位系統 (或稱數位電路) 一般使用二進制來表示所有的訊號，也就是說輸入/ 輸出只有 “0” 與 “1” 兩種值。市面上常用的 TTL 邏輯電路，將 0 伏特到 0.8 伏特之間的電壓定義為「邏輯 0」；而 2 伏特至 5 伏特之間的電壓定義為「邏輯 1」；這兩個範圍之間的電壓 (0.8 伏特～2 伏特) 則當作「間隔電壓」或「禁區電壓」。如果一個 TTL 電路中，某點的電壓值為 4 伏特，則表示該點為邏輯 1；如果某點的電壓值為 0.5 伏特，則表示該點為邏輯 0；如果某點的電壓值為 1.5 伏特，則是一個錯誤的訊號。在標準的數位電路中，由於電路上的訊號只可能是 “0” 或 “1”，所以元件的輸入端訊號與輸出端訊號，在任何時刻應只有兩種可能數值：0 或 1。

Copyright©滄海書局

圖 3.1 是一個 TTL 電路可能的輸出電壓波形，其中「邏輯 1」的理想電壓是 +5 V，又稱為「高電位」訊號或「高準位」訊號，如圖中線段 *BC* 與 *FG*；「邏輯 0」的理想電壓是 0 V，又稱為「低電位」訊號或「低準位」訊號，如圖中線段 *DE* 與 *HI*。當由低電位變成高電位稱為訊號的「正緣」或「上升緣」(positive edge or rising edge)，如圖中線段 *AB* 與 *EF*；而由高電位變成低電位則稱為訊號的「負緣」或「下降緣」(negative edge or falling edge)，如圖中線段 *CD* 與 *GH*。

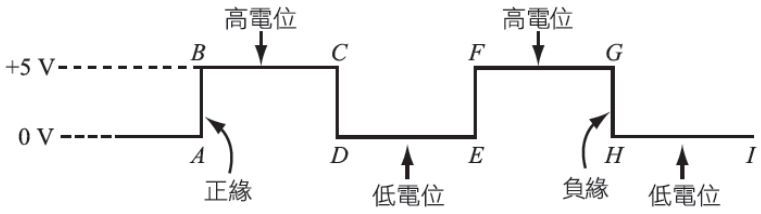


圖 3.1　一個 TTL 電路可能的輸出電壓波形

Copyright©滄海書局

要注意的是，由於數位系統使用一連串的 0 或 1 的位元 (bit) 來表示並儲存輸入/ 輸出訊號，所以系統內的運算處理過程也應以二進制 0 或 1 的方式進行。系統設計者需要決定所使用的 0 或 1 代表什麼資訊、如何解釋、如何透過輸入裝置將輸入訊號輸入系統中 (例如：按鍵被按住表示輸入 1，按鍵被放開表示輸入 0)、系統如何執行所需 0 與 1 的計算、如何將結果展示於輸出裝置上 (例如：輸出 0101 時，螢幕秀出數字 5；輸出 1001 時，螢幕秀出數字 9。或是輸出 0 表示某個 LED 燈要點亮，輸出 1 表示某個 LED 燈要熄滅)。換句話說，如圖 3.2 所示，所謂的數位系統設計就是設計一個電路系統，將輸入的一連串 **0** 或 **1** 的位元串流訊號，按照所規劃的動作進行運算，最後產生另一串所需要的 **0** 或 **1** 位元串流輸出訊號。上述的說明，算得上是數位系統的巨觀想法，如圖 3.2，此數位電路需設計成在輸入訊號為 10 時，會產生 010101 的輸出。

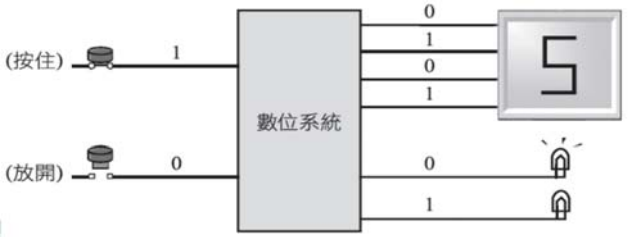


圖 3.2 一個數位系統可能的輸入與輸出方式

Copyright©滄海書局

本書主要是介紹數位電路設計所需的相關入門知識，期望能讓讀者了解：如何針對輸入的 0 與 1 訊號，設計一個數位電路/系統，進行運算以產生此系統所需的輸出 0 與 1 訊號。

Copyright©滄海書局

3.2 基本數位邏輯元件

無論是一個簡單的或者是複雜的數位電路系統，究其根本都是由基本邏輯元件，也就是所謂的邏輯閘 (logic gate)，所組合而成。基本的邏輯元件相對於數位系統，就如同各種細胞相對於生物一樣，不論是何種生物都是由許多不同類的細胞組織而成，而每一個數位系統都是由許多不同類的邏輯閘組合而成。不論是簡單的數位電路系統如「多工/選擇器」或較複雜的數位系統如「個人電腦」，它們都是由基本邏輯閘元件組合而成的。較簡單的數位電路可能由數個到數百個邏輯閘元件連接組合而成；較複雜的數位電路則可能由數萬個、數十萬個到數百萬個邏輯閘元件連接組合而成。每一個基本邏輯閘，會將該閘的幾個輸入數位訊號 (0 或 1) 適當處理後，產生此類閘的數位輸出訊號 (0 或 1)；多個邏輯閘連接而成的數位電路系統，則可將該系統的數位輸入訊號 (0 或 1 的位元串流) 適當處理後，產生系統所需的輸出訊號 (0 或 1 的位元串流)，見圖 3.2。在接下來的章節中，我們將詳細地介紹基本邏輯閘元件，每個基本邏輯閘都擁有 1～數個輸入訊號與 1 個輸出訊號。

Copyright©滄海書局

3.2.1 及閘

及閘 **(and gate)** 又稱為 and 閘，或稱且閘，具有兩個或兩個以上的輸入端及一個輸出端，輸出端與輸入端是「且」的關係，換句話說：
當所有的輸入端皆為「真」**(true)**，其輸出端為「真」**(true)**。
當有任一個輸入端為「假」**(false)**，其輸出端為「假」**(false)**。
一般來說，一個系統若使用高電位當成「邏輯 **1**」並代表某條件為邏輯上的「真」，低電位當成「邏輯 **0**」並代表某條件為邏輯上的「假」，稱之為正邏輯 **(positive logic)** 系統；反之，使用低電位當成「邏輯 1」，高電位當成「邏輯 0」，則稱之為負邏輯 **(negative logic)** 系統。由於正邏輯系統較普遍，本書也沿用此慣例，在往後的章節中，除非特別說明，不然都使用正邏輯方式來表示。因此，一個 and 閘的輸入端與輸出端之關係可定義成：
當所有的輸入端皆是 **1 (**高電位**)** 時，其輸出端為 **1 (**高電位**)**。
當有任一個輸入端為 **0 (**低電位**)** 時，其輸出端為 **0 (**低電位**)**。

