DSnP Final Project Report : FRAIG

姓名：林承德

系籍：電機二

學號：B05901064

E-mail：b05901064@ntu.edu.tw

1. **Class定義及其功用**

在這一段落中主要是講解class裡面幾個重要或特別的data member及 memner function的功能及用處。有些有關Command的方面會移到下一段落介紹。

1. **CirGate及繼承CirGate的 \*Gate**

在設計CirGate時，我只有將大家都需要的變數（\_gateNo、\_flag…等）放在CirGate中，其他變數（\_fanout、\_fanin…等）則是宣告在各自的繼承class，可以節省記憶體使用量。

另外我在CirGate寫了許多存取繼承class變數的vurtual function（getFanin()、getFanout()…等），可以在CirGate讀取到那些變數，方便在CirGate的member function寫完大家都要做的事情（printGate()、reportFanin()…等），方便之後的維護。

1. **size\_t CirGate::\_flag**

此變數有兩種功能，一個是在跑DFS時與CirMgr::\_flag比較確認是否跑過一次，另一個是在FecPair::removeOutDfs()時確認gate是否在dfs的範圍中。

1. **size\_t CirGate::\_dfsPos**

在每次cirMgr執行setDfs()更新DFS 順序時會將此gate在vector的第幾個位置寫在此，供fraig時判斷兩個gate間是誰要merge誰。

1. **static const vector<unsigned> CirGate::nullVector;  
   static const string CirGate::nullString;  
   virtual inline const vector<unsigned> Cirgate::getFanin() const  
   virtual inline const vector<unsigned>& Cirgate::getFanout() const  
   virtual inline const string& CirGate::getName() const**

為求效率，我盡量讓string及vector的getter function是pass by reference。但有些derive class並沒有string或是vector可以回傳，故需要static的nullVector / nullString供他們回傳。

另外，由於fanin並不是以vector的方式儲存，故沒有reference可以回傳，只能用pass by value多複製一次的方式回傳。

1. **FecPair\* AigGate::\_fec  
   FecPair\* ConstGate::\_fec**

指向含有該Gate的 FecPair，讓gate在尋找其FecPair的速度降為O(1)。

1. **void CirGate::connectGate(CirGate\*\*)**

在cirMgr::readCircuit()時用來連結gate與gate之間，設定所有gate的fanout。如果有一個gate的fanin未被定義，則會在傳入參數CirGate\*的array相對應位置創造一個undefine gate。

1. **vector<unsigned> CirGate::remove()  
   vector<unsigned> CirGate::replace(unsigned var)**

此function只會當作cirMgr::removeGate(unsigned, vector<unsigned>)的傳入參數。這兩個function會把gate的fanin / fanout相對應關係進行調整，調整完之後可以直接讓cirMgr delete這個gate。

Function回傳它的fanin給cirMgr，讓他去檢查fanin是不是變成 defined but not used，如果是的話就將其加入 cirMgr::\_notUsed。

1. **CirMgr**
   1. **CirGate\*\* CirMgr::\_gates**

儲存所有CirGate\* 的array，以hash的概念儲存，第i號gate就放在\_gates[i]的位置（i.e. hash key = gate Number），讓getGate(unsigned)速度壓在O(1)。這樣可以讓其他地方只要儲存gate number或是literal ID就好，節省空間使用量。

* 1. **Vector<unsigned> CirMgr::\_notUsed**

存放 defined but not used gate的gate number。這樣在sweep時可以不用run全部的gate，在cirprint –fl時也不用重新跑一遍。

* 1. **Vector<FecPair\*>\* CirMgr::\_fec**

存放所有的FecPair。平時並不會對其資料進行排序，只有cirp –fec時才會進行排序。

會用pointer的原因是我的FecPair在simulate完生成下一代FecPair時需要將新的一代放進新的vector，再將其取代原本的vector。為了不要進行複製消耗時間，我將其令成pointer讓他可以不用複製直接使用。

