

Lab 7: A 4-bit Carry Look-ahead Adder

1. 實驗目的

了解 Carry Look-ahead Adder (CLA) 電路之操作原理，使用 Verilog 語言設計電路，並應用 CAD 軟體來輔助驗證所設計的電路。

2. 實驗器材

Quartus II (CAD tools)、DE2

3. 實驗內容

請使用 Verilog HDL 描寫出：4-bit Carry Look-ahead Adder 架構圖如下圖 7.1 所示， $A[3:0]$ 、 $B[3:0]$ 及 C_0 為輸入，輸出為 $S[3:0]$ 及 C_4 。

並用波型圖驗證後，再利用 DE2 實驗板上的 FPGA 晶片 CycloneII 系列 EP2C35F672C6，將其指撥開關作為輸入，並將結果輸出於七段顯示器上加以驗證結果。

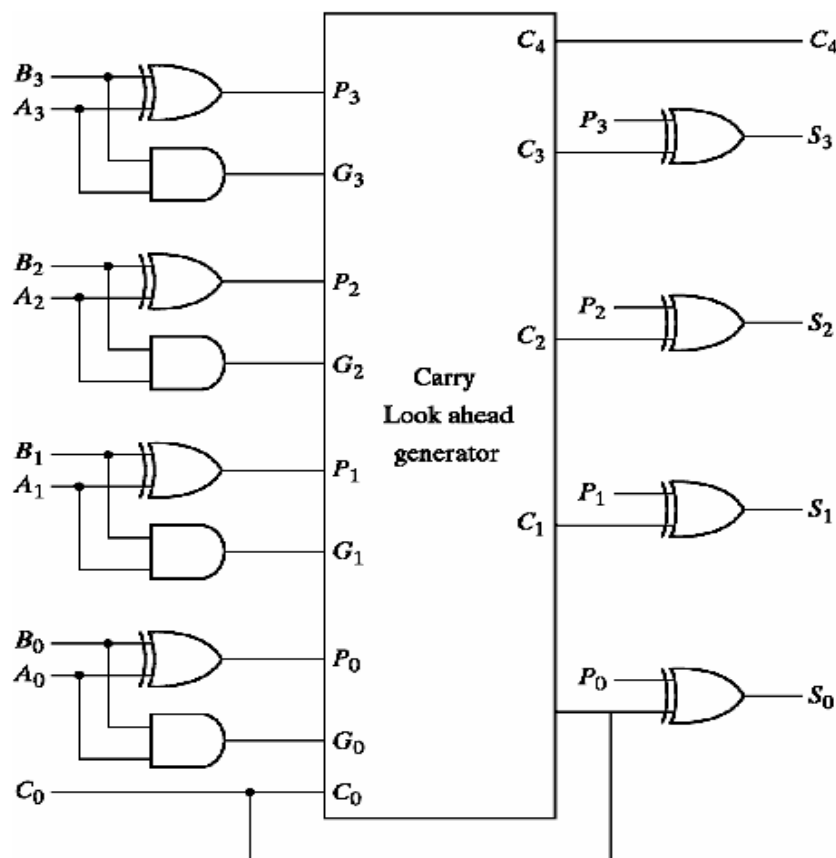


圖 7.1 4-bit Carry Look-ahead Adder

Source: Mano, "Digital Design," Prentice Hall.

4. 實驗原理

Ripple Carry Adder

圖7.2為全加法器，下面是一全加法器之程式碼：

```
module fulladder(S, Co, A, B, Ci);  
    input A, B, Ci;  
    output S, Co;  
    assign Co = { (A ^ B) & Ci } | (A & B);  
    assign S = A ^ B ^ Ci;  
endmodule
```

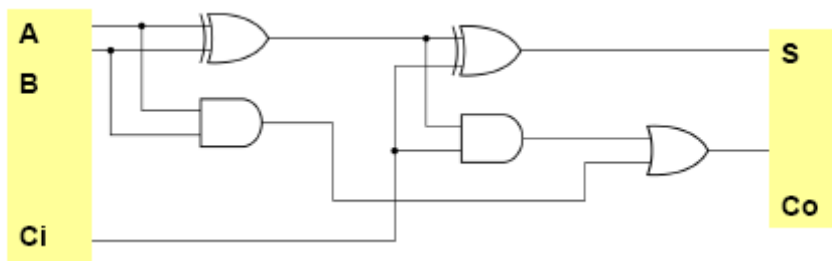


Fig. 7.2: A full adder.

