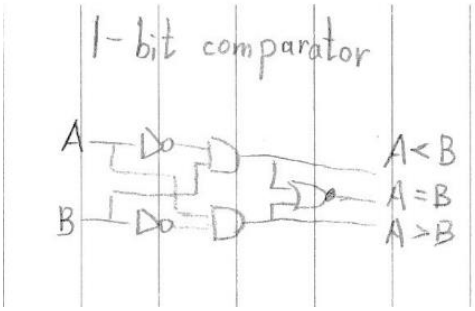
Contribution:
104062261 1/2
103062162 1/2

Q1

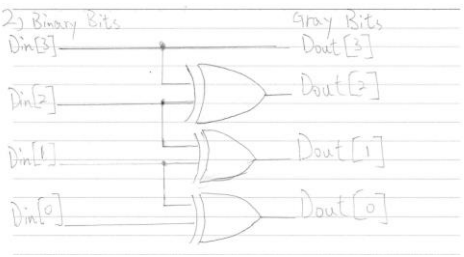
我們首先由 1-bit comparator 去理解，如果 NOT A and B 的結果為 1 的時候，則代表 B=1、A=0，所以 A<B 以及 A_lt_B =1。而 NOT B and A 的結果為 1 的時候，則代表 B=0、A=1，所以 A>B 以及 A_gt_B =1。但假若以上兩個情況的結果均為 0 時，而這種情況我們可以用 NOR Gate 來判斷，表示 A=0、B=0 或 A=1、B=1，所以 A=B 以及 A_eq_B=1。

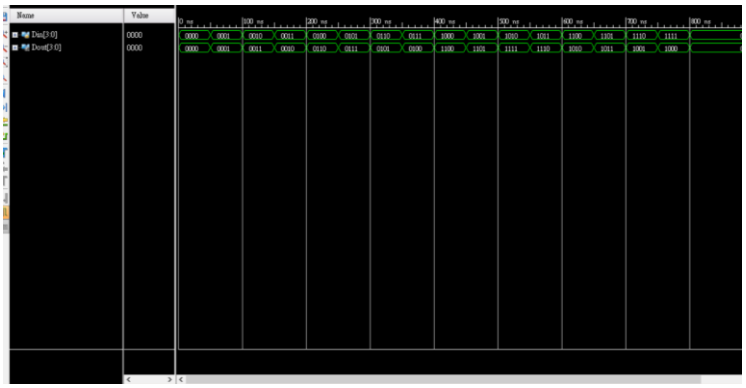


而我們可以把這個 1-bit comparator 套用到 4-bit comparator 中：假若 A3<B3，則 A<B，而 A3>B3，則 A>B，但如果 A3=B3 的話，我們就需要再判斷 A2 與 B2 的大小，假若 A2<B2 且 A3=B3，則 A<B，而 A2>B2 且 A3=B3，則 A>B，如果 A3=B3 且 A2=B2 的話，則再來判斷 A1 與 B1 的大小，如果 A1=B1 且 A2=B2 且 A3=B3 的話，則 A=B，如此類推。

Q2

第二題是要把 binary code 換成 grey code，我們發現它們的 MSB 並不會因為互換而有所改變，所以 Dout[3] = Din[3]。然後，下一個位元的新值會跟它本來的值還有本來的上一個位元的值做 exclusive OR，例如 Dout[2] = ~Din[3] & Din[2] | Din[3] & ~Din[2]。



