Final Project:Fraig

B07901055 電機二 許宸睿

一、內容

1.引用：此份作業使用的是電機二朱哲廣（廣哥）所寫的 CirGate 進行改編。

2.sweep:

這個 function 是最直白易懂的，只要將從 output 抵達不了的 gate 取消即可，但麻煩的是必須跑過所有的 aig gate，而要判斷這個 aig gate 可能又要跑過整個 dfs list，因此，廣哥在每個 gate 裡面增加了一個 bool 來確認這個 gate 有沒有在 dfs list 裡面，省下了不少時間。

其餘需要注意的便是刪減的時候要將 fanin 的 gate 和 fanout 的 gate 去做處理，需要非常小心。這邊我自訂了三個 function，分別是
remove_from_fanin、remove_from_fanout、remove_from_aig
後面像 fraig、optimize、strash 也有用到需要取代的部份，有了這三個函式之後，將 gate 取代掉就變得十分輕鬆寫意了。

3.optimize:

optimize 需要比較兩個 gate 的 fanin 腳位，我將他們分成五種情形，分別是兩隻相反，同源且同相，同源但反向，接過來是零，左腳是一及右腳為一。但其實真正的情況可以縮減成四種，因為同源反向和零的狀況是相同的，而同源則可跟一腳為一的合併。這裡的其他部份需要注意的皆跟上面相同，注意刪掉之後上下也須一起變化即可。

4.strash

strash 要做的是將兩隻腳位及相性相同的 gate 合併在一起，而如果使用一對一比對需要耗時 $O(n^2)$，因此我採用 hash 來執行，這邊因為已經忘記 hw7 的內容了，所以使用了 STL 的 unordered_map，配合上自己寫的 hash function 來完成。而 hash function 的部份，因為兩個腳位的順序可互換，我將腳位一的 address 和另一腳位的 address 做 xor，如果任意腳位有反相的話，便會將該腳位的 address 向左推移 1 個 bit。如此一來，腳位順序相反也可以找到對應的 gate。

5.simulate

我在各個 gate 上加上了 size_t _sim 的變數，當執行 simulate 的時候，便會從 PO 一路遞迴往下，直到 PO 或是 UNDEF。

處理 pattern 的部份，因為老師說測資不會給怪力亂神，因此我只檢查了有無非 0，1 的元素在裡面，還有是否可供所有的 PI 使用這兩個條件而已。而只要蒐集滿一個 size_t 的大小，便會先拿去做 simulate，因此如果 pattern 是在 64 行之後才出錯的話，我仍然會蒐集到一定數量的 FECpairs。

至於 random 的部份，我使用 STL 裡面的 rand()函數來產生一個整數，並將其一往前 4bytes 後合併而成，得到一個 size_t 的測資，至於測試到什麼時候，則是確認 FECpair 的數量有無變化，若已經經過 pi_list/2 次沒有變化，則便測試完畢。

第一次做 simulate 的時候，也就是完全沒有 FECpair 的時候，我會用 hash 去將之進行分類，而 hash key 便是透過測資後得到的數值(_sim)，此時不管只有一個 gate 或是多個 gate 皆會封裝成一個 FECpair。而進行第二次的時候，我只針對 FECpair 裡超過一個元素的元素進行再次的分類，但個數因為少了不少，所以我將內容物在 100 以上的 FECpair 繼續使用 hash 分類，而其他的則使用 map 來分組。

6.fraig

fraig 是利用 simulate 之後的結果去進行分類，而在分類之前我先進行排序，將只有一個 gate 的 pair 砍掉，再將同個 pair 的合併。因為沒有時間去研究 sat 的東西了，所以就用了自己覺得可以達到目的的方式完成這個功能。

7.write gate

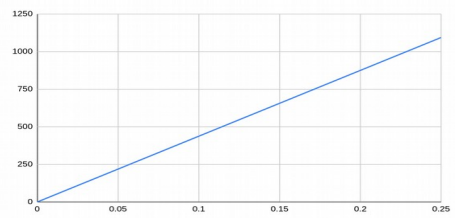
這邊特別寫出來是因為不小心多幫 pi 做了 sort，雖然沒有必要但看起來比較好看。

二、比較

由於沒什麼時間，所以只用了 sim6~sim15(沒有 sim11,sim8)來當作測資

1. sweep

關於這個函數，會跟在 dfs 外的 gate 有比較大的關聯性，而在這次的測試中，除了 sim06 花的時間比較大外，其他皆差不多為 0 秒，那也實際去測了一下，也大致符合這個結果。但效率比老師的低非常多。




2.optimize

雖然會因為 gate 的數量影響，但是影響最大的原因是 gate 被合併的數量多寡

| | |
|---|---|
| 4270 | 0.01 |
| 9437 | 0.01 |
| 3286 | 0.03 |
| 716 | 0 |
| 9364 | 0.01 |
| 81710 | 0.01 |
| 886 | 0 |
| 886 | 0 |

這是以 gate 數量對時間做的圖表，左列為 gate 數，右列為秒數，可以發現 gate 最多的 sim13 也只不過花了 0.01 秒。而 3000 多個 gate 的 sim09,因為 optimize 的時候減去了 1209 個 gate，所以耗時最長。

3.simulate

simulate 耗時有兩個不同的因素，一個是 gate 的數量，一個是 input 的數量，

| | | |
|---|---|---|
| 4 | 9437 | 0.01 |
| 178 | 3286 | 0.03 |
| 36 | 716 | 0.01 |
| 277 | 9364 | 0.26 |
| 3357 | 81710 | 133.2 |
| 41 | 886 | 0.01 |
| 41 | 886 | 0 |

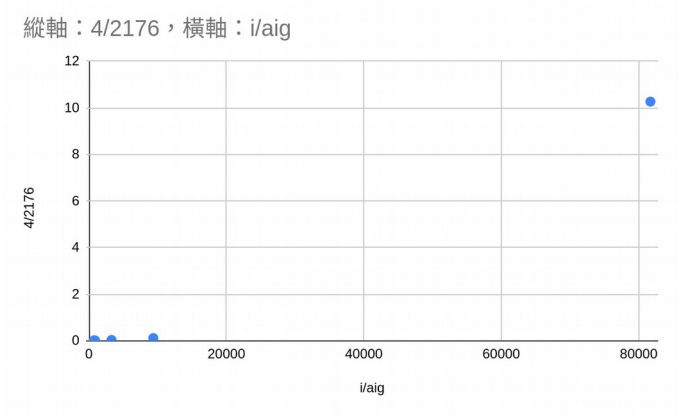
第一列是 input 個數,第二列是 aig 個數,第三列是秒數，可以看出來趨勢大約為，input 數量越多，耗時就越長，但 aig 數量也多的話，就會造成許多樣式的結果。而且我的停止條件是跟 input 數量有關，也就是就算很快就完成分類，還是需要跑一段時間才會完成。

4.fraig

這主要就和 FECpair 的數量有關了，數量越多則耗時越長。

5.strash

strash 則完全與 gate 數量有關




（由於 google 圖表無法繪出線條，只能這樣）

可以看出有非常明顯的正相關。

三、結語

修過這學期的 DSnP 後，真的覺得不虛此行，不僅讓我在短短的三個月內從零到有學習 c＋＋，還增進了不少對於打 code 及資料結構的概念。『心中要有記憶體』這已成為這堂課最熟習的一句話，希望之後能將這句話銘刻到腦內以及打出來的程式中。