Copyright©滄海書局

真值表 **(truth table)** 是一種最適合用來表示邏輯元件輸出端與輸入端關係的一種方式，其中左邊的欄位代表輸入 (通常會列出所有可能輸入的組合)，右邊的欄位表示輸出 (會列出每一種輸入組合的相對應輸出結果)。圖 3.3 表示一個兩個輸入的 and 閘之真值表及其元件符號，其中 *I*0 與 *I*1 為輸入，*Y* 是輸出。兩個輸入變數 *I*1 與 *I*0 的可能組合有四種：分別是 00、01、10 與 11，每一種組合的相對應輸出 *Y* 則分別是：0、0、0 與 1。而圖 3.4 則表示一個三個輸入的 and 閘之真值表及其元件符號。三個輸入變數 *I*2、*I*1 與 *I*0 的可能組合有八種：分別是 000、001、010、011、100、101、110 與 111，每一種組合的相對應輸出 *Y* 則分別是：0、0、0、0、0、0、0 與 1。

I_1	I_0	Y
0 (假)	0 (假)	0 (假)
0 (假)	1 (真)	0 (假)
1 (真)	0 (假)	0 (假)
1 (真)	1 (真)	1 (真)

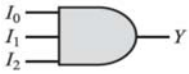
(a) 真值表

(b) 元件符號

圖 3.3 兩輸入 and 閘之真值表及其元件符號

I_2	I_1	I_0	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(a) 真值表



(b) 元件符號

圖 3.4 三輸入 and 閘之真值表及其元件符號

Copyright©滄海書局

我們也可以將 and 閘的輸入端與輸出端的關係寫成運算式，and 運算通常使用 “·” 符號來表示，所以圖 3.3 中的輸入 *I*0、*I*1 與輸出 *Y* 之運算式可表示為：

$$Y = I_1 \cdot I_0$$

也可以將 “·” 省略，而簡寫成 $Y = I_1 I_0$

同理，圖 3.4 的運算式可表示為 *Y* = *I*2 · *I*1 · *I*0 或 *Y* = *I*2*I*1*I*0。很明顯地，二輸入 and 閘的運算為：

(1) 當 *I*1 = 0 且 *I*0 = 0 時 → *Y* = 0。
(2) 當 *I*1 = 0 且 *I*0 = 1 時 → *Y* = 0。
(3) 當 *I*1 = 1 且 *I*0 = 0 時 → *Y* = 0。
(4) 當 *I*1 = 1 且 *I*0 = 1 時 → *Y* = 1。

也就是說，“·” (and) 的運算法則可表示為

(1) $0 \cdot 0 = 0$

(2) $0 \cdot 1 = 0$

(3) $1 \cdot 0 = 0$

(4) $1 \cdot 1 = 1$

Copyright©滄海書局

剛好與一般數學的乘法運算完全相同。
and 閘也可以用一個實際的例子說明如下：假設 *I*0 是代表「我有沒有空」這個事件，其中我沒空時 *I*0 = 0，我有空時 *I*0 = 1。*I*1 代表「天氣是不是晴朗」這個事件，其中天氣晴朗時 *I*1 = 1，天氣不晴朗時 *I*1 = 0。*Y* 代表「我要不要去台北」這個事件，若 *Y* = 1 時我會去台北，*Y* = 0 時不去。要決定去不去台北，可以使用一個 and 閘，表示成 $Y = I0 \cdot I1$，此 and 閘能確保，只有在我有空時 (*I*0 = 1) 而且天氣晴朗時 (*I*1 = 1)，去台北 (*Y* = 1) 這件事才會發生。只要我沒空 (*I*0 = 0) 或者天氣不晴朗 (*I*1 = 0) 兩個條件中任一個發生，我就不去台北 (*Y* = 0)。

Copyright©滄海書局

圖 3.5 是特定的輸入訊號 *I*0 與 *I*1，經過 and 閘運算後的輸出訊號變化示意圖。圖中的橫軸代表時間軸，而輸入訊號 *I*0 的變化為：在 *t*0 這個時段內的值為 0、在 *t*1 這個時段內的值為 1、在 *t*2 這個時段內的值為 0、在 *t*3 這個時段內的值為 1，一直持續變化下去。同理，另一個輸入訊號 *I*1 的變化如圖所示。在不同時段中，不同的 *I*0 與 *I*1 值，經過 and 閘之後會產生不同的輸出 *Y* 值。例如：在 *t*1 及 *t*5 兩個時段中，輸入訊號 *I*0 與 *I*1 同時為 1，故輸出訊號 *Y* 是 1；其他時段的輸入訊號 *I*0 與 *I*1 之中，至少有一個為 0，故輸出訊號 *Y* 是 0。

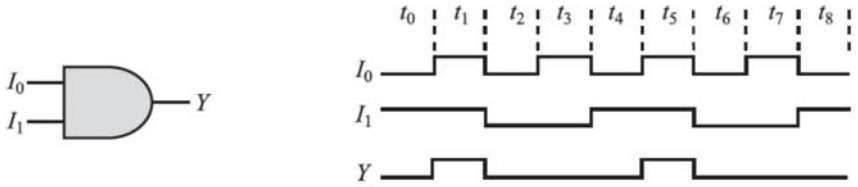


圖 3.5　兩輸入 and 閘運算之訊號變化示意圖

Copyright©滄海書局

兩輸入的 and 閘還有另一個功用——可當作數位電路中的「可控制開關」。如圖 3.6 所示，將 *I*1 連接至控制訊號，則控制訊號的值可決定輸入端 *I*0 的訊號是否能傳送到輸出端 *Y*：

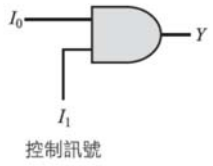


圖 3.6 數位電路中的 and 閘「可控制開關」

(1) 當 *I*1 = 1 時，*Y* = *I*0。
當 *I*1 固定為 1 時，若 *I*0 = 0，則 *Y* = 0；若 *I*0 = 1，則 *Y* = 1。
表示當控制訊號端 ***I*1** 為 **1** 時，輸出端 ***Y*** 與輸入端 ***I*0** 的訊號值是「相同的」，亦即輸入端 ***I*0** 的訊號「可以」傳送 **(pass)** 到輸出端 *Y*。
(2) 當 *I*1 = 0 時，*Y* = 0。
當 *I*1 固定為 0 時，若 *I*0 = 0，則 *Y* = 0；若 *I*0 = 1，則 *Y* = 0。
表示當控制訊號端 ***I*1** 為 **0** 時，無論輸入端 ***I*0** 是何種訊號 **(1** 或 **0)**，輸出端 ***Y*** 恆為 **0**，亦即輸入端 ***I*0** 之訊號「無法」傳送到輸出端 ***Y***。

