



正確學會

改訂新版 デジタル回路と Verilog HDL

Verilog

的16堂課

第 9 章

基本運算電路

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

數位電路的運算是使用 2 進制的，基本運算電路包含有可以做 1 位元的加減法的半加法器，全加法器，半減法器，全減法器。

- 9.1 基本運算電路
- 9.2 加法電路

9.1 基本運算電路

- 9.1.1 半加法器
- 9.1.2 全加法器
- 9.1.3 半減法器
- 9.1.4 全減法器

9.1.1 半加法器

- 半加法器是沒有低位元加法且進位到高位元功能的加法器
- 2 進制 1 位元的加法，其可能的 4 種組合如下：
 - ▣ $0+0=0$
 - ▣ $0+1=1$
 - ▣ $1+0=1$
 - ▣ $1+1=0$ 且進位
- 根據上述推導，來作成真值表：

(a) 真值表

| B | A | CY_OUT | SUM |
|---|---|--------|-----|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

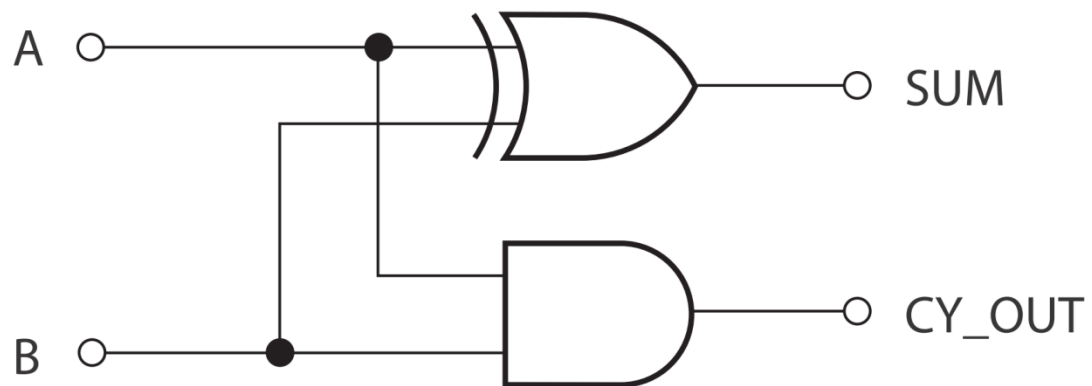
進位「CY_OUT」

| B \ A | 0 | 1 |
|-------|---|---|
| | 0 | 1 |
| 0 | | |
| 1 | | 1 |

和「SUM」

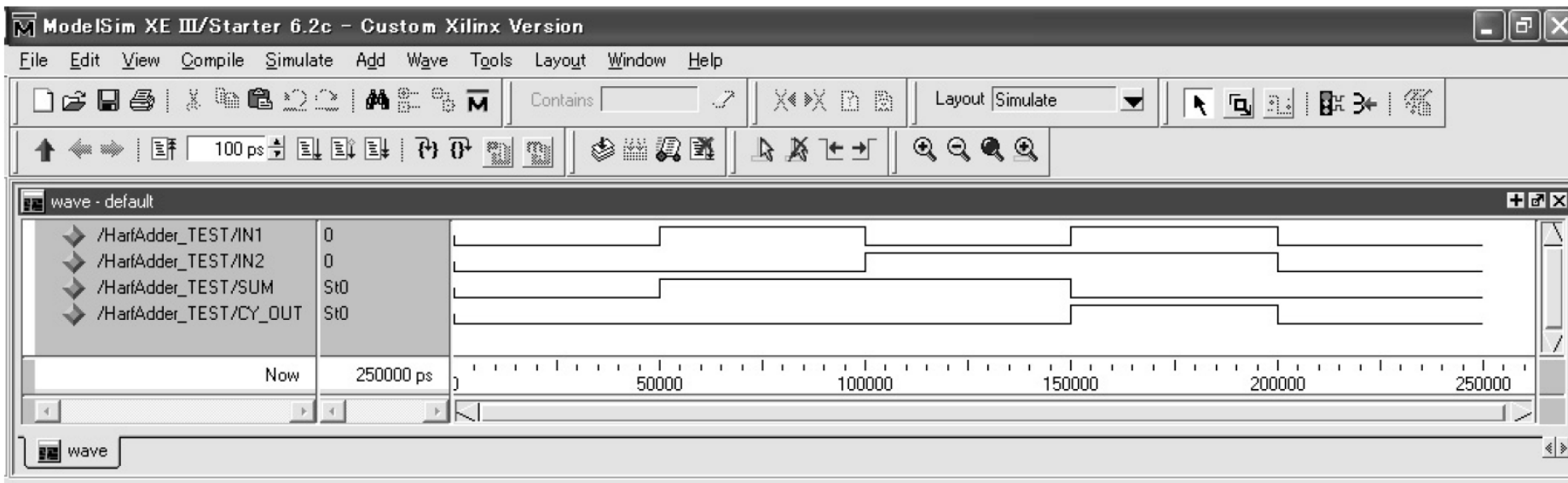
| B \ A | 0 | 1 |
|-------|---|---|
| | 0 | 1 |
| 0 | | 1 |
| 1 | 1 | |

(c) 電路圖



位元運算子做成的半加法器的 Verilog HDL 描述

- 「SUM」與「CY_OUT」分別使用到位元運算子 '^' 跟 '&'



▲ 圖 9.2 半加法器的模擬結果

用加法運算子 '+' 來作半加法器的 Verilog HDL 描述

程式 9.3 加法器的 Verilog HDL (行為層)

LST9_3.V

```
module HalfAdder ( A, B, SUM, CY_OUT );
input A, B;
output SUM, CY_OUT;
*1 wire [1:0] TOTAL;
    assign TOTAL = A + B;
    *2 [assign SUM = TOTAL[0];
      assign CY_OUT = TOTAL[1];
endmodule
```

- 1. 加法的結果有「SUM」跟「CY_OUT」兩位元的輸出，所以中間信號(網絡)的「TOTAL」就定義為兩位元大小的匯流排
- 2. 把「TOTAL」的低位元帶入到「SUM」，高位元帶入到「CY_OUT」。