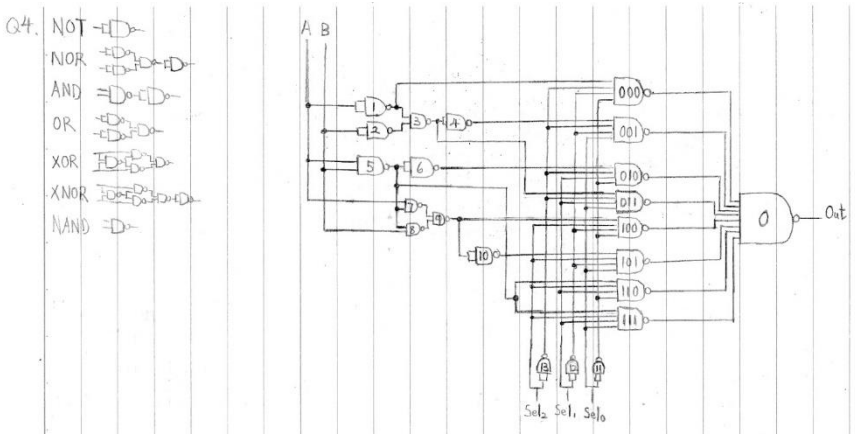


Q3

我們用 if 來根據 OP_Code 來選擇不同的計算模式，使得 Rt 會變成不同的值，例如:加法、減法、乘法、bitwise nand、bitwise nor、logical shift 以及是否等於。加法和減法部分我們用了 Full Adder 來製造，乘法則利用了 Logical Shift，NAND 跟 NOR 分別用了 4 個 Gate 來判定每一個位元，Logical Shift left &right 就用了 if 來判定五種情況，分別是不用位移、位移一格、位移兩格、位移三格以及全部位移(即是 0)。我們在幾種功能(減法、等於)有利用 Q1 的 4bit comparator 的 Rs_gt_Rt、Rs_eq_Rt、Rs_lt_R1t 等來協助我們決定 Rt 的結果。

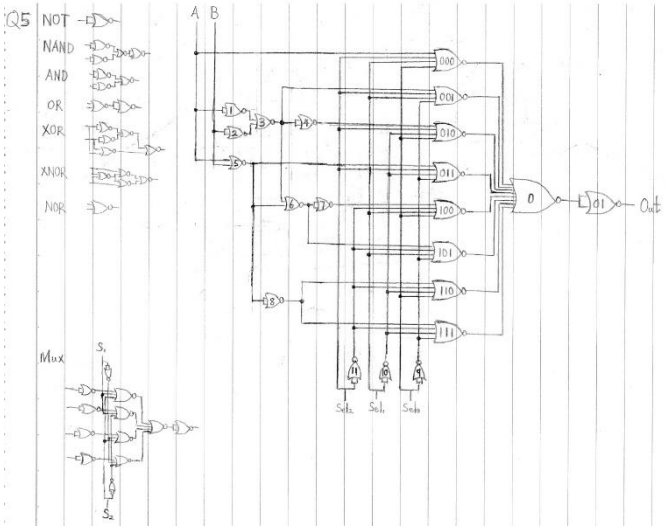
Q4

我們用不同的 NAND gate 來製造出 NOT、NOR、AND、OR、XOR、XNOR 以及 MUX 等等 functions 的電路，然後 A 和 B 的值會分別經過這些不同的電路，再用 3bit Sel 的值來選擇哪一個電路組合的值會通過，未通過的值將會強制變成 1，最後再用一個大型的 NAND Gate 把這些電路的值接起來，所以最後的值只會由 Sel 通過的值所控制。



