

微算機系統 Fall 2020

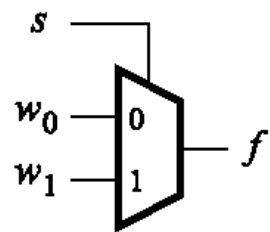
Microprocessor Systems

Instructor : Yen-Lin Chen(陳彥霖), Ph.D. Professor
Dept. Computer Science and Information Engineering
National Taipei University of Technology

一、多工器

MULTIPLEXERS

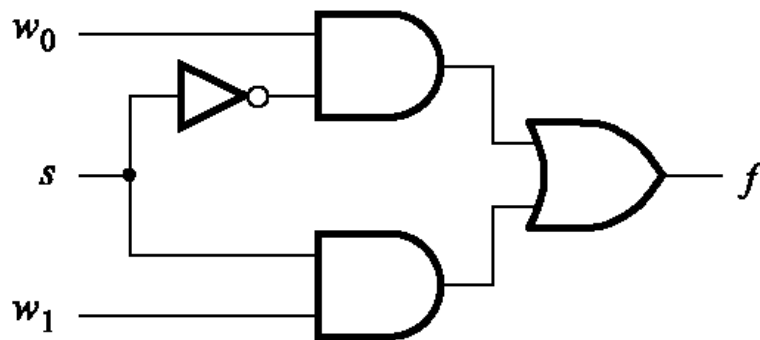
- 多工器電路有許多資料輸入，一個或多個選擇輸入，以及一個輸出。它將其中一個資料輸入傳遞到輸出，此資料輸入由選擇輸入的值所決定。
- 圖4.1為二對一多工器。
 - (a) 部分為常用的符號。選擇 (select) 輸入 s 選擇 w_0 或 w_1 成為多工器的輸出。
 - (b) 部分真值表描述多工器的功能。
 - (c) 部分為二對一多工器的乘積總和實作。
 - (d) 部分說明如何以傳輸閘建構而成。



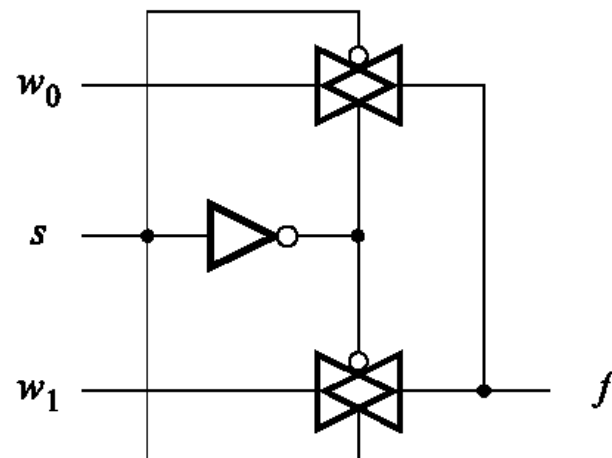
(a) 圖形符號

s	f
0	w_0
1	w_1

(b) 真值表



(c) 乘積總和電路



(d) 運用傳輸閘的電路

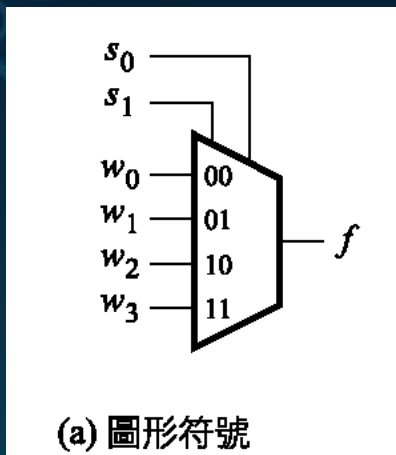
圖3.1 二對一多工器。

MULTIPLEXERS (CONT'D)

- 圖3.2a為較大的多工器，四個資料輸入 w_0 、 w_1 、 w_2 和 w_3 ，兩個選擇輸入 s_0 和 s_1 。
- 圖中(b)部分的真值表，二位元數值 $s_1 s_0$ 選擇其中一個資料輸入作為多工器輸出。
- 四對一多工器的乘積總和實作如圖3.2c所示。它實現下列多工器函數

$$f = \overline{s_1}\overline{s_0}w_0 + \overline{s_1}s_0w_1 + s_1\overline{s_0}w_2 + s_1s_0w_3$$

MULTIPLEXERS (CONT'D)



s_1	s_0	f
0	0	w_0
0	1	w_1
1	0	w_2
1	1	w_3

(b) 真值表

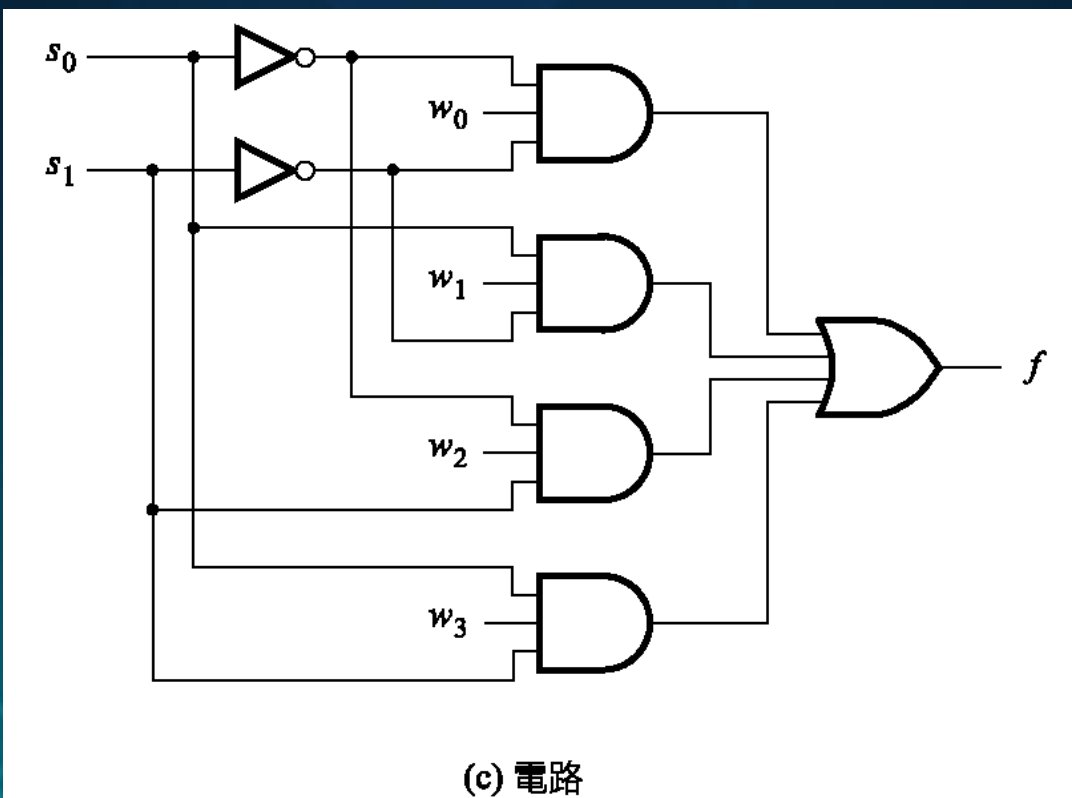


圖3.2 四對一多工器。

MULTIPLEXERS (CONT'D)

- 較大的多工器也可以由小型多工器組成：
 1. 如圖3.3所示，四對一多工器可以由三個二對一多工器組成。
 2. 圖3.4是以五個四對一多工器構成十六對一多工器。

MULTIPLEXERS (CONT'D)

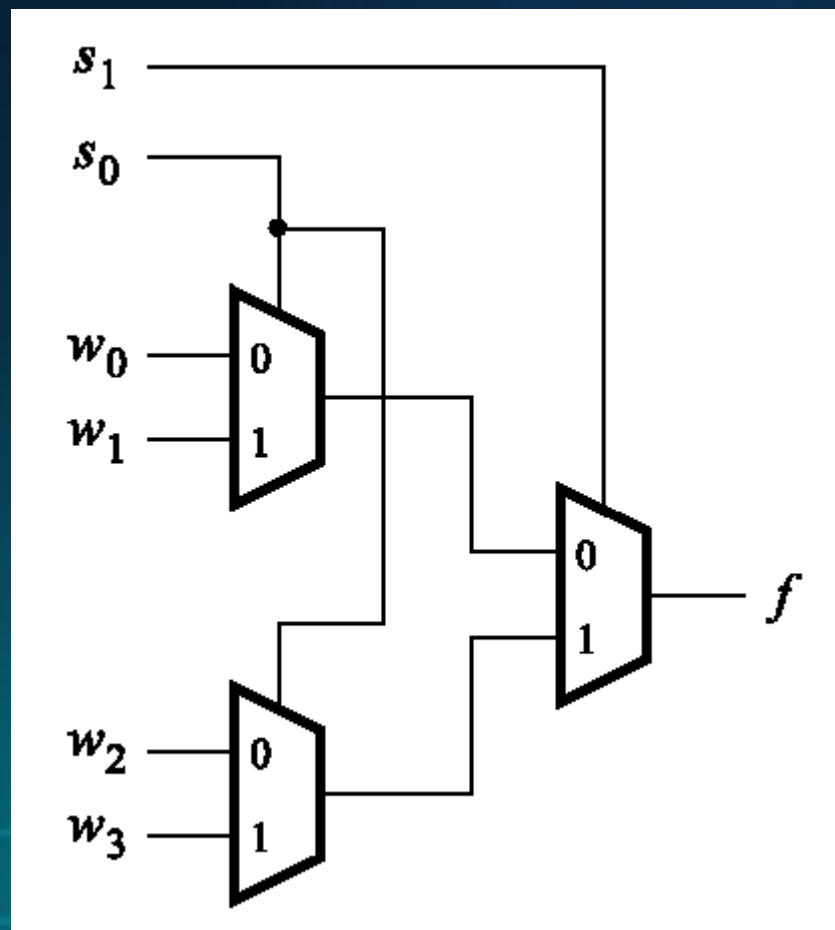


圖3.3 以二對一多工器構成四對一多工器。

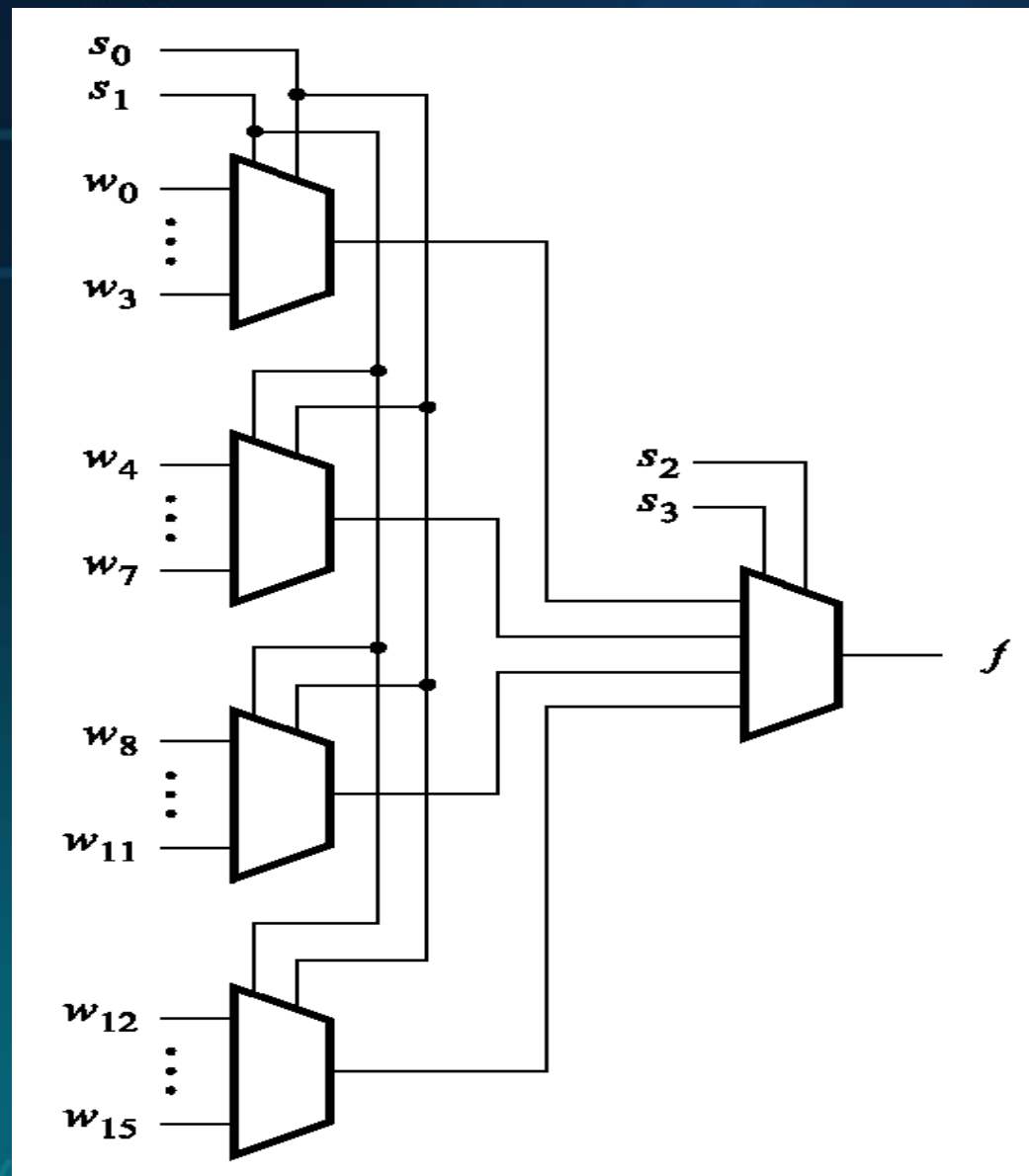


圖3.4 十六對一多工器。

範例一 交叉開關_(CROSSBAR SWITCH)

