DD Lab7
Carry Lookahead Adder w/ Structural Modeling

助教：德漢、文駿、韋廷、冠良、泰翔

Outline

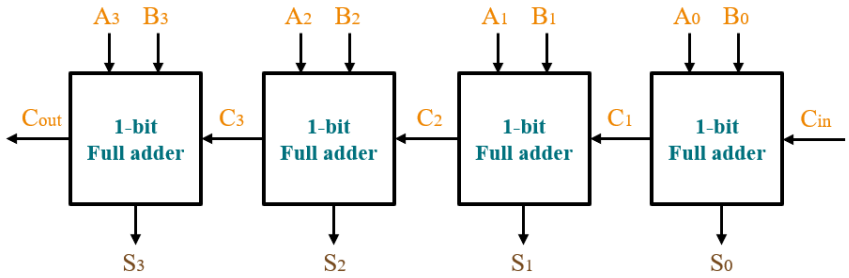
- 課程目標
- Introduction of CLA
- Structure of CLA
  - Basic : 4-bit CLA
  - Hierarchical 16-bit CLA
- 實驗範例
- 實驗練習
- 實驗作業
- Demo事項

課程目標

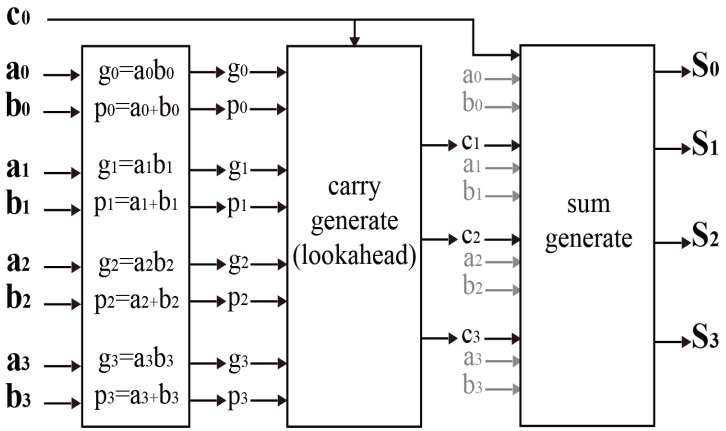
- 在本次課程中，同學們將利用 Lab 6 學到的 structural modeling 設計方式來設計 Carry Lookahead Adder (CLA)，以4-bit CLA 為範例，練習設計 16-bit CLA ，最後實作出 64-bit CLA

Introduction of CLA (1/2)

- 上個 Lab 學到的 RCA 在執行加法運算時必須等上一級的進位(carry)產生後才能繼續運算下去，這種方式會產生嚴重的延遲時間，而carry lookahead adder (CLA)就是為了減少延遲所設計出來的架構，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，是一種相對高效率的加法器。



Ripple carry adder



4-bit CLA

Introduction of CLA (2/2)

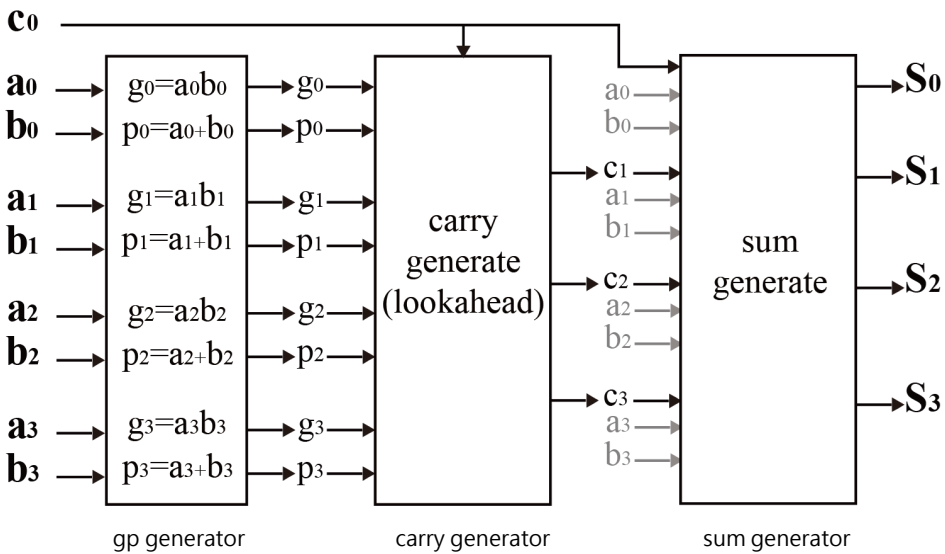
- CLA 只需要 input value 和 carry in 就可以產生其他 carry (如下列算式)，通過這樣的通式，我們就可以分別推導出 C_{i+1} 是多少；但當 CLA 的邏輯閘輸入超過 4 個時，就會造成大量的電路延遲，因此在設計上輸入會盡量小於等於 4。

