Programming Assignment #2

PODEM

Introduction:

We provide the CPP source codes of an ATPG (automatic test pattern generation) program for single stuck-at faults. This ATPG system has two major functions: test pattern generation and fault simulation. For test pattern generation, it implements the PODEM algorithm [Goel 1981]. In this PA, we first show you how to run *ATPG* mode. Then, you are asked to fill in the holes in the PODEM code.

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. Otherwise, it may not work. Simply type the following commands.

```
cd bin
./golden_atpg ../sample_circuits/c17.ckt > ../reports/c17.input
```

Then you will see results generated by PODEM.

Second, leave the bin directory, enter the *src* directory, and compile the source code by typing.

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

This code should be compiled correctly on Linux machines. 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 also be compiled correctly on edaU1 of edaunion. If you succeed, an executable file '`atpg`' is generated. Then, in the same directory, run this software in ATPG mode by typing the following command.

```
./atpg ../sample_circuits/c17.ckt > ../reports/c17.input
```

The number of total faults, detected faults, and fault coverage should be showed in *./reports/c17.input*. 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*.

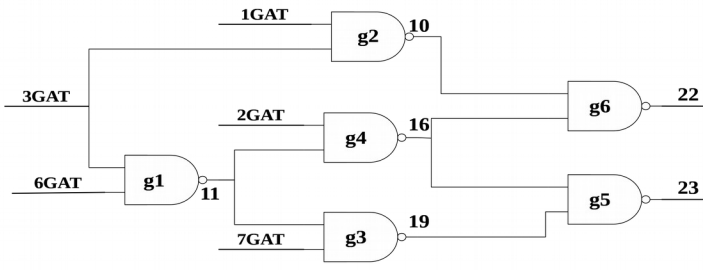

Figure 1. *c17* circuit

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, the rest of gates go

Programming Assignment #2

after the keyword “g” and its gate index. Take “g1 nand 6GAT(3) 3GAT(2) ; 11GAT(5)” for example, it means g1 is a NAND gate, and inputs of this NAND gate are signal 6 and signal 3, and output of this NAND gate is signal 11. The numbers in the parentheses are total gates index, which can be ignored here.

```
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 (*c17.input*), which is generated by our ATPG tool. The circuit data is shown at the beginning. Then, all test vectors are shown after the keyword “T”. The other ATPG results are shown after test vectors: 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 = 28
#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
#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 = 28
```

Programming Assignment #2

```
#number of equivalent detected faults = 28
#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)

Assignments:

Before you start, you should read the *readme* file that explains important data structure of this PODEM code. Please fill in the holes in the fault simulation code. Enter *src* directory and find the file *podem.cpp*, there are five holes in this file. You can simply type the keyword "TODO" to find out the holes. You are asked to complete the following five function,

1) *find_pi_assignment*, this function can find which input we need to imply.
2) *find_easiest_control*, this function can find the input with least level.
3) *trace_unknown_path*, this function can do the x-path tracing taught in class.
4) *check_test*, this function can check whether the fault effect reach output.
5) *set_uniquely_implied_value*, this function can set the uniquely implied value to PI, please trace the code in *backward_imply* and recursively imply the value to PI.

After you fill in the holes, your result should be same as our golden results.

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

circuit number	number of gates	number of total faults	number of detected faults	number of undetected faults	fault coverage	number of test vector	run time
C432							
C499							
C880							
C1355							
C2670							
C3540							
C6288							
C7552							

2) Please show the critical parts of your code and explain it in your report.

Bonus: Please analyze the complexity of function, *trace_unknown_path*. Can you implement this function with O(n) complexity where n is the number of nodes? What is the tradeoff of your implementation? Please explain what you did in your report.

Hint: *trace_unknown_path* can be implemented easily by recursively searching the unknown fanout of each node. However, the run time may be exponential growth if you trace the same path many times. You can reduce the search space by recording the path (or node) you have traced.

Grading:

80% correctness

20% report

20% bonus

Submission:

Make a directory *<student_id>_pa2*

Programming Assignment #2

Please copy 3 items /*src*, *report*, *readme* into directory. Then submit a single *.*tgz* file to CEIBA system. Please submit your code on *ceiba*. 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 <filename>.tgz <dir>`

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.

Copying source code results in zero grade for both students!