- 圖3.5的電路有兩個輸入 x_1 和 x_2 ，以及兩個輸出 y_1 和 y_2 。如同圖中的紫線，此電路的函數透過輸入 s 的控制，唯一功能為允許任一輸入連接到任一輸出，通常稱為交叉開關(crossbar switch)。
- 交叉開關有各種尺寸(n 個輸入和 k 個輸出)，也就是各種不同的輸入和輸出個數。
- 下圖為有兩個輸入和兩個輸出交叉開關。

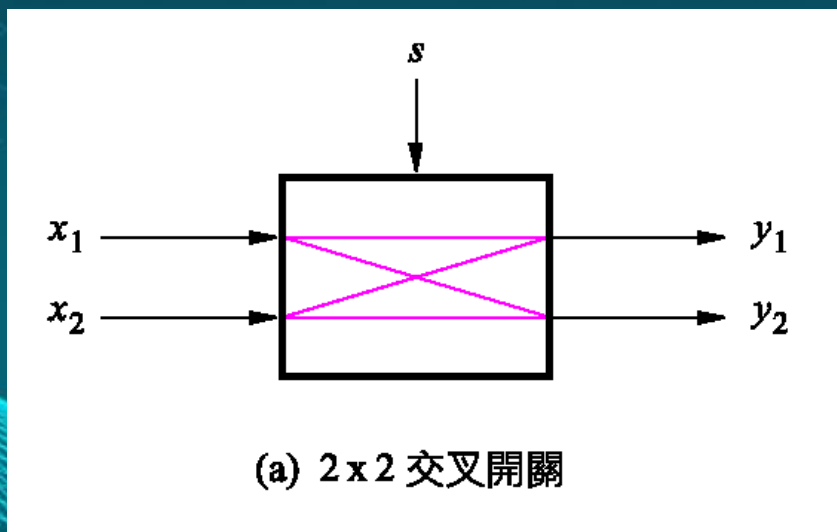
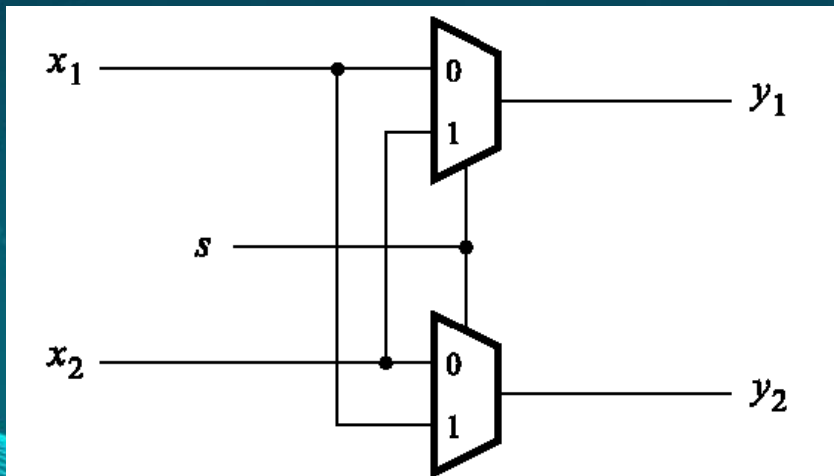


圖3.5 多工器的實際應用。

範例一 交叉開關_(CROSSBAR SWITCH)

- 圖3.5b顯示如何以二對一多工器實作 2×2 交叉開關。多工器的選擇輸入由訊號 s 控制。
 - 若 $s = 0$ ，交叉開關將 x_1 連到 y_1 ，將 x_2 連到 y_2 。
 - 若 $s = 1$ ，交叉開關將 x_1 連到 y_2 ，將 x_2 連到 y_1 。
- 交叉開關在很多實際應用是相當有用的，連接一組接線到另一組接線，而這些連接樣式會隨時間改變。



(b) 以多工器實作

圖3.5 多工器的實際應用。

範例二 FPGA 實作可程式開關

- FPGA 中的邏輯區塊有兩個輸入，每個繞線通道有四個軌道。將邏輯區塊的輸入或輸出連接到互連接線的每個可程式開關都以表示。圖3.6a顯示單一邏輯區塊和連接其輸出的互連接線與開關。

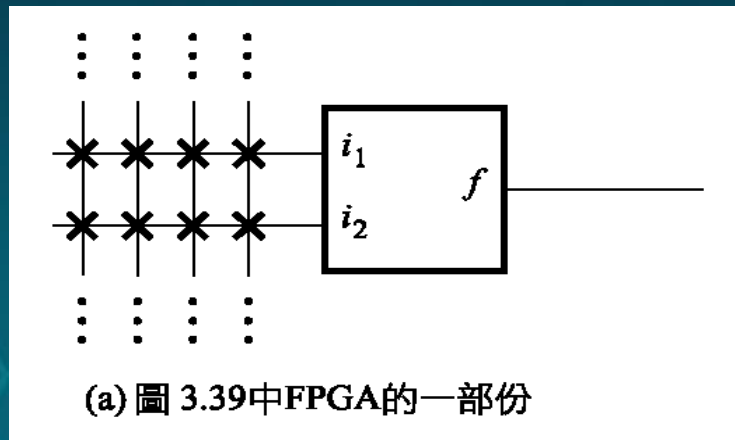


圖3.6 實作可程式開關於FPGA。

範例二 FPGA 實作可程式開關

- 圖3.6b為一種實作可程式開關的方法。圖中(a)部分的每個都以儲存單元控制的NMOS電晶體實作。
- 每個單元都儲存單一邏輯值，0或1，且提供此值作為該單元的輸出。
- 每個儲存單元都是由數個電晶體所構成。因此圖中的八個單元使用相當大的晶片面積。所需的儲存單元個數可以用多工器來減少，如圖3.6c所示。

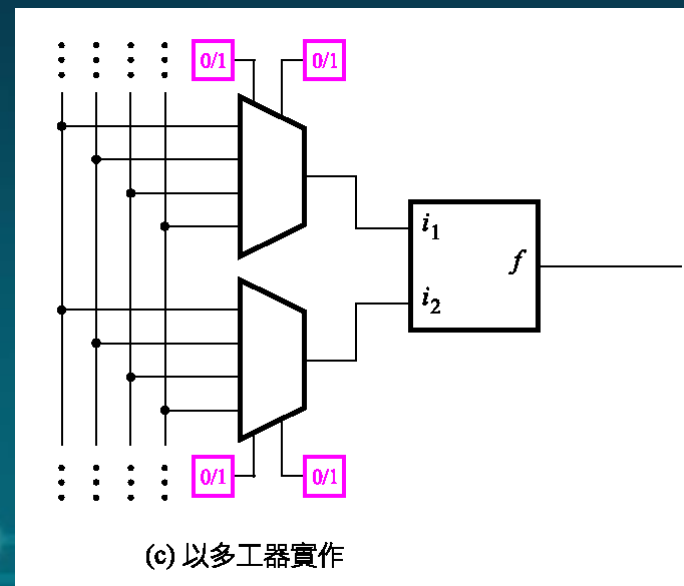
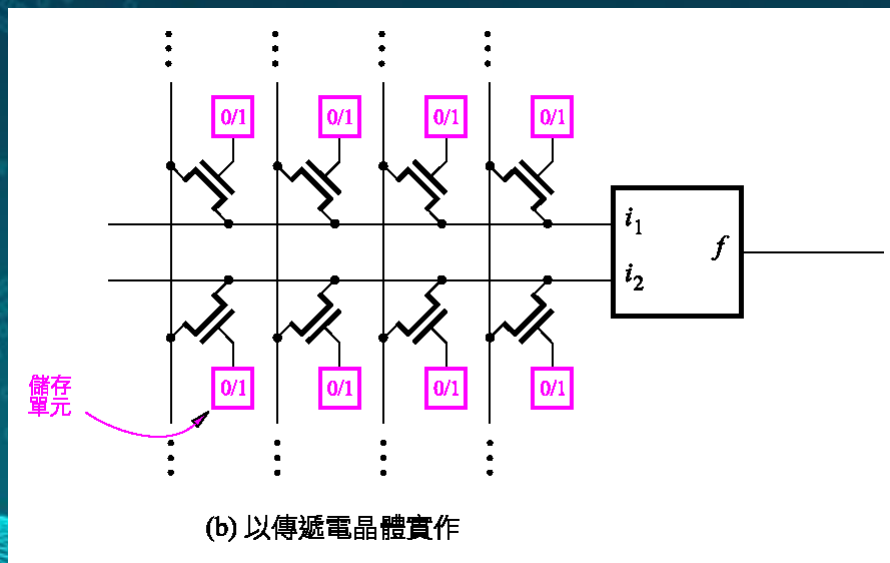


圖3.6 實作可程式開關於FPGA。

二、解碼器 由簡化繁 DECODERS

- 解碼器電路是用來將編碼資訊解碼。
- 圖3.7的二進位解碼器 n 個輸入和 2^n 個輸出的邏輯電路。
- 一次只有一個位元被設為1的 n 位元二進位編碼稱為單一熱門編碼 (one-hot encoded)，表示被設為1的位元很『熱門』。
- 圖3.8為二對四解碼器。
- 圖3.9是以兩個二對四解碼器建構三對八解碼器。
- 圖3.10是以五個二對四解碼器建構四對十六解碼器。因為其架構為樹狀，這種電路稱為解碼器樹 (decoder tree)。

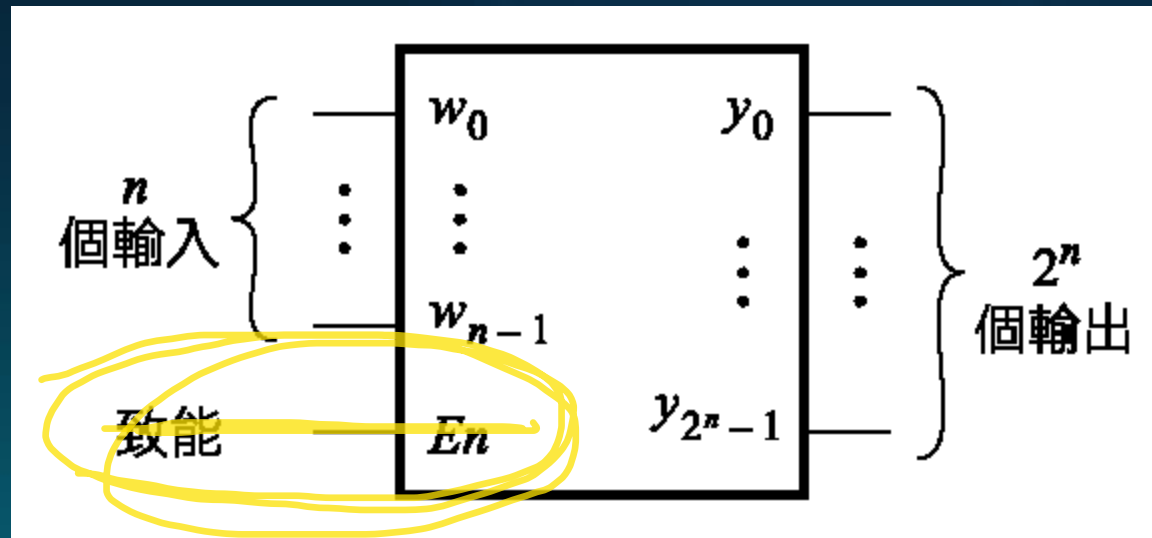
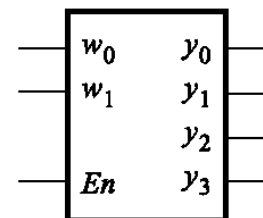


