VLSI Testing PA2 Report b09901081 施伯儒

Circuit number	number of gates	number of total faults	number of detected faults	number of undetected faults	fault coverage	number of test vector	run time
C432	**245**	**604**	**108**	**496**	**17.88%**	**20**	**0.5s** **0.5s**
C499	**554**	**1278**	**1233**	**45**	**96.48%**	**74**	**0.1s** **0.2s**
C880	**545**	**1051**	**663**	**388**	**63.08%**	**62**	**0.3s** **0.3s**
C1355	**554**	**1646**	**686**	**960**	**41.68%**	**63**	**2.3s** **2.3s**
C2670	**1785**	**3000**	**2882**	**118**	**96.07%**	**156**	**0.7s** **0.7s**
C3540	**2082**	**4145**	**1224**	**2921**	**29.53%**	**95**	**18.9s** **18.9s**
C6288	**4800**	**7808**	**7681**	**127**	**98.37%**	**47**	**1.3s** **1.5s**
C7552	**5679**	**8100**	**7963**	**137**	**98.31%**	**267**	**3.3s** **3.4s**

Programming details and BONUS:

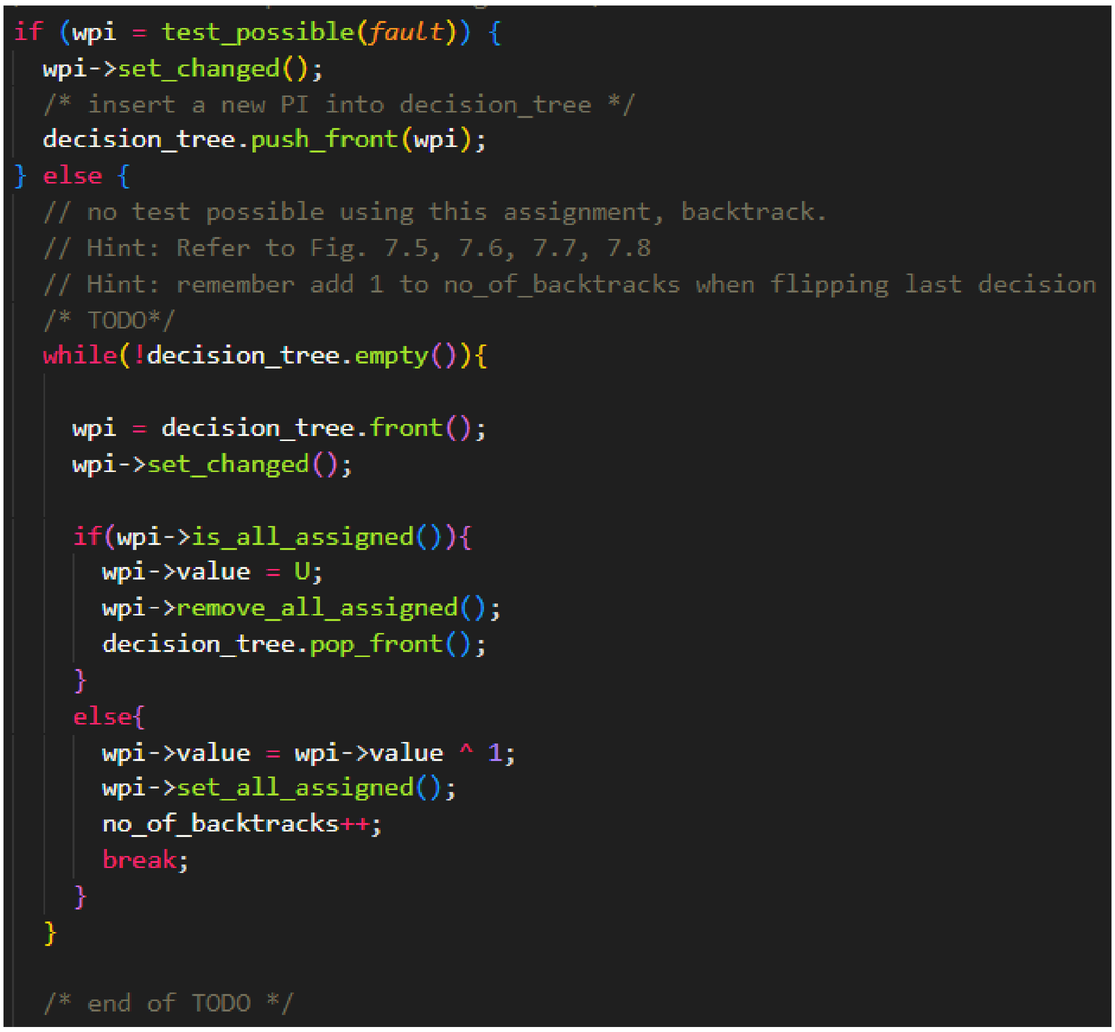
1.

