DSnP Final Project:
Fraig Report

B06705033 資管三 莊海因
E-mail: b06705033@ntu.edu.tw

一、前言：

此作業使用去年修的同學(b06705028)的hw6架構，以及自己hw7的myHashSet.h來完成。

二、架構：

- **CirGate**

```
class CirGate
{
    unsigned  _id;                      // variable ID of gate
    unsigned  _mark;                    // to dfsTraverse
    unsigned  _fanin_idx[2];            // fanin的variable ID
    bool      _fanin_invert[2];         // fanin的invert性質
    size_t    _simVal = 0;              // simulate value
    int  _grpN = -1;                    // belonging group number
    CirGate*  _fanin[2];                // fanin gates
    GateList  _fanout;                  // fanout gates
}
```

繼承CirGate：

```
class AigGate;
class PiGate;
class PoGate;
class ConstGate;
class UndefGate;
```

- **CirMgr**

```
class CirMgr
{
    HashSet<GateNode> _gateHash;          // for Strash
    HashSet<FECnode> _FECHash;            // for Simulate
    GateList          _gates;             // list of all gates
    GateList          _dfsList;           // list of DFS
    vector<PoGate*>   _pos;               // list of PO gates
    vector<unsigned>  _pis_idx;           // list of PI gates' ID
    vector<vector<FECnode>> _FECgroup;    // FEC(IFEC) groups
    unsigned m, i, l, o, a;               // read from aag file
}
```

三、功能實作：

- **Sweeping**

Sweep要移除無法從PO觸及的gate，也就是不在_dfsList中的gate。因此先以dfsTraversal()更新_dfsList後，挑選出不在_dfsList中的AigGate或UndefGate（PI不刪除，PO不可能）。但若要比對_gates與_dfsList中所存在的gate，直觀來說要跑兩個for迴圈O(n^2)較沒有效率，因此以for迴圈跑一遍_dfsList標記有在裡面的gate，再用for迴圈跑一遍_gates尋找沒有被標記到的gate。

接著並以rmFromFanin()與rmFromFanout()刪除_fanin、_fanout與本身的連結後，從_gates中刪除此gate。

```
CirGate::rmFromFanin(){ //remove the gate's _fanin's _fanout list
if there's _fanin
      for _fanin's _fanout list
            if _fanout[i] is this gate
                  erase this gate from _fanout list vector
}

CirGate::rmFromFanout(){ //remove the gate from its _fanouts' _fanin
for this gate's _fanout list
      if _fanout's one _fanin is this gate
            set the _fanin the NULL
}
```

- **Optimization**

更新_dfsList後，跑for迴圈，只針對AigGate需要處理四種情況，而四種具有重複滿足的可能，與ref執行結果比較後發現(c).(d)的考慮順序在(a).(b)之前。在這裡使用兩個function來執行以_fanin取代或以CONST0取代，並且配合上部分的rmFromFanin()、rmFromFanout()來使用。最後從_gates中刪除被取代的gate，並且再次更新_dfsList。

```
CirGate::replaceFanin(int x){  //replace this gate with _fanin[x]
rmFromFanin()
for this gate's _fanout list
      if _fanout's one _fanin is this gate
            set the _fanin = _fanin[x]
            add _fanout to _fanin[x]'s _fanout list
}
```

```
CirGate::replaceConst(ConstGate){  //replace this gate with _fanin[x]
rmFromFanin()
for this gate's _fanout list
     if _fanout's one _fanin is this gate
          set the _fanin = ConstGate
          add _fanout to ConstGate's _fanout list
}
```

- **Strash**

此功能所使用的是hw7的HashSet.h，參考hw7中TaskNode的作法，自己定義了class GateNode，並以literal ID來比較兩個GateNode的兩_fanin是否相等（相當於structurally equivalent）。在class CirMgr中定義了HashSet<GateNode> _gateHash，儲存著不重複(structurally equivalent)的gate。實作方式為：
(1)在_dfsList的for迴圈之中只看存在此種情況的AIGgate，為每個gate建立GateNode
(2)以此GateNode進行query：

若_gateHash中已經有重複的GateNode，則使用mergeGate()將已儲存於_gateHash中的GateNode取代尚未放入_gateHash的GateNode，並於_gates中刪除被取代的gate。