圖3.7 二進位解碼器。

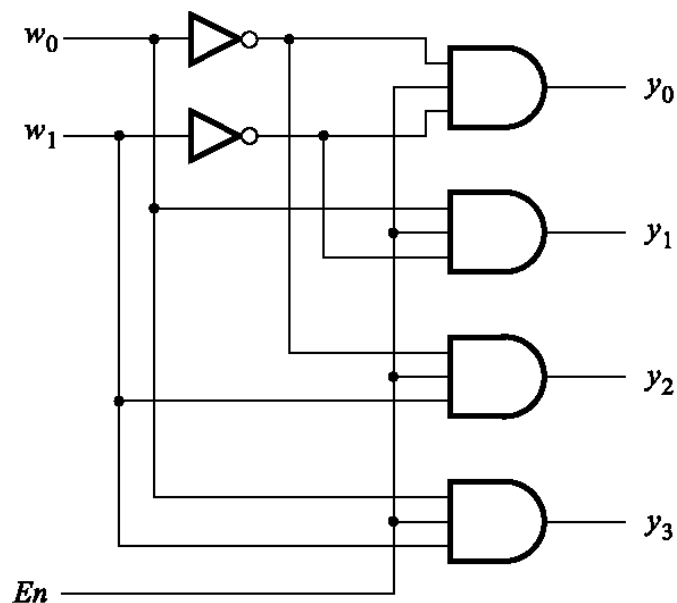
enable 要是“1”
才会運作

En	w_1	w_0	y_0	y_1	y_2	y_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0



(a) 真值表

(b) 圖形符號



(c) 邏輯電路

圖3.8 二對四解碼器。

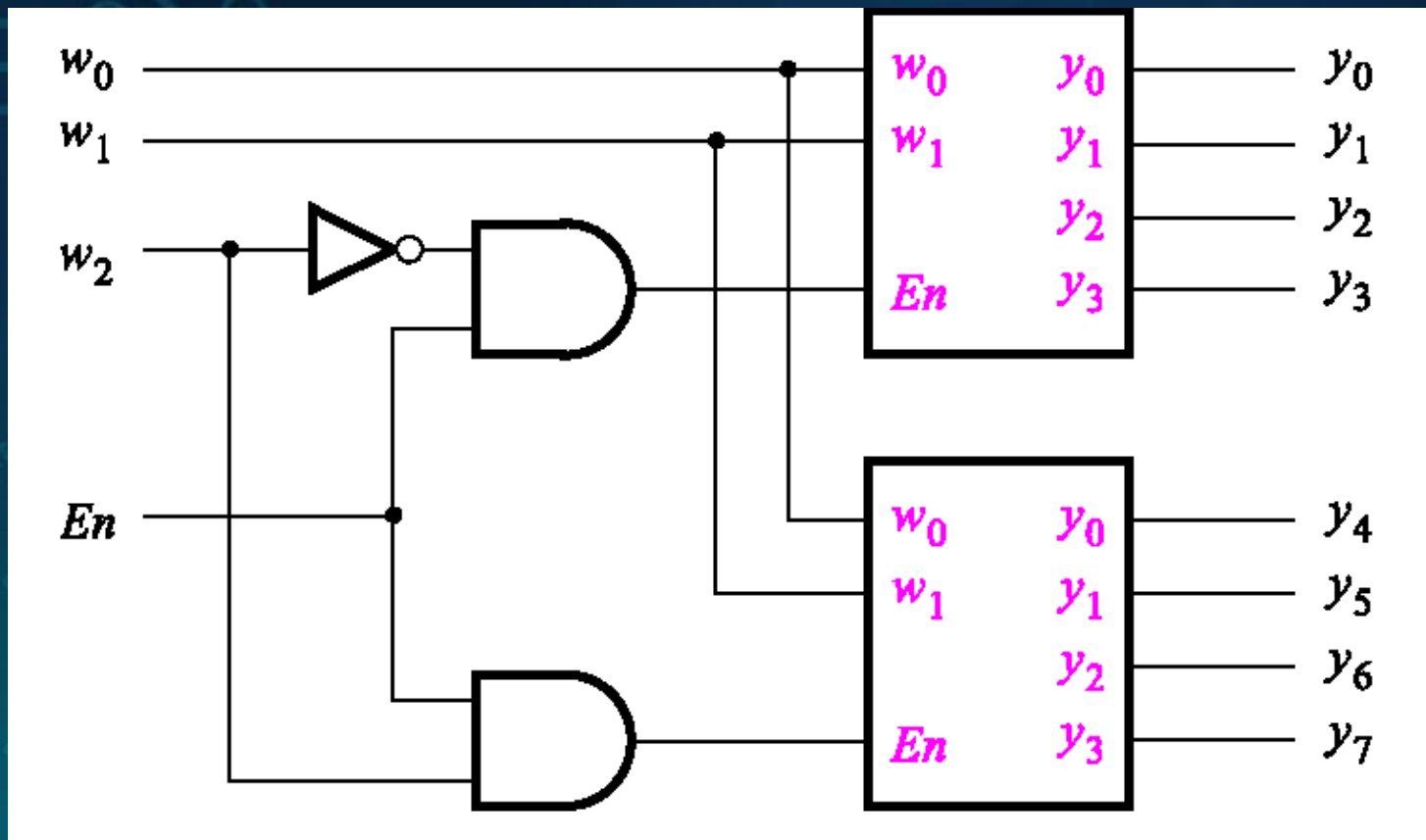


圖3.9 以兩個二對四解碼器建構三對八解碼器。

51個
2→4

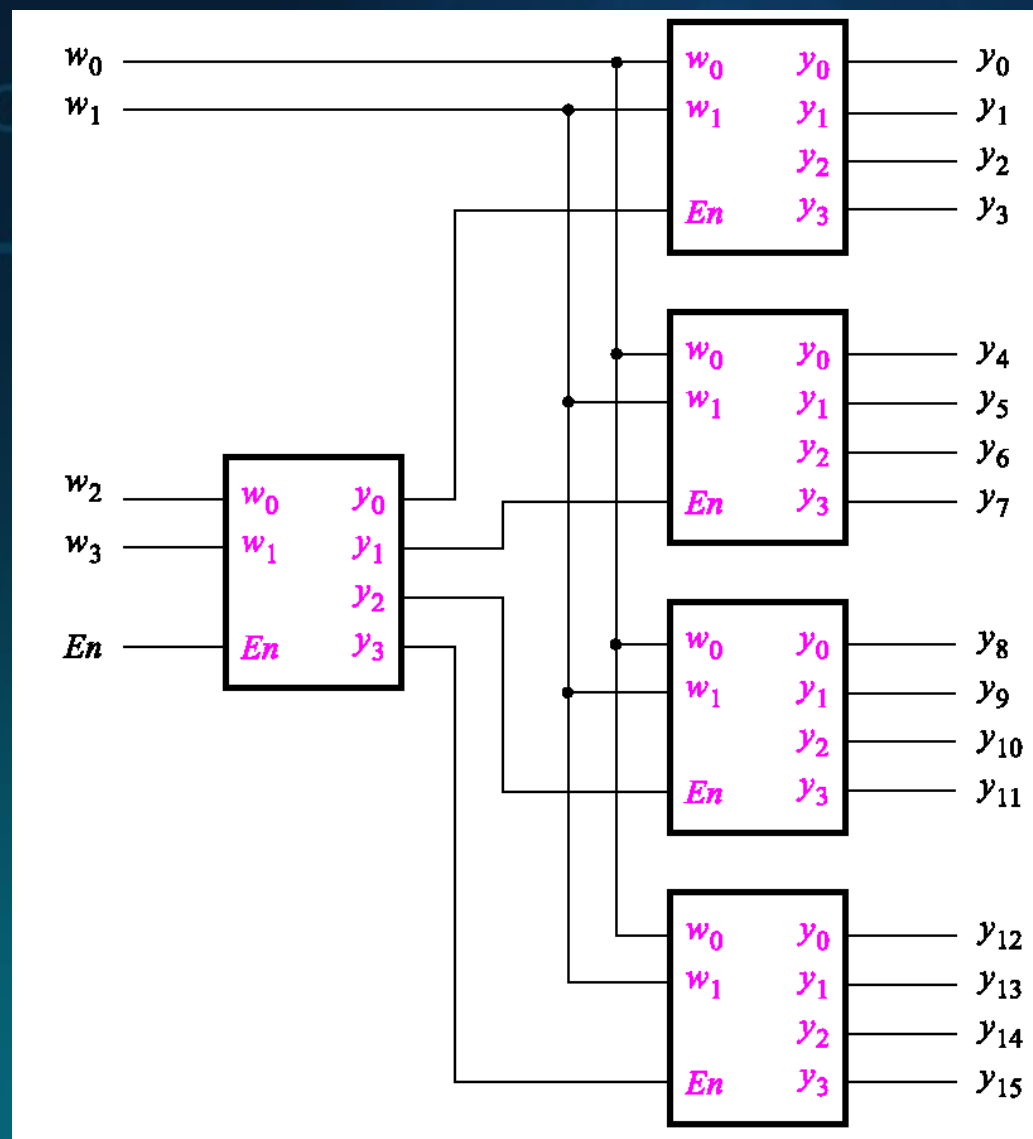


圖3.10 以解碼器樹(decoder tree) 建構四對十六解碼器。

範例三 解碼器建構四對一多工器

- 解碼器在許多實際用途上相當有用。圖3.2c為四對一多工器的乘積總和實作，需要AND邏輯閘將輸入 s_1 和 s_0 的四個不同估值分開。
- 因為解碼器對其輸入作估值，所以可用來建構多工器。如圖3.11所示。此例中解碼器的致能訊號是不需要的，被設為1。解碼器的四個輸出代表選擇輸入的四個估值。

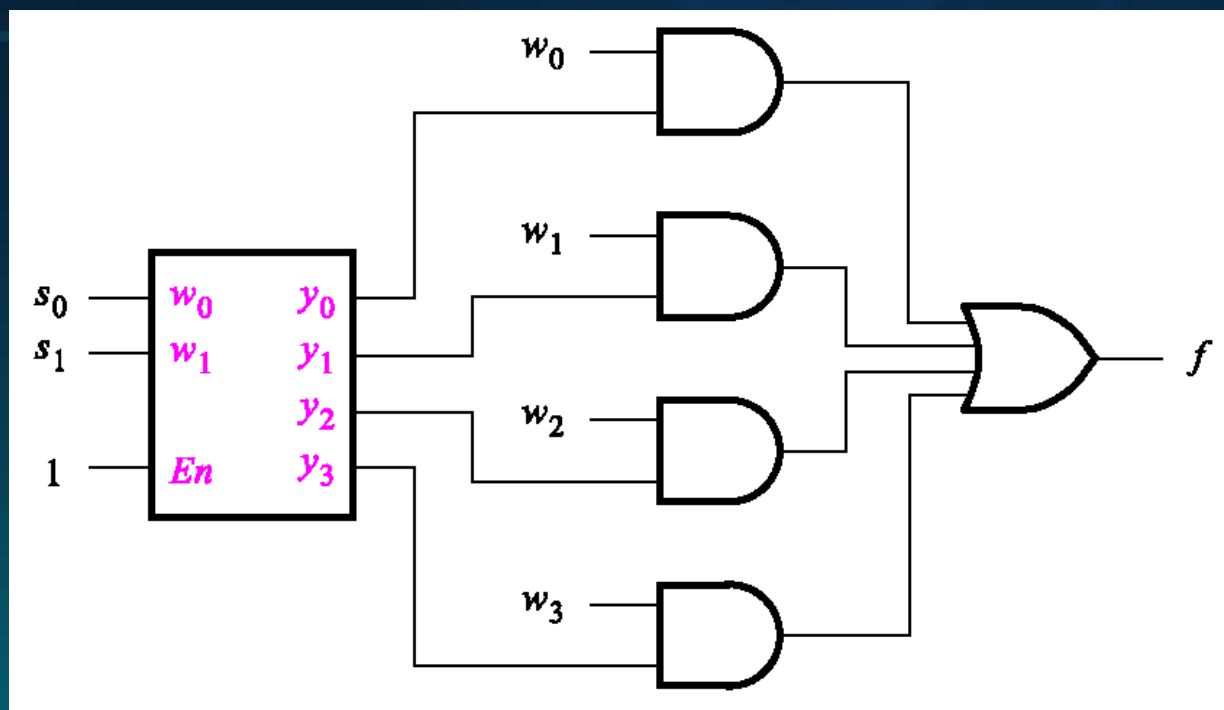


圖3.11 以解碼器建構四對一多工器。

範例四

解碼器和三態緩衝器建構四對一多工器

- 我們說明過如何以兩個三態緩衝器建構二對一多工器。這觀念可以應用在任何尺寸的多工器和解碼器。範例如圖3.12所示。解碼器對每個選擇線的估值致能其中一個三態緩衝器，而此三態緩衝器以選定的資料輸入驅動輸出 f 。
- 我們已經知道實作多工器的許多方法。根據晶片上可用的資源，可以選擇 1.乘積總和形式、2.傳輸閘 3.三態緩衝器。
- 大多數使用對照表作為邏輯區塊的FPGA不包含三態緩衝器。因此必須以1.對照表 2.乘積總和形式實作多工器。

