資料結構 Final Project

B06901049 林泓均

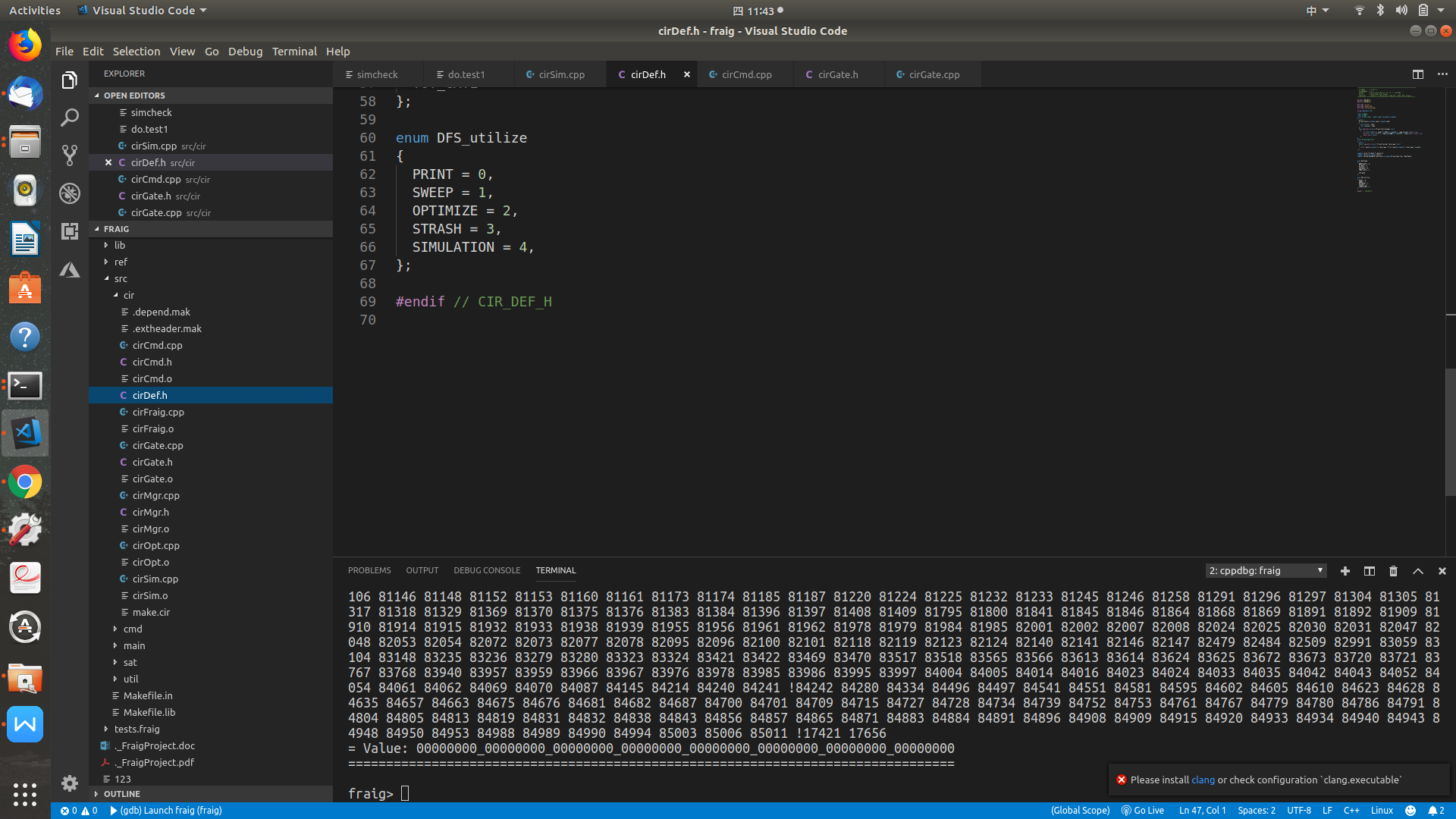
聯絡方式:0905729688

1. 電路基本架構

本次電路使用B06901048陳昱行同學的HW6，fraig部份(即期末專題部份)則是自行實做。

在CirMgr中，將所有data member放入ParsedCir的class中，並利用Circuit存取。

在CirGate中，fanin(unsigned\*)及fanout(vector<unsigned>)皆是存literal，這樣子在進行DFS的時候有許多好處，不必多存許多額外變數。

