

1 Circuit diagram

(1) Two's complement

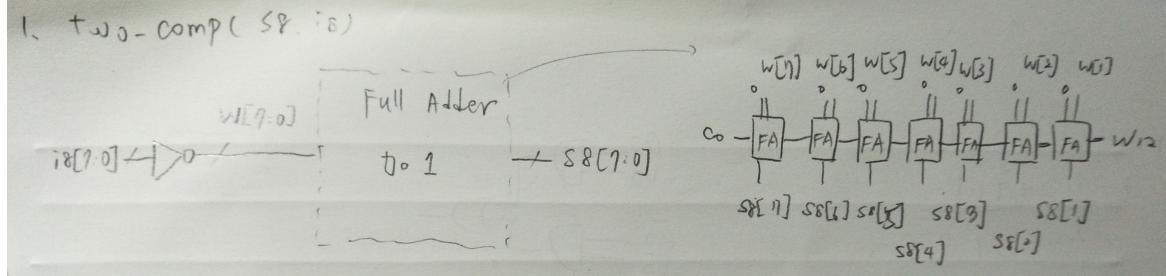
```
module two_comp(s8,i8);
    output [7:0]s8;
    input [7:0]i8;
    wire [7:0]w;
    wire w12,w23,w34,w45,w56,w67,w78;
    wire ni_1;
    wire Co;
    DRIVER gnd1(gnd,0);
    DRIVER dr(s8[0],i8[0]);
    IV carry1(w12,i8[0]);
    IV iv2(w[1],i8[1]);
    IV iv3(w[2],i8[2]);
    IV iv4(w[3],i8[3]);
    IV iv5(w[4],i8[4]);
    IV iv6(w[5],i8[5]);
    IV iv7(w[6],i8[6]);
    IV iv8(w[7],i8[7]);
    //FA1 fa1(w12,s8[0],w[0],1,0);
    FA1 fa2(w23,s8[1],w[1],gnd,w12);
    FA1 fa3(w34,s8[2],w[2],gnd,w23);
    FA1 fa4(w45,s8[3],w[3],gnd,w34);
    FA1 fa5(w56,s8[4],w[4],gnd,w45);
    FA1 fa6(w67,s8[5],w[5],gnd,w56);
    FA1 fa7(w78,s8[6],w[6],gnd,w67);
    FA1 fa8(Co ,s8[7],w[7],gnd,w78);
endmodule
```

說明：

將輸入的 8 byte 數字做 2's complement，先將所有 bit 取 inverter，然後再利用加法器的串接，做+1。

化簡：

因為第一個 bit 不會有 Cin 進來，因此如果原本是 0，補數完還是 0 (取 complement 後，加一)，因此我省略不使用加法器。



(2) Cout

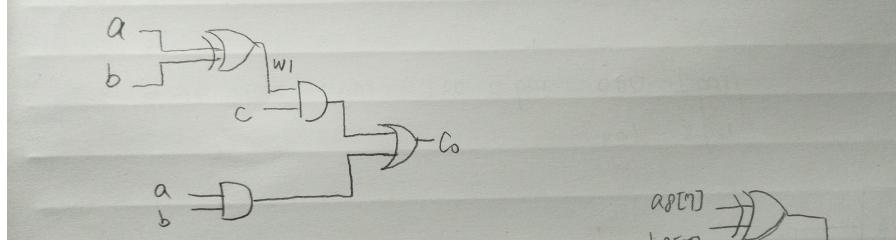
```
module Cout(co, a, b,c);
    output co;
    input a,b,c;
    E0 eo1 (w1,a,b);
    AN2 an1 (w2,w1,c);
    AN2 an2 (w3,a,b);
    OR2 or1 (co,w3,w2);
endmodule
```

說明：

Cout 就是 FA1 裡面的 Cout 部分，因為我們後面要做進位的計算，但不需要算出 S 所以重新定義一個 Cout，以減少不必要的計算。

2. Cout (Co, a, b, c)

$$Co = ab + (a \oplus b)c$$



(3) FA_8

```
//only output 1 bit least sign.
module FA_8(s8, a8, b8, cb8);
    output s8;
    input [7:0]a8, b8,cb8;
    wire w12,w23,w34,w45,w56,w67,w78;
    wire w_xo,tmp_s8;

    AN2 co1 (w12,a8[0],cb8[0]);
    Cout co2 (w23,a8[1],cb8[1],w12);
    Cout co3 (w34,a8[2],cb8[2],w23);
    Cout co4 (w45,a8[3],cb8[3],w34);
    Cout co5 (w56,a8[4],cb8[4],w45);
    Cout co6 (w67,a8[5],cb8[5],w56);
    Cout co7 (w78,a8[6],cb8[6],w67);
    EO3 co8 (tmp_s8,a8[7],cb8[7],w78);
    EO xo2 (w_xo,a8[7],b8[7]);
    MUX21H mx1(s8,tmp_s8,a8[7],w_xo);

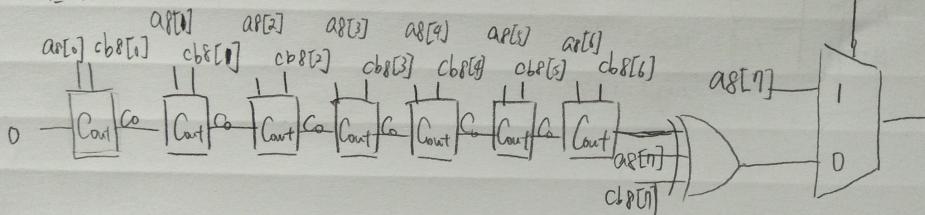
endmodule
```