This part of code is focusing on the backtrack in decision tree. First, we should put the code inside a loop ensuring decision tree is not empty. Then, we get the front element of the decision tree, check whether it is all assigned or not. If it is not all assigned, we should flip its value, set it all assigned and add 1 to number of backtracks.

Later, we should break this loop, and execute the code outside this if-else, which is simulating the circuit and trying to find a pattern.

If it still doesn’t satisfy the conditions to break the outer while loop, we may go back to backtrack again.

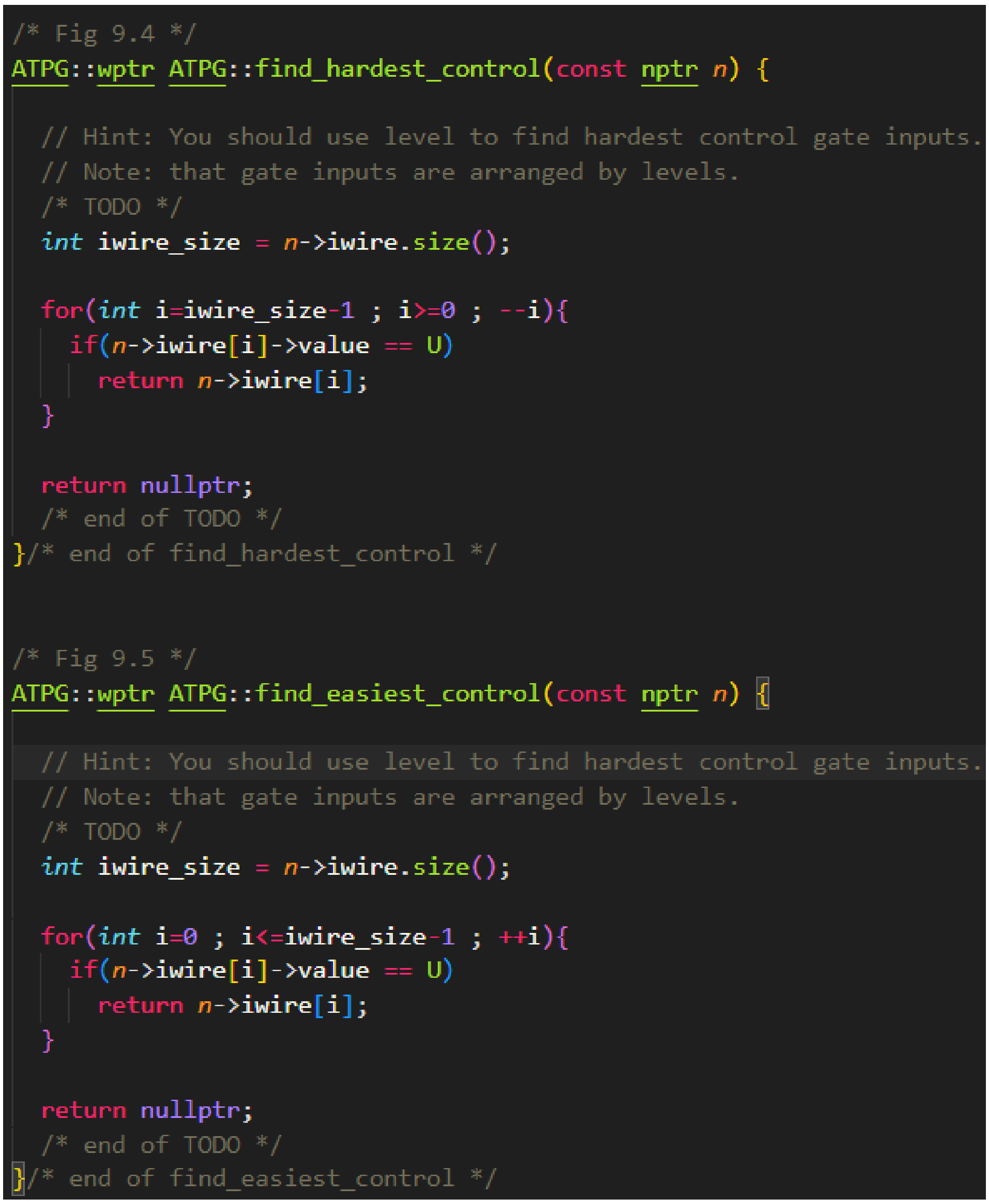
This time, if the wire is all assigned, we should pop this wire out of the decision tree and get next element from decision tree.



```
if (wpi = test_possible(fault)) {
  wpi->set_changed();
  /* insert a new PI into decision_tree */
  decision_tree.push_front(wpi);
} else {
  // no test possible using this assignment, backtrack.
  // Hint: Refer to Fig. 7.5, 7.6, 7.7, 7.8
  // Hint: remember add 1 to no_of_backtracks when flipping last decision
  /* TODO*/
  while(!decision_tree.empty()){

    wpi = decision_tree.front();
    wpi->set_changed();

    if(wpi->is_all_assigned()){
      wpi->value = U;
      wpi->remove_all_assigned();
      decision_tree.pop_front();
    }
    else{
      wpi->value = wpi->value ^ 1;
      wpi->set_all_assigned();
      no_of_backtracks++;
      break;
    }
  }

  /* end of TODO */
```