另外，只要進行DFS，在程式中就是呼叫同一個函式。因此我額外在CirDef.h中寫了一個enum，看是要進行什麼用途，那麼在進行DFS時就會執行不同的動作。

1. 本次實做函式內容及巧思

**1.SWEEP:**

**基本思路：**在進行DFS之後，建立DFS\_List，刪除掉所有不在DFS\_List中的gate。

**使用技巧：**為了節省空間，因此我實際上是使用了vector<bool>來儲存是否跑過DFS\_List。因此在進行完DFS之後，只要針對跑完還是對應false的gate進行刪除即可。

**2.OPTIMIZE：**

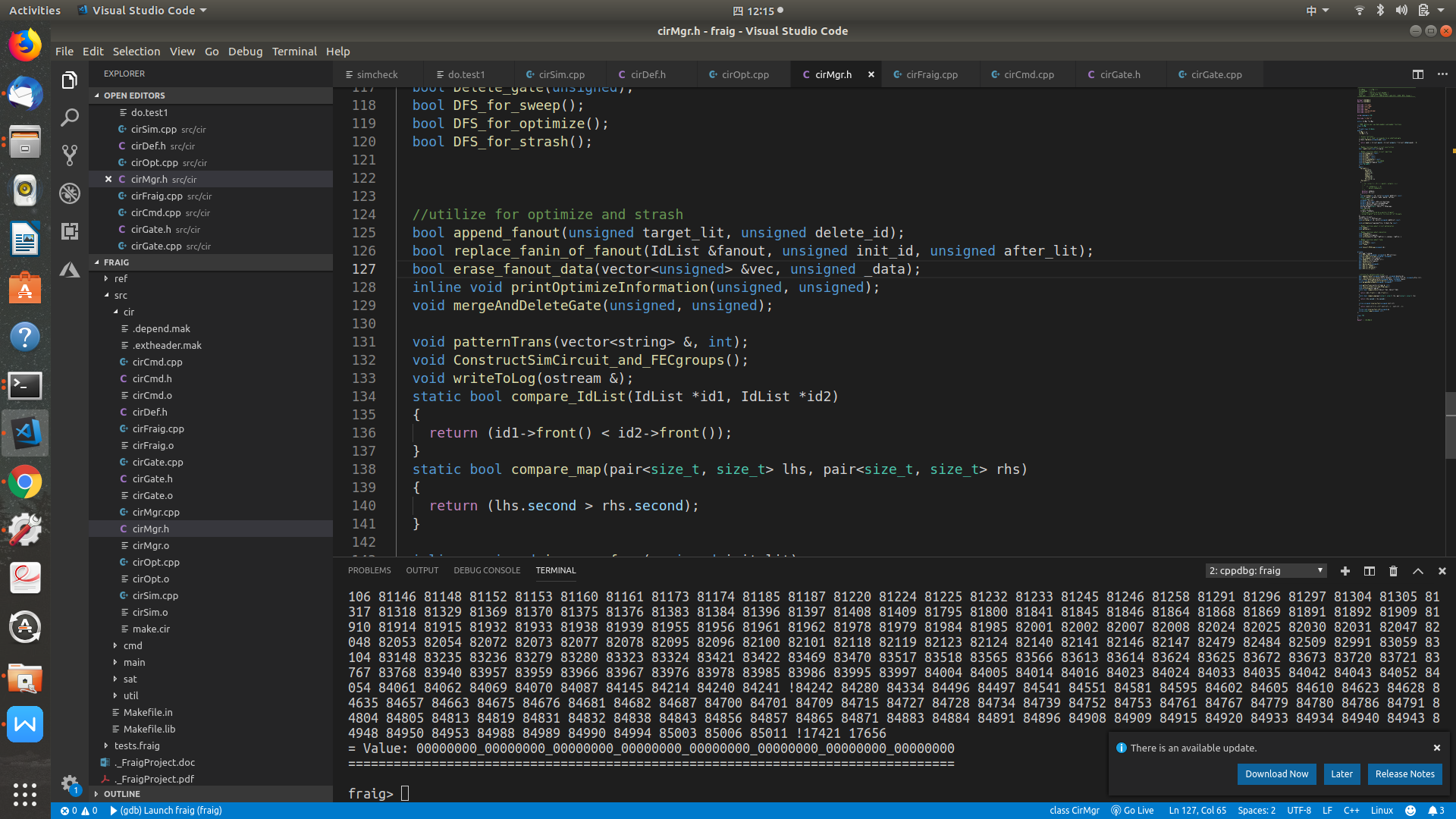
**基本思路：**進行DFS之後，針對有在DFS\_List中的gate檢查其fanin是否有以下4個狀況:

1. 其中一個input為0
2. 其中一個input為1
3. 2個fanin來自相同的gate且訊號相同
4. 2個fanin來自相同的gate但是訊號相反

若有的話則進行化簡，到跑完DFS\_List為止。

**使用技巧**：為了因應刪除gate所具備的情況，我寫了3個泛用的函式:

（\_lit表示傳入的變數是literal，\_id表示傳入的變數是id，可增加可讀性，避免搞混）



其中**append\_fanout**功用為：

將target\_lit對應到的delete\_id給刪除，並加上delete\_id的fanout中所有的lit。

**Replace\_fanin\_of\_fanout**:

傳入欲刪除id的fanout\_list，並將各fanout中的fanin刪除原本的id換成opt之後的訊號。

**Erase\_fanout\_data**:直接刪除在傳入的fanout中的\_data。

（三者bool皆回傳成功與否）

因此，舉例而言，對兩個input都相同的case而言，我只要執行如下動作:

replace\_fanin\_of\_fanout(fanout,目前id,fanin的lit);

append\_fanout(fanin的lit,目前id);

即可完成。

1. **Strash**

**基本思路：**在進行DFS時即檢查（只檢查AIG\_GATE）id的fanin是否和目前有找到的相同。若無，則用fanin當key，id當value存入unordered\_map中。若有，則執行mergeAndDelete的函式進行合併。

