實用數位系統設計

HW1-2 Voting and Median

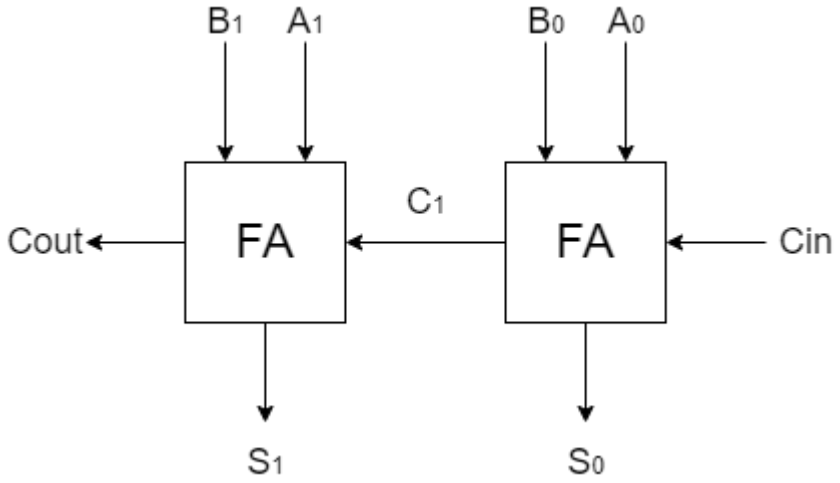
姓名:林凡皓

學號:B103012002

一、 設計原理

1. Voting :

由於輸入有 one-hot 的特性，所以只要將五筆輸入的第一個位元相加就可以得到輸入為 001 的個數，利用這樣的想法就可以得到各種輸入的個數。為了完成將五個一位元數字相加，我先使用了兩台 full adder，一台將 a0[0]、a1[0]、a2[0]相加(將 cin 設定成 a2[0])，另外一台將 a3[0]、a4[0]相加(cin 為零)，接著使用一台二位元加法器(電路如圖(一))將剛才的兩筆結果相加即可達成目標。

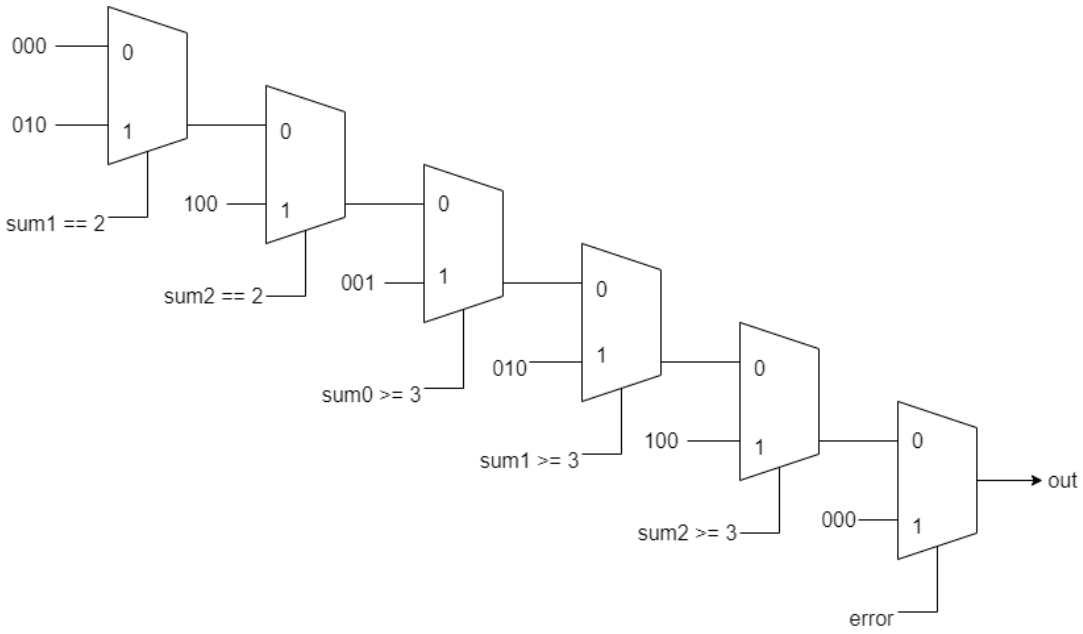


圖(一)

