

正確學會

改訂新版 ディジタル回路と Verilog HDL

Verilog 的 16 堂課

第 10 章

進階運算電路設計

本投影片（下稱教用資源）僅授權給採用教用資源相關之旗標書籍為教科書之授課老師（下稱老師）專用，老師為教學使用之目的，得摘錄、編輯、重製教用資源（但使用量不得超過各該教用資源內容之80%）以製作為輔助教學之教學投影片，並於授課時搭配旗標書籍公開播放，但不得為網際網路公開傳輸之遠距教學、網路教學等之使用；除此之外，老師不得再授權予任何第三人使用，並不得將依此授權所製作之教學投影片之相關著作物移作他用。

本章重點

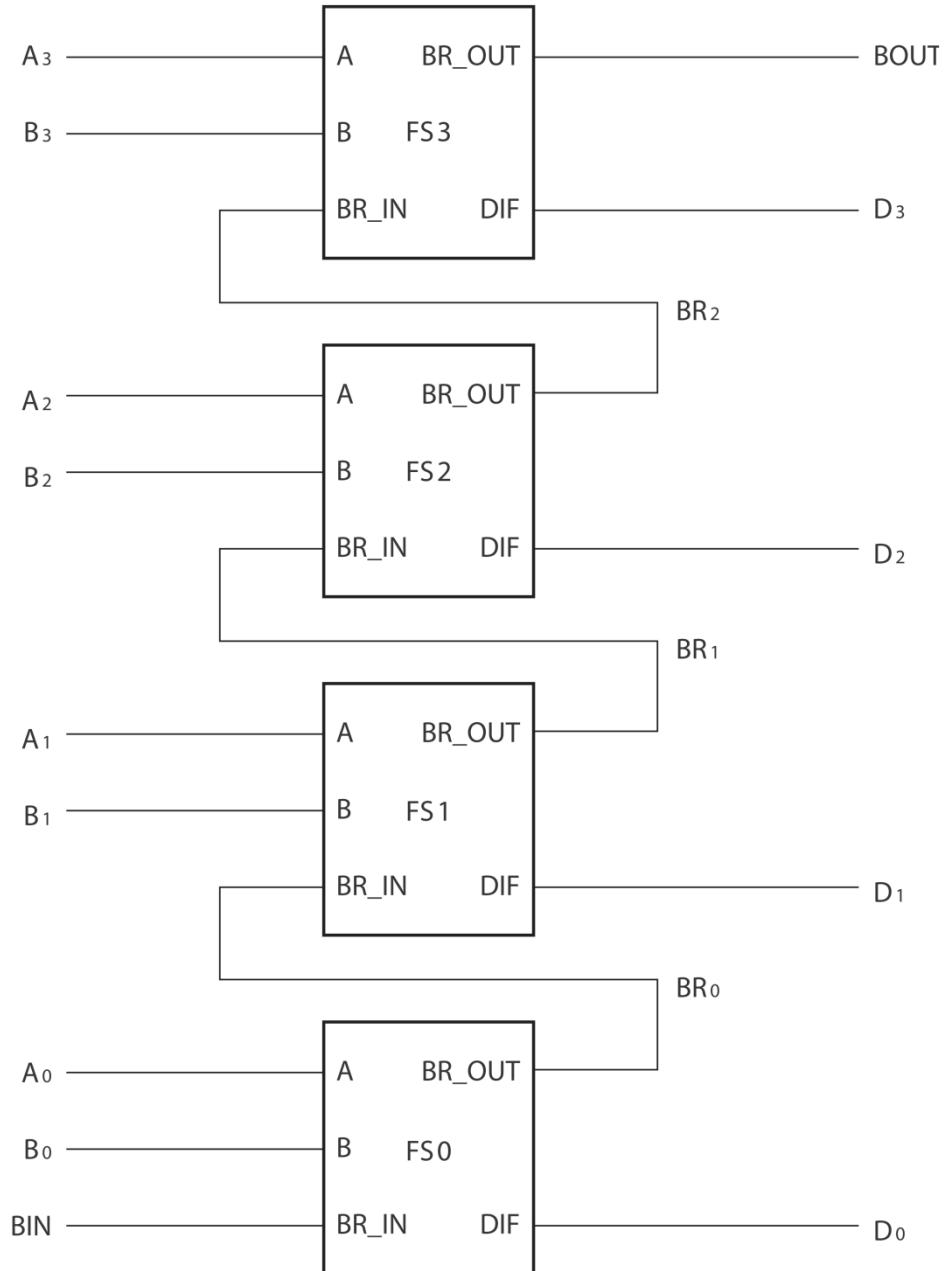
本章將建立 4 位元的減法電路，並且說明利用補數原理設計的加減法電路以及帶正負號運算電路設計上的差異。

