台灣大學電機學系複選必修

資料結構與程式設計 Final project

電機二莫絲羽

系級：電機二

姓名：莫絲羽

學號:b04901164

手機：0956332108

Email: evamo0508@yahoo.com.tw

指導老師：黃鐘揚老師

資料結構

本次 final project 有相當多的 class，大致區分如下

- Class CirGate

  Cirgate 紀錄所有 gate 包含 PI/PO/CONST/UNDEF/AIG 的資料

```
    //member
    vector<unsigned> faninList;//lit
    vector<unsigned> fanoutList;//lit
    unsigned SimValue;

private:
  GateType _type;
  string _symbolicName;
  unsigned _lineNo;

protected:
};

#endif // CIR_GATE_H
```

  1.這裡沒有使用繼承來寫這個 class，因為覺得 faninList 和 fanoutList 都用 vector 存就已經可以達到不同 gate 有不同數量的 fanin 和 fanout 的效果。

  2.較特別的是上述兩個 vector 存的皆是 unsigned 的 literal，直接把 ID(literal/2)，和 invert(literal%2)記下來。

  3.SimValue 為後面 simulate 時紀錄值的變數，在此採用 unsigned，因為一次會 simulate 32 個 patterns。

- Class CirMgr

  1. 為一個管理整個電路操作的 class，因此這個 class 相當複雜，所有的指令:Sweep/Optimize/Strash/Simulation/Fraig 皆與他有關

  2. 除了存放一些和操作電路有關的變數及函數，我在這裡還使用了一些較為特殊的元件：

```
    //self-defined hw6
    void resetFlag(){for(size_t i=0;i<_gateNo;++i) flag[i]=0; }
    CirGate* _gateList;//id
    bool* flag;//for constructing _dfsList



private:
    ofstream           *_simLog;
    int M,I,L,O,A;
    unsigned _gateNo;//total gate no.
    void DFS(int id);
    vector<unsigned> _dfsList;//id,actually includes UNDEF_GATE
    vector<unsigned> _piList;//id
    vector<unsigned> _poList;//id

    //self-defined final
    void eraseFanout(unsigned,unsigned);
    void changeFanin(unsigned,unsigned,unsigned);
    void simulate(unsigned*);
    bool checkSimulate(unsigned);
    HashMap<SimValueKey,FECGroup> fecGrps;
    bool ok2FEC;
};
```

(1) 透過 void DFS(int id)我們可以在不 print 出任何東西的情況下創造出 netlist，如此一來，就不需要另外加一些新的 function 來複雜化電路，而 bool* flag 以及 resetFlag()都是幫助建立 Netlist 的 help member variable and function。

(2) _gateList 為整個程式紀錄所有 gateID 最主軸的 dynamic array，選用此 ADS 是因為 find 是 O(1)，而 index 經過設計使得剛好等於 gateID，使用起來十分方便。

(3) _dfsList、_piList、_poList 分別是紀錄 DFS,PI_GATE,PO_GATE 的 gateID 的 unsigned vector，方便之後的 print function 用。

(4) eraseFanout、changeFanin 是在 sweep、optimize、strash 都有使用到的重新連接電路的模組化 function，讓程式看起來更為簡潔。

(5) simulate 和 checkSimulate 當然就是 simulate 用到的 function，其中後者是 recursive function，能達到從 PI 一路 simulate 到 PO 的效果。

3. 在寫 Cirmgr 時，我很注重 function 化各種功能，不要讓一個 function 有超過一個螢幕大小的機會，這樣不但能使 code 更易讀且更易 maintain 和 debug

- Class myHashmap

1. hash map 的概念與 hash 相當類似，我們定義 HashMap 為 HashMap(HashKey,HashData)，我們使用 HashKey 來置放 HashData 在 HashMap 的位置

2. 為了操作方便，我們定義了多個 function 來讀取 HashMap 的 Data，包括：

   (1) check 用來檢查 HashData 是否存在，若存在，則回傳 HashData 回來。

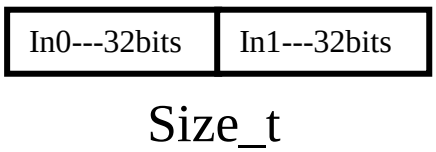
   (2) insert & forceInsert 是用在不同情況下，insert 為如果 data 不存在，則 insert 否則則回傳 false;ForceInsert 要確定 hashkey 是否存在且 hashdata 不能存在，並強制執行 insert。透過各種功能的 function 的操作，可以簡化 code 及增加可閱性

