DSnP Final Project

Student : 陳孟宏, 電機二, B07901103
Professor : Ric Huang

CIRSWeep :

1. Find Unused Gates :

Because after sweeping, there will be some gates or inputs became unused, so I set a variable called “NotUsed” and store the inputs that do not exist in _DFSList first !

```
void
CirMgr::sweep()
{
  // find unused input
  for (int i = 0 ; i < _pilist.size() ; i++){
    int count = 0;
    for (int j = 0 ; j < _dfslist.size() ; j++){
      if (_pilist.at(i) == _dfslist.at(j)) { count = 1; }
    }
    if (count == 0) { notUsed.push_back(_pilist.at(i)->getVar()); }
  }
  // Remove unused gates
  for (int i = 0 ; i < _totalList.size() ; i++){
    GateType type = _totalList.at(i)->getType();
    string gatetype = "";
    if (type == UNDEF_GATE || type == AIG_GATE){
      for (int j = 0 ; j < _dfslist.size() ; j++){
        if (_dfslist.at(j) == _totalList.at(i)) break;
        if (j == _dfslist.size()-1){
          CirGate* unused = _totalList.at(i);
          unsigned NOT_USE = unused->getVar();
          // delete at notUsed and fltFanins
          vector<unsigned>::iterator n = find(notUsed.begin(), notUsed.end(), NOT_USE);
          vector<unsigned>::iterator f = find(fltFanins.begin(), fltFanins.end(), NOT_USE);
          if (n != notUsed.end()) { notUsed.erase(n); }
          if (f != fltFanins.end()) { fltFanins.erase(f); }
          // delete at aiglist
          vector<CirAigGate*>::iterator it = find(_aiglist.begin(), _aiglist.end(), unused);
          map<unsigned, CirGate*>::iterator iter;
          iter = _gatelist.find(unused->getVar());
          if (it != _aiglist.end()) { _aiglist.erase(it); }
          if (iter != _gatelist.end()) { _gatelist.erase(iter); }

          /*
          if (unused->getType() == UNDEF_GATE) { gatetype = "UNDEF"; }
          if (unused->getType() == AIG_GATE) { gatetype = "AIG"; }
          cout << "Sweeping: " << gatetype << "(" << unused->getVar() << ") " << "removed..." << endl;
          */
          // delete
          delete unused;
        }
      }
    }
  }
}
```

CIRSWeep :

2. Remove Unused Gates :

For those gates that cannot reach Pos, that is, do not include in _DFSList, are redundant. Hence, compared with _TotalGateList, if the gate in _TotalGateList but not in _DFSList, we know that it can be swept, and because the gate is deleted, other lists which store the gate should also erase this gate.

```
void
CirMgr::sweep()
{
  // find unused input
  for (int i = 0 ; i < _pilist.size() ; i++){
    int count = 0;
    for (int j = 0 ; j < _dfslist.size() ; j++){
      if (_pilist.at(i) == _dfslist.at(j)) { count = 1; }
    }
    if (count == 0) { notUsed.push_back(_pilist.at(i)->getVar()); }
  }
  // Remove unused gates
  for (int i = 0 ; i < _totalList.size() ; i++){
    GateType type = _totalList.at(i)->getType();
    string gatetype = "";
    if (type == UNDEF_GATE || type == AIG_GATE){
      for (int j = 0 ; j < _dfslist.size() ; j++){
        if (_dfslist.at(j) == _totalList.at(i)) break;
        if (j == _dfslist.size()-1){
          CirGate* unused = _totalList.at(i);
          unsigned NOT_USE = unused->getVar();
          // delete at notUsed and fltFanins
          vector<unsigned>::iterator n = find(notUsed.begin(), notUsed.end(), NOT_USE);
          vector<unsigned>::iterator f = find(fltFanins.begin(), fltFanins.end(), NOT_USE);
          if (n != notUsed.end()) { notUsed.erase(n); }
          if (f != fltFanins.end()) { fltFanins.erase(f); }
          // delete at aiglist
          vector<CirAigGate*>::iterator it = find(_aiglist.begin(), _aiglist.end(), unused);
          map<unsigned, CirGate*>::iterator iter;
          iter = _gatelist.find(unused->getVar());
          if (it != _aiglist.end()) { _aiglist.erase(it); }
          if (iter != _gatelist.end()) { _gatelist.erase(iter); }

          /*
          if (unused->getType() == UNDEF_GATE) { gatetype = "UNDEF"; }
          if (unused->getType() == AIG_GATE) { gatetype = "AIG"; }
          cout << "Sweeping: " << gatetype << "(" << unused->getVar() << ") " << "removed..." << endl;
          */
          // delete
          delete unused;
        }
      }
    }
  }
}
```