接下來要判斷哪一個輸入的個數最多，由於輸入總共有五筆，所以只要某個輸入的個數大於等於三，他就會是輸出。此外，考慮到有平手情況要以 MSB priority 方式輸出，我選擇使用 if-else 來進行選擇，因為 100 的優先順序最高，因此先判斷 100 的個數是否大於等於三，接著是判斷 010 個數是否大於等於三，再來才是判斷 001。如果沒有任何一筆輸入超過三個的話，由於還要考慮優先順序，所以 001 將不可能被輸出，因此輸出的候選者只剩下 100 和 010，而 100 會優先於 010，所以先判斷 100 是否等於二，再判斷 010 是否等於二。

最後是防呆機制，由於輸入只能是 one-hot，所以每一筆輸入的每一個位元相加後結果都是一，如果不是一就代表輸入不為 one-hot。我使用五台 full adder 得到每一筆輸入的每一個位元相加的結果，再利用 error 訊號來判斷是否有輸入不為 one-hot，如果有，error 為 1，此時輸出會變成 000；如果沒有，error 為 0，此時輸出會是正常結果。

電路如下圖



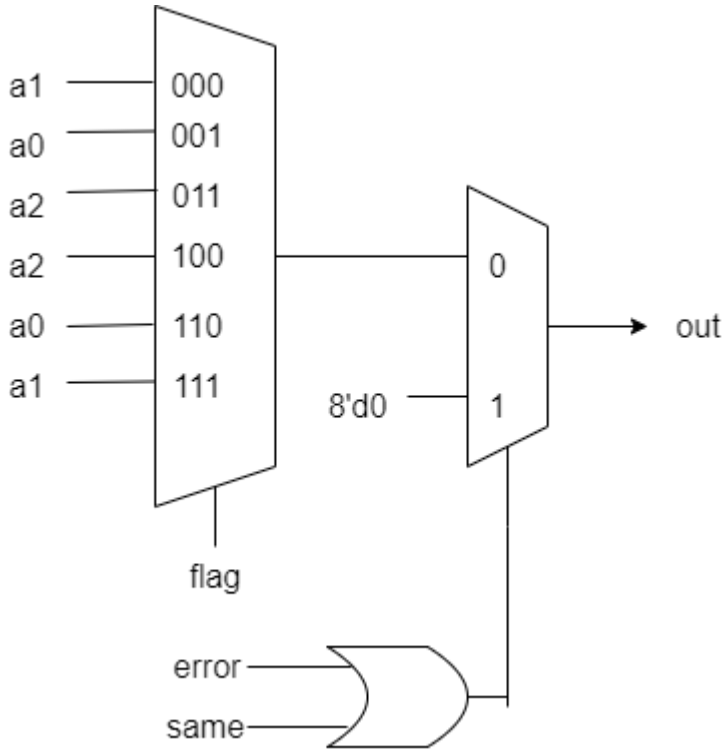
2. Median :

我利用比較的方式來找到中間值。flag 訊號的第一個位元用來判斷 a1 與 a0 的大小(flag[0] == 1 代表 a1 > a0)，flag 訊號的第二個位元用來判斷 a2 與 a0 的大小(flag[1] == 1 代表 a2 > a0)，flag 訊號的第三個位元用來判斷 a2 與 a1 的大小(flag[2] == 1 代表 a2 > a1)。利用以上關係，我將所有狀況分成六種狀態(先不考慮輸入非 one-hot 或有一樣的輸入的情況):

(1) flag == 000，即 a0>a1>a2，此時輸出為 a1
(2) flag == 001，即 a1>a0>a2，此時輸出為 a0
(3) flag == 011，即 a1>a2>a0，此時輸出為 a2
(4) flag == 100，即 a0>a2>a1，此時輸出為 a2
(5) flag == 110，即 a2>a0>a1，此時輸出為 a0
(6) flag == 111，即 a2>a1>a0，此時輸出為 a1