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

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

Basic : 4bit CLA (1/5)

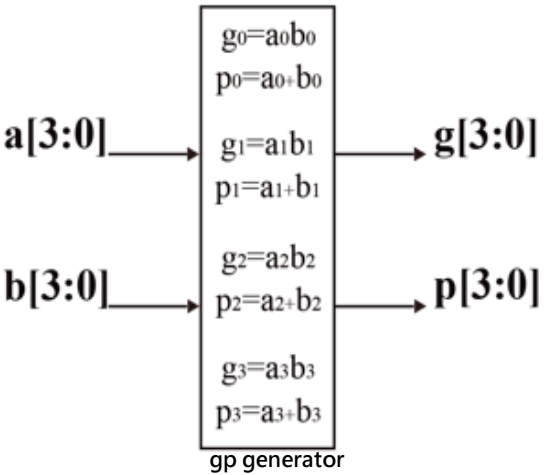
- 4bit CLA 透過 structural modeling 的方式，由三種 module 組合而成，以下將依序以 Verilog 介紹 4bit CLA 的各個 module



4bit CLA 架構圖

Basic : 4bit CLA (2/5)

- 首先 input value 輸入至 gp generator，在此 module 以 and 和 or 產生 Generate (g[i]) 和 Propagate (p[i])
  - Carry generate function : g[i] = a[i] * b[i]
  - Carry propagate function : p[i] = a[i] + b[i]

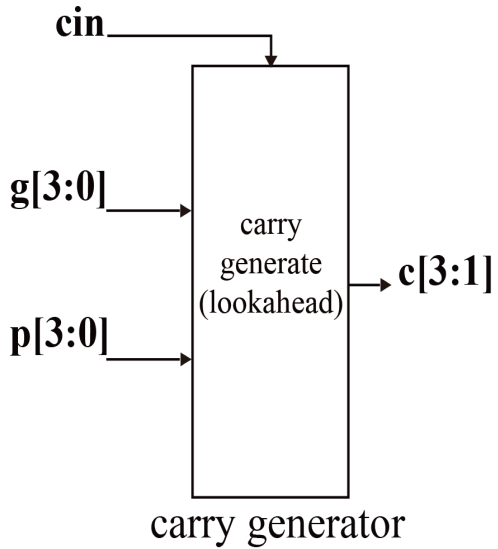



```
module gp_generator(a,b,g,p);

    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

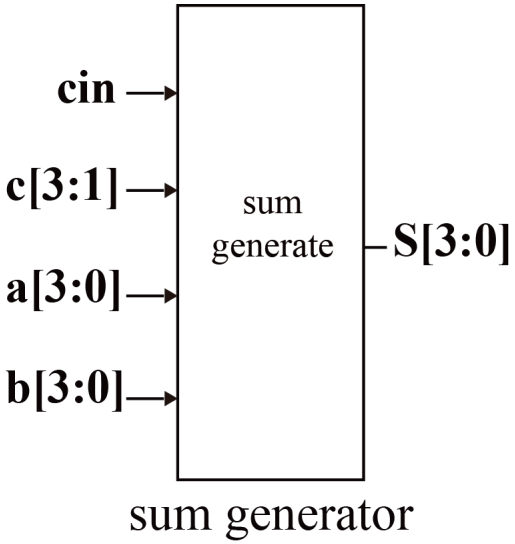



