數位系統導論實驗 LAB07
Carry Lookahead Adder w/ Structural Modeling

助教 : 徐孟澤, 簡睿宇

Outline

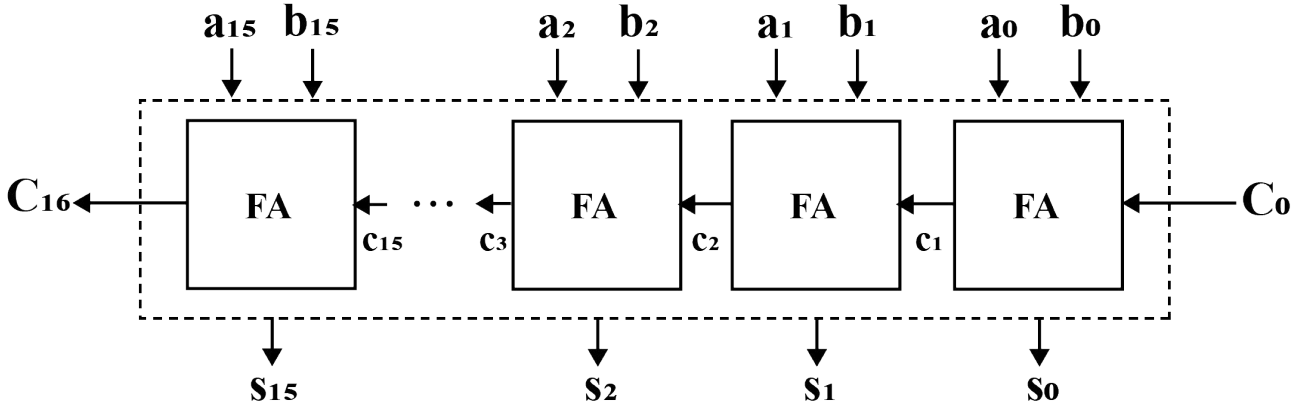
- 實驗目的
- Review : RCA
- Introduction of CLA
- Structure of CLA
  - Basic : 4-bit CLA
  - Hierarchical 16-bit CLA
- Propagation delay of CLA
- 實驗範例
- 實驗練習
- 實驗作業
- 評分方式

實驗目的

- 在 LAB6 的實驗中, 同學們已練習過利用Verilog structural modeling 來描述硬體設計, 並使用 gtkwave 以邏輯閘層次觀察波形; 而在本次的實驗中, 同學們將在Verilog 上熟悉 carry lookahead adder (CLA) 的架構, 以 4bit CLA 作為範例, 練習設計 16bit CLA, 最後能夠自行實作出 64-bit CLA

Review : RCA

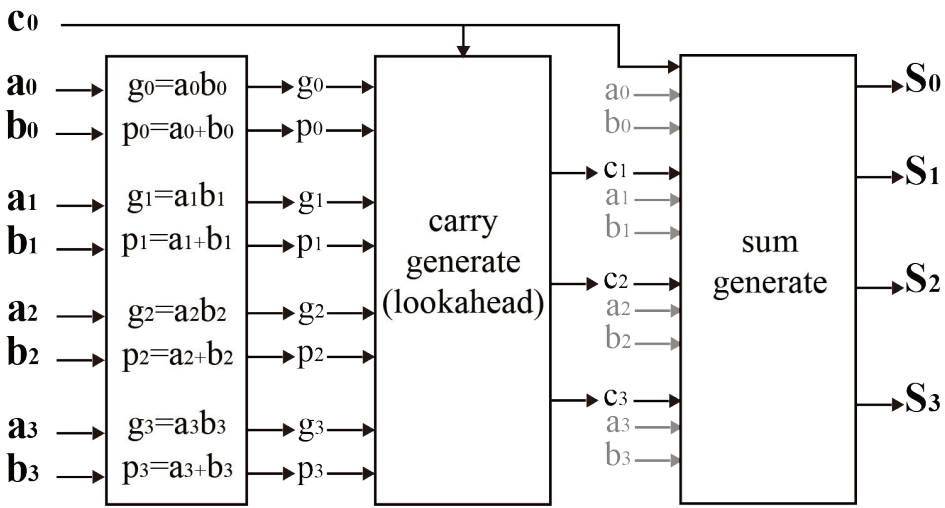
- 在課堂中以及 LAB6 中提過的 ripple carry adder (RCA) 是以多個 full adder (FA) 連接的方式組成, 其中每個 FA 都必須等待前一個加法的進位(carry) 傳入後才能開始運算, 如水波 (Ripple) 般依序傳遞 carry, 依下圖所示



