# Final Project Report — FRAIG

b04901152 劉芸欣

email : b04901152@ntu.edu.tw

## 一、資料結構實作

這次的Project完成了CIRSWeep、CIROPTimize 、CIRSTRash 、CIRSimulate這幾個功能，CIRFraig 則因為時間關係沒有完成。在過程中使用了幾個比較大的資料結構，用CirGate來實現電路中的gate，CirMgr用來操作整個電路，HashMap則是用來分類在分FEC Group和Strash找相同fanin的gate上。

- CirGate

```
Class CirGate {
        protected:
        vector<GateV>       _faninList;       // 記錄fanin的GateV
        vector<GateV>       _fanoutList;      // 記錄fanout的GateV
        unsigned            _id;              // 記錄Gate的id
        unsigned            _lineNo;          // 記錄一開始aag檔gate被建立的行數
        bool                _indfs;           // 記錄是否在dfs List裡面
        static unsigned     _globalRef;       // 用在DFS trace上面
        mutable unsigned    _ref;
        size_t              _value;           // 用一個size_t記simulate 64個pattern的值
}
```

每個gate用兩個GateV來回互相連接，GateV中記錄了gate和是否inverse兩個參數。各個gate種類用繼承的方式，而不同的gate裡面也會有各自的data member，像是只有PoGate PiGate有string _name來記aag中定義的別名，而AigGate中有個size_t _grp來記是屬於第幾個FEC group。

- CirMgr

```
Class CirMgr {
        private:
         int                  _Max;          // aag檔第一行定義
         int                  _Pi;
         int                  _Latch;
         int                  _Po;
         int                   _Andgate;
        vector<CirGate *> _totalList;      // 所有gate照Id依序放入，如果不存在的gate
                                              則會是_totalList[i] = 0
        vector<CirGate *> _piList          // 所有Pi依序放入
        vector<CirGate *> _flfaninList;    // 所有floating fanin gate
        vector<CirGate *> _notusedList;  // 所有not used gate
        vector<CirGate *> _dfsList;        // 依照dfs將gate依序push_back進來
        ofstream              *_simLog;     // simulation output LogFile使用
        bool                  _simFirst;     // 記錄是否為第一次執行sim
        vector<FECGrp>    _FECGrps;    // 用一個vector記錄所有FEC Groups，每次
                                              sim完就會更新
}
```

CirMgr中可以操作電路和執行各項指令，而關於電路的記錄如上面的data member。首先，記錄了_totalList和_piList，而因為po的id是從_max開始往上數且不會有空號所以不用特別記。接下來，在電路完成讀取之後則需執行CirMge : : findFltInandNotUse()找出Floating 和 Not Used gate，記錄在_flfaninLis和t_notusedList中，同時也須跑CirMge : : dfsTraversal() 記錄在_dfsList中。然而建置好以上List之後，再有可能更動時都需要重新建置List，像是執行CIRSTRash之後需要重新dfsTraversal更新List，還有CIRSWeep需要更新 findFltInandNotUse()，而CIROPTimize則是floating和dfs都需要重跑。

- HashMap

HashMap是由一個HashKey對應到一個HashData，而將<HashKey, HashData>包裝成一個HashNode作為儲存單元。

而在此次作業中，用到兩次HashMap，一次是strash時比較fanin時使用，另一次是simulation完分FEC Group時，用HashMap作為分組方式，因此我建了兩個作為HashKey的Class，分別使用在不同地方。

```
Class faninKey {
        size_t   operator()  () const {  return(_fanin1+_fanin2);  }
        private:
        size_t      _fanin1;                // 第一個fanin gateV所存的值
        size_t      _fanin2;                // 第一個fanin gateV所存的值
}
```

第一次是在strash時使用，用兩個fanin值做為HashKey，而為了要讓兩個fanin交換時還是同像的值，所以Hash function使用兩fanin相加。

```
Class simValueKey {
        size_t operator() () const {
                if(_simvalue > (SIZE_MAX/2)) return ~_simvalue;
                else  return  _simvalue;
        }
private:
        size_t      _simvalue;              // simnulate出來的值
}
```

第二次是simulation完分FEC Group時使用，我使用simulation的值做為HashKey，Hash function轉換完應該要使_simvalue和～_simvalue對應到同一值，所以用上面的方法作為轉換方式。

