DSnP final project – FRAIG report

name：林秀銓
school ID : B04505028
contact information(e-mail) : B04505028@ntu.edu.tw

design of data structure

1.CirGate

首先，我運用繼承的方式來處理不同型態的gate(e.g. Const gate,PI gate, AIG gate)，而每個gate都用CirGateV 存取兩個 fanin，fanout 則用 vector<CirGateV> 來存取 ，另外還有一個size_t class的值來存取simulate時產生的 pattern 和 gate 最後的結果。

2.CirMgr

在Mgr當中，就利用_vAllGates 將所有的PI，PO 和 AIG 依照gate ID的順序存入，再將每種特定的gate 分類存取，如_vPI，_vPO 和 _vAIG，以方便之後的操作，另外還有一個vector 來存取 DFS 的結果，並用vector<CirGate*>*的vector將FEC group 收集起來(每個vector<CirGate*>即為一個 group)。

My design of the functions

1.CirSweep

首先，將每個gate掃過一遍，若其沒有fanout且是AIG gate 則此 gate 為unused，可直接移除，再來不斷對他的 fanin進行recursuve，直到沒有gate的fanout為0，即沒有gate為unused，就完成了sweep。

2.CirOptimize

先寫好gate的optimize function，檢查每個 AIG gate 的兩個 fanin 是否符合那四種trivial optimization的情況，若是接到Const 0或是有完全相反的fanin，就直接跟Const 0取代。若是接到Const 1或是完全相同的fanin，則利用其中一個 fanin 取代。比較難處理的地方在於被影響到的gate，像是要被刪掉的gate的fanin和fanout，都要處理好它們的fanint和fanout，必須確保erase掉被刪除的gate，以及接好fanout和fanin，須小心處理。因為將gate merge後有可能影響到其他的gate，所以為了要確定何時可以停止操作，所以要有一個check儲存這次merge完的gate數量，若下次操作完的gate與這次一樣，則代表所有的gate都已經是不可再merge的狀態，所以可以停止optimize。

3.CirStrash

3.1 Design of HashKey

一開始，先開一個<HashKey, CirGate*>的HashMap。而HashKey設計的方式為存入gate的2個 fanin，利用operator () 去產生Hash value，用其fanin的gate literal(i.e

(size_t)2*gate ID + phase)，。而Hash fucntion產生的方式就是將兩個fanin的gate literal值作 OR bitwise operation (i0 ^ i1)。

3.2 Comparison of HashKey

利用opetator == 去比較兩者是否有相同的HashKey。若其兩個fanin(i.e. in0, in1)的value相同，或是與其counterpart相同(i.e. i0== i1 && i1 == i0)，就回傳true。

3.3 Method of Strashing

利用一邊query一邊建立HashMap的方式去進行。首先用query將一個key對應到的CirGate*找出來，若query的結果是false，則將那個
HashNode(pair <HashKey, CirGate>)塞入HashMap，反之則直接用找出來的gate取代現在的gate。取代的方法與optimize的函數差不多，所以實做方面很快。

4.CirSimulation

4.1 Method of Gate Simulation

首先先將產生出來的pattern填入進每個PI的_simPattern，在這裡利用size_t去存，就能進行parallel simulation。再來，跑過_dfsList內的每個gate，若是Const 0或是UNDEF gate，則將其_simPattern設成0。若是AIG gate，則直接看其fanin的_simpattern以及是否inverting，再作and bitwise operation(i.e. i0 & i1)，便能得到其gate的_simPattern。最後，若是PO gate，則直接看其單一fanin (fanin0)的_simPattern便能得到。

4.2 Method of FEC groping

知道每個gate的sim value之後，就能來分出新的FEC group了。

4.2.1 Initialization

開始先將所有AIG和Const 放入同一個fecGrp內，再把這個group放入fecGrpList

4.2.2 Design of SimKey

SimKey的設計方式與HashKey差不多。存的就是gate的_simPattern，並且利用operator () 當作Hash function (直接回傳_simPattern)，operator == 來比較SimKey。而comparison則是要看兩個Simkey是完全一樣(_key1 == _key2)，或完全相反(_key1 == ~_key2)，這兩種情形都是同一種FEC Group，就回傳true，如此一來就能解決sim value完全相反的問題。

4.2.3 Algorithm

首先，對每個在 fecGrpList 內的fecgrp都開一個<SimKey,vector<CirGate*>*>的HashMap，因為hash中bucket的數目太多或太少都會讓效能變差，因此在這裡bucket數量大概是現在fecgrp的數量。再來，query每個在fecgrp內的gate是否在HashMap內，若沒有則插入HashMap內，反之則push_back到query出來的fecgrp。

每跑完一次fecgrp，則將HashMap分出來的那些fecgr放入到一個temp fecgrps內，這樣做的原因是避免利用速度很慢的erase，等到最後直接將fecGrpList.clear()後，用這個tmp取代就好。

最後，跑完整個流程後，再將這個新的fecgrps排序，並讓裡面所有fecgrp的所有gate都紀錄自己所在fecGrp的位址，這樣作的原因是若直接記fecgrp會造成非常大的overhead。

4.3 Method of Random Simulation

4.3.1 Generation of pattern

首先，利用util內的rnGen產生一個亂數的pattern。不過要注意的是，產生出來的亂數是unsigned int，所以必須shift 32位數和shift前的數做 OR operation，這樣產生出來的pattern才會有足夠的長度。

4.3.2 stopping criteria

設定一個max標準和條件，若達成則count++， count達到max 則停止simulation。次數則隨電路之複雜度不同而相異，這邊邊我用的公式為

```
max = (_vDfsList.size() < 100) ? _vDfsList.size() : sqrt(_vDfsList.size()) * 5
```

4.4 Method of File Simulation<註3>

此function 與 random simulation的操作差不多，只是Patten不再由亂數產生，而是讀取檔案裡的資料。

5.CirFraig

未完成

Experiment

1.File Simulation

1.1 result

my fileSim of sim13.aag : 21.86 s
ref code fileSim of sim13.aag: 3.25 s

1.2 analysis

可能的bottle neck出現在assign fecgroup id的部份，若直接讓gate記整個FEC group就不用去iterate每個gate，也就不會有這個bottle neck，不過會佔用很多記憶體，這也算是trade-off的一環。又或者是hash中的 bucket 數量太多或太少，造成操作hash 時作態多次 iterator。

2.Random Simulation

2.1 result

my randomSim of sim13.aag : 21.9 seconds , generating 95104 patterns , and 3490 FEC grps
ref code randomSim of sim13.aag: 5.69 seconds ,generate 95296 patterns, and 3525 FEC grps

2.2 analysis

我覺得原因跟 File Simulation 差不多。