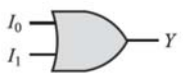
Copyright©滄海書局

3.2.2 或閘

或閘 (or gate) 又稱為 or 閘，具有兩個或兩個以上的輸入端及一個輸出端，輸出端與輸入端是「或」的關係，換句話說：
當所有的輸入端皆是 0 (假) 時，其輸出端為 0 (假)，
當有任一個輸入端為 1 (真) 時，其輸出端為 1 (真)。
圖 3.7 表示一個兩個輸入的 or 閘之真值表及其元件符號，圖 3.8 則表示一個三個輸入的 or 閘之真值表及其元件符號。

I_1	I_0	Y
0 (假)	0 (假)	0 (假)
0 (假)	1 (真)	1 (真)
1 (真)	0 (假)	1 (真)
1 (真)	1 (真)	1 (真)

(a) 真值表

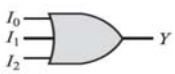


(b) 元件符號

圖 3.7 兩輸入 or 閘之真值表及其元件符號

I_2	I_1	I_0	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

(a) 真值表



(b) 元件符號

圖 3.8 三輸入 or 閘之真值表及其元件符號

Copyright©滄海書局

我們也可以將 or 閘的輸入端與輸出端的關係寫成運算式，or 運算通常使用 “+” 符號來表示，所以圖 3.7 中的輸入 *I*0、*I*1 與輸出 *Y* 之運算式可表示為：$Y = I_1 + I_0$

同理，圖 3.8 三輸入的運算式可表示為 *Y* = *I*2 + *I*1 + *I*0。此處要特別注意的是，當討論數位系統相關課題時， “+” 符號是代表 “or” 運算，並不是加法運算，且 “+” 不可省略。很明顯地，二輸入 or 閘的運算為：

(1) 當 $I_1 = 0$ 且 $I_0 = 0$ 時 $\rightarrow Y = 0$。

(2) 當 $I_1 = 0$ 且 $I_0 = 1$ 時 $\rightarrow Y = 1$。

(3) 當 $I_1 = 1$ 且 $I_0 = 0$ 時 $\rightarrow Y = 1$。

(4) 當 $I_1 = 1$ 且 $I_0 = 1$ 時 $\rightarrow Y = 1$。

也就是說， “+” (or) 的運算法則可表示為：

(1) $0 + 0 = 0$

(2) $0 + 1 = 1$

(3) $1 + 0 = 1$

(4) $1 + 1 = 1$

雖然 or 運算看起來與一般數學的加法運算很相似，實際上卻是不同的，因為 or 運算是 1 + 1 = 1，而數學加法運算則是 1 + 1 = 2。

Copyright©滄海書局

圖 3.9 是特定的輸入訊號 *I*0 與 *I*1，經過 or 閘運算後的輸出訊號變化示意圖。在 *t*2 及 *t*6 兩個時段中，輸入訊號 *I*0 與 *I*1 同時為 0，故輸出訊號 *Y* 是 0；其他時段的輸入訊號 *I*0 與 *I*1 之中，至少有一個為 1，故輸出訊號 *Y* 是 1。

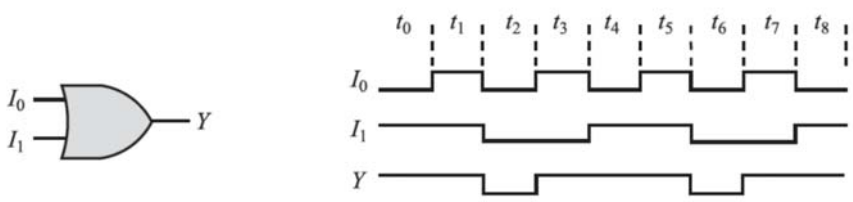


圖 3.9　兩輸入 or 閘運算之訊號變化示意圖

Copyright©滄海書局

兩輸入的 or 閘還有另一個功用——可當作數位電路中的「可控制開關」，如圖 3.10 所示，將 *I*1 連接至控制訊號，則控制訊號的值可決定輸入端 *I*0 的訊號是否能傳送到輸出端 *Y*：

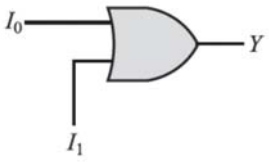


圖 3.10　數位電路中的 or 閘「可控制開關」

(1) 當 I1 = 0 時，Y = I0。
當 I1 固定為 0 時，若 I0 = 0，則 Y = 0；若 I0 = 1，則 Y = 1。
表示當控制訊號端 I1 為 0 時，輸出端 Y 與輸入端 I0 的訊號值是「相同的」，亦即輸入端 I0 的訊號「可以」傳送到輸出端 Y。

(2) 當 I1 = 1 時，Y = 1。
當 I1 固定為 1 時，若 I0 = 0，則 Y = 1；若 I0 = 1，則 Y = 1。
表示當控制訊號端 I1 為 1 時，無論輸入端 I0 是何種訊號 (1 或 0)，輸出端 Y 恆為 1，亦即輸入端 I0 之訊號「無法」傳送到輸出端 Y。

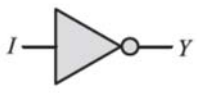
Copyright©滄海書局

3.2.3 反閘

反閘 **(not gate)** 又稱為 not 閘或稱反相閘，只有一個輸入端及一個輸出端，其輸出端的訊號永遠與輸入端相反，換句話說：
當輸入端是 **0 (**假**)** 時，其輸出端為 **1 (**真**)**，
當輸入端是 **1 (**真**)** 時，其輸出端為 **0 (**假**)**。
圖 3.11 表示一個 not 閘之真值表及其元件符號。
我們也可以將 not 閘的輸入端與輸出端的關係寫成運算式，not 運算通常使用 “ — ” 或 “ ′ ” 符號來表示，所以圖 3.11 中的輸入 *I* 與輸出 *Y* 之運算式可表示為：　$Y = I'$　或　$Y = \overline{I}$

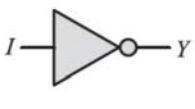
I	*Y*
0 (假)	1 (真)
1 (真)	0 (假)