9.1.2 全加法器

- 要做 2 進制的加法，必須包含低位元進位的計算，其 8 種組合如下：

- $0 + 0 + 0 = 0$

- $0 + 0 + 1 = 1$

- $0 + 1 + 0 = 1$

- $0 + 1 + 1 = 0$ 且進位

- $1 + 0 + 0 = 1$

- $1 + 0 + 1 = 0$ 且進位

- $1 + 1 + 0 = 0$ 且進位

- $1 + 1 + 1 = 1$ 且進位

- 根據上面的推導，求得真值表、邏輯式、電路圖如下：

(a) 真值表

| CY_IN | B | A | CY_OUT | SUM |
|-------|---|---|--------|-----|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

(b) 卡諾圖

進位「CY_OUT」

| CY_IN \ B | A | 00 01 11 10 | | | |
|-----------|---|----------------------------|----|----|----|
| | | 00 | 01 | 11 | 10 |
| 0 | | | | 1 | |
| 1 | | | 1 | 1 | 1 |

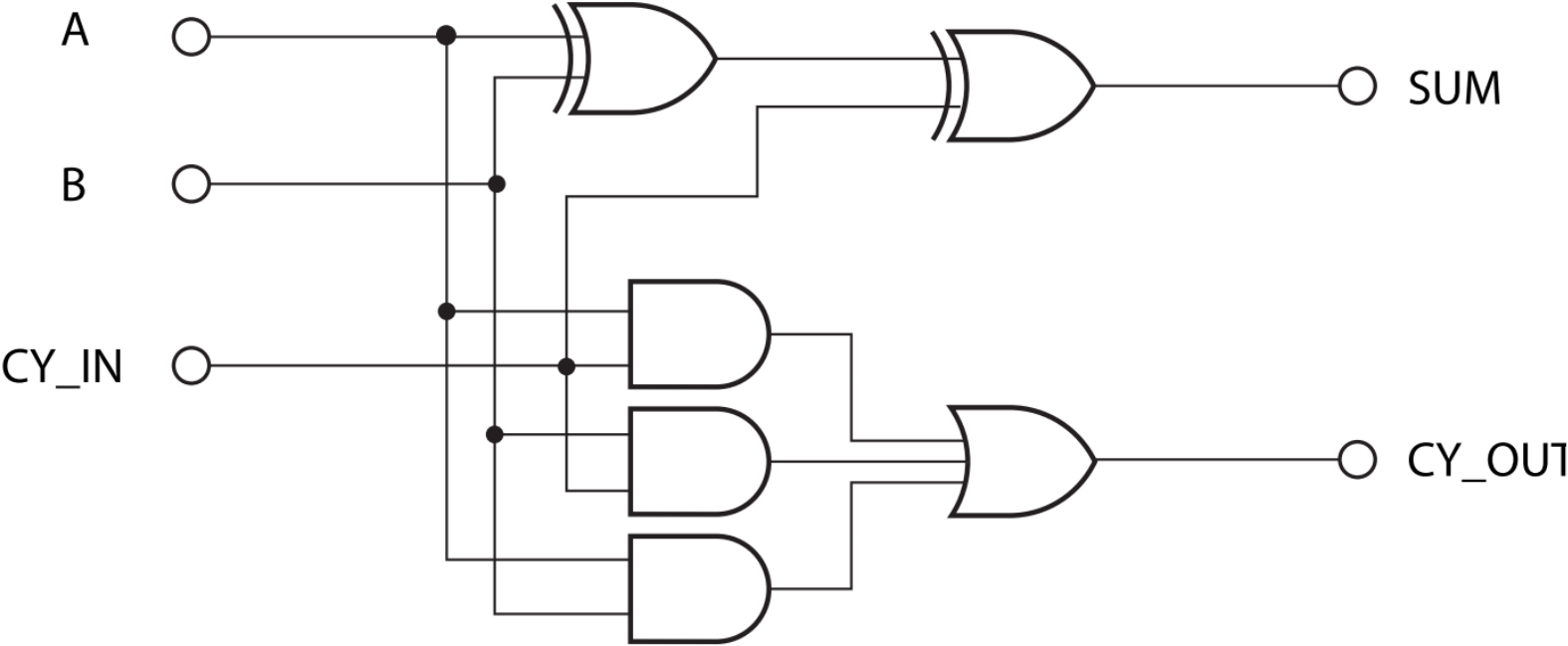
和「SUM」

| CY_IN \ B | A | 00 01 11 10 | | | |
|-----------|---|----------------------------|----|----|----|
| | | 00 | 01 | 11 | 10 |
| 0 | | | 1 | | 1 |
| 1 | | 1 | | 1 | |