```
module carry_generator(g,p,cin,c);

    input [3:0] g,p;
    input cin;
    output [3:1] c;

    //create carries
    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);

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,cin,c,sum);

    input  [3:0] a,b;
    input cin;
    input  [3:1] c;
    output [3:0] sum;

    assign sum = a ^ b ^ {c,cin};

endmodule
```

4bit sum generator

Basic : 4bit CLA (5/5)

- 下圖是以 structural modeling 完成的 4bit CLA verilog code

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

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

    wire [3:0] g,p;
    wire [3:1] 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[3:1]);

    //generate sum
    sum_geneator geneate_sum(a[3:0],b[3:0],cin,c[3:1],sum[3:0]);

endmodule
```

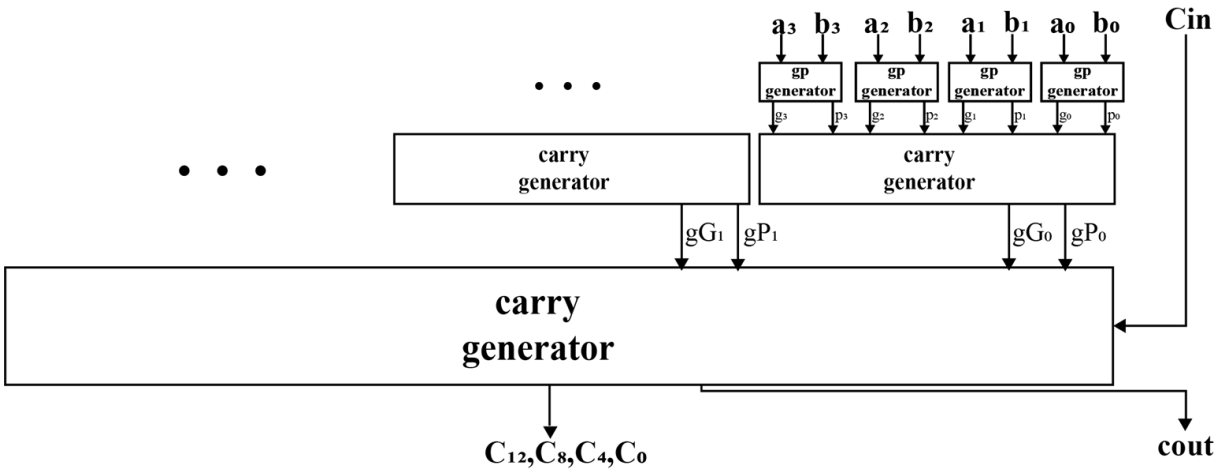
IO definition

Variable definition

Structure modeling

Hierarchical 16bit CLA (1/6)

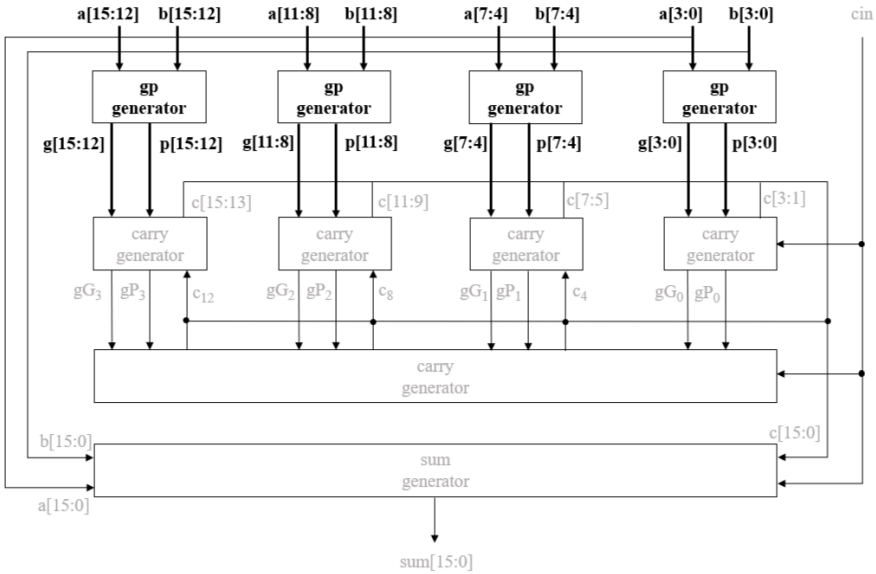
- 透過 structural modeling 的方式將 4bit CLA 內的 module 重複使用來組合成16bit CLA。



