VLSI Testing PA3 Report b09901081 施伯儒

Circuit number	Number of gates	number of total TDFs	number of detected faults	number of undetected faults	transition delay fault coverage
C17	6	34	23	11	67.64%
C432	245	1110	3	1107	0.27%
C499	554	2390	1552	838	64.93%
C880	545	2104	792	1312	37.64%
C1355	554	2726	593	2133	21.75%
C2670	1785	6520	4668	1852	71.59%
C3540	2082	7910	1142	6768	14.43%
C6288	4800	17376	16532	844	95.14%
C7552	5679	19456	17421	2035	89.54%

Programming details:

Transition delay fault simulation is really similar as fault simulation, so I copy some part of code in faultsim.cpp and modify them to fit the procedure of TDF simulation.

1. TDF simulation

```
void ATPG::transition_delay_fault_simulation(int &total_detect_num) {
    int i;
    int current_activated_num = 0;
    int current_detected_num = 0;

    /* for every vector */
    for (i = vectors.size() - 1; i >= 0; i--) {
        tdfault_sim_a_vector(vectors[i], current_activated_num);
        tdfault_sim_a_vector2(vectors[i], current_detected_num);
        total_detect_num += current_detected_num;
        fprintf(stdout, "vector[%d] detects %d faults (%d)\n", i, current_detected_num, total_detect_num);
    }
}
```

We can see the code above resembles the code in faultsim.cpp, but we need to sim the vector twice to detect the TDFs.

2. TDF sim a vector

```
void ATPG::tdfault_sim_a_vector(const string &vec, int &num_of_activated){
  int nckt = sort_wlist.size();

  /* for every input, set its value to the current vector value */
  for (int i = 0; i < cktin.size(); i++) {
    cktin[i]->value = ctoi(vec[i]);
  }

  /* initialize the circuit - mark all inputs as changed and all other
   * nodes as unknown (2) */
  for (int i = 0; i < nckt; i++) {
    if (i < cktin.size()) {
      sort_wlist[i]->set_changed();
    } else {
      sort_wlist[i]->value = U;
    }
  }

  //simultaion
  sim();
  if (debug) { display_io(); }

  for(auto f : flist_undetect){
    if(f->fault_type == sort_wlist[f->to_swlist]->value){
        f->activate = 1;
        num_of_activated++;
    }
    else
        f->activate = 0;
  }
}
```

As you can see, this part of code focuses on good simulation, and after the simulation, we should iterate through the fault list to check which fault is activated.

How to check which fault is activated?

If fault type is identical to the corresponding wire's value, then it is activated.

3. TDF sim a vector 2

```
//vector2
for(int i=0; i<cktin.size(); i++){
  if(i==0)
      cktin[i]->value = ctoi(vec[vec.size()-1]);
  else
      cktin[i]->value = ctoi(vec[i-1]);
}
```

```
if (f->activate && f->fault_type != sort_wlist[f->to_swlist]->value ) {
```

This function runs fault simulation and drops fault, so we can almost copy the corresponding function from faultsim.cpp. Because pasting all the code is tedious, I just simply point out the only two parts I modify.

First, we need to get a new vector from old vector, and circular shifting a bit can simply get what we want.

Second, when we check whether the current fault is detected, we should make sure it has been activated.