$$\text{SUM} = A \oplus B \oplus \text{CY_IN}$$

$$\text{CY_OUT} = A \cdot \text{CY_IN} + B \cdot \text{CY_IN} + A \cdot B$$

(c) 電路圖



用兩個半加法器組成全加法器

- 全加法器可由兩個半加法器跟一個 OR 電路構成

$$CY0 = A \cdot B$$

$$SUM0 = A \oplus B$$

$$CY1 = (A \oplus B) \cdot CY_IN$$

$$= A \cdot \bar{B} \cdot CY_IN + \bar{A} \cdot B \cdot CY_IN$$

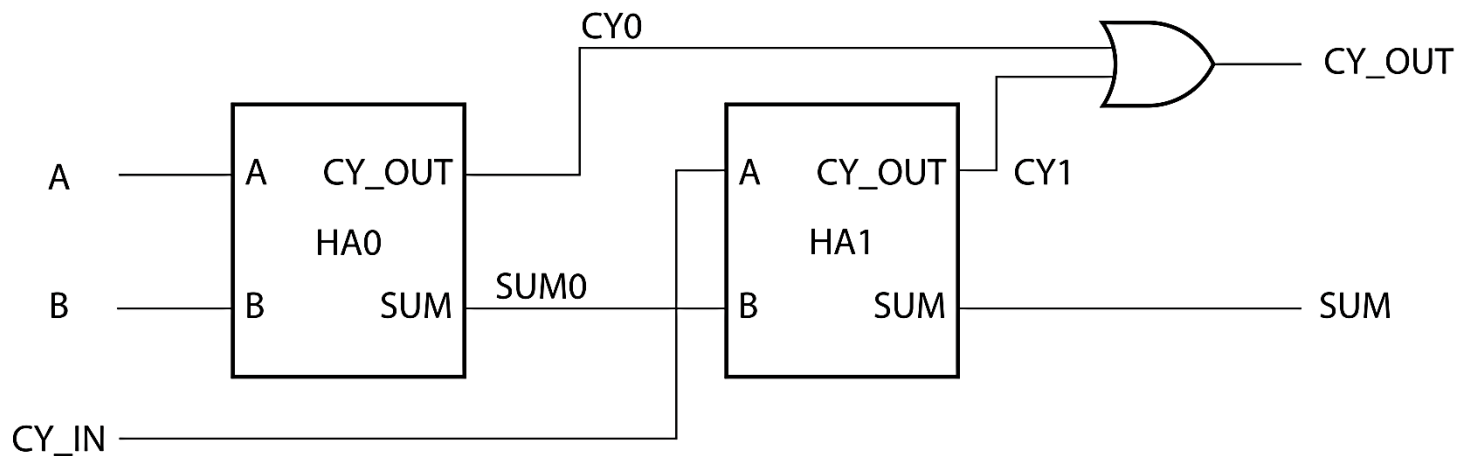
$$CY_OUT = CY1 + CY0$$

$$= A \cdot \bar{B} \cdot CY_IN + \bar{A} \cdot B \cdot CY_IN + A \cdot B$$

(從這個邏輯式做成卡諾圖的話就會如圖 9.3(b))

$$= A \cdot CY_IN + B \cdot CY_IN + A \cdot B$$

$$SUM0 = A \oplus B \oplus CY_IN$$



▲ 圖 9.4 用半加法器來兜全加法器

| CY_IN | B | A | CY_OUT | CY1 | SUM | CY0 | SUM0 |
|-------|---|---|--------|-----|-----|-----|------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |

▲ 表 9.1 用半加法器來兜全加法器

用兩個半加法器組成全加法器的 Verilog HDL 描述

- 用半加法器兜成的全加法器的 Verilog HDL 描述可利用模組、元件宣告語法呼叫下層模組「HalfAdder」兩次
- 注意簡碼名要各自獨立成「HA0」,「HA1」

9.1.3 半減法器

- 2 進位 1 位元的減法, 通常會有以下 4 種可能的情況
 - $1 - 1 = 0$
 - $1 - 0 = 1$
 - $0 - 1 = 1$ 且借位
 - $0 - 0 = 0$
- 被減數為 'A', 減數為 'B', 高位元借位為「BR_OUT」, 差 (difference) 為「DIF」

(a) 真值表

| B | A | BR_OUT | DIF |
|---|---|--------|-----|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |

(b) 卡諾圖

借位「BR_OUT」

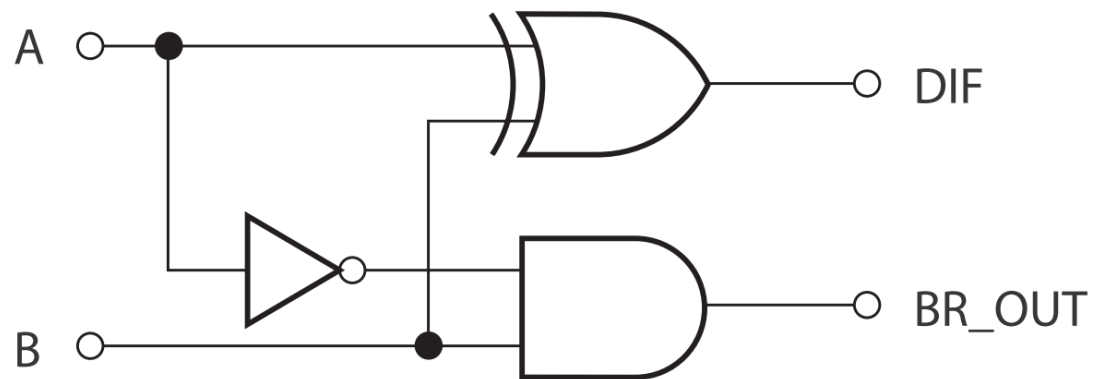
| | | A | |
|---|---|---|---|
| | | 0 | 1 |
| B | 0 | | |
| | 1 | 1 | |

差「DIF」

| | | A | |
|---|---|---|---|
| | | 0 | 1 |
| B | 0 | | 1 |
| | 1 | 1 | |

- 半減器電路

(c) 電路圖

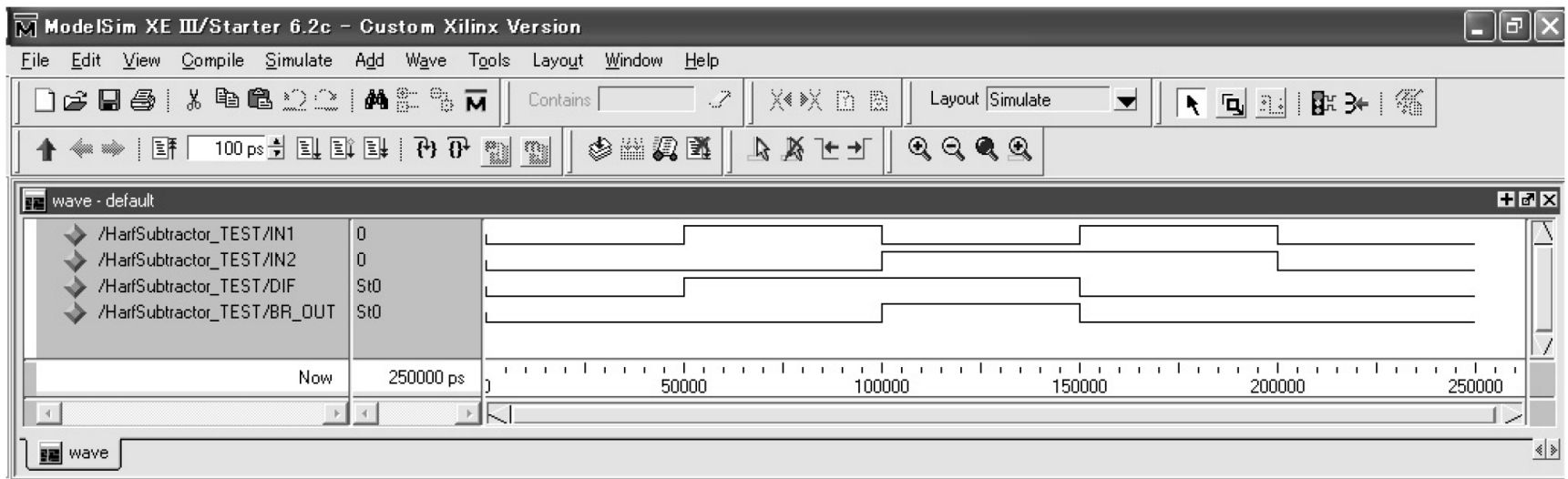


以位元運算子做的半減法器的 Verilog HDL 描述

程式 9.9 半減法器的 Verilog HDL 描述 (資料流層)

