

# FRAIG

Functionally Reduced And-Inverter Graph

B05901011 電機二 許秉倫 [b05901011@ntu.edu.tw](mailto:b05901011@ntu.edu.tw)

## 程式架構

以下會以 `CirMgr`、`CirGate` 兩個最重要的Class說明我整個程式的主架構

基本上我設計架構秉持時著一個重要的精神

1. 不寫重複code，全力把功能相似的動作包成同一個function
2. 要access一個值時，全力讓他可以 $O(1)$ 達成

### CirMgr

```
1  class CirMgr
2  {
3  public:
4      // hw6 function
5      void dfsPostOrder(CirGate*);
6      void resetFlag();
7
8  private:
9      GateList gateList, dfsList;
10     vector<size_t> piOrder, poOrder;
11     size_t maxIndex;
12
13 };
```

### Member Function

- `dfsPostOrder` :
  - 依照postOrder的順序去更新dfsList
- `resetFlag` :
  - reset

### Member Variable

- `gateList` :

- 存放 `CirGate*` 的指標，大小為  $M+O+1$ ，初始化為0
- `dfsList` :
  - 存放在 `dfsPostOrder` 時traverse過的 `CirGate*`
- `piOrder/poOrder` :
  - 依照readfile的順序存放pi, po至 `vector` 中，以利在writeFile時順序仍維持跟原始相同
- `maxIndex` :
  - 存放 `maxIndex` 的值，供write時使用

## CirGate

```

1  class CirGate
2  {
3  public:
4      // hw6 function
5      // fanout也是做一樣的事，就不一一列出
6      void dfsFanin(int, string = "", bool = 0) const;
7      void addFanin(CirGate* g, bool inv = false);
8      CirGate* getFanin(size_t i) const;
9      bool isFaninInv(size_t i) const;
10 protected:
11     GateList fanin, fanout;
12     GateType type;
13     string symbol;
14     size_t line, id;
15     mutable bool flag;
16 };

```

## Member Function

- `dfsFanin` :
  - 對gate做dfs，print出他的fanin cone
- `addFanin` :
  - 加一個gate到fanin中，把invert與否存放在第一個bit
- `getFanin` :
  - 取得fanin的 `CirGate*`

- `isFaninInv` :
  - 取得fanin的invert

## Member Variable

- `fanin fanout` :
  - 存放fanin, fanout的 `CirGate*`
- `type, symbol, line, id` :
  - 存放gate的基本資訊
- `flag` :
  - 用來做各種標註

# Sweep, Opt and Strash

## removeGate

Core Utils Function for Gate Removing

```
1  class CirGate
2  {
3      public:
4          // Utils
5          void updateFan(FanType, CirGate*, CirGate*);
6          void removeGate(int, int);
7
8          // USAGE
9          removeGate(拿來取代原fanin的fanout的literal ID,
10                  拿來取代原fanout的fanin的literal ID)
11          // call updateFan 去處理接fanin/fanout的事情
12          // 附註：若沒有gate會接到原gate的fanin/fanout上，則literalID傳入-1
13  };
```

## Sweep

1. 將有在 `dfsList` 中的gate的flag標示為1
2. 將沒被標示到的gate做 `removeGate(-1, -1)`
3. 更新`dfsList`

## Opt

1. 先分三種狀況
  1. 左方literal = 1 || 右方literal = 1
  2. 左方literal = 0 || 右方literal = 0
  3. 左方literal == 右方literal
2. 將要拿來merge的gate的literalID存到 `size_t litSub`
3. 執行 `removeGate(litSub, -1)`
4. iterate `dfsList` 去刪除UNDEF且unused的gate
5. 更新 `dfsList`

## Strash

1. 開一個 `unordered_map<size_t, CirGate*> map;`
2. iterate `dfsList`，把在裡頭的gate的兩個fanin的literal ID排序過後，合成一個 `size_t` (將較小的左

移32位元)，存到 `map` 中

3. 若 `map[key].count == 1`，則執行 `removeGate(lit,-1)`

## 結語

1. 透過設計良好的utils function可以讓程式碼變的簡潔易懂，我僅透過一個utils function搭配壹些簡單的判斷式就完成了這三個basic function
2. 適當的使用literal ID可以表示是否為invert的優勢，能讓程式碼變得優雅好寫