- 10.1 使用全減法器的減法電路
- 10.2 使用補數的加減法電路
- 10.3 `unsigned` 與 `signed`
- 10.4 延伸學習

10.1 使用全減法器的減法電路

- 2 進制的減法，是從高位元借來（借位）作下一個位元的減法
- 此運算要使用到 4 個全減法器來實現 2 進制 4 位元的減法電路

4 位元減法電路



▲ 圖 10.1 4 位元減法電路

- 程式根據圖 10.1 把必須的信號對全減法器做元件宣告

程式 10.2 4 位元減法電路的 Verilog HDL 描述

LST10_2.V

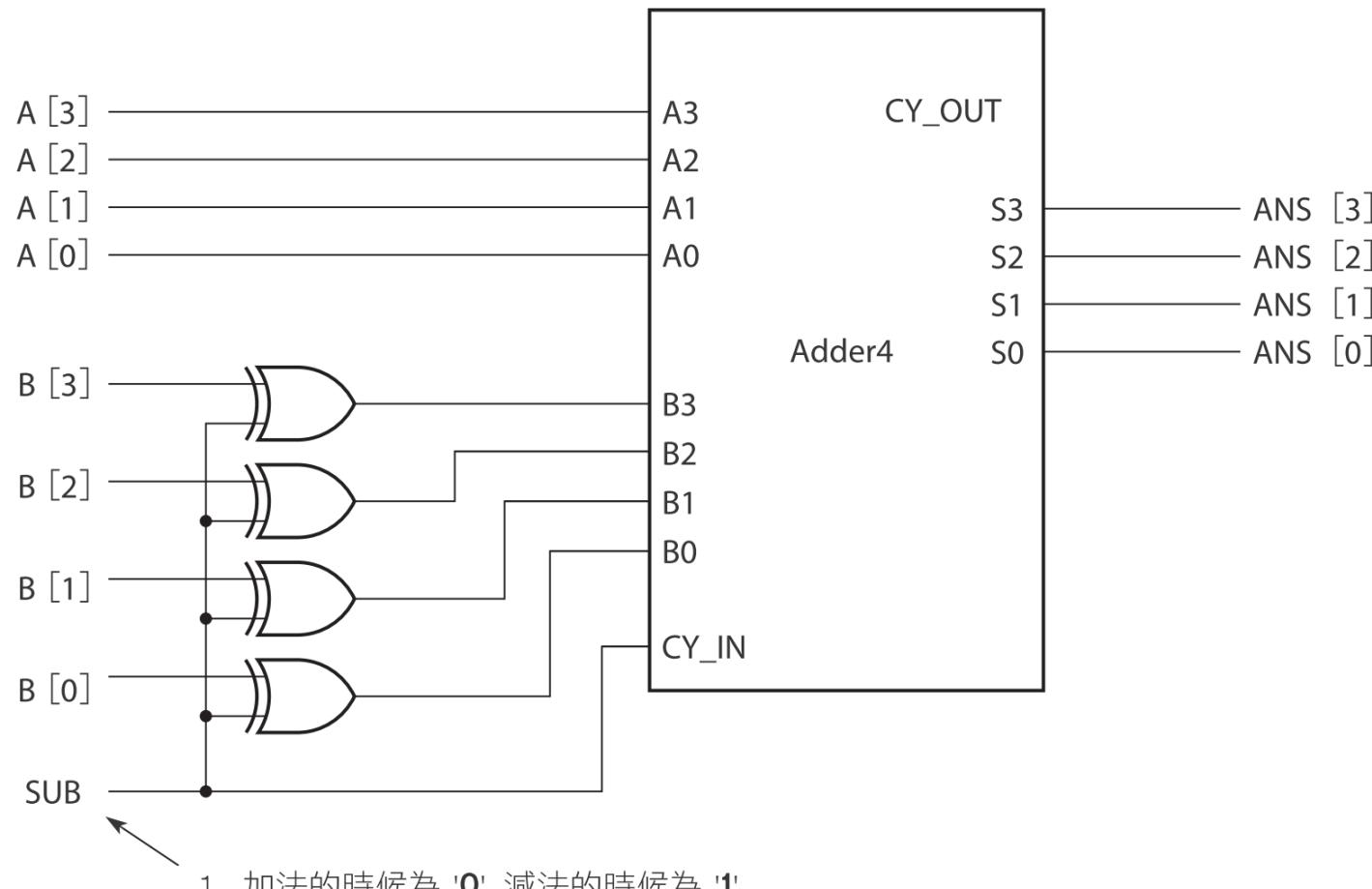
```
module Subtractor4 ( A, B, BIN, DIF, BOUT );
  input [3:0]      A, B;
  input BIN;
  output [3:0]     DIF;
  output BOUT;
  wire [3:0]       BR;
    FullSubtractor FS0  ( A[0], B[0],   BIN, DIF[0], BR[0] ),
                     FS1  ( A[1], B[1], BR[0], DIF[1], BR[1] ),
                     FS2  ( A[2], B[2], BR[1], DIF[2], BR[2] ),
                     FS3  ( A[3], B[3], BR[2], DIF[3], BOUT );
endmodule
```

