# DSnP Final Project Report : FRAIG

姓名：林承德

系籍：電機二

學號：B05901064

E-mail：b05901064@ntu.edu.tw

# 一、 Class 定義及其功用

在這一段落中主要是講解 class 裡面幾個重要或特別的 data member 及 memner function 的功能及用處。有些有關 Command 的方面會移到下一段落介紹。

## A. CirGate 及繼承 CirGate 的 *Gate

在設計 CirGate 時，我只有將大家都需要的變數（_gateNo、_flag…等）放在 CirGate 中，其他變數（_fanout、_fanin…等）則是宣告在各自的繼承 class，可以節省記憶體使用量。

另外我在 CirGate 寫了許多存取繼承 class 變數的 vurtual function（getFanin()、getFanout()…等），可以在 CirGate 讀取到那些變數，方便在 CirGate 的 member function 寫完大家都要做的事情（printGate()、reportFanin()…等），方便之後的維護。

### i. size_t CirGate::_flag

此變數有兩種功能，一個是在跑 DFS 時與 CirMgr::_flag 比較確認是否跑過一次，另一個是在 FecPair::removeOutDfs()時確認 gate 是否在 dfs 的範圍中。

### ii. size_t CirGate::_dfsPos

在每次 cirMgr 執行 setDfs()更新 DFS 順序時會將此 gate 在 vector 的第幾個位置寫在此，供 fraig 時判斷兩個 gate 間是誰要 merge 誰。

### iii. static const vector<unsigned> CirGate::nullVector;
### static const string CirGate::nullString;
### virtual inline const vector<unsigned> Cirgate::getFanin() const
### virtual inline const vector<unsigned>& Cirgate::getFanout() const
### virtual inline const string& CirGate::getName() const

為求效率，我盡量讓 string 及 vector 的 getter function 是 pass by reference。但有些 derive class 並沒有 string 或是 vector 可以回傳，故需要 static 的 nullVector / nullString 供他們回傳。

另外，由於 fanin 並不是以 vector 的方式儲存，故沒有 reference 可以回傳，只能用 pass by value 多複製一次的方式回傳。

### iv. FecPair* AigGate::_fec
### FecPair* ConstGate::_fec

指向含有該 Gate 的 FecPair，讓 gate 在尋找其 FecPair 的速度降為 O(1)。

**v. void CirGate::connectGate(CirGate**)**

在 cirMgr::readCircuit()時用來連結 gate 與 gate 之間，設定所有 gate 的 fanout。如果有一個 gate 的 fanin 未被定義，則會在傳入參數 CirGate*的 array 相對應位置創造一個 undefine gate。

**vi. vector<unsigned> CirGate::remove()**
**vector<unsigned> CirGate::replace(unsigned var)**

此 function 只會當作 cirMgr::removeGate(unsigned, vector<unsigned>)的傳入參數。這兩個 function 會把 gate 的 fanin / fanout 相對應關係進行調整，調整完之後可以直接讓 cirMgr delete 這個 gate。

Function 回傳它的 fanin 給 cirMgr，讓他去檢查 fanin 是不是變成 defined but not used，如果是的話就將其加入 cirMgr::_notUsed。

## B. CirMgr

**i. CirGate** CirMgr::_gates**

儲存所有 CirGate* 的 array，以 hash 的概念儲存，第 i 號 gate 就放在 _gates[i]的位置（i.e. hash key = gate Number），讓 getGate(unsigned)速度壓在 O(1)。這樣可以讓其他地方只要儲存 gate number 或是 literal ID 就好，節省空間使用量。

**ii. Vector<unsigned> CirMgr::_notUsed**

存放 defined but not used gate 的 gate number。這樣在 sweep 時可以不用 run 全部的 gate，在 cirprint –fl 時也不用重新跑一遍。

**iii. Vector<FecPair*>* CirMgr::_fec**

存放所有的 FecPair。平時並不會對其資料進行排序，只有 cirp –fec 時才會進行排序。

會用 pointer 的原因是我的 FecPair 在 simulate 完生成下一代 FecPair 時需要將新的一代放進新的 vector，再將其取代原本的 vector。為了不要進行複製消耗時間，我將其令成 pointer 讓他可以不用複製直接使用。

