資料結構與程式設計期末報告

DsNP Final Project Report

系級：台大電機工程學系二年級

學號：b06901017

姓名：鐘民憲

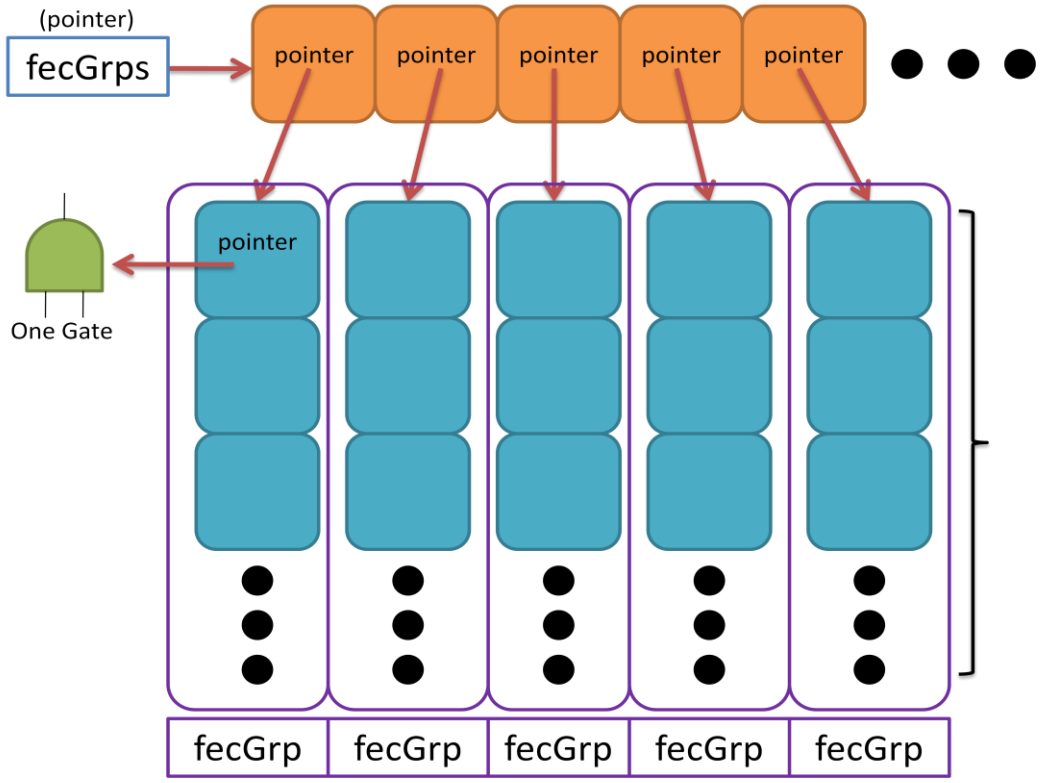
一、Data Structure

1.CirMgr：

主要用來操控整個 circuit 的 object，其 function 包含 read .aag file and translate into circuit，print summary, netlist, primary inputs(PI), primary outputs(PO), floating gates，以及 wrtie the circuit into a new .aag file。

內部有一個大小為 5 的 int array，儲存.aag file 第一行的五個數字(m,i,l,o,a)，此外有三個 vector，第一個 vector(IdList)是依照.aag file 的順序儲存 PI, PO, AIG, CONST 0, UNDEF gates 的 vaariable ID，第二個 vector(GateList)是儲存指到每一個 gate 的 pointer，而且每一個 gate 的 ID 都正好對應到 vector 的索引值，因此能夠快速地查找 gate，有以上兩個 vector 就能很快速的找到或印出任何一個 gate，第三個 vector 是 depth first search list，照順序對每一個 PO 做 DFS 並將結果紀錄於此 vector 中，print netlist 便是透過這個 dfslist 來實現，還有第二節的 Algorithm 包括 sweep, optimize, strash, simulation，都是根據 dfslist 去實作，而各個演算法的內容下一節會再詳細提及。

最後有兩個 pointer，第一個 pointer 是當 Command:simulation 後面有指定 output log 時用來做檔案輸出的，第二個 pointer(稱作 fecGrps)則是指向一個 vector，而這個 vector 裡面儲存的又是指向 GateList 的指標，也就是二維的 vector(見下圖)，而這個 GateList 裡面儲存的就是 simulatoin 之後得到的 functionally equivalent circuit(FEC) groups，每一組 FEC group 都會存在同一個 GateList 中，不同組的就存在不同 GateList 內，而全部都用 pointer 來儲存的好處就是能避免 object 的 copy，也能減少 memory usage。