10.2 使用補數的加減法電路

- 減法可以用補數相加來實現
 - $B+B-1=0$
 - $B=0-B-1$
- $A-B$ 的減法變成由 A 與 B 的補數相加來求出：
 - $A-B=A-(0-B-1)=A+(B-1)$
- 「2的補數」是「1的補數」加 '1'
- 「1的補數」則是 1 / 0 反轉得到

1 的補數

- 要做 1 的補數，只需要加 NOT 電路即可；但此處是用一個電路處理加減法，因此要改用 XOR
- 根據輸入「SUB」來決定是否要反轉：
 - $SUB = '0'$ → 相加動作 → 'B' 不需要反轉
 - $SUB = '1'$ → 相減動作 → 'B' 需要反轉



▲ 圖 10.3 用 2 的補數來做加減法電路

2 的補數

- 要得到 2 的補數，要把 4 位元加法電路「Adder4」的「CY_IN」連接到 ‘1’ 上面
- 當然也只有減法時才需要，所以要把「CY_IN」接到「SUB」即可

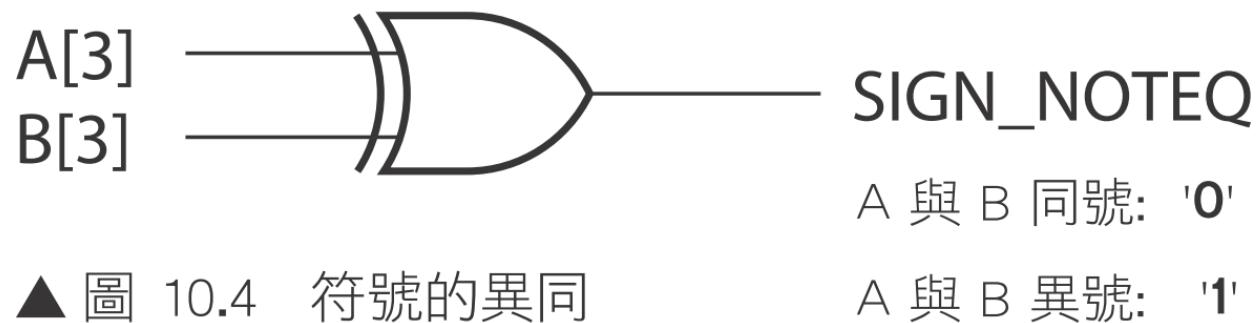
進位/借位與正負號

相加結果 位元 4	相加結果 位元 3	符號的異同	進位/借位結果	結果的符號
0	0	同號	無	正
0	0	異號	—	—
0	1	同號	有	正
0	1	異號	無	負
1	0	同號	有	負
1	0	異號	無	正
1	1	同號	無	負
1	1	異號	—	—

▲ 表 10.1 2 的補數的加減法

↑ ↑ ↑
同號: '0' 無: '0'
異號: '1' 有: '1'
 正: '0'
 負: '1'

- 「相加結果位元 4」是電路的「CY_OUT」
- 「相加結果位元 3」是電路的「S3」
- 「符號的異同」是表示 A 與 B 有沒有同號
- 用補數表示數值, MSB (最高位元) 用來表示符號



▲ 圖 10.4 符號的異同

(a) 進位/借位 (CY_BR_OUT)

		b3	b4	SIGN_NOTEQ	
		00	01	11	10
0	0		—		1
	1	1		—	

$$\begin{aligned} CY_BR_OUT = & b4 \cdot \overline{b3} \cdot \overline{SIGN_NOTEQ} \\ & + \overline{b4} \cdot b3 \cdot \overline{SIGN_NOTEQ} \end{aligned}$$

$$'_BR_OUT = (b4 \oplus b3) \cdot \overline{SIGN_NOTEQ}$$

(b) 結果的符號 (ANS_SIGN)

		b3	b4	SIGN_NOTEQ	
		00	01	11	10
0	0		—	1	
	1	1		—	1

$$ANS_SIGN = \overline{b4} \cdot SIGN_NOTEQ + b4 \cdot \overline{SIGN_NOTEQ}$$

$$ANS_SIGN = b4 \oplus SIGN_NOTEQ$$

用 2 的補數與 XOR 電路做成的 1 的補數

- 上述電路因為 XOR 的輸出是 1 的補數，所以有可能會無法正確判斷「符號的異同」

10 進制	2 的補數	套用 XOR 的 1 的補數	10 進制符號反轉之後的 2 的補數
+7	0111	1000	1001
+6	0110	1001	1010
+5	0101	1010	1011
+4	0100	1011	1100
+3	0011	1100	1101
+2	0010	1101	1110
+1	0001	1110	1111
+0	0000	1111	* 0000
-1	1111	0000	0001
-2	1110	0001	0010
-3	1101	0010	0011
-4	1100	0011	0100
-5	1011	0100	0101
-6	1010	0101	0110
-7	1001	0110	0111
-8	1000	0111	4 位元不能表示