LST9_9.V

```
module HalfSubtractor ( A, B, DIF, BR_OUT );  
input A, B;  
output DIF, BR_OUT;  
    assign DIF = A ^ B;  
    assign BR_OUT = ( ~A ) & B;  
endmodule
```



▲ 圖 9.10 半減法器的模擬結果

使用減法運算子 '-' 做成的半減法器的 Verilog HDL 描述

程式 9.10 半減法器的 Verilog HDL 描述 (行為層)

LST9_10.V

```
module HalfSubtractor ( A, B, DIF, BR_OUT );  
input A, B;  
output DIF, BR_OUT;  
wire [1:0] DIFFERENCE;  
    assign DIFFERENCE = A - B;  
    assign DIF = DIFFERENCE[0];  
    assign BR_OUT = DIFFERENCE[1];  
endmodule
```

9.1.4 全減法器

- 要執行 2 進位的減法，需要跟低位元借位來做運算的能力，共計會有以下 8 種可能：
 - $1 - 1 - 1 = 1$ 且借位
 - $1 - 1 - 0 = 0$
 - $1 - 0 - 1 = 0$
 - $1 - 0 - 0 = 1$
 - $0 - 1 - 1 = 0$ 且借位
 - $0 - 1 - 0 = 1$ 且借位
 - $0 - 0 - 1 = 1$ 且借位
 - $0 - 0 - 0 = 0$

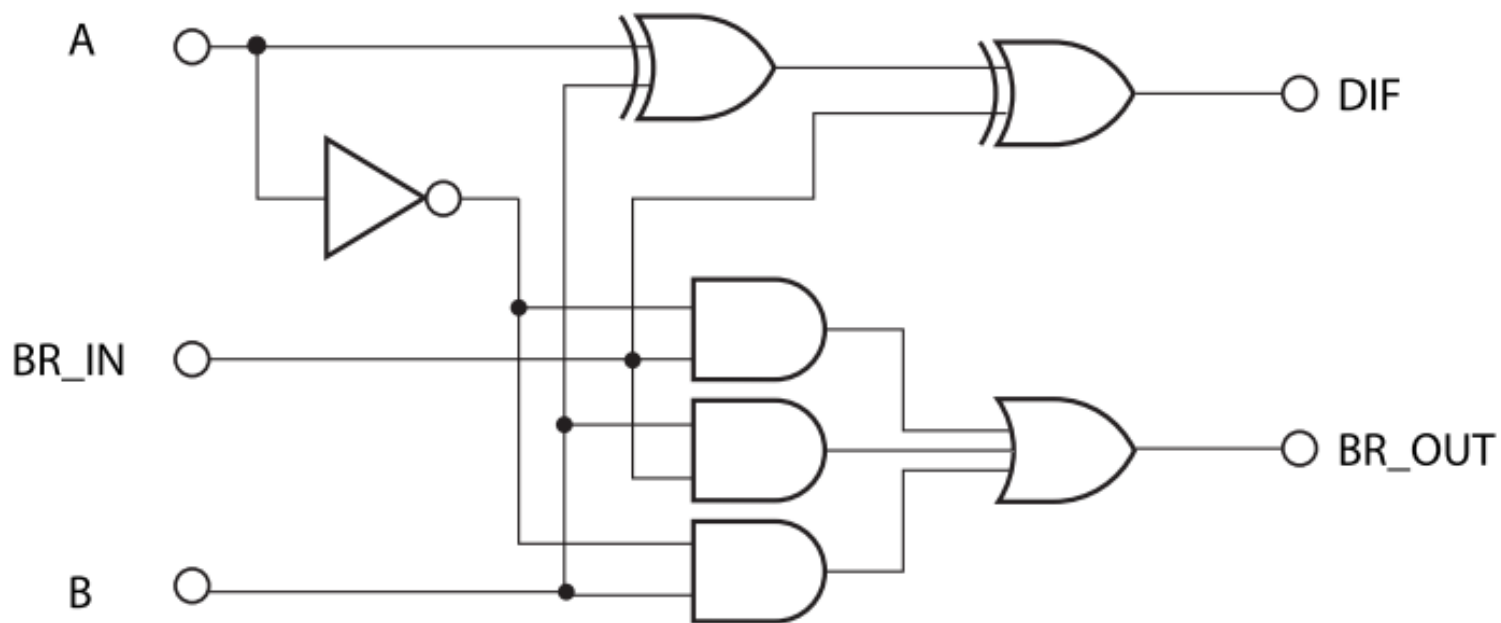
(a) 真值表

| BR_IN | B | A | BR_OUT | DIF |
|-------|---|---|--------|-----|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

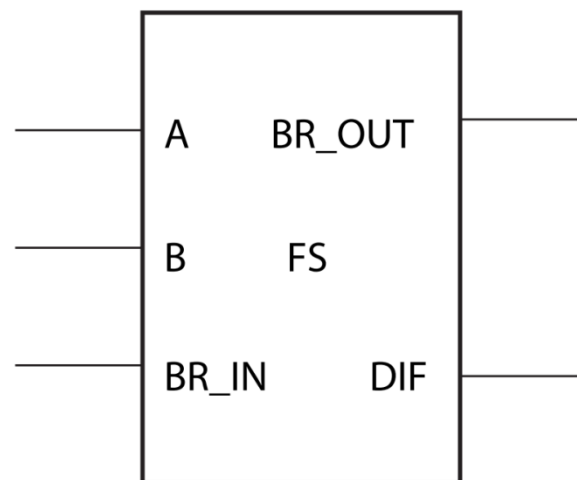
差 (difference) $DIF = A \oplus B \oplus BR_IN$

