Project 3

PODEM Test Generator

Georgia Institute of Technology

ECE 6140

Martin, Michael

902912673

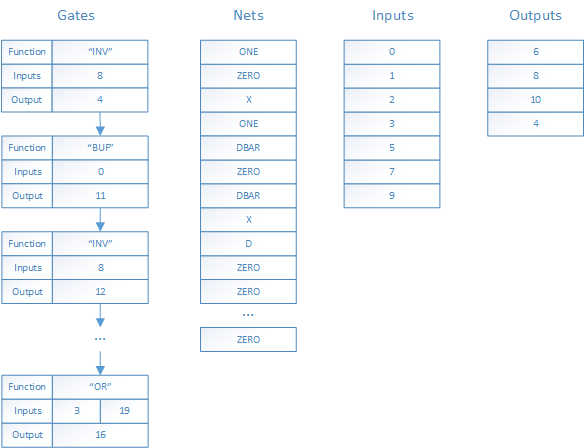
December 6, 2017

**Data Structures**

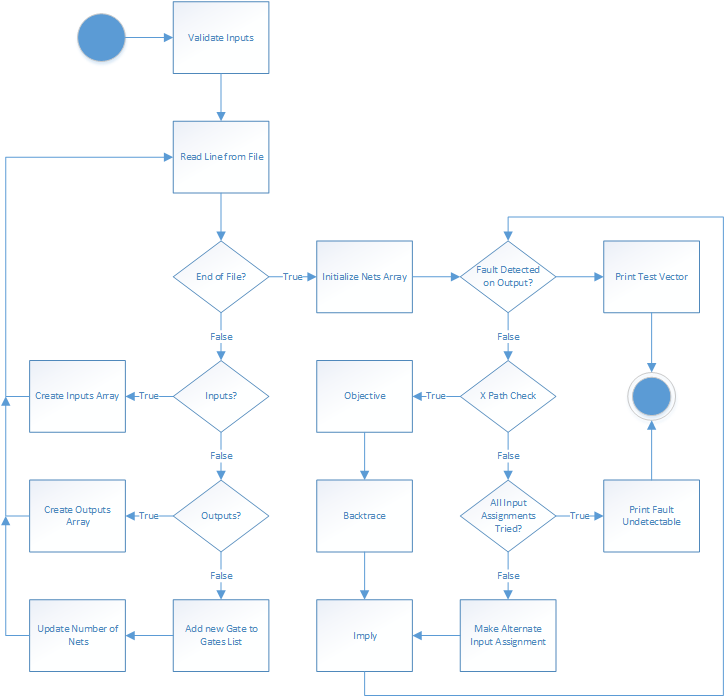
The simulation defines one class, Gate, and one enumerated type, Logic. The Gate class has three data members, a String to store the gate function (e.g. “INV”, “BUF”, etc.), an integer array to store the input nets, and an integer to store the output net. The Logic enumerated type has five possible values: ZERO, ONE, X, D, and DBAR. Instances of the Gate class and Logic type are stored in three arrays and two linked lists. The “nets” array stores an instance of the Logic type at each index corresponding to the net number. The “inputs” array stores the index to the “nets” array of each net that is an input to the circuit. The “outputs” array stores the index to the “nets” array of each net that is an output to the circuit. The “gates” linked list stores instances of the Gate class corresponding to each gate in the circuit. Finally, the “dFrontier” linked list stores instances of the Gate class corresponding to each gate that is on the D-frontier at a given step during test generation. Figure 1 shows a snapshot of these data structures.

**Simulation Flow**

The simulation has two main sections. The first section parses the input file line by line and creates a model of the circuit. The second section executes a recursive branch and bound algorithm called PODEM to generate a test vector to detect a given fault. In the first section of the program, a new instance of the Gate class is created and added to the end of the “gates” linked list for each line in parsed in the file. Next, the “nets” array is created and each net is initialized with a value of X to indicate that the value is unknown. The second section of the program first checks if the fault is detectable on an output of the circuit. Then the algorithm performs an X path check to determine if there is a possible path from the D-frontier to an output of the circuit. Next, the algorithm chooses an objective and finds an X path from that objective to a primary input. The algorithm sets the value of the primary input and runs a forward simulation. The recursive function is called again, and the program checks if the fault is detectable on an output of the circuit. Figure 2 shows the flow of the test generator.



**Figure 1.** Snapshot of data structures after running the test generator on circuit s27 for fault 16 s-a-0.



**Figure 2.** Algorithmic State Machine that describes the flow of the test generator.

**Table 1.** Test Vectors Generated for Faults in Four Test Circuits

|  |  |  |
| --- | --- | --- |
| Circuit | Fault | Test Vector |
| s27 | 16 s-a-0 | 10X10X0 |
| 10 s-a-1 | X00XXX0 |
| 12 s-a-0 | 1XXX1XX |
| 18 s-a-1 | 11X101X |
| s298f\_2 | 70 s-a-1 | 01X1XXXXXXXXXX0XX |
| 73 s-a-0 | 111XXXXXXXXXXX0XX |
| 26 s-a-1 | XX1X1XXX0XXXXXXXX |
| 92 s-a-0 | X10101XXXXXX0X0XX |
| s344f\_2 | 166 s-a-0 | 01X00XXXXX011XX0XXXXXXXX |
| 71 s-a-1 | 10XXXXXXXXXXXXXXXXXXXXXX |
| 16 s-a-0 | 10XXXXXXXXXXXXX1XXXXXXXX |
| 91 s-a-1 | 111XXXXXXXXXXXX0XXXXXXXX |
| s349f\_2 | 25 s-a-1 | XXXXXXXXXXXXXXX1XXXXXXXX |
| 51 s-a-0 | 00XXXXXXXXXXXXX0XXXXXXXX |
| 105 s-a-1 | 01X1000XXX01XX10XXXXXXXX |
| 7 s-a-0 | XXXXXX1XXXXXXXXXXXXXXXXX |