ECE 353
Final Project Report

Group 1

Christopher Wilcox
Matthew Asplund

December 8, 2011

Table of Contents:

Part I: Combinational Test Generation

1. Problem Statement:

We are presented with a circuit with many gates. It is the goal to generate test vectors and discover the coverage available of the circuit. After generating tests, we will run the generated test against provided circuits to find a faulty circuit. We will then proceed to diagnose the issue.

2. Result:

(a) Total number of tests

1557 Test Vectors found out of 1566 possible

(b) Fault Coverage

1557/1566 = 99.325%

(c) List of undetected/undetectable faults

There are nine total undetectable Faults. Below is a list of faults found to be undetectable.

Fault: 1485/1193/1

Fault: 329/118/1

Fault: 1117/14/1

Fault: 1117/123/1

Fault: 1330/1

Fault: 274/1

Fault: 1465/872/1

Fault: 1465/872/1

Fault: 1120/1096/1

(d) The test result for four circuits

After simulating each circuit using the lsim simulator we forwarded the output simulation to the respective circuit name followed by *.sim.

Initial Circuit:	PART_I/circuit-1.sim
Circuit 1:	PART_I/v_chip1-1.sim
Circuit 2:	PART_I/v_chip1-2.sim
Circuit 3:	PART_I/v_chip1-3.sim

After simulating each circuit against the generated test vectors we ran a diff comparison of the simulation outputs of the known fault free circuit (the initial circuit) against each test circuit. The first two circuits were found to have no differences and we accepted as fault free, however the third test circuit had a grand total of

We recorded the comparisons of the original circuit with the test circuits in three files.

Circuit 1:	PART_I/v_chip1-1.err
Circuit 2:	PART_I/v_chip1-2.err
Circuit 3:	PART_I/v_chip1-3.err

(e) The diagnosis result on faulty circuit with lowest index

By analyzing the differences between the expected output of the original circuit to the test circuit we deduced that circuit three was the only circuit with faults present. We then went on to diagnose the fault that was causing the error.

In our case we found that vector 1195 caused Primary Output 77 to flip from a 1 to a 0 (D). This test vector showed that 497/522/1 is present when the output 77 went from 1 to 0. The fault must exist on the fanout from AND gate 522 to AND gate 497, as this is the fault that is found.

(f) Every 15th test vector from the set of first 100 vectors

1st

Fault: 1775/0

PO: 1612/D

Test: 1775/1 28/1 37/1 296/1

15th

Fault: 1253/0

PO: 1107/D

Test: 1253/1 37/1 296/1

30th

Fault: 858/1

PO: 1249/U

Test: 858/0 28/1 37/1 296/1

45th

Fault: 515/0

PO: 1859/D

Test: 515/1 1259/1 1765/1 519/1 858/1 254/1 412/0 686/1 1253/0 184/1 381/1

Cont: 761/1 1473/1 798/1 1408/0 1196/0 1271/0 1766/1 296/1 982/1 1775/1 714/0

Cont: 841/1 439/1 811/1 37/1 135/1 952/0 928/1 187/0 1096/0

60th

Fault: 184/1

PO: 77/U

Test: 184/0 37/1 296/1

75th

Fault: 1418/0

PO: 1107/D

Test: 1253/1 37/1 296/1

90th

Fault: 1139/1

PO: 1859/D

Test: 928/1 1775/1 187/0 858/1 519/1 1259/1 841/1 184/1 1253/1 798/0 1196/0

Cont: 982/1 1408/1 1473/0 1765/0 714/1 1271/1 686/1 439/1 412/0 811/1 37/1

Cont: 254/0 381/1 761/1 135/1 1766/1 296/0 515/0 952/0 1096/1

105th

Fault: 689/0

PO: 753/D

Test: 1408/0 811/0 135/1 1259/1 1765/1 519/1 858/1 254/1 412/0 686/1 1253/0

Cont: 184/1 381/1 761/1 1473/1 798/1

3. Test Generation:

(a) The originally planned strategy

We plan to generate our tests using faultgen and podem. The result would then be a vector file that we could use to diagnose circuits.

