# DSnp Final Project:Report

B06901040 楊千毅

Mail:b06901040@ntu.edu.tw

目錄：

# 一、電路的結構與建立方法

## 1. 電路的結構

AIG 只有 AND-GATE 及 INVERTER，而每個 Gate 需要記得的資訊有 GateType、fanin、fanout，以及 fanin 的 invert 與否，這些資訊我用以下 member 來存

```
GateType      _type;
GateList      _fanin;
GateList      _fanout;
vector<bool> _inv;
```

其中，為了節省記憶體，可能很多人都會用繼承的方式來存取 GateType，但我因為在 HW6 遇到一點編譯的小麻煩(原因暫且不明)，所以將 GateType 存在 Gate 的 member 當中。

另外，我的 fanin 用的是 vector 來存，而不是分開兩個 fanin0,fanin1。因為使用後者會造成我維護上很大的困難(很容易看錯)，而用 vector 在 access 時的模式是 _fanin[0],_fanin[1]，相對而言在視覺上較舒適。

另外，在這個結構之中，_inv 的編號與_fanin 的編號是相對的。

## 2. 電路的建構方法

我在讀 aag 檔案的時候，是順著 parse 下來的，也就是說，讀一個資訊，處理一個資訊。

最一開始會先建構一個 const0 的 gate。首先 PI 跟 PO 會在 aag 檔案的最前面，所以只要讀進一個 PI 或 PO 就建構一個 PI 或 PO，接著是建構 AIG，也是跟前面一樣的方法。

**瓶頸 1：**眾所周知，一邊 parse 一邊建構電路很大的難題是，建構的 Gate 的 fanin 不一定已經被 define 了。

**方法 1：**還沒被 define 到的 gate，會直接先接一個 UNDEF_GATE 上去，之後如果被 define 了，再接一個 AIG_GATE 上去，然後把本來的 UNDEF_GATE 刪掉。

**瓶頸 2：**把所有 Gate 存起來的方法有很多種，要怎麼樣存比較好呢？

**方法 2：**直接使用 vector 來存取，而在 vector 裡面的位置對應到的就是 Gate 的 ID。這樣的好處是可以有 O(1) 的 find()，缺點是很多 ID 沒有對應到的 Gate，造成記憶體的浪費，也就是記憶體換時間。

# 二、Function 的實作

## 1. Sweep

要 implement Sweep 首先的要件要有 DFSlist。然後把不在 DFSlist 的 Gate 刪除。

**瓶頸 1：**確認每個 Gate 是否在 DFSlist 當中使用的方法最好可以是 O(n)，也就是有 O(1)的 find()。

**方法 1：**我使用跟儲存 GateList 一樣的方法，使用 vector<bool>。首先 parse 一次 DFSlist，把在 DFSlist 裡面的 GateID 對應到的 vector 位置打勾(=true)，然後再搜尋 GateList 裡面的 Gate 對應到的 vector<bool>的位置是否為真，就可以得到 O(n)的 Sweep 了。

## 2. Optimize

DFSlist 中一個一個確認是否符合那四種條件並將符合條件的 Gate 刪除。

**瓶頸 1：**雖然 Optimize 的概念簡單，但實際 implement 時，發現 Gate 的 fanin, fanout 的維護相當複雜，必須同時維護 fanin 的 fanout 以及 fanout 的 fanin。

**方法 1：**將要刪除的 gate 先存起來，然後使用 function void CirGate::DeleteFanout(new fanin of fanouts

of deleted gate,invert);

void CirGate::DeleteFanin(new fanouts of right fanin,new fanouts of left fanin);

把所有 gate 維護完後，才一次刪掉所有 Gate。像這樣將 gate 的維護跟表面的操作分開，可以讓 code 變得簡潔許多。

另外，在 optimization 中，還要注意要將沒有用到的 UNDEF gate 刪除。要特別注意的是，只能刪除本來在 DFSlist 當中的 UNDEF gate。

3. Strash

**關於 Hash：**我使用的 Hash 是 STL 的 unordered_map。因為我不想要維護 bucket。我使用的 Hashkey 是 size_t。

**瓶頸 1：**unordered_map 裡面一個 bucket 只能放一個元素，所以這樣宣告：unordered_map<size_t,IdList>，也就是 bucket 裡面存的其實是一個 vector，就可以儲存複數個 GateId 了。

**關於 Hash function：**好的 Hash fuction 盡量不會造成同一個 Hashkey 有不同的 structure，而我的 Hash function 將 size_t 的前 32 個位元儲存左邊 faninid，

後 32 個位元儲存右邊 faninid(兩者的 LSB 存 inv)。值得注意的是，左邊 faninid 跟右邊 faninid 必須要 sort 過一遍，才不會造成相反的 id 有不同的 Hashkey。這樣一來，只要 Gateid 在 31 位元的大小之內，每組 fanin 得到的 hashkey 是唯一的，也就是只要在同一個 bucket 裡面的 gate，一定是 structural identical。

