(108-1)資料結構與程式設計(Data Structure and Programming)

Final Project Report

FRAIG (Functionally Reduced And-Inverter Graph)

系級：資管三

學號：**B06705027**

姓名：黃柏叡

信箱：**B06705027@ntu.edu.tw**

一、 程式資料結構 – 基礎架構

在這次Final Project中，我們需要實作五項指令：

Sweep, Optimize, Strash, Simulate, Fraig。在陳述這些指令如何執行時，需要先知道其基礎資料結構，即Cirmgr和CirGate是如何運作的

※作業6之基礎架構來自B06901103之Hw6

1. Cirmgr資料結構

Class Members:

```
vector<CirGate*> PIlist;
vector<CirGate*> POlist;
vector<CirGate*> totallist;
int AIG_num;
int max_id;
ofstream *_simLog;
```

Cirmgr負責執行各種CirCmd，其中PIlist負責記錄PI gate，POlist負責記錄PO gate，而totallist則記錄了各種gate，並將其在visualize成電路圖的編號對應到其在陣列的index，以便迅速完成查找。

2. Cirgate 資料結構

Class Members:

```
  static unsigned _globalRef;
  mutable unsigned _ref;
```

這兩項class member為再生成DFSlist時所依靠的指標：先將_globalRef++，並把符合進入DFSlist條件但未在裏頭的gate的_ref同步至_globlalRef，並將其加入。

```
void Function()
{
if (!_fanin[i]->isGlobalRef()) {
_fanin[i]->setToGlobalRef();
_fanin[i]-> Function();
}
// Fuction implememtation
}
```

```
protected:
    unsigned id;
    vector <CirGate*> _fanin;
    vector <bool> _ifInvIn;
    vector <CirGate*> _fanout;
private:
    unsigned line;
```

對一個Gate而言，其會有兩扇指向其的fanin gate，多扇自身指向他人的fanout gate，而這兩項數據分別記錄在_fanin和_fanout的vector中；vector _ifInvIn則配合_fanin做使用，以判別該扇fanin是否為invert。

Inheritance Child Class

```
class AIG
class PI
class PO
class UNDEF
class CONST
```

這五個Child Class分別對應至Hw6所要求之五扇不同Type的Class

二、 程式資料結構 – 指令實作

1. CIRSWeep

a. 目標：

移除掉無法被PO reach到的gate

b. 實現：

因為不能移除掉PI、PO以及CONST0，所以一開始判定時只取用在totallist裏頭getTypeStr為PO或UNDEF的，並針對其fanout.size()是否為0；若為0則刪除。

跑for迴圈判定時，我選擇從totallist的尾端開始跑，以防遇到gate的fanout在後續判斷要刪掉但因迴圈沒跑到則該gate沒被刪到之情形。

2. CIROPTimize

a. 目標：

移除掉下列4種情況之gate：

1. fanin含有CONST0的Gate： OPT to CONST0
2. fanin含有CONST1的Gate： OPT to CONST1
3. 擁有相同fanin且兩者inv相同： OPT to 該gate
4. 擁有相同fanin且兩者inv相反： OPT to CONST0

b. 實現：

對於滿足上述4條件的Gate，其操作如下：

1. 對於該fanin0和fanin1，在他們的fanout中刪除this
2. 判定該由哪個fanin繼承其fanout後，將this的fanout其fanin改換對象
3. 判斷this的fanout其fanin之原本inv屬性，並將其和this的fanin之fanout的inv進行xor，以判斷新屬性之fanin是否為inv

3. CIRSTRash

a. 目標：

將擁有相同fanin之gate透過hash比較，將重複的gate merge起來。

b. 實現：

STRash指令透過unordered_map進行實作：

1. 創建一unordered_map，其key為該gate之fanin，value為該gate之指標
2. Overload Operater ==，透過比較兩者fanin和其對應inv是否一樣；需要留意兩者fanin相同，但可能gate A之fanin 0 = gate B之fanin 1 的情況
3. 對於hash function如何產生出唯一值，operztor()的辦法為將該gate的各fanin和其inv取size_t後進行xor；之後在將兩者

xor後的值再進行xor；由於沒有shift bit，A xor B = B xor A，而也只有在兩者的fanin和inv相同時才會產生相同值。

4. 在將一個AIG insert進去map時，先判定是否有相同hash值的pair已經存在在map中；若不存在則成功insert進去
5. 若存在相同hash之pair:
    i. 得知該pair屬於哪一個gate
    ii. 將原先要insert之gate和該gate merge
    iii. 將原先要insert之gate的fanout push進存在在map的gate中，並更改那些fanout的fanin值；因為merge掉後原先要insert之gate會被delete，故需要調整其fanin的fanout，刪掉該gate。

以上各指令在運行給予的測資中皆未出現差錯；檢測CIRPrint及CIRGate也都正常；CIRSIM以及CIRF並未實做出結果，故不談及。

三、 效能測試

在測試過多組sim.aag後，我最終選擇將sim9，sim12以及sim13三項測資當作sample進行上述三指令之比較。

1. sim09.aag

指令	自身程式效能	REF程式效能
CIRR	0.01s; 1.863M	0s; 1.441M
CIRSW	0s; 1.863M	0s: 1.441M
CIROPT	0.06s; 1.871M	0s; 1.484M
CIRSTR	0.09s; 1.816M	0s; 1.715M

由圖可見，在sim09.aag上時間差距並不大；在指令OPT及STR上的些許時間差距估計是因為都會跑到totallist的for迴圈，因此有些time waste；而在空間上，因為有不少vector，然而其內裝的可能是重複的值，因此在追尋快速查找時，亦耗費了一些多餘的空間。

2. sim12.aag

指令	自身程式效能	REF程式效能
CIRR	0.04s; 3.688M	0.1s; 2.414M
CIRSW	0s; 3.688M	0s; 2.414M
CIROPT	0.01s; 3.688M	0s; 2.754M
CIRSTR	0.14s; 3.801M	0s; 3.266M

在sim12.aag中，我們可以看出雖然差距不大，但在CIRSTR中已和ref code方面有肉眼可見的時間落差；估計落差是因為跑

DFSlist以及totallist的for迴圈，遇到更大筆的資料而耗費更多時間；然而相較ref code的好處是，在跑完CIRSTR後，總和刪除的totallist以及新增map的記憶體，自身程式相較ref code的記憶體消耗較小。

3. sim13.aag

指令	自身程式效能	REF程式效能
CIRR	0.43s; 28.5M	0.07s; 14.74M
CIRSW	0.01s; 28.5M	0s; 14.74M
CIROPT	0.06s; 28.5M	0s; 15.78M
CIRSTR	6s; 28.5M	0.02s; 18.62M

從sim13中，因為有較多AIG，不僅CIRR耗費記憶體較多，跑迴圈跟DFSlist的時間代價也有顯著落差；然而在記憶體上，跑完CIRSTR後依舊沒有增加。

四、 討論及總結

1. 在CIRRead上，因為**同筆資料會重複放入不同vector**中，因此雖讓之後的**指令查找速度增加**，但造成**記憶體上的浪費**。

2. CIRSW及CIROPT上的時間效能和ref code只有些微差異，跑完CIROPT後**空間**也沒有增加，是其相較下之**優點**。

3. CIRSTR因為**引入DFSlist以及totallist的for迴圈**，可以在較多筆測資中發現**時間上有顯著差異**；雖然空間有所減少，但

在以時間換取空間上表現並不好。