為了操作方便而新增的 class:

- strashKey:為了方便 strash 的操作，我們設定了 strashKey 來做 operator overloading，strashKey 的資料為兩個 fanin 的 literal

- 在這裏我們 overloading()/==，特別設計這個 strashkey 為兩個 fanin 較小者左 shift 32 個 bits 再加上 fanin 較大者，如此便可達成使用一個 size_t 變數就能確保擁有相同的兩個 fanin 的 gate key 都一樣。

- 圖示:



  結構如下所示：

```
class strashKey
{
 public:
   strashKey() {}
   strashKey(unsigned _in0=0,unsigned _in1=0) {in0=_in0;in1=_in1;}

   size_t operator() () const
   {
      if(in0>in1) return ((in1<<32)+in0);
      else return ((in0<<32)+in1);
   }

   bool operator == (const strashKey& k) const
   {
      if(in0>in1) return ((in1<<32)+in0 == k());
      else return ((in0<<32)+in1 == k());
   }

 private:
   size_t in0,in1;
};
```

- CirGateV：CirGateV 也用於 strash，旨在用一個 class 來包住 gateID，方便 hashmap 使用。

- 結構如下:

```
class CirGateV
{
 public:
   CirGateV(){}
   CirGateV(unsigned _id){id=_id;}
   ~CirGateV(){}
   unsigned getID() {return id;}
 private:
   unsigned id;//or cirgate* ??
};
```

- SimValueKey:是為了 simulation 而定義的 hashkey class，在 operator ==的地方，特別將 key 剛好是 bitwise inverse 的情況也納入其中，就能判斷 IFEC pair。

- Flag 的設計是讓在進行 simulation 的時候，iterator 能知道哪些是新分進來的 group，那些是舊的 group，幫助 iterator 的正確運行。

- 結構如下:

```
class SimValueKey
{
 public:
   SimValueKey():flag(0){}
   SimValueKey(unsigned _SimValue):flag(0){SimValue=_SimValue;}
   SimValueKey(unsigned _SimValue,unsigned _flag){SimValue=_SimValue;flag=_flag;}
   ~SimValueKey(){}
   unsigned operator() () const { return SimValue; }
   bool operator == (const SimValueKey& k) const { return (k()==SimValue || k()==(~SimValue))
   unsigned getSimValue() const{return SimValue;}
   void setFlag (unsigned i) {flag=i;}
   unsigned getFlag() const {return flag;}
 private:
   unsigned SimValue;
   unsigned flag;
};
```

- FECGroup:同樣是為了 simulation 而定義的 hashData class。
- 不同於 CirGateV，這個 class 存的是 gateID 的 vector，紀錄在同一個 fecgroup 裡面的 gate 們。
- 結構如下:

```
class FECGroup
{
 public:
   FECGroup(){}
   FECGroup(vector<unsigned>& _id){id=_id;}
   ~FECGroup(){}
   unsigned& operator [] (size_t i) { return id[i]; }
   const unsigned& operator [](size_t i) const { return id[i]; }
   void pushID(unsigned _id) {id.push_back(_id);}
   size_t size() const {return id.size();}
   unsigned getID() {return 0;}
   //bool operator == ( FECGroup& k) const { return k.getID()==id; }
 private:
   vector<unsigned> id;
};
```

設計理念

- Sweep:我們建立一個 dfslist which is from netlist，沒有 reach PO 的代表其不在 netlist 裡，故可以直接刪除之
- Optimize:我們分為四種情況，我們考慮 gate 的兩個 input

  (1)必為零的情況 ：由 const0 取代之

(2)必為一的情況：由非一的 fanin 取代之

(3)inverse 的情況，由 const 0 取代之

(4)相同的情況:由任一 fanin 取代之

另外，我們使用 function 使每一道手續 function 化

--eraseFanout:將有 fanout 是欲刪除 gate 者的 fanout 從 fanoutList erase 掉

--changeFanin:若 A 本來 fanout 到 B，今天要將 A 刪除，使 C 連到 B，則使用此 function。

--resetGate:將要刪除的 gate reset，包括讓 gateType 變回 TOT_GATE，faninList 和 fanoutList 的 clear 等等。