(b) The actual strategy used

We were able to use the planned strategy. We, however, did need to extend podem's backtracking

(c) The details of the test generation process

The tests were generated using faultgen and podem together. The following commands were run.

```
PART_I $ faultgen circuit-1 circuit-1-faults
PART_I $ podem -n circuit-1 -f circuit-1-faults -o vectors.v -l 100000000
```

circuit-1: circuit-1 netlist

circuit-1-faults: output of faultgen and input to podem

vectors.v: output of podem showing test vectors

The Results of the commands were as follows:

Total Faults	1566
Undetectable faults found	9
Aborted faults	0
Total Time	831 sec.
Time/Fault	0.530885 sec.
Backtracks	9.35669e+07
Implications	1.87162e+08

By using a high number for iteration on PODEM, we were able to have no aborted faults and have 9 undetectable faults. Our resulting coverage percentage is 99.425%.

(d) Discussion

Using faultgen, we found most faults and were ultimately able to have no aborted faults. We are very satisfied with the results of our test generation process.

4. Fault Diagnosis: (no more than 2 pages)

(a) The originally planned strategy

For diagnosis we planned on simulating each simply comparing the output of the fault free/original circuit with that of the unknown circuits.

(b) The actual strategy used

As originally planned we simulated each circuit with the test vectors created from the original circuit and using a diff command we found all the outputs that did not match what was expected. Once we knew which circuit was faulty (the third one in our case) we went into more detailed analysis of which fault vector stimulated and observed the exact fault causing the error.

(c) The details of the diagnosis process

By analyzing the differences between the expected output of the original circuit to the test circuit we deduced that circuit three was the only circuit with faults present.

Because the files we so large we were forced to use a grep to find all the faults and a diff to find all the places where the original circuit simulation did not match up with the three unknown circuits. If no differences were found then the circuit was assumed to be fault free. In this manner we found the only circuit to contain errors was the third and final circuit.

Be looking at which test vector had varying outputs we were able to deduce which fault was responsible for the error. When looking at each test vector we had to verify that the test saw the expected error case, if not then the designated fault for that test was not present. Only when the fault vector produced the EXPECTED output change was the specific fault detected.

```
tux-74:PART_I$ lsim -n circuit-1 -v vectors.v > circuit-1.sim
imx-74:PART_I$ tux-74:PART_I$ lsim -n v_chip1-1 -v vectors.v > v_chip1-1.sim
imx-74:PART_I$ tux-74:PART_I$ lsim -n v_chip1-2 -v vectors.v > v_chip1-2.sim
imx-74:PART_I$ tux-74:PART_I$ lsim -n v_chip1-3 -v vectors.v > v_chip1-3.sim

tux-74:PART_I$ diff circuit-1.sim v_chip1-1.sim > v_chip1-1.err
tux-74:PART_I$ diff circuit-1.sim v_chip1-2.sim > v_chip1-2.err
tux-74:PART_I$ diff circuit-1.sim v_chip1-3.sim > v_chip1-3.err
```

(d) Discussion and Comments

After simulating the test vectors for circuit we compared the expected output with the actual output. In the case where the outputs did not match we found the test vector that had the correct error on the correct output pin. By doing this we found the actual fault that was causing the error in the design. While it was not necessary for our this example it might be useful to have two separate sets of test vectors: one set that lets us know when an error is present and a second more in depth set of tests that we can use to determine which fault is causing the erroneous data.