# Simulation

---

## FecGroup

新加入的角色： FecGroup !  
裡頭存放隸屬於同個FecGroup的gate literal ID

```
1  class FecGroup
2  {
3      public:
4          void removeMember(size_t);
5          void setMember(size_t, size_t);
6          void pushMember(size_t);
7      private:
8          IdList member;
9  };
```

## Member Function

- `removeMember` :
  - 將gate從這個group中移除
  - 方法是將他用最後一個取代，且將最後一個pop掉，時間複雜度為 $O(1)$
- `setMember` :
  - 將variable ID相同的gate的literal ID重設
  - 例如: 5->I5
- `pushMember` :
  - 將gate `push_back` 進這個group

## Member Variable

- `member` :
  - 以 `IdList` 存放裡面每個gate的literal ID

## CirGate

```

1 | class CirGate
2 | {
3 | public:
4 |     virtual void simulate() = 0;
5 | protected:
6 |     FecGroup* fecGroup;
7 |     size_t simVal;

```

## Member Function

- `simulate`
  - 依照gate的功能不同，更新每個gate的 `simVal`

## Member Variable

- `fecGroup`
  - 存放 `*fecGroup` 放它可以以O(1)時間access到他的 `fecGroup`
- `simVal`
  - 存放gate output的simulation value

## CirMgr

```

1 | class CirMgr
2 | {
3 | public:
4 |     // simulation
5 |     void sim64(vector<size_t>&, bool = 0);
6 |     void log64(size_t l = 64);
7 |     void initFecList();
8 |     bool removeFecGroup(size_t);
9 | private:
10 |     FecList fecList;
11 |     bool hasSim;
12 | };

```

## Member Function

- `sim64`
  - 平行模擬64個bits，傳入一個size為PI個數的 `vector<size_t>`，給在dfsList當中的 `gate` 去

做simulation

- `log64`
  - 以string形式輸出64bits的PO `simValue`
- `initFecList`
  - 將CONST和所有在 `dfsList` 中的AIG放入一個fecGroup，並 `push` 進 `fecList`
- `removeFecGroup`
  - 將個數為0或1的 `fecGroup` 從fecList中移除

## Member Variable

- `fecList` :
  - 型態為 `vector<*FecGroup>`
  - 存放 `FecGroup` 的指標
- `hasSim`
  - 判斷是否有simulation過

## Algorithm

主要的演算法執行在 `sim64`，`randomSim`、`fileSim` 僅是將pattern處理過後傳入 `sim64` 中執行，因此以下僅介紹如何實作sim64

### 1. 前置作業(在random/fileSim中)

1. 若 `!hasSim`，call `initFecList` 將所有的gate加到同一個 `*fecGroup` 中

### 2. 用input去simulate circuit

1. 先將input存入pi當中
2. 依照 `dfsList` 順序去做simulation

```
1 | for(size_t i = 0; i < piOrder.size(); i++)
2 |     gateList[piOrder[i]]->simVal = input[i];
3 |
4 | for(size_t i = 0; i < dfsList.size(); i++)
5 |     dfsList[i]->simulate();
```

### 3. 處理 `fecGroup`

1. iterate `fecList`，對於當中的每個 `fecGroup` 產生一個 `HashMap` 去檢查 `simVal` 是否有



對應的 `*fecGroup`

2. iterate `fecGroup->member`

1. 若他是當中的第一個gate，強迫致使他留在原始的group

2. 若 `map.count(simVal)`

1. 若 `map[simVal]` 等於原始的group，call `fecGroup->setMember` 去更新那個index 中的值(因為可能正反會不同, 例如!5->5)

2. 若 `map[simVal]` 不等於原始的group，先call `fecGroup->removeMember` 將他從原始的 `fecGroup` 中刪除，再call `fecGroup->pushMember` 將他正的literal ID，push到新的 `fecGroup` 中

3. 若 `map.count(~simVal)` 則執行類似2的動作，只是將正改為反

4. 若都不符合上述兩種情況，則開創一個新的 `fecGroup` 並把它將到裡頭

3. 如果這個 `fecGroup` 分完後，裡面只剩一個member，則可以將它從刪除(重要 大大提升效率)

```
1   for(size_t i = 0; i < fecList.size();){
2       unordered_map<size_t, FecGroup*> map;
3       for(size_t j = 0; j < fecList[i]->member.size();){
4           //讓第一個留在原group
5           if(j == 0){...}
6           if(map.count(simVal)){...}
7           else
8           if(map.count(~simVal)){...}
9           else{...}
10      }
11      //如果它的大小是1，砍掉，continue
12      if(removeFecGroup(i)){continue;}
13  }
```

