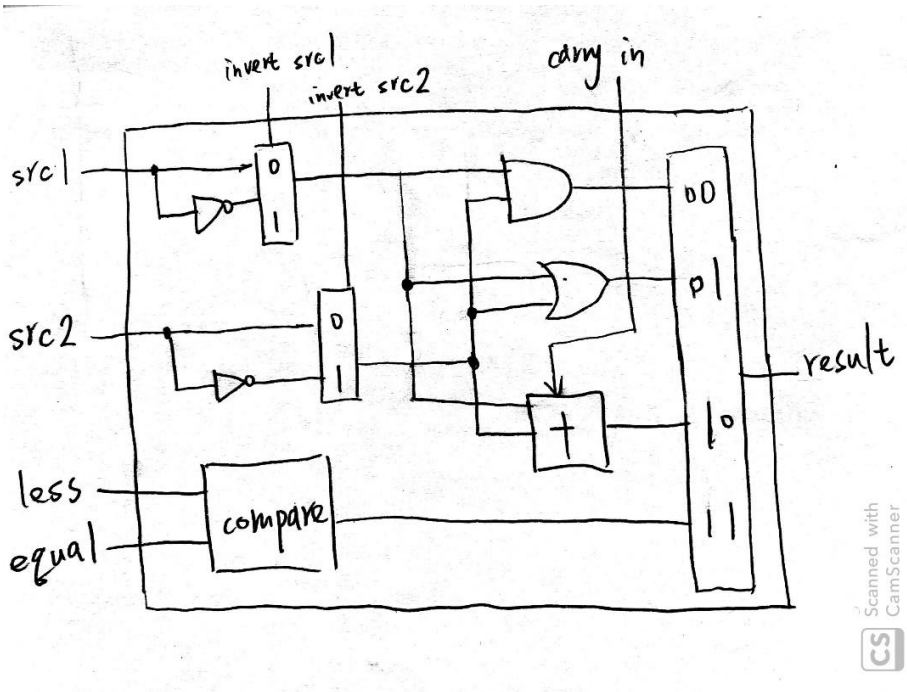
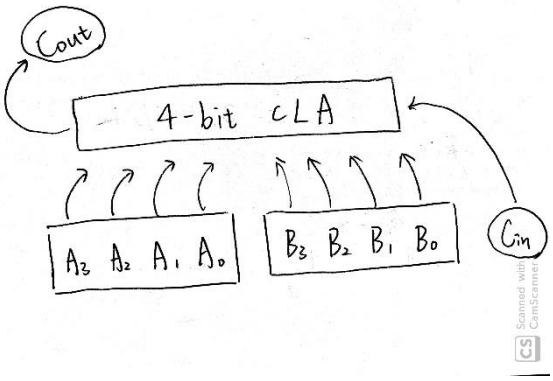
0716221 余忠旻