* 1. **size\_t CirMgr::\_flag**

與CirGate::\_flag一樣，用來確認gate是否已經跑過DFS，以及在FecPair中確認gate有沒有在DFS的順序中。

* 1. **void inline CirMgr::reportFanin(int, CirGate\*)  
     void inline CirMgr::reportFanout(int, CirGate\*)**

由於我的flag標準放在CirMgr中，故我讓Cirgate –fani/fano function改成呼叫cirMgr，先++CirMgr::\_flag再跑CirGate::reportFanin/reportFanout確認其是否已經跑過。

* 1. **void CirMgr::setDfs()**

此function會重新跑一次DFS，更新CirMgr::\_dfs及CirGate::\_dfsPos。如果有Fec Group存在則讓他們確認自己group內的gate還在不在DFS順序裡，不在就拿掉。如果刪完有一個FEC只剩下一個gate或是沒有gate，則將其刪除。

* 1. **void CriMgr::removeGate(unsigned, vector<unsigned>)**

此function的vector<unsigned>參數必是CirGate::remove()或是CirGate::replace(unsigned)，先讓CirGate處理好gate之間的交互關係，再delete gate本身，實現OO的精神。

1. **FecPair**
   1. **unordered\_set<unsigned>\* FecPair::\_hash**

主要用來儲存gate number的container。用hash的好處是在insert、find和erase都可以做到O(1)的速度。用pointer的原因是要避免一直複製，拖垮效率。

* 1. **vector<unsigned> FecPair::\_vec  
     bool FecPair::\_isVecSet**

\_vec是主要是用來print的container。平常完全不維護也不放入資料，只有要print時才會從hash複製一份資料過來並排序。\_isVecSet則是記錄\_vec是否是設定好的狀態。

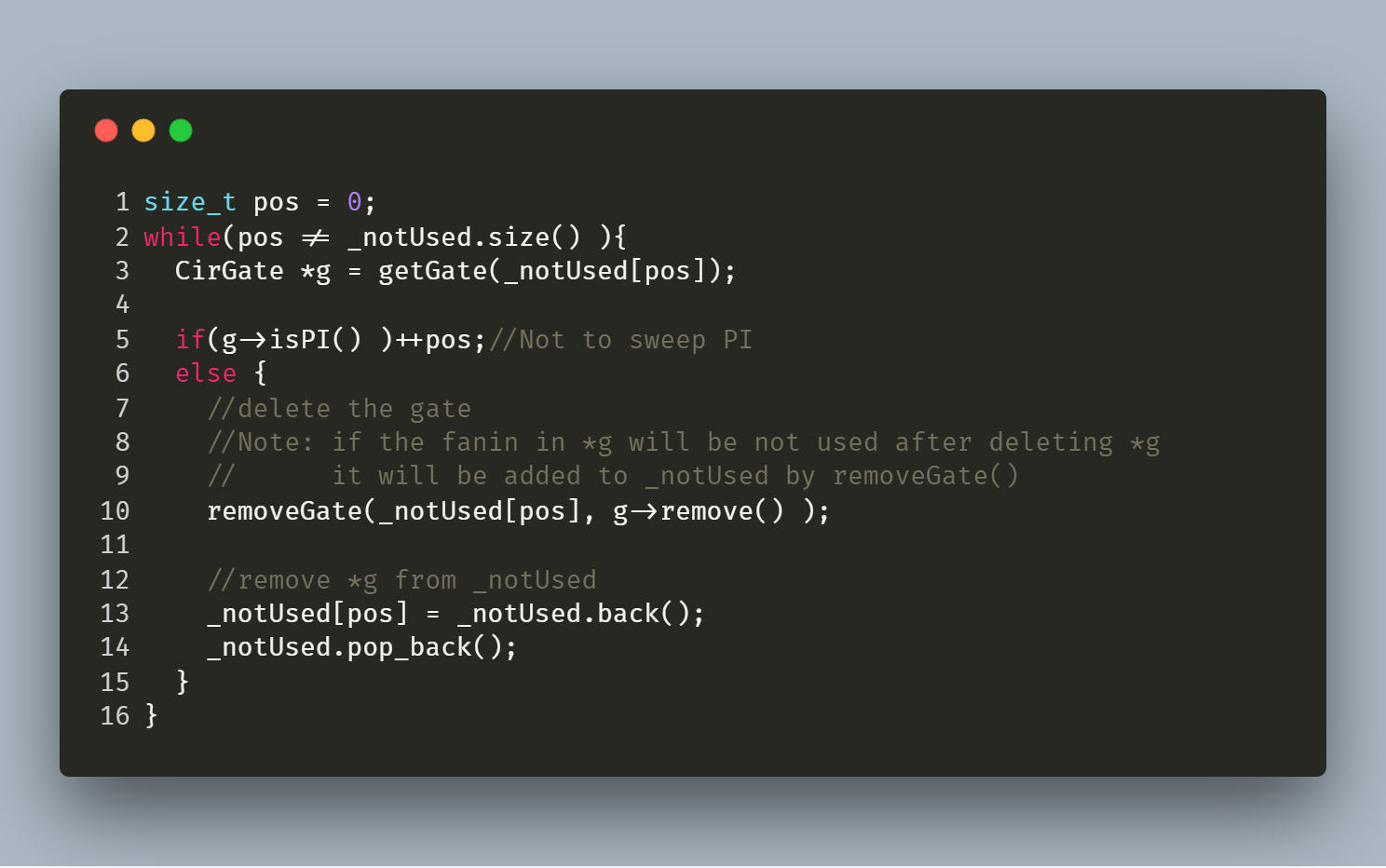
* 1. **int FecPair::\_min  
     int FecPair::\_dfsMIn**