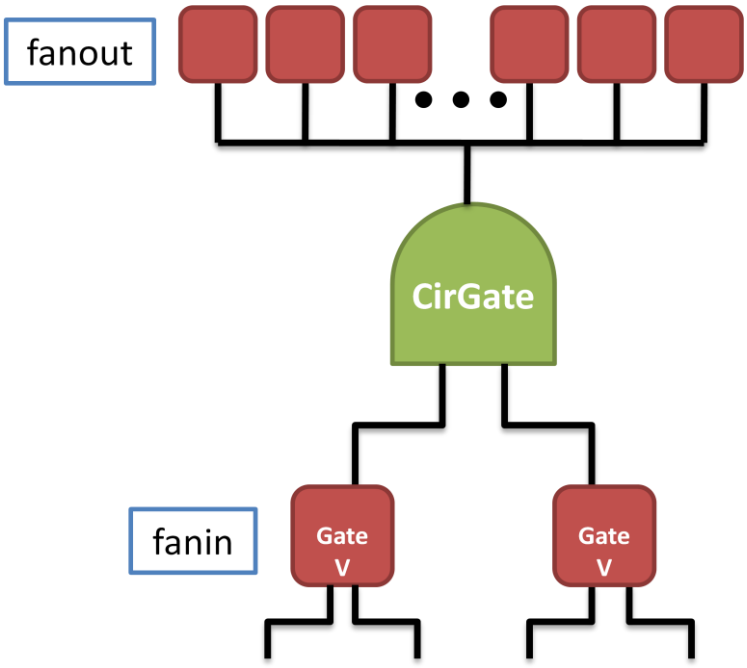


2.CirGate：

為一個 Abstract object，CirPiGate, CirPoGate, CirAigGate, CirConstGate, CirUndefGate 均是繼承此 object，想當然爾，每一個 CirGate 就代表一個電路中的 gate。其內部儲存了 Variable ID、定義在.aag file 的第幾行、做 DFS 用的 mark、comment 中的名字、輸入端(fanin)(vector)、輸出端(fanout)(vector)，以及 simulation 的 signal。

它的 function 除了基本的 data access 之外，還可以增加或刪除 fanin 跟 fanout，印出這個 gate 的 information(basic information, FEC, signal)，以這個 gate 為基準向 fanin 或 fanout 印出幾層 gate，執行 simulation，檢查這個 gate 是否有 floating fanin 及 unused fanout。

比較特別的是，fanin 跟 fanout vector 裡儲存的 pointer 並不是指向另一個 CirGate，而是指向 GateV，GateV 其實就是把指向 CirGate 的 pointer 進行加工，由於一個 pointer 的大小等於 size_t(在 64 位元電腦為 8 bit)，所以每一個 pointer 的 LSB(least significant bit)都會是 0，藉由這個 LSB 我們便可以儲存 invert 的資訊，如此一來便可實現 NOT gate 而不需要再另外定義一個 object 了。



二、Algorithm

1.Sweep：

簡單來說就是將不在 DFS list 內的 AIG 跟 UNDEF gate 都刪掉，在 DFS list 內的 gate 就是可以由 PO 沿著 fanin 連接到的 gate，換言之，只有在 DFS list 內的 gate 才會對整個電路的 output 有實質性的影響，其他的 gate 老實說都是多餘的，因此刪掉對整個電路也不會有任何影響。

我實作的方法就是開一個 removelist 先 copy gatelist，然後再 traverse 過一遍 dfslist，如果 gate 在 dfslist 裡就把 removelist 裡的值設為 0，最後 traverse 過

removelist，遇到不是 0 的就代表這個 gate 要被刪掉。而我刪掉 gate 的方法，就是從它的 fanin gate 的 fanout list 裡面 erase 指向這個 gate 的 pointer，然後再從它的 fanout gate 的 fanin list 裡面將指向這個 gate 的 pointer 設為 0，之所以不 erase 的原因是為了之後接 fanin 時會優先接在 0 這個接腳，這樣便能保持 gate 之間的相對關係。

在實作 Sweep 的時候我才發現 UNDEF gate 不一定會被刪掉，由於我原本的 DFS list 裡面都不包含 UNDEF gate(print netlist 的時候沒有)，所以在 Sweep 之後 UNDEF gate 一定會消失。因此我讓 DFS list 內也會存 UNDEF gate 但是在 print netlist 的時候就跳過，讓它不會出現，這樣子這些在 DFS list 內的 UNDEF gate 就不會被 Sweep 掉。

2.Optimize：

可以 optimize 的 case 有四種：

(1)其中一隻腳接 1 的話 output 等於另外一個 input

(2)其中一隻腳接 0 的話 output 等於 0

(3)兩隻腳接同一個 gate 且 phase 相同的話 output 等於一個 gate+phase

(4)兩隻腳接同一個 gate 且 phase 相反的話 output 等於 0。

我實作的方法就是 traverse DFS list 一遍，如果是 AIG gate 的話就去判斷是否為上述四種情形之一，如果是的話就刪掉這個 gate(刪除方法在 Sweep 有提及)，然後根據不同情形做相應的接線，以下敘述：

