Programming Assignment #1
Fault Simulator

Introduction:

We want to start up an EDA company that creates an NTU-ATPG *(automatic test pattern generation)* tool for single stuck-at faults. This NTU-ATPG system has two major functions: test pattern generation and fault simulation. For test pattern generation, it implements the PODEM algorithm [Goel 1981]. For fault simulation, it implements the parallel fault simulation algorithm. In this PA, we first show you how to run this tool in the *fault simulation* mode. Then, you are asked to fill in the holes in the fault simulator and logic simulator.

Tutorial:

First, please enter the bin directory, and run the golden binary to see the results of fault simulation mode. Notice that you should run these commands on Linux platforms. Otherwise, it may not work. Simply type the following commands.

```
cd bin
./golden_atpg -fsim  ../patterns/golden_c17.ptn ../sample_circuits/c17.ckt
```

Then you will see results generated by fault simulation.
Second, leave the bin directory, enter the *src* directory, and compile the source code by typing.

```
cd ..
cd src
make
```

Make sure that the version of the compiler must support C++ 11. In case you see any compilation error, please fix it before you proceed. If you succeed, an executable file '`atpg`' should be correctly generated. Then, in the same directory, run this software in fault simulation mode by typing the following command.

```
./atpg -fsim ../patterns/golden_c17.ptn ../sample_circuits/c17.ckt
```

(Please note that if you copy this line and paste it to the command line, you can fail because the '-' is a different character. Please remember to type the '-' on the keyboard.)
The number of total faults, detected faults, and fault coverage should be shown. Notice that the fault coverage is not correct now because there are some holes in the source code. Your job is to fill in the holes so that your outputs are the same as those generated by the golden binary.

I/O Files:

We explain the results using the following circuit, *c17*, which is an ISCAS (international symposium of circuit and system) benchmark circuit.

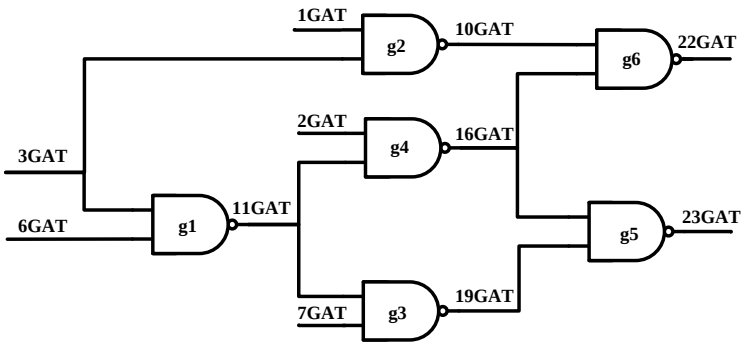

Figure 1. *c17* circuit

Programming Assignment #1

The following file is our circuit file (*c17.ckt*). At the beginning, the circuit name is shown after the keyword "*name*". Second, circuit primary inputs (PIs) go after the keyword *"i"*. Third, the circuit primary outputs (POs) go after the keyword *"o"*. Last, logic gates are named by the keyword *"g"* followed by a gate index. Then there goes a list of input wires followed by an output wire, separated by a semicolon. For example, *"g1 nand 6GAT(3) 3GAT(2) ; 11GAT(5)"* means NAND gate *g1* has two inputs: wire 6GAT and wire 3GAT. The output of NAND gate *g1* is wire 11GAT. This circuit has total 11 wires. Please note that a wire can have multiple fanout branches. For example, wire 11GAT has two branches, one feeds *g3* and the other feeds *g4*.

(The numbers in the parentheses are *total gates indices*. They are unique numbers for each PI, PO, or logic gate.)

```
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 2. *c17* circuit file

The following file is our test pattern (*golden_c17.ptn*), which is generated by our ATPG tool. The ATPG information is shown at the beginning. Then, all test vectors are shown after the keyword *"T"*. Finally, ATPG results are shown, including fault coverage, number of backtracks, etc.

