DSnP Final Project (FRAIG)

B07901112 劉聿珉

一、實作目標

1. 繼承 HW6 的功課內容，將 aag 電路成功在電腦中建立。
2. 利用函式 sweep 將沒有連接到 output 的 gate 刪除以縮小電路。
3. 函式 optimize 的原理是根據 AND GATE 的性質可以列舉出邏輯運行上的四種特例，利用此性質進一步將電路縮小，其中，我們需要考慮的 gate 為 dfslist 中的 AIG GATE，因為若一 GATE 不存在於 dfslist 中，那必不會連接至 outpute，所以不需考慮。
4. Strash 功能是將兩 input 以及 output 相同的 GATE 融為一體，其中會運用到作業七雜湊表的觀念，將 GATE 有效分類及融合。
5. Simulate 的部分則是將 input 賦予真正的值，觀察電路運行後的結果，並在模擬中找出 FEC pair。
6. 在步驟 5.中，FEC pair 為我們觀察後的結果，此步驟 fraig 中，就是希望用大量的數據測試他們是否真的為一組 FEC pair。

二、實作進度

完整功能只寫完 sweep, optimize, strash。Simulate, fraig 的部分遇到一些瓶頸，應該沒有寫到可以評分的階段。

三、函數以及實作的方法

1. Sweep:

   說明：Sweep 這個 function 其實滿單純的，他的目標是將沒有連接到 outpute 的 GATE 刪掉，但其中不刪掉無用的 PI 以及 PO GATE。

   實作方式：

   (1).有鑑於他的目標，我們很容易就可以想到可以從 dfslist 作為一個參考(因為它求得的方式為從 PO 出發)。

   (2).在已寫好的 aiglist 中一個一個選出來做比對，若它不存在於 dfslist 中，就把它移除。

   補充：就上述方法，時間複雜度應為 O(n^2)，但是在我所建立的電路

中，GATE 有一個布林參數是儲存它是存在於 dfslist 中，所以在執行程式的時候我只需要走一次 aiglist 就可以將無用的 GATE 刪掉，時間複雜度為 O(n)。

2. Optimize:

說明：本人寫起來覺得有點像複雜本的 Dlist，比作業多了判斷 GATE 優化的類別，其他將指標打開在接起來的步驟其實跟 Dlist 有異曲同工之妙。

(1).Optimize 最重要的部分為判斷該 GATE 適用於何種優化的方法，我將優化方法分成了三種(定義 fanin[0]為左腳，fanin[1]為右腳)：

1.取決於右 fanin(Depend on right)：

此種狀況出現在，左腳如入恆為 1 的狀況，因為此電路的 GATE 為 AND GATE，所以只要右腳輸入為 1 輸出即為 1，輸入 0 則為 0。

還有一種況為兩腳都接在同一 GATE 上，兩腳的 invert 相同，這時輸出值器可以取決於右腳也可以取決於左腳，但為了簡化問題，所以將它歸為這類。

2.取決於左 fanin(Depend on left)：

此種狀況出現在，右腳輸入恆為 1 的狀況，理由如上。

3.輸出恆為 0 (Condratiction)：

這種輸出，出現在左右腳「任一腳」為 0 狀況。此外若左右腳所接的 GATE 為同一個，且 invert 的狀態相反，此狀況輸出也必為 0。

```
Optimizegate
CirMgr::opttype(CirGate* gate)
{
   assert(gate->_gateType == AIG_GATE);
   size_t t0 = gate->_fanin[0].gate()->_var ,t1 = gate->_fanin[1].gate()->_var;
   if ((t0 == t1) && (gate->_fanin[0].inv() == gate->_fanin[1].inv())) return DependOnRight;
   else if ((t0 == t1) && (gate->_fanin[0].inv() != gate->_fanin[1].inv())) return Cond;
   else if ((t0 == 0) && (gate->_fanin[0].inv())) return DependOnRight;
   else if ((t1 == 0) && (gate->_fanin[1].inv())) return DependOnLeft;
   if ((t0 == 0) && (gate->_fanin[0].inv() == false)) return Cond;
   if ((t1 == 0) && (gate->_fanin[1].inv() == false)) return Cond;
   return Nothing;
}
```