16bit RCA

Introduction of CLA (1/2)

- RCA 是一種常見的加法器, 但因為每個FA 都要等待前一個 carry 傳入, 進而影響了它的速度, 因此有了 CLA 的概念, 雖然增加了電路複雜度, 但能夠同時產生所有 carry 以減少電路的延遲, 因此相較於 RCA 而言是一種高效率的加法器



4bit CLA 示意圖

Introduction of CLA (2/2)

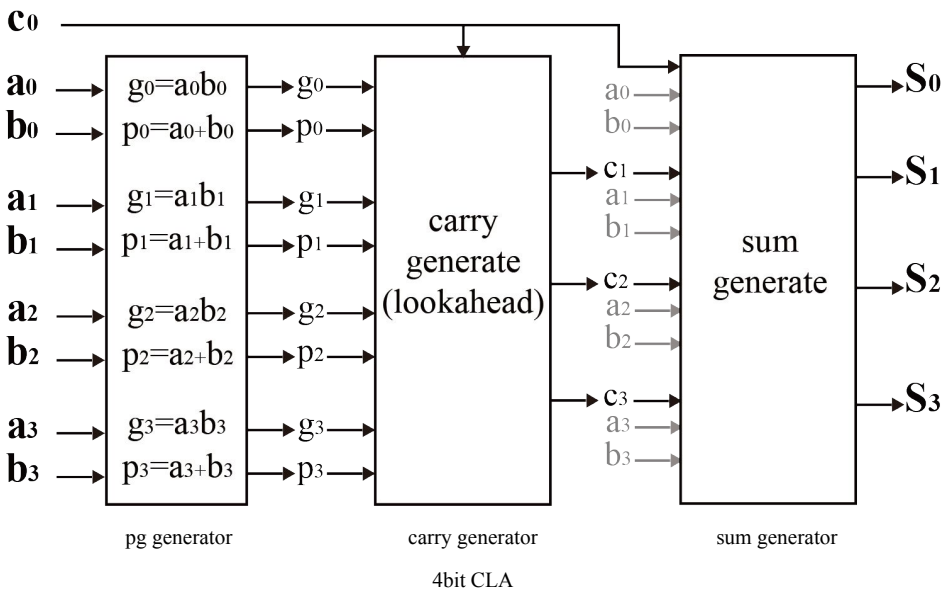
- CLA 只需要 input value 和 carry 就可以產生其他 carry, 如下列算式;但當 CLA 的邏輯閘輸入 (fan-in) 超過 4 個以上時, 就會造成大量的電路延遲, 因此fan-in 會盡量小於等於 4

$$
\begin{aligned}
c_{i+1} &= a_i b_i + b_i c_i + a_i c_i \\
&= a_i b_i + (a_i + b_i) \cdot c_i \\
&= g_i + p_i \cdot c_i \\
&= g_i + p_i \cdot (g_{i-1} + p_{i-1} \cdot c_{i-1}) \\
&= g_i + p_i \cdot g_{i-1} + p_i \cdot p_{i-1} \cdot c_{i-1} \\
&\ldots
\end{aligned}
$$

$$
\begin{aligned}
c_1 &= g_0 + p_0 \cdot c_0 \\
c_2 &= g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_0 \\
c_3 &= g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_0
\end{aligned}
$$

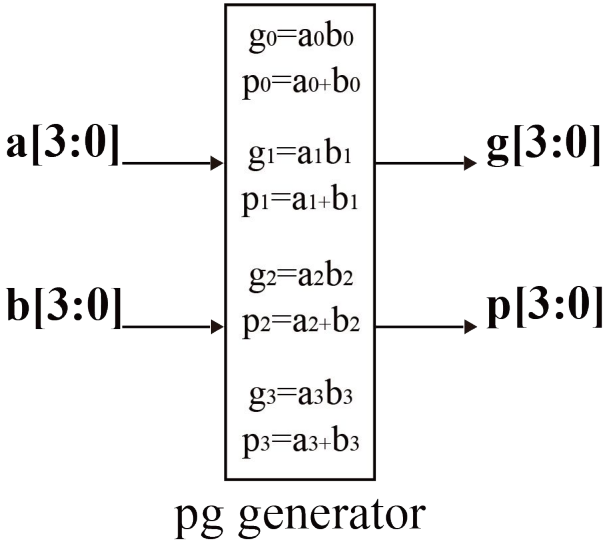
Basic : 4bit CLA (1/5)

- 4bit CLA 分別以以下三種 module 組合，透過 structural modeling，可將複數 4bit CLA module 層層組合出如樹狀般的高位元CLA，以下將依序以 Verilog 介紹 4bit CLA 的各個 module