▲表 10.3 2 的補數與套用 XOR 的 1 的補數

- B 為 '0' 的時候，XOR 電路的輸出為「1111」，會被誤認為是負值，甚至 B 為「-8」的時候，其補數無法用 4 位元的 2 的補數來表示

- 解決問題的電路如圖10.7, 雖然省略掉一些細節, 不過減法時, 發現 B 為 '0' 會反映到「CY_BR_OUT」

用補數做成的加減法電路的 Verilog HDL

程式 10.5 用 2 的補數做成的加減法電路的 Verilog HDL 描述

LST10_5.V

```
`module ADD_SUB ( A, B, SUB, ANS, CY_BR_OUT);
input [3:0] A, B;
input SUB;
output [3:0] ANS;
output CY_BR_OUT;
wire SIGN_NOTEQ;
wire SIGN_REVISE;
wire ADDER_CY_OUT;
wire ANS_SIGN;
wire [3:0] ADDER_B_IN;
wire [3:0] ADDER_S_OUT;

ADDER4 ADDER4 ( A, ADDER_B_IN, SUB, ADDER_S_OUT, ADDER_CY_OUT );

assign ADDER_B_IN = B ^ { SUB, SUB, SUB, SUB };
assign SIGN_REVISE = ADDER_B_IN[3] & ADDER_B_IN[2]
& ADDER_B_IN[1] & ADDER_B_IN[0] & SUB;
assign SIGN_NOTEQ = ADDER_B_IN[3] ^ A[3];
assign ANS_SIGN = ADDER_CY_OUT ^ SIGN_NOTEQ;
assign ANS = { ANS_SIGN, ADDER_S_OUT[2:0] };
assign CY_BR_OUT = (ADDER_CY_OUT ^ ADDER_S_OUT[3]) & (~SIGN_NOTEQ )
& (~SIGN_REVISE);

endmodule
```

使用加法運算子的 Verilog HDL 描述

程式 10.6 用 2 的補數做成加減法電路的 Verilog HDL 描述 (使用加減法運算子)

```
module ADD_SUB      ( A, B, SUB, ANS, CY_BR_OUT);                      LST10_6.V
  input  *1signed [3:0]      A, B;
  input                               SUB;
  output *1signed [3:0]      ANS;
  output                           CY_BR_OUT;

  *1assign { CY_BR_OUT, ANS } = FUNC_ADD_SUB ( A, B );

  function signed [4:0]      FUNC_ADD_SUB;
    input   signed [3:0]      A, B;
    reg     signed [4:0]      TOTAL;
```

- 要處理有帶符號的數值需要使用到保留字的「`signed`」在「`range`」的前面先描述。

```

begin
    if ( !SUB )
        *2 → TOTAL = A + B;
    else