(a) 真值表



(b) 元件符號

圖 3.11　not 閘之真值表及其元件符號



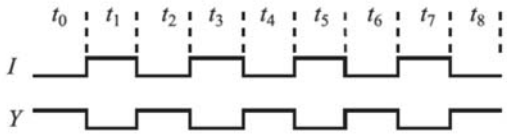


圖 3.12　not 閘運算之訊號變化示意圖

Copyrigl

很明顯地，not 閘的運算為：

(1) 當 $I = 0$ 時 $\rightarrow Y = 1$。

(2) 當 $I = 1$ 時 $\rightarrow Y = 0$。

換句話說，“ ′ ” (not) 的運算法則可表示為：

(1) $0' = 1$ (或 $\overline{0} = 1$)

(2) $1' = 0$ (或 $\overline{1} = 0$)

not 閘的主要作用是將一個訊號作「反相」的運算，亦即將 “1” 變成 “0”，或是將 “0” 變成 “1”。圖 3.12 是特定的輸入訊號 *I*，經過 not 閘
運算後的輸出訊號變化示意圖，其中輸出值 *Y* 剛好與輸入值 *I* 相反。

Copyright©滄海書局

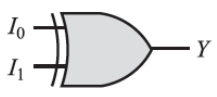
3.2.4 互斥或閘

互斥或閘 **(exclusive-or gate; xor gate; XOR gate)** 又稱為 xor 閘，具有兩個或兩個以上的輸入端及一個輸出端，輸出端與輸入端的關係為：
當輸入端訊號為 **1** 的總個數是「奇數」時，輸出端為 **1**，
當輸入端訊號為 **1** 的總個數不是「奇數」時，輸出端為 **0**。
圖 3.13 表示一個兩個輸入的 xor 閘之真值表及其元件符號，圖 3.14 則是一個三個輸入的 xor 閘之真值表及其元件符號。當有奇數個輸入為 1時，其輸出為 1；有偶數或零個輸入為 1 時，其輸出為 0。
我們也可以將 xor 閘的輸入端與輸出端的關係寫成運算式，xor 運算通常使用 “⊕” 符號來表示，所以圖 3.13 中的輸入 *I*0、*I*1 與輸出 *Y* 之運算式可表示為：

$$Y = I_1 \oplus I_0$$

I_1	I_0	Y
0	0	0
0	1	1
1	0	1
1	1	0

(a) 真值表

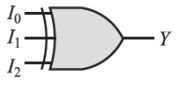


(b) 元件符號

圖 3.13 兩輸入 xor 閘之真值表及其元件符號

I_2	I_1	I_0	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(a) 真值表



(b) 元件符號

圖 3.14 三輸入 xor 閘之真值表及其元件符號

很顯然地，二輸入 xor 閘的運算為：

(1) 當 $I_1 = 0$ 且 $I_0 = 0$ 時 $\rightarrow Y = 0$。

(2) 當 $I_1 = 0$ 且 $I_0 = 1$ 時 $\rightarrow Y = 1$。

(3) 當 $I_1 = 1$ 且 $I_0 = 0$ 時 $\rightarrow Y = 1$。

(4) 當 $I_1 = 1$ 且 $I_0 = 1$ 時 $\rightarrow Y = 0$。

也就是說，“⊕” (xor) 的運算法則可表示為：

(1) $0 \oplus 0 = 0$

(2) $0 \oplus 1 = 1$

(3) $1 \oplus 0 = 1$

(4) $1 \oplus 1 = 0$

同理，圖 3.14 三輸入的運算式可表示為 *Y* = *I*2 ⊕ *I*1 ⊕ *I*0 = *I*2 ⊕ (*I*1 ⊕ *I*0) = (*I*2 ⊕ *I*1) ⊕ *I*0。換句話說，三輸入 xor 的運算乃先對其中兩個輸入 (如 *I*1 與 *I*0) 執行 xor 的動作，所得結果再跟另一個輸入 (*I*2) 做 xor 的動作

Copyright©滄海書局

圖 3.15 則是特定的輸入訊號 *I*0 與 *I*1，經過 xor 閘運算後的輸出訊號變化示意圖。在 *t*1、*t*2、*t*5 及 *t*6 四個時段中，輸入訊號 *I*0 與 *I*1 皆相同 (同時為 0 或是同時為 1)，故輸出訊號 *Y* 是 0；其他時段的輸入訊號 *I*0 與 *I*1 都是相反 (一個為 0，另一個為 1)，故輸出訊號 *Y* 是 1。

一個兩個輸入的 xor 閘，可使用數個 and、or 與 not 閘來組合得出，如圖 3.16 所示。觀察此電路，輸入訊號 *r* 與 *s* 皆相同時 (同時為 0 或同時為 1)，*m* 與 *n* 都為 0，故輸出訊號 *Y* 是 0；若輸入訊號 *r* = 0 且 *s* = 1，則 *m* = 0 且 *n* = 1，故輸出訊號 *Y* 為 1；若輸入訊號 *r* = 1 且 *s* = 0，則 *m* = 1 且 *n* = 0，故輸出訊號 *Y* 為 1。顯然地，輸入及相對應輸出完全符合 xor 真值表 (圖 3.13) 的定義。

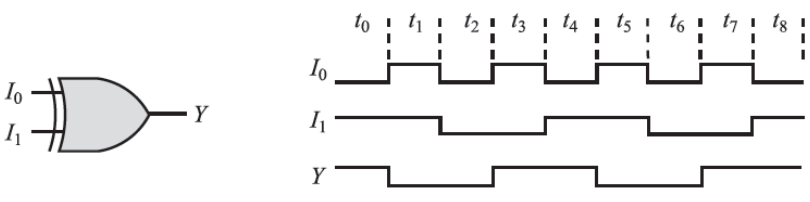


圖 3.15　兩輸入 xor 閘運算之訊號變化示意圖

Copyright©滄海書局

此外，根據 3.2.1 節 and 閘與 3.2.2 節 or 閘的表示，我們觀察圖 3.16 可得知 $m = r \cdot s'$ 而且 $n = r' \cdot s$。因為 $Y = m + n$，故此例中 xor閘的輸入端與輸出端的關係，除了原來的 $Y = r \oplus s$ 這個寫法之外，也可寫成

$$Y = m + n = (r \cdot s') + (r' \cdot s) = rs' + r's$$

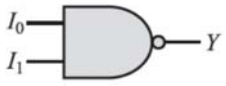
Copyright©滄海書局

3.2.5 反及閘

反及閘 **(nand gate)** 又稱為 nand 閘 (或稱 not and 閘)，具有兩個或兩個以上的輸入端及一個輸出端，輸出端與輸入端的關係為：
當所有的輸入端皆是 **1** 時，其輸出端為 **0**，
當有任一個輸入端為 **0** 時，其輸出端為 **1**。

