
Chapter
Chapter
Chapter
Chapter
Chapter
Chapter
Chapter
Chapter
11

可規劃邏輯裝置

概　　述

- **可規劃邏輯裝置** (Programmable Logic Devices; *PLD*) 是由許多**可規劃邏輯方塊** (Programmable Logic Blocks) 與**連接方塊** (Programmable Interconnection Blocks) 所組成的，即可規劃邏輯方塊可依**使用者之需要**，以實現各種不同功能之**邏輯函數** (Logic Function)。
- 若**可規劃邏輯方塊**包含**組合邏輯**(Combinational Logic)與**正反器** (Flip-Flop)，則 *PLD* 便可用來實現任何**組合邏輯電路** (Combinational Logic Circuits) 與**序向邏輯電路** (Sequential Logic Circuits)，以構成完整之**數位系統**。
- 由於使用上具有許多之**彈性**，因此 *PLD* 已廣泛使用在**實現數位系統**之設計上，且由於積體電路製程之持續進步，目前 *PLD* 可用來實現**相當複雜之數位系統**，以取代許多 *MSI*、*LSI* 與 *VLSI* 等元件所組成之**電路模組**，如此不但可省去電子工程師選擇**不同邏輯函數** *IC* **之時間**，亦可減少 *IC* **庫存**之風險，故 *PLD* 已漸漸成為**特殊應用積體電路** (Application Specific Integrated Circuits; *ASIC*) 的主要技術。

◆ *PLD* 之規劃(Programming)方式可分為光罩規劃(Mask Programming)與現場規劃(Field Programmable)等兩種方式：

1. 光罩規劃需由使用者提供電路之規劃模型，再送進 *IC* 製造廠進行規劃，故在製造完成後，此種 *PLD* 之功能便已固定，即無法再改變 *PLD* 所能實現之邏輯功能。

2. 現場規劃可透過各種規劃裝置 (Programming Devices)，使用者可依實際電路之需求，自行定義 *IC* 所實現之邏輯功能。

◆ 依使用可程式開關之不同，目前 *PLD* 大致上可分為下列兩種：

1. 可重複規劃之 *PLD*：使用 *SRAM*(Static Random Access Memory)或 *EEPROM* (Electrically Erasable Programmable *ROM*) 來製作可程式開關，則經過規劃之 *PLD*，可再清除舊規劃程式後，再重新規劃 *IC* 所實現之邏輯函數。

2. 一次規劃 (One-Time Programming) 之 *PLD*：使用熔絲 (Fuse) 或 *PROM* (Programmable ROM) 來製作可程式開關，則一旦經過規劃後，便無法再重新規劃 *IC* 所能實現之邏輯函數。

◆ 依 *PLD* 之電路架構與連接方式的不同，目前常用之 *PLD*，大致上可分為下列三種：

1. 由具完全解碼之解碼器 (Decoder) 與 *OR* 陣列所構成之 *PLD*，稱為唯讀記憶器 (Read Only Memory; *ROM*)。

2. 由 *AND-OR* 閘陣列所構成之 *PLD*，稱為可規劃邏輯閘陣列 (Programmable Logic Array; *PLA*)。

3. 由許多可規劃之邏輯方塊 (Logic Blocks) 與連接方塊 (Interconnection Blocks) 所構成之 *PLD*，稱為現場可規劃之邏輯閘陣列 (Field Programmable Gate Array; *FPGA*)。

◆ 唯讀記憶器 (*ROM*) 與可規劃邏輯閘陣列 (PLA) 之電路結構，皆採用 *AND-OR* 陣列所構成之 *PLD*，因 *AND-OR* 陣列所能實現之邏輯函數會受到一定的限制，故 *ROM* 與 *PLA* 僅可實現較簡單之數位電路。

◆ *FPGA* 是由許多可實現邏輯函數 (包括組合邏輯與正反器) 之邏輯方塊與連接這些邏輯方塊的可規劃連接方塊所組成，雖然每個邏輯方塊所能實現之邏輯函數相當有限，然結合許多邏輯方塊後，便可用來執行較大之邏輯電路，故 *FPGA* 可用來實現較複雜之數位電路。

唯讀記憶器(基本結構)

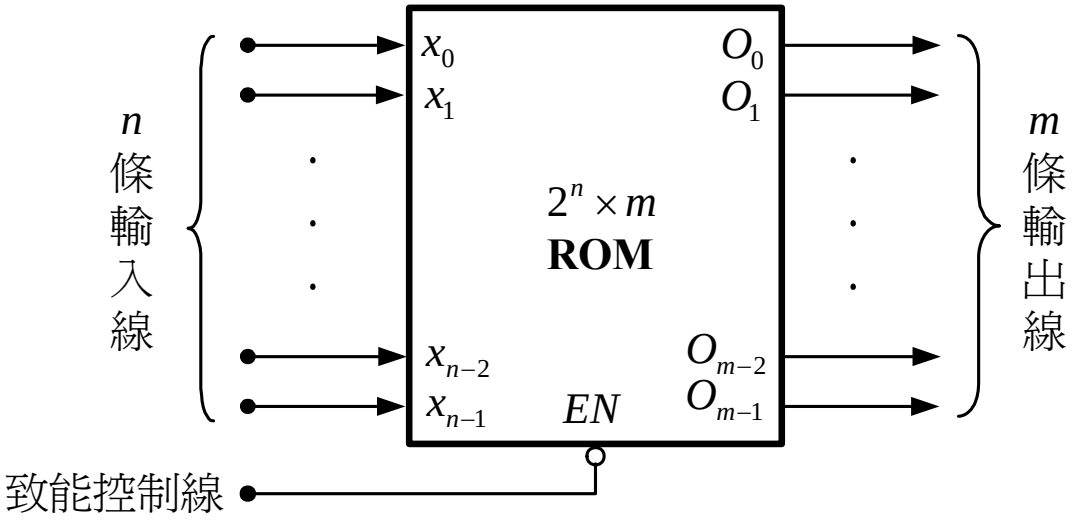
- **唯讀記憶器** (Read Only Memory; ROM) 僅能用來儲存一組固定之二進位資料，而無法再寫入新的二進位資料，即一旦將 *ROM* 所能實現之組合邏輯規劃完成後，使用者僅可執行固定功能之邏輯運算，而無法改變 *ROM* 所能執行之邏輯函數，故 *ROM* 可用來儲存永久性或不常改變之二進位資料。

- 真值表(Truth Table)可用來表示任何布林函數，對使用一個可提供完全解碼(對 n 個輸入變數而言，解碼器輸出可產生 2^n 個不同最小項之組合)之解碼器 (Decoder) 做為輸入，而使用可規劃之 *OR* 閘當作輸出，即可用來實現任何組合邏輯電路 (Combinational Logic Circuits)，而此種電路結構稱為唯讀記憶體 (*ROM*)。

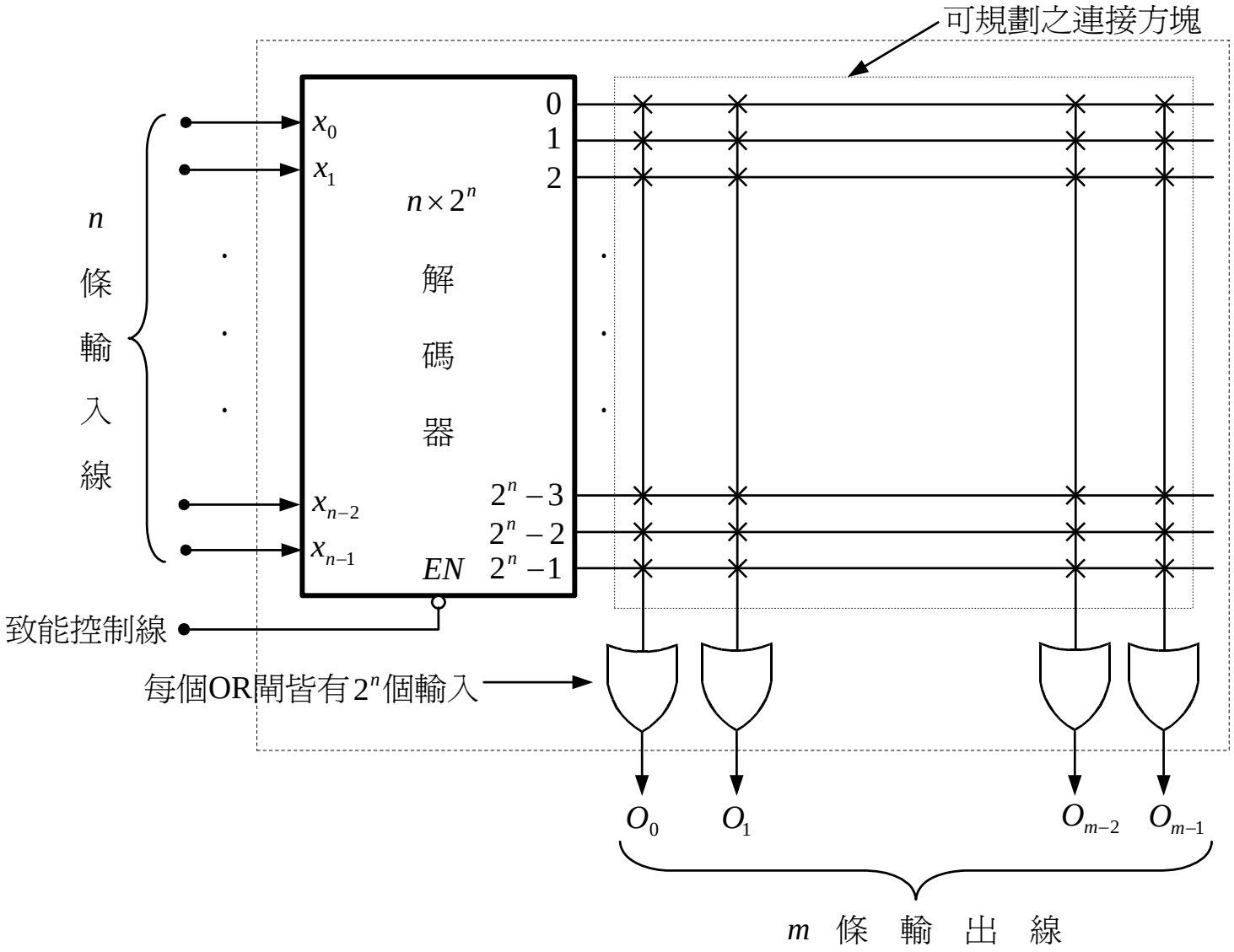
- *ROM* 包含解碼器與可規劃 *OR* 閘所組成 *IC* 裝置，而解碼器之輸出與 *OR* 閘輸入間，可藉由規劃方式，以指定彼此間 (*AND* 輸出與 *OR* 輸入間) 的連接關係，因此只需要輸入一個指定位址 (解碼器之輸入端)，便可藉由既定的規劃，在輸出端 (*OR* 閘之輸出端) 讀出該位址之內容。