CIROPTimize :

1. Recursively optimize :

First, call the function “Strash()” to merge same function gates.
Second, recursively call the function “Trivial Optimization” to delete some redundant gates

```
void
CirMgr::optimize()
{
   // structural hash
   strash();
   // trivial optimization
   for (int i = 0 ; i < _polist.size() ; i++){
      trivial_opt(_polist.at(i)->_fanin[0].gate());
   }
}
```

CIROPTimize :

2. Trivial Optimization :

When entering a gate, first determine what scenario it belongs to:

1. **Const0 + Input**
2. **Const1 + Input**
3. **Input + Input (same)**
4. **Input + Input (same but inverse)**

PS :

In scenario 2. because we do not have Const1, Const1 should be “Const0 + inverse”

```
void
CirMgr::trivial_opt(CirGate* const& gate){
  // recursive
  int follow = 0;
  for (int i = 0 ; i < gate->_fanin.size() ; i++){
    if (gate != 0){
      trivial_opt(gate->_fanin[i].gate());
    }
  }
  CirGateV out_1, out_2, in_1, in_2;
  if (gate->_fanout.size() == 1) { out_1 = gate->_fanout[0] ; out_2 = gate->_fanout[0]; }
  if (gate->_fanout.size() == 2) { out_1 = gate->_fanout[0] ; out_2 = gate->_fanout[1]; }
  if (gate->_fanin.size() == 2) { in_1 = gate->_fanin[0] ; in_2 = gate->_fanin[1]; }
  // determine const0
  int if_zero = -1;
  if (gate == Const0 && gate->_fanout[0].gate()->_fanin[0].gate() != gate->_fanout[0].gate()->_fanin[1].gate()){
  }
  // const1 + input (const1 = const0 + inv)
  if (if_zero == 1){…
  }
  // const0 + input (some gates will be defined-but-not-used if gate->fanin.size == 1)
  if (if_zero == 0){…
  }
  // input + input
  if (gate->_fanout.size() == 2 && gate->_fanout[0].gate() == gate->_fanout[1].gate() && (gate->_fanout[0].gate(
  }
  // input + inversed-input (consider new undefined-gate)
  if (gate->_fanout.size() == 2 && gate->_fanout[0].gate() == gate->_fanout[1].gate() && (gate->_fanout[0].gate(
  }
  // delete
  if (follow != 0){…
  }
}
```

CIROPTimize :

2. Trivial Optimization :

To distinguish case (1) and (2) :

1. Const0 + Input
2. Const1 + Input

I determine whether Const0 is inversed
If it does, set “if_zero = 1”
If it does not, set “if_zero = 0”

```
void
CirMgr::trivial_opt(CirGate* const& gate){
  // recursive
  int follow = 0;
  for (int i = 0 ; i < gate->_fanin.size() ; i++){
    if (gate != 0){
      trivial_opt(gate->_fanin[i].gate());
    }
  }
  CirGateV out_1, out_2, in_1, in_2;
  if (gate->_fanout.size() == 1) { out_1 = gate->_fanout[0] ; out_2 = gate->_fanout[0]; }
  if (gate->_fanout.size() == 2) { out_1 = gate->_fanout[0] ; out_2 = gate->_fanout[1]; }
  if (gate->_fanin.size() == 2) { in_1 = gate->_fanin[0] ; in_2 = gate->_fanin[1]; }
  // determine const0
  int if_zero = -1;
  if (gate == Const0 && gate->_fanout[0].gate()->_fanin[0].gate() != gate->_fanout[0].gate()->_fanin[1].gate()){
  }
  // const1 + input (const1 = const0 + inv)
  if (if_zero == 1){…
  }
  // const0 + input (some gates will be defined-but-not-used if gate->fanin.size == 1)
  if (if_zero == 0){…
  }
  // input + input
  if (gate->_fanout.size() == 2 && gate->_fanout[0].gate() == gate->_fanout[1].gate() && (gate->_fanout[0].gate(
  }
  // input + inversed-input (consider new undefined-gate)
  if (gate->_fanout.size() == 2 && gate->_fanout[0].gate() == gate->_fanout[1].gate() && (gate->_fanout[0].gate(
  }
  // delete
  if (follow != 0){…
  }
}
```

