

資料結構 Final Project

B06901049 林泓均

聯絡方式:0905729688

一、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。

```
enum DFS_utilize
{
    PRINT = 0,
    SWEEP = 1,
    OPTIMIZE = 2,
    STRASH = 3,
    SIMULATION = 4,
};
```

二、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：爲了節省空間，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)

```
//utilize for optimize and strash
bool append_fanout(unsigned target_lit, unsigned delete_id);
bool replace_fanin_of_fanout(IdList &fanout, unsigned init_id, unsigned after_lit);
bool erase_fanout_data(vector<unsigned> &vec, unsigned _data);
inline void printOptimizeInformation(unsigned, unsigned);
void mergeAndDeleteGate(unsigned, unsigned);
```

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，我只需要執行如下動作：

```
replace_fanin_of_fanout(fanout, 目前 id, fanin 的 lit);
```

```
append_fanout(fanin 的 lit, 目前 id);
```

即可完成。

3.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。

(2) 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 unordered_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。

```
class Strash_fanin : public pair<unsigned,unsigned>
{
public:
    Strash_fanin(unsigned num1,unsigned num2)
    {
        this->first = num1;
        this->second = num2;
    }
    bool operator==(const Strash_fanin &comp) const
    {
        if (this->first == comp.first&&this->second == comp.second) return true;
        else if ( this->first == comp.second&&this->second == comp.first) return true;
        else return false;
    }
};
class Strash_hash_func
{
public:
    size_t operator()(const Strash_fanin& _fanin_pair)const
    {
        return hash<unsigned>()(_fanin_pair.first)+hash<unsigned>()(_fanin_pair.second);
    }
};
typedef unordered_map<Strash_fanin,unsigned,Strash_hash_func> Hashtable;
```

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

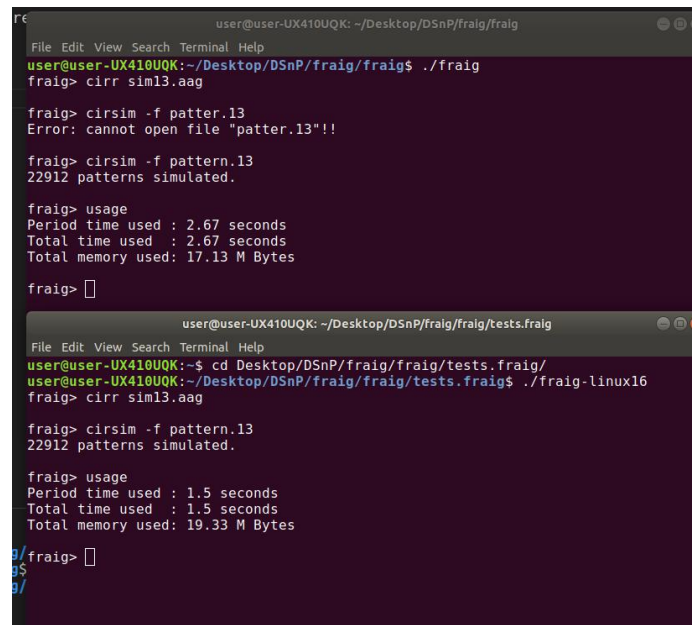
IdList 的部份全部改為

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。



```
user@user-UX410UQK: ~/Desktop/DSnP/fraig/fraig
File Edit View Search Terminal Help
user@user-UX410UQK:~/Desktop/DSnP/fraig/fraig$ ./fraig
fraig> cirr sim13.aag

fraig> cirsim -f patter.13
Error: cannot open file "patter.13"!!

fraig> cirsim -f pattern.13
22912 patterns simulated.

fraig> usage
Period time used : 2.67 seconds
Total time used : 2.67 seconds
Total memory used: 17.13 M Bytes

fraig>

user@user-UX410UQK: ~/Desktop/DSnP/fraig/fraig/tests.fraig
File Edit View Search Terminal Help
user@user-UX410UQK:~$ cd Desktop/DSnP/fraig/fraig/tests.fraig/
user@user-UX410UQK:~/Desktop/DSnP/fraig/fraig/tests.fraig$ ./fraig-linux16
fraig> cirr sim13.aag

fraig> cirsim -f pattern.13
22912 patterns simulated.

fraig> usage
Period time used : 1.5 seconds
Total time used : 1.5 seconds
Total memory used: 19.33 M Bytes

fraig>
```

(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: $0.95M \sim 1.25M$) 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 Cirq

在 Cirq 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 Cirqate 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。

三、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。