

Report: PODEM

Vaishnavi Sanjeev

GTID: 903797718

1 PROGRAM FLOW DESCRIPTION

The Python program implements the Path-Oriented Decision Making (PODEM) algorithm for automatic test pattern generation in digital circuits.

1.1 Functions Overview

Description of PODEM.py Code

1. `Read_netlist(file_name)`: Parses the netlist file to extract gate, input, and output information by scanning each line, categorizing gate types, inputs, and outputs.
2. `create_gate_list(gate_type)`: Generates gate lists based on gate types by reading the file, isolating specific gates, and extracting their inputs and outputs.
3. `create_fault_list(fault_list)`: Reads a fault list file, extracts fault details, and creates a fault list containing details about each fault.
4. `find_gate(pin)`: Identifies the gate associated with the target pin which has the stuck-at fault by scanning gate lists to determine which gate's output connects to the pin.
5. `create_D_front()`: Locates gates that have an 'x' output while having 'D' or 'Dbar' inputs, populating a list of gates known as the D-frontier.
6. `objective(target_pin, stuck_at)`: Implements the PODEM algorithm's objective function, selecting a gate and a value to propagate towards the target fault.
7. `backtrace(target_pin, pin_val)`: Traces back from the target pin to identify a path and determine a new value to be set for a particular pin.
8. `eval_logic()`: Evaluates gate logic based on the given inputs, performing multiple iterations to simulate gate behavior and determine pin values.
9. `PODEM(target_pin1, stuck_at1)`: The main PODEM algorithm function that generates test patterns by iterating through the fault list, backtracing, setting objectives, and performing forward implication to identify test vectors.
10. `output_write(all_output)` and `input_write(all_input)`: Functions responsible for recording results after PODEM execution.

Description of PODEM_Logic_Gates.py Code:

The code consists of functions representing various logic gates used in digital circuit design. Each function takes input signals and computes the output based on the logic of the corresponding gate.

- `not_gate(inp1)`: Generates the output of a NOT gate. It inverses the input signal (`inp1`).
- `buf_gate(inp1)`: Represents a buffer gate. It simply outputs the input signal (`inp1`).
- `and_gate(inp1, inp2)`: Implements an AND gate. The output is based on logical AND operation between `inp1` and `inp2`.
- `or_gate(inp1, inp2)`: Represents an OR gate. It computes the logical OR operation between `inp1` and `inp2` to generate the output.
- `nor_gate(inp1, inp2)`: Computes the output of a NOR gate by using the OR gate function and then inverting the result using the NOT gate.
- `nand_gate(inp1, inp2)`: Calculates the output of a NAND gate by using the AND gate function and then inverting the result using the NOT gate.
- `xor_gate(inp1, inp2)`: Determines the output of an XOR gate by checking whether `inp1` and `inp2` are equal. If they are equal, the output is 0; otherwise, it is 1.

These functions collectively represent the basic logic gates used in digital electronics to perform logical operations on binary signals.

1.2 Program Execution Overview

1. The program initializes by parsing netlist and fault-list files, extracting relevant gate and fault information.
2. Functions like `create_gate_list`, `create_fault_list`, and `Read_netlist` interpret the circuit's structure, gate types, and fault scenarios.
3. Core PODEM steps involve backtracing, setting objectives, evaluating forward implications (`eval_logic`), and continuously updating the D-frontier for test pattern generation.
4. Results, encompassing PODEM outcomes, generated test stimuli, and test vectors, are saved to output files for subsequent analysis and further circuit testing.

The output test vectors generated by the PODEM code can be fed to the Deductive Fault Simulator developed in Project 2 to verify if the fault is indeed detected.

2 PODEM SIMULATION RESULTS

Circuit	Net Stuck-At-Value	PODEM Output
S27	Net 16 s-a-0	00XX011
	Net 10 s-a-1	1000XX0
	Net 12 s-a-0	1XXX1XX
	Net 18 s-a-1	10X1011
	Net 17 s-a-1	10X00X0
	Net 13 s-a-0	1XXX1XX
	Net 6 s-a-1	X0X10X0
	Net 11 s-a-0	X00XXX1
S298_2	Net 70 s-a-1	01X1XXXXXXXXXX0XX
	Net 73 s-a-0	111XXXXXXXXXXXX0XX
	Net 26 s-a-1	XX110XXXXXXXXXXXX
	Net 92 s-a-0	X10101XXXXXX0X0XX
	Net 38 s-a-0	10X0XXXXXXXXXX0XX
	Net 46 s-a-1	X11XX0XXXXXXXXXX0XX
	Net 3 s-a-1	X10010X0XXXXXXXXXX
	Net 68 s-a-0	10001XXXXXXXXX00XX
S344f_2	Net 166 s-a-0	11X0100XXX011010XXXXXXXX
	Net 71 s-a-1	10XXXXXXXXXXXXXXXXXXXXXX
	Net 16 s-a-0	1101110XXX1XXXX1XXXXXXXX
	Net 91 s-a-1	111XXXXXXXXXXXXXXXXXXXXXX
	Net 38 s-a-0	X1XXXXXXXXXX1XXXXXXXXXX
	Net 5 s-a-1	XXX0XXXXXXXXXXXXXXXXXXXX
	Net 138 s-a-0	001X0XXXXX1XXXX0XXXXXXXX
	Net 91 s-a-0	00XXXXXXXXXXXX0XXXXXXXX
S349f_2	Net 25 s-a-1	XXXXXXXXXXXXXXXX1XXXXXXXX
	Net 51 s-a-0	010XXXXXXXXXXXX0XXXXXXXX
	Net 105 s-a-1	0011000XXX01XX10XXXXXXXX
	Net 105 s-a-0	0011XXXXX1XXXX0XXXXXXXX
	Net 83 s-a-1	11XX000XXX0X0X10XXXXXXXX
	Net 92 s-a-0	X1XXX1XXXXX1XXXXXXXXXX
	Net 7 s-a-0	XXXXXX1XXXXXXXXXXXXXXXX
	Net 179 s-a-0	101XXXXXXXXXXXX0XXXXXXXX

APPENDIX

User Manual: Fault Simulator and PODEM

3 USER MANUAL: DEDUCTIVE FAULT SIMULATOR

This codebase is written using Python3 in PyCharm installed in a Linux machine in VMware Workstation (Virtual Machine) and utilizes the following libraries as part of Python's standard library - random, argparse, re and sys. Additionally, the following package is required:

- `matplotlib.pyplot` : You can install it using the command `pip install matplotlib`.