I_1	I_0	Y
0	0	1
0	1	1
1	0	1
1	1	0

(a) 真值表



(b) 元件符號

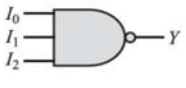
圖 3.17　兩輸入 nand 閘之真值表及其元件符號

Copyright©滄海書局

圖 3.17 表示一個兩個輸入的 nand 閘之真值表及其元件符號，圖 3.18 則表示一個三個輸入的 nand 閘之真值表及其元件符號。
比較圖 3.3 及圖 3.17 之真值表可發現，相同的輸入訊號，卻得相反的輸出訊號，可見它們互有「反相」的效果。換句話說，可以把 nand 閘看作是由一個 and 閘串接一個 not 閘而成，如圖 3.19 所示。在此，左圖的小圓圈可被視為是一個 not 閘，執行反相的運算。

I_2	I_1	I_0	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

(a) 真值表



(b) 元件符號

圖 3.18 三輸入 nand 閘之真值表及其元件符號

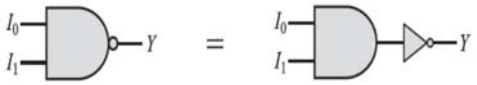


圖 3.19 nand 閘與 and 閘的關係圖

Copyright©滄海書局

圖 3.17 中的輸入 *I*0、*I*1 與輸出 *Y* 之運算式可表示為：$Y = (I_1 \cdot I_0)' = (I_1 I_0)'$

同理，圖 3.18 三輸入的運算式可表示為 Y = (I2 · I1 · I0)′。圖 3.20 則是特定的輸入訊號 I0 與 I1，經過 nand 閘運算後的輸出訊號變化示意圖。在 t1 及 t5 兩個時段中，輸入訊號 I0 與 I1 皆為 1，故輸出訊號 Y 是 0；其他時段的輸入訊號 I0 與 I1 至少有一個為 0，故輸出訊號 Y 是 1。

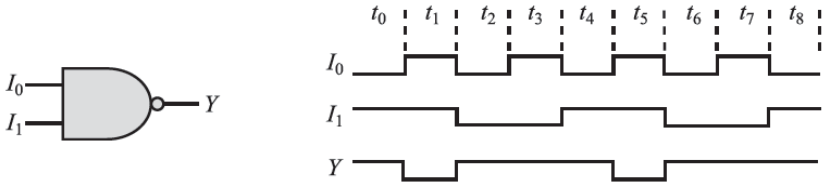


圖 3.20 兩輸入 nand 閘運算之訊號變化示意圖

Copyright©滄海書局

就如同 and 閘與 or 閘一樣，兩輸入的 nand 閘亦可當作電路中的「可控制」開關。如圖 3.21 所示，將 I1 連接至控制訊號，則控制訊號的值可決定輸入端 I0 的訊號是否能傳送到輸出端 Y：

(1) 當 I1 = 0，Y = 1。

當 I1 固定為 0 時，若 I0 = 0，則 Y = 1；若 I0 = 1，則 Y = 1。表示控制訊號端 I1 為 0 時，無論輸入端 I0 是何種訊號 (1 或 0)，輸出端 Y 恆為 1，亦即輸入端 I0 之訊號「無法」傳送到輸出端 Y。

(2) 當 I1 = 1，Y = I0′。

當 I1 固定為 1 時，若 I0 = 0，則 Y = 1；若 I0 = 1，則 Y = 0。表示控制訊號端 I1 為 1 時，輸入端 I0 訊號的反相值「可以」傳送到輸出端 Y，亦即輸出端 Y 的值是輸入訊號 I0 的反相值。

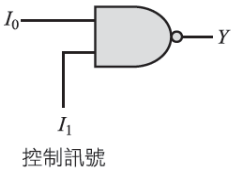


圖 3.21　數位電路中的 nand 閘「可控制開關」

Copyright©滄海書局

3.2.6 反或閘

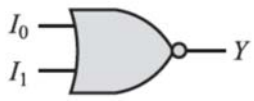
反或閘 (nor gate) 又稱為 nor 閘 (或稱 not or 閘)，具有兩個或兩個以上的輸入端及一個輸出端，輸出端與輸入端的關係為：

當所有的輸入端皆是 0 時，其輸出端為 1，

當有任一個輸入端為 1 時，其輸出端為 0。

I_1	I_0	Y
0	0	1
0	1	0
1	0	0
1	1	0

(a) 真值表



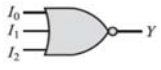
(b) 元件符號

圖 3.22　兩輸入 nor 閘之真值表及其元件符號

Copyright©滄海書局

I_2	I_1	I_0	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

(a) 真值表



(b) 元件符號

圖 3.23　三輸入 nor 閘之真值表及其元件符號

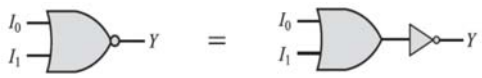


圖 3.24　nor 閘與 or 閘的關係圖

圖 3.22 表示一個兩個輸入的 nor 閘之真值表及其元件符號，圖 3.23 則表示一個三個輸入的 nor 閘之真值表及其元件符號。
比較圖 3.7 及圖 3.22 的真值表可發現，相同的輸入訊號，卻得相反的輸出訊號，可見它們互有「反相」的效果。換句話說，可以把 nor 閘看作是由一個 or 閘串接一個 not 閘而成，如圖 3.24 所示。
圖 3.22 中的輸入 *I*0、*I*1 與輸出 *Y* 之運算式可表示為：$Y = (I_1 + I_0)'$

Copyright©滄海書局

同理，圖 3.23 三輸入的運算式可表示為 *Y* = (*I*2 + *I*1 + *I*0)′。圖 3.25 則是特定的輸入訊號 *I*0 與 *I*1，經過 nor 閘運算後的輸出訊號變化示意圖。
在 *t*2 及 *t*6 兩個時段中，輸入訊號 *I*0 與 *I*1 皆為 0，故輸出訊號 *Y* 是 1；其他時段的輸入訊號 *I*0 與 *I*1 至少有一個為 1，故輸出訊號 *Y* 是 0。
兩輸入的 nor 閘和 nand 閘一樣，可當作「可控制」開關，如圖 3.26 所示，將 *I*1 連接至控制訊號，則控制訊號的值可決定輸入端 *I*0 的訊號是否能傳送到輸出端 *Y*：