2.

This part of code follows a simple logic: Because all the wires in iwire is sorted by its level (check out level.cpp), the easiest/hardest wire is the first/last element in the iwire. But! We should make sure the value of the wire is unknown. So I iterate the vector from beginning/end to check which is the first one to be unknown.

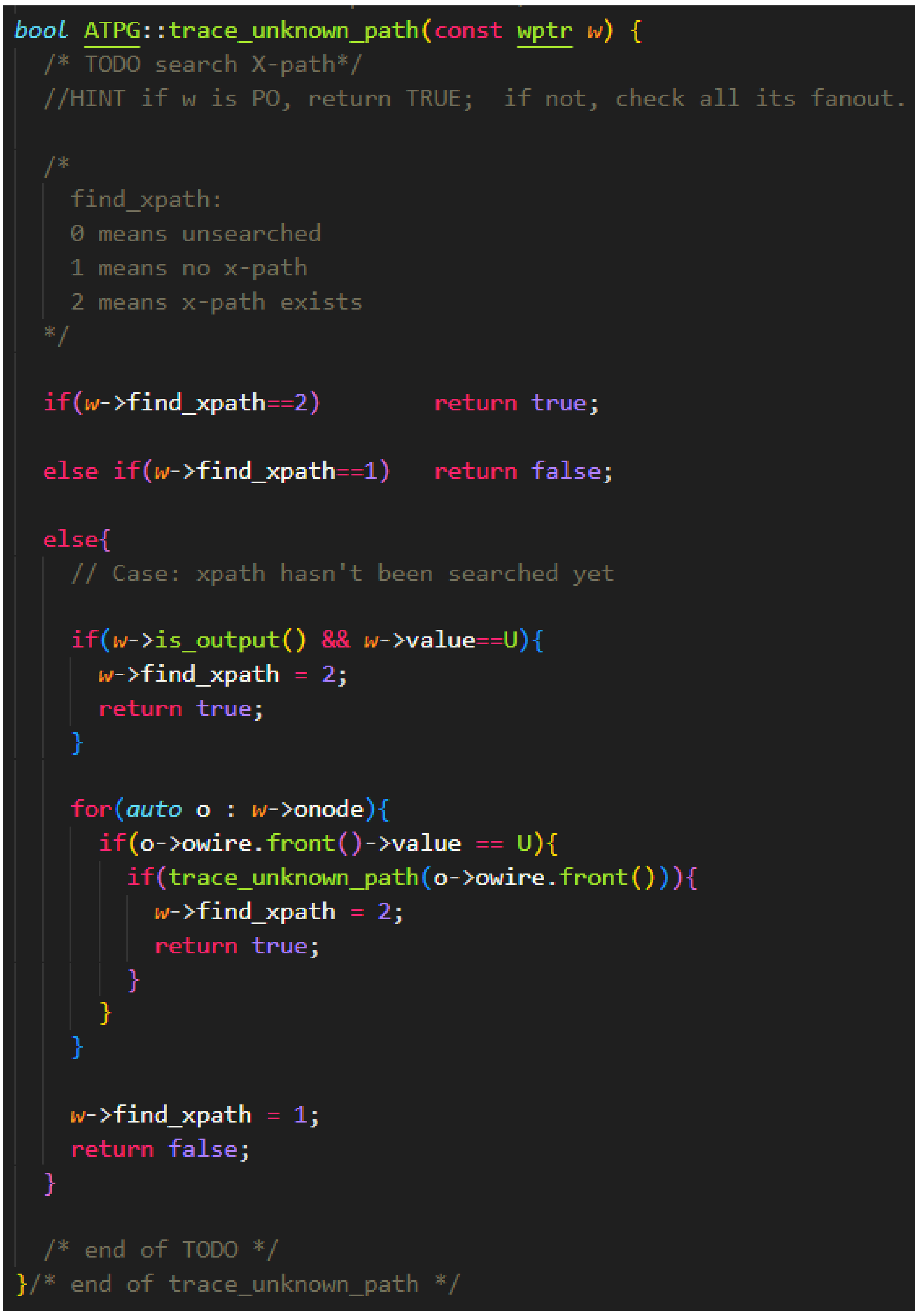


```
/* Fig 9.4 */
ATPG::wptr ATPG::find_hardest_control(const nptr n) {

  // Hint: You should use level to find hardest control gate inputs.
  // Note: that gate inputs are arranged by levels.
  /* TODO */
  int iwire_size = n->iwire.size();

  for(int i=iwire_size-1 ; i>=0 ; --i){
    if(n->iwire[i]->value == U)
      return n->iwire[i];
  }

  return nullptr;
  /* end of TODO */
}/* end of find_hardest_control */


/* Fig 9.5 */
ATPG::wptr ATPG::find_easiest_control(const nptr n) {

  // Hint: You should use level to find hardest control gate inputs.
  // Note: that gate inputs are arranged by levels.
  /* TODO */
  int iwire_size = n->iwire.size();

  for(int i=0 ; i<=iwire_size-1 ; ++i){
    if(n->iwire[i]->value == U)
      return n->iwire[i];
  }

  return nullptr;
  /* end of TODO */
}/* end of find_easiest_control */
```

