

FRAIG REPORT

電機二 孟妍 B06901066

• Member Functions and Variables

(1) CirGate

gate 使用繼承的方法，每個 gate 分別有 `vector<CirGate*>` 存取 fanin/fanout 的 gate 跟 `vector<unsigned>` 存取 faninID/fanoutID（可作為判斷 inverse 用）。

`size_t SimVal` → 儲存 sim 一輪後的 output value（一直保持最新的 sim output）

`string SImOutStr` → 用來儲存 simulation 後的 gate value，用 0/1 形式的 string 儲存，方便 cirg 指令印出 gate value

`void gateSim()` → simulation 用，在 PI signal 改變時，set 對應的 AIG 及 PO gate value，也就是改變 SImOutStr

(2) CirMgr

在 CirMgr 中，主要分成幾種類別的 member functions：

1) Circuit 基本資料

1. `size_t M, I, L, O, A` → 存取各個 gate 數目

2. `CirGate** _gatelist` → 存放所有 gate 的 list, 使用 pointer 較容易操作

3. `vector<CirGate*> PIlst;` → 為了有一個維持跟 file 一樣順序的 PI, `printPIs()`, `writeAag()` 跟 simulation 會用到，邊建電路就邊建立 PIlst

4. `vector<CirGate*> POlist` → 跟建 PIlst 的目的差不多

5. `connect()` → 連接好各個 gate 的 fanout/fanin gate, 以及取得第一次讀進電路的 floating gates

6. `vector<CirGate*> Floatlist1` → 存取 gates defined but not used，可一直更新

7. `vector<CirGate*> Floatlist2` → 存取 gates with floating fanin(s)，可一直更新

2) DFS functions

`vector<CirGate*> DFSlist;` → 整個 circuit 的主要精神所在（？）很多 function 都會用到 DFS。取得 DFSlist 的方法是使用老師投影片上的，用 `globalRef` 去跟自己的 `ref` 做比較，檢查是否為 `isVisit`

1. `void getDFSlist()` → 可以從這個 function 產生 DFSlist，之後進行 `sweep`, `strash`, `optimize` 要更新 DFSlist 也是從這裡

2. `void DFStraversal(CirGate*)` → 被 `getDFSlist()` 呼叫，進行 circuit 的 DFStraversal

3) Functions for optimization

這一部分主要就是為了 `sweep()`, `optimize()`

有三個主要的 function：

1. `void removeGate(CirGate*)` → for `sweep()`

2. `void mergeGate(unsigned, unsigned, bool)` → for `optimize()` and also `strash()`

3. `void updateFloat()` → 再進行完 circuit optimization 相關指令會用到，更新

Floatlist1/Floatlist2

4) functions for fraig

void strash()

void printFEC() const

void fraig()

5) Functions for simulation

-> Data members:

vector<string> _simfile → 存入 fileSim 的 file

vector<size_t> _signal → 把 _simfile 轉換成 size_t 或 randSim 直接產生 signal，存入 _signal，之後餵進每個 PI gate 中

vector<unsigned> FECList → 存 FEC group 中第一個 gate 的 ID

-> Functions

1. bool readSimFile(ifstream&) → check file format

2. void Stringto64() → 把 file 中每 64 個 input pack 成一個 size_t，一次餵一組 parallel pattern

3. void loadSignal() → 把 _signal 餵入 PI gate 中，每個 signal 都是一個 64 bit 的 size_t，

4. void All_Gate_Sim() → 每個 PI gate 都有 gate value 後，對 DFSlist 中的 AIG 及 PO 做 gate simulation(gateSim())

5. void getFECs() → 第一輪的 signal 餵進去之後，先 collect 最原始的 fec pairs

6. void getFECList(HashMap<SimKey,unsigned>*) → 第一輪 getFECs() 後，建立第一版的 FECList

7. void updateFEC() → 一但 FECList 不是空的，之後再有新的 signal 進來，只能從 updateFEC()來更新

8. void updateFECList(unsigned) → 改變 FECList

9. void resetFECs(unsigned) → 清掉一個 FEC group 中的所有 gate 的 fec pair (to update FECList)

10. void writeSim(unsigned) → for _simLog

• Overall Structure

About Circuit Parsing

一行一行讀，順序為先 new 出 PI gate，再來讀到 PO 的時候，先 new 一個 gate ID 為 M+i+1 的 PO gate，因為還不確定 PO 連接的 gate 的性質，所以先將 PO 連接的 gate new 為 UNDEF_Gate，該 gate 之後若出現在 AIG definition 中再來更改成 AIG。再來讀到 AIG 的時候也是一樣，若該 gate 已被建立，則一定是剛剛讀 PO 的時候建立的 UNDEF_Gate，所以就把該 gate 的資訊 copy 到一個新的 AIG gate 中。AIG 的兩個 fanin 也是跟 PO 類