        → TOTAL = A - B;

    if ( TOTAL < -8 )
        begin
            *3 { TOTAL = TOTAL + 8;
                  FUNC_ADD_SUB = { 1'b1, TOTAL[3:0] };
            end
        else if ( TOTAL > 7 )
            begin
                *4 { TOTAL = TOTAL - 8;
                      FUNC_ADD_SUB = { 1'b1, TOTAL[3:0] };
            end
        else
            *5 FUNC_ADD_SUB = { 1'b0, TOTAL[3:0] };
    end
endfunction

endmodule

```

- 加減法的結果是用阻礙指定代入到TOTAL裡面

```

begin
    if ( !SUB )
        *2 → TOTAL = A + B;
    else
        → TOTAL = A - B;

    if ( TOTAL < -8 )
        begin
            *3 { TOTAL = TOTAL + 8;
                  FUNC_ADD_SUB = { 1'b1, TOTAL[3:0] };
        end
    else if ( TOTAL > 7 )
        begin
            *4 { TOTAL = TOTAL - 8;
                  FUNC_ADD_SUB = { 1'b1, TOTAL[3:0] };
        end
    else
        *5 FUNC_ADD_SUB = { 1'b0, TOTAL[3:0] };

endfunction

endmodule

```

- TOTAL的值如果比「-8」還小,會把 function 的回傳值「CY_BR_OUT」變為 ON, 並把 TOTAL 的值加 '8' 代入「ANS」

```

        else if ( TOTAL > 7 )
            begin
                * 4 { TOTAL = TOTAL - 8;
                end
            else
                * 5 FUNC_ADD_SUB = { 1'b1, TOTAL[3:0] };
        end
    endfunction

endmodule

```

- TOTAL的值比 '7' 還大, 除了「CY_BR_OUT」變為 ON, 也會把TOTAL減 '8' 代入到「ANS」
- TOTAL」的值不是前述情況, 「CY_BR_OUT」為 OFF, 直接把「TOTAL」的值代入到「ANS」裡面

變更過 +8, -8 的 Verilog HDL 描述

- 前述程式有對「TOTAL」做 +8/-8 的動作,所以電路會變得很複雜
- 我們可以將這部分的電路,改以以下邏輯實作
 - +8 → TOTAL 的 2 進制 5 位元的「01000」做 OR
 - 8 → TOTAL 的 2 進制 5 位元的「10111」做 AND
- 修改後的結果如程式 10.7