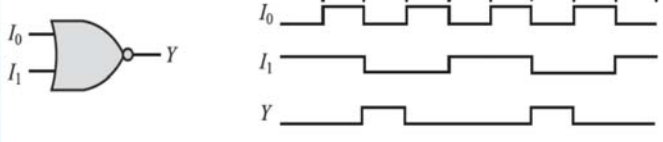


圖 3.25　兩輸入 nor 閘運算之訊號變化示意圖

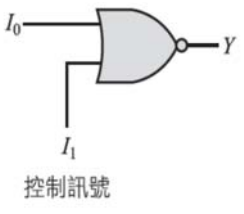


圖 3.26　數位電路中的 nor 閘「可控制開關」

Copyright©滄海書局

(1) 當 *I*1 = 1，*Y* = 0。
當 *I*1 固定為 1 時，若 *I*0 = 0，則 *Y* = 0；若 *I*0 = 1，則 *Y* = 0。表示控制訊號端 I1 為 1 時，無論輸入端 I0 是何種訊號 (1 或 0)，輸出端 Y 恆為 0，亦即輸入端 I0 之訊號「無法」傳送到輸出端 Y。
(2) 當 I1 = 0，Y = I0′。
當 I1 固定為 0 時，若 I0 = 0，則 Y = 1；若 I0 = 1，則 Y = 0。表示控制訊號端 I1 為 0 時，輸入端 I0 訊號的反相值「可以」傳送到輸出端 Y，亦即輸出端 Y 的值是輸入訊號 I0 的反相值。

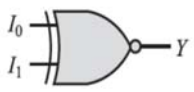
Copyright©滄海書局

3.2.7 反互斥或閘

圖 3.27 表示一個兩個輸入的 xnor 閘之真值表及其元件符號，圖 3.28 則表示一個三個輸入的 xnor 閘之真值表及其元件符號。

I_1	I_0	Y
0	0	1
0	1	0
1	0	0
1	1	1

(a) 真值表

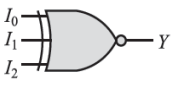


(b) 元件符號

圖 3.27　兩輸入 xnor 閘之真值表及其元件符號

I_2	I_1	I_0	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

(a) 真值表



(b) 元件符號

圖 3.28　三輸入 xnor 閘之真值表及其元件符號

Copyright©滄海書局

比較圖 3.13 及圖 3.27 之真值表可發現，相同的輸入訊號，卻得相反的輸出訊號，可見它們互有「反相」的效果。換句話說，可以把 xnor 閘看作是由一個 xor 閘串接一個 not 閘而成，如圖 3.29 所示。

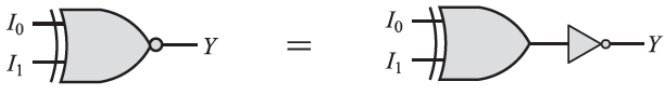


圖 3.29　xor 閘與 xnor 閘的關係圖

我們也可以將 xnor 閘的輸入端與輸出端的關係寫成運算式，xnor 運算通常使用 “ ” 符號來表示，所以圖 3.27 中的 I0、I1 與 Y 之運算式可表示為：$Y = I_1 \odot I_0$

也可將 xnor 閘的輸入端與輸出端的關係寫成：$Y = (I_1 \oplus I_0)'$

Copyright©滄海書局

xnor 閘的運算為：

(1) 當 $I_1 = 0$ 且 $I_0 = 0$ 時 $\rightarrow Y = 1$。

(2) 當 $I_1 = 0$ 且 $I_0 = 1$ 時 $\rightarrow Y = 0$。

(3) 當 $I_1 = 1$ 且 $I_0 = 0$ 時 $\rightarrow Y = 0$。

(4) 當 $I_1 = 1$ 且 $I_0 = 1$ 時 $\rightarrow Y = 1$。

換句話說，“ ” (xnor) 的運算法則可表示為：

(1) $0 \odot 0 = 1$

(2) $0 \odot 1 = 0$

(3) $1 \odot 0 = 0$

(4) $1 \odot 1 = 1$

Copyright©滄海書局

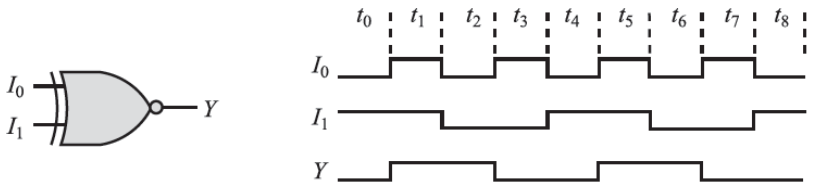


圖 3.30　兩輸入 xnor 閘運算之訊號變化示意圖

圖 3.30 則是特定輸入訊號 I0 與 I1，經過 xnor 閘運算後的輸出訊號變化示意圖。在 t1、t2、t5 及 t6 四個時段中，輸入訊號 I0 與 I1 都相同 (同時為 0 或同時為 1)，故輸出訊號 Y 是 1；其他時段的輸入訊號 I0 與 I1 都相反 (一個為 0，另一個為 1)，故輸出訊號 Y 是 0。

一個兩個輸入的 xnor 閘，和 xor 閘一樣，可使用數個 and、or 與 not 閘來組合完成，如圖 3.31 所示。觀察此電路，當輸入訊號 r = 0 且 s = 1，則 m = 0 且 n = 0，故輸出訊號 Y 為 0；當輸入訊號 r = 1 且 s = 0，則 m = 0 且 n = 0，故輸出訊號 Y 為 0；當輸入訊號 r = 0 且 s = 0，則 m = 0 且 n = 1，故輸出訊號 Y 為 1；當輸入訊號 r = 1 與 s = 1，則 m = 1 且 n = 0，故輸出訊號 Y 為 1。顯然地，輸入及相對應輸出完全符合 xnor 真值表 (圖 3.27) 的定義。

Copyright©滄海書局

觀察圖 3.31，我們知道 $m = r \cdot s$ 而且 $n = r' \cdot s'$。因為 $Y = m + n$，故此例中 xnor 閘的輸入端與輸出端的關係，除了原來的 $Y = r$　　$s = (r \oplus s)'$ 這個寫法外，也可寫成

$$Y = m + n = (r \cdot s) + (r' \cdot s') = rs + r's'$$

觀察上式，可知當 r 與 s 兩數同時為 1 或同時為 0 時，輸出為 1。

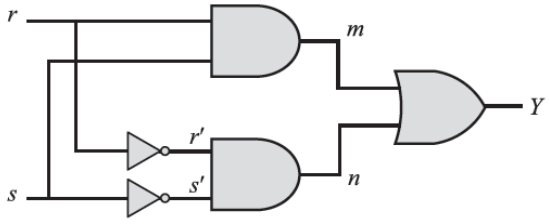


圖 3.31　兩輸入 xnor 閘的內部電路圖

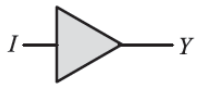
Copyright©滄海書局

3.2.8 緩衝閘

緩衝閘 (buffer gate) 又稱為 buffer 閘 (或稱緩衝器)，只有一個輸入端及一個輸出端，其輸出端的訊號值永遠與輸入端相同，換句話說：
當輸入端是 0 時，其輸出端為 0，
當輸入端是 1 時，其輸出端為 1。
圖 3.32 表示一個 buffer 閘之真值表及其元件符號。
圖 3.33 則是 buffer 閘與 not 閘的關係，如果將 not 閘之輸出再串接一個 not 閘，則可以得到一個 buffer 閘。