Basic : 4bit CLA (2/5)

- 首先 input value 輸入至 gp generator, 在此 module 以 and 和 or 邏輯閘產生 Generate (g[i]) 和 Propagate (p[i])
  - Generate carry : g[i] = a[i] * b[i]
  - Propagate carry : p[i] = a[i] + b[i]

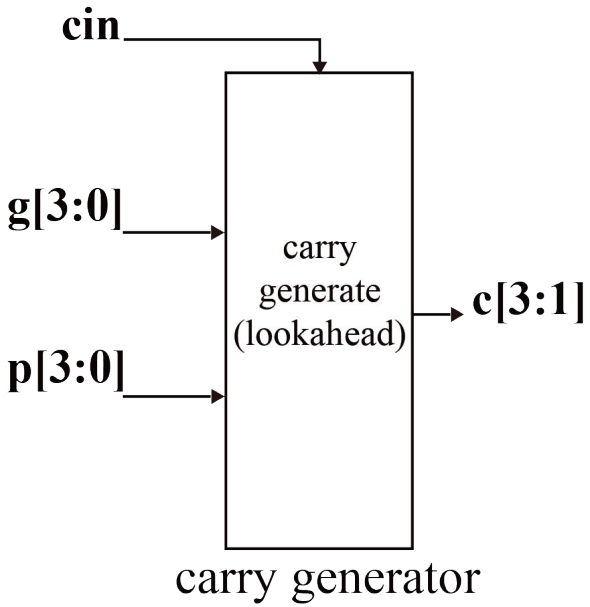



```
module gp_generator (a,b,g,p);

    input [3:0] a,b;
    output [3:0] g,p;

    assign g = a & b; // g = a x b
    assign p = a | b; // p = a + b

endmodule
```

4bit gp generator

Basic : 4bit CLA (3/5)

- gp generator 產生的 g[i] 和 p[i] 以及 carry-in 輸入至 carry generator，並在此 module 合成其他的 carry

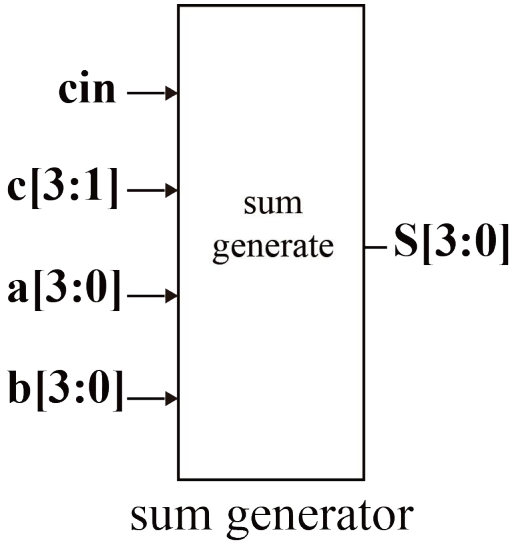



```
module carry_generator(g,p,cin,c,cout);

    input [3:0] g,p;
    input cin;
    output [3:0] c;
    output cout;

    assign c[0] = cin;
    assign c[1] = g[0] | (p[0] & cin);
    assign c[2] = g[1] | (p[1] & g[0]) | (p[1] & p[0] & cin);
    assign c[3] = g[2] | (p[2] & g[1]) | (p[2] & p[1] & g[0]) | (p[2] & p[1] & p[0] & cin);

    //cout
    assign cout = g[3] | (p[3] & g[2]) | (p[3] & p[2] & g[1]) | (p[3] & p[2] & p[1] & g[0]) |
                  (p[3] & p[2] & p[1] & p[0] & cin);

endmodule
```

4bit carry generator

Basic : 4bit CLA (4/5)

- 合成所有的 carry 後, input value 和 carry 便可以 xor 邏輯閘合成最後的加法結果




```
module sum_geneator(a,b,c,sum);

    input [3:0] a,b,c;
    output [3:0] sum;

    assign sum = a ^ b ^ c;

endmodule
```

4bit sum generator

Basic : 4bit CLA (5/5)

- 下圖是以 structural modeling 完成的 4bit CLA 行為區塊

```
module CLA_4bit(a,b,cin,sum,cout);

    input [3:0] a,b;
    input cin;
    output [3:0] sum;
    output cout;

    wire [3:0] g,p,c;

    //generate g & p
    gp_generator gp_geneator1(a[3:0],b[3:0],g[3:0],p[3:0]);

    //generate all carrys
    carry_generator carry_geneator_c0(g[3:0],p[3:0],cin,c[3:0],cout);

    //generate sum
    sum_geneator geneate_sum(a[3:0],b[3:0],c[3:0],sum[3:0]);

endmodule
```