接著是處理輸入有一樣的情況。訊號 same == 1 代表輸入有一樣，此時找不到中間值，所以輸出會變成 0。

最後是防呆機制，由於輸入只能是 one-hot，所以我使用訊號 error 來判斷每個輸入是否符合規定。跟 Voting 不同的是，這次我沒有選擇使用 full adder 來將每一個位元加起來判斷結果是否為一，因為輸入的位元數比較大，所以會需要很多的 full adder。One-hot 除了會使每個位元相加的結果為一之外，還會使數值只會等於 2 的 N 次方，所以只要判斷輸入的值是否等於 1、2、4、8、16、32、64、128 就可以達成判斷輸入是否為 one-hot。若 error == 1 就代表有輸入不符合規定，則輸出會變成 0。電路如下圖

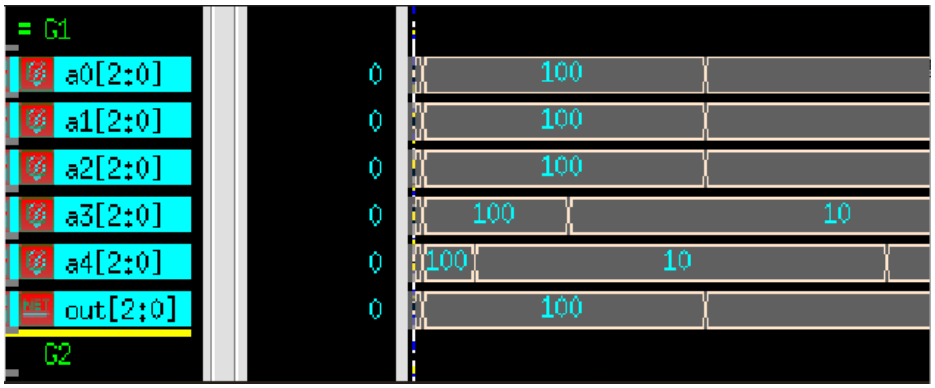


二、 結果討論

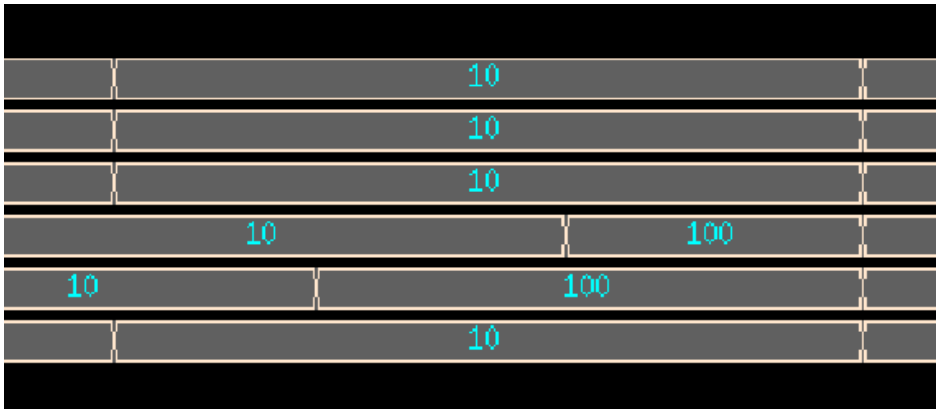
1. Voting:

先討論有某一個輸入大於三個的情況。

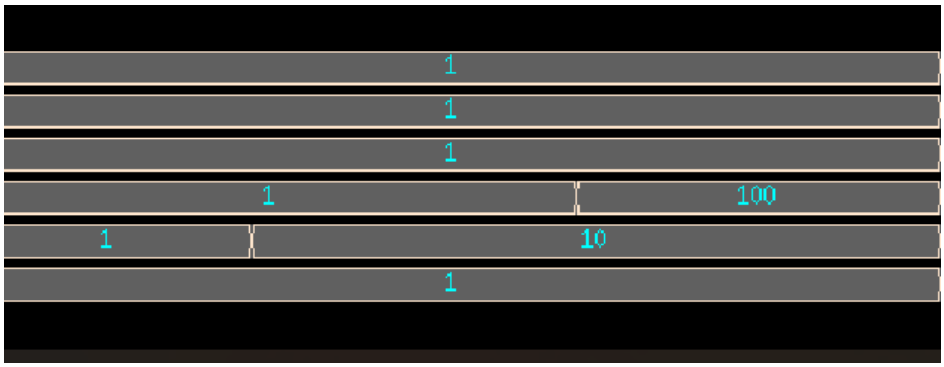
(1) 100 的個數大於等於三，此時輸出為 100，結果如下圖



(2) 010 的個數大於等於三，此時輸出為 010，結果如下圖

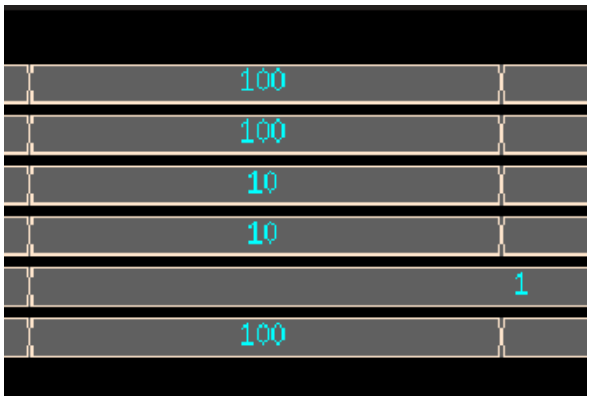


(3) 001 的個數大於等於三個，此時輸出為 001，結果如下圖

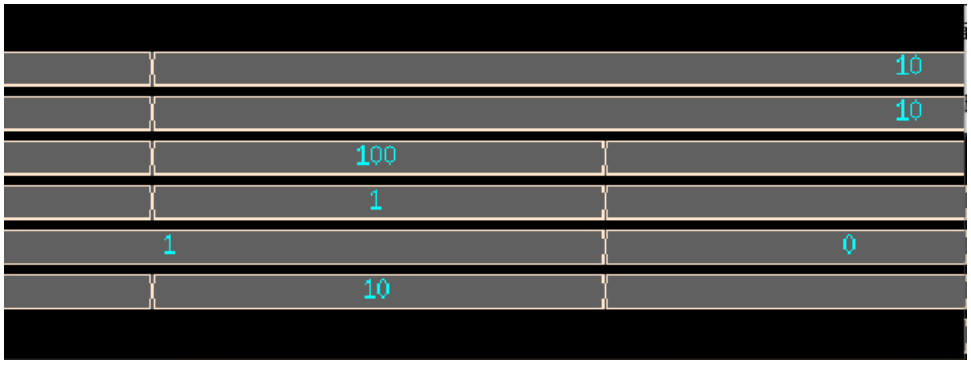


接著討論輸入個數為 2:2:1 的情況，此時只有 100 或 010 有可能被輸出。

(1) 100 有兩個，此時輸出為 100，結果如下圖

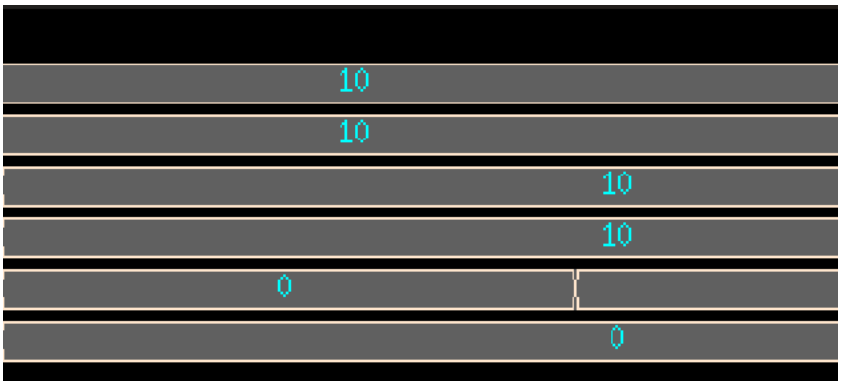


(2) 010 有兩個，此時輸出為 010，結果如下圖

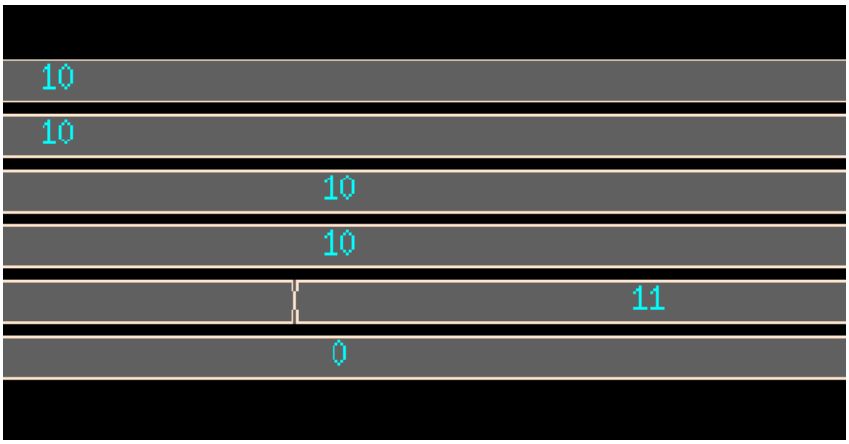


最後討論輸入有錯誤的狀況。

(1) 有一筆輸入為 000，此時輸出為 000，結果如下圖



(2) 有一筆輸入有兩個位元為 1，此時結果為 000，結果如下圖