(1)把非 1 的 fanin 接到這個 gate 的所有 fanout

(2)把 0 接到這個 gate 的所有 fanout

(3)把任一 fanin(因為 fanin 相同)接到這個 gate 的所有 fanout

(4)把 0 接到這個 gate 的所有 fanout

當然，接到 fanout 後還要考量 fanout 的 phase，若 fanin 跟 fanout 同時 invert 的話，那就可以抵銷變為沒有 invert，若其中一個是 invert 的，那就不能抵銷。最後由於 DFS list 會改變，所以再做一次 DFS，重新建立新的 DFS list。

3.Strash：

概念上是，當兩個 gate 它們的 fanin(gate+phase)完全相同的話，便可以直接 merge 在一起，用一個 gate 代替兩個 gate，而要如何快速的確認兩個 gate 的 fanin 是否完全相同，便要使用到 HashMap。

本 project 使用的 HashMap 為自己從 hw7 的 HashSet 修改而來，使用的 key 是一個自訂的 object，裡面存兩個 size_t 變數，這兩個 size_t 分別就是兩個 fanin 的 GateV 裡面儲存的 size_t 變數(pointer to CirGate+phase)，Key 裡面 overload () 跟==兩個 operator 以使用 HashMap，()回傳兩個 size_t 變數相加，==會比較兩個 object 的 size_t 變數是否相等(次序沒差)；而 HashMap data 便是 CirGate*。

在實作上是先建立一個空的 HashMap，然後 traverse DFS list，如果是 AIG gate

的話，就將它 insert 到 HashMap 裡，如果 insert 不進去就代表 HashMap 裡面已經有 gate 跟它有完全相同的 fanin，此時便可將兩個 gate merge 起來。

Merge 的方式為將某一個 gate 刪掉，然後將它所有 fanout 與另外一個 gate 連接，如此一來便大功告成。

4.Simulation：

這有點像是畫真值表的感覺，若現在有 n 個 PI，理論上如果能夠輸入2^n個不同的 0,1pattern 去模擬，便可以找出所有 signal 永遠相同的 AIG gate 然後把它們通通 merge 在一起，但是在 n 很大的情況下根本不可能模擬得完所有可能的 pattern，因此我們只能夠先模擬一些 pattern，找出 signal 相同的 AIG gate，並懷疑它們是 FEC，接下來再用 SAT 去證明它們是否真的是 FEC，證明為真的話再把它們 merge 在一起。

Simulation 可以是隨機的，也可以讀 pattern file，然而在電路中模擬時 pattern 的大小永遠都是 64 bit(未達 64bit 便補 0)，也就是 size_t 的大小，如此一來便不需要一個一個 bit 的去模擬，AIG gate 在操作時只需要直接使用&(Bitwise AND)跟~(Bitwise NOT)就能夠對兩個 fanin gate 裡面的 signal 做 AND 跟 NOT，操作上快速且方便許多。

至於如何找 FEC groups 同樣要使用到 HashMap，不過在此我使用的是 Standard library 裡面的 map，key 是 size_t 大小的 signal，data 則是一個 GateList。在真正 simulation 之前，先檢查 fecGrps 是否為空，為空的話代表尚未 simulation 過，便需要先將在 DFS list 裡面的 AIG gate 全部存到一個 GateList 內，再將這個 GateList 存到 fecGrps 內，所以在 simulation 之前，所有 gate 都屬於同一個 FEC group。每一次 simulation 過後，創建一個 newfecGrps，此時每一個在 DFS list 裡面的 AIG gate 都會有 signal，接下來 traverse DFS list，以每一個 AIG gate 的 signal 為 key 去檢查這個 signal 是否已經存在在 map 裡，如果存在，把這個 AIG gate 加入到對應這個 key(signal)的 data(GateList)內，如果不存在，建立一個新的 GateList 裡面包含這個 gate，將 pair<signal,Gatelist>插入到 map 裡面，traverse 完 DFS list 之後，traverse map 看是否有 data(GateList)的大小超過 1，超過 1 的 GateList 便是一個 FEC group，將這個 GateList 插入 newfecGrps，最後以 newfecGrps 取代原本的 fecGrps，如此以來便成功收集到所有 FEC group 了。

在 simulation 之後可以選擇是否要 output 成 log file，裡面會記錄每一筆 input pattern 以及每一個 input pattern 對應到的 output pattern。