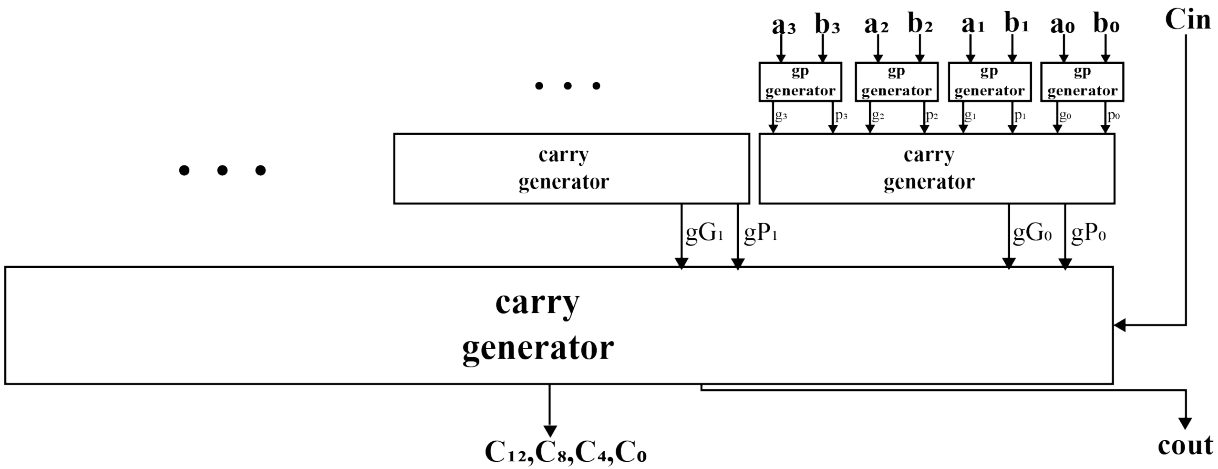
I/O definition

Variable definition

Structural modeling

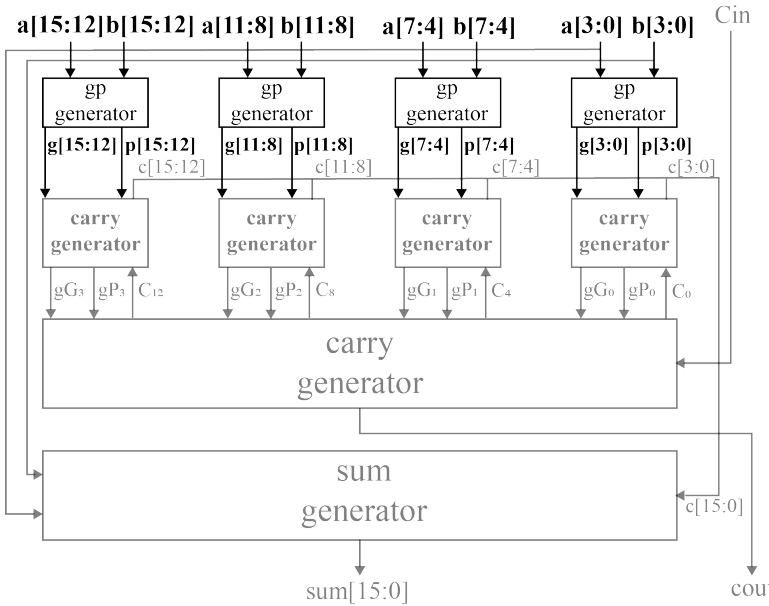
Hierarchical 16bit CLA (1/6)

- 如前頁投影片提及, 低位元的CLA 可以使用重複堆疊的方式組合出高位元CLA, 而組合出的架構如樹狀一般, 為CLA 架構的特性
- 在 Verilog 中可使用 structural modeling 將 4bit CLA 的 module 重複使用來組合成16bit CLA, 以下將以 Verilog 介紹 16bit CLA 的各 module



Hierarchical 16bit CLA (2/6)