若沒有找到重複的，便將此GateNode insert進入_gateHash中。
(3)以_dfsTraversal()更新_dfsList

```
class GateNode{
    size_t operator()() const { return (size_t)(_fanin0*_fanin1); }
    bool operator == (const GateNode& k) const {
        (k._fanin0 == _fanin0) && (k._fanin1 == _fanin1)
                               or
         (k._fanin0 == _fanin1) && (k._fanin1 == _fanin0)
    }
    CirGate* _gate;
    unsigned _fanin0; //literal ID
    unsigned _fanin1; //literal ID
}
```

判斷是否重複：

```
if(_gateHash.query(node)){
    _dfsList[i]->mergeGate(node.getG()); //replace this with hash's
CirGate* _gate
    _gates[_dfsList[i]->_id] = NULL; //delete from gate list
}
else _gateHash.insert(node)
```

以gate* g將此gate合併：

```
CirGate::mergeGate(CirGate* g){
rmFromFanin()
for this gate's _fanout list
      if _fanout's one _fanin is this gate
            set the _fanin = g
            add _fanout to g's _fanout list

}
```

- **Simulation**

以setPI()或simTraveral()將所有gate的_simVal設定好之後，使用setFEC()或updateFEC()將AIG gate分組。這裡也是參考hw7中TaskNode的作法，自己定義了class FECnode，並以_simVal判斷兩個gate是否為FEC/IFEC pair。在class CirMgr中定義了HashSet<FECnode> _FECHash，將屬於不同組的gate存放在HashSet中的不同_buckets裡。最後每次都將更新後的_FECHash分組一組組放入vector<vector<FECnode>> _FECgroup之中。實作方式為：

(1)在_dfsList的for迴圈之中只針對AIGgate，為每個gate建立FECnode

(2)以此FECnode進行check()：

如果找到相同的_simVal表示為FEC pair，insert進入_FECHash中。

如果沒找到，則以inverse _simVal找，若找到相同表示為IFEC pair，同樣insert進入_FECHash中。

若兩種方式都沒有找到，可能是這個gate沒有pair，或者還沒insert進入_FECHash，因此仍然將之insert。

(3)將_FECHash中有儲存且大於一個FECnode的buckets(一個bucket為一組，至少要有兩個才稱為一組)push_back進入_FECgroup之中。

如果不是第一次，則只針對_FECgroup中的每一組個別更新。

fileSim：

```
while(patternFile >> line){
        lineCnt++;
        checkPatten(line);
        simPI(line); //設定PI gate的_simVal
        if(lineCnt % 64 == 0){ //每64行simulate一次
            simTraversal();    //由PI開始設定所有gate的_simVal
            if(FECset)  updateFEC(); //非首次分組
            else    setFEC();        //首次分出_FECgroup
            clearPI(); //初始化PI的_simVal
        }
}
```

randomSim：

notchange為連續不再產生新的group的次數，為與ref比較之後之判斷。

```
max = sqrt(AIG+PI+PO);
if(max>20){
    if(sqrt(max)/4 > 20)
        max = sqrt(max)/4;
    else max = 20;
}

while(notchange <= max){  //連續不再產生新group的次數超過max，即停止
    for(unsigned i = 0; i < 64; i++){
        for(unsigned j = 0; j < _pis_idx.size(); j++){
            setPI(rnGen(2));
            }            //為PI產生隨機的simulate number
        }
    simTraversal(); //由PI開始設定所有gate的_simVal
    if(FECset)  updateFEC(); //非首次分組
    else    setFEC();        //首次分出_FECgroup
    //更新連續不產生新分組次數
    if(_FECgroup.size() == old){
        notchange++;
    }
    else{
        notchange = 0;
        old = _FECgroup.size();
        }
}
```

- **Fraig**

沒有做到這一步QQ

四、心得：

這學期慕名而來修課，果真不負眾望（？）花費很多時間在寫作業，大概用了九學分的時間在修三學分的課XD。不過也的確受益良多，在每兩週一次的作業訓練下，覺得自己有把大二修資料結構的coding能力撿回來，不只是重拾許多已經忘記的概念，也更深入了解了資料結構的世界（好恐怖x）。感謝教授與助教這個學期的辛勞，有上到最後一年的DSnP深感值得！