資料結構與程式設計 Final

學號：R05943092
姓名：曾育為
電話：0961157937
Email：ywtseng@eda.ee.ntu.edu.tw

一、前言

在作業六中，用 Map 去存取每個 GATE，但因搜尋一個 Gate 效能考量 O(log n) 和 O(1)的差距，已將Map 改成 Vector 去存取每個 Gate。

二、資料結構

主要二大 class 為 CirMgr 和 CirGate

1. CirMgr：為操控整個電路的介面

Data Member	資料型態	功能
_max_var	unsigned	用來存取 PI+AIG 個數
_pi_num	unsigned	用來存取 PI 個數
_po_num	unsigned	用來存取 PO 個數
_aig_num	unsigned	用來存取 AIG 個數
_latch	unsigned	用來存取 Latch 個數
_gate	vector<CirGate*>	用來存取每個 CirGate，依照 ID 大小順序
_dfsList	vector<CirGate*>	用來存取進行 DFS 後的每個 CirGate 順序
_*temp*_net	vector<unsigned>	用來暫存每個 AIG 的 Input ID，讀取電路讀完所有 AIG 後，進行連線
_feclist	Feclist	為 Feclist class 的 instance
*_simLog	ofstream	用來作為 simulation 後，要輸出指標
Constructor	CirMgr()	

2. CirGate：代表每一個 Gate

Data Member	資料型態	功能
_line_no	unsigned	用來存取在讀檔時為第幾行
_id	unsigned	用來存取此 CirGate 的 ID
_name	string	用來存取此 CirGate 的型態為 PI、PO、AIG、UNDEF、CONST
_symbol_name	string	用來存取此 CirGate 的名字(PI、PO 才有)
_trace	bool	用來存取 DFS 整個電路，是否有探訪過，true 為有，false 為無
_value	unsigned	用來存取 simulation 時，每個 CirGate 的值
_write_value	bool	在 cirwrite gate[ID]時用來分辨是否為此 gate 的 fanin 和 fanout
_fecgroup	vector<CirGate*>	為一個 Vector 用來存放每個 CirGate
_fec	bool	用來判斷_fec 輸出是否需加上!
_fanin	vector<Net>	用來存取二個 fanin 的 Net [詳見第三點]
_fanout	vector<Net>	用來存取數個 fanout 的 Net [詳見第三點]
Constructor	CirGate(unsigned d,unsigned n):_id(d),_line_no(n) 讀入_id 和_line_no 的值	

備註：Class PI、PO、AIG、UNDEF 用來繼承 CirGate

- 檢討：雖然用了繼承方法，但因為時間因素，沒有去細分哪些 function 屬於哪些 class(PI PO)單獨使用，而是將所有 gate 的 function 都寫在 CirGate 內，會多佔用記憶體空間，影響效能。

3. Net： 代表每條連接 Gate 和 Gate 之間的線

Data Member	資料型態	功能
_id	unsigned	存取 fanin 的 ID
_not	bool	存取 fanin 是否有 not 閘，true 為有，false 為無
Constructor	Net(unsigned i,bool n):_id(i),_not(n) 讀入_id 和_not 的值	

4. Feclist：操控 fecgroup 的介面

Data Member	資料型態	功能
_Fecgroup	vector< vector<CirGate*> >	一個 vector 存取每個 vector<CirGate*>
Constructor	Feclist()	

以下二個 class 為 HashKey 和 Key，是為了當作 HashMap 的 key 值而建立

5. Key：在做 Cirstrash 時，用來判斷的每個 Gate fanin 是否重複的 key 值

Data Member	資料型態	功能
_p1	size_t	存取此 CirGate 的 fanin[0]值，以 size_t 表示之
_p2	size_t	存取此 CirGate 的 fanin[1]值，以 size_t 表示之
Constructor	Key(CirGate *g)　讀入一個 CirGate*，將 fanin[0]和 fanin[1]值讀入，如有 not 閘則取補數	

6. HashKey：在做 Cirsimulation 時，用來判斷的每個 Gate simulation 值是否重複的 key 值

Data Member	資料型態	功能
_value	unsigned	用來存取此 CirGate 在 Simulation 所灌入的值
Constructor	HashKey(CirGate* g)　讀入一個 CirGate*	

三、演算法

1. CIRSWeep： 移除沒有到達 PO 的 Gate

Function	void CirMgr::sweep() bool CirMgr::sweep_delete_gate(unsigned id)
Algorithm	1. 在 DFS 時，判斷是否走過 Gate，以_trace 布林值存取 2. 將無走過的 Gate 刪除，並重新連接 fanin 和 fanout

