資料結構與程式設計

Final Project --- FRAIG

電機三　B03901012　蔣采容

Email: b03901012@ntu.edu.tw

Phone: 0987528073

Date: 2017/1/18

1. 資料結構

1.1 CirGate

Data Members：為所有 Gate 的基本單位，儲存的資料包括 ID、Name、於 AAG 檔中的 Line 和 Column、Gate 本身的 Type (0=PI, 1=PO, 2=AIG, 3=CONST, 4=UNDEF)，以及這個 Gate 的 Fanin List 和 Fanout List。此外，每個 Gate 還儲存了用來標記是否 traverse 過的 Color、在 simulation 中的 32-bit Output Result、和在 SAT solver 中的 Var。

Member Functions：包括用來取 data members 的眾多函式、修改 data members 的眾多函式、還有一些 printing 的函式。此外，我還有以 gate 為單位來做 DFS 的遞迴函式，以及用來和另外一個 Gate Merge 的函式。

1.2 CirMgr

Data Members：包含儲存輸入和輸出的 vector，PI List 和 PO List，另外所有的 Gates 會用一個 map<unsigned, CirGate*>來儲存，方便由 ID 找到 Gate 本身。另外還有儲存 MILOA 的數量、DFS List 的 vector、所有 FEC Groups 的 vector、以及指向 SatSolver 的 pointer。

Member Functions：主要是 project 規範的所有 command 對應到的函式，另外我有定義一些私人的函式來更新 DFS List 和把現有的 FEC Groups 更進一步切割。

1.3 fecGrp

用來儲存一個 FEC Group 的 candidates 的 class，內含一個 pair<CirGate*,bool>的 vector，CirGate*用來指向 candidate，bool 則是用來儲存這個 candidate 是否為反向，如果是的話，之後判斷其 simulation result 都要反向來看。之所以要這樣，是因為 FEC groups 中還要存放反向的 candidate，這樣遇到可以 merge 的狀況時就能反向再接過去，更進一步地簡化電路。

1.4 HashMap

主要是在 Strash 和 Simulate 的地方有用到，以下簡述：

Strash：HashKey 使用 AIG 兩個 input gate ID*2+1(if inverted)的數值的平方和(感覺可以使 Hash 中 element 分布的情形更平均，且符合交換率)，HashData 則是 CirGate*。

Simulate：HashKey 使用 AIG gate 做 simulation 之後的 32-bit Output Result，HashData 則使用 FEC Group 的指標，表示同個 simulation 結果相同的 Gates 會被歸類到同一個 FEC Group。

2. 演算法

2.1 CirSweep

更新 DFS List，並把所有在 List 上的 Gate 上色，最後跑一遍 Total List，將所有未上色的 AIG gate 移除。Map 的 traverse 需要 O(n) time，erase 則需要 O(lg n) time，所以在要 sweep 掉的 gate 不多的情況下，這個 command 接近 linear time。

2.2 CirOptimize

考慮 trivial optimize 的四個 case，按照 DFS 的順序，從 PI 處看看有沒有 Gate 可以消掉，或是跟 0 或!0 merge。之所以要按照順序從 PI 處開始，是因為前面如果有 Gate merge 在一起了，很可能會對後面那些 fanout 的 Gate 產生連鎖反應，所以不按照 DFS 的順序的話效能會差很多。因為 Optimize 就是 traverse 一次 DFS List，再從 map 中刪除一些 gate，所以時間應為 linear time 到 O(n lgn) time 之間。

2.3 CirStrash

按照 DFS List 的順序，從 PI 處把 AIG Gate 丟到 HashMap 裡面，並用 AIG Gate 兩個 fanin 的 literal 的平方和去分到不同的 bucket，如果找到 fanin 和自己相同的 AIG Gate 的話，就把這個 Gate merge 到 HashMap 中的那個 Gate，以此類推。之所以要按照這個順序，一樣是為了對後面的 Gates 造成連鎖反應，進一步簡化電路。因為有高機率在 constant time 就可以找到和自己相同 Fanin 的 AIG Gate，或者什麼也沒找到，因此這個步驟整體來說是 linear time 的。再加上 merging 的步驟，其複雜度和 Optimize 相同。

2.4 CirSimulate

Random：將 rnGen 回傳的亂數改為 unsigned，如此一來每次就可以拿到 32 組隨機的 input pattern，接著視 PI 的數量，不斷將隨機的 unsigned integer 丟到 PI Gate 的 simulation value。做完之後，就可以按照 DFS List 的順序，算出每個 AIG ouput 的結果，並存入 Gate 的 Output Result 中。此步驟後，每個 gate 所存的 simulation value 有 2^32 種可能性，數量大到一定程度，就可以丟入 HashMap 去整理成 FEC Groups 了。需要考慮的 Gates 只有存在於現有 FEC Groups 的 Gates。

