Programming Assignment #3
TDF Simulator

Introduction:

As the leading academic EDA tool provider, we provide our customers with a PODEM-based *NTU ATPG* (automatic test pattern generator) for single stuck-at faults. This ATPG system is very good for stuck-at faults so our customers hope that we can also support *transition delay fault* (TDF) simulation. Our customers wrap their *CUT* (circuit under test) by flip-flops, which are stitched in a scan chain. Therefore, we need to chain the original primary inputs (PI) and primary outputs (PO) so that we can do the TDF simulation by using *launch-on-shift* (LOS) technique. In this PA, we first show you how to run *tdfsim* mode. Then, you are asked to build the whole TDF simulator.

Tutorial:

First, please enter the bin directory, and run the golden binary to see results of *ATPG* mode. Notice that you should run these commands on *Linux* platforms with *GCC* version 4.8 or newer. In case you see any compilation error, please check if your platform supports *C++11*. If not, you can update your OS or GCC version. The code can be compiled correctly on edaU1 of edaunion. Simply type the following commands.

```
cd bin
./golden_tdfsim -tdfsim ../tdf_patterns/c17.pat ../sample_circuits/c17.ckt
```

Then you will see results generated by *tdfsim* on the screen.
Second, please enter the *src* directory, then compile the source code by typing

```
cd src
make
```

If compilation is successful, an executable file '`atpg`' is generated. In the same directory, you can run this software in *tdfsim* mode by typing the following command.

```
./atpg -tdfsim ../tdf_patterns/c17.pat ../sample_circuits/c17.ckt
```

Notice that you will see nothing happened because we do not build the simulator yet. Your job is to use what you have learned in the testing class to build the whole TDF simulator. Our customers expect to see the following information shown on the screen. Please note that because we do TDF simulation in reverse order, so the first pattern for simulation would be vector[7] instead of vector[0].

```
#Circuit Summary:
#---------------
#number of inputs = 5
#number of outputs = 2
#number of gates = 6
#number of wires = 11
#atpg: cputime for reading in circuit ../sample_circuits/c17.ckt: 0.0s 0.0s
#atpg: cputime for levelling circuit ../sample_circuits/c17.ckt: 0.0s 0.0s
#atpg: cputime for rearranging gate inputs ../sample_circuits/c17.ckt: 0.0s 0.0s
#atpg: cputime for creating dummy nodes ../sample_circuits/c17.ckt: 0.0s 0.0s
#atpg: cputime for generating fault list ../sample_circuits/c17.ckt: 0.0s 0.0s
vector[7] detects 3 faults (3)
vector[6] detects 5 faults (8)
vector[5] detects 3 faults (11)
vector[4] detects 1 faults (12)
vector[3] detects 3 faults (15)
vector[2] detects 0 faults (15)
```

Programming Assignment #3

```
vector[1] detects 4 faults (19)
vector[0] detects 4 faults (23)

# Result:
----------------------
# total transition delay faults: 34
# total detected faults: 23
# fault coverage: 67.647059 %
#atpg: cputime for test pattern generation ../sample_circuits/c17.ckt: 0.0s 0.0s
```

Figure 1. TDF simulation result

I/O Files:

We explain the I/O information using the following circuit, *c17*. Please note that our customers wrap the circuit so PIs and POs are stitched into a scan chain. The entrance of the chain is the first PI (*e.g. 1GAT*), and the exit of the chain is the last PO (*e.g. 23GAT*).