**iv. size_t CirMgr::_flag**

與 CirGate::_flag 一樣，用來確認 gate 是否已經跑過 DFS，以及在 FecPair 中確認 gate 有沒有在 DFS 的順序中。

**v. void inline CirMgr::reportFanin(int, CirGate*)**
**void inline CirMgr::reportFanout(int, CirGate*)**

由於我的 flag 標準放在 CirMgr 中，故我讓 Cirgate –fani/fano function 改成呼叫 cirMgr，先++CirMgr::_flag 再跑 CirGate::reportFanin/reportFanout 確認其是否已經跑過。

**vi. void CirMgr::setDfs()**

此 function 會重新跑一次 DFS，更新 CirMgr::_dfs 及 CirGate::_dfsPos。如果有 Fec Group 存在則讓他們確認自己 group 內的 gate 還在不在 DFS 順序裡，不在就拿掉。如果刪完有一個 FEC 只剩下一個 gate 或是沒有 gate，則將其刪除。

**vii. void CriMgr::removeGate(unsigned, vector<unsigned>)**

此 function 的 vector<unsigned>參數必是 CirGate::remove()或是 CirGate::replace(unsigned)，先讓 CirGate 處理好 gate 之間的交互關係，再 delete gate 本身，實現 OO 的精神。

## C. FecPair

**i. unordered_set<unsigned>* FecPair::_hash**

主要用來儲存 gate number 的 container。用 hash 的好處是在 insert、find 和 erase 都可以做到 O(1)的速度。用 pointer 的原因是要避免一直複製，拖垮效率。

**ii. vector<unsigned> FecPair::_vec**
**bool FecPair::_isVecSet**

_vec 是主要是用來 print 的 container。平常完全不維護也不放入資料，只有要 print 時才會從 hash 複製一份資料過來並排序。_isVecSet 則是記錄_vec 是否是設定好的狀態。

**iii. int FecPair::_min**
**int FecPair::_dfsMIn**

用來記錄 gate number 最小值以及 dfs 最前面的 gate 的 number，每次 insert 都會檢查這個質是否需要修改。如果最小值被移除則會讓其值變為-1，等待下次需要或是檢查 gate 是否在 DFS 中時重新搜尋。

**iv. void FecPair ::generate(vector<FecPair*>*)**
**bool FecPair ::_first**
**void FecPair::generateFirst**
**(vector<FecPair*>*, unordered_map<size_t, FecPair*>&)**
**void FecPair::generate**
**(vector<FecPair*>*, unordered_map<size_t, FecPair*>&)**

FecPair 用來產出下一代 Fecpair 並推到新的 vector<FecPair*>*中。利用 hash

把具有相同 value 的 gate 加到同一個 FecPair。

_first 則是記錄這是否一開始初始化的 FecPair，如果是的話要執行 gererateFirst 將具有相反 value 的 FecPair 合在一起。

# 二、 Commands 實作方法及細節

## A. CirSweep

```
size_t pos = 0;
while(pos != _notUsed.size() ){
  CirGate *g = getGate(_notUsed[pos]);

  if(g->isPI() )++pos;//Not to sweep PI
  else {
    //delete the gate
    //Note: if the fanin in *g will be not used after deleting *g
    //      it will be added to _notUsed by removeGate()
    removeGate(_notUsed[pos], g->remove() );

    //remove *g from _notUsed
    _notUsed[pos] = _notUsed.back();
    _notUsed.pop_back();
  }
}
```

_notUsed 會在 readCircuit()時一起建好，又在刪除 gate 時如果產生新的 defined but of used gate 會被 removeGate()自動放入，故只要一直針對_notUsed 刪到剩下 PI 就算結束了。

它的好處是不用跑完全部所有的 gate 就可以完成，對於大量的 AIG 的 circuit 有所幫助。

## B. CirOptimize

```
for(auto &i : _dfs){
  Cirgate *g = getGate(i);

  //return -1 if *g can't be optimize
  //else return the literal ID which can replace *g
  int replaceLiteralID = g->optimize();

  if(replaceLiteralID != -1)removeGate(i,g->replace(replaceLiteralID) );
}
```

CirGate ::optimize()是一個 virtual function，除了 Aig 以外的 gate 都會 return -1，Aig 則是會藉由他的兩個 fanin 判斷它是否可以被其他 literal Id 代表的 value 取代。