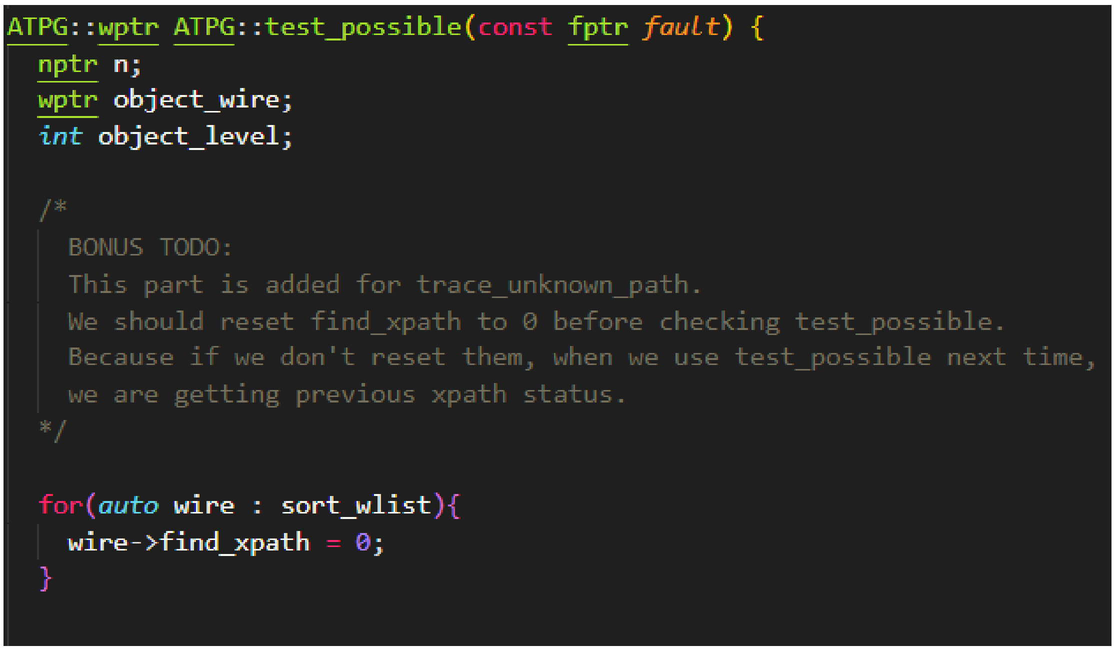
3.