Hierarchical 16bit CLA

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]

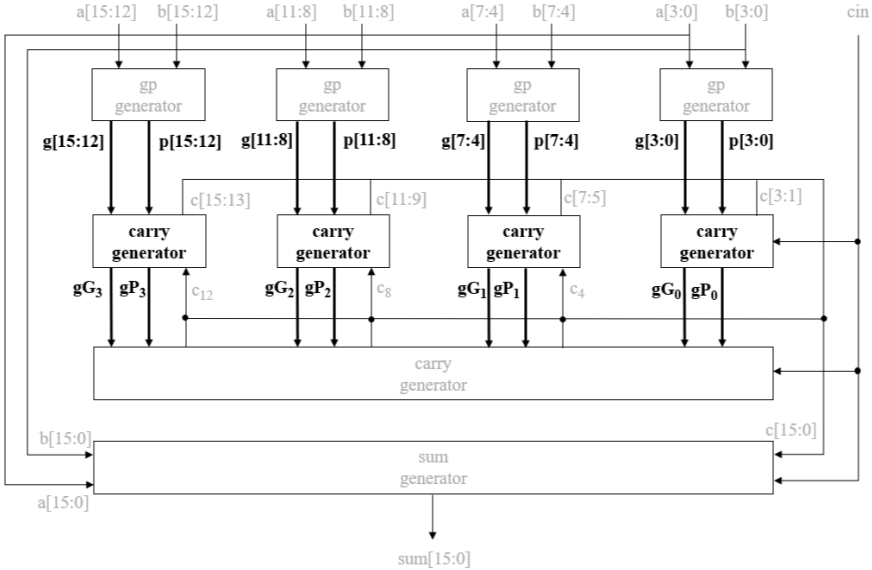



```
//generate g & p
  gp_generator gp_gen1( a[3:0],   b[3:0],   g[3:0],   p[3:0]);
  gp_generator gp_gen2( a[7:4],   b[7:4],   g[7:4],   p[7:4]);
  gp_generator gp_gen3( a[11:8],  b[11:8],  g[11:8],  p[11:8]);
  gp_generator gp_gen4( a[15:12], b[15:12], g[15:12], p[15:12]);
```

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



$$gP_0 = p_0p_1p_2p_3$$
$$gG_0 = g_0g_1g_2g_3+g_1p_2p_3+g_2p_3+g_3$$

$$gP_1 = p_4p_5p_6p_7$$
$$gG_1 = g_4g_5g_6g_7+g_5p_6p_7+g_6p_7+g_7$$

$$gP_2 = p_8p_9p_{10}p_{11}$$
$$gG_2 = g_8g_9g_{10}g_{11}+g_9p_{10}p_{11}+g_{10}p_{11}+g_{11}$$