- Strash:使用 HashMap<strashKey,CirGateV>來儲存結構相同的 gate。
- Simulation:使用 HashMap<SimValueKey,FECGroup>來儲存有相同 simvalue 的 FEC。

  首先，我們先將所有 signal 放入一個 initial 的 FecGrp，再把他加進 CirMgr 的 member:fecgrps 中，成為目前唯一的 group。

  使用 unsigned dynamic array 來記錄 pattern file 裡的數字，每個 PI 都有自己的一個 unsigned int 來記錄即將要 simulate 的訊號，在此也以 bitwise operator '^' XOR 來達成此目的。而為了增加速率，我們不會每放進一個 data 就執行一次 sim，而是用 patternNO 來記錄訊號數，當讀進 32 個 patterns 時，unsigned 剛好放滿訊號，我們才會開始跑這部分，超過 32 個 pattern 程式會再重新讀進 patternInput，繼續進行 simulate。

  在 collect valid group 這邊採用的想法是，假設一個舊的 group 被拆成 3 個 group 了，那就把舊的 group 從 fecGrps 裡面 remove，再 insert 新的 3 個 group 進來。

- Fraig:這邊來不及寫了，並沒有完成。

實驗設計與比較

- Optimize:因為 opt 我們會去紀錄曾經去過的 path，故整體比較接近線性操作，速度比較快。

	中電路（sim09.aag）	大電路(sim12.aag)
Gate number	3000	10000
time	0.03s	32.2s
Ref time	1.45s	0.02

我發現 ref program 的速度在小電路比較慢，但在大電路卻超級快，而我則相反，我猜測應該是因為有些做過的部分有重複做所以影響到操作時間

- Simulation:整體上，simulation 速度比較慢，但是都是在能夠接受的範圍

	中電路（sim09.aag）	大電路(sim12.aag)
Gate number	3000	10000
time	32.8s	120.3s
Pattern 產生數	27360	81400
Ref time	0.02s	0.25s
Ref pattern	1888	3520

雖然速度上是比 ref program 慢，但是我猜測部分原因是我產生的 pattern 比較多，或是因為我的 fail max 設的比較大。（且我設定 fail 必須是連續的 fail 才能計算）

- Fraig:因為寫不出來，故放棄實驗比較

Feedback for DSnP

當初想來修資結的原因有兩個:一是覺得程式是電機系的基礎，如果沒有處理大程式的能力，很多事情其實都不能做，走到哪不管是 verilog、matlab，都要寫程式，不走 CS 也逃不太了的感覺，所以想來讓自己變強。二是有點跟風吧，一群同學約著要一起修，自己也覺得沒修過資結就像沒讀過台大電機，就來了。

其實自己寫 code 的能力一直都沒有很好，所以也沒有到很有興趣，可能成就感不大吧，但因為常常告訴自己這是必備能力，所以從大一上到現在，舉凡計程、計概、又

或者資結，都是挑最重的老師修，就是想讓自己程式變強。而經過這一學期的摧殘，我想不變強也很難了，已經習慣處理十幾個檔案的 project，也習慣看千行以上的 code 了。每次到了死線前兩三天，神經都緊繃到不行，尤其這學期又把自己搞得很忙，每次的交作業簡直都是在夾縫中求生存，常常到了死線前不到一小時才終於 de 出 bug，那種驚險刺激的感覺我一定永生難忘。

也許修完資結，我的選擇不是繼續走上 CS 這條路(發現自己好像不太喜歡一直寫 code)，但那個紮實的能力還是 get 啦!相信對我以後軟硬整合的課程有很大的幫助。我也能大膽地說，從今以後不會再害怕看 code 啦!!!

修了這門課其實也才比較了解所謂資料結構到底有多種要，印象最深的是 homework5 的 memory arrangement，當時了解作業要求後是震懾於事物的美麗的那種心情，很讚嘆資料結構的威力。謝謝 Ric，讓我和姿婕約會了一學期，得到許多寶貴的經驗!