## Performance Compare

```
#####sim12#####
< Period time used : 0.73 seconds
< Total time used : 0.73 seconds
< Total memory used: 4.145 M Bytes
---
> Period time used : 0.78 seconds
> Total time used : 0.78 seconds
> Total memory used: 4.004 M Bytes
#####sim13#####
< Period time used : 10.99 seconds
< Total time used : 10.99 seconds
< Total memory used: 35.2 M Bytes
---
> Period time used : 10.73 seconds
> Total time used : 10.73 seconds
> Total memory used: 23.59 M Bytes
```

## 測試

- dofile(generated by my **FRAIG TESTING ENGINE**)

```
1 | cirr ./tests.fraig/simxx.aag
2 | cirsim -f tests.fraig/pattern.xx -o xx.log
3 | cirp -fec
4 | cirg *
```

## 分析

除了sim12, sim13以外，period time幾乎都趨近零秒，所以僅拿這兩個來做效能分析  
紅色為my program，綠色為ref program

在數次的優化後，效能終於和ref program相近，我認為其中最重要的兩項優化為

1. 將 `erase` 改為hw5中 `remove` 的做法， $O(n) \rightarrow O(1)$
2. 將數量僅剩一個的 `fecGroup` 刪除，若不刪除的話，需要花時間產生大量無意義的 `unordered_map`，runtime會數倍之多

# Fraig

在這個部分，因為演算法較為複雜，直接介紹class裡頭多加了哪些member可能會讓讀者看得一頭霧水，所以我決定先解析我在設計這套演算法時的思路，再說明要使用這套演算法時，需要多加入哪些member

## Algorithm

### 笨方法

單打獨鬥下的結果

要做fraig，最直覺的方法是什麼？

先吃盡整張圖後，將所有的 `fecGroup` 利用 `SAT ENGINE` 一一證明，若證出不一樣便將使其不一樣的 `pattern` 丟入 `sim`，把 `fecGroup` 拆群，這樣一來，最終留在同一個group的即為可相互合併的gate，挑選出其中id最小的gate，使用 `removeGate` 以他為基準去合併別人

這樣會有什麼問題

1. 會莫名 `segmentation fault`
2. 效率奇差，`sim12` 要跑 `80s`、`sim13` 沒有跑完過...

WHY？

1. 經過 林承德 大神的精闢解析後，我意識到gate是不能亂merge的，否則可能會產生cycle，因此會導致 `segmentation fault`，最安全的方法是把dfsOrder順序在最前面的gate當作base去merge別人
2. 經過 歐瀚墨 大神的開示後，我理解到SAT的時間複雜度和 吃進去的圖 呈正相關，一次吃進去整張圖，當然會使程式慢的跟蝸牛一樣

## 🔥🔥 HOCL Algorithm 🔥🔥

designed by Byron Hsu, Hanmo Ou, Perry Chen, Jerry Lin

終極目標

使每次吃進去的圖最小化！

思路

1. 若想使吃進 `solver` 的圖變小，最簡單的做法就是在prove兩個gate的時候，不要吃進整張圖，而是只吃入他們 `fanin cone` 的聯集  
但我每次都一定得吃入原圖中的 `fanin cone` 嗎？  
不，我可以在每次merge過後更新他的圖再把變小的圖吃進去，  
但怎樣才可以讓每個gate平均吃到的gate最少呢？  
讓gate按照postOrder順序去被merge！利用postOrder的特性，這樣一來在父親被merged時，小孩已經被merge過了，代表他的 `fanin cone` 會越縮越小🚀（速度的關鍵）

## 2. 還可以做什麼優化？

在gate已經被merge過後，把沒有在新的 `dfsList` 當中所出現的gate從 `fecGroup` 刪除，這樣一來可以減少非常多無謂的 SAT PROVE ✈️ (加速的關鍵)！因為最終那些gate都會在 `sweep` 時被去除掉