高位元借位 (borrow) $BR_OUT = \overline{A} \cdot B + \overline{A} \cdot BR_IN + B \cdot BR_IN$

(c) 電路圖



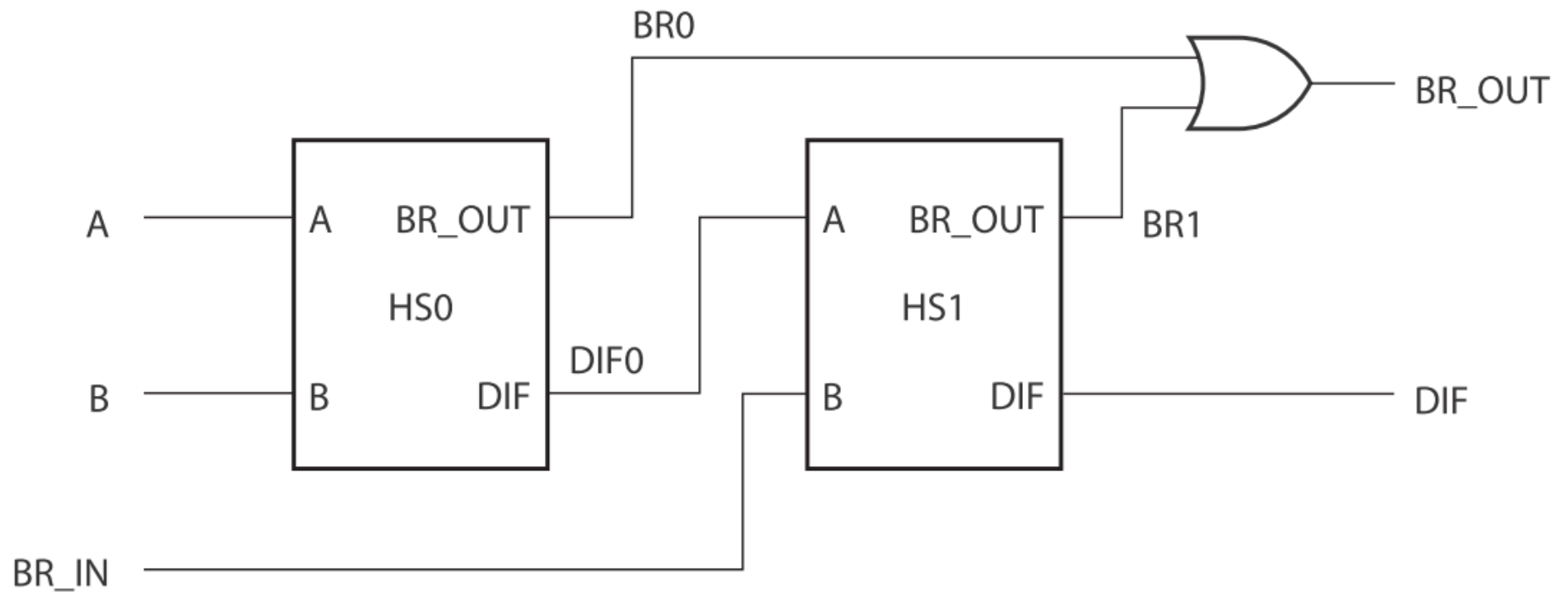
- 本書全減法器電路符號：



FS : Full Subtractor

用半減法器實現全減法器

- 全減法器可用兩個半減法器與一個OR電路來構成
 - ▣ 第一級的半減法器 (HS0) 的輸入為 'A' 與 'B' 的 DIF, 運算結果的借位會從「BR0」, 差會從「DIF0」輸出
 - ▣ 第 2 級的半減法器 (HS1) 輸入為「BR_IN」, 「DIF0」, 運算結果的借位會從「BR1」, 差會從「DIF」輸出



用位元運算子實現全減法器的 Verilog HDL 描述

程式 9.12 全減法器的 Verilog HDL 描述 (資料流層)

LST9_12.V

```
module FullSubtractor      ( A, B, BR_IN, DIF, BR_OUT );
input  A, B, BR_IN;
output DIF, BR_OUT;
    assign DIF      = A ^ B ^ BR_IN;
    assign BR_OUT = ( ~A & B ) | ( ~A & BR_IN ) | ( B & BR_IN );
endmodule
```

用減法演算子 '-' 做成全減法器的 Verilog HDL 描述

程式 9.13 全減法器的 Verilog HDL 描述 (行為層)

LST9_13.V

```
module FullSubtractor ( A, B, BR_IN, DIF, BR_OUT );
input A, B, BR_IN;
output DIF, BR_OUT;
wire [1:0] DIFFERENCE;
assign DIFFERENCE = A - B - BR_IN;
assign DIF = DIFFERENCE[0];
assign BR_OUT = DIFFERENCE[1];
endmodule
```

用半減法器實現全減法器的 Verilog HDL 描述

- 用模組與元件宣告語法來呼叫下層模組「HalfSubstrator」兩次
- 元件名要分別獨立為「HS0」與「HS1」
- 埠的宣告要把必需的信號跟埠做連接

程式 9.14 用半減法器實現全減法器的 Verilog HDL 描述

LST9_14.V

```
module FullSubtractor ( A, B, BR_IN, DIF, BR_OUT );
input A, B, BR_IN;
output DIF, BR_OUT;
wire [1:0] DIFFERENCE;
    assign DIFFERENCE = A - B - BR_IN;
    assign DIF = DIFFERENCE[0];
    assign BR_OUT = DIFFERENCE[1];
endmodule
```

9.2 加法電路

9.2.1 使用全加法器實現的加法電路

- 2 進位的加法，為實現低位元的進位 (carry) 與下一個位元的加法，一個 2 進制 4 位元的加法電路使用到 4 個全加法器
- 使用的全加法器和位元數一樣，所有位元的加法會同時進行，所以稱為**並列加法電路**
- 只用 1 個全加法器，透過移位暫存器來控制被加數 A 與加數 B，一次只運算 1 個位元的電路則稱為**串列加法電路**

