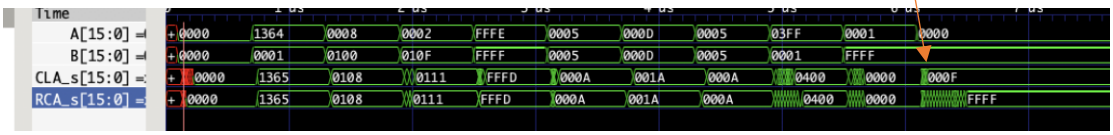
Verilog project2 b10601002 廖品捷

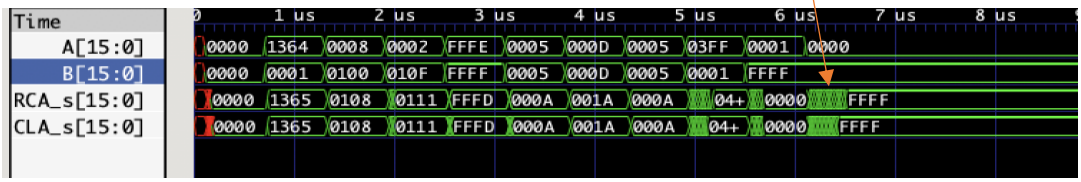
Github: https://github.com/JamesLiao714/DLD-verilog

這次做了兩組測試，一組是照著講義的 CLA unit 做，另一組是用 4 個 CLA 接起來組成一個 16 bit CLA。兩組都拿來跟 RCA 做比較。

1 用講義的 Multiple Level CLA：



2 直接用 4 個 CLA 接:



經比較後可發現，用 Multiple Level CLA 實作在 delay 上快出不少，而這也在意料之內因為用 4 個 CLA 接的話，後面的要等前面的 Cout 出來才能進行，所以會慢很多。

DISCUSSION:

(a)(b) Is CLA always faster than RCA? If not, explain your simulation results.

由上圖的結果可以發現兩種方式的 CLA 的 delay 都不一定少於 RCA，可能是它的計算量很大所以 gate delay 很多，在位數不多的情況下無法展現出其優勢，但是在某幾組測資中 CLA 完全主宰速度，而且輸 RCA 的時候 delay 不會差太多，所以 CLA 在速度上還是比較有保障

上網做了一些查詢，發現當 bit 超過 16 位的加法 CLA 能完全展現出對 RCA 的優勢

of the *k*-bit CLA block. For N>16, the carry-lookahead adder is generally much faster than the ripple-carry adder. However, the adder delay still increases linearly with *N*.

(c) If the circuit delay is not a fixed value, which one should we care about?

RCA 最重要的就是 fulladder 的速度，如果想讓 RCA 加速勢必得讓每個 fulladder 執行的速度越快越好。CLA 最重要的則是 carryin 產生的速度，所以在實作時我們要將產生 carryin 的 CLA-unit 加速得越快越好。

(d) Judge the two adders in terms of cost and performance：

兩種 adder 就 performance 來說，CLA 在理論上和實作上的速度表現都比較穩定

尤其是位數越多時，雖然蠻訝異有些測資是 RCA 較快，或許是我在設計 gate 時沒到最佳化。但成本方面也是 CLA 大很多因為需要用到很多 gate 來做運算，想讓 CLA 變快也要從硬體升級。

Source code:

Test bench:

```
//RCA & CLA
`timescale 1 ns/100ps
//-TestBench-
module Adder_TB ();
  reg [15:0]A;
  reg [15:0]B;
  wire [15:0]CLA_s;
  wire [15:0]RCA_s;
  wire Cout1;
  wire Cout2;
  reg Cin;

  CLA_16bit CLA(A,B,Cin,CLA_s,Cout2);
  RCA_16bit RCA(A,B,Cin,RCA_s,Cout1);


  initial
    begin
      #100
      A = 16'b0; B = 16'b0; Cin = 1'b0;
     #600 A = 16'b0001001101100100; B = 16'b0000000000000001; Cin = 1'b0;
      #600 A = 16'b0000000000001000; B = 16'b0000000100000000; Cin = 1'b0;
      #600 A = 16'b0000000000000010; B = 16'b0000000100001111; Cin = 1'b0; //2+16
      #600 A = 16'b1111111111111110; B = 16'b1111111111111111; Cin = 1'b0;
      #600 A = 16'b0000000000000101; B = 16'b0000000000000101; Cin =1'b0;
      #600 A = 16'b0000000000001101; B = 16'b0000000000001101; Cin =1'b0;
      #600 A = 16'b0000000000000101; B = 16'b0000000000000101; Cin =1'b0;
      #600 A = 16'b0000001111111111; B = 16'b0000000000000001; Cin =1'b0;
      #600 A = 16'b0000000000000001; B = 16'b1111111111111111; Cin =1'b0;
      #600 A = 16'b0000000000000000; B = 16'b1111111111111111; Cin =1'b0;
    end
initial #9000 $finish;
initial $dumpvars;
endmodule
```

Full adder:

```
//Full Adder
module FullAdder(S, Co, x, y, Ci);
  input x , y , Ci;
  output S, Co;
  wire s1 , d1 , d2;

  xor #(20) g1(s1, x, y);
  and #(10) g2(d1, x, y);
  and #(10) g3(d2, Ci, s1);
  xor #(20) g4(S, Ci, s1);
  or #(15) g5(Co, d1, d2);
endmodule
```