BONUS part: I add find_xpath in the wire class and initialize the value to be 0. Also, I reset all the wires' find_xpath value to 0 whenever function test_possible is

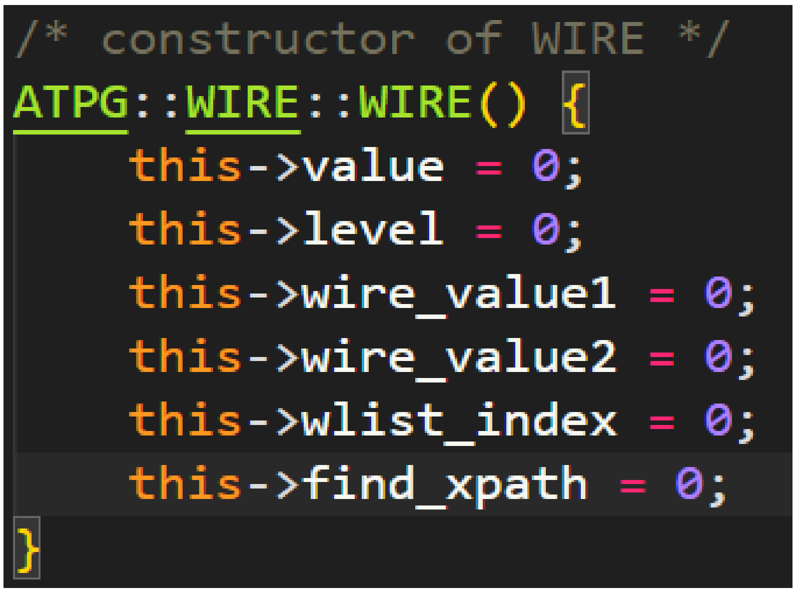
called. The reason is that I use find_xpath to record the status of x-path: 0 means not searched yet, 1 means no x-path, 2 means x-path exists.
As you can see, I set find_xpath to 2 when the wire reaches PO or the output node has x-path. And if all the nodes don’t have the x-path, I will set the wire’s find_xpath to 1.
This makes sure that all the wires I already iterated get a value recording their status, so if we meet the wire again, we can return the function immediately, avoiding recursively iterating the same path.
Therefore, the total time complexity should be O(n).



```cpp
bool ATPG::trace_unknown_path(const wptr w) {
  /* TODO search X-path*/
  //HINT if w is PO, return TRUE;  if not, check all its fanout.

  /*
    find_xpath:
    0 means unsearched
    1 means no x-path
    2 means x-path exists
  */

  if(w->find_xpath==2)         return true;

  else if(w->find_xpath==1)    return false;

  else{
    // Case: xpath hasn't been searched yet

    if(w->is_output() && w->value==U){
      w->find_xpath = 2;
      return true;
    }

    for(auto o : w->onode){
      if(o->owire.front()->value == U){
        if(trace_unknown_path(o->owire.front())){
          w->find_xpath = 2;
          return true;
        }
      }
    }

    w->find_xpath = 1;
    return false;
  }

  /* end of TODO */
}/* end of trace_unknown_path */
```



```
ATPG::wptr ATPG::test_possible(const fptr fault) {
  nptr n;
  wptr object_wire;
  int object_level;

  /*
    BONUS TODO:
    This part is added for trace_unknown_path.
    We should reset find_xpath to 0 before checking test_possible.
    Because if we don't reset them, when we use test_possible next time,
    we are getting previous xpath status.
  */

  for(auto wire : sort_wlist){
    wire->find_xpath = 0;
  }
```



```
/* constructor of WIRE */
ATPG::WIRE::WIRE() {
    this->value = 0;
    this->level = 0;
    this->wire_value1 = 0;
    this->wire_value2 = 0;
    this->wlist_index = 0;
    this->find_xpath = 0;
}
```