107-1 VLSI Testing

Programming Assignment #1

電子所碩一 R07943107 徐晨皓

1. **Please fill in the following table in your report.**

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%
C1355	63	554	2726	1702	1024	62.44%
C6288	42	4800	17376	17109	267	98.46%
C7552	289	5679	19456	19144	312	98.40%

2. **Please print out the critical parts of your code and explain it.**

(1) *sim.cpp*

In this part, we perform the fault-free simulation, which is claimed to be event-driven. However, from the code and the hints from TAs, this function is not exactly event-driven since all input gates are assumed to have the value change (marked as *CHANGED*) when the primary inputs are assigned by new test vectors.

```
/* for every input */
/*TODO1*/
//Hint:For every primary input. schedule the gates connected to it
//---------------------------------------- hole ----------------------------------------
for (i = 0; i < ncktin; ++i)                                    // For every primary input
  for (j = 0, nout = cktin[i]->onode.size(); j < nout; ++j)     // Schedule the gates connected to it
    cktin[i]->onode[j]->flag |= SCHEDULED;                      // Schedule the gates connected to it
//--------------------------------------------------------------------------------------
/*TODO1*/

/*TODO2*/
//Hint:
/* 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. */
//---------------------------------------- hole ----------------------------------------
for (i = 0; i < nckt; ++i)                                      // Walk through all wires in increasing order
  if (sort_wlist[i]->inode.front()->flag & SCHEDULED) {         // Evaluate every scheduled gate
    sort_wlist[i]->inode.front()->flag &= ~SCHEDULED;           // Reset SCHEDULED flag
    evaluate(sort_wlist[i]->inode.front());                     // Evaluate every scheduled gate
    if (sort_wlist[i]->flag & CHANGED)                          // Propagate any changes
      for (j = 0, nout = sort_wlist[i]->onode.size(); j < nout; ++j) // Propagate any changes
        sort_wlist[i]->onode[j]->flag |= SCHEDULED;             // Propagate any changes
  }
//--------------------------------------------------------------------------------------
/*TODO2*/
```

(2) *fault_sim.cpp*

(a) Fault injection

In this part, the faults are injected into faulty wires by change the

value of the wires. If the fault type is stuck-at-0, the value the faulty wire will be changed into 0; otherwise, if the fault type is stuck-at-1, the value of the faulty wire will be converted into 1.

```
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
  //-------------------------------------- hole ----------------------------------------
  faulty_wire->fault_flag |= Mask[bit_position];
  if (fault_type == STUCK1)
    faulty_wire->wire_value2 |= Mask[bit_position];
  else if (fault_type == STUCK0)
    faulty_wire->wire_value2 &= ~Mask[bit_position];
  else
    fprintf(stderr,"Ignore the fault type (%d)!\n", fault_type);
  //-----------------------------------------------------------------------------------------
  /*TODO*/
}/* end of inject_fault_value */
```

(b) Fault detection

In this part, if we detect that the faulty value of an output wire is different from its fault-free value, then we mark the fault as detected and drop it out from the fault list.

```
/*TODO*/
//Hint:Use mask to get the value of faulty wire and check every fault in packet
//-------------------------------------- hole ----------------------------------------
if (w->flag & OUTPUT) {
  for (i = 0; i < num_of_fault; ++i) {
    const int v1 = w->wire_value1 & Mask[i]; // good value
    const int v2 = w->wire_value2 & Mask[i]; // faulty value
    if ((v1 != v2) && (v1 != Unknown[i]) && (v2 != Unknown[i])) {
      simulated_fault_list[i]->detect = TRUE;
    }
  }
}
w->wire_value2 = w->wire_value1; // IMPORTANT! remember to reset the wires' faulty values
                                 // back to fault-free values.
//-----------------------------------------------------------------------------------------
/*TODO*/
```

3. **(10% bonus) In our parallel fault simulation algorithm, faults will be dropped once they have been detected. Now, we would like to support *N*-detect in our fault simulation, and that mean every fault should be detected *N* times before dropping. You should support the command below,**

./atpg −fsim < pattern file > -ndet < *N* > < circuit file >

***N* is a number from 1 to 8. For example,**

./atpg −fsim ../sample_circuits/c17.input -ndet 3 ../sample_circuits/c17.ckt

When you print *vector[i] detects m faults* on the screen. Please make sure that these *m* faults have been detected for *N* times.

I have completed the N-detect in fault simulation. The table below shows the experimental results of the fault coverage by applying N-detect fault simulation

on c7552.ckt. We can see that the fault coverage decreases as N increases.

N	Fault coverage (%)
1	98.40
2	94.50
3	91.90
4	90.27
5	88.28
6	86.71
7	84.88
8	83.21