- 在 16bit CLA 中，input value 可被拆為 4 個 4bit input value，這些被拆解的 input value 各自以 gp generator 合成 4 個 4bit g[i] 和 p[i]



gp generator

carry generator

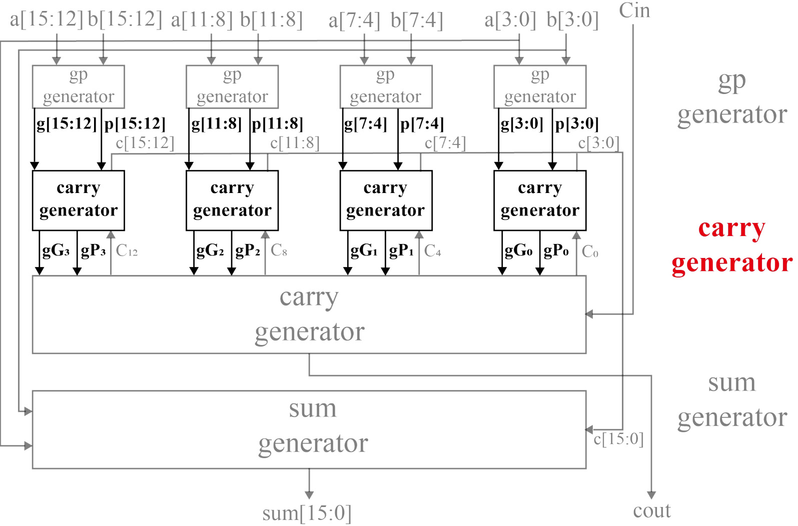
sum generator

```
//generate g & p                                    input                output
gp_generator gp_generator1(a[3:0]  ,b[3:0]  ,g[3:0]  ,p[3:0]);
gp_generator gp_generator2(a[7:4]  ,b[7:4]  ,g[7:4]  ,p[7:4]);
gp_generator gp_generator3(a[11:8] ,b[11:8] ,g[11:8] ,p[11:8]);
gp_generator gp_generator4(a[15:12],b[15:12],g[15:12],p[15:12]);
```

4x 4bit gp generator w/ structural modeling

Hierarchical 16bit CLA (3/6)

- 合成 4 個 4bit g[i] 和 p[i] 後，並非直接產生剩下的 carry，而是以 and 和 or 邏輯閘合成共 4bit 的 group of g (gG) 和 group of p (gP)，見下圖算式



$P_0 = p_0 p_1 p_2 p_3$

$G_0 = g_0 p_1 p_2 p_3 + g_1 p_2 p_3 + g_2 p_3 + g_3$

$P_1 = p_4 p_5 p_6 p_7$

$G_1 = g_4 p_5 p_6 p_7 + g_5 p_6 p_7 + g_6 p_7 + g_7$

$P_2 = p_8 p_9 p_{10} p_{11}$

$G_2 = g_8 p_9 p_{10} p_{11} + g_9 p_{10} p_{11} + g_{10} p_{11} + g_{11}$

$P_3 = p_{12} p_{13} p_{14} p_{15}$

$G_3 = g_{12} p_{13} p_{14} p_{15} + g_{13} p_{14} p_{15} + g_{14} p_{15} + g_{15}$

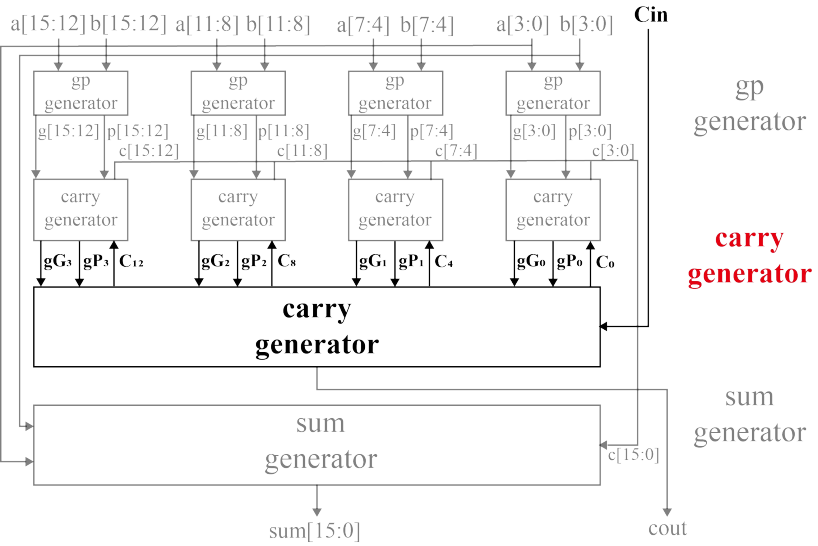
input / output

```
  //generate carrys
carry_generator carry_generator_c0(g[3:0]   ,p[3:0]   ,c4_c8_c12[0],c[3:0]   ,gG[0],gP[0],);
carry_generator carry_generator_c4(g[7:4]   ,p[7:4]   ,c4_c8_c12[1],c[7:4]   ,gG[1],gP[1],);
carry_generator carry_generator_c8(g[11:8]  ,p[11:8]  ,c4_c8_c12[2],c[11:8]  ,gG[2],gP[2],);
carry_generator carry_generator_c12(g[15:12],p[15:12],c4_c8_c12[3],c[15:12],gG[3],gP[3],);
```