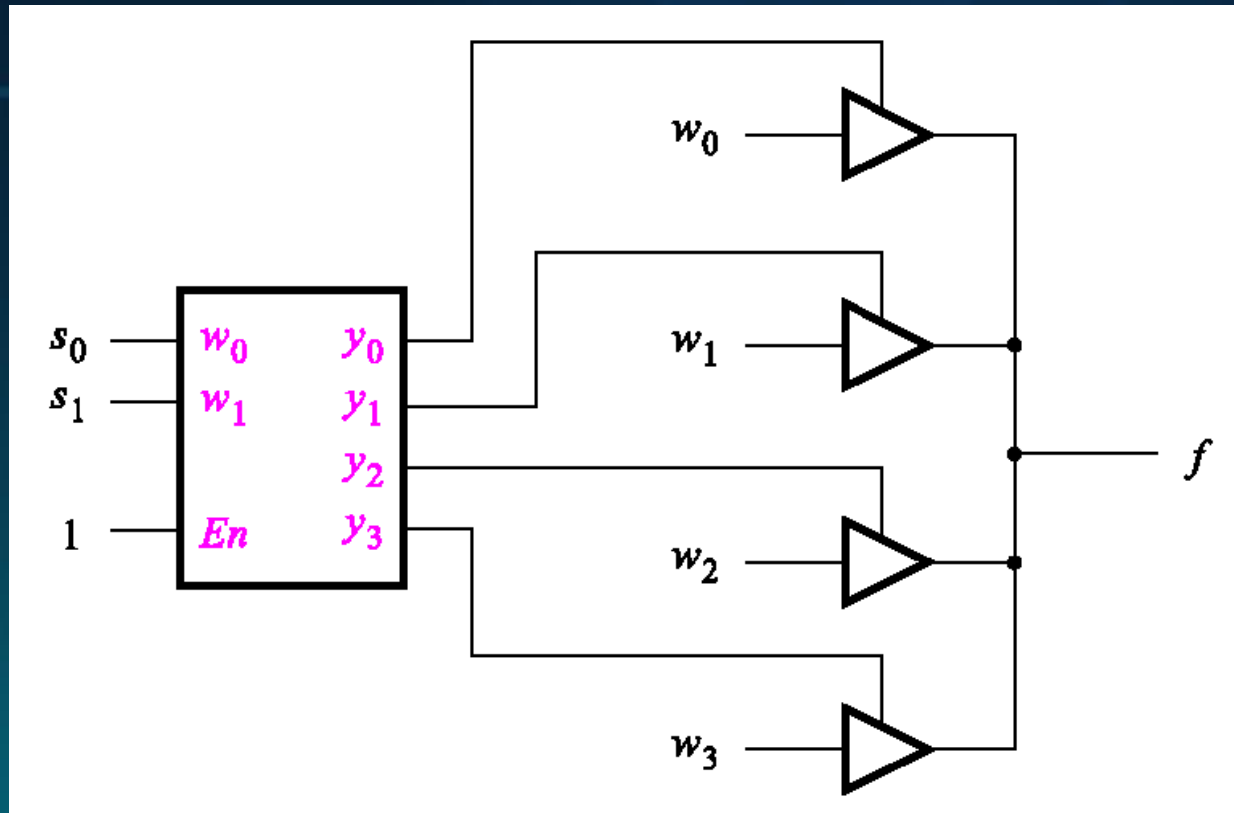


圖3.12 以解碼器和三態緩衝器建構四對一多工器。

解多工器 DEMULTIPLEXERS

- 多工器電路的目的是在選擇輸入的控制下，將 n 個資料輸入多工(multiplex)到單一資料輸出。執行相反函數的電路稱為解多工器(demultiplexer)，也就是將單一資料輸入放到多個資料輸出。
- 2-to-4 decoder = 1-to-4 demultiplexer
- n-to- 2^n decoder = 1-to-n demultiplexer

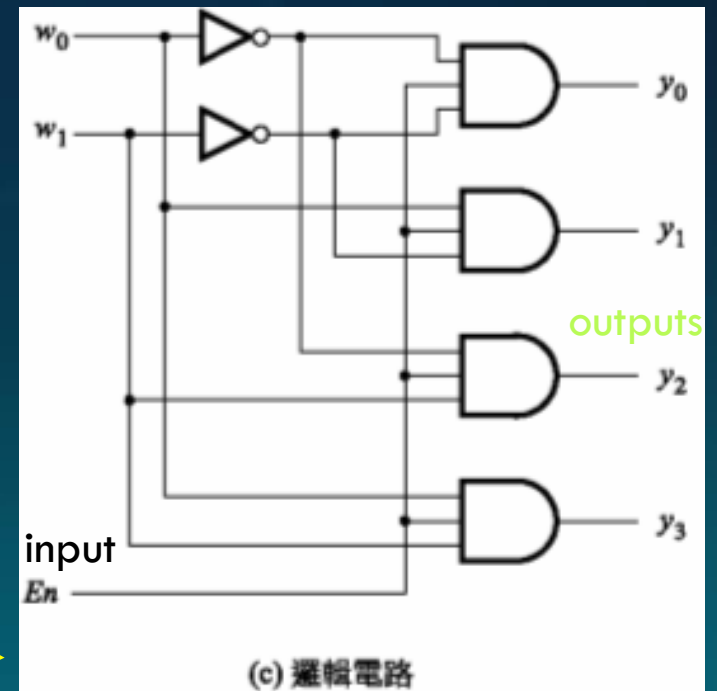


圖6.16 二對四解碼器

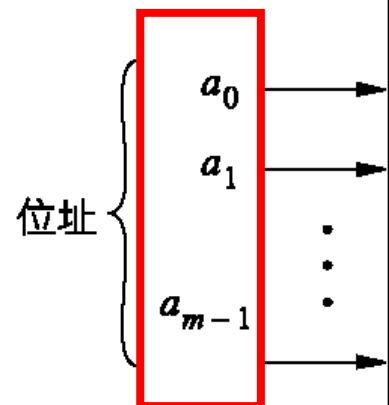
範例五 ROM 解碼器 DECODERS

- 解碼器最重要的應用之一為記憶體區塊，用來儲存資訊。這些記憶體區塊包含於數位系統，例如電腦，必須大量儲存電子化資訊。一種記憶體區塊稱為唯讀記憶體(read-only memory, ROM)。
- ROM是由一組儲存單元所組成，每個單元都永久儲存單一邏輯值，0或1。
- 圖3.13為ROM區塊的例子。儲存單元排列成 2^m 列，每列有 n 個單元。每一列儲存 n 位元的資訊。每一列在ROM的位置都由其位址 (address) 表示。

範例五 ROM 解碼器 DECODERS

- 此圖中ROM最上方的列位址為0，最下方的列位址為 2^m-1 。透過選擇線到，可以存取儲存在列裡的資訊。
- 如圖所示， m 個輸入和 2^m 個輸出的解碼器是用來產生選擇線的訊號。因為解碼器的輸入選擇了特定的位址（列），稱為位址線 (address line)。
- 儲存在列裡的資訊表現在ROM的資料輸出 d_{n-1}, \dots, d_0 ，稱為資料線 (data line)。
- 圖3.13顯示每個資料線都有對應的三態緩衝器，被ROM輸入所致能，其名稱為 $Read$ 。
- 若要從ROM存取或讀資料，必須將要求列的位址放在位址線，且將 $Read$ 設為1。

m -to- 2^m decoder
1-to- m
demultiplexer



No En input

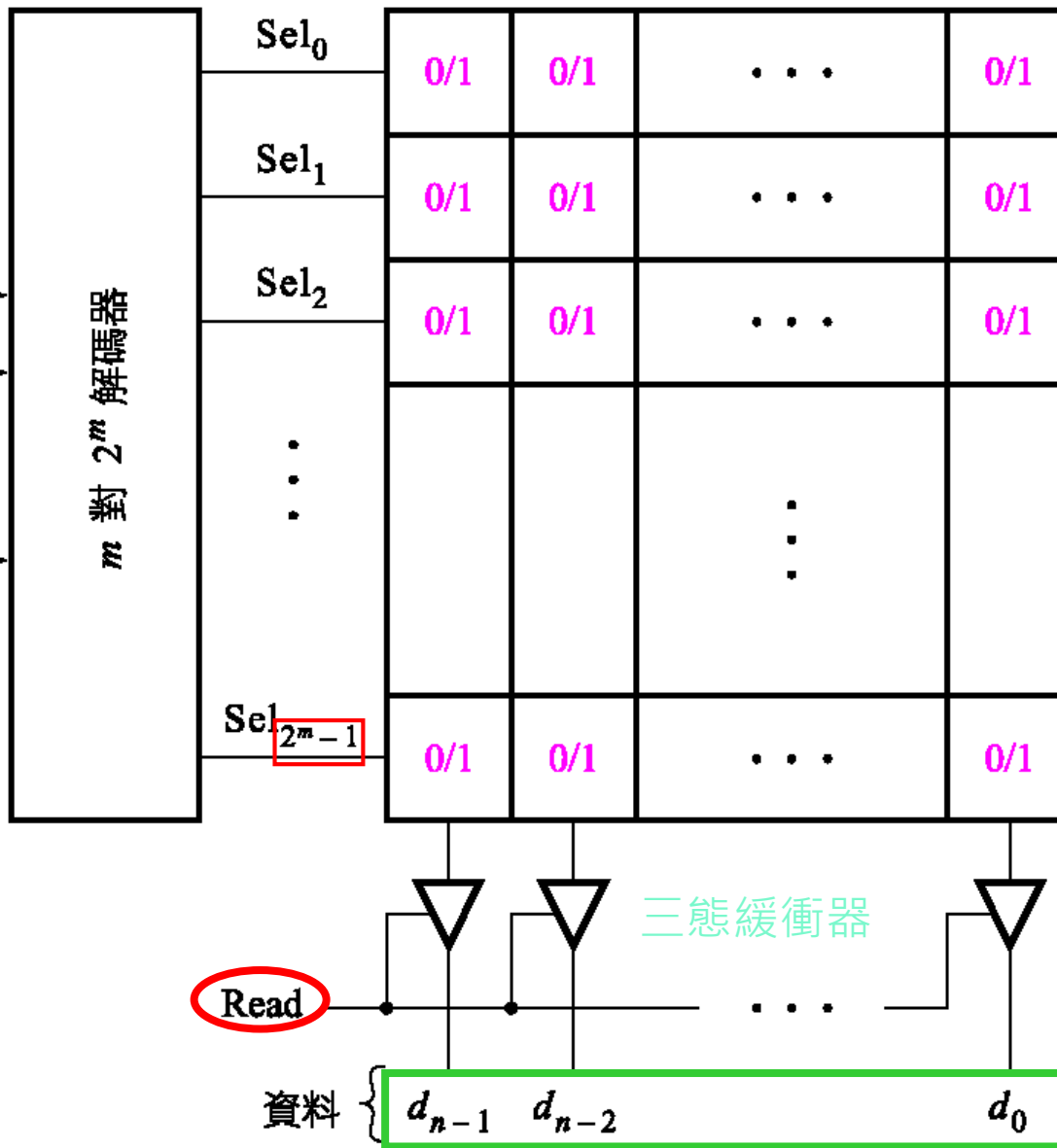


圖3.13 $2^m \times n$ 唯讀記憶體(ROM)區塊。

三、編碼器 由繁化簡

ENCODERS

- 編碼器的功能和解碼器相反，將給定資訊編碼成更精簡的形式。

二進位編碼器 由繁化簡 BINARY ENCODERS

- 二進位編碼器 (binary encoder) 將資訊從 2 個輸入編碼成 2^n 位元，如圖3.14所示。
- 只有一個輸入值為1，輸出則為該輸入的二進位數值。四對二編碼器的真值表如圖3.15a所示。
- 因此這些輸出可由圖3.15b的電路產生。我們假設輸入為單一熱門編碼。有多個輸入為1的輸入樣式沒有顯示於此真值表，它們被視為不理會條件。

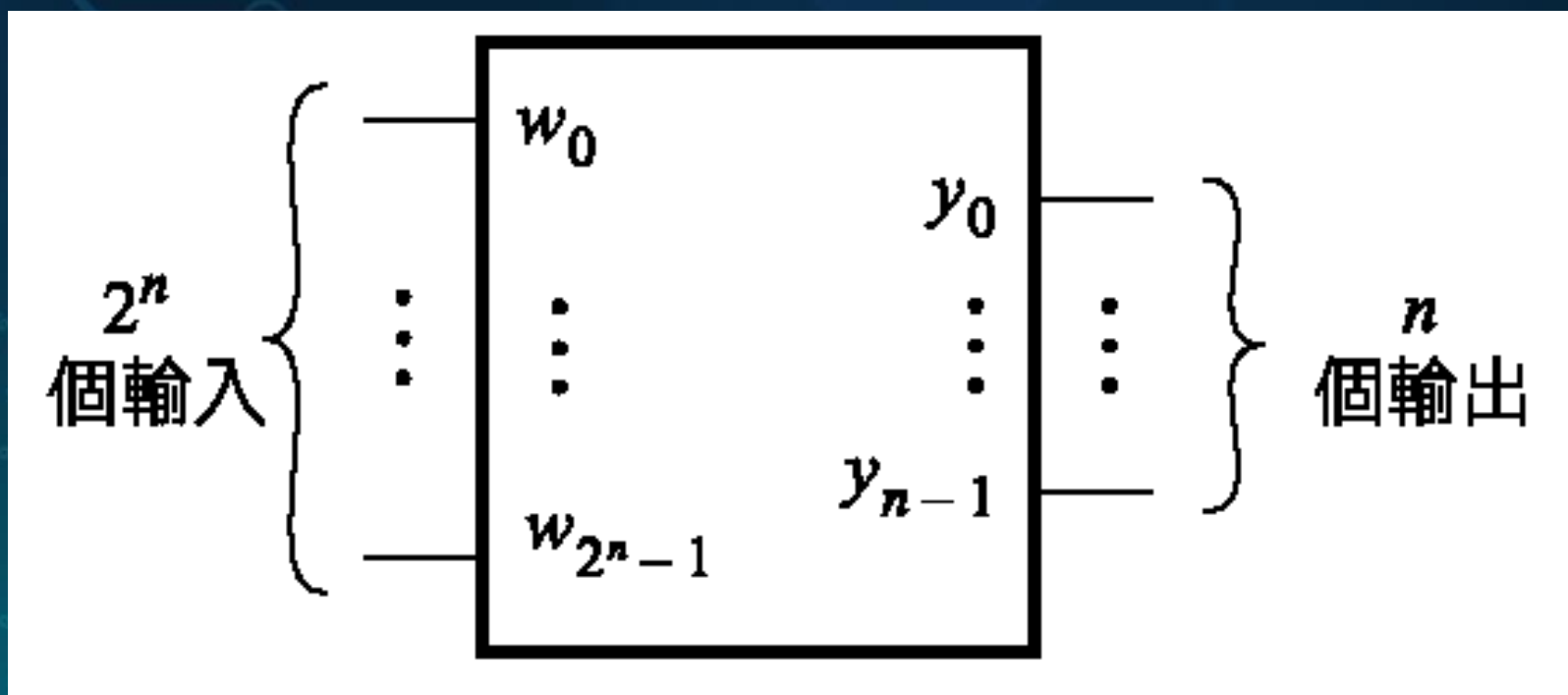
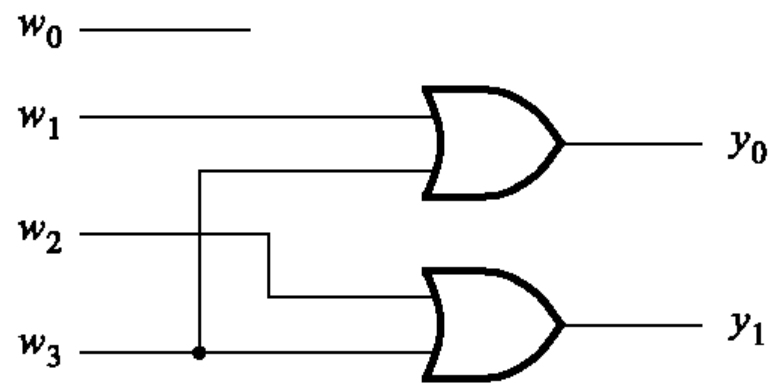