## 如何實作？

Base：fecGroup中拿來merge別人的gate，為fecGroup中dfsOrder順序最前面者

### 流程

1. 在 `cirfraig` 前先更新一次每個 `fecGroup` 中的base
2. 開始iterate `dfsList`，如果找到 AIG 且他的 `fecGroup` 不為 `NULL` 則
  1. 若他的 `base` 等於他自己，則直接continue，因為他不可能被merge
  2. 跑dfs將兩個他與他的 `base` 的 `fanin` 聯集存到 `coneList` 中
  3. 將 `coneList` 丟入 `genProofModel`
  4. 若 SAT
    - 把 `pattern` 丟入sim64將其拆開
5. 若 UNSAT
  1. 將它從他的 `fecGroup` 中拔除
  2. 若刪完後他的原 `fecGroup` 的大小變為0或1，則把它從 `fecList` 中拔除
  3. call `removeGate` 去做merge gate
  4. `updateFraigDfs`，如果不 `updateDfs` 的話，跑sim的時候最會炸掉，因為sim是仰賴於 `dfsList`，如果不更新的話，會跑到已經被 `delete` 掉的gate.
  5. 對每個 `fecGroup` 做 `setBase` 和 `removeFecGroup`，`setBase` 時先將不在 `dfsList` 當中gate移除，再找出當中 `dfsOrder` 最前面的當作 `base`，`removeFecGroup` 則是把更新過後大小變為0或1的 `fecGroup` 從 `fecList` 移除
  6. 從新的 `dfsList` 的頭開始跑

在class中新增添的member

## FecGroup

```
1 class FecGroup
2 {
3     public:
4         void setBase();
5     private:
6         size_t baseIdx;
7 };
```

### Member Function

- `setBase`
  - 把沒在 `dfsList` 中的gate移除
  - 藉由 `dOrder` 更新 `baseIdx`

### Member Variable

- `baseIdx`
  - 儲存base在 `fecGroup` 中的哪個 `index`

## CirGate

```
1 class CirGate
2 {
3     public:
4         void setVar(size_t& v);
5         size_t getVar();
6     protected:
7         size_t fIdx, var;
8         int dOrder;
9 };
```

### Member Function

- `setVar` `getVar`
  - 設置，取得在 `solver` 中的 `var`

## Member Variable

- `fIdx`
  - 儲存這個gate在他的 `fecGroup` 當中的 `index`，會在 `removeMember` 等等函式去做動態的更新
  - 為了使每個 `gate` 被merge時，我可以以O(1)時間得知他在 `fecGroup` 的何處並快速刪除
- `dOrder`
  - 存放每個gate在 `dfsList` 當中的 `order`，會在 `updateFraigDfs` 去做動態的更新

## CirMgr

```
1 | class CirMgr
2 | {
3 | public:
4 |     void genProofModel(GateList&);
5 |     bool isSAT(size_t, size_t, vector<size_t>&, GateList&);
6 |     void updateDfsFraig();
7 |     void findFaninCone(CirGate*, GateList&);
8 | private:
9 |     bool hasFraig;
10 | };
```

## Member Function

- `genProofModel`
  - 傳入 `coneList` 讓他建圖
- `isSAT`
  - 判斷兩個gate SAT
- `updateDfsFraig`
  - merge後更新 `dfsList` 和 `dOrder`
- `findFaninCone`
  - 讓gate跑遞迴，把fanin存到 `ConeList` 中

## Member Variable

- `hasFraig`

- 判斷是否有做過 `cirfraig`，如果有，我就直接在 `cirFraig` 中把他擋掉
- 因為我的程式能保證一次畫到最簡

## Performance Compare

#####sim13#####	#####sim07#####
36,38c36,38	36,38c36,38
< Period time used : 37.42 seconds	< Period time used : 2.34 seconds
< Total time used : 37.42 seconds	< Total time used : 2.34 seconds
< Total memory used: 35.39 M Bytes	< Total memory used: 3.758 M Bytes
---	---
> Period time used : 93 seconds	> Period time used : 7.69 seconds
> Total time used : 93 seconds	> Total time used : 7.69 seconds
> Total memory used: 47.34 M Bytes	> Total memory used: 8.191 M Bytes
AIG 0	AIG 0

紅色為my program、綠色為ref program

## 測試

- dofile(generated by my **FRAIG TESTING ENGINE**)

```

1 | cirr ./tests.fraig/xx.aag
2 | // 直接做cirfraig以精準測試cirfraig的效能(cirstrash ciropt都有可能是他的子集合)
3 | // iterate 3 times
4 | cirsim -r
5 | cirfraig...
6 | // print fec to ensure fecList is empty after fraig
7 | cirp -fec
8 | // 做三種化簡以確保他化至最簡
9 | cirsw
10 | ciropt
11 | cirstrash
12 | cirp
13 | // 印出他的summary比對aig個數、不印其他的因為最後結構可能長的不同
14 | usage
15 | q -f
16 | // 最後再透過ref的miter驗證他們是否相等