使用邏輯運算子的 Verilog HDL 描述

程式 9.16 4 位元加法電路的 Verilog HDL 描述 (使用全加法電路)

LST9_16.V

```
module    ADDER4      ( A_IN, B_IN, CIN, S, COUT );

input     [3:0]       A_IN, B_IN;
input     CIN;
output    [3:0]       S;
output    COUT;
wire      [2:0]       CY;

    FullAdder FA0    ( A_IN[0], B_IN[0],    CIN, S[0], CY[0] );
    FullAdder FA1    ( A_IN[1], B_IN[1],    CY[0], S[1], CY[1] );
    FullAdder FA2    ( A_IN[2], B_IN[2],    CY[1], S[2], CY[2] );
    FullAdder FA3    ( A_IN[3], B_IN[3],    CY[2], S[3],    COUT );

endmodule
```

使用加法運算子的 Verilog HDL 描述

程式 9.17 4 位元加法電路的 Verilog HDL 描述 (行為層)

LST9_17.V

```
module    ADDER4    ( A_IN, B_IN, CIN, S, COUT );

input     [3:0]     A_IN, B_IN;
input     CIN;
output    [3:0]     S;
output    COUT;
wire      [4:0]     TOTAL;

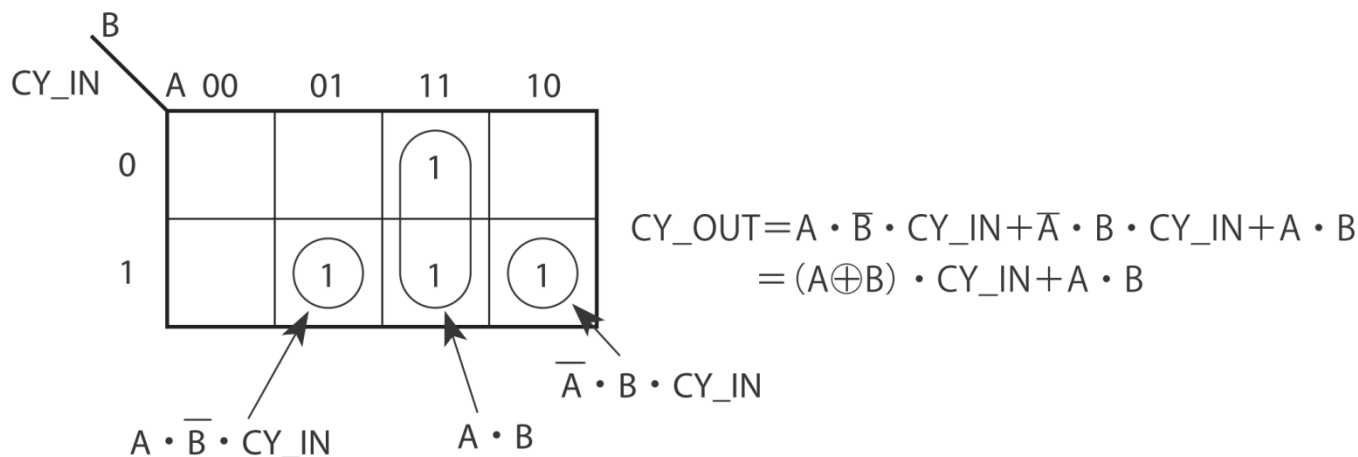
        assign      TOTAL = A_IN + B_IN + CIN;
        assign      S     = TOTAL[3:0];
        assign      COUT  = TOTAL[4];

endmodule
```

9.2.2 加法電路的高速化

- 加法電路，「由低位元的進位依序傳遞到高位元」的關係，運算速度會因此比較慢
- 要高速化加法電路，必須直接觀察低位元的資料來判斷有沒有發生進位。這種方式稱為**進位查詢方式** (carry look up) 或者**同時進位** (simultaneous carry)。