使用四個全加法可構成一個四位元加法電路，如圖7.3，此型式稱為ripple-carry adder。

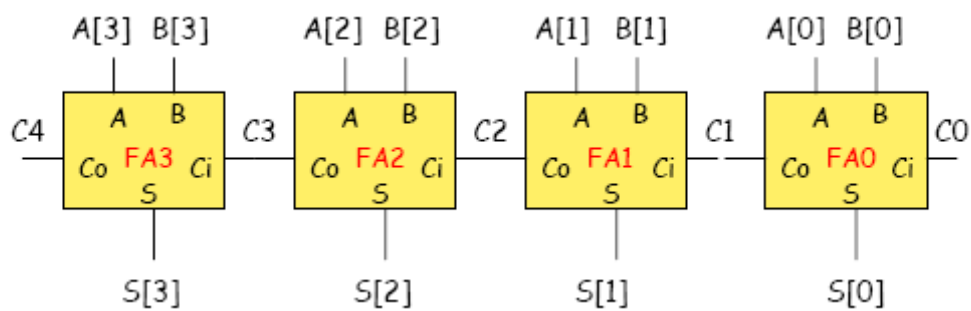


Fig. 7.3: A 4-bit ripple-carry adder.

程式碼如下：

```
module adder(S, C4, A, B, C0);
    input [3:0] A, B;
    input C0;
    output [3:0] S;
    output C4;
    wire C1, C2, C3; //Intermediate carries
    //Instantiate the fulladder
    fulladder FA0(S[0], C1, A[0], B[0], C0);
    fulladder FA1(S[1], C2, A[1], B[1], C1);
    fulladder FA2(S[2], C3, A[2], B[2], C2);
    fulladder FA3(S[3], C4, A[3], B[3], C3);
endmodule
```

4-bit Carry Look-ahead Adder

我們知道，傳統的 ripple carry adder 在做加法時須要等上一級的 carry out 產生後才能繼續下去。所以對於 16-bit adder，甚至於 n-bits adder 所造成嚴重之 delay，將不樂見。而 CLA 就是針對改善 delay 而設計出來的架構，其想法是希望將所有的進位一次運算完成。首先我們令

carry generate $G_i = A_i B_i$
carry propagate $P_i = A_i \oplus B_i$

由圖7.2 之 Full Adder 我們可知

$$\text{Sum } S = A \oplus B \oplus C_0$$

$$\text{Carry } C_1 = A B + (A \oplus B) C_0$$

將 carry out 與 sum 改寫推導如下：

$$S_i = A_i \oplus B_i \oplus C_i$$

$$= P_i \oplus C_i$$

$$C_{i+1} = A_i B_i + (A_i \oplus B_i) C_i$$

$$= G_i + P_i C_i$$

可推得：

$$C_1 = G_0 + P_0 C_0$$

$$C_2 = G_1 + P_1 C_1 = G_1 + P_1(G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0$$

由上我們可觀察出，我們只須經過一個 gate delay，就可產生所有的 G_i 和 P_i 。再經 Carry look-ahead generator 即可得到所有的 C_i ，圖7.4 為Carry look-ahead generator。這樣雖然大大的改善了我們 delay 的問題，相對的我們須付出較大的面積。