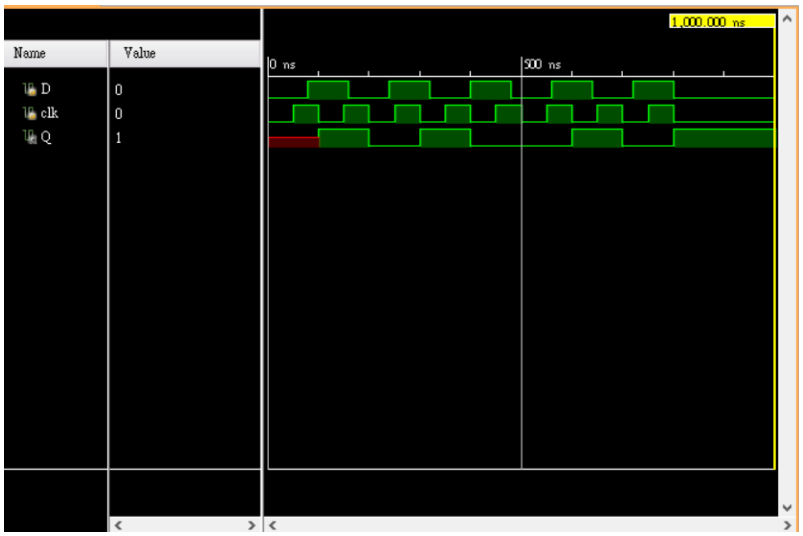
Q5

我們用不同的 NOR gate 來製造出 NAND、AND、OR、XOR、XNOR 以及 MUX 等等 functions 的電路，然後 A 和 B 的值會分別經過這些不同的電路，再用 3bit Sel 的值來選擇哪一個電路組合的值會通過，未通過的值將會強制變成 0，最後再用一個大型的 NOR Gate 串聯一個用 NOR Gate 製成的 NOT 把這些電路的值接起來，所以最後的值只會由 Sel 通過的值所控制。



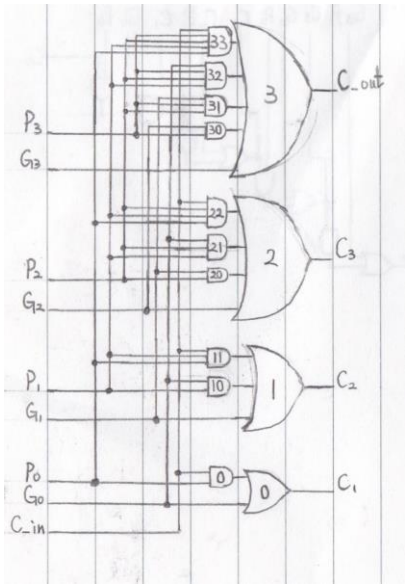
Q6

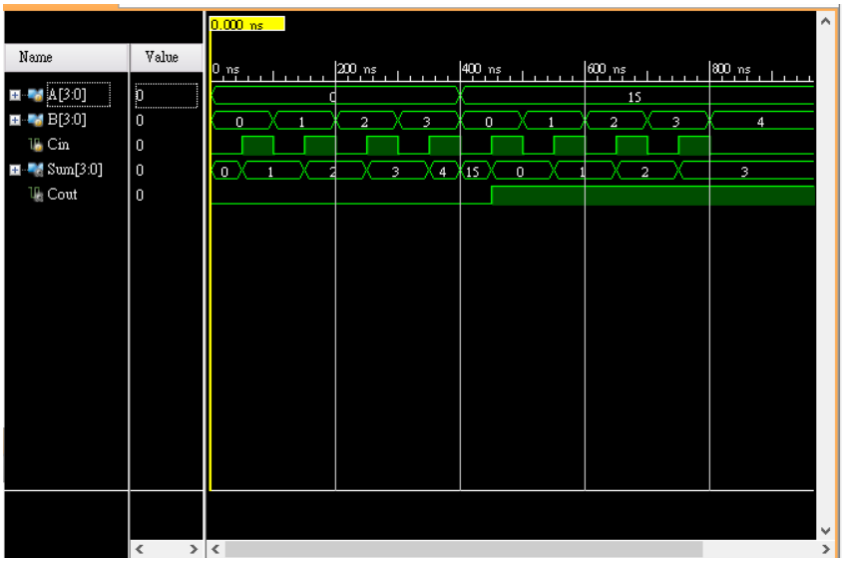
首先 latch 當 CLK 保持在 0 時，Q 的輸出跟上一次輸出保持不變，然後當 CLK 保持在 1 時，Q 的輸出將會根據 D 的值而作出改變。而 Clk negative flip-flop 則只有在 Clk 由 1 變成 0 時把 Q 的值更新為 D 的值。把兩個合起來用的話則可以先把舊的 D 值存起來，等到下一個 Clk1 變成 0 時把 Q 的值更新為舊 D 的值。