似的情形，若該 fanin ID 所代表的 gate 還未被建立（非 PI, 可能為 AIG or UNDEF），在建立的過程中，則先將該 fanin ID 所代表的 gate 設為 UNDEF_Gate。另外，parse circuit 的過程只先紀錄該 gate 的 fanin/fanout literal ID。等 parse 完整個 circuit 後再呼叫 connect() 連接 fanin/fanout gate（使用 CirGate*），同時建立 DFSlist 以及取得 Floating gates 資訊。

• Algorithm

(1) CIRSweep

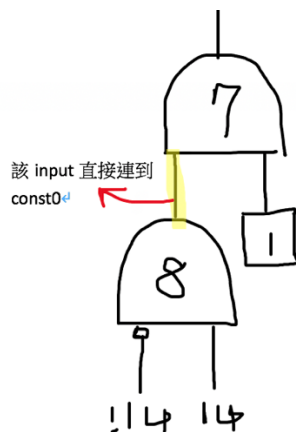
每呼叫一次 sweep() 就先清除原先的 DFSlist 再 gen 一次 DFSlist，確保有在 DFSlist 中的 gate 都有被標記過。再來建立一個 sweeplist，也就是存要被 sweep 掉的 gate。跑過整個 gatelist，若該 gate 為 AIG or UNDEF，且不在 DFSlist 中，則放入 sweeplist，呼叫 removeGate 清除所有在 sweeplist 中的 gate。清完後更新 Floatlist 和 DFSlist。

(2) CIROPTimize

Perform trivial optimization，這裡主要用到的 function 是 mergeGate。先大致分為三個 case 處理：

1. 至少一個 fanin 是 const0
2. 至少一個 fanin 是 const1
3. 兩個 fanin gate 相同。

mergeGate(unsigned, unsigned, bool) 須提供三個參數，第一個是要保留的 gate，第二個是要被 merge 掉的 gate，第三個 bool 是判斷 fanin 是否為 inverse。Merge gate 的方式為直接作 gate 的取代
如圖（舉 case3 為例）：



(3) CIRSTRash

我把作業七的 HashSet 改成 HashMap，用的 HashKey 是以 gate 的兩個 fanin 組成的 StrashKey。遇到可以被 merge 的 gate，用的是跟 Ciropt 一樣的 mergeGate() function。

(4) CIRSimulate

CirSimulate 的部分真的頗亂 QQ... 如果是 fileSim 的話，把 file 一行一行 input signal pack 成 size_t 的 parallel pattern，等於一次進行 64 組 pattern 的 simulation，若 file 的 pattern 不滿 64 組就補零。所以要 sim 幾輪就由 file 的長度決定。

舉例來說，一個 PI 所獲取的 signal string 長這個樣子：

[illegible]

這是一個由 `size_t` 然後 `bitset<64>` 轉換而來。

PI 的 SimVal 直接是 `_signal[i]`，然後的 SimOutStr 由 SimVal 轉成 `bitset<64>` 而來。AIG 的 SimVal 由兩個 fanin 做 `size_t` 的 bitwise and (`in0 & in1`)而來，PO 直接由唯一的 input 而來，只需判斷 input 是否為 inverted，要不要加 bitwise not 就好。

```
void
AIGGate::gateSim()
{
    size_t in0=this->faninlist[0]->getSimVal();
    size_t in1=this->faninlist[1]->getSimVal();
    if(this->faninID[0]%2!=0) in0=~(in0);
    if(this->faninID[1]%2!=0) in1=~(in1);
    size_t o= in0 & in1 ;
    this->setSimVal(o);
    string ss=bitset<64>(getSimVal()).to_string();
    reverse(ss.begin(),ss.end());
    this->setSimOutStr(ss);
}
```

進入 collect FEC group 的部分：

當 `circuit` 還沒被 `Simulate` 過，就呼叫 `getFEC()`，取得第一版的 `FEClist`。同樣使用 `Hash` 來抓出 `FEC group`，`SimKey` 是由 `SImVal` 組成，但是和 `Strash` 不同的是，在檢查時一次要建立兩個 `SImKey`，`k` 和 `not_k`，因為要考慮 `invert FEC pair` 的情形。

第一輪之後，一旦抓出一些 FEC 了，只要再從 FEClst 中檢查就好。清掉同一個 group 原本的 fec pair，再建立小小 Hash 細分 FEC group。把細分出的 FEC group 中最小的 gateID push_back 進 FEClst。