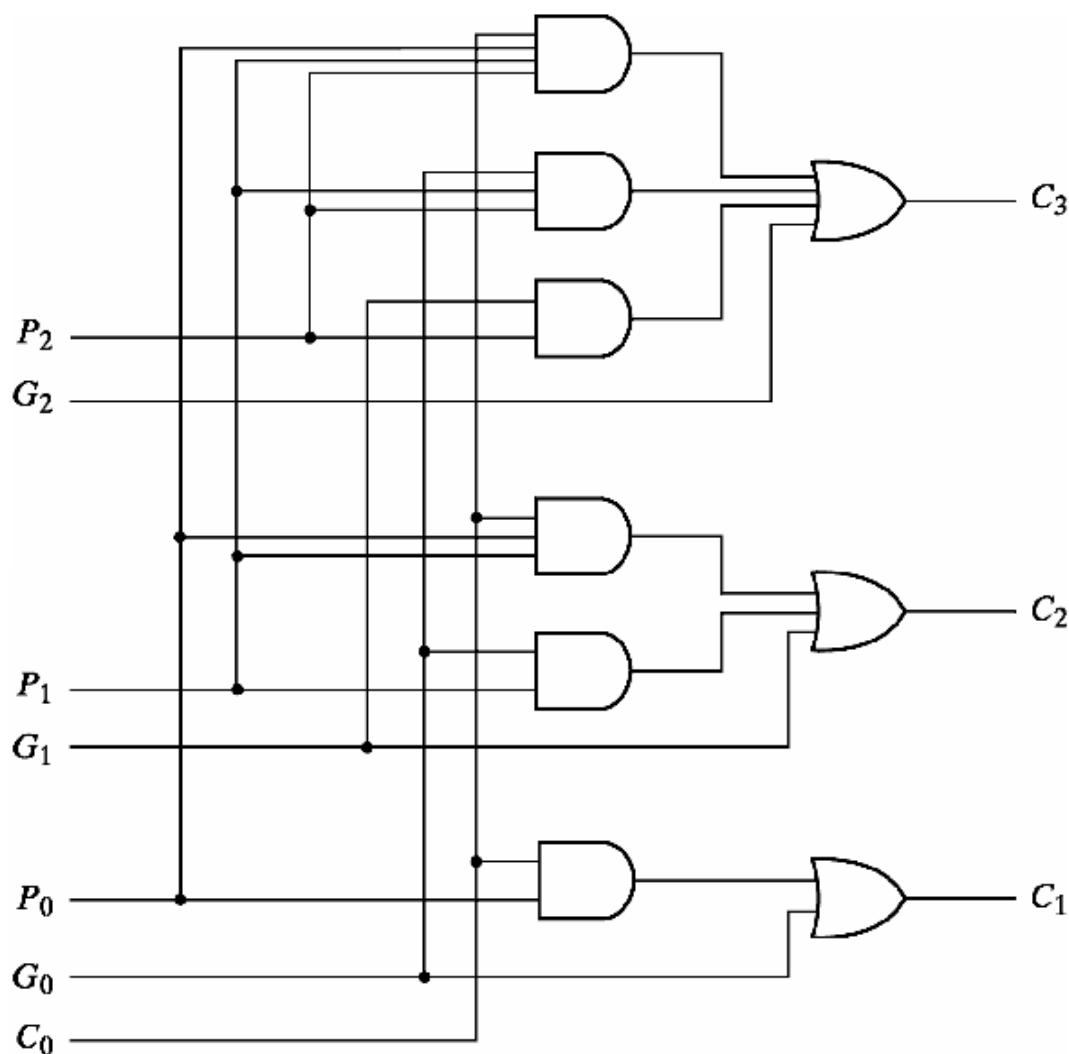


圖7.3 Carry Look-ahead Generator

Source: Mano, "Digital Design," Prentice Hall.

5. 實驗範例與結果

程式碼

//to de2

```
module test(seg7_out1,seg7_out2,A,B,c0);
```

```
input  [3:0]A,B;
```

```
input  c0;
```

```
output [6:0] seg7_out1,seg7_out2;
```

```
wire  [3:0] s;
```

```

wire    cout;
cla4    test1(s,cout,A,B,c0);
binary_to_7seg    test2(cout,seg7_out2);
binary_to_7seg    test3(s,seg7_out1);
endmodule

```

```

// behavioral description of 4-bit carry look-ahead adder
module cla4 (s, cout, A, B ,c0);
input [3:0] A;
input [3:0] B;
input c0;
output [3:0] s;
output cout;
wire [3:0] s;
wire cout;
wire [3:0] g;
wire [3:0] p;
wire [3:1] c;
assign g[3:0] = A[3:0] & B[3:0]; //carry generation
assign p[3:0] = A[3:0] ^ B[3:0]; //carry propagation
assign c[1] = g[0] | (p[0] & c0); //calculate each stage carryout
assign c[2] = g[1] | (g[0] & p[1]) | (p[0] & p[1] & c0);
assign c[3] = g[2] | (g[1] & p[2]) | (g[0] & p[1] & p[2]) | (p[0] & p[1] & p[2] & c0);
assign cout = g[3] | (g[2] & p[3]) | (g[1] & p[2] & p[3])
| (g[0] & p[1] & p[2] & p[3]) | (p[0] & p[1] & p[2] & p[3] & c0);
assign s[0] = p[0]^c0; //calculate summation
assign s[3:1] = p[3:1] ^ c[3:1];
endmodule