2. Median :

先討論輸入符合 one-hot 和輸入沒有一樣的結果:

(1) 當 flag == 000 時，即 a0>a1>a2，此時輸出為 a1

(2) 當 flag == 001 時，即 a1>a0>a2，此時輸出為 a0

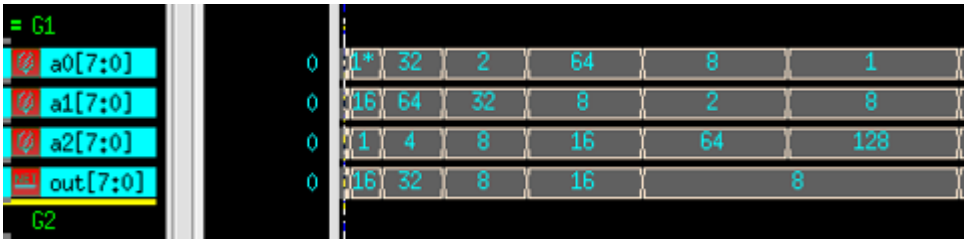
(3) 當 flag == 011 時，即 a1>a2>a0，此時輸出為 a2

(4) 當 flag == 100 時，即 a0>a2>a1，此時輸出為 a2

(5) 當 flag == 110 時，即 a2>a0>a1，此時輸出為 a0

(6) 當 flag == 111 時，即 a2>a1>a0，此時輸出為 a1

波型圖如下圖



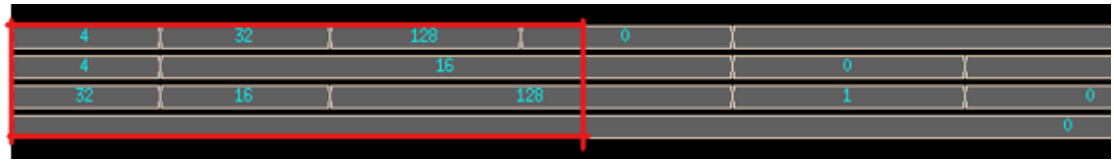
接著討論輸入有一樣的情況:

(1) a0 == a1，此時輸出為 8'b0

(2) a1 == a2，此時輸出為 8’b0

(3) a0 == a2，此時輸出為 8’b0

波型圖如下圖



最後討論輸入不為 one-hot 的情況:

(1) a0 == 8’b0，此時輸出為 8’b0

(2) a1 == 8’b0，此時輸出為 8’b0

(3) a2 == 8’b0，此時輸出為 8’b0

(4) a2 == 8’b00000011，此時輸出為 8’b0

(5) a1 == 8’b00010001，此時輸出為 8’b0

(6) a0 == 8’b10000001，此時輸出為 8’b0

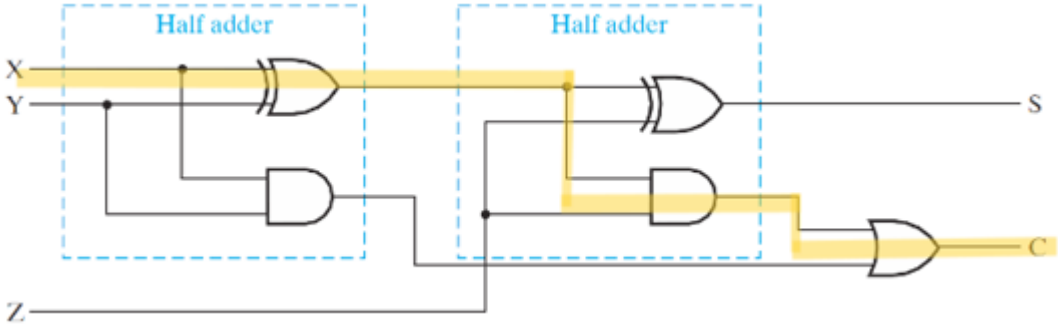
波型圖如下圖

三、 電路分析

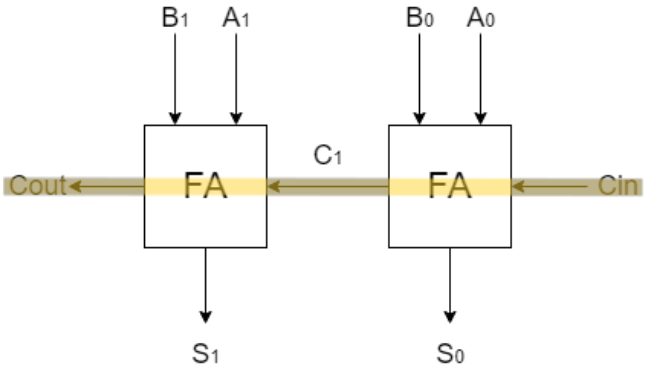
1. Voting :

Critical path :

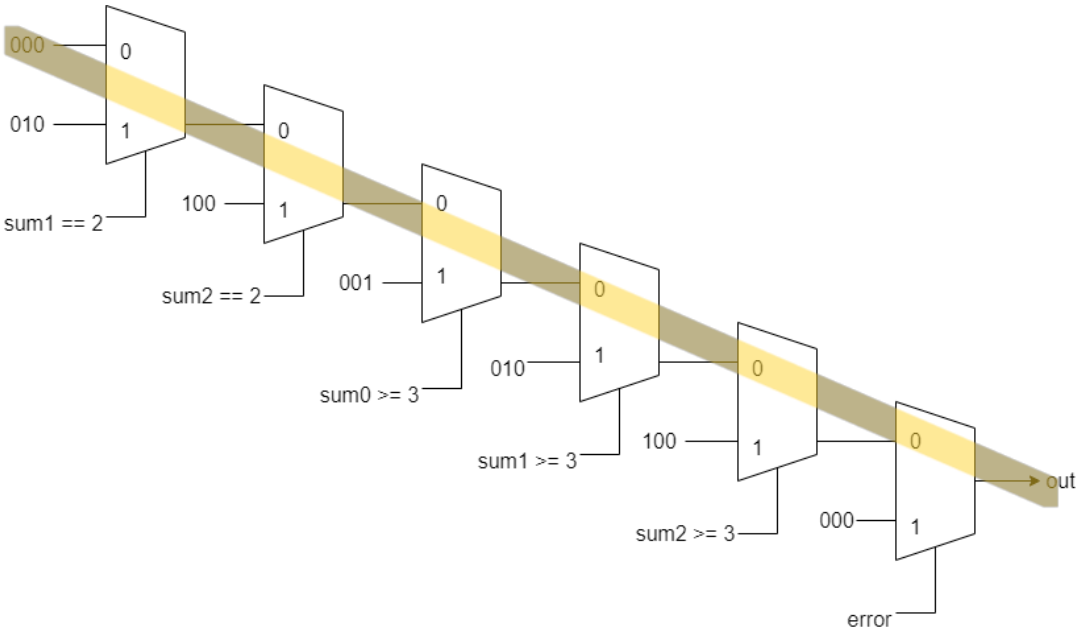
首先要計算出輸入分別為 001，010，100 的數量，以 001 的數量為例，此步驟需要先經過兩台 full adder，如下圖