由於一個 gate 經由 optimize 被 merge 後只會影響其 fanout optimize 的結

果，故我照 DFS 順序進行簡化，避免遞迴呼叫拖慢速度。

## C. CirStrash

```
//the key is the combination of the two fanins of Aig
//the value is the gate number of Aig
unordered_map<size_t, unsigned> map;

for(auto &i :_dfs){
  CirGate *g = getGate(i);
  if(!g->isAig() )continue;

  //key from *g
  //M is the the first number of MILOA
  size_t key = ((AigGate*)g)->getStrashKey(M);
  unordered_map<size_t, unsigned>::iterator pos = map.find(key);

  if(pos == map.end()){
    //there is no gate can merge so far.
    map.insert(strashData(key, i) );
  }else{
    //already exists a gate
    removeGate(i, g->replace((pos->second)*2) );
  }
}

inline size_t AigGate::getStrashKey(const unsigned M) const {
  return _fanin1>_fanin2 ? (size_t)_fanin1 *(size_t)(_fanin1 +1)/2 +_fanin2 :
                           (size_t)_fanin2 *(size_t)(_fanin2 +1)/2 +_fanin1 ;
}
```

藉由 hash 的幫助，我可以在 O(1)的速度下找到是否有 gate 可以跟自己合併。

由於一個 gate 經由 optimize 被 merge 後只會影響其 fanout optimize 的結果，故我照 DFS 順序進行簡化，避免遞迴呼叫拖慢速度。

在 hash key 部分，雖然運算速度比直接把兩個 fanin 前後接在一起還慢，但我的 key 可以妥善運用 size_t 的每一個 bit，可以表現出 $2^{64}$ 不同種 fanin 的組合狀況。

## D. CirSimulate

```
void CirMgr::simulate(vector<size_t>& data) {
  //set the input to PI
  for(size_t i = 0; i < _inputs.size(); ++i)
    ((PIGate*)getGate(_inputs.at(i) ) )->setInput(data.at(i) );
  //run the simulation
  for(auto& i : _dfs)getGate(i)->simulate();

  if(!_fec){//fecPair doesn't exist!
    _fec = new vector<FecPair*>;
    FecPair* group = new FecPair(0);
    //create the first fecPair
    for(auto &i : _dfs)
      if(getGate(i)->isAig() )group->insert(i*2);
    //generate the next fecGroup
    group->generate(_fec);
    delete group;
  }else{
    //create new vector
    vector<FecPair* >* temp = new vector<FecPair*>;
    //generate all fecPair
    for(auto &i : *_fec){ i->generate(temp); delete i; }
    //change _fec
    delete _fec;
    _fec = temp;
  }
}
```

data 是一個 size 跟 PI 數量一樣多的 vector，data[i]是在 simulation 時給第 i 個 PI 的值，randomSim()和 fileSim(ifstream&)皆是用他們自己的方法生出 data 並呼叫此 function。

CirGate ::simulate()是每個 gate 根據其 fanin 現在的值決定自己的值，只會影響到 fanout 的反應。故我在給完所以 PI 值之後，照 DFS 的順序跑 simulate，避免遞迴呼叫。

在 FEC 的部分，如果還沒有產生的話，就先創一個 fecPair 把所有在 DFS 的 AigGate 和 Const0 丟進去，再針對那個新的 fecPair 進行 generate。如果已經存在 fecPair，就把每一個 fecPair 都拿去 generate。最後拿到新的 vector<fecPair*>*再取代原本的_fec。

### E. CirFraig

```
while(!_fec->empty() ){
  for(auto &i : _dfs){
    if(!getGate(i)->isAig() )continue;
    FecPair *group = getGate(i)->getFecPair();
    if(!group)continue;//the aig doesn't have fecPair

    unsigned var = group->getMinDfs();//the gate in fecPair which is the most front in dfs
    if(i == var/2)continue;// the gate i has the min dfsPos
    //const0
    if(var/2 == 0){
      if(fraigConst(group->isInverse(0) == group->isInverse(i) ? 2*i : 2*i +1) )break;
      else continue;
    }
    //anothor aig
    if(cirMgr->getGate(i)->getDfsPos() < cirMgr->getGate(var/2)->getDfsPos() )continue;
    if(fraigAigs(2*i,(group->isInverse(i)==group->isInverse(var/2) ? (var/2)*2 : (var/2)*2)+1 ))break;
  }
}
```

