Final project report

Functionally Reduced And-Inverter Graph (FRAIG)

Course: NTU EE3011 Data Structure and Programming (DSnP)
Teacher: Ric Huang
Name: Shannon Lee (李尚倫)
ID: r07921001
Email: r07921001@ntu.edu.tw
Full code: https://github.com/shannon112/DSnPorygon/tree/master/fraig
Course website: https://github.com/ric2k1/DSnP.open

I. Introduction

Functionally Reduced And-Inverter Graph (FRAIG) 是 Electronic Design Automation (EDA)的一環，主要功能是化簡重複功能的電路片段，讓整個電路變得更精簡，但保有和原本一樣的功能。可以從三個方面來看他的好處：

1.Area
- Reduce the number of gates
- Moreover, using library cells of smaller sizes, but they will have weaker driving capability

2. Timing
- Shorten the longest path
- Additionally, insert buffers and/or enlarge the cells to increase the driving capability

3. Power
- Reduce the switching activities
- Moreover, shutdown the sub-circuit that is not currently used

II. Problem description

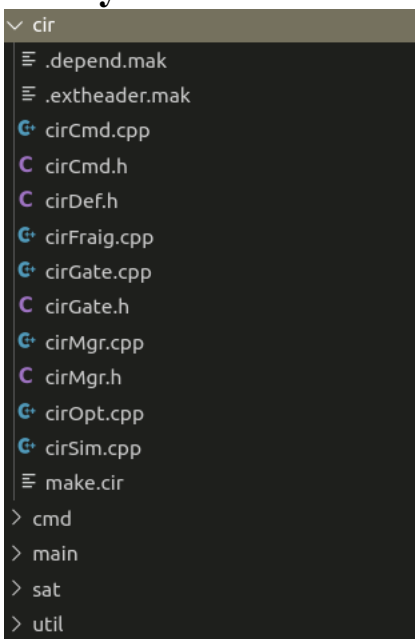
In this final project, we are going to implement a special circuit representation, FRAIG (Functionally Reduced And-Inverter Graph), from a circuit description file. The generated executable has the following usage:

```
fraig [-File <dofile>]
```

Other than the commands in Homework #3#4#6, we will support these new commands:

```
#CIRRead: read in a circuit and construct the netlist
CIRRead <(string fileName)> [-Replace]

#CIRPrint: print circuit
CIRPrint [-Summary | -Netlist | -PI | -PO | -FLoating | -FECpairs]

#CIRGate: report a gate
CIRGate <(int gateId)> [-FANIn (int level) | -FANOut (int level)]

#CIRWrite: write the netlist to an ASCII AIG file (.aag)
CIRWrite [(int gateId)] [-Output (string aagFile)]

#----------------------------------------------------------------------#

#CIRSWeep: remove unused gates
CIRSWeep

#CIROPTimize: perform trivial optimizations
CIROPTimize

#CIRSTRash: perform structural hash on the circuit netlist
CIRSTRash

#CIRSIMulate: perform Boolean logic simulation on the circuit
CIRSimulate <-Random | -File <string patternFile>> [-Output (string logFile)]

#CIRFraig: perform FRAIG operation on the circuit
CIRFraig

#CIRMiter: create a miter circuit
CIRMiter <(string inFile1)> <(string inFile2)>
```

III. System overview



主要的系統有分兩大塊，一塊是負責維持 command line 相關功能的程式以 cmdMgr 這個物件為中心來運作，於作業三以前已經完成。一塊是負責處理 AIGER (.aag) format 電路的讀寫相關功能，以 cirMgr 這個物件為中心來運作，並於作業六中已經完成電路讀取和各種顯示的部分，本次 project 的內容則會以我的作業六為基礎疊加，加入編輯電路的功能，更新的原始碼集中於 cir/中，其他未改，以達到運行以下功能的能力：

1. Unused gate sweeping (cirsweep)
2. Trivial optimization (constant propagation) (ciroptimize)
3. Simplification by structural hash (cirstrash)
4. ~~FRAIG: Equivalence gate merging (cirsimulate, cirfraig) 未完成~~

IV. Date Structure

```
class SatSolver;

class CirGate;
class CirMgr;

class CirPiGate;
class CirPoGate;
class CirAigGate;

typedef map<unsigned, CirGate*>   GateMap;
typedef pair<unsigned, CirGate*>  GatePair;
typedef unordered_map<unsigned long long int, CirGate*>  GateHash;
typedef pair<unsigned long long int, CirGate*>  HashPair;


typedef vector<CirGate*>          GateList;
typedef set<unsigned>            GateIntSet;

enum GateType
{
   UNDEF_GATE = 0,
   PI_GATE    = 1,
   PO_GATE    = 2,
   AIG_GATE   = 3,
   CONST_GATE = 4,

   TOT_GATE
};
```