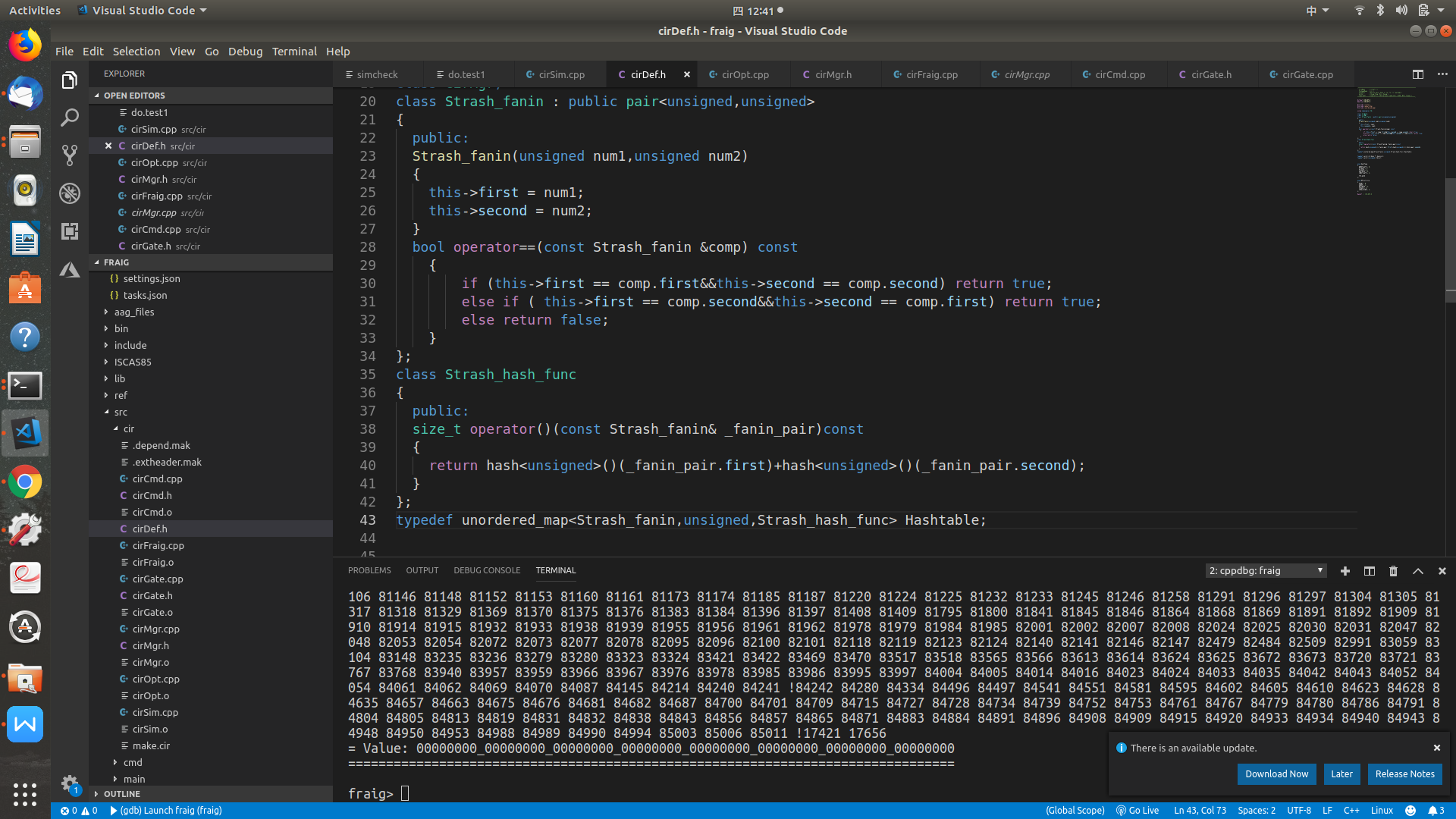
**困境**：

(1)比較fanin時，(a,b)和(b,a)需要視為相同，然而若只用pair<unsigned,unsigned>資料結構的話，那麼他的==並不會將上述狀況列為相等。

(2)若要使用pair作為key，那麼必須重寫hash\_function，因為unordered\_map並沒有內建。

**解決方法**：(1)利用繼承的手法直接overload ==的operator。這是我首次學到原來可以直接繼承STL的資料結構並進行修改。

1. unordered\_map是可以自己寫hash\_function的。因此我的作法是將兩個fanin個別產生的hash\_func加起來。因為unoreder\_map對int、unsigned等基本的data\_type已經有內建hash\_func，而hash\_func只要return size\_t即可，就算因為拿原本的直接相加而溢位也只會再從0(size\_t特性)開始，因此不必擔心這個問題。