另外，Strash 的 Gate 維護較 Optimize 簡單，只需要把 fanout 轉移及 fanin 的 fanout 刪除即可，可以使用 Optimization 中使用的兩個 function(將 DeleteFanin 的輸入改成空集合)。同樣，將所有 Gate 維護完之後，才將要刪掉的 Gate 一次刪掉。

## 4. Simulation

**關於 patternfile：**將 patternfile 讀進來以後，我將要拿來模擬的 input 訊號每 64 個訊號一組(size_t)，存成一個 vector，順序如下：

PI0 的 64 個訊號、PI1 的 64 個訊號、PI2 的 64 個訊號、PI0 的再 64 個訊號、PI1 的再 64 個訊號…以此類推。

關於 simulation：我在 CirGate 中存了另外一個 member size_t _simVal;

按 DFSlist 的順序，每個 Gate 取得 fanin 的_simVal 以後，更新自己的_simVal。

關於 FEC：FECgroups 的資料結構是 vector<Idlist*>，這裡面每個元素都是每個 FECgroup 的指標。要這麼做的原因如下。

關於每個 FECgroup 的 Hash：這個 Hash 會因應 Sim 的結果產生非常多的 FECgroup(Hashkey 是 Sim 的結果)，而最後要將這些 group 蒐集到 FECgroups 裡面，這個時候我只要將 FECgroup 的指標複製給 FECgroups，就可以避免 copy IdList，也就是避免 STL container 的複製，這樣做可以大大提升速度。此處的演算法與投影片相同。

關於 RandomSim：隨機產生 PI*10 個 pattern 以後，做跟上面一樣的 simulation。

關於 CIRGate command 的 FECgroup：在做完 Simulation 之後，我將 FEC 的結果一個一個加入 CirGate 的 member

GateList _FEC;

vector<bool> _FECinv;

其中，_FECinv 儲存的是對應編號的 FEC 是否反相。

但儲存這些資訊會造成 CirGate 需要非常多的 memory。

## 5. Fraig

我的演算法如下：

建立 proof model

for_each_fecgroup

prove every pair of gates in fecgroup by handshaking method(also perform merging in this stage)

update dfslist

strash

update fecgroups

其中，電路的 maintain 與前面幾個 function 相同。

**瓶頸 1：**因為我在呼叫 SAT engine 時沒有太多策略，所以如果 FEC 太相像的話，需要太多時間做 SAT proof。這個問題我在這次 final project 中沒辦法解決。

# 三、心得

## 1.Final project 心得

(一) 這次 project，我解決問題的方法通常是定義更多的 function 跟 data member，而不是去構思更好的程式架構。因此我的 memory 用量非常大，code 也又臭又長。當然這是因為要想辦法 meet deadline 的緣故，所以沒時間砍掉重練，但也因為這樣，以後寫 code 的時候，我會更重視在 coding 前的架構構思。

(二) 這次 project 中我學到最多的，是如何從需求中挑選合適的資料結構。首先因為 find()的時間複雜度是我這次作業的主要考量因素，所以最好的資料結構是 hash，或是以位置編號為 hashkey 的 hash(也就是 array)。但相對的，犧牲了記憶體的用量，也就是拿記憶體換時間。

(三) 這次 final release 的時候已經接近期末考了，導致我為了準備期末考，放寒假之前完全沒有進度，因此沒辦法在需要策略的 fraig 上多下工夫，覺得有點可惜。

2. 學期心得

(一) 這學期我收穫頗豐。除了基本的資料結構以外，學到最多的是 coding sense。這是除了大量練習以外沒辦法得到的。這個 sense 小至變數的取名、縮排的方式、pointer 的使用、時間複雜度的分析、debug 的邏輯，大至程式架構的設計、演算法融入介面的方式等等。

總歸而言，修完這門課我有達到當初學期初老師的目標：擁有 handle 1000 行以上 code 的自信。

(二) 我認為本學期 schedule 的安排上有可以改進之處。首先在期中考前的 deadline 只有 HW1 2 3，而大部分的同學更是在暑假就做完 HW1 了，也就是說，大部分的同學在上半學期只寫了比較簡單的 HW2 3。而下半學期寫了較困難的 HW4 5 6 7。我自己在下半學期，幾乎每天都在寫資結的作業。而且為了配合 HW，final project 釋出的時間也接近期末考，大大壓縮可以寫 final 的時間。也就是說，上下半學期的 loading 分配太不均。

(三) 因為 DSnP 很久以來都是電機系熱門的課，也因此有

很多以前學長姊留下的資源或是有問題時可以直接問學長姐，導致作業跟 final 的實質難度越來越低，造成很多時候為了拚那多出來的一點點分數，必須要投入相當不成正比的大量時間，我認為這不是很健康的現象。同時，沒有認識學長姐的同學，他們的作業難度比起認識學長姐的同學難非常多，我認為這稍稍有失公平。