OQ_1

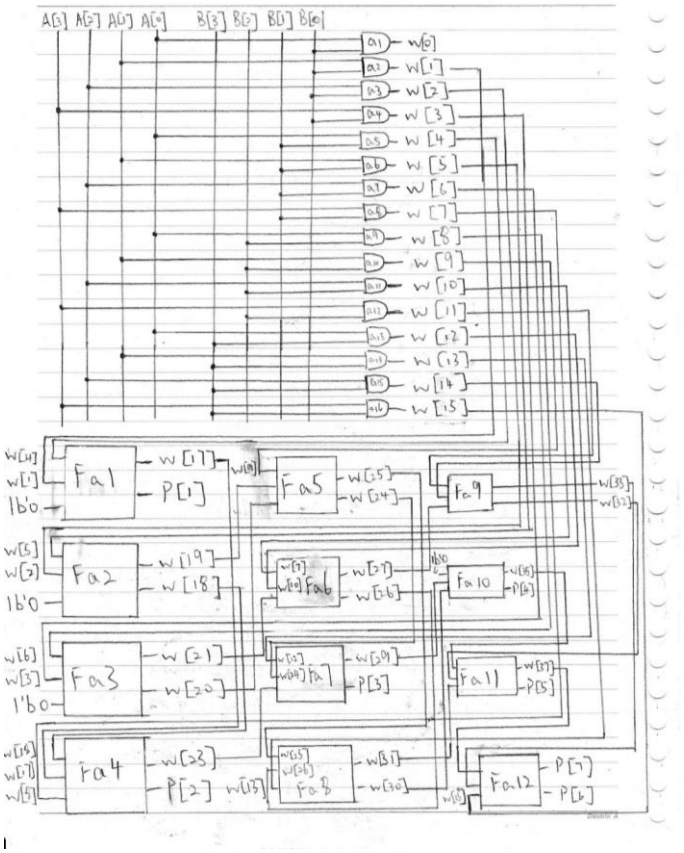
普通的 Full Adder 的 Carry 是以(A&B | A&cin | B&cin)來組成的，現在我們則可以把它化簡為 (G + P&cin)，當中的 P = A & B、G = A ^ B。而 Carry-Lookahead Adder 是用來計算電路外部的已知訊號，而非低一位的計算結果。它的好處是不用等待低位元的進位處理完成，就可以知道高位元是否需要進位，這可以節省了不少等待處理的時間。

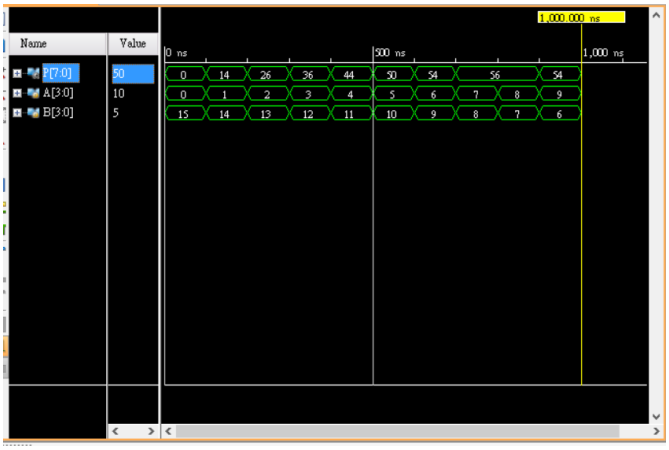




OQ_2

首先我們可以利用 16 個 AND Gate 先計算好 A 和 B 各位元之間(4*4=16)是否有進位，然後再把這些結果根據題目所給予的圖片提示：總共有 12 個 Carry out，所以我們可以分別以 12 個 Fuller Adder 來串聯起來，以達到乘法器的效果。





心得：

總覺得第三題 code 的部分我們應該想得太複雜了，應該有更簡單的方法可以寫出同樣的功能。