另外 randSIIm 同樣是一次餵一組 parallel pattern，signal 產生方法用 rnGen(INT_MAX)。

(5) **CIRFraig**

真的來不及寫了.....QQQQQQQ

• Performance

Optimize(sim09.aag)

mine:

ref:

```
fraig — fraig — 80×24
Simplifying: 3238 merging 13241...
Simplifying: 3243 merging 13246...
Simplifying: 3248 merging 13251...
Simplifying: 3253 merging 13256...
Simplifying: 3330 merging 13333...
Simplifying: 3335 merging 13338...
Simplifying: 3340 merging 13343...
Simplifying: 3345 merging 13348...
Simplifying: 3350 merging 13353...
Simplifying: 3355 merging 13358...
Simplifying: 3278 merging 13281...
Simplifying: 3283 merging 13286...
Simplifying: 3288 merging 13291...
Simplifying: 3293 merging 13296...
Simplifying: 3298 merging 13301...
Simplifying: 3303 merging 13306...
Simplifying: 3372 merging 13375...

fraig> uSAGE
Period time used : 0 seconds
Total time used : 0.04 seconds
Total memory used: 2.168 M Bytes

fraig>
```

```
ref — fraig-mac — 80×24
Simplifying: 3238 merging 13241...
Simplifying: 3243 merging 13246...
Simplifying: 3248 merging 13251...
Simplifying: 3253 merging 13256...
Simplifying: 3330 merging 13333...
Simplifying: 3335 merging 13338...
Simplifying: 3340 merging 13343...
Simplifying: 3345 merging 13348...
Simplifying: 3350 merging 13353...
Simplifying: 3355 merging 13358...
Simplifying: 3278 merging 13281...
Simplifying: 3283 merging 13286...
Simplifying: 3288 merging 13291...
Simplifying: 3293 merging 13296...
Simplifying: 3298 merging 13301...
Simplifying: 3303 merging 13306...
Simplifying: 3372 merging 13375...

(5) CIRFraig
真的來不及寫了.....QQQQQQ

fraig> uSAGE
Period time used : 0 seconds
Total time used : 0.01 seconds
Total memory used: 0.9688 M Bytes

fraig>
```

跑大型 optimize 時，時間差不多，但 memory 的用量差很多，可能我的 circuit 中有太多儲存各種資訊的 vector 有關

Strash (sim09.aag)

mine:

ref:

```
fraig — fraig — 80×24
Strashing: 3278 merging 3280...
Strashing: 3283 merging 3285...
Strashing: 3288 merging 3290...
Strashing: 3293 merging 3295...
Strashing: 3298 merging 3300...
Strashing: 3303 merging 3305...
Strashing: 3372 merging 3374...
Strashing: 2648 merging 3388...
Strashing: 2910 merging 3390...
Strashing: 3108 merging 3401...
Strashing: 3376 merging 3403...
Strashing: 3395 merging 3431...
Strashing: 3397 merging 3432...
Strashing: 3398 merging 3433...
Strashing: 3405 merging 3436...
Strashing: 3407 merging 3437...
Strashing: 3408 merging 3438...

fraig> uSAGE
Period time used : 0.01 seconds
Total time used : 0.04 seconds
Total memory used: 2.168 M Bytes

fraig>
```

```
ref — fraig-mac — 80×24
Strashing: 3278 merging 3280...
Strashing: 3283 merging 3285...
Strashing: 3288 merging 3290...
Strashing: 3293 merging 3295...
Strashing: 3298 merging 3300...
Strashing: 3303 merging 3305...
Strashing: 3372 merging 3374...
Strashing: 2648 merging 3388...
Strashing: 2910 merging 3390...
Strashing: 3108 merging 3401...
Strashing: 3376 merging 3403...
Strashing: 3395 merging 3431...
Strashing: 3397 merging 3432...
Strashing: 3398 merging 3433...
Strashing: 3405 merging 3436...
Strashing: 3407 merging 3437...
Strashing: 3408 merging 3438...

(5) CIRFraig
真的來不及寫了.....QQQQQQ

fraig> uSAGE
Period time used : 0.01 seconds
Total time used : 0.01 seconds
Total memory used: 0.9453 M Bytes

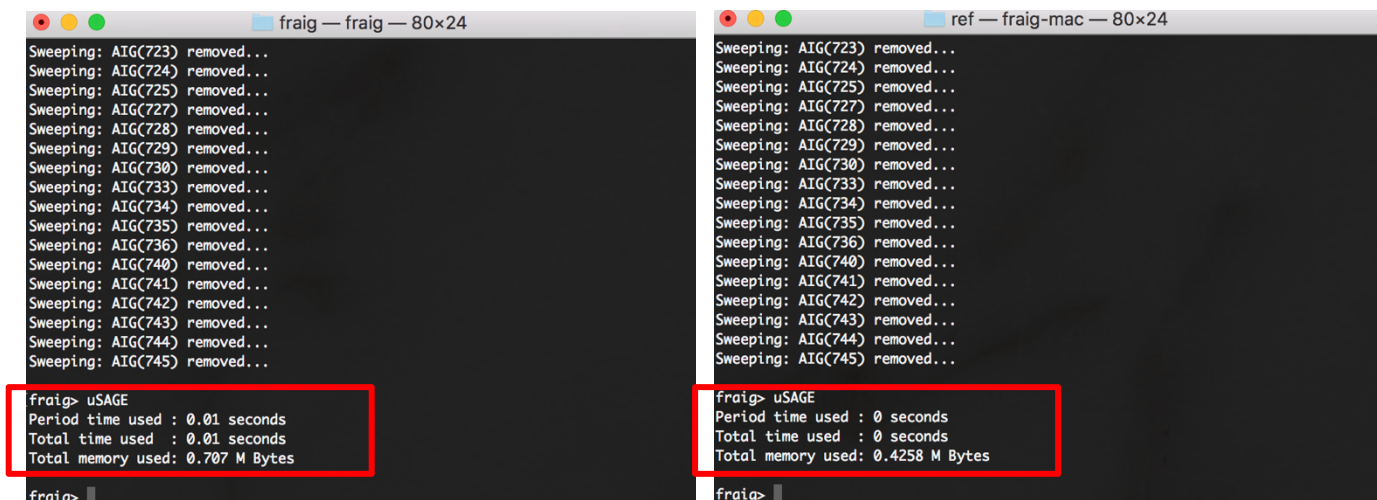
fraig>
```