◆ *ROM* 是由具一個完全解碼功能之解碼器(Decoder)、可規劃之連接方塊與 *OR* 閘所構成的，即 n 個輸入變數（位址數入線）可提供 2^n 條（位址線）不同組合，而每一個輸出線之組合稱為一個字 (Word)，故對具 2^n 不同組合之 *ROM* 而言，總計可產生 2^n 個不同之字。

◆ 對具 n 條輸入與 m 條輸出規格之 *ROM* 而言，則此電路所包含之總位元數 (Bit Numbers) 為 $2^n \times m$ 個，而此位元數稱為 *ROM* 之容量 (Capacity)，而對一個容量為 $2^n \times m$ *ROM* 而言，其方塊圖與電路結構，如下圖所示。



$2^n \times m$ *ROM* 之方塊圖



$2^n \times m$ ROM 之電路結構圖

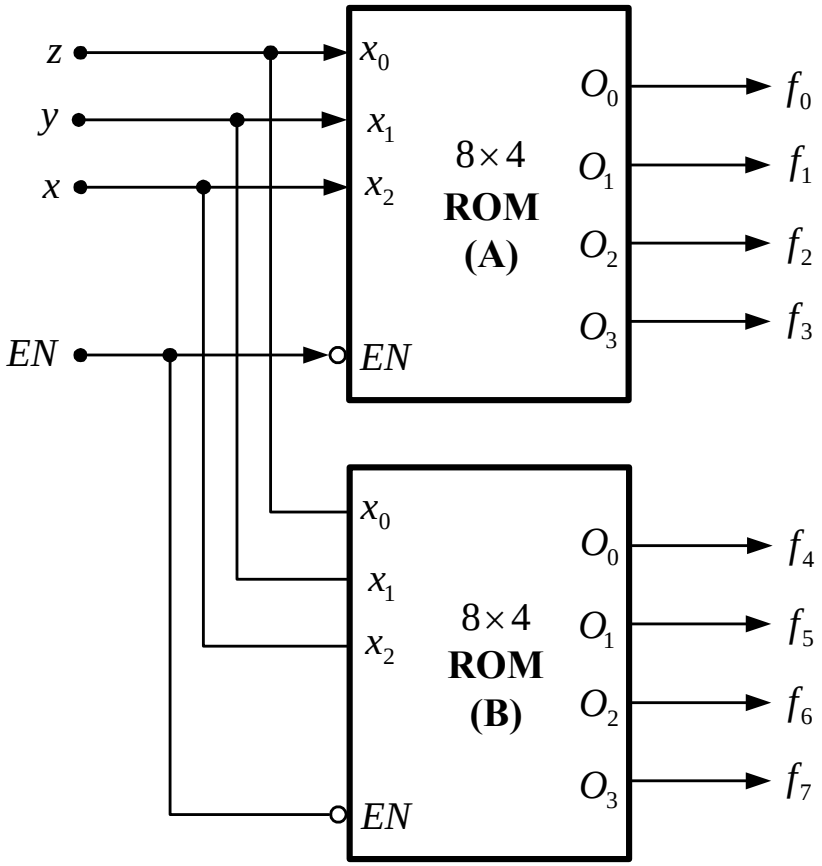
◆ 觀察上圖可知，*ROM* 所使用之邏輯元件與其規格成指數 (Exponential) 函數之正比關係，即 *ROM* 所需之邏輯閘數量與輸入、輸出線成指數正比關係。為減少邏輯元件之使用量，當需要使用較大規格之 *ROM* 時，可串接幾個較小規格之 *ROM*，以擴充成一個較大規格之 *ROM*。

例題 11-1

試利用兩個具致能 (Enable) 控制線之 8×4 *ROM*，以擴充成一個 8×8 *ROM* (結合兩個 *ROM*，以擴充成 1 個較多輸出線之 *ROM*)。

解

將兩個 8×4 *ROM* 之三條位址輸入線 (分別為 *x*、*y* 與 *z*) 與致能控制線 (*EN*) 分別並接在一起，而將每個 *ROM* 之輸出線分別獨立輸出 (即不做任何並接，即 8×4 *ROM* (A) 之輸出為 $f_0 \sim f_3$，而 8×4 *ROM* (B) 之輸出為 $f_4 \sim f_7$)，便可將兩個 8×4 *ROM* 擴充成一個 8×8 *ROM*，如下圖所示。



觀察上圖可知，兩個 *ROM* 同時受到「*EN*」之控制，以擴充 *ROM* 規格之原理如下：

1. 當 $EN = 0$ 時，兩個 8×4 *ROM* 皆會執行邏輯運算，因此 *ROM* 之輸出線會增加 1 倍，而輸入線不改變。

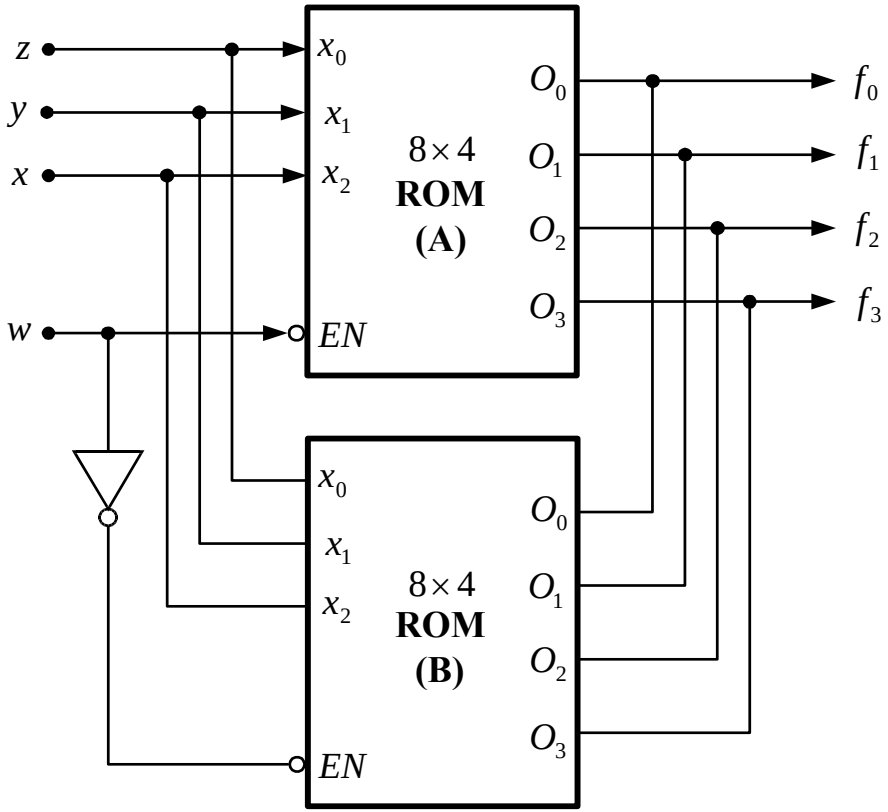
2. 當 $EN = 1$ 時，兩個 8×4 *ROM* 皆不會執行邏輯運算。

例題 11-2

試利用兩個具致能 (Enable) 控制線之8×4 *ROM*，以擴充成一個16×4 *ROM* (結合兩個 *ROM*，以擴充成 1 個較多輸入線之 *ROM*)。

解

將兩個8×4 *ROM* 之三條位址輸入線 (分別為 x、y 與 z) 並接在一起，接著將致能控制線 (*EN*) 連接至8×4 *ROM* (A) 之 *EN*，並加上一個反相器後，再連接至8×4 *ROM* (B) 之 *EN*，最後再將兩個 *ROM* 輸出線 ($O_1 \sim O_3$) 分別並接在一起，以作為 *ROM* 之輸出線，便可將兩個8×4 *ROM* 擴充成一個16×4 *ROM*，如下圖所示。



觀察上圖可知，將兩個 *ROM* 之「*EN*」，以反相和非反相之方式連接在一起，當作輸入位址線之最高有效位元 (w)，而擴充之原理如下：

1. 當 $EN=0\ (w=0)$ 時，則 8×4 *ROM* (*A*) 執行邏輯運算，即解碼器執行 0～7 之解碼運算。

2. 當 $EN=1\ (w=1)$ 時，則 8×4 *ROM* (*B*) 執行邏輯運算，即解碼器執行 8～15 之解碼運算。

ROM 之種類與特性(一次規劃)

- 依使用不同製造技術之可程式開關，*ROM* 之種類可分為一次規劃 (One-Time Programming) 與可重複規劃 (Reprogramming) 兩大類。
- 使用熔絲、Mask *ROM* 與 *PROM* 作為可程式連接開關，皆屬於一次規劃 (One-Time Programming) 之 *ROM*，即此類 *ROM* 之內容經規劃後，所儲存之內容便永遠固定，接著討論這 3 種 *ROM* 之規劃方式與特性如下：

1. 使用 Mask *ROM* 作為可程式開關，需由使用者提供規劃 *ROM* 之資料，送至 *IC* 製造廠作光罩 (Mask) 處理，以指定 *ROM* 所執行之邏輯函數，而於包裝完成後，*ROM* 所儲存之內容便無法改變，因製作光罩之單位成本較高，故適合大量生產時使用，以降低生產成本。
2. 使用熔絲與 *PROM* 作為可程式開關而言，使用者依實際之需要，自行燒錄 *ROM* 所執行之邏輯函數，故可稱為現場可規劃 (Field Programmable) 之 *ROM*，但一經燒錄完成後，則 *ROM* 所儲存之內容亦無法改變，因此較適合少量生產時使用。

ROM 之種類與特性(可重複規劃)

◆ 使用 *UV EPROM* 與 *EEPROM* 作為可程式開關，皆屬於可重複規劃 (Reprogramming) 之 *ROM*，使用者可自行使用燒錄器，將資料燒錄到記憶電晶體，以指定所需之邏輯函數。且當 *ROM* 所執行之邏輯函數有改變時，使用者亦可將所規劃之內容清除 (Erasable) 後，便可再一次規劃 *ROM* 之內容，以實現新的邏輯函數，接著分成 2 個部份來討論規劃方式與特性如下：

1. 使用 *UV EPROM* 作為可程式開關而言，使用者可自行使用燒錄器(Programmer)，將 *ROM* 所儲存之內容規劃完成後，再利用高強度之紫外線照射 20～30 分鐘後(需在 *IC* 增加一個玻璃窗)，便可清除 *ROM* 所儲存之內容，以提供再一次燒錄新的內容。若僅需修改部分 *ROM* 之內容時，亦必須清除全部內容後，再全部重新燒錄新的儲存內容，此為 *UV EROM* 之最大缺點。