說明：

這部分就像是一個 FA，只是我們只計算最後一個 bit 值是多少，其他位數，只算進位的部分。而兩組數比大小的時候，只有兩種情況，一種是都是正或都是負另一種則是一正一負的情況，第二種情況不用比較，第一種情況，就看上面減完最後一個 bit 是 0 還是 1，因此我們在最後見一個 MUX 來區分上述兩種情況。

b →

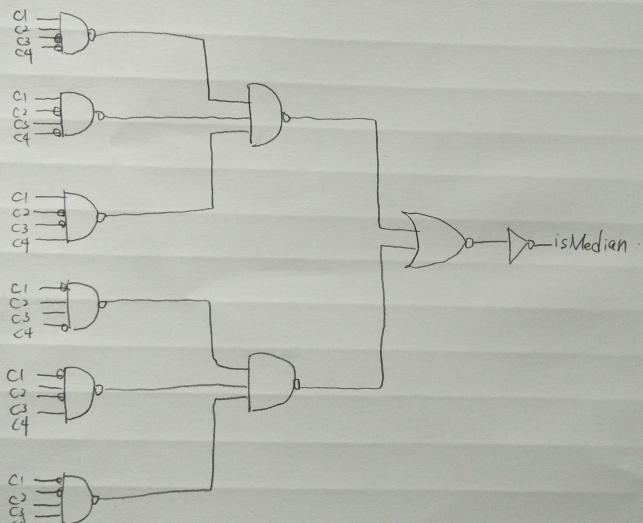
3. FA_8



(4) Median

```
module median(isMedian, c1, c2, c3, c4);
    output isMedian;
    input c1, c2, c3, c4;
    wire nc1, nc2, nc3, nc4;
    IV iv1 (nc1,c1);
    IV iv2 (nc2,c2);
    IV iv3 (nc3,c3);
    IV iv4 (nc4,c4);
    wire t1,t2,t3,t4,t5,t6,t7,t8,t9;
    ND4 nd1 (t1,c1,c2,nc3,nc4);
    ND4 nd2 (t2,c1,nc2,c3,nc4);
    ND4 nd3 (t3,c1,nc2,nc3,c4);
    ND4 nd4 (t4,nc1,c2,c3,nc4);
    ND4 nd5 (t5,nc1,c2,nc3,c4);
    ND4 nd6 (t6,nc1,nc2,c3,c4);
    ND3 nd7 (t7,t1,t2,t3);
    ND3 nd8 (t8,t4,t5,t6);
    NR2 nr1 (t9,t7,t8);
    IV iv5 (isMedian,t9);
endmodule
```

4. median



說明：這是接續上面比較完兩個數之後，我們用來判斷是不是此數有比兩個數還要大，如果有比兩個數還要大，那就一定是中位數。而二個數還要大就是其中 c1,c2,c3,c4 這四個比較結果中，一定有兩個是 1，因此將任兩個拿 invert 之後 4 個數字放到 NAND 一定會是 0，導致最後結果 isMedian 變成 1。

2 Discussion

(1) 計算方式：

Part 1:

我們先將 i_0, i_1, i_2, i_3, i_4 等五個數字做好補數並存下來，然後將這些數字倆倆比大小(丟到 FA_8 中)，因此一共有 20 組，P5 取 2 組，其中因為 0,1 比較和 1,0 比較是 invert 關係，因此我們只做其中一個，另一個用 inverter 做出。

Part2:

將 20 組比較的值全部存下來後，在把 5 個數分別放入看是不是中位數。
以 median me1 ($m_0, c_{01}, c_{02}, c_{03}, c_{04}$);為例子， $c_{01}, c_{02}, c_{03}, c_{04}$ 分別代表 i_0 和 i_1, i_2, i_3, i_4 的比較結果，如果為 0 就是 i_0 比較大，反之則後者大。放到 median 中判斷是不是有兩組的 1，這樣就可以決定 i_0 是不是中位數。

重複比較以下可能：

```
median me1 (m0,c01,c02,c03,c04);
median me2 (m1,c10,c12,c13,c14);
median me3 (m2,c20,c21,c23,c24);
median me4 (m3,c30,c31,c32,c34);
median me5 (median[2],c40,c41,c42,c43);
```

Part3:

這部分我們要輸出 median：

median[2]由 i_4 是不是中位數決定是不是為 1，因此直接當 me5 的 output。
median[1]則是由 i_2, i_3 是不是中位數決定，所以這裡使用 OR2 o1
(median[1],m2,m3);來看看是不是要寫入 1。
median[0]則是由 i_1, i_3 決定，因此同樣方式，我們用 OR2 o2 (median[0],m1,m3);
來判斷是不是要寫入 1。

(2) 比較機制：

比較機制，這裡我只使用兩數的相減，因此我們做 2's complement 之後，比兩數相加，但是如同上面 circuit diagram 解釋，因為我們只需要知道兩數減完是正還是負就可以知道誰大誰小，因此我重點算出 $a[7], b[7], w67$ 相加的結果多少，其他就進位就好。減完如果是正的，就是第一個數比較大，反之，則是第二個數比較大。

(3) 如何減少 critical path：

我們利用增加記憶的數量來減少多餘的重複計算，a.因此我在 COMPARATOR_51 中，就先將 two's complement 算好，並且增加一個 FA_8 的 input 用來存已經算好的 two's complement，b.另外我們也避免在 median 才比較每一個數的大小關係，因此我在 COMPARATOR_51 先算好所有的組合，如此 FA_8 最多只需要進行 10 次，然後所有組合資料存下來，再來做後面的 median 的比大小，以上 a,b 兩點，將原本我第一次做出來的電路速度提升到<6ns。