4x 4bit carry generator w/ structural modeling

Hierarchical 16bit CLA (4/6)

- 合成 4bit gG 和 gP 後，它會再次輸入carry generator，以下列算式合成C0、C4、C8、C12，這 4 個 carry 可用來合成剩下的所有carry



$$c_4 = G_{3:0} + P_{3:0} \cdot c_0$$
$$c_8 = G_{7:4} + P_{7:4} \cdot G_{3:0} + P_{7:4} \cdot P_{3:0} \cdot c_0$$
$$c_{12} = G_{11:8} + P_{11:8} \cdot G_{7:4} + P_{11:8} \cdot P_{7:4} \cdot G_{3:0} + P_{11:8} \cdot P_{7:4} \cdot P_{3:0} \cdot c_0$$

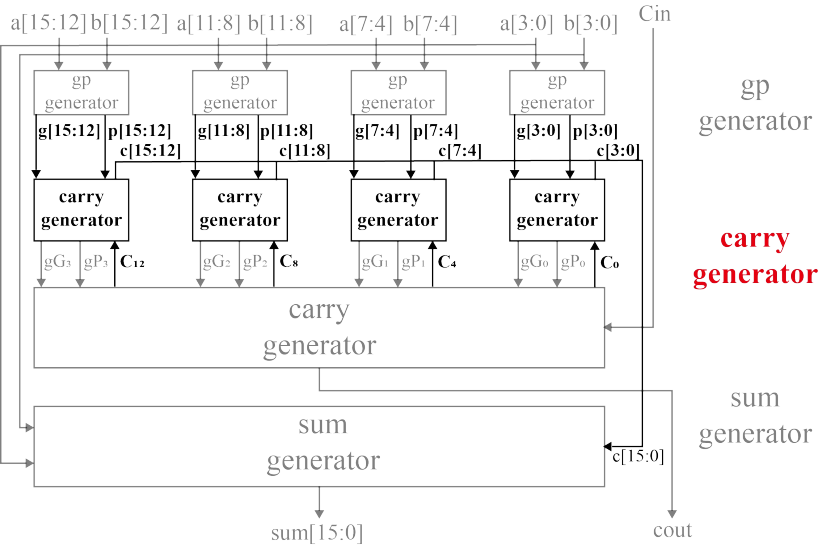
```
//generate all carrys
carry_generator carry_generator_cgs(gG[3:0],gP[3:0],cin,c4_c8_c12[3:0],,,cout);
```

input output

carry generator generate c4,c8,c12

Hierarchical 16bit CLA (5/6)

- 最後 C0、C4、C8、C12 可用來合成剩下所有的carry，並以 sum generator 合成結果




```
//generate carrys                                                                    input          output
carry generator carry generator c0(g[3:0]    ,p[3:0]   ,c4  c8  c12[0],c[3:0]   ,gG[0],gP[0],);
carry generator carry generator c4(g[7:4]    ,p[7:4]   ,c4  c8  c12[1],c[7:4]   ,gG[1],gP[1],);
carry generator carry generator c8(g[11:8]   ,p[11:8]  ,c4  c8  c12[2],c[11:8]  ,gG[2],gP[2],);
carry_generator carry_generator_c12(g[15:12],p[15:12],c4_c8_c12[3],c[15:12],gG[3],gP[3],);
```

carry generator generate all carrys

Hierarchical 16bit CLA (6/6)