```

```

//DE2_7seg
module binary_to_7seg(binary_in,seg7_out);
input  [3:0] binary_in;
output [6:0] seg7_out;
reg      [6:0] seg7_out;
always @ (binary_in)
begin
case(binary_in)
4'h0: seg7_out = 7'b1000000;

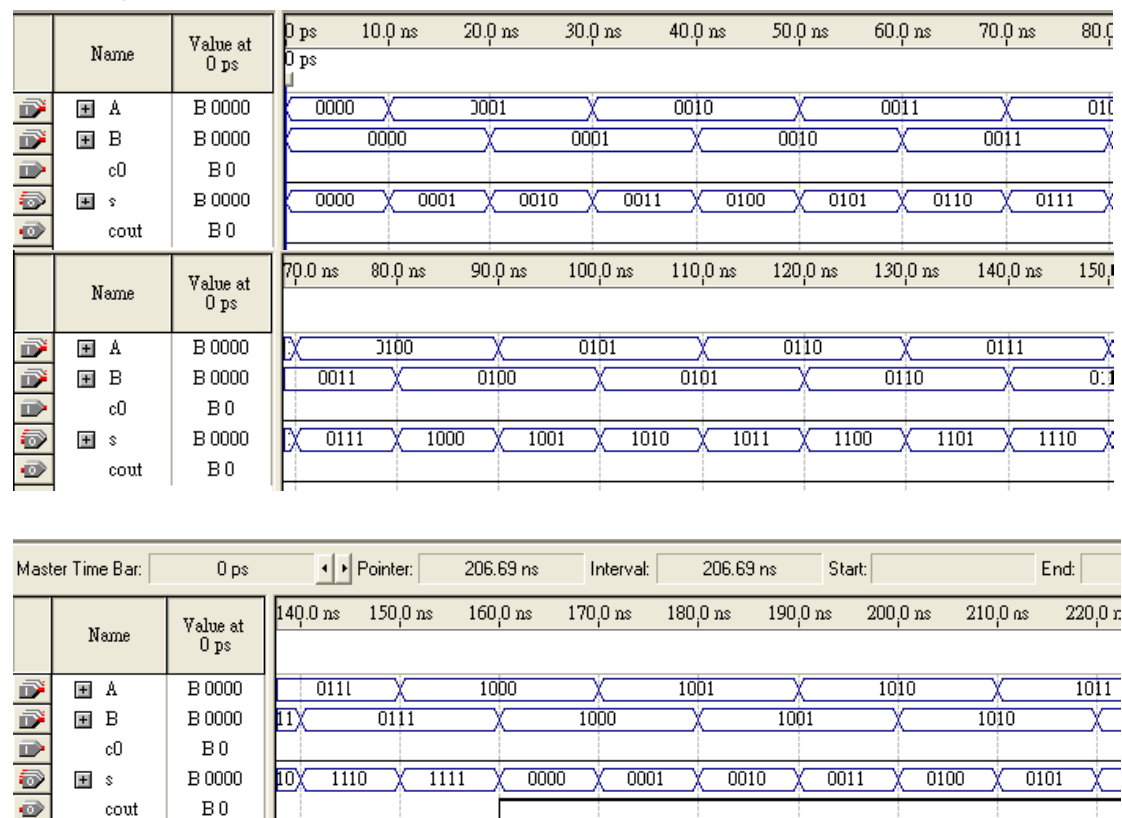
```









```









4'h1: seg7_out = 7'b1111001;
4'h2: seg7_out = 7'b0100100;
4'h3: seg7_out = 7'b0110000;
4'h4: seg7_out = 7'b0011001;
4'h5: seg7_out = 7'b0010010;
4'h6: seg7_out = 7'b0000010;
4'h7: seg7_out = 7'b1111000;
4'h8: seg7_out = 7'b0000000;
4'h9: seg7_out = 7'b0011000;
4'hA: seg7_out = 7'b0001000;
4'hB: seg7_out = 7'b0000011;
4'hC: seg7_out = 7'b1000110;
4'hD: seg7_out = 7'b0100001;
4'hE: seg7_out = 7'b0000110;
4'hF: seg7_out = 7'b0001110;
default: seg7_out = 7'b1111111;
endcase
end
endmodule