2. 使用 *EEPROM* 作為可程式開關而言，使用者亦可自行使用燒錄器，將 *ROM* 所需儲存之內容規劃完成後，再利用電壓快速清除 *ROM* 所儲存之內容(不需在 *IC* 增加一個玻璃窗)，以提供再一次燒錄新的內容之用。使用電壓清除 *ROM* 所儲存之內容，可直接在電路上進行。若僅需修改一部分內容時，*EEPROM* 僅需清除與重新燒錄修改之部分即可，此為 *EEROM* 之最大

優點。

使用 ROM 實現邏輯電路

◆ 因 *ROM* 之輸入包含一個**具完全解碼功能之解碼器**(對 n 個輸入變數而言，解碼器輸出可產生 2^n 個不同最小項)，且在解碼器之輸出與 *OR* 閘輸入間，使用一個**可規劃之連接方塊**，以指定彼此間的連接關係，因此只要**適當的規劃**連接方塊之開關，便可使用 *ROM* 來實現**任何積項之和** (SOP) 所構成之組合邏輯電路。

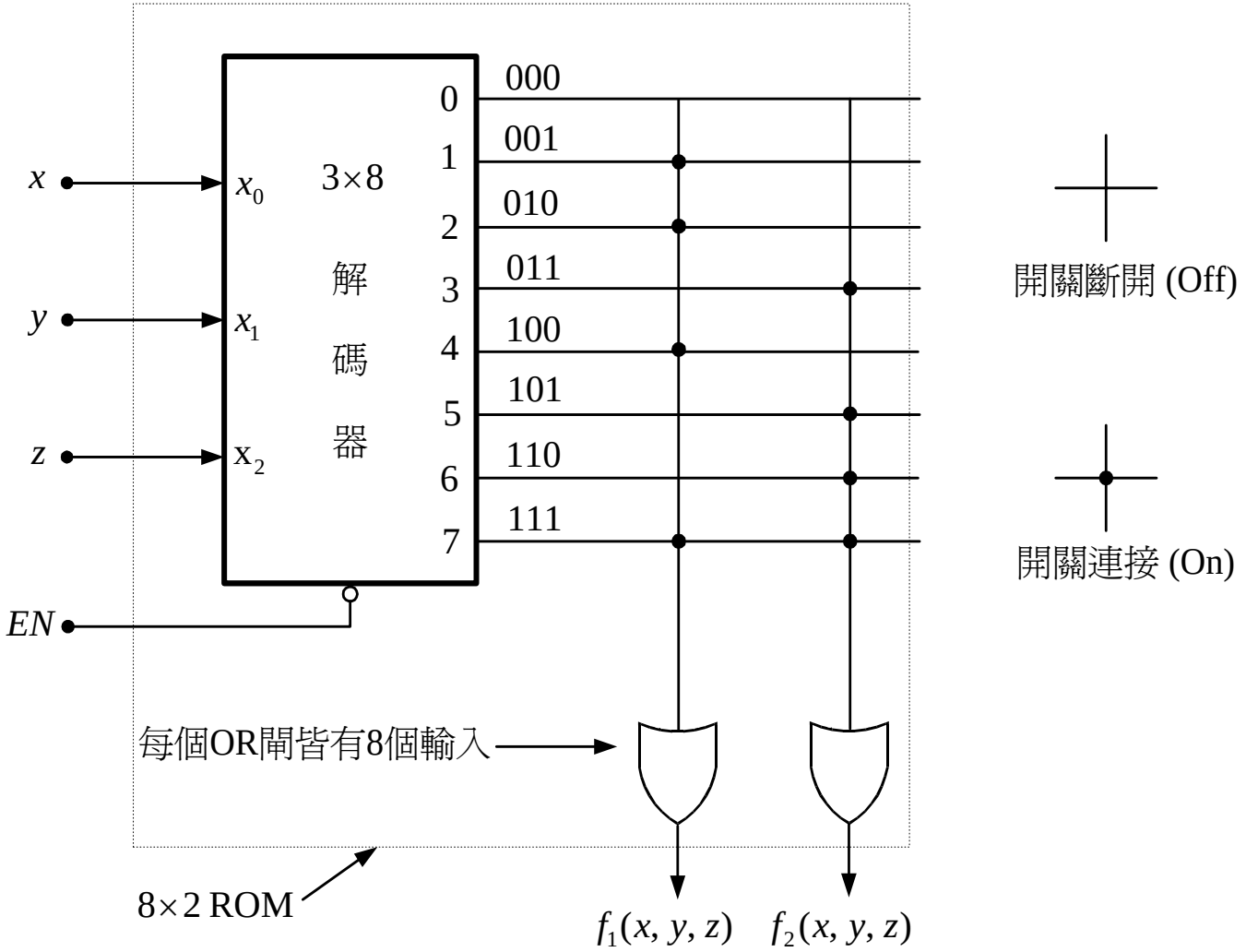
例題 11-3

試利用**適當大小**之 *ROM* 來實現 $f_1(x, y, z) = \sum(1, 2, 4, 7)$ 與 $f_2(x, y, z) = \sum(3, 5, 6, 7)$ 之**布林函數式**

解

因 $f_1(x, y, z)$ 與 $f_2(x, y, z)$ 等 2 個輸出，皆包含 3 個輸入變數，故需採用 8×2 *ROM* 來實現所求之布林函數。接著**適當的規劃**可規劃連接方塊之開關連接情況，即輸出 $f_1(x, y, z)$ 與 $f_2(x, y, z)$ 為**邏輯 1** 之最小項，在可規劃連接方塊之相對位置的開關設定為**導通** (On)，輸出 $f_1(x, y, z)$ 與 $f_2(x, y, z)$

為邏輯 0 之最小項，在可規劃連接方塊之相對位置的開關設定為斷開 (Off)，即可實現所求之組合邏輯電路，如下圖所示。


x
y
z
EN
x_0
x_1
x_2
3×8
解
碼
器
0
1
2
3
4
5
6
7
000
001
010
011
100
101
110
111
開關斷開 (Off)
開關連接 (On)
每個OR閘皆有8個輸入
8×2 ROM
$f_1(x, y, z)$
$f_2(x, y, z)$

例題 11-4

試使用適當大小之 *ROM* 來設計一組合邏輯電路，它接受一個 3 位元之二進位數字，而產生一個等於輸入 7 倍輸出的二進位數字。

解

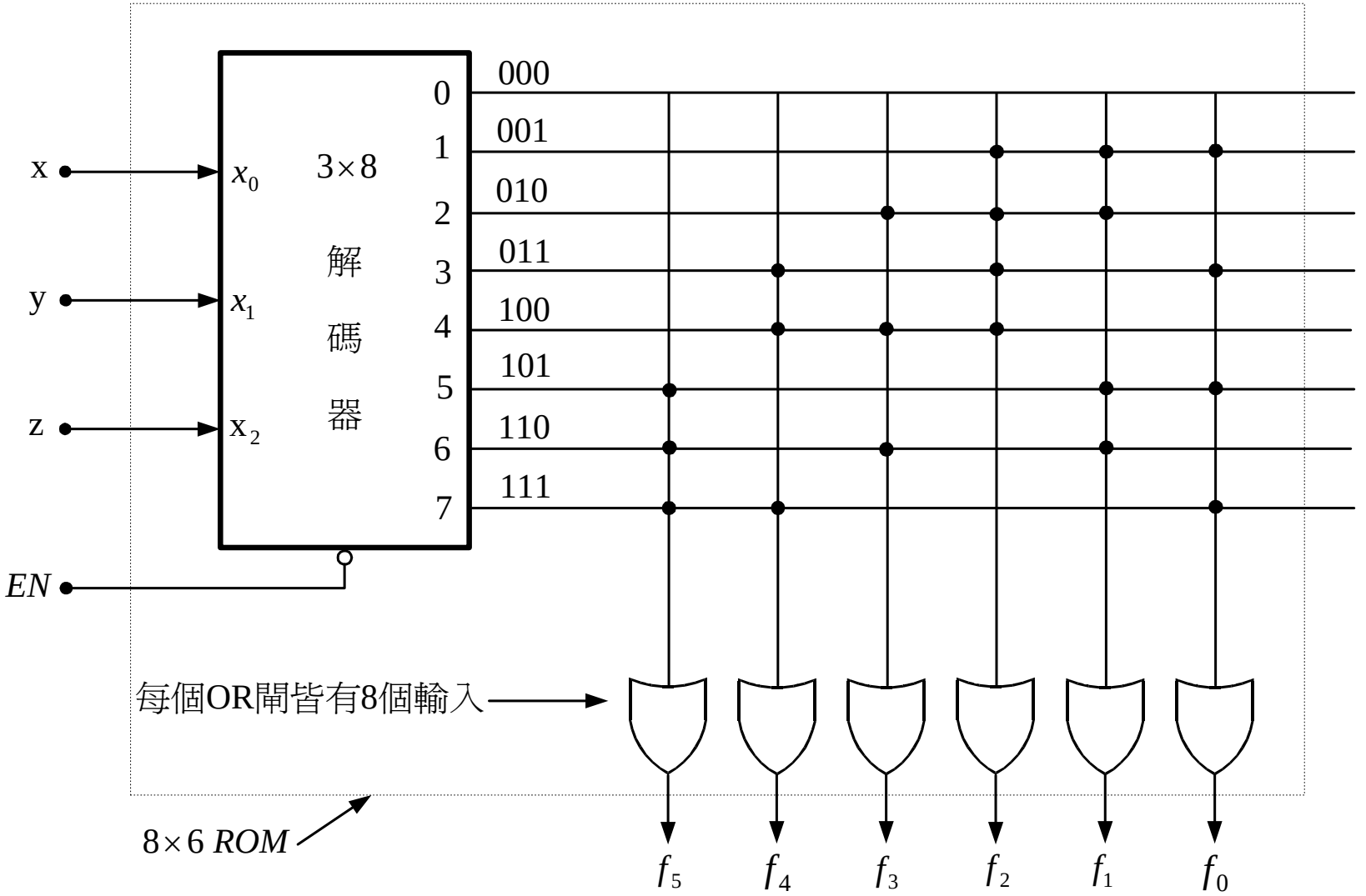
所求組合邏輯電路需 3 個輸入變數，分別標示為 x、y 與 z 等 3 個符號，而 3 位元二進位數之最大值為 $(7)_{10}$，且 $(7)_{10}$ 之 7 倍為 $(49)_{10}$，因 $(49)_{10}$ 必需用 6 個位元之二進位數表示，故所設計之電路需有 6 個輸出，分別標示為 f_5、f_4、f_3、f_2、f_1 與 f_0 等 6 個符號，並以下面之真值表來定義輸出與輸入間的關係示。