跑大型 strash 時，時間差不多，但就是 memory 用量問題

Sweep (sim10.aag)

mine:

ref:



```
fraig — fraig — 80x24
Sweeping: AIG(723) removed...
Sweeping: AIG(724) removed...
Sweeping: AIG(725) removed...
Sweeping: AIG(727) removed...
Sweeping: AIG(728) removed...
Sweeping: AIG(729) removed...
Sweeping: AIG(730) removed...
Sweeping: AIG(733) removed...
Sweeping: AIG(734) removed...
Sweeping: AIG(735) removed...
Sweeping: AIG(736) removed...
Sweeping: AIG(740) removed...
Sweeping: AIG(741) removed...
Sweeping: AIG(742) removed...
Sweeping: AIG(743) removed...
Sweeping: AIG(744) removed...
Sweeping: AIG(745) removed...

fraig> uSAGE
Period time used : 0.01 seconds
Total time used : 0.01 seconds
Total memory used: 0.707 M Bytes

fraig>
```

```
ref — fraig-mac — 80x24
Sweeping: AIG(723) removed...
Sweeping: AIG(724) removed...
Sweeping: AIG(725) removed...
Sweeping: AIG(727) removed...
Sweeping: AIG(728) removed...
Sweeping: AIG(729) removed...
Sweeping: AIG(730) removed...
Sweeping: AIG(733) removed...
Sweeping: AIG(734) removed...
Sweeping: AIG(735) removed...
Sweeping: AIG(736) removed...
Sweeping: AIG(740) removed...
Sweeping: AIG(741) removed...
Sweeping: AIG(742) removed...
Sweeping: AIG(743) removed...
Sweeping: AIG(744) removed...
Sweeping: AIG(745) removed...

fraig> uSAGE
Period time used : 0 seconds
Total time used : 0 seconds
Total memory used: 0.4258 M Bytes

fraig>
```

跑大型 sweep，時間慢了一些

Simulation

再來就進入可怕的部分了.....

在 fileSim 時，gate 不多的時候，sim 的時間就很快跟 ref 差不多，但稍微多一些例如 sim12，就要跑個 8,9 秒，不用說 sim13，要跑個 4 分鐘 QQ

Sim 的速度真的很不行⊗

另外 randSim 在電路小的時候如 sim01~sim03 等，不需要 sim 很多 pattern 即可找出 fec group，但對於大電路，可能因為我終止條件 max fail time 太少（用的是 sqrt(AIG 數目）或是用 rnGen 產生的_signal 太類似（用_simLog 觀察），第一次 cirSim -r 的 pattern 都比 ref 少，抓出的 fec group 也較少。

• 瓶頸與心得

Simulation 的部分真的頗挫折...花了很多時間研究+寫，寫出的 sim performance 還是很差 QQ 但我實在是找不出原因，也許是在開 hash 時一直需要 copy 資料到新的 container 中，但想要改確有困難，因為原本的架構已定，很難只改某些東西，而不會出現 segmentation fault。至於 fraig 就完全來不及了...

在大學之前完全沒碰過程式，然後自認大一的時候計程學得不太好...所以本來是蠻排斥寫程式的，不過一整個學期下來真的覺得自己是有很大進步的，這門課學到很多！至少不會那麼害怕，也比較能夠自己寫出比較龐大的 function，雖然可能 code 很亂，performance 也還差很多...