4. TDF generate fault list

```
void ATPG::generate_tdfault_list() {
  int fault_num;
  nptr n;
  fptr_s f;

  /* walk through every wire in the circuit*/
  for (auto w : sort_wlist) {
    n = w->inode.front();

    /* for each gate, create a gate output STR fault */
    f = move(fptr_s(new(nothrow) FAULT));
    if (f == nullptr) error("No more room!");

    f->node = n;
    f->io = GO;      // gate output SA0 fault
    f->fault_type = STR;
    f->to_swlist = w->wlist_index;
    f->eqv_fault_num = 1;     // GO SA0 fault itself

    if(n->type == OUTPUT) continue;

    if (n->type == NOT || n->type == BUF){
        for (wptr iw : n->iwire){
            if (iw->onode.size() > 1)
                f->eqv_fault_num++;
        }
    }
```

```
flist_undetect.push_front(f.get()); // initial undetected fault list contains all faults
flist.push_front(move(f));          // push into the fault list



/* for each gate, create a gate output STF fault */
f = move(fptr_s(new(nothrow) FAULT));
if (f == nullptr) error("No more room!");

f->node = n;
f->io = GO;
f->fault_type = STF;
f->to_swlist = w->wlist_index;
f->eqv_fault_num = 1;

if(n->type == OUTPUT) continue;

if (n->type == NOT || n->type == BUF){
    for (wptr iw : n->iwire){
        if (iw->onode.size() > 1)
            f->eqv_fault_num++;
    }
}

flist_undetect.push_front(f.get()); // initial undetected fault list contains all faults
flist.push_front(move(f));          // push into the fault list
```

```
/*if w has multiple fanout branches */
if (w->onode.size() > 1) {

  for (nptr nptr_ele: w->onode) {

    //we do not create fault in the NOT BUF gate input
    if (nptr_ele->type == BUF || nptr_ele->type == NOT)
        continue;


    /* create STR */
    f = move(fptr_s(new(nothrow) FAULT));
    if (f == nullptr) error("No more room!");
    f->node = nptr_ele;
    f->io = GI;
    f->fault_type = STR;
    f->to_swlist = w->wlist_index;
    f->eqv_fault_num = 1;
    /* f->index is the index number of gate input,
        which GI fault is associated with*/
    for (int k = 0; k < nptr_ele->iwire.size(); k++) {
        if (nptr_ele->iwire[k] == w) f->index = k;
    }
    num_of_gate_fault++;
    flist_undetect.push_front(f.get());
    flist.push_front(move(f));
```

```
        /* create STF */
        f = move(fptr_s(new(nothrow) FAULT));
        if (f == nullptr) error("No more room!");
        f->node = nptr_ele;
        f->io = GI;
        f->fault_type = STF;
        f->to_swlist = w->wlist_index;
        f->eqv_fault_num = 1;
        for (int k = 0; k < nptr_ele->iwire.size(); k++) {
            if (nptr_ele->iwire[k] == w) f->index = k;
        }
        num_of_gate_fault++;
        flist_undetect.push_front(f.get());
        flist.push_front(move(f));
      }
    }
  }

  flist.reverse();
  flist_undetect.reverse();

  /*walk through all faults, assign fault_no one by one  */
  fault_num = 0;
  for (fptr f: flist_undetect) {
    f->fault_no = fault_num;
    fault_num++;
    num_of_tdf_fault += f->eqv_fault_num;
  }

}
```

This part of code mostly is copied from initfist.cpp, but we don’t need the fault-collapsing part; therefore, we must delete all the fault-collapsing.

When I implement this function, I find out there is a strange message “something is fishy”.

```
switch (f->node->type) {

  /* this case should not occur,
   * because we do not create fault in the NOT BUF gate input */
  case NOT:
  case BUF:
    fprintf(stdout, "something is fishy(get_faulty_net)...\n");
    break;
```

It turns out that I put input faults of NOT and BUF into fault list.

In order not to modify other files and maintain same TDF fault number as the golden ATPG, I add some code below.

```
//we do not create fault in the NOT BUF gate input
if (nptr_ele->type == BUF || nptr_ele->type == NOT)
    continue;
```

When wire is a fanout stem and its output node is BUF or NOT, I will ignore it, not creating a fault on that wire. But, this may result in wrong TDF numbers and wrong fault coverage, therefore, I add another part of code in the process of getting STR, STF faults from iterating the whole fault list.

```
if (n->type == NOT || n->type == BUF){
    for (wptr iw : n->iwire){
        if (iw->onode.size() > 1)
            f->eqv_fault_num++;
    }
}
```

In this way, if the wire's input node is NOT or BUF, I will check out the size of the input wire's output node.

If it is bigger than 1, I will add extra 1 to the fault's equivalent fault number.

Here is a example explaining why I do this:

Wire A --1-- → AND-- wire B

--2-- → NOT-- wire C

(suppose wire A is a fanout stem connecting to two nodes)

When we meet wire C in the sorted wire list, because NOT's input wire's output node size is bigger than 1, The equivalent fault number of wire C is 2.

When we deal with the fanout stem, the fault on the input of NOT will be ignored, but its fault is already counted in the previous section.

This simple modification gives same result as the golden ATPG, and I don’t need to modify other files.