4 bit CLA:

```
  input [3:0]A;
  input [3:0]B;
  input Cin;
  output Cout;
  output[3:0]S;
  wire [3:1]C;
  wire [0:3]P;
  wire [0:3]G;

  //G
  and #(10) g0(G[0] , A[0] , B[0]);
  and #(10) g1(G[1] , A[1] , B[1]);
  and #(10) g2(G[2] , A[2] , B[2]);
  and #(10) g3(G[3] , A[3] , B[3]);

  //P
  xor #(20) p0(P[0] , A[0] , B[0]);
  xor #(20) p1(P[1] , A[1] , B[1]);
  xor #(20) p2(P[2] , A[2] , B[2]);
  xor #(20) p3(P[3] , A[3] , B[3]);

  //C1
  wire tmp1;
  and #(10) c11(tmp1 , P[0] , Cin);
  or #(15) c12(C[1] , G[0] , tmp1);
  //C2
  wire tmp2;
  wire tmp3;
  and #(10) c21(tmp2 , P[1] , G[0]);
  and #(10) c22(tmp3 , P[1] , tmp1);
  or #(20) c23(C[2] , tmp2 , tmp3 , G[1]);
  //C3
  wire tmp4;
  wire tmp5;
  wire tmp6;
  and #(10) c31(tmp4 , P[2] , G[1]);
  and #(10) c32(tmp5 , P[2] , tmp2);
  and #(20) c33(tmp6 , P[2] , tmp3);
  or #(25) c34(C[3] , tmp4 , tmp5 , tmp6 , G[2]);
  //Cout
  wire tmp7;
  wire tmp8;
  wire tmp9;
  wire tmp10;
  and #(10) c41(tmp7 , P[3] , G[2]);
  and #(10) c42(tmp8 , P[3] , tmp4);
  and #(10) c43(tmp9 , P[3] , tmp5);
  and #(10) c44(tmp10 , P[3] , tmp6);
  or #(30) c45(Cout , tmp7 , tmp8 , tmp9 , tmp10 , G[3]);

  /// Making Sums
  xor #(20) s0(S[0] , P[0] , Cin);
  xor #(20) s1(S[1] , P[1] , C[1]);
  xor #(20) s2(S[2] , P[2] , C[2]);
  xor #(20) s3(S[3] , P[3] , C[3]);
endmodule
```

16 bit RCA using full adder:

```
//RCA

module RCA_16bit(A,B,Cin,S,Cout);
  input [15:0]A;
  input [15:0]B;
  input Cin;
  output [15:0]S;
  output Cout;
  wire [15:1]carry;


  FullAdder f0(S[0] , carry[1] , A[0] , B[0] , Cin);
  FullAdder f1(S[1] , carry[2] , A[1] , B[1] , carry[1]);
  FullAdder f2(S[2] , carry[3] , A[2] , B[2] , carry[2]);
  FullAdder f3(S[3] , carry[4] , A[3] , B[3] , carry[3]);
  FullAdder f4(S[4] , carry[5] , A[4] , B[4] , carry[4]);
  FullAdder f5(S[5] , carry[6] , A[5] , B[5] , carry[5]);
  FullAdder f6(S[6] , carry[7] , A[6] , B[6] , carry[6]);
  FullAdder f7(S[7] , carry[8] , A[7] , B[7] , carry[7]);
  FullAdder f8(S[8] , carry[9] , A[8] , B[8] , carry[8]);
  FullAdder f9(S[9] , carry[10] , A[9] , B[9] , carry[9]);
  FullAdder f10(S[10] , carry[11] , A[10] , B[10] , carry[10]);
  FullAdder f11(S[11] , carry[12] , A[11] , B[11] , carry[11]);
  FullAdder f12(S[12] , carry[13] , A[12] , B[12] , carry[12]);
  FullAdder f13(S[13] , carry[14] , A[13] , B[13] , carry[13]);
  FullAdder f14(S[14] , carry[15] , A[14] , B[14] , carry[14]);
  FullAdder f15(S[15] , Cout , A[15] , B[15] , carry[15]);

endmodule
```

16 bit CLA:

version1:

```
// CLA

module CLA_16bit(A,B,Cin,S,Cout);
  input [15:0]A;
  input [15:0]B;
  input Cin;
  output [15:0]S;
  output Cout;
  wire [3:1]C;

  CLA_4bit F0_3 (S[3:0] , C[1] , A[3:0] , B[3:0] , Cin);
  CLA_4bit F4_7 (S[7:4] , C[2] , A[7:4] , B[7:4] , C[1]);
  CLA_4bit F8_11 (S[11:8] , C[3] , A[11:8] , B[11:8] , C[2]);
  CLA_4bit F12_15 (S[15:12] , Cout , A[15:12] , B[15:12] , C[3]);

endmodule
```

version2(Multiple Level CLA) #using CLA 4-bit with P and G version

```
// CLA

module CLA_16bit(A,B,Cin,S,Cout);
  input [15:0]A;
  input [15:0]B;
  input Cin;
  output [15:0]S;
  output [3:0]PP, GG;
  output Cout;
  wire [3:1]C;
  wire [3:0]temp;
  wire temp1, temp2;

  CLA_4bit oh(temp[3:0] , temp1, temp2, PP[3:0] , GG[3:0] , Cin);
  CLA_4bit F0_3 (S[3:0] , PP[0], GG[0], A[3:0] , B[3:0] , Cin);
  CLA_4bit F4_7 (S[7:4] , PP[1], GG[1], A[7:4] , B[7:4] , temp[0]);
  CLA_4bit F8_11 (S[11:8], PP[2], GG[2], A[11:8] , B[11:8] , temp[1]);
  CLA_4bit F12_15 (S[15:12] ,PP[3], GG[3], A[15:12] , B[15:12] , temp[2]);

endmodule
```