對於每個 FEC Group 而言，我讓 bucket 的數目等於這個 group 中 Gate 的數量，如此平均而言每個 bucket 就會有一個 element。接著一一把 Gates 丟到 HashMap 裡面，一旦找到和自己 simulation 結果一樣的 gate，就加入和那個 gate 相同的 FEC group。等到所有 Gate 都丟完了，就可以把 HashMap 中 size 大於 1 的 FEC groups 撿起來放進新的 FEC groups。接著就可以考慮下一組 FEC Group 了，再重新建個 Map 去丟 Gates，以此類推。

這裡我將 Fail time 設為 90，也就是說如果一共有 90 次 simulate 完後，FEC group 數量都不變的話，就結束這個動作，避免無窮迴圈。

File：和 Random 的演算法相似，不過是等到蒐集滿 32 組 input pattern 後再做 simulation。時間複雜度應正比於目前 FEC Group 的大小。

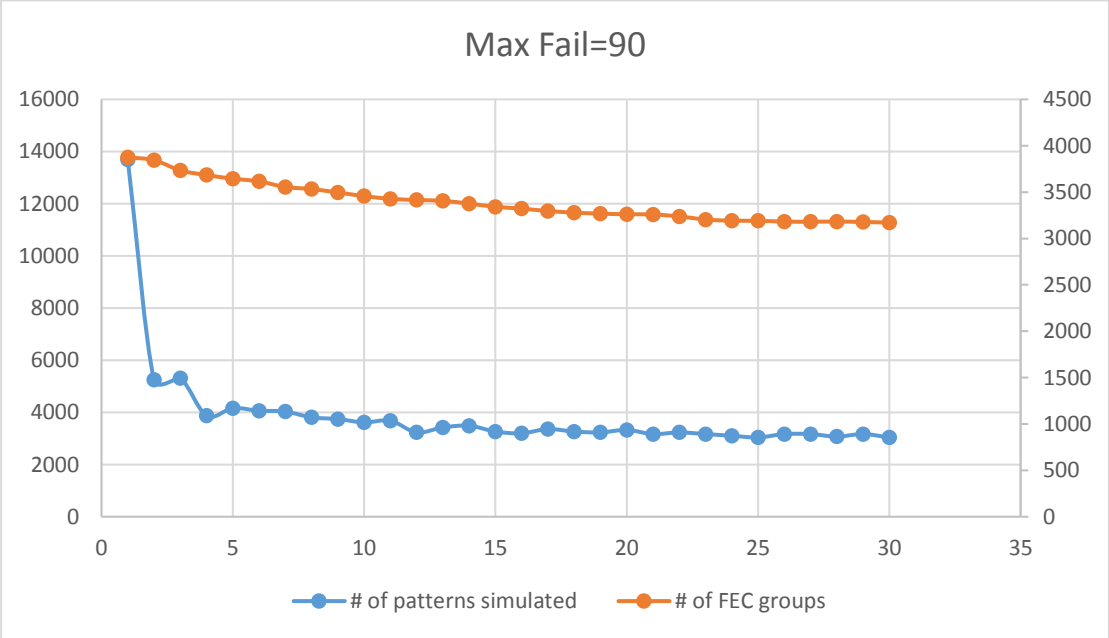
2.5 CirFraig

這裡主要是將 FEC Groups 丟到 SAT Solver 去驗證究竟這些 candidate 是否真的 functionally equivalent。如果是的話，就將兩個 Gate merge 在一起。按照老師的上課內容，如果兩個 Gate 不 equivalent 的話，可以使用反例的 input pattern 去進一步分解 FEC Group。但礙於我實作的程式效能並不佳，所以就沒有進行這個步驟了。我的方法是，在幫 SAT Solver 建好電路之後，對於每個已排序好的 FEC Group，都讓第一個 candidate 和其他所有 candidate 丟到 SAT Solver 去比較並 merge。如此一來，就不會比較太多次，才不會太花時間，而且也可以比較到 0 和其他 Gate merge 的狀況(case 較多)。所花時間正比於 FEC Groups 中 gate 的數量，但 SAT 是 NP-Complete 的問題，所以只要電路較大，整體花費時間相當長。