至於 random simulation 何時應該停止，我決定藉由 experiment 去測試 simulation 多少 patterns 需要多少時間來決定。若 simulation 太多 patterns，不但運行時間會相當久，且能夠判別出的 FEC groups 也不一定會有多少改變，然而 simulation 太少 patterns，有可能會得到過多的 FEC groups 需要 SAT proof，這又會花更多的時間，所以我覺得應該定個限制說 simulation 多少 patterns 之後若 FEC groups 的數量持續不變的話就要停止 simulation。

5.Fraig：

由於我沒有時間完成 fraig，所以在此只描述一下 fraig 的概念以及我的構想。fraig 要做的事情應該是要對在同一個 FEC group 裡面的 gate 兩兩去做 SAT proof，如果證明兩個 gate 確實是 FEC 的話就執行 merge，但是若兩個 gate 不是 FEC 的話 SAT 會給出一組反例 pattern。

由於 FEC group 裡面的 gate 數量可能很多，如果真的要全部證明完畢的話至少要證明C_2^n次，時間複雜度等於n^2，這樣實在太過耗時，再加上由於兩個 gate 可能處在電路中的任何位置，在 merge 上面會有難度，因此，我認為應該要先證明比較接近 PO 的 gate，因為如果證明出來結果一樣的話，電路便可以大幅簡化，剩下要證明的 gate 數量便會大大減少，省下不少時間，再者，若證明結果不同，得到的反例 pattern 可以再丟進電路 simulate 一次，有機會可以將現存的 FEC group 拆成更小的 FEC group 或甚至減少 FEC groups 的數量。

三、Experiment

1.Memory usage and time usage of reading a circuit：

cirr …

Test file	My program	Reference program
Sim13.aag	0.54s	0.08s
	30.05M	13.92M
C7552.aag	0.03s	0s
	1.93M	0.9414M

2.Sweep+optimize test：

cirr…

cirsw

ciropt

Test file	My program	Reference program
Sim13.aag	0.76s	0.09s
	29.94M	14.93M
C7552.aag	0.12s	0.01s
	1.863M	1.242M

3.strash test：

cirr …

cirstr

Test file	My program	Reference program
Sim13.aag	0.75s	0.09s

	32.71M	17.78M
C7552.aag	0.08s	0s
	2.359M	1.383M

無論是在 read、sweep、optimize 還是 strash，我的速度都比老師的還慢，且記憶體用量還比較大，之所以會用掉這麼多記憶體應該是因為我在 CirMgr 裡面開了一個 GateList 讓索引值對應到 gate 的 ID，因此這個 GateList 的大小就會是.aag file 第一行的 M+o+1。

4.simulation test(read pattern file)：

cirr simxx.aag

cirsim –f pattern.xx

Test file	My program	Reference program
Sim12.aag	1.27s	0.25s
	4.363M	2.547M
Sim13.aag	10.71s	1.91s
	34.7M	18.79M
Sim14.aag	0.02s	0s
	0.6133M	0.3281M

不意外的，我的速度慢，記憶體用量大，但 simulation 耗時其實大致可以拆成兩個部分，第一個部分是 simulation 的過程，第二個部分是 collect FEC groups 的時間，我的程式在尚未 collect FEC groups，單純 simulation 的時候跑出來的 runtime 大概就跟上表差不多，而上表是兩個部分都有的，換言之，我大部分比老師多花的時間是在 simulation 上而非 collect FEC groups 上。我覺得我 simulation 會比老師慢的原因可能是因為我每一次都會讓每一個 gate 都 simulate 一次，重新計算 signal，然而更好的做法應該是當 fanin 的 signal 有變化時再做 simulate 就好，這樣能省下一些時間，雖然我曾經試著這樣做，但是最後 simulate 出來的結果都會有錯誤，所以最後我只好改回全部都 simulate，以確保正確性。

5.simulation test(random pattern)：

cirr sim13.txt

cirsim –r

以 sim13.aag 為基準，以下為我控制模擬 pattern 的數量所得到的表格：

Pattern amount	Take times
1024	0.74s
2048	1.29s
4096	2.16s

8192	3.91s
16384	6.96s
32768	13.4s

接下來我限定連續模擬幾次 FEC groups 的數量都沒有變化就停止模擬(一次 64 個 pattern)，這樣會模擬多少 pattern 及所花費的時間：

Limit	Total pattern simulated	Take times	Total FEC groups
2	11136	5.12s	4070
4	24000	10s	3853
8	30912	12.6s	3750
16	65280	25.32s	3482

可以見到當限制變為兩倍，模擬的時間大約就會變為兩倍，而找到的 FEC groups 數量減少速率並不固定，猜測 FEC groups 減少的速度會逐漸趨緩，最好的 simulation 應該是根據 FEC groups 曲線的斜率來決定何時該停止，也就是當斜率低於某個程度時就結束模擬。礙於時間因素沒辦法做到這些，所以在我的 program 中是設定 Limit=5。當然，先做完 sweep, optimize, strash 再來 simulation 效率一定高上許多。