I	*Y*
0	0
1	1

(a) 真值表



(b) 元件符號

圖 3.32 buffer 閘之真值表及其元件符號

圖 3.33 buffer 閘與 not 閘的關係圖

Copyright©滄海書局

因為緩衝閘的輸出端訊號等於輸入端訊號，所以它並沒有執行任何邏輯運算功能。在一個數位電路中，緩衝閘通常被使用來提升某一訊號的電壓及增強該訊號的輸出電流，進而達到功率放大，增加該訊號「驅動能力」的目的。通常在數位電路中，若某個訊號「同時」連接到太多個輸入端而導致驅動能力不足時，可先串接一個「緩衝閘」，再連接到所需的輸入端。例如：假設某類型的 and 閘，它的輸出端可接的標準負載個數最多為 4 (也稱 fan-out (扇出數) 為 4)。若電路中有一個此類型 and 閘的輸出需要接到 5 個其他邏輯閘的輸入，如圖 3.34 所示，則此電路會有驅動能力不足的問題。為了確保電路工作行為正確，我們需插入適當的緩衝閘，如圖 3.35 所示。插入緩衝閘後，電路的邏輯行為並不會被改變，但是所增強的驅動能力可確保電路行為正確無誤。

Copyright©滄海書局

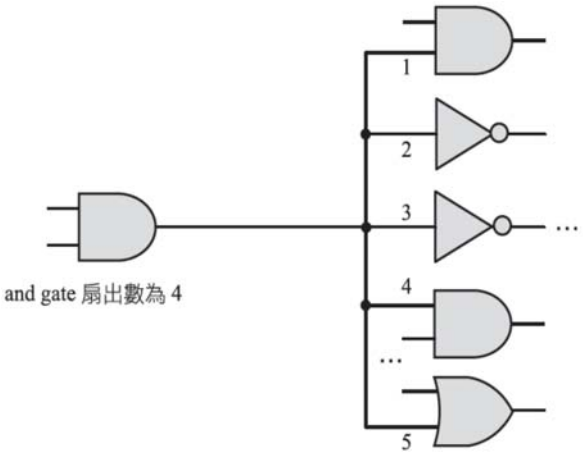


圖 3.34 驅動能力不足的電路

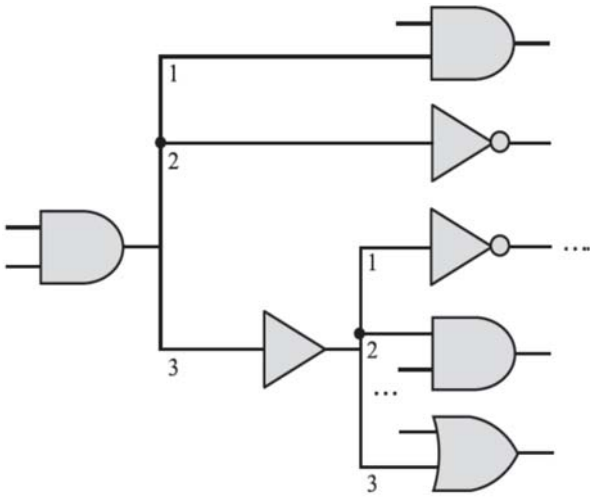
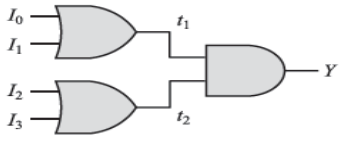


圖 3.35 插入緩衝閘增加驅動能力的電路

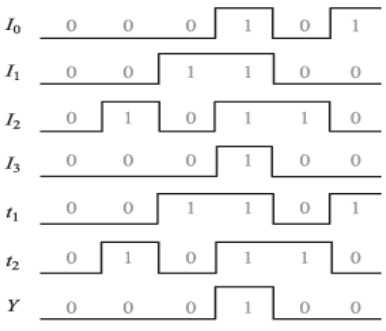
Copyright©滄海書局

3.3 數位系統之設計與實作

簡單的數位電路可能由數個到數百個邏輯閘元件連接組合而成，較複雜的數位電路系統則可能由數萬個、數十萬，到數百萬個邏輯閘元件連接組合而成。圖 3.36 是一個簡單的數位電路 (僅含三個邏輯閘) 與其輸出訊號變化示意圖。很明顯地，在不同的輸入 (*I*0、*I*1、*I*2 與 *I*3)狀況時，會有不同的中間值 (*t*1 與 *t*2) 與不同的輸出結果 (*Y*)。中間值 *t*1 是輸入 *I*0 與 *I*1 執行 or 運算後的結果，可表示成 *t*1 = *I*0 + *I*1。另一個中間值 *t*2 則是輸入 *I*2 與 *I*3 執行 or 運算後的結果，可表示成 *t*2 = *I*2 + *I*3。輸出值 *Y* 則是中間值 *t*1 與 *t*2 執行 and 運算後的結果，可表示成 *Y* = *t*1 · *t*2 = (*I*0 + *I*1) · (*I*2 + *I*3)。



(a) 邏輯電路圖



(b) 輸入/中間/輸出訊號變化示意圖

圖 3.36 邏輯電路及其訊號變化示意圖

Copyright©滄海書局

此外，我們可以畫出此電路四個輸入訊號的所有輸入組合及其相對應的輸出值，得出其真值表如表 3.1。

表 3.1　真值表

I_0	I_1	I_2	I_3	t_1	t_2	Y
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	1	1	1
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	0	0
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	0	0
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

Copyright©滄海書局

畫出真值表後，可以很清楚知道輸入與輸出訊號間的關係。再舉另一個例子，圖 3.37 之電路的真值表則如表 3.2 所示。

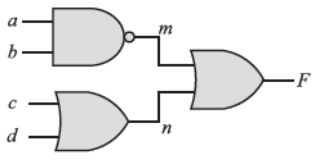


圖 3.37　某邏輯電路圖

表 3.2　真值表

a	b	c	d	m	n	F
0	0	0	0	1	0	1
0	0	0	1	1	1	1
0	0	1	0	1	1	1
0	0	1	1	1	1	1
0	1	0	0	1	0	1
0	1	0	1	1	1	1
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	0	1
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	0	0	0
1	1	0	1	0	1	1
1	1	1	0	0	1	1
1	1	1	1	0	1	1