**使用技巧**：有些人會用IdList作為value，等到做完DFS之後再檢查是否有重複。但是這樣子做會浪費不必要的記憶體。所以若是在做完DFS時直接進行strash就可以同時節省記憶體和時間。

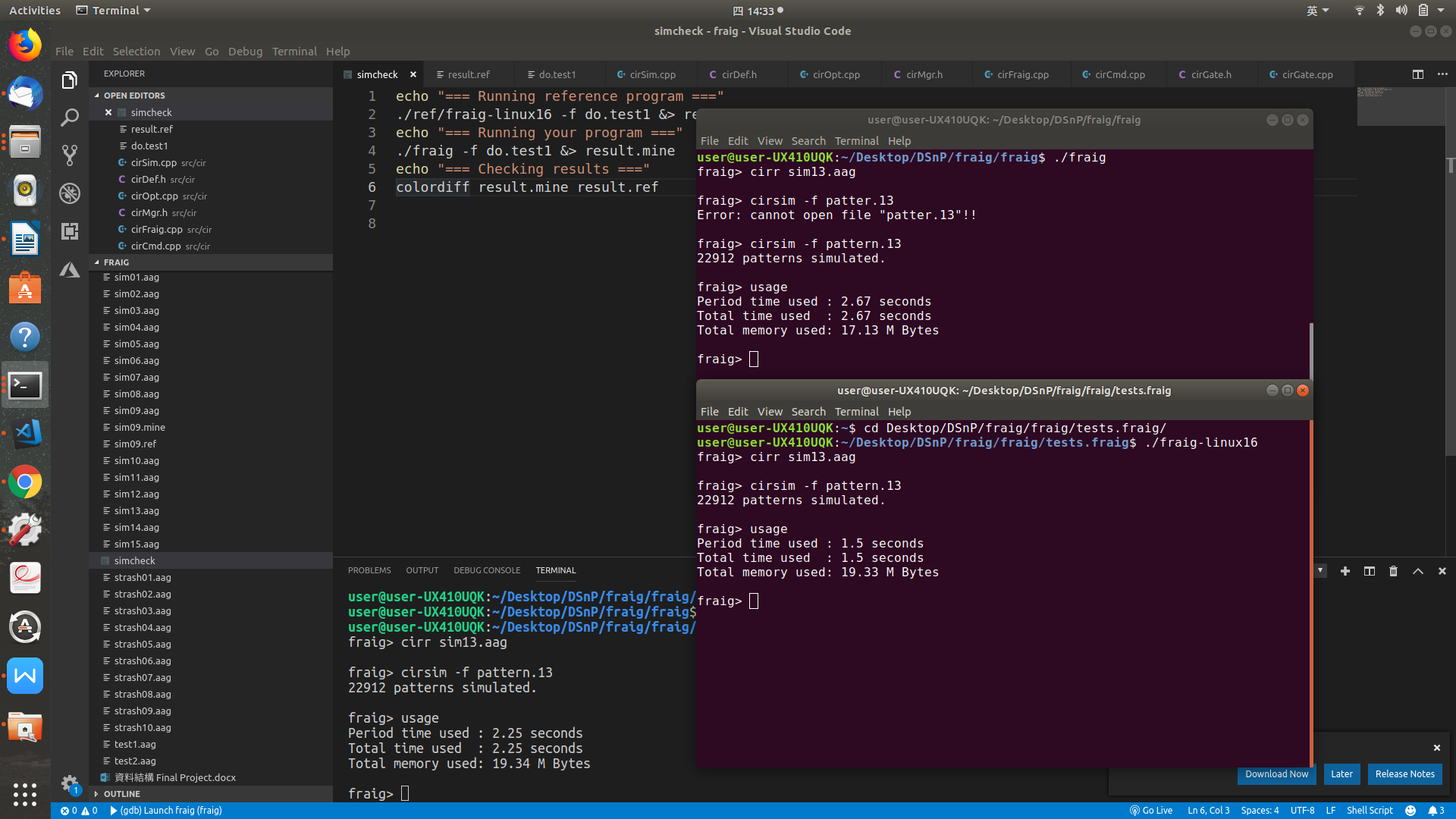
**4-1.SIM -file**

**基本思路：**基本上，SIM的思路可分為以下四大部份：

1. 決定PI訊號
2. 建立SIM過後各個gate的pattern
3. 從那些pattern建立FECGroups
4. 對FECGroups進行整理

其中1.的部份是-r和-f的主要差異。讀檔時，若已經讀了64次即進行步驟2、3。直到結束為止再進行步驟4。

**使用技巧**：(1)利用size\_t儲存給予PI訊號(test\_pattern)，可一次進行64bit模擬，而模擬電路時只要使用~、&等基本的operator即可完成。

(2)儲存資料方面，我選用vector<IdList\*>儲存不同的FECGroup。因為第一次寫完sim時，sim13總共花了1400秒左右。因此我將所有原本使用IdList的部份全部改為IdList\*。在改為使用pointer後，速度甚至可達reference code的2倍，且使用記憶體更少。

(3)除了利用vector<IdList\*>之外，我還額外用了unordered\_map<size\_t,IdList\*>去檢驗各個FECGroup的元素（gateid）是否還在原本FECGroup。如果沒有，則開一個新的IdList給他存入hashmap。最後若這個IdList的size()>1，則把她存到一個新的vector<IdList\*>。做完所有FECGroup之後，就可以把舊的vector<IdList\*>清掉，並直接用old\_vec = move(new\_vec)。

**4-2.SIM -random**

在這個部份，只要利用random的部份給予PI模擬的pattern即可。值得一提的是，這部份我是使用<random>中的mt19937\_64來做為64bit的亂數產生器。不只比rand()有效率，產生的亂數品質也較好。sim\_pattern的數量則是由MILOA中的M決定。若M太大則生成較少的sim\_pattern(如:0.95M~1.25M)以爭取時間。並且用unif\_int\_distribution<size\_t>決定上下界中的隨機測試值。