圖3.14 對二進位編碼器。

w_3	w_2	w_1	w_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

(a) 真值表



(b) 電路

圖3.15 四對二的二進位編碼器。

優先權編碼器 由繁化簡

PRIORITY ENCODERS

- 在優先權編碼器 (priority encoder) 中每個輸入都有其優先等級。編碼器輸出顯示有效的輸入為有高優先權。四對二優先權編碼器的真值表如圖3.16所示。
- 只有當相同索引的輸入 w_k 為1時，每個訊號 i_k 才等於1，代表最高優先權的輸入為1。 i_0, \dots, i_3 的邏輯表示式為

最低	↓	$i_0 = \bar{w}_3 \bar{w}_2 \bar{w}_1 w_0$
優先權		$i_1 = \bar{w}_3 \bar{w}_2 w_1$
		$i_2 = \bar{w}_3 w_2$
最高		$i_3 = w_3$

優先權編碼器 由繁化簡

PRIORITY ENCODERS

- 使用中間訊號，優先權編碼器電路的其他部分就和圖 3.15 的二進位編碼器架構一樣，即為

$$y_0 = i_1 + i_3$$

$$y_1 = i_2 + i_3$$

- 輸出 z 為

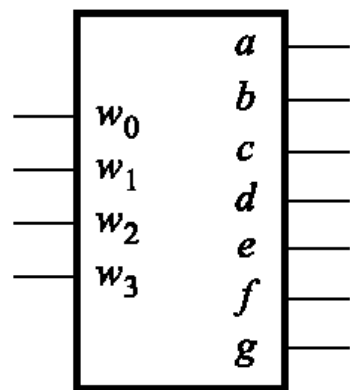
$$z = i_0 + i_1 + i_2 + i_3$$

w_3	w_2	w_1	w_0	y_1	y_0	z
0	0	0	0	d	d	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

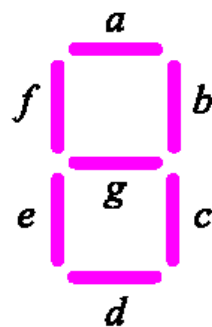
圖3.16 4對2優先權編碼器的真值表

四、編碼轉換器（類似解碼器）由簡化繁 CODE CONVERTERS

- 編碼器和解碼器電路的目的，是將一種輸入編碼方式轉換成不同的輸出編碼方式。
- 如圖3.17a所示，此電路將BCD數字轉換成七個訊號，用來驅動顯示器的區段。每一段都是小型的發光二級體 (LED)，以電子訊號驅動時會發光。圖中的區段標示成 a 到 g 。BCD對七段解碼器的真值表如圖3.17c所示。
- 雖然通常稱此電路為解碼器，但是稱為編碼轉換器比較適合。
- 解碼器這個詞通常更適合於產生單一熱門編碼輸出的電路。



(a) 編碼轉換器



(b) 七段顯示器

w_3	w_2	w_1	w_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

(c) 真值表

圖3.17 BCD對七段顯示器編碼轉換器。

五、算術比較電路

ARITHMETIC COMPARISON CIRCUITS

- 一種有用的算術電路可以比較兩個二進位數值的相對大小。這種電路稱為**比較器 (comparator)**。
- 比較器產生三個輸出，稱為AeqB、AgtB和AltB。若A和B相等，則AeqB輸出為1。若A大於B，AgtB輸出為1。若A小於B，則AltB輸出為1。
- 實做四位元比較器的邏輯電路如圖3.18所示。這方法可以用來設計任何n值的比較器。

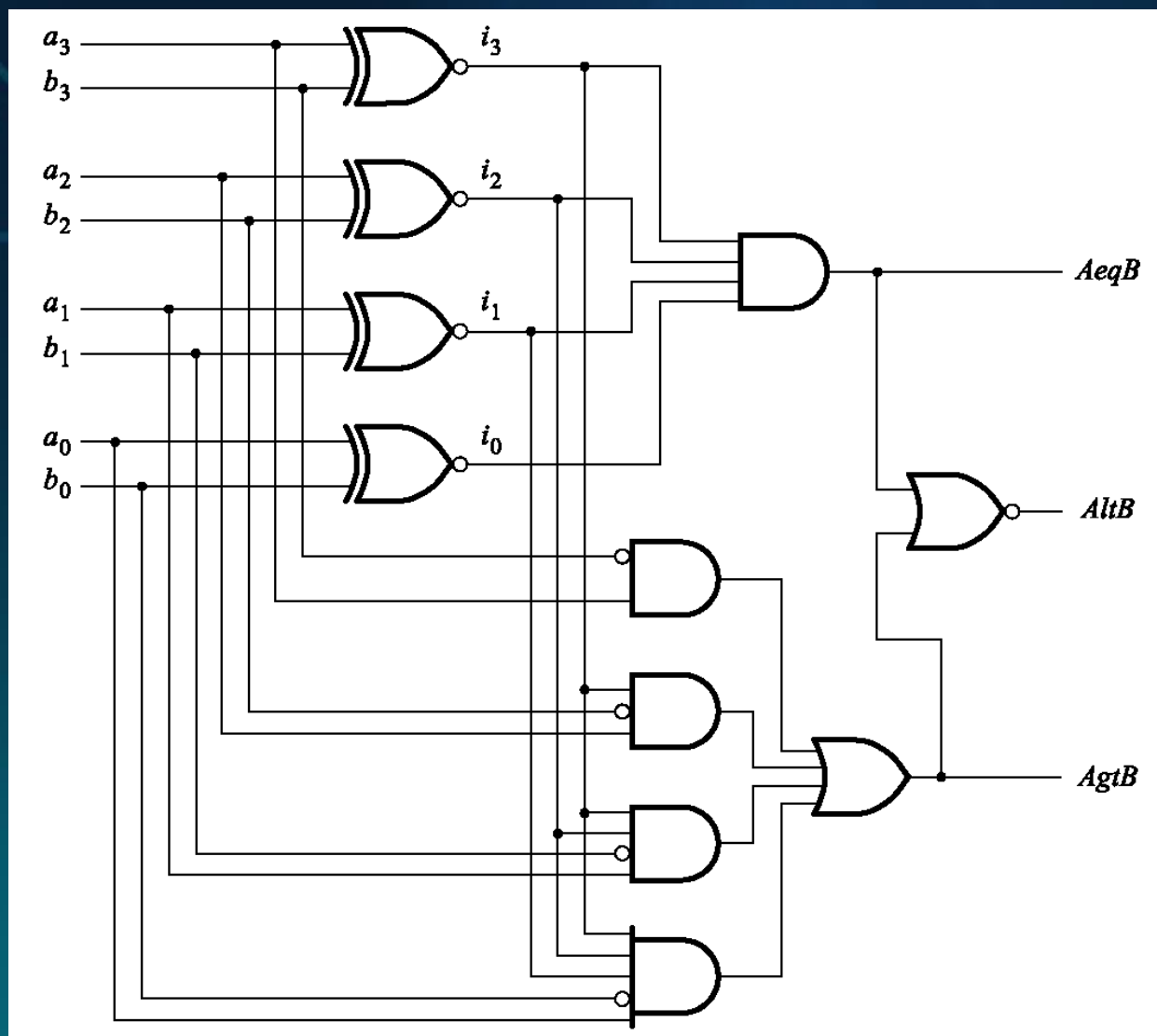


圖3.18 四位元比較器電路。

六、組合電路的VHDL程式碼

VHDL FOR COMBINATIONAL CIRCUITS

- (一) 指定敘述
- (二) 選定訊號指定
- (三) 條件訊號指定
- (四) GENERATE敘述
- (五) 同作與循序指定敘述
- (六) 程序敘述 PROC
- (七) Case敘述

(一) 指定敘述

- VHDL提供一些敘述類型，可用於指定邏輯值給訊號。
- 本章將介紹其他類型的指定敘述：
 1. 選定訊號指定 Selected Signal Assignment 、
 2. 條件訊號指定 Conditional Signal Assignment 、
 3. generate敘述 Generate Assignment 、
 4. if-then-else敘述 if-then-else Assignment 、和
 5. case敘述 Case Assignment 。

(二) 選定訊號指定 WITH s SELECT SELECTED SIGNAL ASSIGNMENT

- 選定訊號指定允許訊號被指定一些值的其中一個，根據選擇準則。圖3.19顯示如何用它來描述二對一多工器。
- 選定訊號指定以關鍵字**WITH**開始，指定使用s作為選擇準則。

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT ( w0, w1, s    : IN    STD_LOGIC ;
          f          : OUT  STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
    WITH s SELECT
        f <= w0 WHEN '0',
          w1 WHEN OTHERS ;
END Behavior ;
```

圖3.19 二對一多工器的VHDL程式碼。

範例六 四對一多工器

- 圖3.21為四對一多工器，以實體*mux4to1*描述。兩個選擇輸入即圖3.20的 s_1 和 s_0 ，是以二位元STD_LOGIC_VECTOR訊號 s 表示。
- 選定訊號指定根據 s 的值，設定 f 為輸入值 w_0, \dots, w_3 的其中一個。編譯此程式碼會產生圖3.20c的電路。
- 在圖3.21的結尾，實體 *mux4to1* 被定義為包裝 *mux4to1_package* 的一個元件。元件宣告允許將實體作為其他VHDL程式碼的子電路。

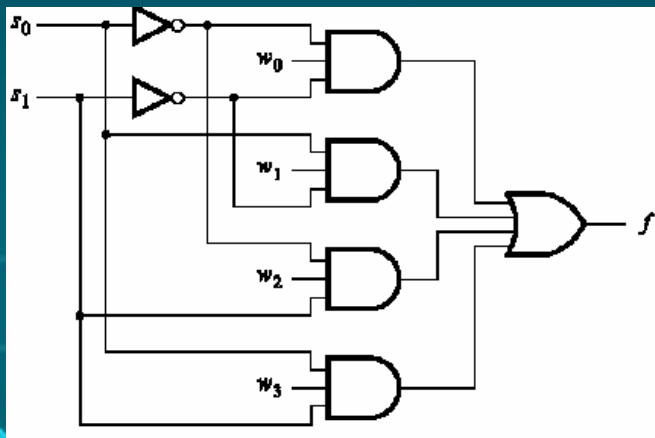


圖3.20 四對一多工器。

```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux4to1 IS
    PORT (    w0, w1, w2, w3      : IN      STD_LOGIC ;
             s                    : IN      STD_LOGIC_VECTOR(1 DOWNTO 0) ;
             f                    : OUT     STD_LOGIC ) ;
END mux4to1 ;

ARCHITECTURE Behavior OF mux4to1 IS
BEGIN
    WITH s SELECT
        f <= w0 WHEN "00",
            w1 WHEN "01",
            w2 WHEN "10",
            w3 WHEN OTHERS ;
END Behavior ;

```

圖3.21 四對一多工器的VHDL程式碼。