## 二、演算法

- Simulate

整個電路simulate的方式是靠recursive呼叫的方式，每個gate呼叫自己fanin的simulate（CirGate : : simulate( ) ），一直到trace到Pi或Const的地方，用bitwise operator的方式算出每個gate出來的值，記在CirGate : : _value中。而如果sim第一次之後，兩個fanin的值跟前次一樣沒有改變，simulate()會回傳false則不會再計算一次。

- Simulation -Flie
  1. 先在CirMgr裡建一個bool _simFirst，紀錄是不是第一次執行Simulation，如果是第一次就會要先將所有DFS的gate都放進_FECGrps裡面（CirMgr : : initFECGrps( ) ）。
  2. 讀出一組一組pattern，將64組放在一起組成Parallel Pattern，放入各個Pi，然後接著從各個Po往回Simulation（CirMgr : : simulate ）。
  3. 每組Parallel Pattern simulate完後，開始將各個gate用sim結果用HashMap的方式分成好幾個Group，再把Group放入_FECGrps之中（CirMgr : : splitFECGrps( ) ）。

- Simulation -Random
  1. Simulation -Random跟Simulation -Flie很像，產生出隨機的64組pattern組成一組Parallel Pattern，放入Pi，從Po往回Simulation。
  2. 重複產生Parallel Pattern，一直到Group的數量0，或著連續20次分組前後的Group數量差在(-Group數/1000 , Group數/1000)的範圍之間，則停止。

## 三、實驗設計

1.Sweep - Optimize - Strash

重複下面指令跑完opt01.aag到opt07.aag和sim01.aag到sim15.aag的電路，來測試這三個function的速度及佔的記憶體，和reference code比較

```
cirr test/opt02.aag -r
cirsw
ciropt
cirstrash
usage
```

結果：

mine

```
fraig> usage
Period time used : 0 seconds
Total time used  : 0.65 seconds
Total memory used: 20.71 M Bytes
```

reference:

```
fraig> usage
Period time used : 0 seconds
Total time used  : 0.12 seconds
Total memory used: 16.84 M Bytes
```

由結果可以看出自己記憶體用量還算可以接受，但速度比老師慢了很多，雖然只是跑了三個fuction就可以明顯看出速度差異，可能在很多地方找東西都用linear的方式，所以花了比較多時間。

2. Sweep - Optimize - Strash - Simulation(File)

重複下面指令跑完sim01.aag到sim15.aag的電路，來測試這加上Simulation的速度及佔的記憶體，和reference code比較

```
cirr test/sim01.aag -r
cirsweep
ciropt
cirstrash
cirsim -file test/pattern.01
usage
```

結果：

mine

```
fraig> usage
Period time used : 0.04 seconds
Total time used  : 22.39 seconds
Total memory used: 23.46 M Bytes
```

reference:

```
fraig> usage
Period time used : 0 seconds
Total time used  : 2.89 seconds
Total memory used: 20.07 M Bytes
```

這是一開始的實驗結果，一樣記憶體比老師的多一點，但時間將近多了十倍，由於真的太慢了，所以我回去看自己的code，試著找出simulation可以加快的地方。我發現在分FEC Group的時候從Hash中要用Key搜尋（query），然後找到後還要再把它Insert回去，跑了有點久，所以我又在做了一個HashMap 的function 叫做queryAndInsert()，可以同時query和Insert，減少了時間，下面就是新的實驗結果。

結果：

mine 2

```
fraig> usage
Period time used : 0.04 seconds
Total time used  : 18.93 seconds
Total memory used: 22.21 M Bytes
```

看得出有進步了一些，時間從22秒變為18秒

3. Sweep - Optimize - Strash - Simulation(Random)

跑下面指令在sim15.aag的電路上，觀察速度及佔的記憶體和reference code比較

```
cirr test/sim13.aag -r
cirsweep
ciropt
cirstrash
cirsim -r
usage
q -f
```

結果：

mine

```
Total #FEC Group = 3481

102528 patterns simulated.

fraig> usage
Period time used : 58.94 seconds
Total time used  : 58.94 seconds
Total memory used: 21.23 M Bytes
```

reference:

```
Total #FEC Group = 3526

94336 patterns simulated.

fraig> usage
Period time used : 6.82 seconds
Total time used  : 6.82 seconds
Total memory used: 18.84 M Bytes
```

除了速度之外，事實上使用的記憶體、pattern數、分出來的FEC Group都跟老師的差不多，感覺random設的停止參數還算合理。