輸　入	輸　出

x	y	z	f_5	f_4	f_3	f_2	f_1	f_0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	1	1
0	1	0	0	0	1	1	1	0
0	1	1	0	1	0	1	0	1
1	0	0	0	1	1	1	0	0
1	0	1	1	0	0	0	1	1
1	1	0	1	0	1	0	1	0
1	1	1	1	1	0	0	0	1

因 $f_5 \sim f_0$ 等 6 個輸出，皆包含 3 個輸入變數，故需採用 8×6 *ROM* 來實現所求之布林函數，即使用一個 3 對 8 線之解碼器與 6 個 8 個輸入 *OR* 閘所組成的 *ROM*，以執行上面之真值表。

接著適當的規劃可規劃連接方塊之開關連接情況，即輸出 $f_5 \sim f_0$ 為邏輯 1 之最小項，在可規劃連接方塊之相對位置的開關設定為導通 (On)，而輸出 $f_5 \sim f_0$ 為邏輯 0 之最小項，在可規劃連接方塊之相對位置的開關設定為斷開 (Off)，即可實現所求之組合邏輯電路，如下圖所示。


x
y
z
EN
x_0
x_1
x_2
3×8
解
碼
器
0
1
2
3
4
5
6
7
000
001
010
011
100
101
110
111
每個OR閘皆有8個輸入
8×6 ROM
f_5
f_4
f_3
f_2
f_1
f_0

使用多層 *ROM* 實現組合邏輯電路

◆ 當使用 *ROM* 來實現輸入變數較多之組合邏輯電路時，為避免輸入線（位址線）過多，使 *ROM* 所使用之邏輯元件快速增加，，除了可用多個較小規格之 *ROM*，以擴充成較大規格 *ROM* 之方法外，亦可使用多個較小規格 *ROM*，組成多層 *ROM* 結構方式來實現。

例題 11-5

試利用多層 *ROM* 來實現下列布林函數式

$$f_1(u, v, w, x, y, z) = \sum(0, 1, 3, 17, 32, 33, 46, 47, 62, 63)$$

$$f_2(u, v, w, x, y, z) = \sum(1, 3, 10, 15, 19, 20, 26, 47, 48, 51, 52)$$

$$f_3(u, v, w, x, y, z) = \sum(1, 4, 17, 19, 20, 32, 33, 51, 52, 62, 63)$$

$$f_4(u, v, w, x, y, z) = \sum(0, 1, 19, 20, 26, 28, 46, 47, 48, 49, 51, 52)$$

$$f_5(u, v, w, x, y, z) = \sum(1, 3, 15, 17, 19, 20, 28, 51, 52, 62, 63)$$

解

若直接使用 *ROM* 實現所求之 6 變數布林函數，則需使用 $2^6 \times 5$ 位元之 *ROM*，故可考慮使用 2 個較小規格之 *ROM*，且分成 2 層來實現，以減少實現所求布林函數之邏輯元件。首先列出所求布林函數之輸出為邏輯 1 部份的真值表，如下表所示。

十進位數	輸入						輸出				
	u	*v*	*w*	*x*	*y*	*z*	f_1	f_2	f_3	f_4	f_5
0	0	0	0	0	0	0	1	0	0	1	0
1	0	0	0	0	0	1	1	1	1	1	1
3	0	0	0	0	1	1	1	1	0	0	1
4	0	0	0	1	0	0	0	0	1	0	0
10	0	0	1	0	1	0	0	1	0	0	0
15	0	0	1	1	1	1	0	1	0	0	1
17	0	1	0	0	0	1	1	0	1	0	1
19	0	1	0	0	1	1	0	1	1	1	1
20	0	1	0	1	0	0	0	1	1	1	1
26	0	1	1	0	1	0	0	1	0	1	0
28	0	1	1	1	0	0	0	0	0	1	1
32	1	0	0	0	0	0	1	0	1	0	0
33	1	0	0	0	0	1	1	0	1	0	0
46	1	0	1	1	1	0	1	0	0	1	0

47	1	0	1	1	1	1	1	1	0	1	0
48	1	1	0	0	0	0	0	1	0	1	0
49	1	1	0	0	0	1	0	0	0	1	0
51	1	1	0	0	1	1	0	1	1	1	1
52	1	1	0	1	0	0	0	1	1	1	1
62	1	1	1	1	1	0	1	0	1	0	1
63	1	1	1	1	1	1	1	0	1	0	1

觀察上表可知，較低 4 位元之輸入變數 (表中 w、x、y 與 z 等 4 個變數) 之 16 個二進位組合，僅出現其中之 8 個組合 (十進位數為 0, 1, 3, 4, 10, 12, 14, 15 之部份)，故可先對這 8 個二進位組合進行編碼 (即指定 0→000、1→001、3→010、4→011、10→100、12→101、14→110 與 15→111) 後，即可得另一個真值表 (假設真值表之輸入為 w、x、y 與 z，而編碼後之輸出為 m_2、m_1 與 m_0) 如下表所示。

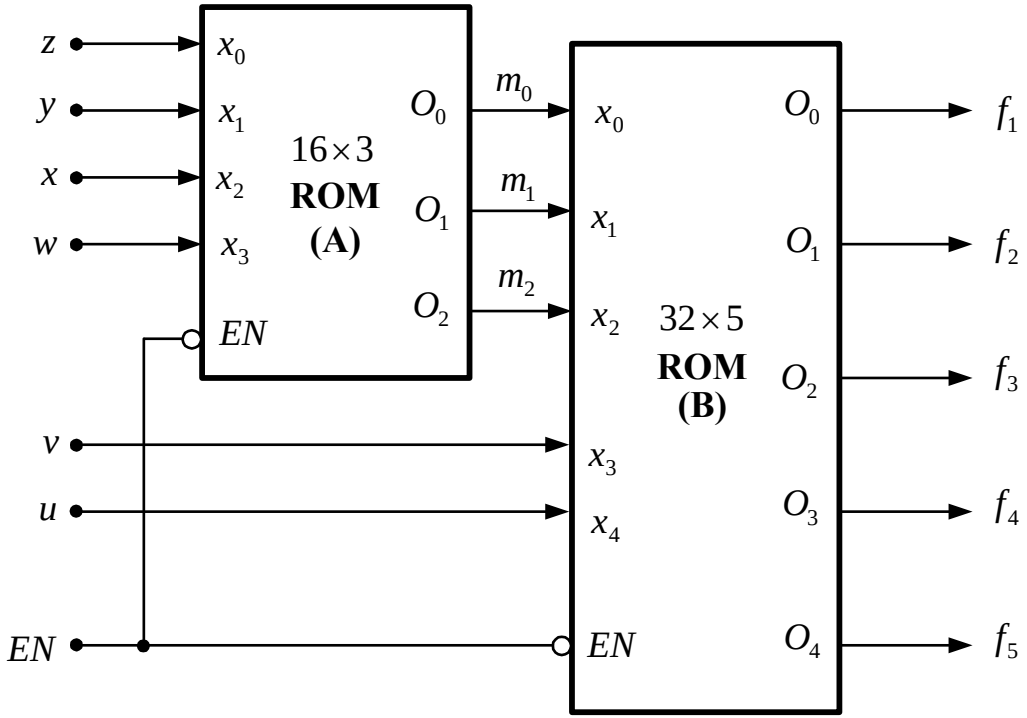
十進位數	輸　入				編碼輸出		
	w	x	y	z	m_2	m_1	m_0
0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1
3	0	0	1	1	0	1	0
4	0	1	0	0	0	1	1
10	1	0	1	0	1	0	0
12	1	1	0	0	1	0	1
14	1	1	1	0	1	1	0
15	1	1	1	1	1	1	1

觀察上表可知，首先使用 1 個16×3 *ROM* (*A*) 來實現這 4 個位元(表中 w、x、y 與 z 等 4 個變數)所代表之布林函數。接著將 2 個最高位元之輸入變數 (u 與 v) 與16×3 ROM (A) 之輸出 (左表之m_2、m_1與m_0) 結合成一個真值表，即此真值表包含 5 個變數 (u、v、m_2、m_1與m_0)，如下表所示 (m_2、m_1與m_0之二進位值，請對照表中之陰影部份，並配合左表之編碼輸出轉換而得)，故可再用 1 個32×5 *ROM* (*B*) 來實現所求之布林函數。

十進位數	輸入					輸出				
	u	v	m_2	m_1	m_0	f_1	f_2	f_3	f_4	f_5
0	0	0	0	0	0	1	0	0	1	0
1	0	0	0	0	1	1	1	1	1	1
3	0	0	0	1	0	1	1	0	0	1
4	0	0	0	1	1	0	0	1	0	0
10	0	0	1	0	0	0	1	0	0	0
15	0	0	1	1	1	0	1	0	0	1
17	0	1	0	0	1	1	0	1	0	1
19	0	1	0	1	0	0	1	1	1	1
20	0	1	0	1	1	0	1	1	1	1
26	0	1	1	0	0	0	1	0	1	0
28	0	1	1	0	1	0	0	0	1	1
32	1	0	0	0	0	1	0	1	0	0
33	1	0	0	0	1	1	0	1	0	0
46	1	0	1	1	0	1	0	0	1	0
47	1	0	1	1	1	1	1	0	1	0
48	1	1	0	0	0	0	1	0	1	0
49	1	1	0	0	1	0	0	0	1	0
51	1	1	0	1	0	0	1	1	1	1