- 下圖是以 structural modeling 完成的 16bit CLA 行為區塊

```
module CLA_16bit(a,b,cin,sum,cout);

    input [15:0] a,b;
    input cin;
    output [15:0] sum;
    output cout;

    wire [15:0] g,p;
    wire [15:0] c;
    wire [3:0] gG,gP;
    wire [3:0] c4_c8_c12;

    //generate g & p
    gp_generator gp_generator1(a[3:0]  ,b[3:0]  ,g[3:0]  ,p[3:0]);
    gp_generator gp_generator2(a[7:4]  ,b[7:4]  ,g[7:4]  ,p[7:4]);
    gp_generator gp_generator3(a[11:8] ,b[11:8] ,g[11:8] ,p[11:8]);
    gp_generator gp_generator4(a[15:12],b[15:12],g[15:12],p[15:12]);

    //generate carry c0,c4,c8,c12
    carry_generator carry_generator_c0(g[3:0]   ,p[3:0]  ,c4 c8 c12[0],c[3:0]  ,gG[0],gP[0],);
    carry_generator carry_generator_c4(g[7:4]   ,p[7:4]  ,c4 c8 c12[1],c[7:4]  ,gG[1],gP[1],);
    carry_generator carry_generator_c8(g[11:8]  ,p[11:8] ,c4 c8 c12[2],c[11:8] ,gG[2],gP[2],);
    carry_generator carry_generator_c12(g[15:12],p[15:12],c4_c8_c12[3],c[15:12],gG[3],gP[3],);

    //generate all carrys
    carry_generator carry_generator_(gG[3:0],gP[3:0],cin,c4_c8_c12[3:0],,,cout);

    //generate sum
    sum_geneator generate_sum(a[15:0],   b[15:0],     c[15:0],     sum[15:0]);

endmodule
```

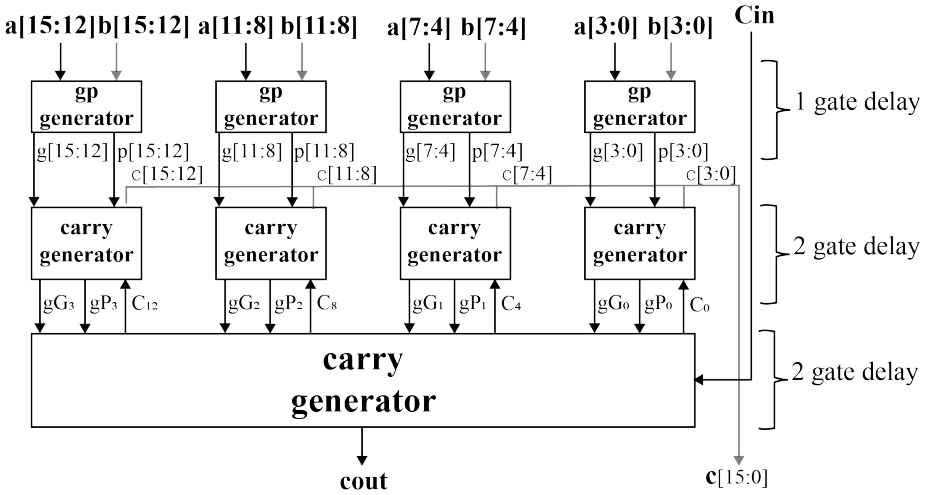
I/O definition

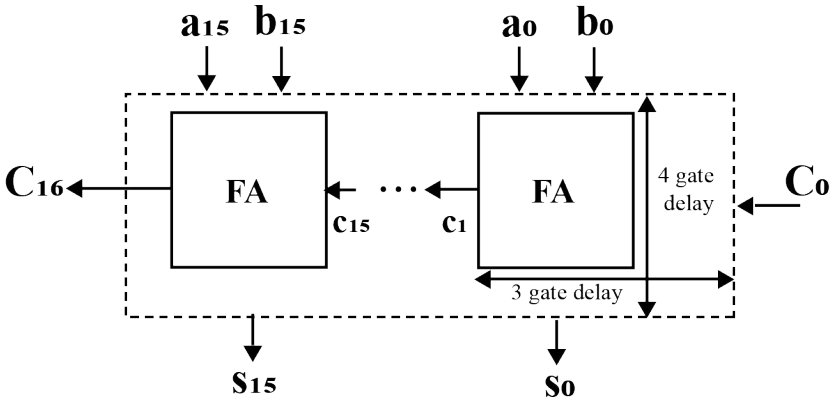
Variable definition

Structural modeling

Propagation delay of CLA

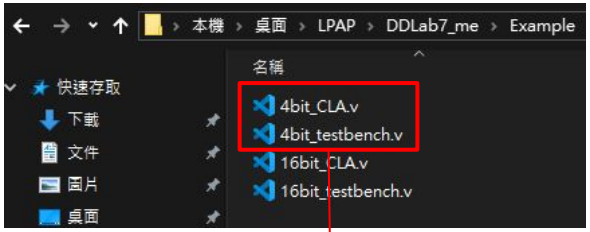
- 以在 LAB6 的 FA 連續組合的 16bit RCA，由輸入到輸出產生所有 carry 共需要消耗 3*16 個 gate delay；但若使用 16bit CLA 則只需要 1+2+2+2 個 gate delay 來產生所有 carry，故 16bit CLA 比 16bit RCA 有更好的效率





