資料結構期末專題報告

-Functionally Reduced And-Inverter Graph Report

作者:陳冠豪

學號:B05901168

聯絡方式:b05901168@ntu.edu.tw

1. Goal:

實作一個程式能夠從 AAG 檔建構出電路圖且具有化簡電路的功能,關於讀檔案以及建構電路的部分在 HW6 已完成,此次專題目標在完成電路化簡的功能.在第三點會列出此程式支援化簡電路的方法(透過 command interface 觸發)以及該方法所需要的演算法.

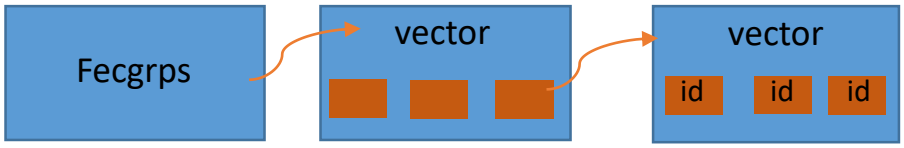
2. Data structure:

在說明化簡電路的方法前,在此先介紹此程式的 data structure

1) GATE: 總共分為五種 GATE(PI , PO , AIG , CONST , UNDEF) , 此五種 GATE 皆繼承 CirGate , CirGate 裡儲存每種 Gate 皆需要紀錄的 Property(ID,sim,visit...),而每種 Gate 再依各自的性質不同有特殊的 data member,例如 PO 需要紀錄 Fanin ,AIG 需要紀錄 FEC 等.
2) CirMgr: 這個 class 是來記錄 AAG 檔所描述的電路性質,例如 AIG , PI , PO 的 ID,以及 DFSlist,並且有一個 Global 的 array 來記

錄 ID 對應到的 Gate Pointer.以及其他可以用來幫助化簡電路的資料.

3) 關於 FEC 以及 Fraig: 這裡要記錄的有哪幾組 Gate 是 functionally equivalent candidate(FEC) pair，因此在 CirMgr 裡增加一個 data member – Fecgrps ,其為一個 pointer 指向一個 vector, 這個 vector 裡儲存的是指向 fecgrp (為一 vector,裡面存的是同一個 fecgrp 的 ID) 的 pointer,整個結構如下圖:



會選擇用 pointer 儲存的原因在後面會提到.

3. Support Command

1) CIRSWEEP:

這個 command 能夠把沒辦法從 PO(output gate)到達的 GATE 刪除掉,要完成這個的功能方法很簡單,只要從 PO 做一次 DFS,做完之後,所有可以從 PO 到達的 Gate 都會被造訪,所以把 visit == false 的 Gate 刪除掉即可.

2) CIROPTIMIZE:

這個 command 化簡四類 Gate ,前兩類 Gate 的 Fanin 有 CONST0 或 CONST1 ,對於 CONST0 的 Fanin,這個 Gate 必定為 0,因此可以把這

個 Gate 直接用 CONST0 取代, 對於 CONST1 的 Fanin , 這個 GATE 的 Output 為 1 或 0 由另一個 Fanin 決定,因此這個 GATE 可以被省略,直接把 Fanin 接到 Gate 的 Fanout 即可,要完成這個功能的演算法也很簡單,因為經過這個 command 後 CONST0 的 Fanout 不可以有 AIG,所以只要檢查 CONST0 的 Fanout 有沒有 AIG,有的話就做上述的動作,直到 Fanout 沒有 AIG 為止.後兩類 GATE 的兩個 FANIN 完全一樣,因此可以被 CONST0 或 CONST1 取代,取代後就變成前兩類 Gate,再依照上述的化簡方式即可.

3) CIRSTRASH:

這個 command 把有著一樣 Fanin 的 Gate 找出來並合併,想法很簡單,但是在實作的時候如果直接取兩個 Gate 來比較的話,這樣複雜度是 n^2,因此這裡採用另一種方法-透過 Hashmap 減少計算的複雜度,只要把每個 gate 的 Fanin 當作 Hashmap 的 key,如果兩個 Gate 的 Key 一模一樣,則這兩個 Gate 就是有一模一樣的 fanin,因此可以被合併.

4) CIRSIMULATION:

這個 command 可以找出 FEC 以及 IFEC pair.實作的方式為一開始將所有的 GATE 放在同一個 Fecgrp,接著重複以下步驟

一, 賦予 PI 一個大小為 size_t 的值,並依照 DFSList 去 simulation 每

個 Gate, (simulation 之後,如果是 FEC 或是 IFEC pair,兩個 Gate 的值應該會是一模一樣,或是值相加為 SIZE_MAX)

二,建立一個 Hashmap, hashdata 是一個 vector 的 pointer,而 hashkey 是每個 gate 經過 simulation 之後的值或是 simulation 之後的值再經過 invert 的楚理,這樣 FEC 和 IFEC 就會有同樣的 HashKey, 此時每個 gate 在放入 hashmap 時假設有 query 到相同的 key,則把這個 gate 的 id 放入 hashdata 所指到的 vector 裡,沒有的話則建立一個新的 vector 並將 pointer 放入 hashmap 裡,將所有 gate 都放進去後,再將 hashmap 裡的 vector pointer(size 要大於二)放到 fecgrps 就得到新的 fecgrps

透過上述兩個步驟來不斷更新 fecgrp,那要更新到哪時候才要停止呢?我採用的方法是設定一個 baseline,當 simulation 之後,fecgrps 更新後所有 fecgrp 裡的 Gate 數目減少沒有超過 baseline 就停止.

效率討論:這個 command 我整個砍掉重寫過,因為第一次寫的程式跑的速度實在是太慢了,後來發現原因是當時紀錄 fecgrps 是直接用 vector,而不是用 pointer,這也造成在更新 fecgrps 時許多額外的 vector copy,在 simulation 上萬次時速度就會很明顯變慢.