52	1	1	0	1	1	0	1	1	1	1
62	1	1	1	1	0	1	0	1	0	1
63	1	1	1	1	1	1	0	1	0	1

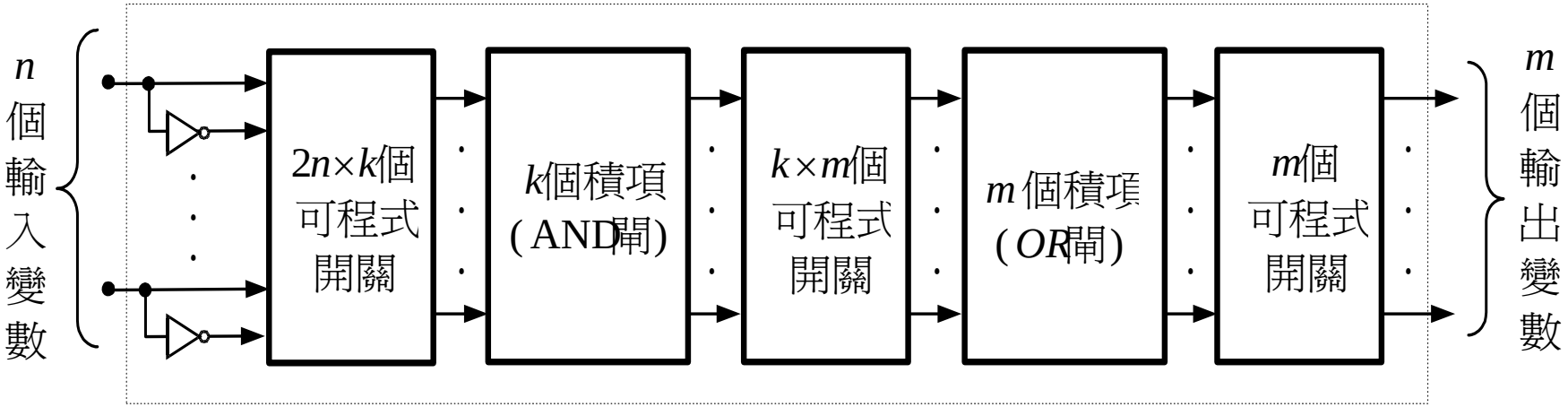
最後將16×3 ROM (*A*) 之 3 個輸出 (m_2、m_1與m_0) 連接至32×5 *ROM* (B) 較低 3 位元 (x_2、x_1與x_0) 之輸入線，而將 *u* 與 *v* 分別連接至 x_4與x_3，即使用 2 層之 *ROM* 來實現所求之布林函數，如下圖所示。



◆ 由前面之討論可知，若直接使用 1 個 64×5 *ROM* 來實現所求之布林函數，則所需 *ROM* 之容量為 $64 \times 5 = 320$ 個位元；而使用上圖之兩層 *ROM* (包括 16×3 與 32×5 之 *ROM* 各 1 個)，則所需 *ROM* 之總容量為 $16 \times 3 + 32 \times 5 = 208$ 個位元。因此使用 *ROM* 來實現輸入變數較多之布林函數，可考慮使用多層較小規格之 *ROM*，以較節省所需之邏輯元件。

可規劃之邏輯陣列

- 可規劃之邏輯陣列 (Programmable Logic Arrays; *PLA*) 所能實現之邏輯電路與唯讀記憶器 (*ROM*) 相似，即這兩種可規劃邏輯裝置，皆可用來實現積項之和 (*SOP*) 之布林函數式。
- 因可提供完全解碼之解碼器所需之邏輯元件與所實現之布林函數的輸入變數數目成指數函數之正比關係，因此對使用 *ROM* 來實現不採條件較多之布林函數 (即輸入變數之二進位組合較少) 而言，因有較多之位元 (解碼器之積項) 未被使用，所以會造成許多不必要之浪費，故會變得比較不經濟。
- 可規劃邏輯閘陣列 (*PLA*) 可用來實現不採條件較多之布林函數，而不會形成位元浪費之情況發生的可規劃邏輯裝置。因 *PLA* 在輸入變數－*AND* 閘之間與 *AND* 閘輸出－*OR* 輸入之閘，分別使用一個可規劃之連接方塊來指定彼此的連接關係，以實現所求積項之和的之布林函數，而一個可實現 n 個輸入變數與 m 個輸出變數之 *PLA* (所能實現之布林函數相當於容量 $2^n \times m$ 之 ROM) 的結構，所需可程式開關 (Programmable Switches) 為 $2n \times k + k \times m + m$ 個，如下圖所示。


n
個
輸
入
變
數
2n×k個
可程式
開關
k個積項
(AND閘)
k×m個
可程式
開關
m個積項
(OR閘)
m個
可程式
開關
m
個
輸
出
變
數

使用 PLA 實現邏輯電路

- 因 *PLA* 僅提供所求布林函數所需之積項，而不提供輸入變數之全部解碼工作，故對較少積項之布林函數而言，使用 *PLA* 實現會比 *ROM* 節省相當多之邏輯元件。
- 使用 *PLA* 來實現布林函需有兩次規劃步驟 (即需分別規劃積項與和項之連接開關)，故設計上比使用 *ROM* 複雜，而在設計 *PLA* 時，必須先化簡所求布林函數 (化簡原則為盡量尋找所求各個布林函數之公共項，而不是求取各個布林函數之最簡式)，求出最少之積項為原則，以減少 *PLA* 所需之邏輯元件 (即可減少 *PLA* 所需之容量)，接著製作一個 *PLA* 之規劃表，作為規劃連接方塊之開關的依據。

例題 11-6

試利用適當規格之 *PLA* 來實現 $f_1(x, y, z) = \sum(0, 1, 6, 7)$ 與 $f_2(x, y, z) = \sum(0, 1, 2, 6)$ 之布林函數。

解

利用卡諾圖化簡所求之布林函數，並盡量尋找所求布林函數之公共項後，所得之最簡布林函數為

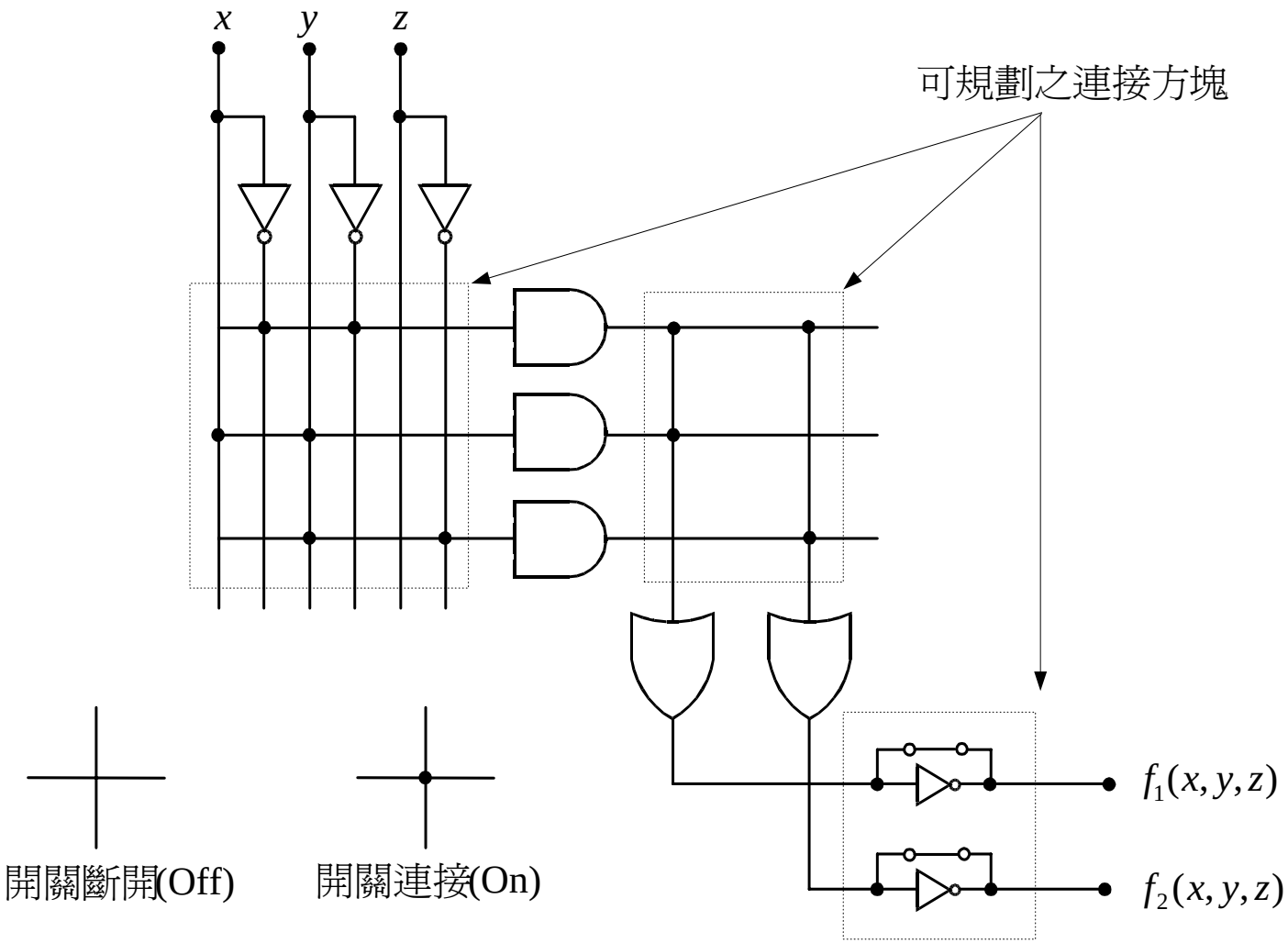
$$f_1(x, y, z) = \bar{x} \cdot \bar{y} + x \cdot y \qquad \text{與} \qquad f_2(x, y, z) = \bar{x} \cdot \bar{y} + y \cdot \bar{z}$$

觀察化簡所得之布林函數 $f_1(x, y, z)$ 與 $f_2(x, y, z)$ 可知，此兩個布林函數共有 $\bar{x} \cdot \bar{y}$、$x \cdot y$ 與 $y \cdot \bar{z}$ 等 3 個積項，接著利用化簡後之布林函數，製作一個 *PLA* 之規劃表，如下表所示。

積項		輸入			輸出	
		x	y	z	$f_1(x,y,z)$	$f_2(x,y,z)$
1	$\bar{x} \cdot \bar{y}$	0	0	–	1	1
2	$x \cdot y$	1	1	–	1	–
3	$y \cdot \bar{z}$	–	1	0	–	1
					T	T

上表中之輸入欄位之 0 與 1，可作為規劃輸入至 *AND* 閘與 *AND* 閘輸出至 *OR* 閘輸入間之可程式連接方塊之用途，即「0」表示指定相對變數之補數型態連接至 *AND* 閘之可程式開關必須指定為導通 (On)，而「1」指定相對指定變數之非補數型態連接至 *AND* 閘之可程式開關必須指定導通 (On)，其它未指定開關，皆指定為斷開 (Off)。

而「*T*」指定相對輸出函數 (*OR* 閘之輸出) 連接至 *PLA* 輸出的可程式開關必須指定導通 (On)；而「*C*」指定相對輸出函數 (*OR* 閘之輸出) 連接至 *PLA* 輸出的可程式開關，必須設定為斷開 (Off)。根據上面之討論，配合 *PLA* 結構圖，即可繪出可實現所求布林函數之 *PLA* 的詳細電路圖，如下圖所示。


x
y
z
可規劃之連接方塊
$f_1(x,y,z)$
$f_2(x,y,z)$
開關斷開(Off)
開關連接(On)