用來記錄gate number最小值以及dfs最前面的gate的number，每次insert都會檢查這個質是否需要修改。如果最小值被移除則會讓其值變為-1，等待下次需要或是檢查gate是否在DFS中時重新搜尋。

* 1. **void FecPair ::generate(vector<FecPair\*>\*)  
     bool FecPair ::\_first  
     void FecPair::generateFirst  
     (vector<FecPair\*>\*, unordered\_map<size\_t, FecPair\*>&)  
     void FecPair::generate  
     (vector<FecPair\*>\*, unordered\_map<size\_t, FecPair\*>&)**

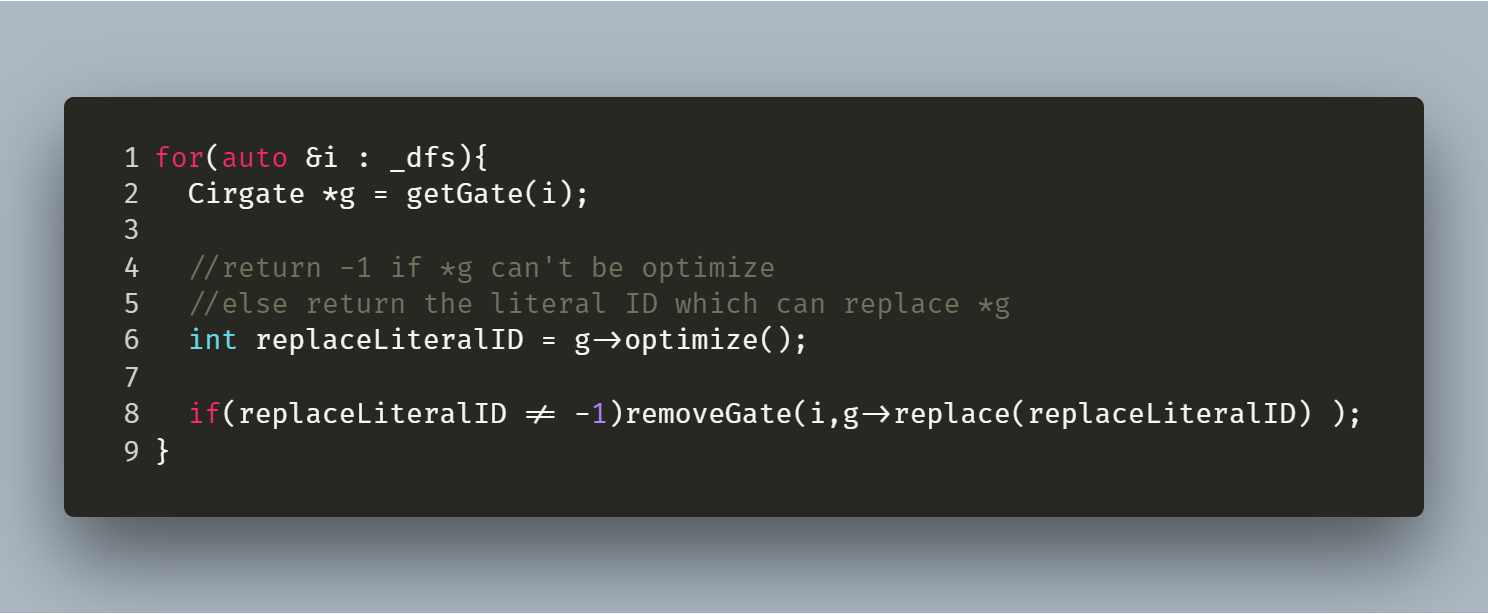
FecPair用來產出下一代Fecpair並推到新的vector<FecPair\*>\*中。利用hash把具有相同value的gate加到同一個FecPair。