```
#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.sim: 0.0s 0.0s
#atpg: cputime for levelling circuit ../sample_circuits/c17.sim: 0.0s 0.0s
#atpg: cputime for rearranging gate inputs ../sample_circuits/c17.sim: 0.0s 0.0s
#atpg: cputime for creating dummy nodes ../sample_circuits/c17.sim: 0.0s 0.0s
#number of equivalent faults = 22
#atpg: cputime for generating fault list ../sample_circuits/c17.sim: 0.0s 0.0s
T'00110'
T'10111'
T'10001'
T'01000'
T'11011'
T'01100'
T'10000'
T'01111'
#number of aborted faults = 0
#number of redundant faults = 0
#number of calling podem1 = 8
#total number of backtracks = 0
```

Programming Assignment #1

```
#FAULT COVERAGE RESULTS :
#number of test vectors = 8
#total number of gate faults = 34
#total number of detected faults = 34
#total gate fault coverage = 100.00%
#number of equivalent gate faults = 22
#number of equivalent detected faults = 22
#equivalent gate fault coverage = 100.00%
#atpg: cputime for test pattern generation ../sample_circuits/c17.sim: 0.0s 0.0s
```

Figure 3. *c17* input file (generated by ATPG)

Guidance for tracing code:

Tracing code is not an easy job but it is necessary for a young EDA engineer. You can see the following advice to trace the source code.

1. Read both *src/readme* and *src/atpg.h* files for introduction about the data structure for this ATPG.

2. Then read *src/main.cpp* for the main function of this ATPG. If you want, you can also read *src/input.cpp* and *src/level.cpp* to understand the circuit parsing and levelization. But you do not need to modify them in this programming assignment.

3. Then read *src/sim.cpp* and *src/faultsim.cpp* carefully. In both files, you can search "TODO" to find the part you need to fill. In *sim.cpp, sim()* is a function performing good simulation. In *faultsim.cpp, fault_simulate_vectors()* is a function that will simulate ALL patterns provided in the pattern file. *fault_sim_a_vector()* is a function that simulates a single pattern. The function *inject_fault_value*() injects a fault into the circuit, which is called by *fault_sim_a_vector().* The function *fault_sim_evaluate()* is called by *fault_sim_a_vector()*; it will evaluate a wire's value and schedule new events if its value changes.

Assignments:

Enter *src* directory and find the file *sim.cpp* and *faultsim.cpp*, there are some holes in these files. You can simply type the keyword "TODO" to find out the holes. After you fill in these holes, your results should be the same as our golden results.

1) Please write a report to explain what you have done in this PA. 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						
C1355						
C6288						
C7552						

2) (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 means every fault should be detected *N* times before the fault is dropped. A fault is detected

Programming Assignment #1

N times means that *N* different patterns detect a fault. (Please note that a pattern can detect a fault only once, even though its fault effects are propagated to many outputs.) 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 ../patterns/golden_c17.ptn -ndet 3 ../sample_circuits/c17.ckt
```

Grading:

80% correctness*

20% report

10% bonus

*Please note that your correctness will be evaluated by the results printed on the screen, which is already done by functions *fault_simulate_vectors()* and *compute_fault_coverage()*. For the assignment#1 (*i.e. N* =1), when you print "*vector[i] detects m faults"* on the screen, please make sure that these *m* faults are detected for the first time by the *vector[i]*. Similarly, for assignment #2 the *N*-detect bonus, "*vector[i] detects m faults"* means that these *m* faults are detected for the N_{th} times by *vector[i]*.

Submission:

Make a directory <*student_id*>*_pa1*

Please copy 2 items /*src*, *report.pdf* into the directory. Then submit a single *.*tgz* file to the 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>_pa1.tgz <dir>
```

Here's a reference file structure:

r10943095_pa1/

├──*src/* #Including *.*cpp*, *.*h* and makefile only

└──*report.pdf* #Fill the table above and highlight your change of code

(NOTE: you do not need to submit patterns or circuits.)

Reference:

[Goel 1981] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," IEEE Trans. on Computers, Vol. 30, No.3, pp215-222, 1981.

CAUTION!! We will check for plagiarism. Copying source code results in zero grade for both students!!