(2)連接：

連接的方法與 Dlist 類似，從 dfslist 中的第一個開始做優化，若該 GATE 符合上面所說的分類則進行重新連接該 GATE 的 fanin 以及 fanout，在連接完畢後便將該 GATE 從 dfslist 中刪除。

3. Strash:

說明：Strash 的 unordered_map 就是 HW7 的應用，利 unordered_map 找尋資料時間複雜度為 O(1)的特性，快速的找到資料，如此就不用一個一個比對了 O(n^2)。

(1).建立 unordered_map：

在此建立 unordered_map 其實有兩種方法(我都寫過)，這個部分會在「所遇問題及解決方法」中深刻討論。

(2).尋找：

從 dfslist 中從「最靠近 PI」的 GATE 開始在 unordered_map 中尋找，若沒有找到對應的 GATE 則將之存入 unordered_map 中，若有則進行融合的動作。

(3).融合：

跟 Optimize 的動作非常像，就是將後查詢到 GATE 的 fanout 加入已存在並且等價的 GATE 的 fanout 中，之後同 optimize，將該 GATE 刪除。

四、所遇問題及解決方法

1. 在 Optimize 階段所遇到的問題：

Optimize 所做的事情其實算單純，只要找到要優化的 GATE 並且正確的將其上下的 GATE 連接起來就可以成功寫完了，但是我在寫的時候忘記將 GATE 從他的 fanin GATE 的 fanout 中拔掉，以至於我之後運行時跑的速度變很慢，而且有些狀況，將其修好 Optimize 的功能就可以正常運

行了。

在 optimize 的部分也也跟同學討論過這項關的問題，同學主張在每一個 GATE 優化之後都要去檢查它 fanout 的所有 GATE 是否可以優化，如果可以便要將其優化。在此時實作的方式則為用 recursive 重複呼叫，直到沒有任何一個 GATE 需要被 optimize 為止，才可以換下一個 GATE。但是經過討論後發現，其實只要由 dfslist 的順序進行優化便可以得到正確的結果了，因為之所以需要用 recursive，是想避免如果上面的 GATE 已被優化，再將下面的 GATE 優化會進而影響到上面的 GATE 使得它可以再次被優化，但是 dfslist 所形成的排序便是從下到上，所以並不會有上面所述的狀況發生，所以只要順著 dfslist 的順序便可以得出理想的結果。

2. 在 Strash 階段所遇到的問題：

   這部分我在寫的時候因為一些問題所以我寫過兩種寫法，我想要分別討論。

   (1).第一種：

   宣告方式：unordered_map<size_t, CirGate*>

   設計理念：每一個 GATE 的兩個 fanin 有獨特的位置，只要多考慮兩隻腳 inv 的問題就可以製造出一個獨特的 size_t，使得該 GATE 在 map 中可以被查詢及判斷是否為等效 GATE。此為我產出 size_t 的函數，將 pointer 強制轉為 size_t 並且加上 inv 的變因。但是這個函數其實並不夠好，因為在大量的測試下，發現還是有機會會產出一樣的 size_t 但是卻並應該被消失。所以我就嘗試了第二種寫法。

```
size_t
CirMgr::ptrtosize_t(CirGate* gate){

  size_t tt = 0;
  vector<CirGateV>::iterator iter;
  for (iter = gate->_fanin.begin();iter != gate->_fanin.end();++iter){
     tt += (size_t)iter->gate()>>(size_t)iter->inv();
  }
  return tt;
```