\_first則是記錄這是否一開始初始化的FecPair，如果是的話要執行gererateFirst將具有相反value的FecPair合在一起。

1. **Commands 實作方法及細節**
   1. **CirSweep**

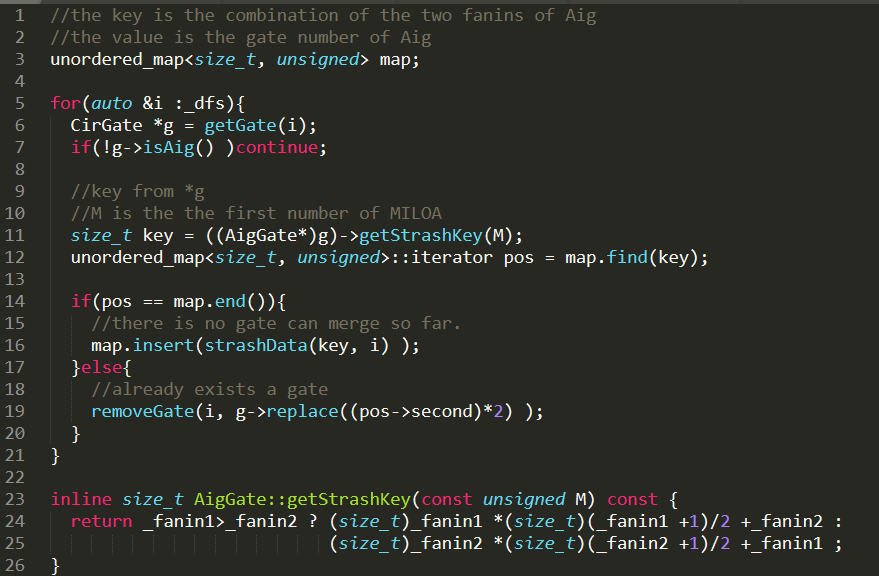
\_notUsed會在readCircuit()時一起建好，又在刪除gate時如果產生新的defined but of used gate會被removeGate()自動放入，故只要一直針對\_notUsed刪到剩下PI就算結束了。

它的好處是不用跑完全部所有的gate就可以完成，對於大量的AIG的circuit有所幫助。

* 1. **CirOptimize**

CirGate ::optimize()是一個virtual function，除了Aig以外的gate都會return -1，Aig則是會藉由他的兩個fanin判斷它是否可以被其他literal Id代表的value取代。