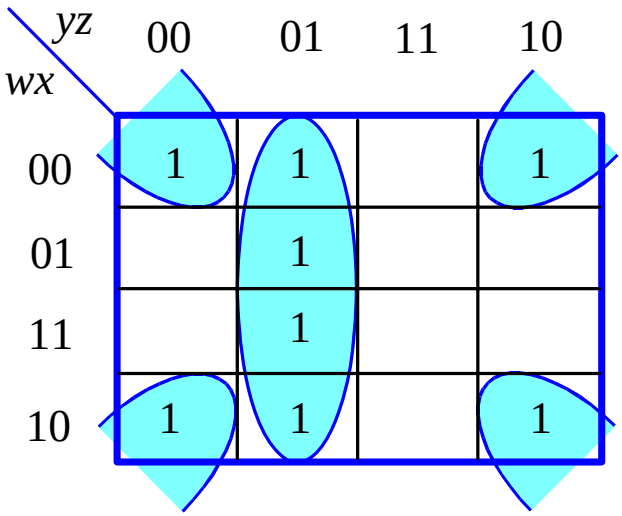
例題 11-7

試試利用適當規格之 *PLA* 來實現 $f_1(w, x, y, z) = \sum(0, 1, 2, 5, 8, 9, 10, 13)$ 與 $f_2(w, x, y, z) = \sum(3,4,6,7,12,14,15)$ 布林函數。

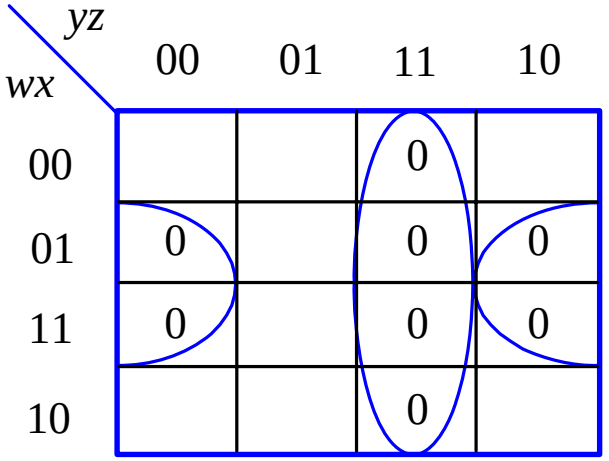
解

使用 *PLA* 來實現組合邏輯電路時，積項數目直接影響 *PLA* 之容量甚巨，故必需確定化簡所得之積項數目為最少，如此才可得到最佳化之 *PLA* 設計工作，因此須考慮所求布林函數之非補數與補數型態之化簡，以確保所得之積項為最少的求解方式。

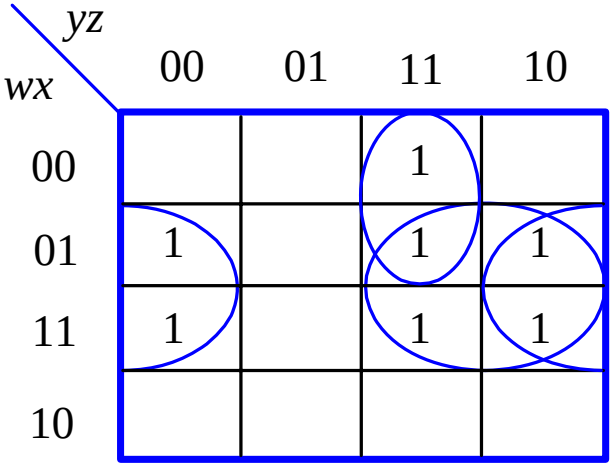
首先利用卡諾圖化簡所求之布林函數，並盡量尋找所求布林函數補數與非補數型態之卡諾圖，如下圖所示，以得到最多公共項之原則。



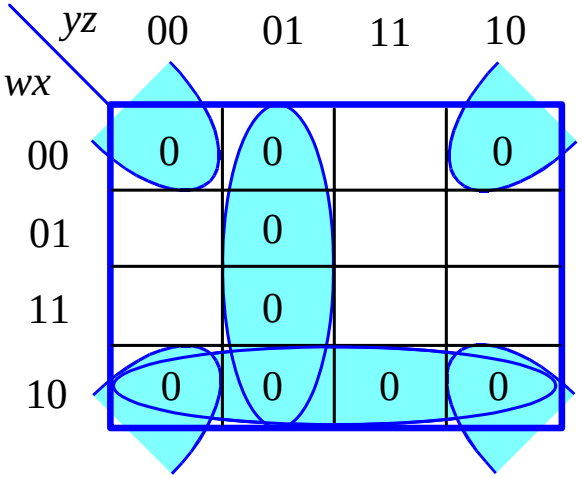
$$f_1(w, x\ y, z) = \overline{x} \cdot \overline{z} + \overline{y} \cdot z$$



$$\overline{f_1(w, x\ y, z)} = x \cdot \overline{z} + y \cdot z$$



$$f_2(w, x\ y, z) = x \cdot \overline{z} + x \cdot y + \overline{w} \cdot y \cdot z$$



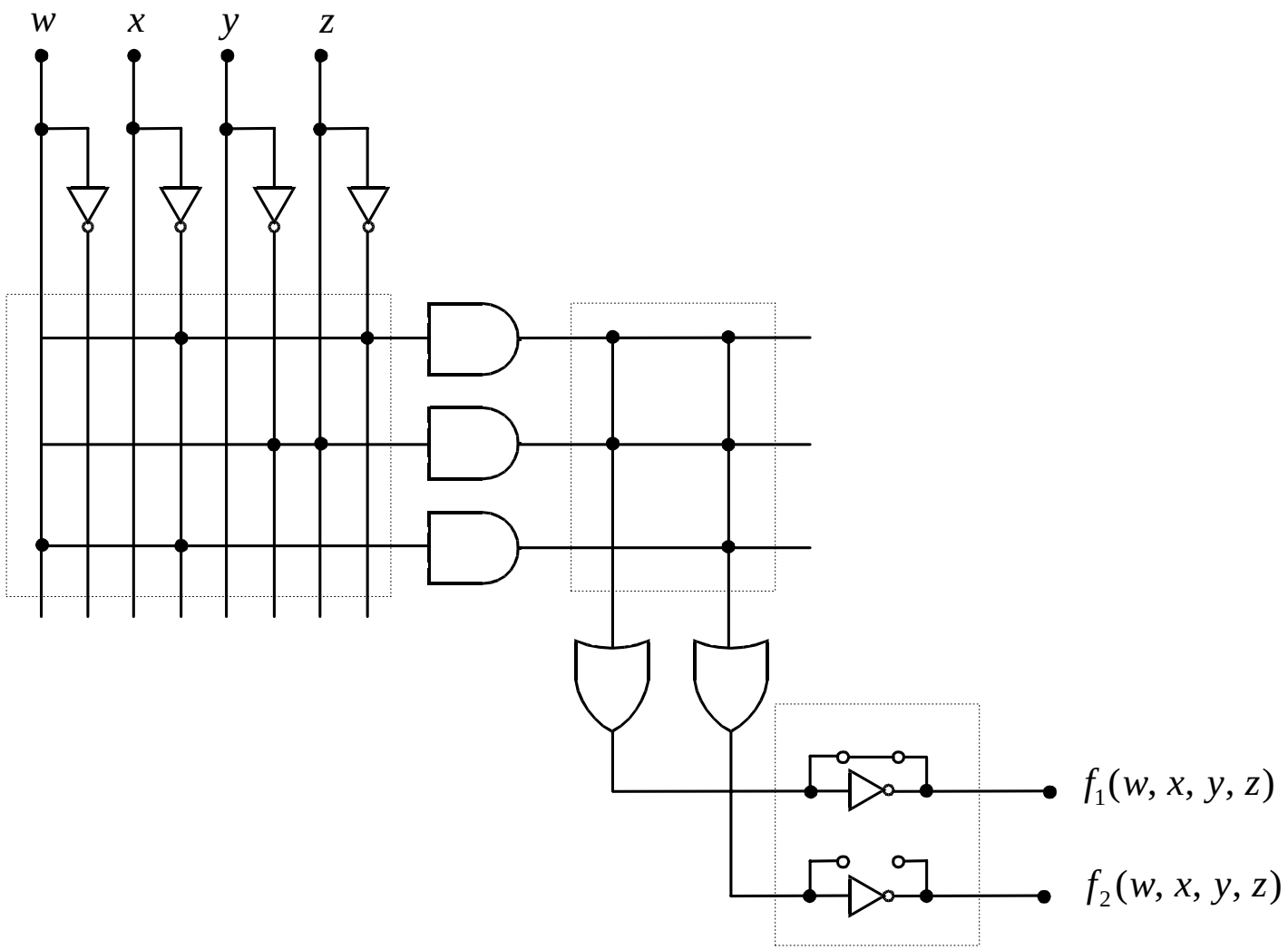
$$\overline{f_2(w, x\ y, z)} = \overline{x} \cdot \overline{z} + \overline{y} \cdot z + w \cdot \overline{x}$$

觀察上圖可知，選擇所求布林函數補數與非補數型態，以得到最少積項之方式，計有 $f_1 - f_2$ 、$\overline{f_1} - f_2$ 、 $f_1 - \overline{f_2}$ 與 $\overline{f_1} - \overline{f_2}$ 等 4 種選擇方式，而以選取 $f_1(w, x, y, z) = \bar{x} \cdot \bar{z} + \bar{y} \cdot z$ 與 $\overline{f_2(w, x, y, z)} = \bar{x} \cdot \bar{z} + \bar{y} \cdot z + w \cdot \bar{x}$ 具有最少之積項，即此兩化簡後之布林函數僅有 $\bar{x} \cdot \bar{z}$ 、 $\bar{y} \cdot z$ 與 $w \cdot \bar{x}$ 等 3 個積項。

接著利用上述化簡後之布林函數，製作一個 *PLA* 之規劃表，如下表所示。

積項		輸入				輸出	
		w	x	y	z	$f_1(w, x, y, z)$	$f_2(w, x, y, z)$
1	$\bar{x} \cdot \bar{z}$	–	0	–	0	1	1
2	$\bar{y} \cdot z$	–	–	0	1	1	1
3	$w \cdot \bar{x}$	1	0	–	–	–	1
						T	C

