DSnP Final Project Report

電機三 B06901062 陳彥霖

手機：0912612181、信箱：B06901062@gmail.com

一、Data Structure

1.Gate:

Class Name	用途
CirGate	做 basic class 使用，處理 ID、SAT variable 等項目
PIGate	Derived class，對 PI 的 fanIn、fanOut 做特殊定義及處理
POGate	Derived class，對 PO 的 fanIn、fanOut 做特殊定義及處理
AIGGate	Derived class，對 AIG 的 fanIn、fanOut 做特殊定義及處理。同時也處理 Undefined gates

2.CirMgr:

Data member	用途
vector<int> PIList	依順序儲存所有的PI（以ID形式儲存）
vector<int> POList	依順序儲存所有的PO（以ID形式儲存）
vector<size_t> fullList	提供 ID -> CirGate* 的對應
vector<int> DFSList	依DFS順序儲存 Gate（以ID形式儲存）
vector<int> FloatList、vector<int> unUsedList	儲存 floating gate 和 unused gate（以ID形式儲存）
int MaxVariable、int AIGNum、int InputNum…	記錄當前各種 gate 的數量，以供輸出使用
static CirGate* _const0	將 CONST0 作 static data member 儲存
unordered_map<> hashMap、grpHash…	供 strash、simulation使用

二、Algorithms

（一）Simulation

1.File simulation:

(1) 讀取檔案：每次讀取一個字串。若字串長度和 PI 長度不合或有非0/1字元則跳出

(2) 分配數值：將每個0/1放到相對應的 PI 的 simValue 的某個 bit 中。

(3) simulation：當 PI 的 simValue 集滿64個 bit（或檔案輸入已結束），依 DFSList 的順序對每個 gate 做 simulation，以得到每個 gate 的 simValue。

(4) collection：對所有現存的 FECGroup（存在 vector<vector<int>*> simGrps中，每個 vector<int>*指向一個FECGroup），使用 unordered_map<size_t, vector<int>*> grpHash 來對該 FECGroup 裡的 gate進行新的分組。刪去舊的 FECGroup，加入新的 FECGroup。

(5) 重覆步驟 (1)-(4) 直到輸入完畢。

(6) sorting：對每一個 FECGroup 中的gate，用 gateID 進行排序。另外也針對 simGrps 中的 FECGroup做排序（利用其第一個 gate 的 ID）。

(7) assignment：在 simGrps 中，遍歷每一個 gate，將 gate 所在的 FECGroup 指標（為一個 vector<int>*）存進 gate 裡。

2. Random simulation:

(1) Random產生數值：對每一個 PI，用 rnGen() 產生一個 32bit 數值，將其向左平移32位後再加上第二個用 rnGen() 產生的 32bit 數值以形成 64bit 的 simValue。將其存進 PI 裡。

(2) 進行 simulation，同 file simulation 步驟 (3)

(3) 進行 collection，同 file simulation 步驟 (4)

(4) 重複步驟 (1)-(3) 至終止條件*。

(5) sorting：同 file simulation 步驟 (6)

(6) assignment：同 file simulation 步驟 (7)

*終止條件：定義一次 failure 為一次無法再分出新 FECGroup 的 simulation。無法再分出新的 FECGroup 可能是 Random 出來的 simValue太差，也可能是 gate 已經被分得很好。定義終止條件為累積 failure 達2000次。

（二）Fraig

1. Initialize solver
2. 將 CONST0 設為已走過，並 assign Variable。
3. 依 DFSList 順序遍歷每一個 gate。 設為已走過，並 assign Variable。
4. 承步驟 3 ，若正在遍歷的 gate 為 AIG，將其 AIGCNF 加入 solver 裡。
5. 承步驟 4，找到這個 gate 所在的 FECGroup，找到這個 FECGroup 中的第一個（加速1）、（加速2）已走過的 gate，以SAT solver 證明這兩個 gate 相等/不相等（加速3）
6. 承步驟 5，若結果證明兩個 gate 相等，將正在遍歷的 gate merge 進另一個 gate。若結果證明兩個 gate 不相等，則取得所有 PI 的 variable，將其放入 PI 的 simValue的某個 bit 中。
7. 承步驟 6，若 PI 累積滿 64個 bit，用這些 bit 進行 simulation。（加速4）
8. 回到步驟 3 ，繼續遍歷所有 gate。