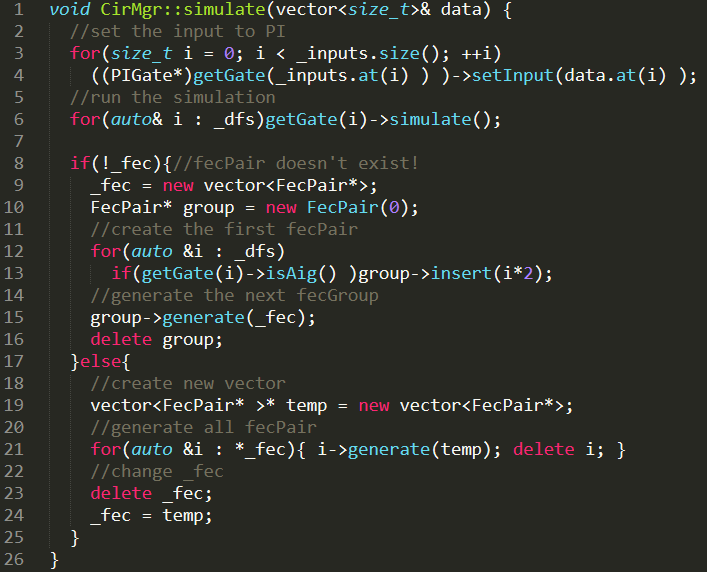
由於一個gate經由optimize被merge後只會影響其fanout optimize的結果，故我照DFS順序進行簡化，避免遞迴呼叫拖慢速度。

* 1. **CirStrash**

藉由hash的幫助，我可以在O(1)的速度下找到是否有gate可以跟自己合併。

由於一個gate經由optimize被merge後只會影響其fanout optimize的結果，故我照DFS順序進行簡化，避免遞迴呼叫拖慢速度。

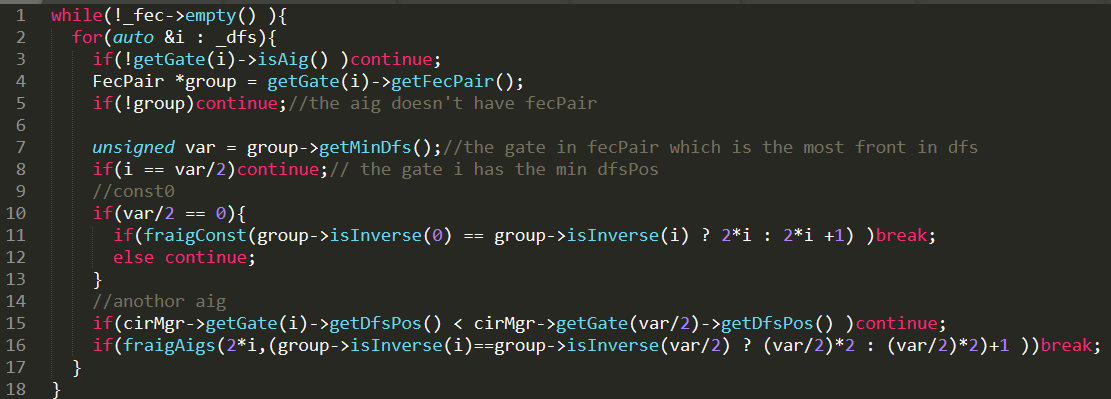
在hash key部分，雖然運算速度比直接把兩個fanin前後接在一起還慢，但我的key可以妥善運用size\_t的每一個bit，可以表現出264不同種fanin的組合狀況。

* 1. **CirSimulate**

data是一個size跟PI數量一樣多的vector，data[i]是在simulation時給第i個PI的值，randomSim()和fileSim(ifstream&)皆是用他們自己的方法生出data並呼叫此function。

CirGate ::simulate()是每個gate根據其fanin現在的值決定自己的值，只會影響到fanout的反應。故我在給完所以PI值之後，照DFS的順序跑simulate，避免遞迴呼叫。