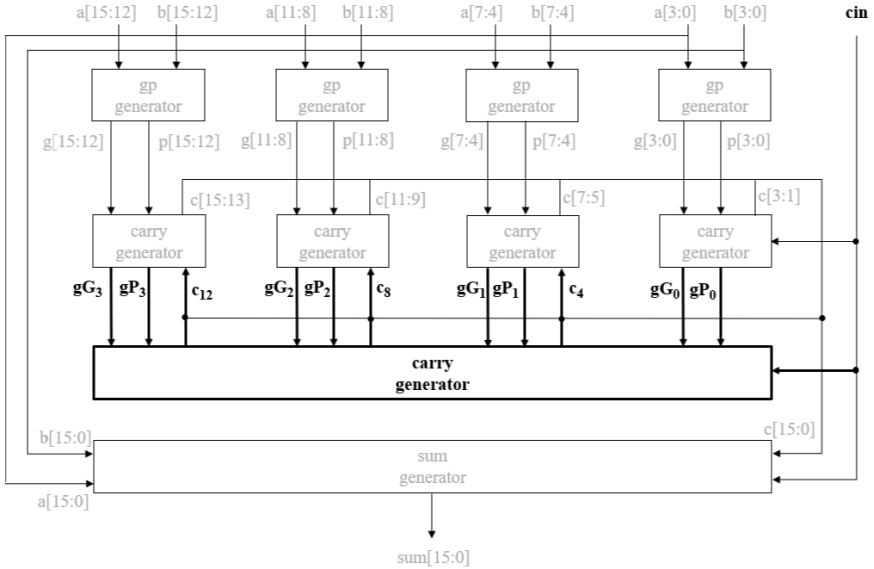
$$gP_3 = p_{12}p_{13}p_{14}p_{15}$$
$$gG_3 = g_{12}g_{13}g_{14}g_{15}+g_{13}p_{14}p_{15}+g_{14}p_{15}+g_{15}$$

```
//generate gG & gP, generate carries
  carry_generator c_gen0( g[3:0],   p[3:0],   cin,   c[3:1],   gG[0], gP[0]);
  carry_generator c_gen1( g[7:4],   p[7:4],   c[4],  c[7:5],   gG[1], gP[1]);
  carry_generator c_gen2( g[11:8],  p[11:8],  c[8],  c[11:9],  gG[2], gP[2]);
  carry_generator c_gen3( g[15:12], p[15:12], c[12], c[15:13], 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 c4、c8、c12，這 3 個 carry 加上 cin 可用來合成剩下的所有 carry



$$c_4 = gG_0 + gP_0 \cdot cin$$

$$c_8 = gG_1 + gP_1 \cdot gG_0 + gP_1 \cdot gP_0 \cdot cin$$

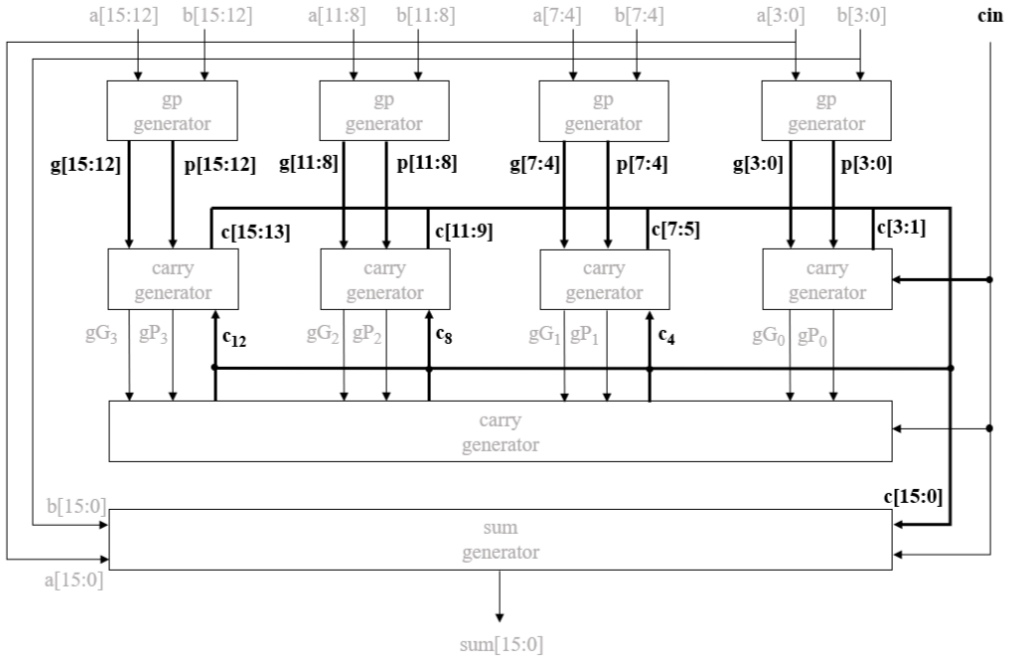
$$c_{12} = gG_2 + gP_2 \cdot gG_1 + gP_2 \cdot gP_1 \cdot gG_0 + gP_2 \cdot gP_1 \cdot gP_0 \cdot cin$$

