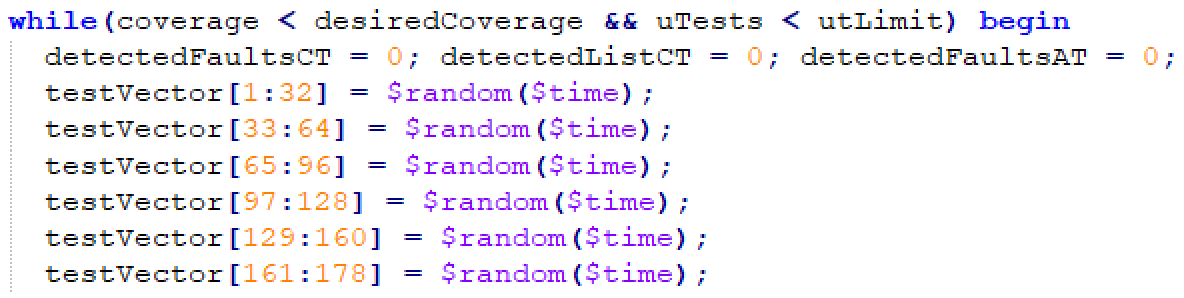
Evan Apinis
ECE 5724 Homework 3
2/25/24

1. Introduction

In this assignment, we were tasked with writing a Verilog testbench that can complete test generation. The testbench is written once again using the c5315 benchmark netlist that was provided to us in the previous assignment. The purpose of this assignment is to avoid using software tools such as Atalanta for creating test patterns. Instead, we will use the Verilog testbench to create these test patterns and estimate the fault coverage by using the collapsed c5315 fault list.

2. Test Bench Design

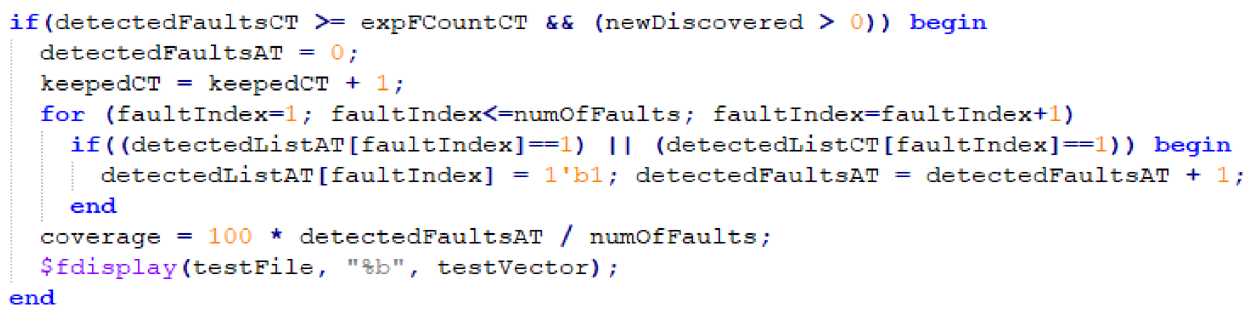
The testbench design followed the week five lectures on test pattern generation methods and algorithms. This meant implementing the adjustable expected coverage method and selecting its parameter values to achieve the best results possible. The first portion of the testbench performs fault collapsing on the fault list for the c5315 benchmark to inject and remove faults later. Once this has been completed the while loop containing the test generation is entered. The requirements for this loop are the achieved coverage being lower than the target and the number of random vectors generated is less than the maximum allowed. If this statement is true, the testbench will create a random test vector using the $random and $time system tasks. These tasks will only return a 32-bit value but the input to the c5315 netlist is 178-bits. For this, multiple statements are used to address sub-portions of the input at a time. The screenshot below shows a sample of this vector generation.



```
while(coverage < desiredCoverage && uTests < utLimit) begin
  detectedFaultsCT = 0; detectedListCT = 0; detectedFaultsAT = 0;
  testVector[1:32] = $random($time);
  testVector[33:64] = $random($time);
  testVector[65:96] = $random($time);
  testVector[97:128] = $random($time);
  testVector[129:160] = $random($time);
  testVector[161:178] = $random($time);
```

Figure 1 – Test Vector Generation Using $Random and $Time System Tasks

The next step is to apply this test vector to the fault list and evaluate the output of the FUT and GUT, comparing results and determining if a fault is observed. There is a fault index pointer to indicate which fault list input caused the observed fault. This is used to determine the number of new faults rather than generating test vectors that only cause duplicate faults. Then after this fault testing is completed the fault is removed from the FUT. Next, some operations are performed regarding the fault count to mark the detected faults of the previous test and determine the current fault coverage. This is used to determine if new additional test patterns are required. Figure 2 demonstrates a snippet of this code implementation.