1. Cirmgr Class
裡面儲存的 data member 有 aag 的 header 以 unsigned 存，在電路編輯過程中 AIG 的數量會跟著變動，而主要儲存所有 gate 的大容器是_gateList，是用 map implement 的以確保在搜尋時快速，且在 traversal 時不會因為改動而順序錯亂(相對於 unodered_map)，_hashmap 則是在 cirstrash 功能時會用到，用來儲存 hash 值和 gate 的 pointer，是 unodered_map 也就是 hash table，可以快速 find, insert 和 delete，_piList 和 _poList 則是一般的 vector，floating gates 和 unused gates 存在_floList 和_notuList 中，用 set 存，方便儲存(消除重複儲存)和刪除(即使沒有，刪了也不會 error)。

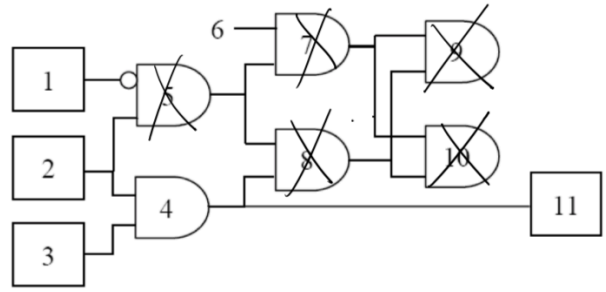
```
// Data member
unsigned _MaxVaIdx, _PI, _LA, _PO, _AIG; //header
GateMap           _gateList;
GateHash           _hashmap;
GateList            _piList;
GateList            _poList;
GateIntSet         _floList;
GateIntSet        _notuList;
ofstream           *_simLog;
```

2. CirGate Class and its family
這個部分使用到 Polymorphism 的技巧，透過 CirGate 來製造出 CirPiGate, CirPoGate, CirAigGate，每個 Class 都代表各自對應的 gate (UNDEF 和 CONST0 屬於 CirPiGate)，而 CirGate 家族裡面儲存的 data member 有基本的 gate 資訊如：_gateID, _lineNo, _gateType, _symbolName，和剛開始讀入電路時需要的 faninID 和他的 inverse list，以簡單的 vector 來儲存，等到讀完 file 要做電路 connection 的時候，再建構每個 gate 的 fanout 和 fanin list 和他們的 inverse list，來儲存 fanin 和 fanout gate 的 pointer，並有多個 access function 配合，來對這四個 vector list 做編輯。

```
protected:
   //itself
   unsigned _gateID;
   unsigned _lineNo;
   string _gateType;
   string* _symbolName = 0;
   //fanout
   GateList _fanoutList;
   vector<bool> _fanoutInvList;
   //fanin
   GateList _faninList;
   vector<bool> _faninInvList;
   //faninId
   vector<unsigned> _faninIdList;
```