```
//generate carries c4, c8, c12
  carry_generator c_gen4(gG[3:0], gP[3:0], cin, {c[12],c[8],c[4]}, ,);
```

carry generator generate c4,c8,c12

Hierarchical 16bit CLA (5/6)

- cin、c4、c8、c12 可用來合成剩下所有的 carry，最後用 sum generator 算出結果




```
//generate gG & gP, generate carries
  carry_generator c_gen0( g[3:0],   p[3:0],   cin,   c[3:1],   gG[0], gP[0]);
  carry_generator c_gen1( g[7:4],   p[7:4],   c[4],  c[7:5],   gG[1], gP[1]);
  carry_generator c_gen2( g[11:8],  p[11:8],  c[8],  c[11:9],  gG[2], gP[2]);
  carry_generator c_gen3( g[15:12], p[15:12], c[12], c[15:13], 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);

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

    wire [15:0] g,p;
    wire [15:1] c;
    wire [3:0] gG,gP;

    //Write down your design here ---//

      //generate g & p
        gp_generator gp_gen1( a[3:0],   b[3:0],   g[3:0],   p[3:0]);
        gp_generator gp_gen2( a[7:4],   b[7:4],   g[7:4],   p[7:4]);
        gp_generator gp_gen3( a[11:8],  b[11:8],  g[11:8],  p[11:8]);
        gp_generator gp_gen4( a[15:12], b[15:12], g[15:12], p[15:12]);

      //generate gG & gP, generate carries
        carry_generator c_gen0( g[3:0],   p[3:0],   cin,   c[3:1],   gG[0], gP[0]);
        carry_generator c_gen1( g[7:4],   p[7:4],   c[4],  c[7:5],   gG[1], gP[1]);
        carry_generator c_gen2( g[11:8],  p[11:8],  c[8],  c[11:9],  gG[2], gP[2]);
        carry_generator c_gen3( g[15:12], p[15:12], c[12], c[15:13], gG[3], gP[3]);

      //generate carries c4, c8, c12
        carry_generator c_gen4(gG[3:0], gP[3:0], cin, {c[12],c[8],c[4]}, ,);

      //generate sum
        sum_generator generate_sum(a[15:0], b[15:0], cin, c[15:1], sum[15:0]);