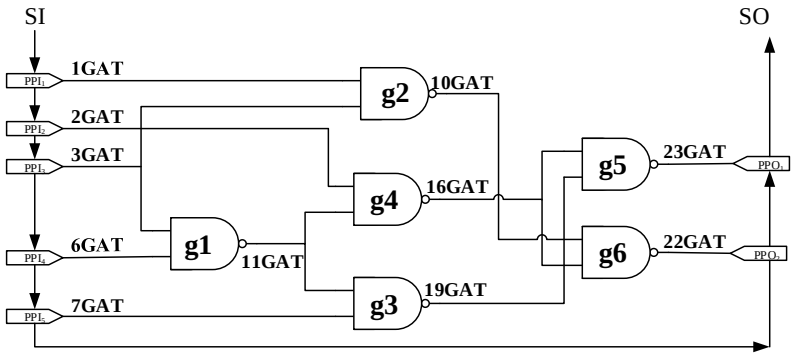

Figure 2. *c17* circuit

Notice that we chain the PIs and POs according to the order they appear in the file. For example, in *c17.ckt*, we chain the PIs and POs from *1GAT* to *23GAT*. So, 1GAT is now PPI_1 (pseudo primary input) and 2GAT is PPI_2, and 7GAT is PPI_5.

```
name C17.iscas
i 1GAT(0)
i 2GAT(1)
i 3GAT(2)
i 6GAT(3)
i 7GAT(4)

o 22GAT(10)
o 23GAT(9)

g1 nand 6GAT(3) 3GAT(2) ; 11GAT(5)
g2 nand 3GAT(2) 1GAT(0) ; 10GAT(6)
g3 nand 7GAT(4) 11GAT(5) ; 19GAT(7)
g4 nand 11GAT(5) 2GAT(1) ; 16GAT(8)
g5 nand 19GAT(7) 16GAT(8) ; 23GAT(9)
g6 nand 16GAT(8) 10GAT(6) ; 22GAT(10)
```

Figure 3. *c17* circuit file

The following file is our test pattern (*c17.pat*). The circuit information is shown at the beginning. Then, all test vectors are shown after the keyword "T", which composed of two parts. The two parts are separated by a space. First part is for our scan inputs (V_1), and the

Programming Assignment #3

second part is one additional bit for V_2 scan input in the LOS. Take the first test vector[0] T'00110 1' as example, the first part of test vector[0] will be '00110'. That means, PPI_1=0, PPI_2=0, PPI_3=1, PPI_4=1, PPI_5=0. After we shift in the scan chain with V_1, we then apply one capture clock to capture the response of CUT. Then we shift out the captured responses of the first part of test vector[0] and apply one more bit to scan in a '1' for V_2. That means, PPI_1=1, PPI_2=0, PPI_3=0, PPI_4=1, PPI_5=1. The same process will repeat for eight LOS test vectors.

```
#Circuit Summary:
#---------------
#number of inputs = 5
#number of outputs = 2
#number of gates = 6
#number of wires = 11
T'00110 1'
T'10111 0'
T'10001 1'
T'01000 0'
T'11011 1'
T'01100 0'
T'10000 1'
T'01111 0'
```

Figure 4. *c17* pattern file

Assignments:

Enter *src* directory and edit the file *tdfsim.cpp*. You can also modify other files, but you should clearly write down what you did in the report.

We give you a simple hint.

Step 1: Assign V_1 pattern and run a good simulation to check which faults are activated.

Step 2: Shift in a bit and obtain V_2 pattern.

Step 3: Do a single stuck-at fault simulation to check which faults (from step 1) are detected.

Step 4: Drop the faults that have been detected and continue to the next pattern.

Notice:

In previous programming assignment, we generate stuck-at fault in function, *generate_fault_list()*. This function will collapse equivalent faults into one fault. However, we can't collapse these TDF, so you should not do any fault collapsing in this assignment. Namely, please implement a new fault list generation function for TDF. Note: You should uncomment the "else atpg.generate_tdfault_list();" at line 80 in main.cpp and complete the atpg.generate_tdfault_list() function in tdfsim.cpp.

1) Please write a report to explain what you have done in *tdfsim.cpp*. Also, fill in the following table in your report, you should round off the fault coverage to second decimal place.

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					
C499					
C880					
C1355					

Programming Assignment #3

C2670					
C3540					
C6288					
C7552					

2) Please show how you generate the transition delay fault list and illustrate why we cannot collapse TDF in your report.

Grading:
80% correctness*
20% report

*Correctness will be evaluated by the results printed to the screen. Your result format should be the same as example shown in Figure 1.

Submission:
Make a directory <*student_id*>*_pa3*
Please copy 2 items /*src*, *report.pdf* into directory. Then submit a single *.*tgz* file to COOL system. Include everything so that your code can be easily compiled using 'make'. You can use the following command to compress a whole directory:

```
tar -zcvf <student_id>_pa3.tgz <dir>
```

Here's a refence file strcture:

R11943147_pa3/
├──*src/* #Including *.*cpp*, *.*h* and makefile only
└──*report.pdf* #Fill the table above and highlight your change of code

---

Copying source code results in zero grade for both students!