接著將 cout 和 sum 變成一個二位元的數字(此時會有兩個由 cout 和 sum 所合成的二位元數字)，再將這兩筆數字送進一個二位元加法器如下圖



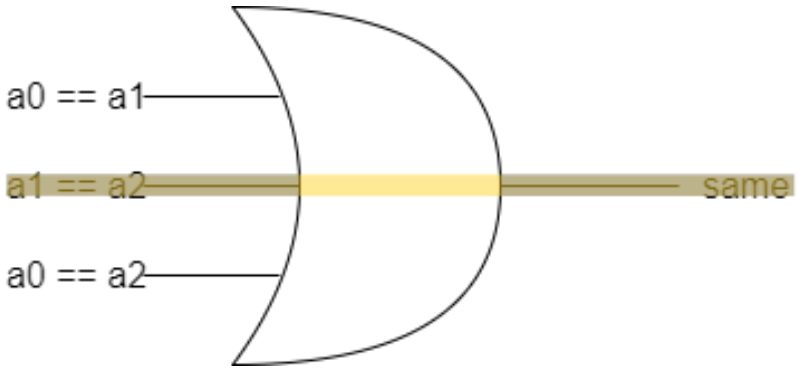
得到三種輸入的各別個數後，就要進行輸出的選擇，經過六個多工器後即可得到輸出結果，如下圖

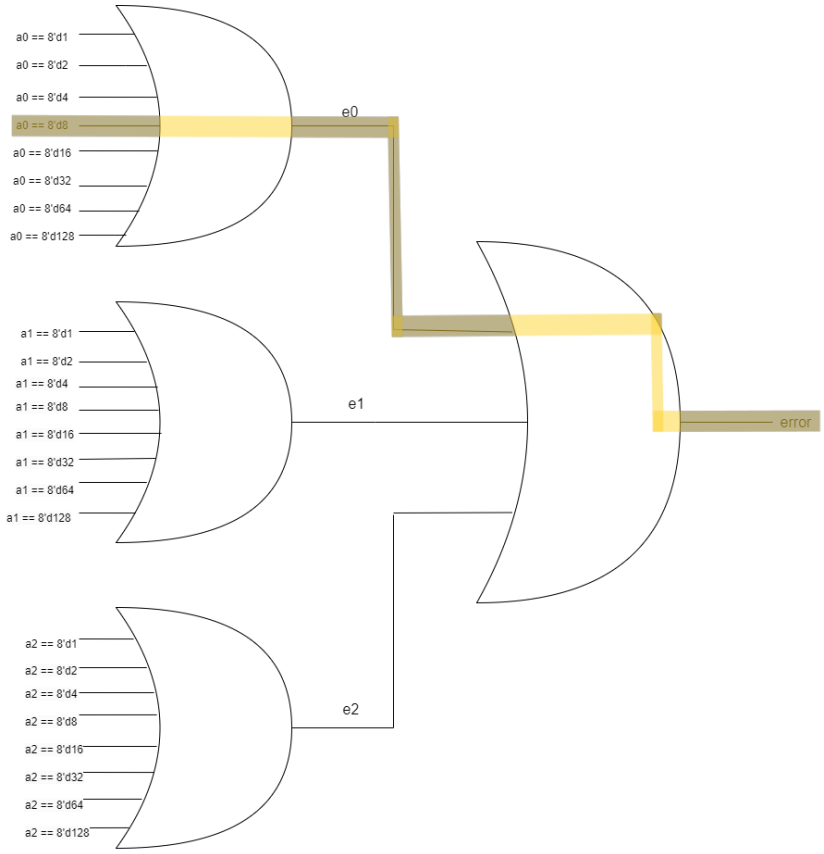


2. Median :

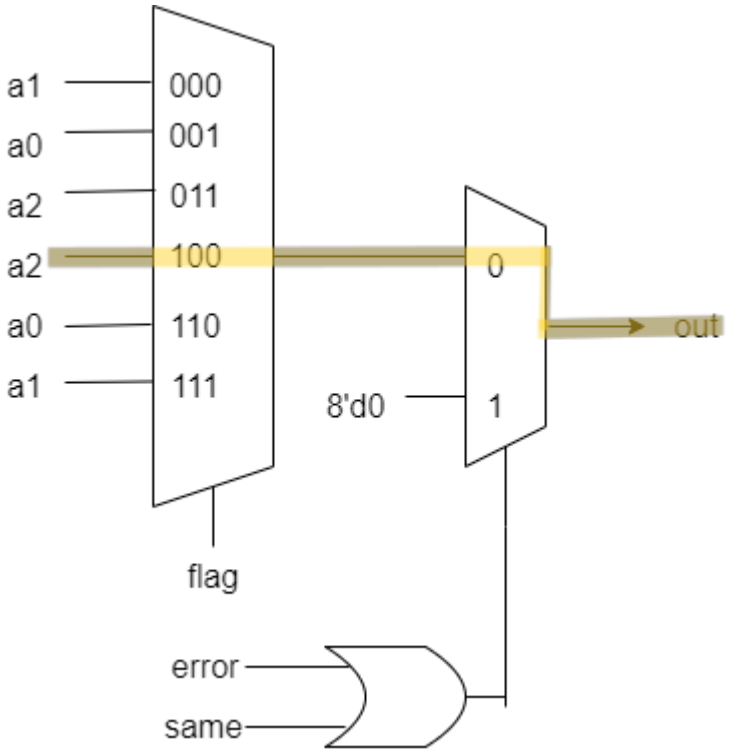
Critical path:

要先產生 error 和 same 訊號，如下圖





接下來是透過 always block 來決定輸出，如下圖



因為一個六對一的多工器的 delay 會比產生 error(兩個 or gate)和產生 same(一個 or gate)的 delay 還要長，所以 critical path 會經過六對一的多工器。

四、 問題討論

1. Voting :

(1) 要怎麼做五個一位元數字的加法?

這是我遇到的第一個問題，雖然說上一個作業做過八位元的加法器，但是不能直接拿來用，因為那個加法器是兩個數字相加，但是這次需要的是五個數字相加。我的解決辦法就是使用兩台 full adder，因為每一台 full adder 可以做三個數字的相加(cin 也算是一個輸入)。這樣做出來的結果會有兩個 cout 和兩個 sum，此時需要將第一台和第二台 full adder 的 cin 和 sum 分別並再一起變成一個二位元的數字，這樣子才會符合運算結果。接著再將兩個二位元的數字送進二位元加法器就可以得到一個 cout 和一個二位元的 sum，再將 cout 和 sum 並在一起就能得到五個一位元數字相加的結果。

(2) 要如何判斷結果?

有了輸入為 001、010、100 的個數後，就要判斷該輸出什麼，這裡有兩個選擇，第一個是使用 if-else loop 來判斷，第二個就是使用 case 來判斷。因為有權重，所以硬體架構會比將像是很多個多工器而不是一個大的多工器，所以我認為 if-else 會比較符合硬體架構應該有的樣子。

2. Median :

(1) 如何討論各種輸入狀況?

要找出中間值就需要經過比大小的步驟，a0、a1、a2 之間的大小關係總夠有六種，所以我設定了一個三位元的參數 flag 代表這六種狀況。接著就是 if-else 和 case 的選擇，這邊我選擇使用 case 因為這個電路不需要考慮權重，硬體架構比較像是一個大的多工器而非很多小的多功器，除此之外，使用 case 的話，速度上的表現也會比 if-else 來的快。

(2) 如何偵測輸入錯誤或輸入一樣?

偵測輸入依樣比較容易，只需要一個三個輸入的 or gate 就可以完成了。至於偵測輸入不符合 one-hot，不使用跟 voting 一樣將每個位元加起來看是否於 1 的原因在於這個方法會許邀非常多的 full adder，這樣會造成運算速度很慢，我目前想到的辦法是用三個八個輸入的 or gate 和一個三個輸入的 or gate 來判斷三筆輸入的值是否都等於 8'd1 or 8'd2 or 8'd4 or 8'd8 or 8'd16 or 8'd32 or 8'd64 or 8'd128，但是這樣的方法也有問題，首先是要找到可以容納八個

輸入的 or gate，再來是八個數入的 or gate 面積會很大，所以未來我還需要思考有沒有更佳的解辦法。

五、 心得

這次作業花的時間比上次的作業多不少，但是從這次作業中的嘗試和失敗中也學到了不少事情，像是 case 和 if-else 變成實際電路後的差別以及使用時機，還有 full adder 可以用各種方式組合起來來完成各種電路功能。除了學到東西之外，從這次作業中我也更加知道自己的不足，像是這次作業遇到最大的問題是寫完 code 之後不太了解如何去優化電路，對於自己設計出來的電路也不太會評估是好是壞，這會是下一次作業我希望能夠改善的問題。