CIROPTimize :

3. AddToUndef :

In the cases of “Const0 + Input” and “Input + Input (same but inverse)”, because “Input” will be replaced by “Const0”, and hence let gates deeper than “Input” be “defined-but-not-used”.
Therefore, I set the function to store these gates.

```
void
CirMgr::AddToUndef(CirGate* const& undefGate){
  int count = 0;
  for (int i = 0 ; i < undefGate->_fanin.size() ; i++){
    if (undefGate->_fanin.size() == 0) {
      count = 1;
      vector<unsigned>::iterator it = find(notUsed.begin(), notUsed.end(), undefGate->getVar());
      if (it == notUsed.end()) { notUsed.push_back(undefGate->getVar()); }
    }
    if (count == 1){
      vector<unsigned>::iterator it = find(notUsed.begin(), notUsed.end(), undefGate->getVar());
      if (it == notUsed.end()) { notUsed.push_back(undefGate->getVar()); }
    }
    else { AddToUndef(undefGate->_fanin[i].gate()); }
  }
}
```

Appendix :

1. The code based on HW6 is copied from 朱哲廣 (B07901016)'s

2. I do not finish the parts of "Simulation" and "Fraig" …QAQ

3. My e-mail : b07901103@ntu.edu.tw

4. My phone : 0966-540-165

5. Comments on Ric's DSnP :

學期初在選這堂課的時候, 我躊躇了好幾天, 因為我自己知道我的程式能力偏弱, 加上大一上的計程課我是修Python的, 所以對於完全沒碰過C++的我, 很怕面對Ric負擔很重的作業量, 深怕自己作業一直交不出來…雖然我不像很多大神一樣很熟悉程式語法, 但我卻有一顆跟他們一樣喜歡打程式, 想要更精進自我的心態, 其實我原本還有考慮過系上複選必修或資工系的資結, 作業量輕很多, 但最後還是鼓勵自己, 要學, 就要學最好的, 我不知道鼓了多少勇氣才在大家紛紛退選跟停修時繼續撐下去, 超級感謝 Ric幾乎是用生命在教這堂課, 做了那麼多PPT, 加上出題目還要自己打reference code, 每個禮拜三都願意花時間陪我們到21:30以後等等, 其實我超級感動上了大學還有教授願意花這麼多時間將知識傳給學生, 所以我告訴自己完全沒有理由鬆懈, 在電機系, 翹掉必修課自己讀是家常便飯的事, 但我很欣慰的說, 只有資結, 我每堂都到, 每堂都聽到最後一刻, 而且每個禮拜雖然花了比別人多滿多的時間去打完一份作業, 但起碼, HW1~HW7我都有打到所有do file都過才交出去, 我完全沒想到可以從不會C++到自學C++到現在這種程度, 真的只有感動和感謝老師跟辛苦的助教!而且還好是身在電機系, 身邊的同學都是電神, 而且是那種無私的電神, 所以每一次作業都獲得了很多的指導和討論, 這大概是我能進步這麼快的主因吧~很可惜最後final project打不出甚麼東西, 雖然說結果也固然重要, 但起碼這一整個學期的收穫和努力的過程, 真的十分值得!老師和助教都辛苦了^^希望還能上到老師的其他課~