5) CIRFRAIG

這個 command 可以證明 CIRSIMULATION 找出的 FEC pair 是否相

等,相同的話會將 Gate 合併.

要證明一組 FEC pair 是相等可以透過呼叫 SAT engine 來證明,這部分不在這次專題的範圍內,我們可以直接呼叫寫好的 SAT engine. 因此要完成這個 command 只要把每組 FEC pair 放到 SAT engine 即可完成,但是為了增加程式執行的速度以及,我透過以下方法來完成

1. 從 PI 到 PO 的順序開始解,要這樣解的原因是因為在靠近 PI 的 Gate 如果解完,不管結果是 SAT 或是 UNSAT,對在解該 Gate 的 fanout 會有幫助,因為這些 fanout 的 simulation 值會受這個 Gate 影響.
2. 每當解完一些 FEC pair 後,把會產生 SAT 的 PI 值存下來,當收集夠一定的量(我是選 8 組)後利用這些值去更新 fecgrps ,這樣就可以大幅減少要解的 FEC pair 數量.
3. 當一組 FEC pair 證明是相同的時候,會先將這兩個 Gate ID 當作 data , fecgrp 的編號當作 key 放入 hashmap,等到解完所有的 FEC pair 後才依照 DFS 的順序將邏輯相同的 Gate 合併,最後在一起合併的好處是不用一直更新 DFS list,也讓寫程式時不用考慮某些 Gate 是否已經被合併了而造成 segmentation fault.

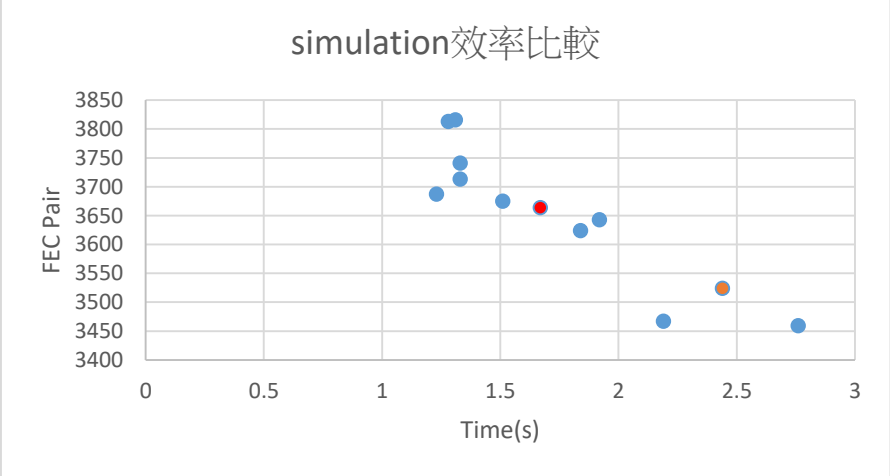
4. 效率分析與討論

為了比較執行的效率,我以執行 sim13.aag 當作分析的依據,以下是程式測試的環境

作業系統	CPU	測試對象	比較依據
Ubuntu18.04	i7-8700	自行開發的 fraig 程式	教授提供的 ref

1)第一個測試的是 simulation 的執行效率,總共測試了 10 次,分別記錄所花的時間,pattern 數量,FEC pair 數目

	Time	FEC	Pattern
1	2.76	3459	76864
2	1.23	3687	29504
3	2.19	3467	60024
4	1.51	3675	41024
5	1.28	3813	31424
6	1.84	3624	49344
7	1.31	3816	32064
8	1.92	3643	51904
9	1.33	3741	33964
10	1.33	3713	33344
平均	1.67	3663.8	43946
Ref	2.44	3524	93604



上圖紅色是平均直,橘色是 ref,可以看到平均花的時間是較少的,

但是有較多的 FEC Pair.在我的程式裡可以調整 baseline(詳見 3.4)讓程式花較多時間來減少 FEC Pair 數目,經過嘗試了許多不同的參數後,上面的結果是我覺得比較適合的,或是可以再多一個 command 來讓使用者調整 baseline 應該也不錯(如果還有時間的話.....)

2) 第二個測試的是 fraig,以下是所花的時間,

	Ref	Mine
時間	46.54	23.68
在 DFS 裡的 gate 數目	84022	83961

雖然我也不太清楚為什麼所花的時間只有 Ref 的一半,而且 gate 的數目也比 Ref 還要少,一開始我以為是我有寫錯,找了很久的 bug,但是後來發現 Ref 的 CirFraig 好像不會把所有 FEC pair 解完,因此要多做兩三次 fraig 後才會化簡成 83961 個 Gate,另外經過 Cirp －n 出來的會有一些差距,但是我有用 CirSim －f 來看電路的功能是不是有變,結果應該是沒有的,所以假設我沒有理解錯誤的話我的這個結果應該是正確的而且效率比 Ref 還要好一點.

5.心得

寫這個專題真的讓我學到蠻多的,尤其是在如何改善程式速度方面,因為在之前的作業好像都沒有遇到複雜度很高的運算,所以基

本上不太需要考慮速度的問題,但是在這次專題裡,有些 command 執行需要到好幾秒鐘,這時候速度差異就很明顯了,記得第一次寫完 CIRSIMULATION 後,發現跑 sim13 要十幾秒,而 Ref 只要不到 3 秒,因此想了很多方法來增加效能,第一次覺得 pointer 的使用真的很重要,以及如何在面對不同的應用時決定系統的資料結構,雖然也因為這個專題讓我期末讀書時間少了很多 QQ,不過真的覺得這是一門好課!!!