2. CIROPTimize： 將特殊的四個 case，將某些 Gate 以其他 Gate 代替並移除

Function	void CirMgr::optimize()
Algorithm	1. 以 DFS 順序，先判斷 Gate 是否為 AIG，再判斷每個 Gate 的 fanin，決定是否為特殊的四個 case 2. 將四種 case 個別處理 (a).Fanin 有一個 const 1 -> 以 const1 外的另一個取代 (b).Fanin 有一個 const 0 -> 以 const0 取代 (c).相同的 Fanin(有偶數個 not 閘) -> 以 fanin 與 not 閘取代 (d).相同的 Fanin(有奇數個 not 閘) ->以 const 0 取代 3. 將 Gate 移除，並重新整理 fanin 和 fanout 4. 如果有進行 merge，重新進行 DFS

3. CIRSTRash： 判斷二個 fanin，結合結構性相同的 Gate

Function	void CirMgr::strash()
Remark	用 hashMap 去儲存 key 值
Algorithm	1. 以 DFS 順序，先判斷 Gate 是否為 AIG 2. 將 Gate 的 fanin 的 ID 值相加，有 not 閘則取補數，作為 key 值 3. 將 key 值與 CirGate*作為 HashNode 存入_buckets 中，每次丟入前會先判斷是否有相同 key 存在於_buckets 中，如果有，則此 Gate 和_buckets 中的 Gate 進行 merge 並重新整理 fanin 和 fanout ，如果沒有，則持續存入_buckets 中 4. 如果有進行 merge，將重新進行 DFS

4. CIRSimulate

<table>
<tr><td>Function</td><td>void CirMgr::randomSim()
void CirMgr::fileSim(ifstream& patternFile)
void CirMgr::simulation(vector<unsigned>& _pattern)
void CirMgr::simLog(vector<string>& _temp_pattern)</td></tr>
<tr><td colspan="2">Algorithm</td></tr>
<tr><td>randomSim()</td><td>fileSim(ifstream& patternFile)</td></tr>
<tr><td>依照 PI 的數量決定模擬的次數
(模擬次數＝6*(PI 數量^1/2)＋10)
取亂數的方式取得 pattern 測資</td><td>將 pattern 檔讀入，每 32 單位為一組測資 pattern</td></tr>
<tr><td colspan="2">先將 DFS 所經過有的 gate 及 const gate 放入初始化的 fecgroup</td></tr>
<tr><td colspan="2">將 pattern 測資灌入 PI Gate 當中，並進行 Simulation，每個 Gate 將得到一個 simulation 的值</td></tr>
<tr><td colspan="2">將每個 Gate 的 simulation 值取 Hashkey，放入 hashmap 中，方便我們直接找到 fec pair，如果相等或是取補數相等，則放入同一個 group，反之，則另外新創一個 group。經過反覆的模擬，將原先的 fecgroup，分類成更小的 fecgroup。</td></tr>
<tr><td></td><td>如果_simLog 為 True，將 Simulation 後的 PO 的值寫入檔案中，反之則不用</td></tr>
</table>

四、效能與討論

	My Code	Ref Code
CIRSWeep [sim06.aag 為例]	Time: 0.04 seconds Memory: 1.824MBytes	Time: 0.02 seconds Memory: 1.004MBytes
	Time: 我覺得二倍的差距在於建立 DFS 的順序差異，因為將 fanin 和 fanout 重新連結皆是 constant time，影響不大，而我 DFS 是用 Stack 資料結構，以遞迴去執行，我覺得這部分是落後 Ref code 的主因	
CIROPTimize [sim09.aag 為例]	Time: 0.02 seconds Memory: 2.129 MBytes	Time: 0.01 seconds Memory: 0.7031 MBytes
	Time:我覺得此部分二倍差距問題依舊在於建立 DFS 的順序，不論是判斷 fanin 還是重新整理 fanin 和 fanout 皆是在 constant time 可以解決。 Memory:我覺得在於 CirGate 我沒有進行有效的分類，耗費太多記憶體在 data member 上，我幾乎為了一個功能，就會去加 data member 去執行，如果能善用繼承，有效分類，將大幅改善 memory	
CIRSTRash [sim09.aag 為例]	Time: 0.02 seconds Memory: 2.273MBytes	Time: 0.01 seconds Memory: 0.8281MBytes
	Time： 除了 DFS 外，另一個影響時間因素在於 HashTable 的差異，但我覺得 HashTable 的方式跟 Ref code 差異不大，還是 DFS 影響二倍的差距 Memory:我覺得問題還是在於 CirGate 的 data member 太多導致，HashTable 影響不大	
CIRSimulate [sim10.aag 為例]	Time: 0.04 seconds Memory: 2.715MBytes	Time: 0.01 seconds Memory: 0.418MBytes
	Time: 為了確保 fecgroup 的正確性，我進行較多次的 simulation，也造成時間上的差距 Memory：將近六倍的差距，我覺得我要回去探討取 reference 的使用量是不是太少，導致消耗過多的記憶體	

五、總結

整體而言，對於前三個功能 CIRSWeep、CIROPTimize、CIRSTRash，經過反覆的 Debug，應該正確度算比較高，而第四個功能 Simulation，有幾個 case 有小不足的地方，但由於時間的因素，也只能完成到這裡，很謝謝老師和助教，讓我的程式能力大幅的進步，這門課確實改變我這一生的程式能力。