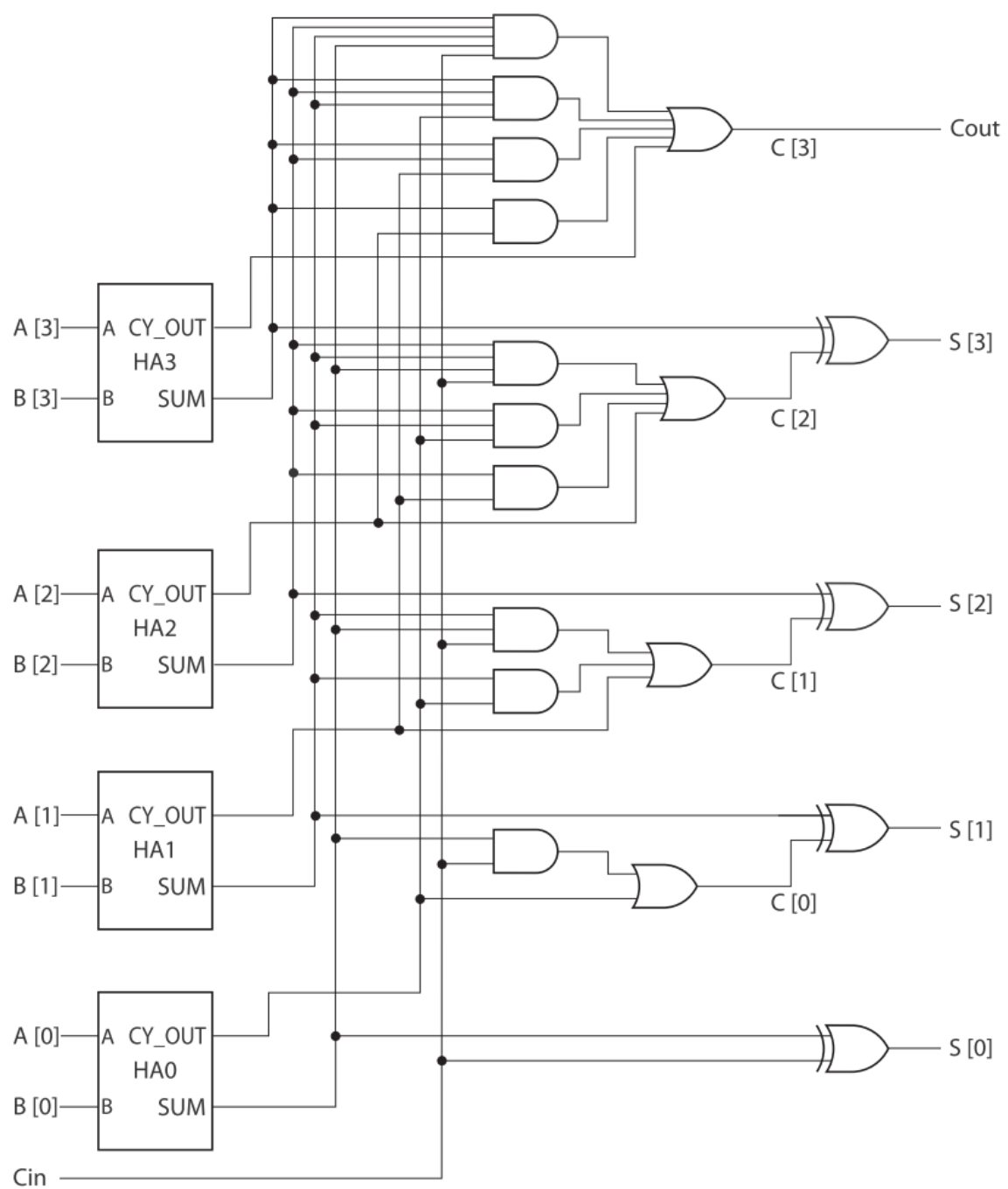
- 改變全加法器進位輸出的簡化方式，重新求邏輯式



▲ 圖 9.17 全加法器的進位的邏輯式

$$CY_OUT = (A \oplus B) \cdot CY_IN + A \cdot B$$

- 把前述的邏輯式放進 4 位元加法電路的各進位裡面，導出的邏輯式不會等待低位元的進位，而是直接判斷低位元資料有沒有進位的發生，所以運算速度可以加快



▲ 圖 9.18 同時進位方式的 4 位元加法電路

同時進位方式的加法電路的 Verilog HDL 描述

- 各信號用網路宣告

- 為半加法器的「SUM」宣告的「HA_SUM_OUT」
- 為半加法器的「CY_OUT」宣告的「HA_CY_OUT」
- 為了公式 (9.6) ~ 公式 (9.9) 進位而宣告的「CY」

```
module ADDER4      ( A_IN, B_IN, CIN, S, COUT );

input  [3:0]      A_IN, B_IN;
input   CIN;
output [3:0]      S;
output  COUT;

wire    [3:0]     HA_SUM_OUT;
wire    [3:0]     HA_CY_OUT;
wire    [3:0]     CY;

    HalfAdder HA0 (A_IN[0], B_IN[0], HA_SUM_OUT[0], HA_CY_OUT[0] );
    HalfAdder HA1 (A_IN[1], B_IN[1], HA_SUM_OUT[1], HA_CY_OUT[1] );
    HalfAdder HA2 (A_IN[2], B_IN[2], HA_SUM_OUT[2], HA_CY_OUT[2] );
    HalfAdder HA3 (A_IN[3], B_IN[3], HA_SUM_OUT[3], HA_CY_OUT[3] );

assign S[0]=HA_SUM_OUT[0] ^ CIN;
assign S[1]=HA_SUM_OUT[1] ^ CY[0];
assign S[2]=HA_SUM_OUT[2] ^ CY[1];
assign S[3]=HA_SUM_OUT[3] ^ CY[2];
assign COUT=CY[3];
```

```
assign CY[0] = (HA_SUM_OUT[0] & CIN) | HA_CY_OUT[0];
assign CY[1] = (CIN & HA_SUM_OUT[0] & HA_SUM_OUT[1])
               | (HA_CY_OUT[0] & HA_SUM_OUT[1])
               | HA_CY_OUT[1];
assign CY[2] = (CIN & HA_SUM_OUT[0] & HA_SUM_OUT[1] & HA_SUM_OUT[2])
               | (HA_CY_OUT[0] & HA_SUM_OUT[1] & HA_SUM_OUT[2])
               | (HA_CY_OUT[1] & HA_SUM_OUT[2])
               | HA_CY_OUT[2];
assign CY[3] = (CIN & HA_SUM_OUT[0] & HA_SUM_OUT[1]
               & HA_SUM_OUT[2] & HA_SUM_OUT[3])
               | (HA_CY_OUT[0] & HA_SUM_OUT[1]
               & HA_SUM_OUT[2] & HA_SUM_OUT[3])
               | (HA_CY_OUT[1] & HA_SUM_OUT[2] & HA_SUM_OUT[3])
               | (HA_CY_OUT[2] & HA_SUM_OUT[3])
               | HA_CY_OUT[3];
```

```
endmodule
```