在FEC的部分，如果還沒有產生的話，就先創一個fecPair把所有在DFS的AigGate和Const0丟進去，再針對那個新的fecPair進行generate。如果已經存在fecPair，就把每一個fecPair都拿去generate。最後拿到新的vector<fecPair\*>\*再取代原本的\_fec。

* 1. **CirFraig**

我的fraig演算法主要概念就是拿兩個gate在DFS List裡面越前面的來用SAT證，因為SAT證明的複雜度遠大於其他動作，故我認為盡量減低SAT的input比較重要。DFS List在越前面的話，他fanin直到PI間有比較少的gate，SAT證明會比較快。另外，如果fecPair裡面有Const0的話要優先拿出來處理，因為0跟gate進行XOR就是gate本身，不用多花gate數量去建造Const0。

如果SAT證明出來他們兩個是一樣的，就依照DFS順序，在前面的並調在後面的，減少電路複雜度也避免迴圈產生。之後重新設定DFS list，再跑一次fraig。如果SAT證明他們不一樣，就直接拿那個解進行simulation把那兩個gate分開，然後繼續照著dfs list繼續跑下去。一直跑到\*\_fec這個vector變成空的為止。

1. **Performance**

簡單來說，我的fecPair在處理generate的時候花太長的時間了。

ref在我的電腦上sim13.aag中simulate pattern.13花了1.7秒左右，而我的fraig需要跑到三秒多才結束。但是當我把處理fecPair這段code拿掉後，我只需要0.56秒就simulate 完 pattern.13了。fecPair佔simulate超過八成，效率不佳。而我的fraig每次只要證明出兩個gate不一樣時就需要simulate一次、fecPair generate一次，這樣累積下來可能導致拖慢fraig的時間。

另外，我原本假設DFS List越前面的gate『大致上』越好證，因為他被期望在證明時有比較少的gate需要建立。但實際上有時候非常不準，也會影響到我fraig的效率。

我自己目前想到的改善方法有幾個。一個是想辦法加速fecPair的速度，看是要減少new delete的次數或是用記憶體來換取更多時間；一個是在fraig時不要拿到一組data就simulate一次，可以等拿到多一點key再來simulate，減少fecPair generate的次數；或者是改變fraig的策略，變成從PI開始往fanout出發……等。

1. **Obstable**
   1. **cirFraig策略**

我原本是設計另外一種fraig的演算法（在cirFraig.cpp #ifdef FRAIG\_MY\_VER）。他的前提是SAT的證明速度只與SAT裡的PI數量有關，以及證明DFS list越後面的gate效果越大。但他在處理sim13.aag兩倍大的電路時直接卡住，在前10個gate就證明了10分鐘還證不完。經過開gdb後，我認為SAT的證明速度可能不是跟PI有關，而是跟全部gate的數量有關。故我才改成上面那種演算法。

* 1. **aag file裡的電路型態**

不管是optimize、strash、還是fraig，每一種演算法都可以處理某一種形狀的電路，卻在另一種電路中表現十分低落。在測資有限的情況下，這讓我感到十分猶豫，不知道哪一種作法才是平均來說最好的。

1. **心得**

這份final project給我們很大的自由度，讓我可以自由的設計我的架構，但也是這樣才會令人更頭痛。我必須一直思考架構好不好寫、效率如何、占用的記憶體多少、好不好維護......等問題，這些也是我之前都沒有認真想過的問題，真的是要發揮腦袋的極致才能打出一份讓自己滿意的code。

這堂課讓我對於電腦和C++有更進一步的了解，透過一次次的作業的訓練讓我們知道要怎樣才可以打出一份好的code，而不是可以跑就好。我在打final的時候把hw6的部分重新打了一次，邊打遍覺得自己之前怎麼可以打出這麼奇怪的code，想必是有多一點成長了吧。

謝謝教授及助教給我這個機會練習我的技巧，我在這堂課上的非常開心XD。