    //------------------------------//

endmodule
```

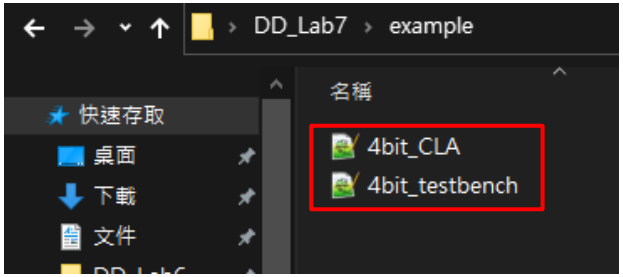
I/O definition

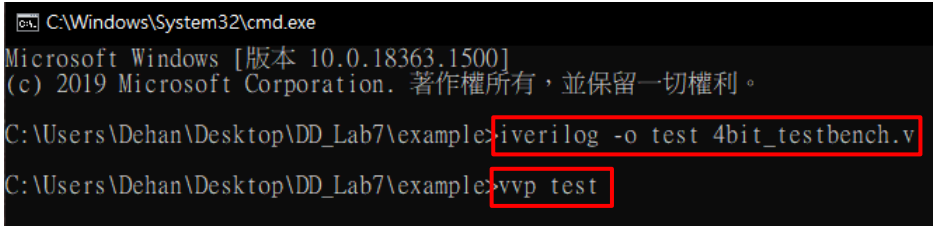
Variable definition

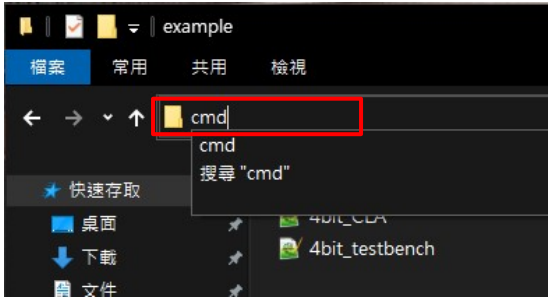
Structural modeling

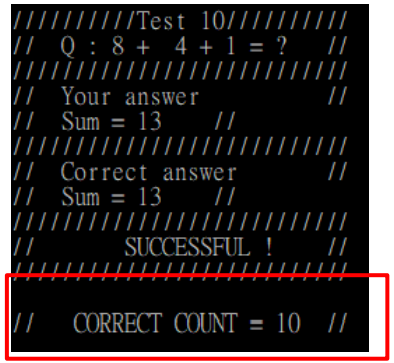
實驗範例

- 請開啟 “example” 資料夾，觀察 “4bit_CLA.v” 內的 structural modeling 設計後，進入cmd執行指令，使用 "4bit_testbench.v" 進行驗證，結果會顯示 10 道運算。



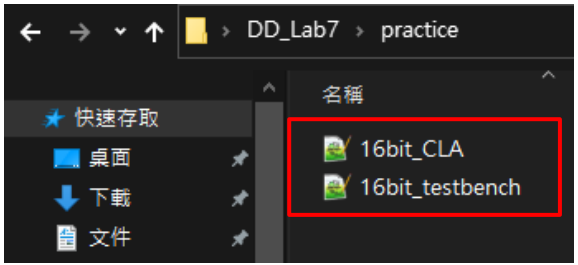


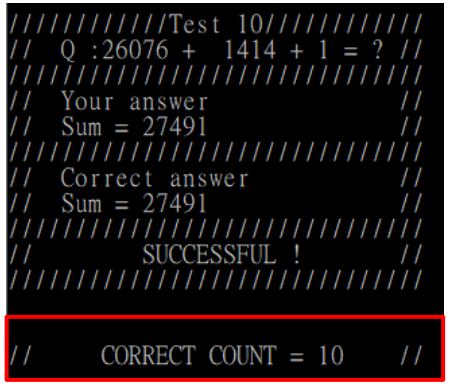




實驗練習

- 請開啟 “practice” 資料夾，觀察 “16bit_CLA.v” 內的 structural modeling 設計後，進入cmd執行指令，使用 “16bit_testbench.v" 進行驗證，結果會顯示 10 道運算。
- 請在 demo 時向助教展示 testbench 驗證結果。





實驗作業

- 請開啟 "homework" 資料夾，請參考實驗範例並以structure modeling設計 “64bit_CLA.v"，使用 “64bit_testbench.v" 進行驗證，結果會顯示 10 道運算。
- 請在 demo 時向助教展示 testbench 驗證結果。

Demo事項

- Demo 地點 : EA 501A
- Demo 時間 : 請以公布時間為主
- 評分方式 :
  - 展示 16bit CLA 練習，以 testbench 驗證結果
  - 展示 64bit CLA 作業，以 testbench 驗證結果
  - 以上兩項皆完成才計分
- 可使用自己的筆電 demo
- 可提前5分鐘入場準備，其餘時間違規進入，一次扣總成績3分
- 安排時段內無法展示請即刻離場，違者一次扣總成績5分
- 若對本次實驗有任何疑惑，請於office hour前來詢問