**4-3.SIM Cirg**

在Cirg的指令中，我們需要print出他的Fec。由於FECGroup的資訊存在CirMgr中，但是reportGate的函式卻是在CirGate之中，因此這個部份我選擇更改CirCmd的函式，讓reportGate可以傳入CirMgr\*的變數，方便CirGate可以呼叫CirMgr中的函式。雖然在架構上可能略有缺憾，但是可以不用在各個gate存FECGroup的資訊，個人認為還是划算的。

**5.Fraig**

由於時間因素，Fraig的部份個人並沒有完整的寫出來。不過簡單的case還是正確的。在我的演算法裡面，我先執行DFS中各個Cirgate的var初始化，並將他們連起來。然後再對各個FECGroup做C n取2的SAT的證明，如果可以證明出來result是false的話，那就將他們merge。只是這樣子做的話就有可能會產生執行時間過久的問題。而且在sim09.aag裡面就有了許多問題，所以可能會要有額外的算法比較好。

1. 修課心得及期末體悟

在此次程式作業之中，我覺得我學到了很多程式的技巧，同時對程式的理解也有了更深的境界。舉例來說，我第一次知道資料結構是可以繼承的。除此之外，我也發現在vector中erase會是O(n)，而且使用後會造成vector的end()位置改變，因此若只用

for (auto it = test\_vec.begin(); it != test\_vec.end(); ++it)

的話，就有可能會有不可預期的bug。在當初學C++的時候，甚至連都iterator不知道。

除此之外，我發現很多時候雖然自己對如何實做有所想法，但是問題卻時常出在對資料結構、pointer、或是class的操作不熟悉上面(inheritance,polymorphism等等)，因此花了很多時間在查資料，以及了解操作上面。我想這大概是我跟其他人最主要的差別之一吧，花費過多時間在debug上面。

回想起剛修資結的時候已經學了一年的C++，但是每次寫作業的時候我還是時常要查很多資料。不過我現在也理解每次的崩潰背後都是成長。我想修資結的這個決定，我是不會後悔的。