3. 實驗

3.1 Random Simulation

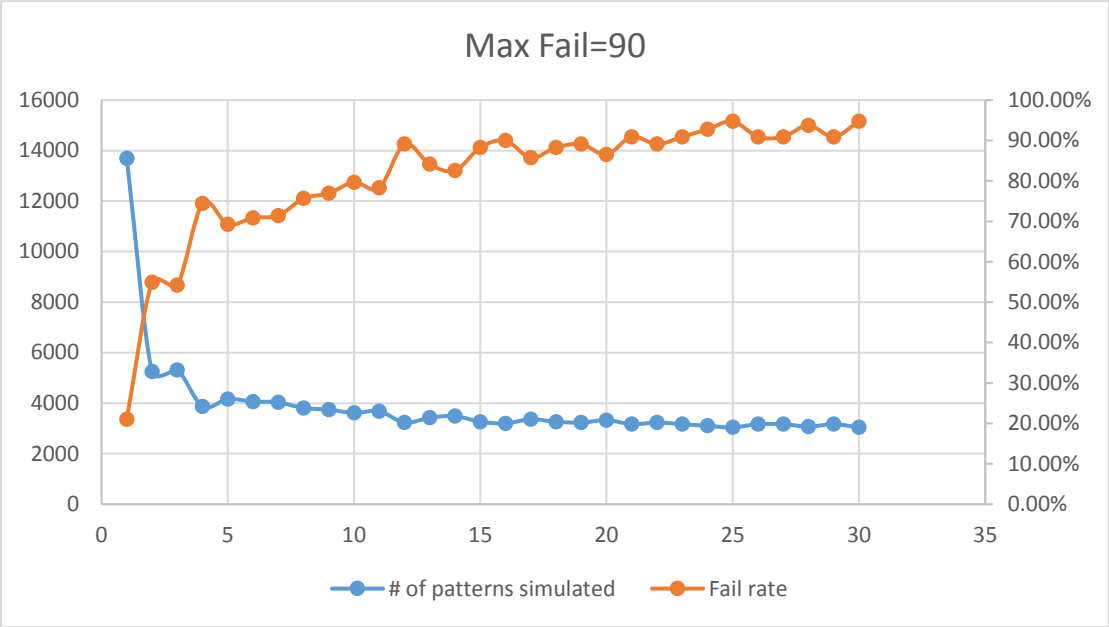
針對 sim13.aag，將 Max Fail 設在 90，做多次 random simulation，觀察嘗試的 patterns 數量和找到的 FEC groups 的數量隨時間的變化：



由實驗結果可知，在 Max Fail 固定的情況下，只有第一次嘗試的 pattern 數量較多。這跟直覺相當符合，因為第一次做 simulation 時，只有一個 FEC group 包含所有的 gates，所以 simulation 後完全無法使 FEC groups 數量改變的機率自然較低。

為了更進一步分析在 sim13.aag 中，究竟 Max Fail 這個參數的最佳數值為

何，因此我用上述數據去計算每一次 random simulation 的失敗率，也就是在 input 完一組 32-bit pattern 後，FEC groups 數量毫無改變的機率(Fail Rate=90/(# of patterns simulated/32))，得到如下結果：



由結果可知，Fail rate 在第一次 simulation 只有約 20%，第二和第三次上升到約 50%，第四次後則幾乎都在 70%以上。如果 simulation 可以分出新的 FEC group 的機率很低的話，我們基本上可以說，再繼續 simulation 的價值並不高。因此以 sim13.aag 的結果來說，Fail rate 的斜率大約在第四次後轉趨飽和，所以我們大概只要嘗試到 Max Fail=90*4=360 就好了，此數值為 Max Fail 的最適解。不過我寫的程式效能並沒有非常好，且沒有更多的 AAG 檔去驗證 Max Fail 和 PI 數量的關係，因此 Max Fail 就暫定在 90 for all cases。

3.2 Fraig

這個實驗是使用老師提供的 do.fraig，觀察我程式的結果和效能，結果大致如下：

1. 做完 cirr sim13.aag, cirp –n, cirp –fl, ciropt, cirsw, cirstrash 後：

Period time used = 0.7 seconds, Total memory used = 33.08 M Bytes

Total gates = 88410 (沒有任何變化，和 ref program 相同)

2. 做完 cirsim –r 後：

Period time used = 25.32 seconds, Total memory used = 984.5 M Bytes

FEC groups = 3874 (ref program: 3862)

3. 做完 cirfraig 後：

Period time used = 1109 seconds, Total memory used = 1015 M Bytes

Total gates = 85716 (減少了 2000 多個，ref program: 85574)

4. 做完 ciropt 後：

Total gates=85708 (略微減少)

5. 做完 cirsw 後：

Total gates=84702 (減少了 1000 個)

6. 程式最後：

Total time used = 1135 seconds, Total memory used = 1015 M Bytes

(ref program: Total time used = 135.3 seconds, Total memory used = 45.64 M Bytes, Total gates = 84013)

由上述結果可知，我的程式在時間和空間上都較差，但化簡到的 Gates 數目和 ref program 沒有相差太多。這表示，雖然我並沒有驗證每個 FEC pairs，但在這個演算法下仍可以相當有效率地化簡電路。