最後根據上面之討論，配合 *PLA* 結構圖，即可繪出可實現所求布林函數之 *PLA* 的詳細電路圖，如下圖示。



使用 ROM 與 PLA 實現邏輯電路之比較

- 若使用 *ROM* 來實現例題 11-6 之布林函數，對 3 個輸入變數之完全解碼，需使用 8 個積項 ($2^3 \times 2 = 16$ 位元)，而使用 *PLA* 來實現相同之布林函數，僅需 3 個積項即可 ($3 \times 2 = 6$ 位元)，故使用 *PLA* 可節省許多邏輯元件
- 若使用 *ROM* 來實現例題 11-7 之布林函數，對 4 個輸入變數之完全解碼，需使用 16 個積項 ($2^4 \times 2 = 32$ 位元)，而使用 *PLA* 來實現相同之布林函數，僅需 3 個積項即可 ($4 \times 2 = 8$ 位元)，故使用 *PLA* 可節省更多邏輯元件。
- 但 *PLA* 額外增加輸入變數至 *AND* 閘間之可程式連接方塊，此連接方塊亦會佔用一些空間，故 *PLA* 比較適合用來實現積項較少之組合邏輯電路，而對積項較多之邏輯電路，則建議採用 *ROM* 實現較為經濟。

現場可規劃之邏輯閘陣列

- 因數位系統複雜度逐漸增加，造成電子科技之蓬勃發展，而前面兩節所討論之 *ROM* 與 *PLA* 等可規劃邏輯裝置，所能實現之邏輯函數，已無法滿足實際之需求。
- 在 1985 由 Xilinx 公司發展出一種可實現較多邏輯函數之可規劃邏輯裝置，稱為現場可規劃之邏輯閘陣列 (Field Programmable Gate Arrays; *FPGA*) 為本節討論之重點。
- 目前單一顆現場可規劃邏輯閘陣列可實現之邏輯函數，已超過一百萬個等效邏輯閘 (Gate Count)，因此 *FPGA* 已是特殊應用積體電路 (Application Specify Integrated Circuits; *ASIC*) 之一種新的趨勢。
- 因 *FPGA* 具有現場可規劃之特性，故可立即用來實現許多複雜之數位電路，而不必像一般數位 IC，需經過製作光罩 (Mask) 之程序，故可大幅減少電路設計之時間與成本，即可以滿足低風險、低成本及簡易設計變更等要求，尤其對少量而多變之產品，更可節省庫存之壓力，因此目前 *FPGA* 已被普遍使用於數位積體電路之設計上。

FPGA 之電路結構

◆ *FPGA* 是由許多可規劃之邏輯方塊 (Configuration Logic Blocks; *CLB*)、開關方塊 (Switch Blocks) 與輸入 / 輸出方塊 (I/O Blocks) 所組成之超大型積體電路 (Very Large Scale Integrated Circuits; *VLSI*)。

1. 可規劃之邏輯方塊 (*CLB*) 可依使用者之需要，提供實際所需之邏輯函數，若在 *CLB* 加上正反器 (Flip-Flop)，則 *FPGA* 亦可用來實現序向邏輯電路。

2. 可規劃之開關方塊，可用來連接許多 *CLB* 與輸入 / 輸出方塊間之繞線路徑 (Routing Paths)，以構成一個完整之數位系統。

3. 輸入 / 輸出方塊可提供 *FPGA* 之內部訊號與外部訊號的連接。

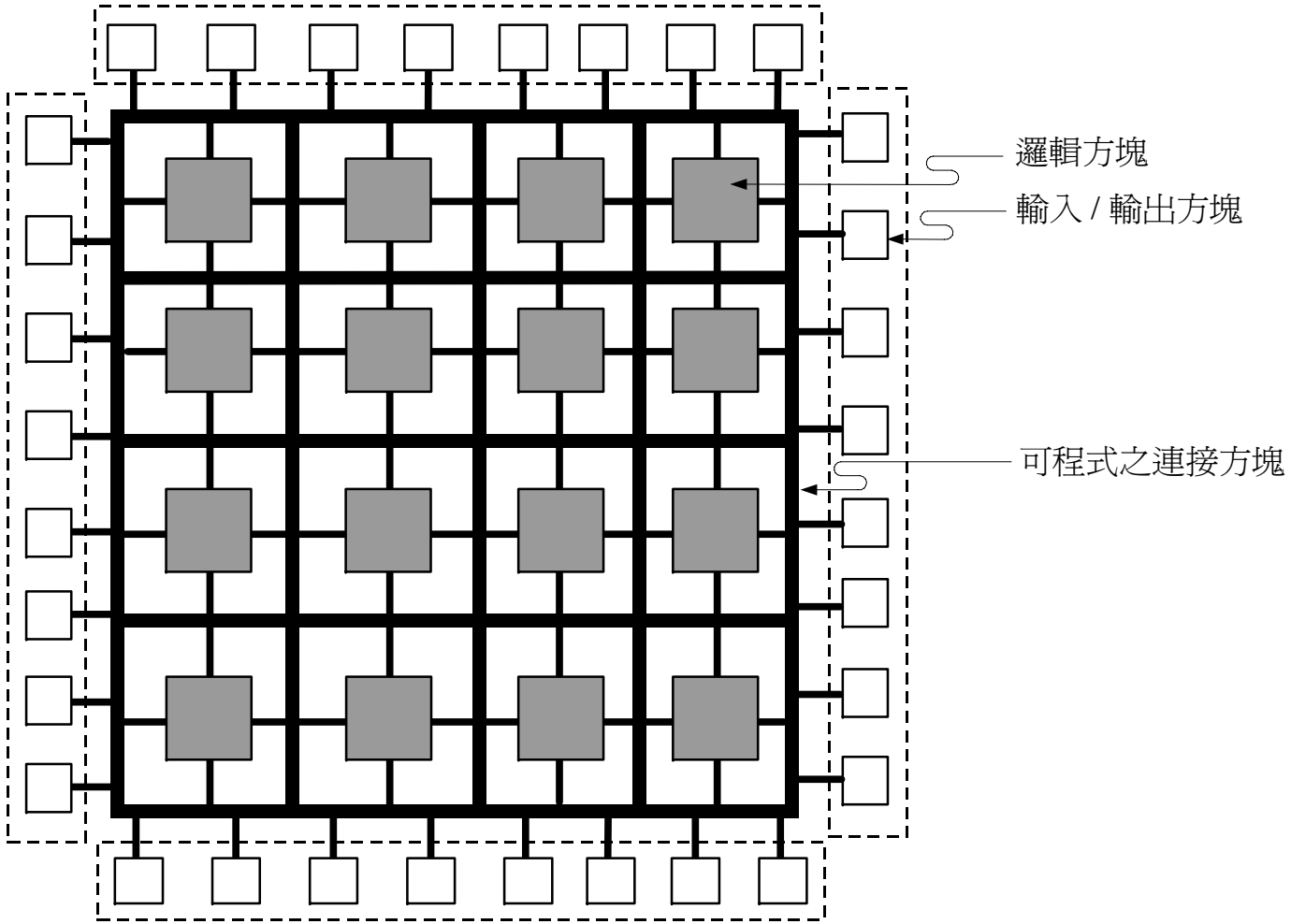
◆ 由 *FPGA* 之基本結構可知，使用者可藉由規劃 (Programming) 晶片，自行定義 *FPGA* 所執行之邏輯函數，並可隨時依實際需要加以修改，以實現任何數位系統之超大型積體電路，故它同時結合客戶定作 (Custom Design) 與標準元件 (Standard Cell) 之 *VLSI* 設計的優點，因此 *FPGA* 愈來愈受到數位系統設計者之重視，已是目前最重要之可規劃邏輯裝置。

- 雖然 *FPGA* 邏輯運算功能，可隨時依實際需要加以修改，以實現任何數位系統，但因開關數量與訊號通過繞線路徑所需之開關數目較多，使得 *FPGA* 之邏輯密度 (Density)與執行速度 (Speed) 僅分別為傳統 *ASIC* 之 $\frac{1}{10}$ 與 $\frac{1}{3}$，故在實際使用上受到相當大之限制。

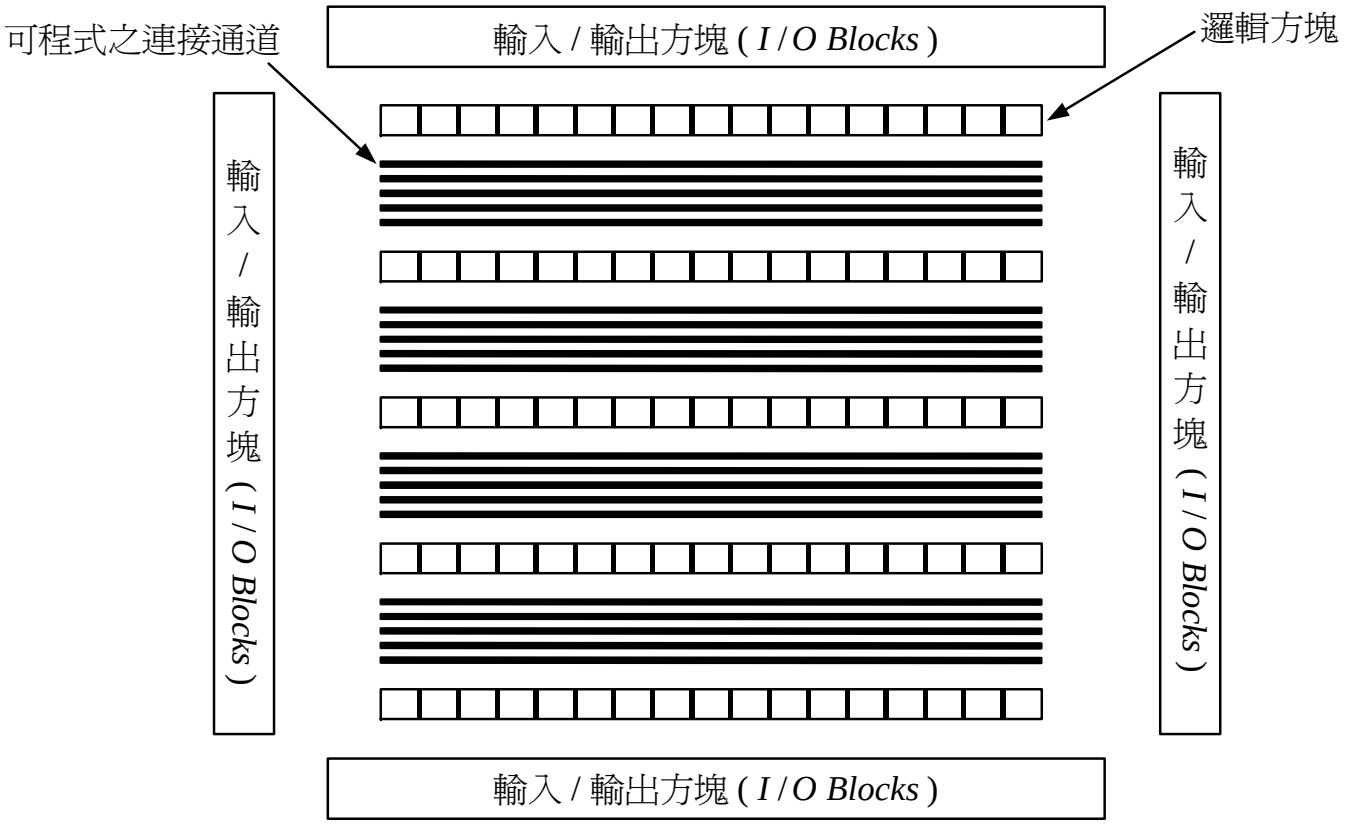
- 因 *FPGA* 之發展愈來愈受到重視，導致製造公司亦在持續成長中，而目前以 Xilinx 公司所開發之 *FPGA* 晶片 (以對稱式連接架構為主) 與 Altera 公司所開發之 *CPLD* (Complex PLD) 晶片 (列－排列式連接架構為主)，可容納較高等效邏輯閘數 (Gate Count) 與具備較為完備發展工具 (Development Tools)，故較常用來實現較複雜之數位電路。