Part II: Partial scan

1. Problem Statement:

Given a sequential circuit, select six Latches to be converted over to Serial Scan Latches that provide us with the best fault coverage. We will attempt to find the best pattern through random sampling of the circuit latches, and will present the result

2. Result: (no more than 3 pages)

(a) Statistics for unmodified circuit

We ran this command from PART_I/unmodified.

```
fastest -c seq_1 -f seq_1-faults -o seq_1-output -v seq_1-vector -h seq_1-statistics
```

PART_I/unmodified/seq_1-faults: result of faultgen seq_1 seq_1-faults
PART_I/unmodified/seq_1-output: the output of fastest
PART_I/unmodified/seq_1-vector: the output vectors of fastest
PART_I/unmodified/seq_1-statistics: statistics of fastest

Result:
Faults: 1073; Detected: 0; Coverage: 0.000000
Total Vectors: 0

(b) Six flip-flops that are included in feedback cycle(s)

The Six flip-flops that are included in feedback cycles were found using a program enclosed as PART_II/findFeedbackCycles. The Program goes through the circuit and finds any loops that involve latches. The pseudo code for the program is as follows:

```
foreach latch
    getCycleOfNodes(latch, latch)

getCycleOfNodes(currNode, endNode)
```

```
push currentNode onto the stack
if the end of the stack is our end node
return the current stack
        else if the last node is on the stack already,
//we have caused a cycle, but not including the latch
return null
        else
                //we need to look at all children, and try to get
//the cycles from there
                foreach child
                        //find the child node in our list of nodes
                        childStack = getCycleOfNodes(childNode, endNode)
                        if return is non-null return childStack
```

The program found six flip-flops with feedback loops:

- 446
- 759
- 993
- 1042
- 1086
- 1126

If you wish to see the sequences that result in loops, you can look at the file PART_II/findFeedbackCycles/ProgramOutputLatchLoops.txt. We did not attempt to find the shortest sequence that results in a loop. We just found a loop existed. For this reason, some sequences are quite long. This only makes the output less than spectacular. It still results in a program that accurately finds all flip flops that have feedback loops.

(c) Strategies for selecting scan flip-flops

With 29 latches to select from, with only 6 being selected, we are faced with 475,020 different options. Our strategy to select the scan flip-flops was to randomly iterate over the circuits with 6 scanable flip flops. After many iterations, we would find the best possible result. The sequences were generated randomly using a small java program and results were stored to avoid duplicate work. Pseudo code for the program is as follows:

```
prevList = empty list; //Create a list to store of previous results
    forever
    produce a random 6 latches to make scanable.
    //check if the that particular set of scanable latches has been tested
    if sequence not in prevList  // it hasn't run fastest
        run fastest
        if the result of fastest is better than the current best,
mark this as the best file
            end forever
```

(d) Selected set of flip-flops

After running our script for an extended time, we had a good sample of data. We checked 324,452 different possibilities, and believe we have found a very good result.

The circuit we were given contains 29 flip flops. In order to represent them below, we have specified '0' for non scanable, and '1' for scanable.

Pattern: 00001000100010000000100010100
File Location: PART_II/scan.n

(e) Statistics for partially scanned circuit

To get the statistics of the partially scanned circuit, we ran the following command from PART_II/partialScan:

```
fastest -c seq_1-00001000100010000000100010100 -f seq_1-faults -o seq_1-output -v seq_1-vector -h seq_1-statistics
```

PART_II/partialScan/seq_1-00001000100010000000100010100: the best found circuit with 6 scanable flip flops
PART_II/partialScan/seq_1-faults: list of all faults and given by fastest -f flag
PART_II/partialScan/seq_1-output: the output file as given by fastest -o flag
PART_II/partialScan/seq_1-vector: the vector file produced by fastest -v flag
PART_II/partialScan/seq_1-statistics: the statistics file made by fastest -h flag

The results of the command are as follows:

Faults: 1073; Detected: 978; Coverage: 0.911463

Total Vectors: 425

(f) Discussion and Comments

We believe with more time we could have attempted all possible six scanable latch combinations. Fortunately, we were able to get a fairly high percentage, and are satisfied with the result. We considered finding the ones that were hardest to detect, but we firmly believe the random selection had a better chance to find a best set than me attempting each by hand. This allows a much better sample than a few hand tested/found results.