```

模擬結果：



	Name	Value at 0 ps	210,0 ns	230,0 ns	250,0 ns	270,0 ns	290,0 ns
	 A	B 0000	1011	1100	1101	1110	
	 B	B 0000	1010	1011	1100	1101	1110
	c0	B 0					
	 s	B 0000	0101	0110	0111	1000	1001
	cout	B 0	1010	1011	1100		

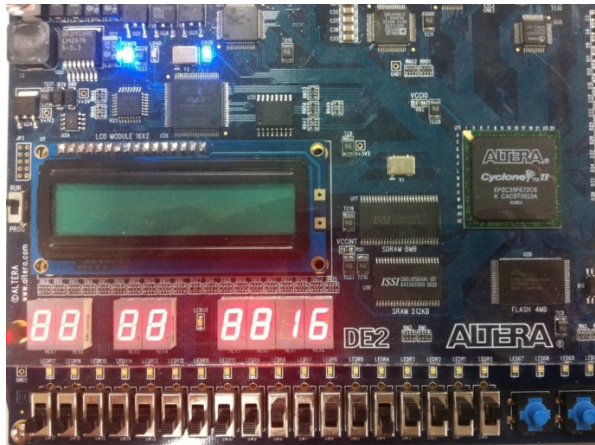
	Name	Value at 0 ps	280,0 ns	290,0 ns	300,0 ns	310,0 ns	320,0 ns	330,0 ns	340,0 ns
	 A	B 0000	1110		1111				
	 B	B 0000	1101	1110		1111			
	c0	B 0							
	 s	B 0000	1011	1100	1101	1110	1111	0001	
	cout	B 0							