- 為了克服邏輯密度 (Density)與執行速度 (Speed)低之缺點，許多不同 *FPGA* 之連接架構陸續被發展出來，而目前以對稱式陣列 (Symmetrical Arrays)、列－排列式 (Row-based) 與階層式 (Hierarchical) 等 3 種連接架構較為實用，如下圖所示。

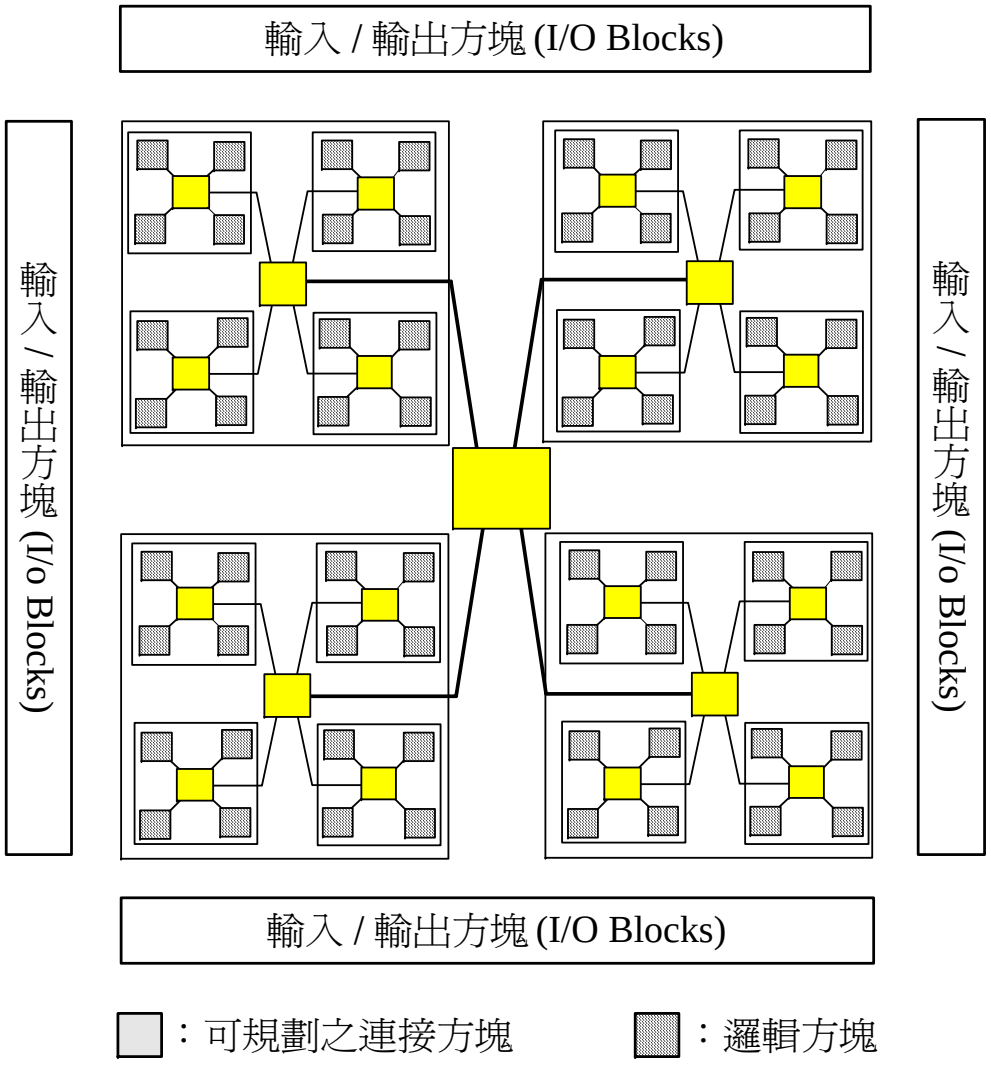
(a) 對稱式 (Symmetrical Arrays) 連接架構



(b) 列－排列式 (Row-based) 連接架構



(c) 階層式 (Hierarchical) 連接架構

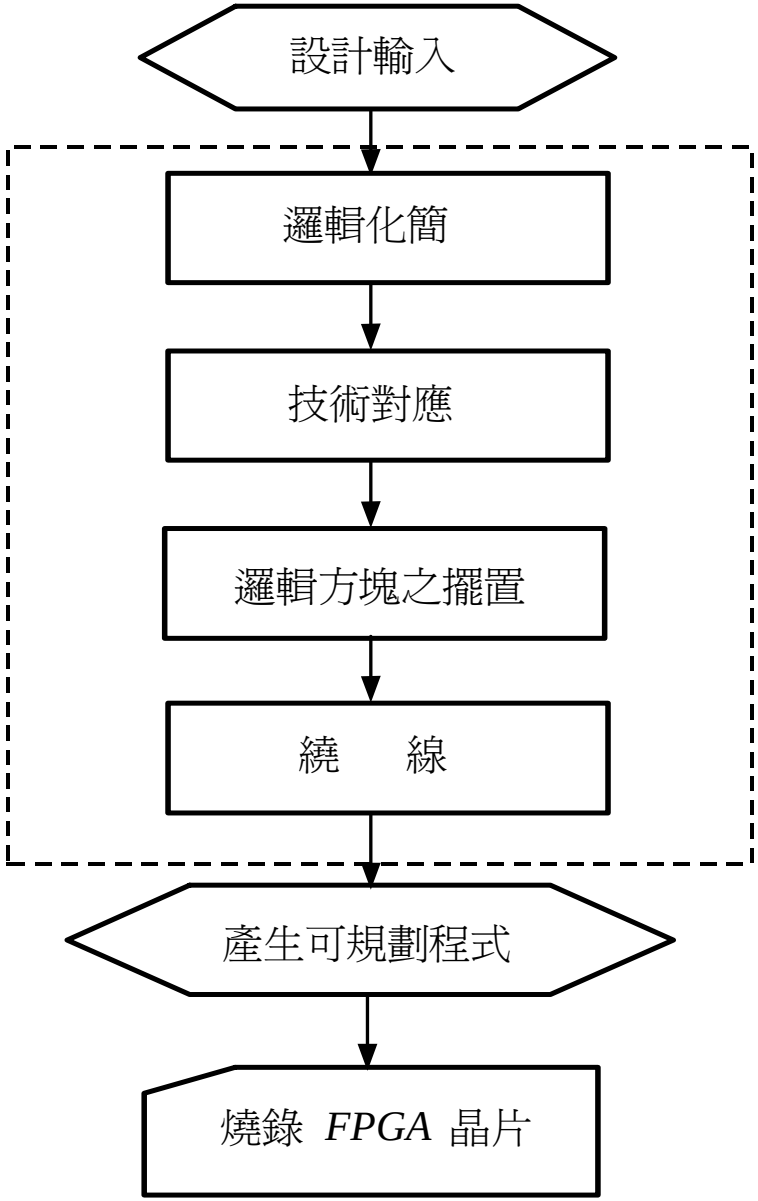


◆ 階層式連接架構是將邏輯方塊與邏輯方塊之連接路徑改用分散式並聯方式來完成，即於階層性連接架構下之每一層，皆是用開關方塊來連接數個邏輯方塊或邏輯方塊組（ 上圖以 4 個邏輯方塊為一個邏輯方塊組 ），直到組成一個完整之可規劃邏輯陣列為止，故此種連接架構之特性可大量減少訊號通過繞線路徑所須經過之電晶體數目，故其執行速度會提高，且所需之總電晶體數目比亦較少，因此其邏輯密度亦相對會提高。

使用 FPGA 實現邏輯電路

- 不論使用 *FPGA* 或 *CPLD* 晶片，皆可實現相當複雜之數位系統，因此佈局設計 (Layout Design) 工作已非人力所能完成，故採用電腦輔助設計 (Computer Aided Design) 已是一種必然之趨勢。

- 實現 *FPGA* 佈局設計自動化之設計步驟，包括邏輯化簡 (Logic Minimization)、技術應對 (Technology Mapping)、邏輯方之擺置 (Placement)、全域繞線 (Global Routing) 與細部繞線 (Detailed Routing) 等電腦輔助設計演算法，接著列出 *FPGA* 佈局設計流程，如下圖所示。


設計輸入
邏輯化簡
技術對應
邏輯方塊之擺置
繞　　線
產生可規劃程式
燒錄 FPGA 晶片

◆ 實現 *FPGA* 佈局設計自動化設計之步驟於后（ 邏輯化簡之方法與已在本書第 3 章作過詳細討論)：

1. 技術應對 (Technology Mapping)：將多階數位電路進行轉換，以分割為許多邏輯方塊可執行之等效邏輯函數的過程。而如何分割後之減少總繞線長度 (Total Routing Length) 與邏輯方塊的數目 (Total Number of Logic Blocks)，以提高 *FPGA* 之執行速度與邏輯密度是技術對應演算法所追求之主要目標。

2. 邏輯方塊擺置 (Placement)：當完成技術對應後，需將分割所得之邏輯方塊分別擺置於 *FPGA* 中的實際邏輯方塊之相對位置上，而將彼此連接愈多之邏輯方塊盡量擺置在接近之位置，且使擺置後之結果，邏輯方塊間連接線為最少，以盡量減少繞線通道 (Routing Channel) 所須之連線 (Tracks) 數目為基本原則。

3. 全域繞線 (Global Routing)：依據邏輯方塊擺置後之結果，對每個繞線通道所需通過的導線 (net) 數作平均分配，以考慮繞線通道 (Routing Channel) 之平衡，使細部繞線 (Detailed Routing) 工作更容易進行為基本原則。

4. **細部繞線 (Detailed Routing)**：根據全域繞線後之結果，考慮如何有效利用**既有之繞線資源** (Routing Resources)，以**指定每一個導線** (net) 所佔用之**連線** (Track) **位置**，而以可達**百分之百的繞線率**為基本原則。