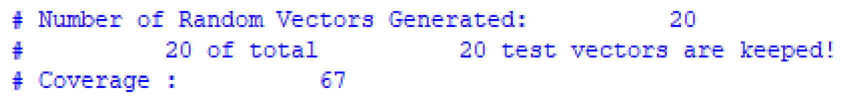
```
if(detectedFaultsCT >= expFCountCT && (newDiscovered > 0)) begin
  detectedFaultsAT = 0;
  keepedCT = keepedCT + 1;
  for (faultIndex=1; faultIndex<=numOfFaults; faultIndex=faultIndex+1)
    if((detectedListAT[faultIndex]==1) || (detectedListCT[faultIndex]==1)) begin
      detectedListAT[faultIndex] = 1'b1; detectedFaultsAT = detectedFaultsAT + 1;
    end
  coverage = 100 * detectedFaultsAT / numOfFaults;
  $fdisplay(testFile, "%b", testVector);
end
```

Figure 2 – Code Snippet of Fault Coverage Determination

The final aspect of this testbench is the parameters that initialize the adjustable expected coverage per test RTG. There are four parameters, the first is the number of expected faults and it is static, being used to determine the fault coverage. The second is the initialExpFCount and this is used for the expected faults, the initial value was set to 2 because this will be adjusted in the testbench. The third parameter was the utLimit and this is the maximum number of test patterns to generate. This will have a large impact on the achieved coverage and several values were used to demonstrate its impact in the results section. The final parameter was the desired coverage. This was set to 99 to ensure that I could achieve the highest coverage possible based on the test limit set in the third parameter.

3. Coverage Results

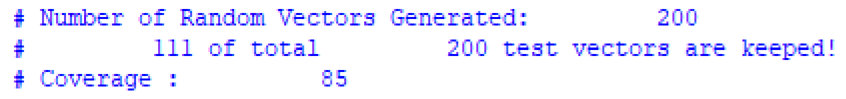
For the results section of this testbench, the first test was run using a test limit of 20, as used in several of the testbench examples provided. The results obtained are shown in Figure 3. Here it is seen that 20 of the total 20 test vectors were kept and a fault coverage of 67% was achieved. This is to be expected because of the few number of tests created, there are likely to be tests that discover unique faults rather than duplicates.



```
# Number of Random Vectors Generated:          20
#          20 of total          20 test vectors are keeped!
# Coverage :           67
```

Figure 3 – Coverage Results of 20 Test Vectors

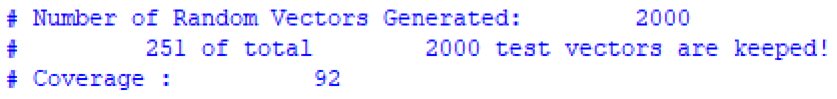
In the second run of this testbench, a value of 200 was used for the test limit. In Figure 4 it is shown that the ten times increase in test vectors created resulted in a coverage of 85%. This was an 18% increase from the previous run. The number of test vectors kept was 111 after removing vectors that didn't produce any new faults.



```
# Number of Random Vectors Generated:         200
#         111 of total         200 test vectors are keeped!
# Coverage :           85
```

Figure 4 – Coverage Results of 200 Test Vectors

In Figure 5 the results of the testbench run with 2000 test vectors are shown. In this run a coverage of 92% was achieved. This was only a 7% increase from the previous run and it's seen that the coverage gained from increasing test vectors has slowed down. This was an expected result as the more tests are created the more exhaustive it becomes. This is particularly evident with the number of kept vectors not increasing proportionally with the number of test vectors generated. As more tests are created fewer new faults are discovered each time. The last part of the testbench is to write the test vectors out into a .tst file so that they can be used later. A final test was run with 10,000 random vectors and it created 265 test vectors for a total coverage of 93%. In the submitted files the .tst output file contains the 265 generated test vectors.



```
# Number of Random Vectors Generated:        2000
#         251 of total        2000 test vectors are keeped!
# Coverage :          92
```

Figure 5 – Coverage Results of 2000 Test Vectors