用 integer 型變數作成的 Verilog HDL

- 我們也可以將前述程式中的「TOTAL」改用 integer 型態

程式 10.8 用 2 的補數作成的加減法電路的 Verilog HDL 描述 (integer 型變數) LST10_8.V

```
module ADD_SUB      ( A, B, SUB, ANS, CY_BR_OUT);
  input   signed [3:0]      A, B;
  input                  SUB;
  output   signed [3:0]      ANS;
  output                  CY_BR_OUT;

  assign { CY_BR_OUT, ANS } = FUNC_ADD_SUB ( A, B );

function signed [4:0] FUNC_ADD_SUB;
  input   signed [3:0]      A, B;
  integer                  TOTAL;
```

```

begin
    if ( !SUB )
        TOTAL = A + B;
    *1 else
        TOTAL = A - B;

    if ( TOTAL < -8 )
        begin
            TOTAL = TOTAL + 8;
            FUNC_ADD_SUB = { 1'b1, TOTAL[3:0] };
        end
    *2 else if ( TOTAL > 7 )
        begin
            TOTAL = TOTAL - 8;
            FUNC_ADD_SUB = { 1'b1, TOTAL[3:0] };
        end
    else
        FUNC_ADD_SUB = { 1'b0, TOTAL[3:0] };
end
endfunction

endmodule

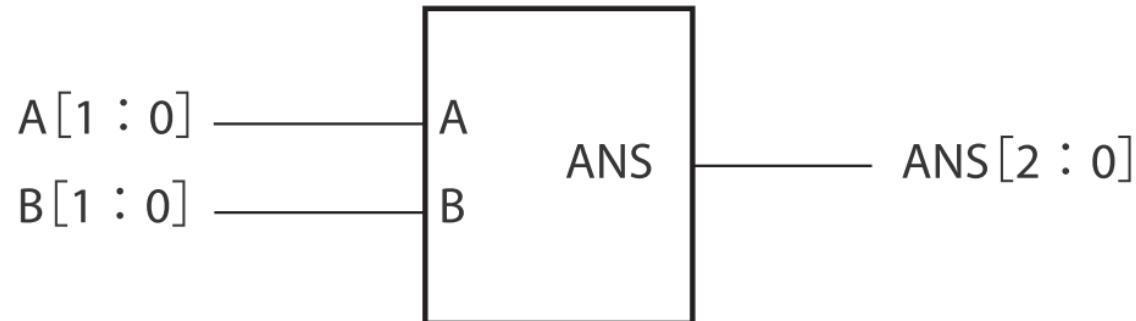
```

- A 與 B 的加法或減法運算都代入到 integer 型變數的「TOTAL」，在 function 的輸入宣告的時候把 A 與 B 宣告為「signed」，所以在代入的時候會自動的作符號擴張

10.3 unsigned 與 signed

- 在 Verilog HDL 裡面，需要考慮符號的時候用「`signed`」做描述，不需要考慮符號的時候就使用「`unsigned`」

unsigned



▲ 圖 10.9 unsigned 的加法電路

- 上圖輸入的 A 與 B 因為是 unsigned 2 位元, 所以可能的輸出就是從 0~3；ANS 輸出結果從0~6

程式 10.10 unsigned 加法電路的 Verilog HDL 描述

LST10_10.V

```
module ADDER      ( A, B, ANS );
input    [1:0]    A, B;
output   [2:0]    ANS;

      assign ANS = A + B;

endmodule
```

signed

- 同一個電路，輸入的 A 與 B 是 signed 2 位元，有可能從 -2~+1，所以ANS 輸出可能從 +2~-4
- 撰寫程式時，A 與 B 不需要描述成 signed，用
- always 來令 B 或 A 做遞增，然後在下層模組的「ADDER」宣告 signed

程式 10.11 signed 加法電路的 Verilog HDL 描述

LST10_11.V

```
module ADDER      ( A, B, ANS );
  input   signed [1:0]      A, B;
  output  signed [2:0]      ANS;

  assign ANS = A + B;

endmodule
```