```

## 分析

僅花ref-program

 三分之一 










左右的時間就把cirFraig跑完，記憶體用量也較ref program少

且最後有透過 `miter` 驗證結果正確

# 心得

## First Commit










Commits on Dec 23, 2017

 ✨ finish sweep and optimize ... ByronHsu committed 25 days ago	 86d924f	
 🐛 add dfsList ByronHsu committed 25 days ago	 727517a	
 📁 merge hw6 ByronHsu committed 25 days ago	 85daa3c	

Commits on Dec 21, 2017

 🚀 fraig first commit ByronHsu committed 27 days ago	 c117359	
--	---	---

## Fraig Engine Completed

 📄 modify readme ByronHsu committed 2 days ago	 daf1664	
 🌟 Byron Fraig Engine completed! ByronHsu committed 2 days ago	 31c66bd	
 🔥 3:15am now ByronHsu committed 2 days ago	 b611f57	

打開github，發現從first commit到Fraig Engine完成已經歷經27天、40個左右的commits了。

從作業發布的第一天開始，只要有閒，就會自動在腦中思索該如何設計架構、function、class，`sim`和`fraig`的演算法又該如何運作最會有效率。

為了想順利完成讀電機系一個重要的milestone 🏆 - complete fraig project

我剛發布作業的那個六日，就投入所有的時間，把前三個function和sim(不過後來有大改)完成

儘管有期末的壓力在，只要一有空就會忍不住把fraig打開，修個幾行在讀書的時候想到的bugs，嘗試各種的加速方法，發個commit到github上

我一直都很享受做大型project的感覺，可以設計自己的架構，自己的演算法，嘗試各種不同天馬行空的想  
法，期待跑出驚人的結果，縱然往往希望落空，但每一次的失敗都會讓自己越來越強。

在這段過程中，被各種大神幫助了很多(尤其是一起想出HOCL的好夥伴)，也可能幫助了很多(?)

我用 `js` 以及 `shell` 寫了一個可以測試六種case的測資

```
1 | # USAGE
2 | ./run.sh <basic|sweep|opt|strash|sim|fraig>
```

## Fraig Testing Engine



## FRAIG-TESTDATA USAGE

### Install

- Mac

```
brew install node  
brew install colordiff
```

- Linux

```
sudo apt-get install nodejs  
sudo apt-get install colordiff
```

### USAGE

1. Put run.sh, clean.sh, mydofile, myoutput in the fraig/ folder
2. Start testing using different command

```
./run.sh <basic|sweep|opt|trash|sim|fraig>
```

只要六種測資都測試成功，基本上就可以確保你有90%左右的正確率了（應該吧👤）  
做完後有提供給一些朋友做測試，大部分都有因此抓出一些debug，真的感到蠻欣慰的😂

途中遇到最大的瓶頸大概就是在資結宿營第一天的時候，那時我和L都各自寫好了第一版的 `cirFraig`，但效能都慘不忍睹...，`sim13` 要跑好幾千秒

我們(HOL)花了整個白天的時間討論到底要如何優化，那張用來寫演算法的白紙被畫得亂七八糟，大家各自提出論點，嘗試從不同面向切入，最終產生了兩個版本的演算法，但，沒下去寫真的不知道效能如何，但一寫又要花數十個小時，因此我們決定要人走一條路，最後再看誰比較快

我們花了很多的時間寫完，可以run之後，卻出現了慘劇，L所寫的雖然較之前快，但還是和ref相差甚多，我寫的則更悲劇...要跑ref的70倍左右，當時真的很絕望，不知道為什麼做了那麼多的優化結果卻不如預期

但這時卻出現了轉機，我將程式碼將上了一行(忘記clear `dfsList`)，突然，我的速度暴增，直逼 `ref` 的二分之一倍⚡！然後我們這群人就決定都套用此演算法，速度都能穩定維持 `ref` 6成時間以下

原本沒想到竟然自己可以做得那麼快，這一切真的都值得了～

## FRAIG與人生

Union is the strength

Keep thing simple, do thing smart.