```
LIBRARY ieee ;  
USE ieee.std_logic_1164.all ;  
PACKAGE mux4to1_package IS  
    COMPONENT mux4to1  
        PORT ( w0, w1, w2, w3  : IN      STD_LOGIC ;  
              s                : IN STD_LOGIC_VECTOR(1 DOWNTO 0) ;  
              f                : OUT    STD_LOGIC ) ;  
    END COMPONENT ;  
END mux4to1_package ;
```

圖3.21 四對一多工器的VHDL程式碼。

範例七

MUX4TO1_PACKAGE -> 十六對一多工器

- 圖3.22顯示如何以五個四對一多工器建構十六對一多工器。圖3.23為此電路的VHDL程式碼，用了*mux4to1*元件。
- 模組*mux16to1*的資料輸入為16位元向量*W*，選擇輸入為四位元向量*S*。在VHDL程式碼，圖3.22左邊四個多工器的輸出名稱是有必要的。
- 第11行定義四位元訊號*M*即作此用途，而第13到16行實體化這四個多工器。例如，第13行對應到圖3.22左上方的多工器。前四個埠，對應到圖3.21的*w*₀, ..., *w*₃，是由訊號*W*(0), ..., *W*(3)所驅動。
- 用語法*s* (1 DOWNT0 0) 將訊號*S*(1) 和*S*(0)連接到*mux4to1*模組的二位元*S*埠。*M* (0)訊號連接到多工器的輸出埠。
- 第17行實體化圖3.22右邊的多工器。訊號*m*(0), ..., *m*(3)連接到資料輸入，而*s*(3 DOWNT0 2)所指定的位元*s*(3)和*s*(2)則連到選擇輸入。輸出埠產生*mux16to1*輸出*f*。編譯此程式碼產生下列多工器函數

$$f = \overline{s}_3 \overline{s}_2 \overline{s}_1 \overline{s}_0 w_0 + \overline{s}_3 \overline{s}_2 \overline{s}_1 s_0 w_1 + \overline{s}_3 \overline{s}_2 s_1 \overline{s}_0 w_2 + \cdots + s_3 s_2 s_1 \overline{s}_0 w_{14} + s_3 s_2 s_1 s_0 w_{15}$$

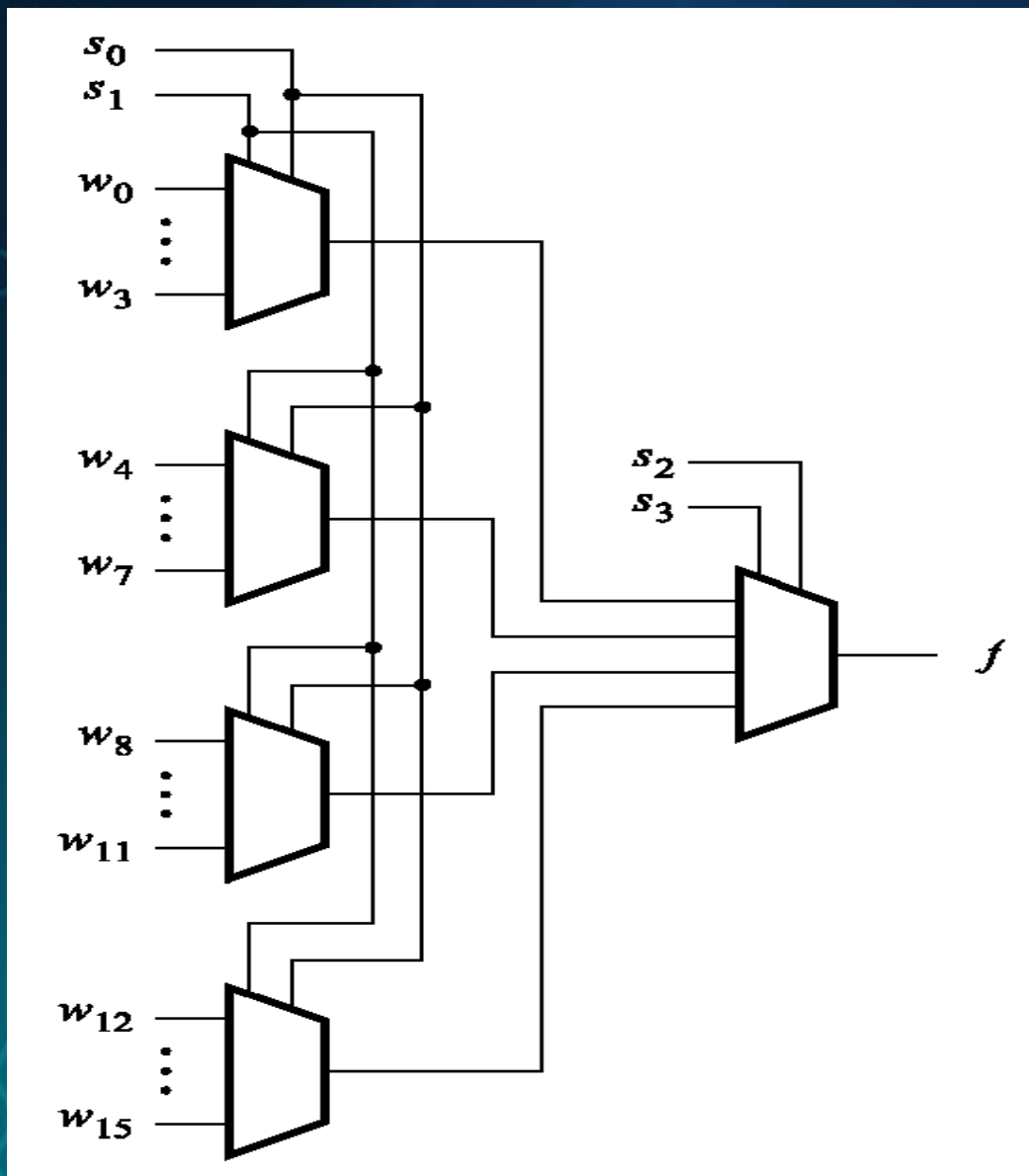


圖3.22 十六對一多工器。

```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
LIBRARY work ;
USE work.mux4to1_package.all ;

ENTITY mux16to1 IS
    PORT ( w    : IN STD_LOGIC_VECTOR(0 TO 15) ;
          s    : IN STD_LOGIC_VECTOR(3 DOWNT0 0) ;
          f    : OUT  STD_LOGIC ) ;
END mux16to1 ;

ARCHITECTURE Structure OF mux16to1 IS
    SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;
BEGIN
    Mux1: mux4to1 PORT MAP ( w(0), w(1), w(2), w(3), s(1 DOWNT0 0), m(0) ) ;
    Mux2: mux4to1 PORT MAP ( w(4), w(5), w(6), w(7), s(1 DOWNT0 0), m(1) ) ;
    Mux3: mux4to1 PORT MAP ( w(8), w(9), w(10), w(11), s(1 DOWNT0 0), m(2) ) ;
    Mux4: mux4to1 PORT MAP ( w(12), w(13), w(14), w(15), s(1 DOWNT0 0), m(3) ) ;
    Mux5: mux4to1 PORT MAP ( m(0), m(1), m(2), m(3), s(3 DOWNT0 2), f ) ;
END Structure ;

```

圖3.23 十六對一多工器的VHDL程式碼。

範例八 二對四的二進位解碼器

- 選定訊號指定也可用來描述其他類型的電路。
- 圖3.24顯示如何用選定訊號指定來描述二對四的二進位編碼器的真值表。實體名稱為`dec2to4`。
- 資料輸入為二位元訊號 w ，致能輸入 En 。四個輸出是以四位元訊號 y 表示。

範例八 二對四的二進位解碼器

- 圖3.8a的解碼器真值表所列的輸入依序為 $En\ w_1\ w_0$ 。為了表示這三個訊號，VHDL程式碼定義三位元訊號 Enw 。
- 敘述 $Enw \leq En \ \& \ W$ 使用VHDL串接運算子，將 En 和 w 組合成 Enw 訊號。因此 $Enw(2)=En$ 、 $Enw(1)=w_1$ 且 $Enw(0)=w_0$ 。
- Enw 訊號是作為選定訊號指定敘述中的選定訊號，描述圖3.8a的真值表。
- 在前四個WHEN敘述中 $En=1$ ，且解碼器輸出的樣式和真值表前四列相同。
- 最後一個WHEN敘述使用OTHERS關鍵字，並且設定解碼器輸出為0000，因為這代表 $En=0$ 的情況。


```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY dec2to4 IS
    PORT ( w : IN STD_LOGIC_VECTOR(1 DOWNTO 0) ;
          En : IN STD_LOGIC ;
          y : OUT STD_LOGIC_VECTOR(0 TO 3) ) ;
END dec2to4 ;

ARCHITECTURE Behavior OF dec2to4 IS
    SIGNAL Enw : STD_LOGIC_VECTOR(2 DOWNTO 0) ;
BEGIN
    Enw <= En & w ;
    WITH Enw SELECT
        y <= "1000" WHEN "100",
        "0100" WHEN "101",
        "0010" WHEN "110",
        "0001" WHEN "111",
        "0000" WHEN OTHERS ;
END Behavior ;

```

圖 3.24 二對四的二進位解碼器的VHDL程式碼。

(三) 條件訊號指定 WHEN ELSE CONDITIONAL SIGNAL ASSIGNMENT

- 類似於選定訊號指定，條件訊號指定允許訊號被設定成某些值的其中一個。
- 圖3.25為圖3.19二對一多工器實體的修改版本使用條件訊號指定。在這個小例子裡，條件訊號指定只有WHEN敘述。

```
LIBRARY ieee ;  
USE ieee.std_logic_1164.all ;  
  
ENTITY mux2to1 IS  
    PORT ( w0, w1, s      : IN STD_LOGIC ;  
          f              : OUT  STD_LOGIC ) ;  
END mux2to1 ;  
  
ARCHITECTURE Behavior OF mux2to1 IS  
BEGIN  
    f <= w0 WHEN s = '0' ELSE w1 ;  
END Behavior ;
```

圖 3.25 使用條件訊號指定的二對一多工器規格。

範例九 四對二優先權編碼器

- 圖3.16為四對二優先權編碼器。描述此真值表的VHDL程式碼如圖3.26所示。編碼器的輸入是以四位元訊號 w 表示。編碼器的輸出為 y 和 z ，其中 y 是二位元訊號。
- 條件訊號指定當 $w(3) = 1$ 時，將值11指定給 y 。若條件為真，則ELSE關鍵字之後的另一個WHEN敘述不會影響 f 的值。因此 $w(2)$ 、 $w(1)$ 和 $w(0)$ 的值不重要，實現了想要的優先權。
- 第二個WHEN敘述表示當 $w(2) = 1$ 時，將值10指定給 y 。這只有當 $w(3) = 0$ 時才會發生。只有當先前的WHEN敘述條件都不為真時，接下來的WHEN才會影響 y 。圖3.26包含輸出 z 的第二個條件訊號指定，當所有四個輸入都是0， z 被指定值為0；否則 z 被指定值為1。

範例九 (續) 四對二優先權編碼器

- 條件訊號指定中每個WHEN敘述對應的優先權，是它和選定訊號指定最關鍵的不同，因為選定訊號指定沒有優先權觀念。我們也可以用選定訊號指定來描述優先權編碼器，但是程式碼會相當繁複。
- 一個可能的寫法如圖3.27所示。第一個WHEN敘述將 y 設為00，當 w_0 是唯一是1的輸入。接下來兩個敘述說明當 $w_3 = w_2 = 0$ 且 $w_1 = 1$ 時， y 應該為01。接下來四個敘述說明當 $w_3 = 0$ 且 $w_2 = 1$ 時， y 應該為10。
- 最後一個WHEN敘述說明對所有輸入估值 y 都應該為11，包含 w_3 為1的所有估值。注意，OTHERS敘述包含輸入估值0000。此樣式產生 $z = 0$ ，在此情況下 y 值不重要。

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY priority IS
    PORT ( w    : IN    STD_LOGIC_VECTOR(3 DOWNT0 0) ;
          y    : OUT   STD_LOGIC_VECTOR(1 DOWNT0 0) ;
          z    : OUT   STD_LOGIC ) ;
END priority ;

ARCHITECTURE Behavior OF priority IS
BEGIN
    y <=  "11" WHEN w(3) = '1' ELSE
          "10" WHEN w(2) = '1' ELSE
          "01" WHEN w(1) = '1' ELSE
          "00" ;
    z <=  '0' WHEN w = "0000" ELSE '1' ;
END Behavior ;
```

圖 3.26 優先權編碼器的VHDL程式碼

```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY priority IS
    PORT ( w : IN      STD_LOGIC_VECTOR(3 DOWNT0 0) ;
          y : OUT      STD_LOGIC_VECTOR(1 DOWNT0 0) ;
          z : OUT      STD_LOGIC ) ;
END priority ;

ARCHITECTURE Behavior OF priority IS
BEGIN
    WITH w SELECT
        y <=  "00" WHEN "0001",
              "01" WHEN "0010",
              "01" WHEN "0011",
              "10" WHEN "0100",
              "10" WHEN "0101",
              "10" WHEN "0110",
              "10" WHEN "0111",
              "11" WHEN OTHERS ;
    WITH w SELECT
        z <=  '0' WHEN "0000",
              '1' WHEN OTHERS ;
END Behavior ;

```

圖 3.27 優先權編碼器的較沒有效率程式碼

範例十 四位元比較器電路

- 我們得到圖3.18的比較器電路，圖3.28顯示如何以VHDL程式碼描述此電路。三個條件訊號指定的每一個都決定其中一個比較器輸出。
- 包裝std_logic_unsigned包含在程式碼中，因為它指定STD_LOGIC_VECTOR訊號，關係運算子提供了方便的方法來指定想要的功能性。
- 圖3.28程式碼產生的電路類似與圖3.18的電路，但不一樣。VHDL編譯器實體化一個預先定義好的模組以實作每個比較運算。
- 不使用std_logic_unsigned資料庫，另一個指定產生的電路必須使用無符號數的方法是包含std_logic_arith資料庫。
- 此例中訊號A和B應該定義為UNSIGNED類型，若我們想要電路以帶正負號的數值工作，訊號A和B應該定義為SIGNED類型。此程式碼如圖3.29所示。


```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;

ENTITY compare IS
    PORT ( A, B                : IN    STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          AeqB, AgtB, AltB : OUT    STD_LOGIC ) ;
END compare ;

ARCHITECTURE Behavior OF compare IS
BEGIN
    AeqB <= '1' WHEN A = B ELSE '0' ;
    AgtB <= '1' WHEN A > B ELSE '0' ;
    AltB <= '1' WHEN A < B ELSE '0' ;
END Behavior ;
```