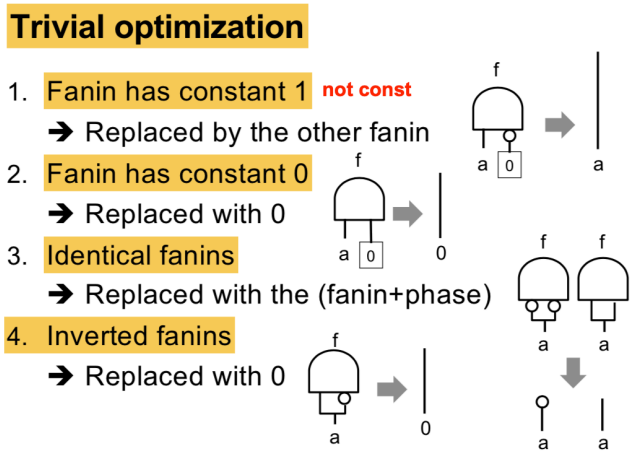
V. Algorithm

1. Unused gate sweeping (cirsweep)



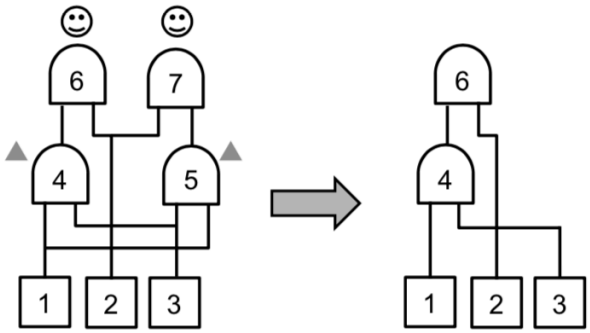
因為是要清掉不是從 PO 做 DFS 可以觸及到的 gate，所以我的做法是先跑過一次 DFS 但並不清(reset)掉 marked，然後直接 traversal 全部的 gate，只要沒 marked 就通通刪掉，即可達成此功能，只是在實作時要特別注意刪掉 gate 時他的 fanin gates 和 fanout gates 的 fanout 和 fanin 的內容也要清乾淨。

2. Trivial optimization (constant propagation) (ciroptimize)



因為這個功能是化簡後前面的 gate 會影響到後面的 gate，所以我的作法也是跑 DFS，DFS 從 PO 一路擴展到 PI，然後在一路往回結束 recursive function 時(DFS 演算法中把 node 塗黑的部分)，來用 if else 做這四種 trivial optimization case 的分析，即可達成此功能，只是在實作時也要特別注意刪掉 gate 時他的 fanin gates 和 fanout gates 的 fanout 和 fanin 的內容也要清乾淨，然後 case1 和 case3 會有 inverse 負負變成正的情況。

3. Simplification by structural hash (cirstrash)



因為這個功能也是化簡後前面的 gate 會影響到後面的 gate，所以我的作法還是跑 DFS，DFS 從 PO 一路擴展到 PI，然後在一路往回結束 recursive function 時(DFS 演算法中把 node 塗黑的部分)，來用 hash table 判斷這個 gate 的 fanin 以前有沒有出現過，有的話就合併，沒有的話就加入 hash table，即可達成此功能，只是在實作時也要特別注意刪掉 gate 時他的 fanin gates 和 fanout gates 的 fanout 和 fanin 的內容也要清乾淨。Hash 方式的部分，我把每個 gate 的 fanin hash 成一串 unsigned long long int：

```
//hashing aig gate fanins
stringstream ss;
short int faninInv0 = (gate->getFaninInv(0)==0) ? 2 : 3;
short int faninInv1 = (gate->getFaninInv(1)==0) ? 2 : 3;
unsigned faninId0 = gate->getFanin(0)->getGateId();
unsigned faninId1 = gate->getFanin(1)->getGateId();
short int faninType0, faninType1;
string faninT0 = gate->getFanin(0)->getTypeStr();
string faninT1 = gate->getFanin(1)->getTypeStr();
if (faninT0=="UNDEF") faninType0=0;
else if (faninT0=="PI") faninType0=1;
else if (faninT0=="PO") faninType0=2;
else if (faninT0=="AIG") faninType0=3;
else if (faninT0=="CONST") faninType0=4;

if (faninT1=="UNDEF") faninType1=0;
else if (faninT1=="PI") faninType1=1;
else if (faninT1=="PO") faninType1=2;
else if (faninT1=="AIG") faninType1=3;
else if (faninT1=="CONST") faninType1=4;

if(faninId0 < faninId1)
  ss<<faninInv0<<faninType0<<faninId0<<faninInv1<<faninType1<<faninId1;
else if (faninId0 == faninId1){
  if (faninInv0<faninInv1)
    ss<<faninInv0<<faninType0<<faninId0<<faninInv1<<faninType1<<faninId1;
  else
    ss<<faninInv1<<faninType0<<faninId0<<faninInv0<<faninType1<<faninId1;
}
else
  ss<<faninInv1<<faninType1<<faninId1<<faninInv0<<faninType0<<faninId0;
//cout<<ss.str()<<endl;
unsigned long long int hash_idx = strtoull(ss.str().c_str(), NULL,10);
GateHash::iterator iter = _hashmap.find(hash_idx);
```

Pseudo code 的部分：

```
for_each_gate_from_pi_to_po(gate, hash)
    // Create the hash key by gate's fanins
    HashKey<...> k(...);  // a function of fanins
    size_t  mergeGate;
    if (hash.check(k, mergeGate) == true)
        // mergeGate is set when found
        mergeGate.merge(gate);
    else  hash.forceInsert(k, gate);
```