(2).第二種：

發想：如果沒辦法寫出一個夠好的函式來產出獨特的 size_t 以供辨別，

那就直接告訴 unordered_map 那些狀況式一樣的 GATE 需要被化簡就好

啦！

所以我就...

```
struct STRCirGate
{
  size_t fanin1;
  size_t fanin2;
  bool inv1;
  bool inv2;
  STRCirGate(size_t f1, size_t f2, bool i1, bool i2){
   fanin1 = f1;
   fanin2 = f2;
   inv1 = i1;
   inv2 = i2;
  }
};
struct STRCirGateHash
{
  size_t operator()(const STRCirGate& str) const{
    return ((size_t)str.fanin1 + (size_t)str.fanin2)/8;
  }
};
struct STRCirGateComp
{
  bool operator()(const STRCirGate& fanina, const STRCirGate& faninb) const {
    if ((fanina.fanin1 == faninb.fanin1) && (fanina.inv1 == faninb.inv1) &&
        (fanina.fanin2 == faninb.fanin2) && (fanina.inv2 == faninb.inv2)) return true;
    if ((fanina.fanin1 == faninb.fanin2) && (fanina.inv1 == faninb.inv2) &&
        (fanina.fanin2 == faninb.fanin1) && (fanina.inv2 == faninb.inv1)) return true;
    else return false;
  }
};
typedef unordered_map<STRCirGate, CirGate*, STRCirGateHash, STRCirGateComp> umap;
```

直接將 unordered_map 重寫，這樣他就可以自己判斷兩個 GATE 是否一

樣，如果一樣就會回傳了。

(3).比較討論：

這兩種寫法個有優缺點，第一種一定有更好的函示寫法但是我當時想不太出來，所以只寫出這樣暫時的版本，雖然還是有機會重複，但是已經可以避免大多數的情況。(在寫完之後有些想法，如果可以將兩個 fanin 的 ID 轉成二進位再多加上前面一位表示是否 invert，將組成的數字小的放前面，大的放後面，這樣便可以組出一個獨特的二進為數字，前後半分別表示兩 fanin 並且可以從中得知是否 invert 的特性)

第二種方法雖然比較聰明且一勞永逸，但是這也是因為 AND GATE 只有兩個 fanin 而已所以比較好寫，如果未來碰到需要考慮很多例外或是需要討論狀況，這樣它本身需要討論的條件便會多很多，撰寫的時間也會變長，有 bug 的機會也會變多，所以我想這兩種方法可以說是個有優缺吧。

五、心得

就這次的 final project 我覺得真的非常的可惜，雖然在考期末之前就有先做了一點功課，但可能是因為我在上 DSnP 這堂課之前其實並沒有 C++的基礎，所以對我來說這份 project 真的難了一點，在連續住了系 k 三天喝了不知道多少罐的提神飲料的努力後，還是並沒有將這次的作業寫完，但是做為一個零基礎的初學者我想我可以撐到現在不停修真的算是一個突破吧。

上 DSnP 這堂課真的讓我大開了眼界，回想一年前修完計算機程式便以為自己已學會 python，現在回首一看，覺得當時的自己真的是可笑無比，code 沒有整理系統化、沒到不得已不用物件導向、所有的事都想在一個 file 裡面解決，這些都是那時的壞習慣，現在經過這堂課的洗禮，終於學會了如何處理較大的 project，以及分門別類將 code 整理乾淨，以及寫出一個他人也看得懂得 code，在這堂課中真得學到了不少，好久沒有那麼喜歡上一堂課了。

在當時選課的時候就有聽學長說這堂課是一個猛讀，會好好得考驗你得意志力，是否真得是喜歡寫程式，或真的適合走軟體這條路，我想我上完這堂課大概對自己有個底了。雖然我 HW 的分數一直都沒有很高，但是我很享受每一次跟同學討論，以及寫作業的過程，由其實是寫出來 de 出 bug 的那份喜悅真的是讓人難以忘懷，我想我的未來現在還說不定，但是我想我之後一定會多多往軟體程式這方面涉略，雖然或許在人生的路上不會同儕還要厲害，但是至少我是在做我覺得開心的事情。

最後，想感謝，各個助教們，當時因為作業二有問題所以跟助教有討論了一段時間，覺得你們真的是很認真，而且花很多時間在我們的作業上，我想，這堂課可以受到那麼多系所的同學喜歡，您們也是不可或缺的功臣之一，真的非常非常感系你們。

六、參考

我作業六的部分是參考電機二朱哲廣的 code。