# VLSI Testing PA1 Report　　B09901081 施伯儒

| Circuit number | number of test vector | number of gates | number of total faults | number of detected faults | number of undetected faults | fault coverage |
|---|---|---|---|---|---|---|
| C499 | 66 | 554 | 2390 | 2263 | 127 | 94.69% |
| C880 | 65 | 545 | 2104 | 1254 | 850 | 59.60% |
| C1355 | 63 | 554 | 2726 | 1702 | 1024 | 62.44% |
| C2670 | 135 | 1785 | 6520 | 6278 | 242 | 96.29% |
| C3540 | 98 | 2082 | 7910 | 2424 | 5486 | 30.64% |
| C6288 | 42 | 4800 | 17376 | 17109 | 267 | 98.46% |
| C7552 | 289 | 5679 | 19456 | 19144 | 312 | 98.40% |

## Programming Details:

1.

This part is sim.cpp.

I just simply follow the instructions, trying to evaluate every node in the circuit.

```
/* TODO */
// Schedule every gate connected to a changed input.
// evaluate() every scheduled gate & propagate any changes.
// walk through all wires in increasing order.
// Because the wires are sorted according to their levels,
// it is correct to evaluate the wires in increasing order.
/*
 * For event-driven simulation, we set_change() after we inject new values (all ncktin wires)
 * So, if a wire value is_changed(), we should clear this flag by remove_changed() and set_schedule() to wait for simulation.
 * We can then start to simulate circuit by sort_wlist, which it is sorted by level.
 * Use evaluate() on connected node and schedule next wire if value changed.
 * Hint: You might need two for loops.
 */

for(int i=ncktin; i<nckt; i++){
    for(auto node : sort_wlist[i]->inode){
      evaluate(node);

      if(node->owire.front()->is_changed()){
        node->owire.front()->remove_changed();
        node->owire.front()->set_scheduled();
      }
    }
}
/*end of TODO*/
```

2.

This part is fault_sim_a_vector in faultsim.cpp.

In this part, we need to check whether the faulty wire's good value and false value are equal or not.

First, we need to check the wire is PO, otherwise the discussion on whether the fault is detected is meaningless.

Second, we need to find out which fault is detected by this pattern, so I use XOR to filter out the difference between good value and false value, and then iterate all the fault, checking all the conditions in the if-statement.
And I use Mask to filter the right position for specific fault, also I remember to check the value should not be Unknown.

```
/*TODO*/

/*
 * After simulation is done,if wire is_output(), we should
 * If these two values are different and they are not unknou
 * Since we use two-bit logic to simulate circuit, you can
 * After that, don't forget to reset faulty values (wire_va
 */

if(w->is_output()) {
  int detected_value = w->wire_value_f ^ w->wire_value_g;

  for(int i=0; i<num_of_fault; i++){
    if (  ((detected_value & Mask[i]) != 0)
      &&  ((w->wire_value_f & Mask[i]) != Unknown[i])
      &&  ((w->wire_value_g & Mask[i]) != Unknown[i]) )
    {
      simulated_fault_list[i]->detect = true;
    }
  }
}
w->wire_value_f = w->wire_value_g;
```

3.
This part is get_faulty_wire from faultsim.cpp.
We need to check whether fault can be propagated or not.
There are many gates in the code, so I only screenshot AND gate for example.
ex: AND gate -> if we want the fault to be passed to the output, input should not be zero or unknown, of course, we must not consider the fault input wire.

```
case AND:
  /*TODO*/
  /* To check if this fault can propagate successfully, we need to check every gate input of this
   * AND gate. You sould avoid checking the faultly input. If any input is zero or unknown, fault
   * f can't propagate, and you sould set (is_faulty) to false
   */

  for(auto wire: f->node->iwire){
    if((wire->value == 0 || wire->value == U) && (wire!=sort_wlist[f->to_swlist])){
      is_faulty = false;
    }
  }

  /*end of TODO*/
  /* AND gate input stuck-at one/zero fault is propagated to
     AND gate output stuck-at one/zero fault */
  if (f->fault_type == 0)
    fault_type = STR;
  else
    fault_type = STF;
  break;
```

4.

This part is inject_fault_value from faultsim.cpp.

We need to inject fault on a specific wire, so first we call inject_fault_at, in order to change the state inside the wire.

Second, I use Mask to set specific bits in the wire's fault value according to what the fault type is.

```
void ATPG::inject_fault_value(const wptr faulty_wire, const int &bit_position, const int &fault_type) {
  /*TODO*/
  //Hint: use mask to inject fault to the right position
  /* Use mask[] to perform bit operation to inject fault (STUCK1 or STUCK0) to the right position
   * Call inject_fault_at() to set the fault_flag of the injected bit position.
   */

  faulty_wire->inject_fault_at(bit_position);

  if (fault_type == STUCK0) {
    faulty_wire->wire_value_f &= ~Mask[bit_position];
  }
  else if (fault_type == STUCK1){
    faulty_wire->wire_value_f |=  Mask[bit_position];
  }

  /*end of TODO*/
}/* end of inject_fault_value */
```

## Bug I encounter:

At first, I didn't know operator != is prior to operator &, so I got some strange bugs. And I fixed it after simply adding some parentheses. 🙁

```
if(w->is_output()) {
  unsigned int detected_value = w->wire_value_f ^ w->wire_value_g;
  //(detected_value & Mask[i] != 0)


  for(int i=0; i<num_of_fault; i++){

    int a = w->wire_value_f & Mask[i];
    int b = w->wire_value_g & Mask[i];

    if ( a!=b
    &&  ((w->wire_value_f & Mask[i]) != Unknown[i])
    &&  ((w->wire_value_g & Mask[i]) != Unknown[i])
    // &&  (a  != Unknown[i])
    // &&  (b  != Unknown[i])
    )
    {
      simulated_fault_list[i]->detect = true;
    }
  }
}
w->wire_value_f = w->wire_value_g;
```