輸入使用九個開關，表示兩個四位數二進制值，還有一個carry in。

輸出則是以16進制值表示兩數和carry in相加之結果。

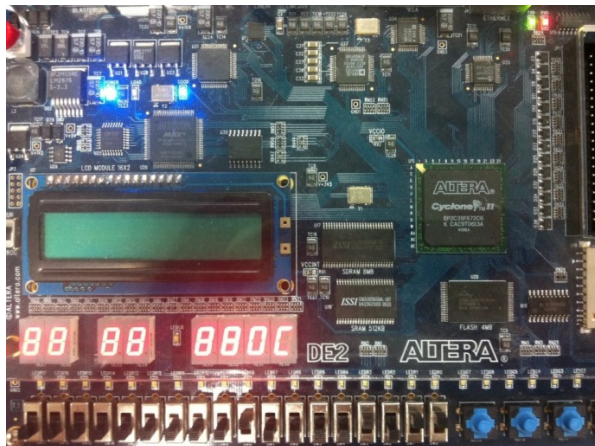
範例：開關撥1111,0110,1

$$\rightarrow 1+2+4+8+2+4+\text{carry}=(22)_{10}=(16)_{16}$$



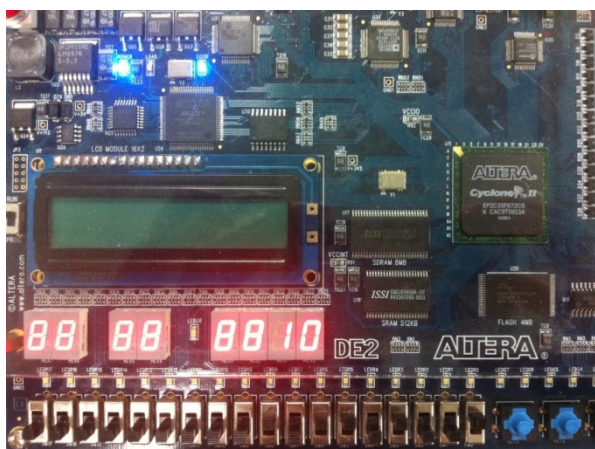
開關撥1101,0000,1

$$\rightarrow 1+2+8+\text{carry}=(12)_{10}=(C)_{16}$$



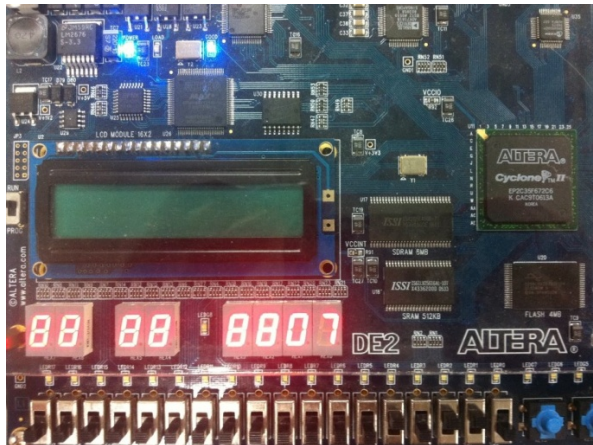
開關撥1001,0110,1

$$\rightarrow 1+8+2+4+\text{carry}=(16)_{10}=(10)_{16}$$



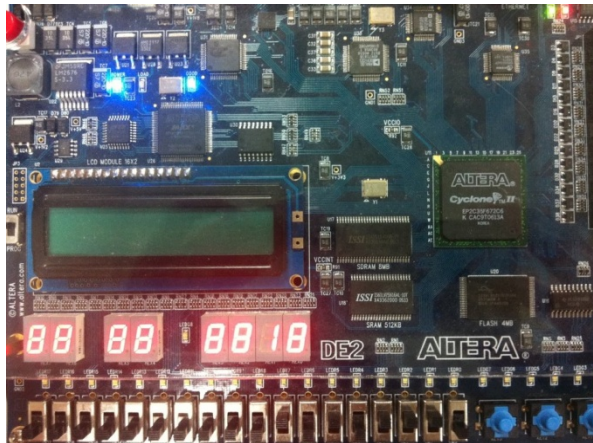
開關撥0110,1000,0

$$\rightarrow 2+4+1 = (7)_{10} = (7)_{16}$$



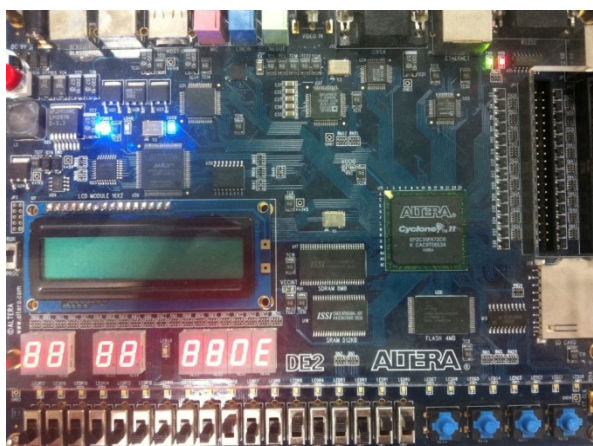
開關撥0111,0101,0

$$\rightarrow 2+4+8+2+8=(24)_{10}=(18)_{16}$$



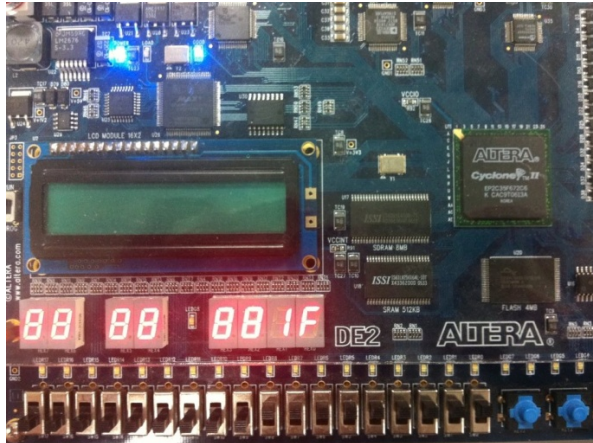
開關撥1101,0100,1

$$\rightarrow 1+2+8+2+\text{carry}=(14)_{10}=(E)_{16}$$



開關全開1111,1111,1

$\rightarrow 1+2+4+8+1+2+4+8+\text{carry} = (31)_{10} = (1F)_{16}$



6. 問題與討論

(1) 利用本實驗之4-bit CLA，設計一個4-bit Adder-subtractor，其中負數是用2's 補數表示。設輸入為A與B，另有一1-bit 之輸入M，當M=1，執行A-B；當M=0，執行A+B。(參考選設課本P.154中的Fig 4.13，將其中ripple-carry adder改成本實驗之carry look-ahead adder)