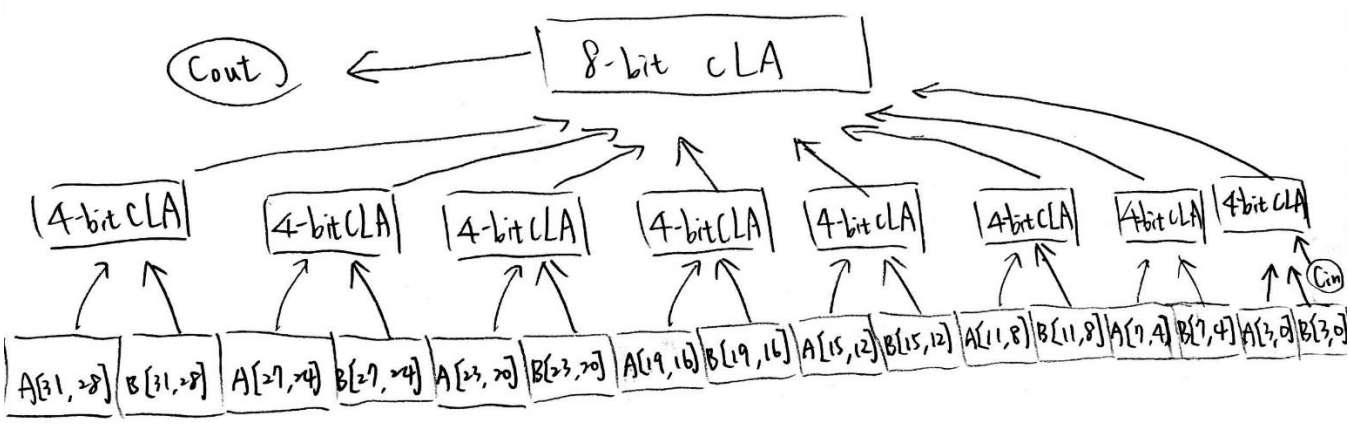
(1) Your architecture diagram



ALU_top



4-bit CLA



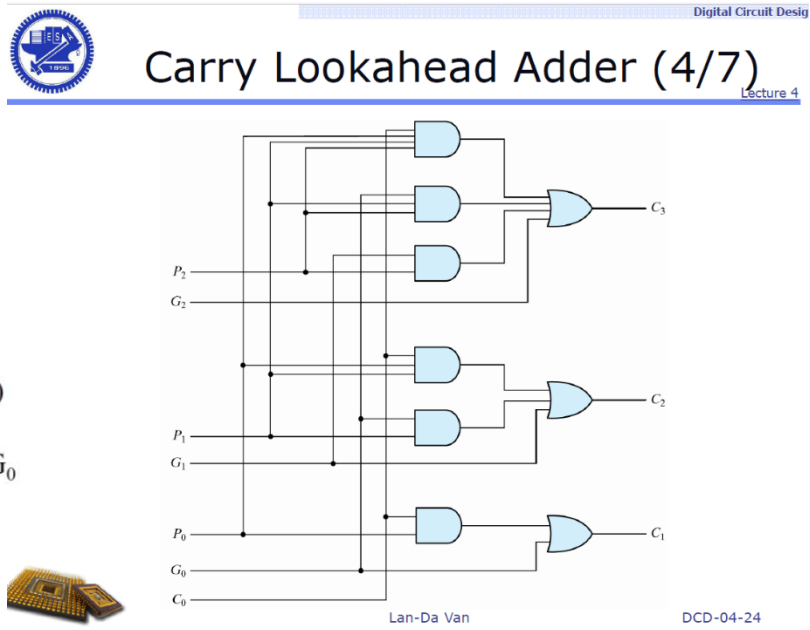
32-bit ALU拆解架構

(2) Detailed description of the implementation

一個32-bit的ALU可以分成32個1-bit的ALU做出來，

但也可以用8個4-bit的ALU或4個8-bit的ALU做出來，

而我用8個4-bit的ALU carry lookahead adder優化，如下圖，



可以用CLA公式優化，然後按architecture diagram操作，

先做出4-bit的CLA，再用8個4-bit的CLA做出32-bit的CLA

然後ALU_control的部分，可以看到ALU_control的後兩位

是決定運算方式，也就是AND，OR，ADD，LESS

而前兩位是決定src是否要invert，

例如ADD的control是0010，SUB的control是0110，

前兩位01就是A不變把B invert，等同於A +(~B) -> A - B

NOR和AND，OR和NAND同理，

而less則是看sign bit，

當進行SUB出現src1-src2大於零的狀況，會把 1 input 到 less 中，

相反小於零則是把 0 input 到 less 中，

而Zero，Cout，Overflow

觀察output，carry out，sign bit 來計算ZCV的值

最後是bonus的部分，除了輸入ALU_control 0111之外，
還會輸入bonus control 3位，這是需要做一個compare的module
來執行是要哪一種比較方式，然後觀察sign bit和 equal，
並和bonus control 做整理，最後把值 input 到 less 中

(3) Commands for executing your source codes

Source code 分為 basic 和 bonus的部分
當在執行basic的時候，
請把alu.v 和testbench.v 的‘define BONUS 這行給註解掉
因為在alu.v 執行basic的時候不會有bonus_control
所以在操作ALU_control 0111時，自動選為LST的operation處理
在alu.v 執行bonus的時候會有bonus_control
所以在操作ALU_control 0111時，還要判讀bonus_control的值，
這時會用compare 的module in 1-bit，選擇用哪個operation處理