- **加速1**：只找第一個已走過的 gate 作證明：理論上，若要全部證完，應該要跟同一個 FECGroup中所有已走過的 gate 都做證明。但因以下原因我決定只挑第一個 gate 證明：
    1. 若現在走過的 gate （以下稱 thisGate）和 FECGroup 中的第一個已走過的 gate（以下稱 mergeGate）相等，則將 thisGate merge 進 mergeGate 後，thisGate 被 delete本來就可以不用再證明剩下的 gate。
    2. 若 thisGate 和 mergeGate 不相等，則需要再找 FECGroup 的下一個 gate（下稱 nextGate），則可能有以下情況：
        a) nextGate 未走過。則又要再找下一個 gate。
        b) nextGate 已走過，但nextGate 和 thisGate 不相等。因已知這個 FECGroup至少有兩種不同的 gate（thisGate 和 mergeGate 已知不相等），因此這個情況的可能性 > 1/2。
        c) nextGate 已走過且和 thisGate 相等。

以上三種情形中，a 執行耗時，b 雖然可以讓我們得到一組 simValue，但因這組 simValue 很可能與先前證明 thisGate 和 mergeGate 得到的 simValue 相等（因都可以分開同一組 FECGroup），因此可能使之後的 simulation 效率低落。唯有 c 是最好的情況，但出現 c 的機率最低，因此我認為可以放棄證明 mergeGate 之後的 gate。

- **加速2：**觀察 simulation 的結果，常常可以發現許多 gate 和 CONST0 有相同的 simValue：

```
fraig> cirp -fec
[0] 0 6789 6840 6842 6845 8110 8242 8243 8304 8305 8363 8364
 8420 8421 8480 8481 8537 8538 8599 8600 8654 8655 8712 8713
 8769 8770 8834 8835 9419 9654 10431 10444 10919 10934 10997
 11144 12557 12732 12740 14001 14288 14417 14418 14484 14485
 14550 14551 14619 14620 14683 14684 14750 14751 14816 14817
 15472 15598 16379 16387 16399 16954 17237 22596 22597 22602
 22644 22668 22669 22747 22748 22818 22819 22839 22840 22847
 22912 22913 22920 22921 22948 22984 22985 22992 22993 23046
 23047 23053 23054 23062 23063 23123 23124 23130 23131 23139
 23140 23151 23187 23200 23201 23207 23208 23216 23220 23221
 23251 23274 23275 23281 23282 23290 23293 23296 23297 23327
 23329 23368 23372 23373 23380 23381 23393 23394 23406 23438
 23443 23444 23451 23452 23464 23465 23476 23516 23518 23522
 23523 23530 23531 23543 23544 23555 23557 23590 23594 23595
 23602 23603 23615 23616 23628 23662 23667 23668 23675 23676
```

我猜測這是因為 gate size 龐大時，離 PI 較遠的 gate 在前面有許多 AND 運算的情況下很難出現 True 的答案，這代表了這些 gate 有一定的機率和 CONST0 相等。但若和 CONST0 不相等，這些 gate 兩兩之間也很有高機率會不相等。因此在執行這組的證明時，只需證明每個 gate 和 CONST0是否相等就好。

- **加速3**：承加速 2 ，要證明每個 gate 和 CONST0 是否會相等，使用類似前面 OPTIMIZE 的手法就可以得到不錯的結果，速度也比較快。因此在此特殊情況下不使用 solver。

- **加速4**：對於三種 FECGroup，我們可以不用再做重新分配：
  1. 包含 CONST0 的 FECGroup：承加速2,3，這組的重點在於證明每個 gate 和 CONST0是否相等而非 gate 兩兩之間的關係，因此不用重分配。
  2. 被 merge 到只剩一個 gate的 FECGroup：可直接刪除。
  3. 全部走完的 group：這種 group 已完成證明。可直接刪除。

三、Experiment

限制 random simulation failure 次數對 FECGroup 和 執行時間的影響
(以sim13.aag測試)

Failure次數	500	1000	2000	3000	3500
FECGroup number	3296	3150	3030	3010	3003
Simulation runtime(sec)	5.62	8.2	12.06	17.64	18.12
Fraig runtime(sec)	28.96	26.07	23.33	23.72	22.28

從上表可以看到，在 2000 次failure以前，做越多 simulation，FECGroup的數量便可以有顯著的下降，fraig的速度也會有明顯提升。
但在 2000 次 failure 以後，再做更多simulation也不會使得表現更好，只會浪費時間在simulation上。
因此將終止條件設為 2000次 failure。