資料結構與程式設計

期末專題報告

FRAIG

姓名：甘德翊

學號：B03901083

E-Mail：b03901083@ntu.edu.tw

聯絡電話：0975719129

一.程式架構：

1. 整體概述：

整個資料結構主要分成兩個部分 CirMgr 及 CirGate，CirGate 主要構能建立每個 gate 的內容及架構，決定每個 gate 包含了哪些功能，而 CirMgr 的主要功能在於管理整個電路，每個電路是由很多的 gate 所組成，而每個 gate 之間關係的建立，以及未來電路的簡化都是由 CirMgr 來處理的，總而言之，CirMgr 就是一個電路的 manager，管控電路的各個功能。

2. CirMgr 架構：

在這部分我主要是直接延續 HW6 的做法，data member 主要有幾個部分：

(1) unsigned int M , unsigned int I , unsigned int L , unsigned int O , unsigned int A 及 unsigned int Real_AIG，分別負責存取電路讀取進來時的 M,I,L,O,A 值，以及實際在電路 deep first search 中的 And-Gate 數量(Real_AIG)。

(2) vector<CirGate*> GATE , vector<CirGate*> PI , vector<CirGate*> PO , vector<CirGate*> DFS，這幾個 vector 分別存取 1. 所有的 gate(GATE) 2. Input gate(PI) 3. Output gate(PO) 4. deep first search 可以經過的 gate(DFS)。

(3) bool issweep , bool isSimulation，分別記憶是否進行過 sweep 及 simulation。

(4) vector<unsigned int> PIsimValue , Cache<simNode,unsigned int>，這兩個 data member 主要是在進行 simulation 時會使用到，再之後會詳盡說明。

(5) list<vector<unsigned int>*> FecGroups，負責存取在 simulation 後形成的 fec group。

3. CirGate 架構：

這部分也是沒有更動，直接延續了 HW6 的做法，並沒有使用繼承，另外新增了一個 class CirGateV，來接電路線，主要的 data member 如下：

(1) unsigned int ID , unsigned int line , GateType type , string IO_Name，這幾個 data member 分別儲存 gate 的 ID , 讀取進來時的所位於的行數 , gate 的類別(AND, PI, PO or UNDEF)以及 gate 的名字(只有 PI gate 及 PO gate 可能有名字)。

(2) vector<CirGateV> Fanin , vector<CirGateV> Fanout，負責儲存 gate 所連接的 input gate 及 output gate。

(3) bool isVisit , bool indfs ，這是在進行 deep first search 時會使用到。

(4) size_t writegate , static size_t globalref，這是在進行 write gate 時會使用到。

(5) unsigned int SimValue，儲存 simulation 後每個 gate 最後的 simulatuon

pattern，若尚未進行 simulation 或是不在 deep first search 中的 gate 或是 UNDEF gate，最將此值設為 0。

(6) vector<unsigned int>* FecGroup，這個 pointer 指向 gate 所在的 fec group。

(7) Var *var* , size_t order，在 fraig 中會使用到。

二.演算法設計：

1. 整體概述：

在這裡我主要只討論 final project 中的五個重要的 command(功能)及其使用到的主要 function。

2. sweep:

在我的資料結構設計中，sweep 的進行非常的簡單及直觀，我在讀取電路時就已經事先存好了 GATE 及 DFS 兩個 CirGate pointer's vector，其中 GATE 的存法是 gate id 即為其在 GATE 中的位置，也就是說當讀取電路時，M+O 值為 n，那 GATE 就會是個長度為 n+1 的 vector，如果得進了一個 variable id 為 a 的 gate，那我就會進行 GATE[a]->new CirGate();去新增一個 gate，而沒有對應到的 id 則是指定 NULL，因此在做 sweep 時，我只要將 GATE 中的 AND gate 及 DFS 中的 AND gate 來做比較，如果 id 為 b 的 gate 沒有存在於 DFS 中，那就進行下面的步驟，delete GATE[b]; GATE[b] = NULL，如此即可將 deep first search 沒辦法經過的 gate 刪除，最後更新 Real_AIG 為電路現有的 AND gate 數。

3. optimization:

首先我先建立了一個 enum 如下，enum fanint_type{F1_CONST0, F1_CONST1, F2_CONST0, F2_CONST1,Equal_NoneInv,Equal_Inv,F1_Inv,F2_Inv,None};這幾個分別代表 1st fanin 為 const 0, 1st fanin 為 const 1, 2nd fanin 為 const 0, 2nd fanin 為 const 1, 1st fanin 等於 2nd fanin 且都沒有 inverse, 1st fanin 等於 2nd fanin 且都是 inverse, 1st fanin 等於 2nd fanin 且 1st fanin 是 inverse, 1st fanin 等於 2nd fanin 且 2nd fanin 是 inverse 以及其他狀況，之後利用 faint_type(CirGate* c)這個 funtion 來判斷 DFS 中的每個 AND gate 分別屬於哪種情況，其中判斷時有可能有同一個 gate 符合一種以上的情況，那此時我將這個 gate 當成在 enum 中值較小的情況，例如當 gate g 同時為 F1_CONST0 及 Equal_NoneInv 時，我會把 gate g 判斷成 F1_CONST0，經過上述判斷後即可知道 merged gate 以及 merging gate，例如當 gate g 為 F1_CONST0 時，merged gate 為 g，merging gate

為 0，將這兩個 gate 丟入 helper function，mergeGate(CirGate* mergedGate , CirGate* mergingGate)即可完成 gate merge，以下詳盡說明 mergeGate 這個 helper function 如何進行 merge。

4. strash:

這個部分我並沒有使用 HashMap，而是直接利用 HW7 寫好的 HashSet，這裡我自訂一個 class faninNode，faninNode 有三個 data member，CirGate* gate , unsigned int fanin1 , unsigned int fanin2，分別儲存 AIG gate 和他的兩個 fanin，另外我在這裡做了 operator overloading 如下，bool operator == (const faninNode& f) const { return (fanin1 == f.fanin1 && fanin2 == f.fanin2); }，也就是只有當兩個 faninNode 的 fanin1 及 fanin2 皆相同時皆，我才會判定這兩個 faninNode 相等，之後我會從 DFS 的 vector 中一個個檢查每個 AIG gate，如果 HashSet 中沒有和此 AIG gate 一樣 fanin 的 gate 那就將這個 gate 形成的 faninNode 插入 HashSet 中，如果 HashSet 中已經存在和此 AIG gate 一樣 fanin 的 gate，那就把這兩個利用上述所說的 mergeGate(CirGate* mergedGate , CirGate* mergingGate)來進行 gate merge，在這裡我會把後來比對的 AIG gate 當成 mergedGate，存在於 HashSet 中的 gate 當成 mergingGate。

5. simulation & fecgroups:

在這裡我使用了 HashMap，並建立一個新的 class 作為 Hashap 的 key，SimValueKey 只有一個 data member 為 SimValue，而在 gate 形成 SimValueKey 時我會先判斷 gate 的 SimValue 和~SimValue(SimValue 的 inverse)誰比較大，儲存較大者，這樣就可以同時判斷出 FEC 和 IFEC，此外我儲存 FecGroups 是利用一個 list，因為 FecGroups 會常常做刪減，而 list 中儲存的是 vector<unsigned int>*，之後我會利用 isSimulation 去斷是不是第一次進行 simulation，如果是第一次那就對所有在 DFS 中的 gate 打包成 SimValueKey 丟入 HashMap<SimValueKey,vector<unsigned int>*>看看有沒有存在同樣的 Simulation 值，如果有就利用 HashMap::query 取得儲存這個 Simulation 值的 FecGroup，並判斷兩者的 SimValue 是相同還是剛好為 inverse，相同則儲存 gate's ID * 2，否則儲存 gate's ID * 2 + 1，而如果不是第一次的 Simulation，那就對於所有的 FecGroup 一個個檢查，重複上一個步驟，將同一 FecGroup 的 gate 丟入 HashMap<SimValueKey,vector<unsigned int>*>驗證，在每次分完 FecGroups 後我還會檢查每個 FecGroup，將 FecGroup 中只有一個 gate 的 FecGroup 刪除。

在 simulation 時有兩種選擇：

(1) randon:

我會利用 rand()產生最大值為 UINT_MAX 的亂數，當成 32-bit pattern 輸入 input，並且設定一個 MAX_FAIL 值，我設定時是利用下式，max = ((Real_AIG + PI.size()) / 32) <= 10 ? ((Real_AIG + PI.size()) / 32) : 10，取 10 為最大值是因為值太大會跑太久，效率不佳。

(2) file:

在這裡我利用 ofstream 和 istringstream 來取得 pattern，並利用下式將 string 轉換成數值，if(string[i] == '1') { PISimulation[i] ^= (1 << bitNum); }，在這裡 bitNum 表示現在已經輸入的 bit 數量，當輸入達到 32bit 時就打包進行 simulation 並將 bitNum 歸零，重複上個步驟直到輸入 file 中全部的 pattern，另外如果最後的 bitNum 數不滿 32 則在最後補零，補滿 32bit 再丟入 simulation。

這裡我的結構較麻煩的地方在於因為 list 和 vector 都沒有依照數字大小排序，因此最後還必須利用 std::sort 去排序，其中在 list 的排序時，因為 vector<unsigned int>沒辦法比大小，要自己建構比較原則，因此我採用 bool comp(const vector<unsigned int>* v1 , const vector<unsigned int>* v2){ return (*v1)[0] < (*v2)[0]; }，作為比較結構傳回 FecGroups.sort(comp);這樣就可以依 vector 的第一個元素由大到小排序。

6. fraig：

對於每一個 FecGroup，將第一個 gate 和其他 gate 分別兩兩丟入 sat 去驗證(EX:若 FecGroup 中有 1,2,3,4 四個 gate，賊驗證(1,2) (1,3) (1,4))，之後如果和第一個 gate 對比結果為 sat 將此 gate 取出 FecGroup，若對比結果為 unsat，則將此 gate 和第一個 gate merge，之後重複多次，達到一定數量後結束，在這裡我並沒有進行 re-simulation。

三.執行結果討論：

1. sweep , optimization , strash：

前三個功能執行速度都很快，因此我只研究有沒有正確，並沒有詳細的觀察執行效率。

2. simulation & fecgroups :

經過測試，我的程式執行效率大約都是 reference program 的 1.5~2 倍，以 sim13.aag 為例，reference program 執行下列指令 cirsim –file pattern.13 –o 13.log 時所花費時間為 33.48 秒，而我的程式執行此指令時，執行時間為 53.54 秒，我

花費了許多時間去研究修改程式，還是無法將低至更小的秒數，這部分可能要在繼續研究是不是有多餘的 variable copy 產生，導致執行時間拉長。

3.fraig:

執行效率很差，似乎建構不佳幾乎達不到消除的效果，因此也沒有多加驗證。

四.總結：

前三個的功能我有用 debug message 去驗證我的結構，在現有的測資中跑出來的結果和 reference program 一樣，而效率也沒有很差，至少在巨大測資時不要和 reference program 有很大的差別，然而最後兩個功能雖然丟測資跑出來的結果都是正確的，但是效率卻會差別很大，導致巨大測資時花費的時間會非常漫長，我研究了很久都還沒有想出是程式的花了太多的時間，因此最後也沒有改善得很好，或許要之後再做進一步的檢查及修改。

五.製作心得：

這次的 final project 對我來說是個前所未有的巨大挑戰，在此之前我從來沒有獨自建構過如此龐大的資料結構，因此在製作的過程中遇到了許多的困難，剛開始時還使採取之前寫程式的方式，走一步算一步的慢慢寫，但因為這樣沒有是先好好思考整個程式的架構，導致常常做好了一個 function，要做另外的 function 時卻因為先前製作 function 時的架構不佳，因此後來的程式會變得十分混亂，最後花了時間全盤思考整個程式的架構，以及所需要的 data function 和 data member，寫起來才變得輕鬆許多，至少程式不會看起來很複雜;最後的兩個功能對我來說真的是非常困難，光是要搞懂要做什麼就花了我許多時間，尤其在做 simulation 時，要如何讓 gate 的兩個 input pattern 可以 and 就花了我許多時間，在這次 final project 前其實我都沒有使用過 bitwise and，因此實在不熟悉這個用法。

雖然這次 project 最後的執行效率並不是很好，所執行巨大的測資時花費的時間非常多，但是能夠自己寫出一個完整的資料結構真的讓人非常興奮，也真的讓人很有成就感！