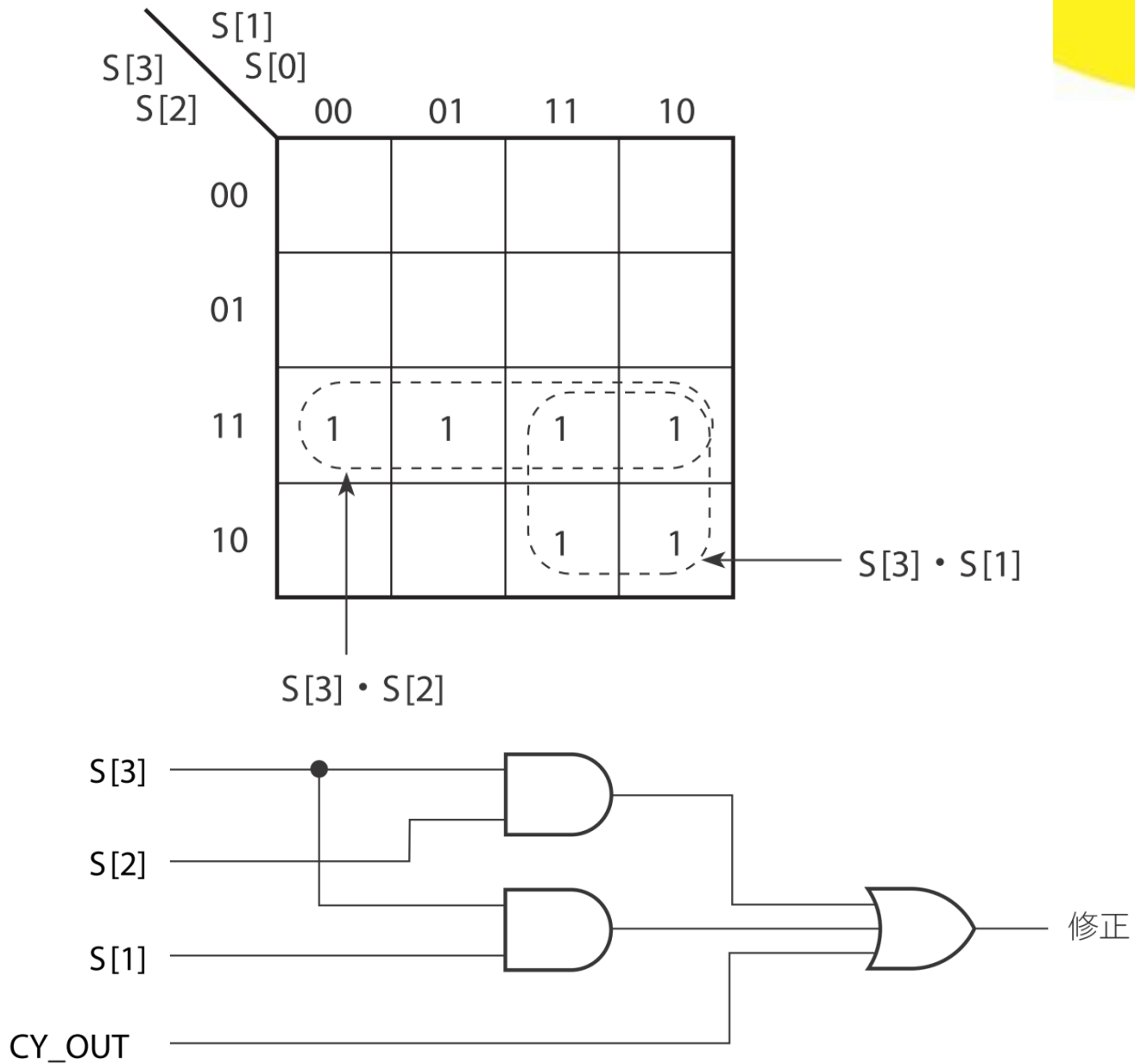
9.2.3 運算速度的驗證

- AND 電路 $\rightarrow 5.5$ [ns]
- OR 電路 $\rightarrow 5.8$ [ns]
- XOR 電路 $\rightarrow 6.5$ [ns]

- 全加法器做成的加法電路的運算速度
 - ▣ 全加法器的進位發生需要 $11.3[\text{ns}]$, 因為有 4 級, 一共需要花上 $45.2[\text{ns}]$
 - ▣ 以 3 級全加法器的進位跟第 4 級的 'S' 的延遲時間計算則為 $46.9[\text{ns}]$
- 用同時進位方式作成的加法電路的運算速度
 - ▣ 不管是哪個進位的邏輯式都是 AND-OR 形式, 所以都是 $11.3[\text{ns}]$, 第 4 級的 'S' 因為有經過一級的 XOR 電路所以需要 $6.5[\text{ns}]$, 總共為 $17.8[\text{ns}]$

9.2.4 BCD 加法電路

- 把 BCD 碼的資料單純地用 2 進制相加，並做適當的補正讓結果不至於跟 BCD 碼互相矛盾
- 2進位的相加結果如果介於‘0’~‘9’之間，就直接作為 BCD 相加結果。
- 2進位的相加結果如果大於「10」，就把結果 +6 之後作為 BCD 相加結果。



▲ 圖 9.24 BCD 修正檢查電路

BCD 加法電路的 Verilog HDL 描述

- 第 1 級的 4 位元加法電路的輸出 為 'S'
- 第 1 級的 4 位元加法電路的進位輸出為「ADDER0 _ CY _ OUT」
- 修正檢查電路的輸出為「BCD _ COUT」



```
module BCD_ADDER (A, B, CIN, BCD_SUM, BCD_COUT);
input  [3:0]  A, B;
input  CIN;
output [3:0]  BCD_SUM;
output BCD_COUT;
wire   [3:0]  S;
wire   ADDER0_CY_OUT, ADDER1_CY_OUT;

    ADDER4  ADDER0 (A, B, CIN, S, ADDER0_CY_OUT);
    ADDER4  ADDER1 (S, {1'b0, BCD_COUT, BCD_COUT, 1'b0}, 0, BCD_SUM,
                    ADDER1_CY_OUT);
                    *1
                    *2

    assign  BCD_COUT= (S[3] & S[2]) | (S[3] & S[1]) | ADDER0_CY_OUT;
endmodule
```

- 第 2 級的 4 位元加法的輸入「CIN」固定為 '0'。
- 第 2 級的 4 位元加法電路的輸出「COUT」從圖
- 9.26 看來是沒有接到任何地方，所以就接到中間信號「ADDER1_CY_OUT」。

用加法運算子的 Verilog HDL 描述

程式 9.24 BCD 加法電路的 Verilog HDL 描述

LST9_24.V

```
module BCD_ADDER ( A, B, CIN, BCD_SUM, BCD_COUT );
input  [3:0]  A, B;
input  CIN;
output [3:0]  BCD_SUM;
output BCD_COUT;

wire [4:0]  ADDER0_OUT;

assign ADDER0_OUT = FUNC_BCD_ADDER ( A, B, CIN );
assign BCD_SUM = ADDER0_OUT[3:0];
assign BCD_COUT = ADDER0_OUT[4];

function [4:0]  FUNC_BCD_ADDER;
input  [3:0]  A, B;
input  CIN;
integer ADDER0_S;

begin
    *1 ADDER0_S = A + B + CIN;
    if ( ADDER0_S >= 10 )
        *2 ADDER0_S = ADDER0_S + 6;

    *3 FUNC_BCD_ADDER = ADDER0_S;
end
```