VI. Experiment & Result

根據作業給的電路，tests.fraig 裡面的所有電路都為測試對象，因此我自己寫了一個 bash script 來對每個電路都自動跑過同一份的 dofile，並且每個電路都要 reference program 和我自己的 program 做 diff 比較差異，以確認 correctness。autoTest.sh 如下：

```
vim autoTest.sh
#!/bin/bash
# automatically go through all test circuit with same do file

DOFILE=do.opt

for entry in "$search_dir"*.aag
do
  echo "$entry"
  sed -i "1s/.*/cirr $entry/" $DOFILE
  ../fraig -f $DOFILE > log/log_"$entry".txt 2>&1
  ../ref/fraig -f $DOFILE > log/log_"$entry".ref.txt 2>&1
  diff log/log_"$entry".txt log/log_"$entry".ref.txt
done
```

而下圖則是全部 pass 時會有的輸出，因為 diff 在兩個檔案完全一致的時候不會有 output，所以 output 就只剩下檔名。

```
→ tests.fraig git:(master) ✗ bash autoTest.sh
C1355.aag
C17.aag
C1908.aag
C3540.aag
C432.aag
C432_r.aag
C499.aag
C499_r.aag
C5315.aag
C6288.aag
C7552.aag
C880.aag
opt01.aag
opt02.aag
opt03.aag
opt04.aag
opt05.aag
opt06.aag
opt07.aag
sim01.aag
sim02.aag
sim03.aag
sim04.aag
sim05.aag
sim06.aag
sim07.aag
sim08.aag
sim09.aag
sim10.aag
sim11.aag
sim12.aag
sim13.aag
sim14.aag
sim15.aag
strash01.aag
strash02.aag
strash03.aag
strash04.aag
strash05.aag
strash06.aag
strash07.aag
strash08.aag
strash09.aag
strash10.aag
test1.aag
test2.aag
```

1. 單測試 cirsweep

dofile 的設計上，簡單讀入電路後，看 cirp 的所有輸出方式，看 cirg 挑幾個 gate 做 fanin 和 fanout，就進行 cirsweep，然後再看 cirp 的所有輸出方式，再看 cirg 挑前面挑的幾個 gate 做 fanin 和 fanout。

2. 單測試 ciroptimize

dofile 的設計上，簡單讀入電路後，看 cirp 的所有輸出方式，看 cirg 挑幾個 gate 做 fanin 和 fanout，就進行 ciroptimize，然後再看 cirp 的所有輸出方式，再看 cirg 挑前面挑的幾個 gate 做 fanin 和 fanout。

3. 單測試 cirstrash
dofile 的設計上，簡單讀入電路後，看 cirp 的所有輸出方式，看 cirg 挑幾個 gate 做 fanin 和 fanout，就進行 cirstrash，然後再看 cirp 的所有輸出方式，再看 cirg 挑前面挑的幾個 gate 做 fanin 和 fanout。
4. 整合測試
dofile 的設計上，簡單讀入電路後，看 cirp 的所有輸出方式，看 cirg 挑幾個 gate 做 fanin 和 fanout，就進行 cirsweep, ciroptimize, cirstrash，然後再看 cirp 的所有輸出方式，再看 cirg 挑前面挑的幾個 gate 做 fanin 和 fanout。

VII. Discussion

根據實驗結果，大致上可以保證這次有完成的三大功能，在 correctness 上沒有問題，但因為是沿用自己上次作業的架構和 function，除了 error detection 和 circuit write(忘了真的輸出成檔案)沒拿到分數外，我還有在 fanin 和 fanout 的顯示上並沒有拿到全部的分數，所以 fanin fanout 的功能上可能還有瑕疵，導致如果在別的測資上有測到與 reference program 不符，也有可能是作業六就已經存在的 bug(雖然部分 function 也有在這次做修改，但不知道有沒有改到)，並不一定是這次實作的功能本身的 bug。
而會有這樣全部通過的結果，也是經過一次一次的測試和修改而來的，像是實驗中提到的三個功能分開來測試時，就已經跟對個別 function 修改到沒有錯，但在同一份 dofile 一起側的時候，又會發現之前沒發現的 bug，如此一直修改下來才有最終的結果，詳細修改的 bug 可以參閱 github 上的 commit，數量眾多就不在此一一列出。

VII. Conclusion

這次因為時間不太夠，只成功 implement 了 sweep, optimization, strash 並驗證他們的 correctness，在我設計的 dofile 下都有成功通過所有可取得的側資，期望之後有時間還可以完成另外 cirsimulate, cirfraig 的功能，並驗證 correctness 和大家的 performance。