圖 3.28 四位元比較器的 VHDL 程式碼

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_arith.all ;

ENTITY compare IS
    PORT ( A, B          : IN    SIGNED(3 DOWNTO 0) ;
          AeqB, AgtB, AltB: OUT   STD_LOGIC ) ;
END compare ;

ARCHITECTURE Behavior OF compare IS
BEGIN
    AeqB <= '1' WHEN A = B ELSE '0' ;
    AgtB <= '1' WHEN A > B ELSE '0' ;
    AltB <= '1' WHEN A < B ELSE '0' ;
END Behavior ;
```


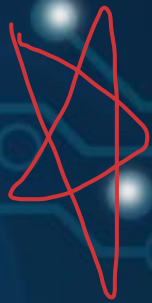


圖 3.29 帶正負號數值的四位元比較器 (圖 6.34) 程式碼



(四) GENERATE敘述 FOR IN .. TO .. GENERATE GENERATE ASSIGNMENT

- VHDL提供 **FOR GENERATE**敘述來描述規則的結構階層性程式碼。
- 圖3.30為圖3.23程式碼重新用 FOR GENERATE敘述撰寫而成。**GENERATE**敘述必須有標籤，因此我們在程式碼中用標籤G1。此迴圈實體化四個mux4to1元件，使用迴圈索引i從0到3。

```
LIBRARY ieee ;  
USE ieee.std_logic_1164.all ;  
USE work.mux4to1_package.all ;
```

```
ENTITY mux16to1 IS
```

```
    PORT ( w    : IN STD_LOGIC_VECTOR(0 TO 15) ;  
          s    : IN STD_LOGIC_VECTOR(3 DOWNT0 0) ;  
          f    : OUT  STD_LOGIC ) ;
```

```
END mux16to1 ;
```

```
ARCHITECTURE Structure OF mux16to1 IS
```

```
    SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;
```

```
BEGIN
```

```
    G1: FOR i IN 0 TO 3 GENERATE
```

```
        Muxes: mux4to1 PORT MAP (
```

```
            w(4*i), w(4*i+1), w(4*i+2), w(4*i+3), s(1 DOWNT0 0), m(i) ) ;
```

```
    END GENERATE ;
```

```
    Mux5: mux4to1 PORT MAP ( m(0), m(1), m(2), m(3), s(3 DOWNT0 2), f ) ;
```

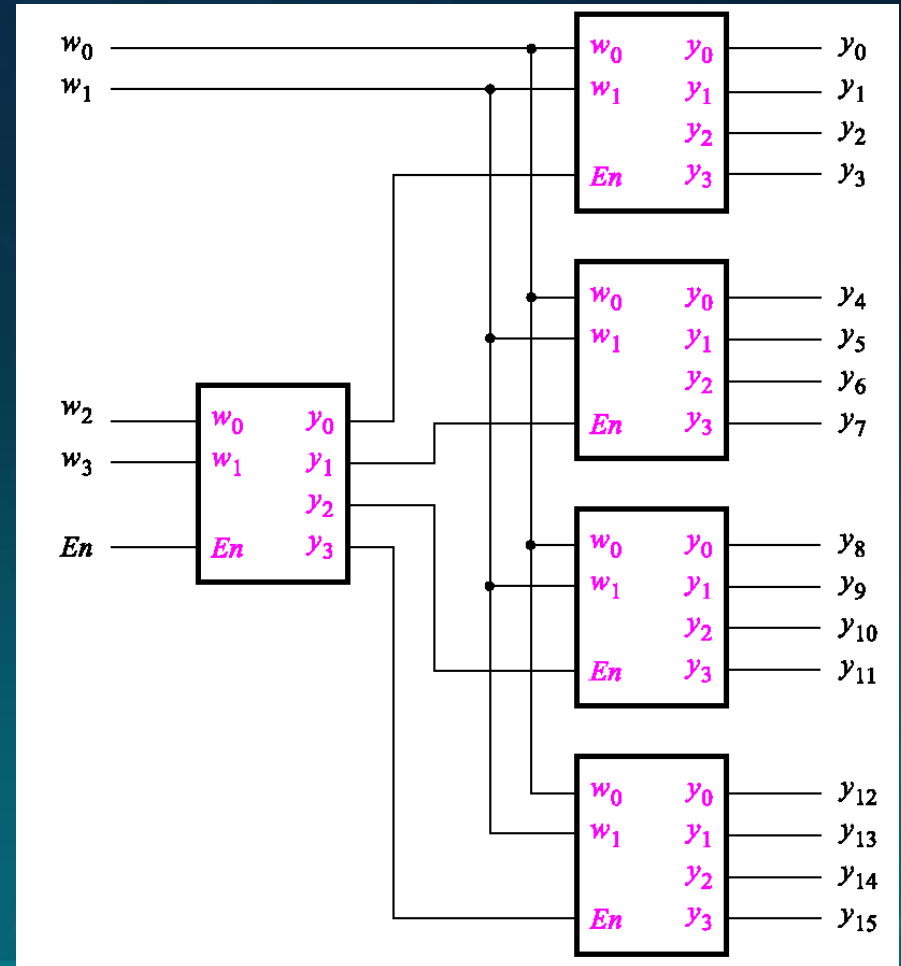
```
END Structure ;
```

圖 3.30 使用 generate 敘述的十六對一多工器的程式碼

範例十一 *DEC2TO4*四對十六解碼器

- 除了FOR GENERATE敘述之外，VHDL提供另一種類型的generate敘述，稱為**IF GENERATE**。圖3.31說明這兩種generate敘述的使用。
- 此程式碼為圖3.10四對十六解碼器的**階層性描述**，使用五個定義於圖3.24的*dec2to4*元件實體。解碼器輸入為四位元訊號*w*、致能輸入*En*，且輸出為16位元訊號*y*。
- 在*dec2to4*次電路物件宣告之後，architecture定義訊號*m*，代表圖3.10左邊的二對四解碼器輸出。
- **FOR GENERATE**敘述實體化五個*dec2to4*元件。在迴圈的每一次遞迴，標示*Dec_ri*的敘述實體化一個*dec2to4*元件，對應到圖3.10右邊的一個二對四解碼器。
- 第一次的迴圈遞迴產生*dec2to4*元件，其資料輸入為*w1*和*w0*，致能輸入為*m0*，輸出為*y0*、*y1*、*y2*和*y3*。其他的迴圈遞迴也可以用資料輸入*w1w0*，但使用不同的*m*和*y*位元。

- 標示為**G2**的IF GENERATE敘述
 在最後一個迴圈遞迴即條件 $i = 3$
 為真時，實體化一個 $dec2to4$ 元件。
 此元件代表圖3.10左邊的二對四
 解碼器，具有二位元資料輸入 $w3$
 和 $w2$ ，致能輸入為 En ，輸出為
 $m0$ 、 $m1$ 、 $m2$ 和 $m3$ 。注意，我們
 不使用IF GENERATE敘述，而
 是在FOR GENERATE敘述之外
 實體化此元件。
- 圖3.30和3.31的generate敘述是用
 來實體化元件。另一個generate
 敘述的用法是產生一組邏輯方程
 式。



```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY dec4to16 IS
    PORT (
        w      : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
        En     : IN      STD_LOGIC ;
        y      : OUT     STD_LOGIC_VECTOR(0 TO 15) ) ;
END dec4to16 ;

ARCHITECTURE Structure OF dec4to16 IS
    COMPONENT dec2to4
        PORT (
            w      : IN      STD_LOGIC_VECTOR(1 DOWNTO 0) ;
            En     : IN      STD_LOGIC ;
            y      : OUT     STD_LOGIC_VECTOR(0 TO 3) ) ;
    END COMPONENT ;
    SIGNAL m : STD_LOGIC_VECTOR(0 TO 3) ;
BEGIN
    G1: FOR i IN 0 TO 3 GENERATE
        Dec_ri: dec2to4 PORT MAP ( w(1 DOWNTO 0), m(i), y(4*i TO 4*i+3) );
        G2: IF i=3 GENERATE
            Dec_left: dec2to4 PORT MAP ( w(i DOWNTO i-1), En, m ) ;
        END GENERATE ;
    END GENERATE ;
END Structure ;

```

圖 3.31 四對十六 二進位解碼器的階層性程式碼

(五) 同作與循序指定敘述

CONCURRENT AND SEQUENTIAL ASSIGNMENT STATEMENT

VHDL：兩大類 assignment statement

1. 同作(同時)指定敘述 (concurrent assignment statement): 簡單指定敘述，選定指定敘述，和條件指定敘述。這些敘述有個共通特性，它們在VHDL程式碼出現的順序不影響程式碼的意義。
2. 循序指定敘述(Sequential Assignment Statement): PROCESS Statement (if-then-else Assignment Statement, Case Assignment Statement)

(六) 程序敘述 PROCESS STATEMENT

目前不能用

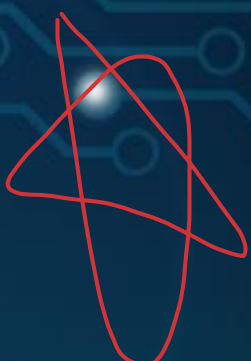
- 程序敘述以關鍵字PROCESS開始，接著是以括號包圍的訊號列表，稱為敏感列表(sensitivity list)。對組合電路例如多工器而言，敏感列表包含程序內使用的所有輸入訊號。VHDL編譯器會將程序敘述轉譯成邏輯方程式。
- 一般來說，程序內有許多敘述，以VHDL專用術語，當程序的敏感列表中任何訊號值改變時，則程序變成有效(active)。一旦有效，則程序內的敘述被依序評估計算。

if-then-else Assignment Statement

範例十二

IF-THEN-ELSE指定敘述的二對一多工器

- 圖3.33的程式碼等效於圖3.32的程式碼。程序的第一個敘述指定 $w0$ 值給 f 。這提供一個預設值給 f ，但這個指定直到程序結束才真的發生。在VHDL專業術語我們稱此指定被排程(scheduled)在程序中所有敘述都評估計算之後才發生。
- 若程序為有效的時候發生另一個指定給 f ，則預設的指定會被覆寫。程序的第二個敘述指定 $w1$ 值給 f ，若 s 值等於1。若此條件為真，則預設指定被覆寫。因此若 $s = 0$ ，則 $f = w0$ ，且若 $s = 1$ ，則 $f = w1$ ，定義了二對一多工器。編譯此程式碼會產生和圖3.19、3.25以及3.32一樣的電路，即
$$f = sw0 + sw1。$$



```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT ( w0, w1, s    : IN    STD_LOGIC ;
          f            : OUT  STD_LOGIC ) ;
END mux2to1 ;

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
    PROCESS ( w0, w1, s )
    BEGIN
        IF s = '0' THEN
            f <= w0 ;
        ELSE
            f <= w1 ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

圖 3.32 用 if-then-else 敘述指定的二對一多工器

```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT ( w0, w1, s    : IN    STD_LOGIC ;
          f    : OUT  STD_LOGIC ) ;
END mux2to1 ;

```

```

ARCHITECTURE Behavior OF mux2to1 IS
BEGIN

```

```

    PROCESS ( w0, w1, s )
    BEGIN
        f <= w0;
        IF s = '1' THEN
            f <= w1;
        END IF ;
    END PROCESS ;

```

```

    PROCESS ( w0, w1, s )
    BEGIN
        IF s = '1' THEN
            f <= w1;
        END IF ;
        f <= w0;
    END PROCESS ;

```

順序顛倒

```

END Behavior ;

```

簡單電路 $f = w0$

圖 3.33 用 if-then-else 敘述指定的二對一多工器的另一種程式碼

if-then-else Assignment Statement

- 圖3.33說明程序中的敘述排序會影響程式碼的意義。考慮將這兩個敘述的順序顛倒，使得 *if-then-else* 敘述可以先被評估計算。若 $s = 1$ ， f 被指定為 $w1$ 的值。這個指定被排程，且直到程序結束才發生。
- 然而，敘述 $f \leftarrow w0$ 是最後被評估計算的。它覆寫了第一個指定，且指定 $w0$ 給 f ，不論 s 值為何。因此，若將程序內的敘述順序顛倒，則此程式碼代表一個簡單電路 $f = w0$ ，而不是多工器。

範例十三 優先權編碼器 (同作指定敘述+程序敘述)

- 圖3.34為包含一個同作指定敘述和一個程序敘述的範例，描述優先權編碼器，其等效程式碼如圖3.26所示。
- 程序敘述用if-then-else敘述描述想要的優先權。若輸入 w_3 為1，則輸出設定為 $y = 11$ 。這個指定與輸入 w_2 、 w_1 或 w_0 的值無關，因此這些值不重要。只有當 $w_3 = 0$ 時才會執行if-then-else的other敘述。
- 第一個ELSEIF敘述說明若 w_2 為1，則 $y = 10$ 。若 $w_2 = 0$ 且 $w_1 = 1$ ，則下一個ELSEIF敘述產生 $y = 01$ 。若 $w_3 = w_2 = w_1 = 0$ ，則ELSE敘述產生 $y = 00$ 。不論 w_0 是否為1都會完成這個指定；圖3.16表示 $w = 0000$ 時 y 可以被設成任何樣式，因為此時 z 會被設成0。