我的 fraig 演算法主要概念就是拿兩個 gate 在 DFS List 裡面越前面的來用 SAT 證，因為 SAT 證明的複雜度遠大於其他動作，故我認為盡量減低 SAT 的 input 比較重要。DFS List 在越前面的話，他 fanin 直到 PI 間有比較少的 gate，SAT 證明會比較快。另外，如果 fecPair 裡面有 Const0 的話要優先拿出來處理，因為 0 跟 gate 進行 XOR 就是 gate 本身，不用多花 gate 數量去建造 Const0。

如果 SAT 證明出來他們兩個是一樣的，就依照 DFS 順序，在前面的並調在後面的，減少電路複雜度也避免迴圈產生。之後重新設定 DFS list，再跑一次 fraig。如果 SAT 證明他們不一樣，就直接拿那個解進行 simulation 把那兩個 gate 分開，然後繼續照著 dfs list 繼續跑下去。一直跑到*_fec 這個 vector 變成空的為止。

## 三、 Performance

簡單來說，我的 fecPair 在處理 generate 的時候花太長的時間了。

ref 在我的電腦上 sim13.aag 中 simulate pattern.13 花了 1.7 秒左右，而我的 fraig 需要跑到三秒多才結束。但是當我把處理 fecPair 這段 code 拿掉後，我只需要 0.56 秒就 simulate 完 pattern.13 了。fecPair 佔 simulate 超過八成，效率不佳。而我的 fraig 每次只要證明出兩個 gate 不一樣時就需要 simulate 一次、fecPair generate 一次，這樣累積下來可能導致拖慢 fraig 的時間。

另外，我原本假設 DFS List 越前面的 gate『大致上』越好證，因為他被期望在證明時有比較少的 gate 需要建立。但實際上有時候非常不準，也會影響到我 fraig 的效率。

我自己目前想到的改善方法有幾個。一個是想辦法加速 fecPair 的速度，看是要減少 new delete 的次數或是用記憶體來換取更多時間；一個是在 fraig 時不要拿到一組 data 就 simulate 一次，可以等拿到多一點 key 再來 simulate，減少 fecPair generate 的次數；或者是改變 fraig 的策略，變成從 PI 開始往 fanout 出發……等。

# 四、 Obstable

## A. cirFraig 策略

我原本是設計另外一種 fraig 的演算法（在 cirFraig.cpp #ifdef FRAIG_MY_VER）。他的前提是 SAT 的證明速度只與 SAT 裡的 PI 數量有關，以及證明 DFS list 越後面的 gate 效果越大。但他在處理 sim13.aag 兩倍大的電路時直接卡住，在前 10 個 gate 就證明了 10 分鐘還證不完。經過開 gdb 後，我認為 SAT 的證明速度可能不是跟 PI 有關，而是跟全部 gate 的數量有關。故我才改成上面那種演算法。

## B. aag file 裡的電路型態

不管是 optimize、strash、還是 fraig，每一種演算法都可以處理某一種形狀的電路，卻在另一種電路中表現十分低落。在測資有限的情況下，這讓我感到十分猶豫，不知道哪一種作法才是平均來說最好的。

# 五、 心得

這份 final project 給我們很大的自由度，讓我可以自由的設計我的架構，但也是這樣才會令人更頭痛。我必須一直思考架構好不好寫、效率如何、占用的記憶體多少、好不好維護.....等問題，這些也是我之前都沒有認真想過的問題，真的是要發揮腦袋的極致才能打出一份讓自己滿意的 code。

這堂課讓我對於電腦和 C++有更進一步的了解，透過一次次的作業的訓練讓我們知道要怎樣才可以打出一份好的 code，而不是可以跑就好。我在打 final 的時候把 hw6 的部分重新打了一次，邊打遍覺得自己之前怎麼可以打出這麼奇怪的 code，想必是有多一點成長了吧。

謝謝教授及助教給我這個機會練習我的技巧，我在這堂課上的非常開心 XD。