實驗範例

- 請開啟 "Example" 資料夾, 觀察 "4bit_CLA.v" 內的 structural modeling 設計, 並使用 "4bit_testbench.v" 進行驗證, 結果會顯示20 道運算
- 在 demo 時向助教展示 testbench 的驗證結果 (20%)



使用檔案

驗證結果

```
C:\Users\ECL\Desktop\LPAP\DDLab7_me\Example>iverilog -o test 4bit_CLA.v 4bit_testbench.v

C:\Users\ECL\Desktop\LPAP\DDLab7_me\Example>vvp test
VCD info: dumpfile 4bit_CLA.fsdb opened for output.
/////////Test  1/////////
//  Q : 8 + 12 + 1 = ?   //
////////////////////////////
//  Your answer           //
//  Cout = 1 Sum =  5     //
////////////////////////////
//  Correct answer        //
//  Cout = 1 Sum =  5     //
////////////////////////////
//       SUCCESSFUL !     //
////////////////////////////

/////////Test  2/////////
//  Q : 2 + 12 + 1 = ?   //
////////////////////////////
//  Your answer           //
//  Cout = 0 Sum = 15     //
////////////////////////////
//  Correct answer        //
//  Cout = 0 Sum = 15     //
////////////////////////////
//       SUCCESSFUL !     //
////////////////////////////
```

實驗練習

- 請開啟 "Example" 資料夾, 請參考實驗範例並以**structural modeling** 設計 "16bit_CLA.v", 可使用 "16bit_testbench.v" 進行驗證, 結果會顯示20 道運算
- 請在 demo 時向助教展示 testbench 驗證結果 (40%)

使用檔案

驗證結果

```
C:\Users\ECL\Desktop\LPAP\DDLab7_me\Example>iverilog -o test 16bit_CLA.v 16bit_testbench.v

C:\Users\ECL\Desktop\LPAP\DDLab7_me\Example>vvp test
VCD info: dumpfile 16bit_CLA.fsdb opened for output.
////////////Test  1////////////
//  Q :18233 +  7947 + 1 = ? //
////////////////////////////////
//  Your answer               //
//  Cout = 0 Sum = 26181      //
////////////////////////////////
//  Correct answer            //
//  Cout = 0 Sum = 26181      //
////////////////////////////////
//         SUCCESSFUL !       //
////////////////////////////////

////////////Test  2////////////
//  Q : 2132 + 33222 + 1 = ? //
////////////////////////////////
//  Your answer               //
//  Cout = 0 Sum = 35355      //
////////////////////////////////
//  Correct answer            //
//  Cout = 0 Sum = 35355      //
////////////////////////////////
//         SUCCESSFUL !       //
////////////////////////////////
```

實驗作業

- 請開啟 "Example" 資料夾, 請參範例和練習題目, 使用**structural modeling** 設計 "64bit_CLA.v" 內的所有 module, 可以 "64bit_testbench" 驗證設計正確性
- 請在 demo 時向助教展示 testbench 驗證結果 (40%)

使用檔案

驗證結果

```
C:\Users\ECL\Desktop\LPAP\LAB7正式版\DDLab7_me\Homework>iverilog -o test 64bit_CLA.v 64bit_testbench.v

C:\Users\ECL\Desktop\LPAP\LAB7正式版\DDLab7_me\Homework>vvp test
VCD info: dumpfile lab06_64bit.fsdb opened for output.
                                Test  1
//  Q :16226723179588100684 + 13176437954090906410 + 1 = ?
//  Your answer
//  Cout = 1 Sum = 10956417059969455479
//  Correct answer
//  Cout = 1 Sum = 10956417059969455479
                          SUCCESSFUL !
                                Test  2
//  Q : 22934289500283699934 + 15686080261204998696 + 0 = ?
//  Your answer
//  Cout = 0 Sum = 17979509211233368630
//  Correct answer
//  Cout = 0 Sum = 17979509211233368630
                          SUCCESSFUL !
```

評分方式

- Demo 時間與梯次 : 請以 E-Course 和社團公布為主
- Demo 地點 : 資工館 501A
- 實驗評分方式 :
  - 展示 4bit CLA 範例的 testbench 運算結果 (20%)
  - 展示 16bit CLA 練習的 testbench 的運算結果 (40%)
  - 展示 64bit CLA 作業, 以 testbench 驗證結果 (40%)
- 請記得填寫意見回饋表以及攜帶口罩入場, 否則不予計分

- 若對本次實驗有任何疑問, 請和對應組別的助教聯繫或是寄信到以下信箱詢問
  - 徐孟澤 sszrop321@gmail.com
  - 簡睿宇 ru03bjo4m385122@gmail.com