範例十三 優先權編碼器 (同作指定敘述+程序敘述)

- 只要至少一個資料輸入為1，優先權編碼器的輸出 z 就會被設成1。此輸出 z 以條件指定敘述定義在圖3.34的最後。VHDL語法不允許條件指定敘述WHEN-ELSE（或選定指定敘述WITH-SELECT）出現在程序內。
- 另一種指定 z 值的方法是在程序內部使用if-then-else敘述。我們在此圖中以這種方法撰寫程式碼，是為了說明同作指定敘述可以和程序敘述一起使用。
- 程序敘述(PROCESS)可以將循序敘述和同作敘述分開。注意，程序敘述和條件指定敘述的順序不重要。VHDL將程序內部的敘述視為循序敘述，而程序敘述(PROCESS)本身為同作敘述。


```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY priority IS
    PORT (   w   : IN  STD_LOGIC_VECTOR(3 DOWNT0 0) ;
            y   : OUT  STD_LOGIC_VECTOR(1 DOWNT0 0) ;
            z   : OUT  STD_LOGIC ) ;
END priority ;

ARCHITECTURE Behavior OF priority IS
BEGIN
    PROCESS ( w )                                ;程序敘述
    BEGIN
        IF w(3) = '1' THEN
            y <= "11" ;
        ELSIF w(2) = '1' THEN
            y <= "10" ;
        ELSIF w(1) = '1' THEN
            y <= "01" ;
        ELSE
            y <= "00" ;
        END IF ;
    END PROCESS ;
    z <= '0' WHEN w = "0000" ELSE '1' ;同作指定敘述(條件指定敘述)
END Behavior ;

```

另一種指定 z 值的方法

改寫以下同作指定敘述於if-then-else敘述中再置於程序敘述中:
 z <= '0' WHEN w = "0000" ELSE '1';同作指定敘述(條件指定敘述)

圖 3.34 使用 if-then-else 敘述指定的優先權編碼器

範例十四 優先權編碼器的另一種風格程式碼

- 圖3.35為優先權編碼器的另一種風格程式碼，使用if-then-else敘述。
 - 第一個敘述提供預設值00給y1y0。
 - 若w1 = 1則第二個敘述將其覆寫，並將y1y0設成01。
 - 若w2或w3為1則第三和第四個敘述將前一個覆寫，並將y1y0分別設成10和11。
- 這四個敘述等效於圖3.34描述優先權的單一if-then-else敘述。用預設指定敘述指定值，然後用if-then-else敘述覆寫預設值，如果w = 0000的話。雖然圖3.34和3.35的範例為等效，圖3.34的程式碼意義可能比較容易瞭解。

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY priority IS
    PORT ( w    : IN  STD_LOGIC_VECTOR(3 DOWNTO 0) ;
          y    : OUT  STD_LOGIC_VECTOR(1 DOWNTO 0) ;
          z    : OUT  STD_LOGIC ) ;
END priority ;

ARCHITECTURE Behavior OF priority IS
BEGIN
    PROCESS ( w )
    BEGIN
        y <= "00" ;
        IF w(1) = '1' THEN y <= "01" ; END IF ;
        IF w(2) = '1' THEN y <= "10" ; END IF ;
        IF w(3) = '1' THEN y <= "11" ; END IF ;

        z <= '1' ;
        IF w = "0000" THEN z <= '0' ; END IF ;
    END PROCESS ;
END Behavior ;
```

圖 3.35 優先權編碼器的另一種程式碼

```
LIBRARY ieee ;  
USE ieee.std_logic_1164.all ;  
  
ENTITY compare1 IS  
    PORT ( A, B      : IN      STD_LOGIC ;  
          AeqB      : OUT     STD_LOGIC ) ;  
END compare1 ;  
  
ARCHITECTURE Behavior OF compare1 IS  
BEGIN  
    PROCESS ( A, B )  
    BEGIN  
        AeqB <= '0' ;  
        IF A = B THEN  
            AeqB <= '1' ;  
        END IF ;  
    END PROCESS ;  
END Behavior ;
```

圖 3.36 一位元相等比較器的程式碼

```
LIBRARY ieee ;  
USE ieee.std_logic_1164.all ;  
  
ENTITY implied IS  
    PORT ( A, B      : IN      STD_LOGIC ;  
          AeqB      : OUT     STD_LOGIC ) ;  
END implied ;  
  
ARCHITECTURE Behavior OF implied IS  
BEGIN  
    PROCESS ( A, B )  
    BEGIN  
        IF A = B THEN  
            AeqB <= '1' ;  
        END IF ;  
    END PROCESS ;  
END Behavior ;
```

圖 3.37 產生隱含記憶體的程序碼例子

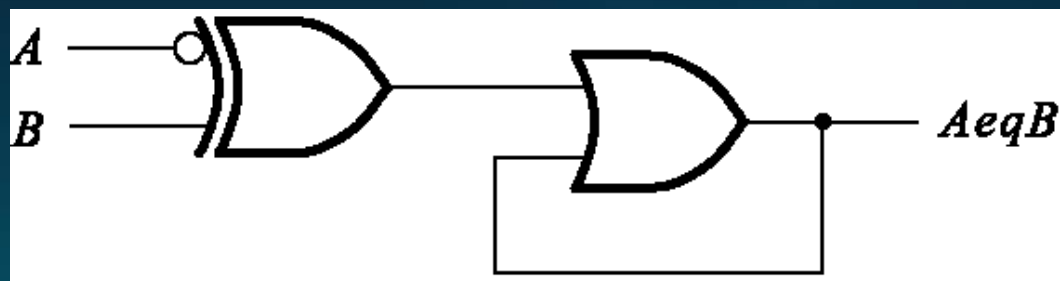


圖 3.38 圖 3.37 產生隱含記憶體程式碼所產生的電路



(七) CASE敘述 CASE ASSIGNMENT

- case敘述類似於選定訊號指定，case敘述有選擇訊號且包含對各種選擇訊號的估值的WHEN敘述。圖3.39顯示如何用case敘述做為另一種描述二對一多工器電路的方法。
- Case敘述以CASE關鍵字開始，指定s為選擇訊號。

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY mux2to1 IS
    PORT ( w0, w1, s : IN  STD_LOGIC ;
          f          : OUT  STD_LOGIC ) ;
END mux2to1 ;
```

```
ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
```

```
    PROCESS ( w0, w1, s )
    BEGIN
        CASE s IS
            WHEN '0' =>
                f <= w0 ;
            WHEN OTHERS =>
                f <= w1 ;
        END CASE ;
    END PROCESS ;
```

```
END Behavior ;
```

圖 3.39 代表二對一多工器的case敘述

範例十五 二對四的二進位解碼器

- 圖3.24為二對四解碼器的程式碼。另一種描述此電路的方法是使用循序指定敘述，如圖3.40所示。首先用一個if-then-else敘述檢查解碼器致能訊號 En 的值。
 - 若 $En = 1$ ，case敘述根據輸入 w 將輸出 y 設成適當的值。case敘述代表圖3.8a真值表的前四列。
 - 若 $En = 0$ ，ELSE 敘述將輸出 y 設成0000，即為真值表的最後一列。


```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY dec2to4 IS
    PORT (    w    : IN  STD_LOGIC_VECTOR(1 DOWNTO 0) ;
            En    : IN  STD_LOGIC ;
            y     : OUT   STD_LOGIC_VECTOR(0 TO 3) ) ;
END dec2to4 ;
```

```
ARCHITECTURE Behavior OF dec2to4 IS
BEGIN
```

```
    PROCESS ( w, En )
    BEGIN
        IF En = '1' THEN
            CASE w IS
                WHEN "00" => y <= "1000" ;
                WHEN "01" => y <= "0100" ;
                WHEN "10" => y <= "0010" ;
                WHEN OTHERS => y <= "0001" ;
            END CASE ;
        ELSE
            y <= "0000" ;
        END IF ;
    END PROCESS ;
```

```
END Behavior ;
```

圖 3.40 描述二對四的二進位解碼器的程序敘述

範例十六 BCD對七段解碼器

- case敘述的另一個例子如圖3.41所示。模組seg7為圖3.17的BCD對七段解碼器。
- **BCD輸入為四位元向量bcd**，七個輸出為七位元向量**leds**。我們列出case敘述的可能值，就如同圖3.17c的真值表。注意，在case敘述右邊有註解，**以字母a到g標示七個輸出**。
- 這些標示是告訴讀者VHDL程式碼**leds**向量的位元和圖3.17b七段的相關性。
- 最後一個case可能性將**leds**所有七個位元都設成「-」。VHDL以「-」代表不在乎條件。此可能值表示圖3.17討論的不在乎條件，也就是bcd輸入不是有效BCD數字的情況。

```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY seg7 IS
    PORT (   bcd    : IN          STD_LOGIC_VECTOR(3 DOWNT0 0) ;
            leds    : OUT         STD_LOGIC_VECTOR(1 TO 7) ) ;
END seg7 ;
ARCHITECTURE Behavior OF seg7 IS

```

```

BEGIN
    PROCESS ( bcd )
    BEGIN
        CASE bcd IS
            --          abcdefg 註解
            WHEN "0000" => leds <= "1111110" ;
            WHEN "0001" => leds <= "0110000" ;
            WHEN "0010" => leds <= "1101101" ;
            WHEN "0011" => leds <= "1111001" ;
            WHEN "0100" => leds <= "0110011" ;
            WHEN "0101" => leds <= "1011011" ;
            WHEN "0110" => leds <= "1011111" ;
            WHEN "0111" => leds <= "1110000" ;
            WHEN "1000" => leds <= "1111111" ;
            WHEN "1001" => leds <= "1110011" ;
            WHEN OTHERS => leds <= "-----" ;
        END CASE ;
    END PROCESS ;

```

```

END Behavior ;

```

圖 3.41 BCD對七段解碼器的程式碼

範例十七 74381 ALU 晶片

- 算術邏輯單元(ALU)是執行各種n位元運算元布林和算術運算的邏輯電路。一個ALU的例子為74381晶片。
- 表3.1為此晶片的函數。它有兩個四位元資料輸入A和B，一個三位元選擇輸入S，和一個四位元輸出F。
- 如表格所示，F由輸入A和B的各種算術或布林運算所定義。此表格中「+」表示算術加法，「-」表示算術減法。為避免混淆，此表格於布林運算使用XOR、OR和AND之詞。每個布林運算都是以按位元(bitwise)方式執行。例如， $F = A \text{ AND } B$ 產生四位元結果 $f_0 = a_0b_0$ 、 $f_1 = a_1b_1$ 、 $f_2 = a_2b_2$ 和 $f_3 = a_3b_3$ 。
- 以VHDL程式碼描述74381 ALU的功能如圖3.42所示。5.5.4節介紹的std_logic_unsigned包裝也包含在內，所以STD_LOGIC_VECTOR訊號A和B可用於無符號的算術運算。圖中的case敘述直接對應到表3.1。
- 為了檢查程式碼的功能，我們合成電路實作於FPGA。對每個s的估值，此電路產生正確的布林和算術運算。

表3.1 74381 ALU的功能。

運算	輸入			輸出 F
	s_2	s_1	s_0	
Clear	0	0	0	0000
$B - A$	0	0	1	$B - A$
$A - B$	0	1	0	$A - B$
ADD	0	1	1	$A + B$
XOR	1	0	0	$A \text{ XOR } B$
OR	1	0	1	$A \text{ OR } B$
AND	1	1	0	$A \text{ AND } B$
Preset	1	1	1	1 1 1 1

```

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE ieee.std_logic_unsigned.all ;
ENTITY alu IS
    PORT (
        s      : IN      STD_LOGIC_VECTOR(2 DOWNTO 0) ;
        A, B    : IN      STD_LOGIC_VECTOR(3 DOWNTO 0) ;
        F       : OUT     STD_LOGIC_VECTOR(3 DOWNTO 0) ) ;
END alu ;
ARCHITECTURE Behavior OF alu IS
BEGIN
    PROCESS ( s, A, B )
    BEGIN
        CASE s IS
            WHEN "000" =>
                F <= "0000" ;
            WHEN "001" =>
                F <= B - A ;
            WHEN "010" =>
                F <= A - B ;
            WHEN "011" =>
                F <= A + B ;
            WHEN "100" =>
                F <= A XOR B ;
            WHEN "101" =>
                F <= A OR B ;
            WHEN "110" =>
                F <= A AND B ;
            WHEN OTHERS =>
                F <= "1111" ;
        END CASE ;
    END PROCESS ;
END Behavior ;

```

圖 3.42
代表 74381 ALU
晶片功能的程式碼

七、總結評論

CONCLUDING REMARKS

- 很多時候給定電路可以用各種方式、不同架構來描述。可以用if-else敘述描述的電路，也可以用case敘述或for迴圈敘述。
- 一般來說，沒有嚴格的規則指定要用哪一種風格。使用者根據經驗會知道，哪一種敘述適合特定的設計情況。個人的偏好也會影響程式碼的撰寫。
- VHDL不是程式語言，撰寫方式也不像電腦程式。
- 設計這類電路的好方法是使用定義好的模組，就和多工器、解碼器和編碼器的方法一樣。



END