Copyright©滄海書局

一般來說，數位電路設計者，會依據客戶的詳細規格需求 (即針對不同的 0 與 1 輸入組合，產生某特定的輸出結果)，設計出相對應的電路 (包含設計所需要的邏輯閘元件之組合與相對應的連接線)。如圖 3.36(a) 所示，此電路會根據不同的輸出組合，產生不同的結果。
在往後的幾章，我們將會詳細介紹數位電路設計時，應注意的相關事項。
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」，目前最受歡迎的就是用半導體 (semiconductor) 技術，將電路以積體電路 (integrated circuit, IC) 或稱晶片 (chip) 的方式來製作。這些晶片 IC 可根據複雜度 (IC 裡面包含的邏輯閘個數) 分成四大類，分別是：

1. 小型積體電路 (small-scale integrated circuit, SSI)一個 IC 包含大約數十個邏輯閘。
2. 中型積體電路 (medium-scale integrated circuit, MSI)一個 IC 包含大約數千個邏輯閘。
3. 大型積體電路 (large-scale integrated circuit, LSI)一個 IC 包含大約數萬個邏輯閘。
4. 超大型積體電路 (very large-scale integrated circuit, **VLSI**)一個 IC 包含大約數十萬個邏輯閘以上。

Copyright©滄海書局

因此，IC 電路設計也通常稱之為 VLSI 設計或是超大型積體電路設計。而隨著這幾年半導體製程技術 (process technology) 的急速進步，目前一個 IC 已可包含數百萬到千萬個邏輯閘以上，有些專家把這樣高複雜度的 IC 歸為新的第五類，稱之為單晶片系統 (system on a chip)，簡稱為 **SoC**。當一個 IC 可包含數千萬個邏輯閘時，整個系統就有可能置放在同一個 IC 裡面。
如果不用複雜度來分類，IC 也可以根據製作時所使用的電路製造技術 (circuit technology) 來區別之。不同的製作技術做出的電路稱之為不同的邏輯家族 (logic family)。常見的邏輯家族有四種：

(1) TTL (transistor-transistor logic) 邏輯家族
早期最普遍被使用的家族。

(2) ECL (emitter-coupled logic) 邏輯家族
適合高速的電路使用。

(3) MOS (metal-oxide semiconductor，金氧半導體) 邏輯家族
適合高密度的電路製作 (亦即同樣大小的 IC，可包含更多數目的邏輯閘)。

(4) CMOS (complementary metal-oxide semiconductor) 邏輯家族
適合低功率消耗的電路製作。

Copyright©滄海書局

不同的邏輯家族乃是使用不同的電路製造技術，因材料的不同或是阻抗、電壓準位的相異，所以很難混合使用，混用的電路可能因為無法匹配的問題而損壞。因為每個家族使用不同的方式與材質來製作一個邏輯閘，因此，一般而言，要評估這些家族的特色，可以從觀察這個家族的邏輯閘特性來著手，主要特性包含：

1. 扇出數 **(fan-out)**
   在不影響正常功能的情況下，一個邏輯閘可以推動的標準負載數。一個 and 閘的扇出數若為 4，則表示此 and 閘的輸出最多只能接到 4 個其他邏輯閘的輸入。若電路中此 and 閘的輸出接到 5 個以上的邏輯閘，如圖 3.34 所示，則此電路會有驅動能力不足的問題，導致這個 and 閘的輸出值無法正確地傳送給後端的 5 個邏輯閘當成輸入。
2. 扇入數 **(fan-in)**
   邏輯閘的輸入個數。
3. 功率消耗 **(power consumption)**
   邏輯閘正常運作時所需消耗掉的功率 (power)。
4. 傳輸延遲 **(propagation delay)**邏輯閘正常運作所需要的時間延遲。也就是說，當目前的輸入，傳入邏輯閘之後，需要多久的時間延遲，此邏輯閘的輸出端才會產生正確結果。以目前的技術而言，閘的傳輸延遲時間通常以奈秒 (10^{-9} 秒) 為單位來計算。

Copyright©滄海書局

不同的家族、不同的 IC 製程、不同的邏輯閘，上述的四項特色值可能都不同。現今 TTL 與 ECL 的使用已愈來愈少，CMOS (互補式金氧半導體) 家族是最被廣泛使用的。CMOS 家族主要乃是用 n-通道(n-channel) 的金氧半導體電晶體 (MOS transistor) 與 p-通道 (p-channel)的金氧半導體電晶體所構成。圖 3.38 是一個 CMOS 反相閘 (not gate or inverter) 的構成電路，共包含兩個電晶體。圖中上面那個 p-通道電晶體的結構圖則如圖 3.39(a) 所示，下面那個 n-通道電晶體的結構圖則如圖 3.39(b) 所示。

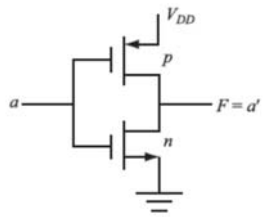
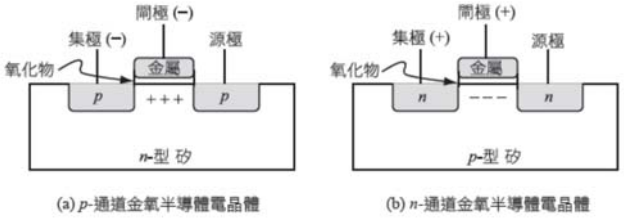


圖 3.38 CMOS 反相器電路



(a) p-通道金氧半導體電晶體　(b) n-通道金氧半導體電晶體

圖 3.39 MOS 電晶體結構圖

Copyright©滄海書局

一般來說，專門幫客戶設計電路的公司，又稱之為「IC 設計公司」或 design house，如著名的聯發科、威盛、聯詠或凌陽即是。而專門將 design house 所設計的電路以半導體技術製作成真正 IC 的公司，則稱之為晶圓代工廠，如著名的台積電 (TSMC) 或聯電 (UMC) 即是。

目前半導體製程技術 (semiconductor process technology) 已從微米 (μm, 10^{-6} 米) 進步到奈米 (nm, 10^{-9} 米) 製程，因此一個 IC 晶片可包含高達數千萬個以上的邏輯閘元件。例如：以一個 Intel Pentium CPU 而言，它是一個相當複雜的 IC，其內部所包含的邏輯閘個數高達千萬個。

Copyright©滄海書局