Ensure these packages are installed in your Python environment to run the code successfully. A basic setup of the Python environment is sufficient to run this code.

3.1 README

Unzip the Zip File in Linux:

To unzip the zip file in a clean Linux workspace, follow these steps:

1. **Access the Terminal:** Open a terminal window in Linux
2. **Copy the Zip File to a Directory:** Use the `cp` command followed by the path to the directory where you want the zip file. For example:

```
cp final_deliverable.zip /path/to/your/directory
```

3. **Unzip the File:** Execute the `unzip` command followed by the name of the zip file to extract its contents:

```
unzip final_deliverable.zip
```

4. **View Extracted Files:** After executing the `unzip` command, the zip file's contents will be extracted into the same directory. You can confirm the extraction by listing the directory's contents using the `ls` command. The zipped folder must contain two sub-folders with the names `Deductive_FS` and `PODEM` containing the code for the respective topics. The main folder contains the various circuit files and their corresponding fault files used as part of the project.
5. **Cleanup (Optional):** If needed, remove the original zip file after extraction using the `rm` command:

```
rm final_deliverable.zip
```

```
(base) vnanjee@vnanjee-virtual-machine:~/PycharmProjects/pythonProject5$ ls
Deductive_FS  output_net_faults.txt  pycache  s27.txt  s298f_2.txt  s344f_2.txt  s349f_2.txt  test_file.txt  Test_Vectors_For_Ded_Fault_Sim.txt
main.py       PODEM                  s27_fault.txt  s298f_2_fault.txt  s344f_2_fault.txt  s349f_2_fault.txt  sorted_output_net_faults.txt  Test_stimulus.txt  venv
```

Figure 1—Files in Directory after Steps 4 and 5

Running the Deductive Fault Simulator:

Running the Deductive Fault Simulator program requires an input circuit file - <circuit>.txt. For explanation, the s344f_2 circuit is used here. The image shown below is that of the s344f_2.txt file.

```
s344f_2.txt
143  INV 10 149
144  INV 6 151
145  INV 94 153
146  INV 103 154
147  INV 6 156
148  BUF 185 157
149  INV 5 159
150  INV 117 161
151  INV 5 163
152  BUF 186 164
153  INV 120 165
154  INV 4 167
155  INV 123 169
156  INV 4 171
157  BUF 187 172
158  INV 126 173
159  INV 131 174
160  INV 134 175
161  BUF 1 177
162  INV 91 178
163  INV 1 182
164  BUF 91 183
165  BUF 2 189
166  INV 3 190
167  INPUT  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 -1
168  OUTPUT 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 -1
```

Figure 2—Format of the s344f_2.txt

The s344f_2.txt lists all the logic gates in the digital circuit and their corresponding input and output nets. For example: INV 6 151 implies that an inverter gate exists in the circuit with net 6 as input and net 151 as output. The last 2 lines in the file (INPUT and OUTPUT as seen in the image below) consolidates the primary inputs and primary outputs for the entire digital circuit.

In the path where you unzipped the zip file, run the following command.

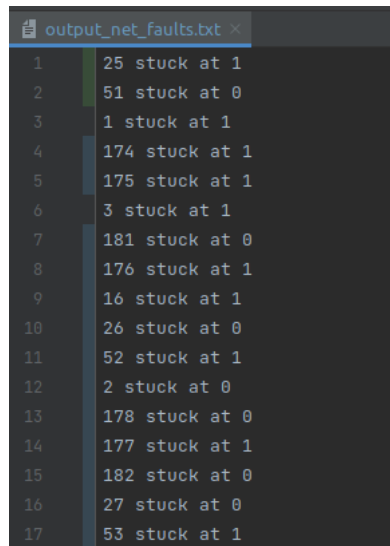
```
python Deductive_FS/logic_simulator.py --filename <circuit>.txt -
-test_vec <test_vector_bits>
```

For example:

```
python Deductive_FS/logic_simulator.py --filename s298f_2.txt -  
-test_vec 010100000000000000
```

Two output files are generated after running this command.

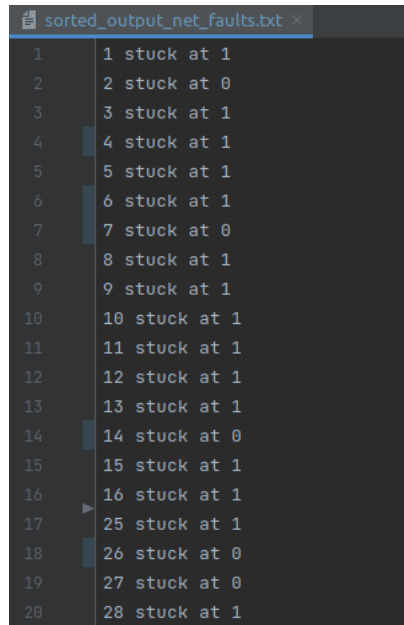
1. **output_net_faults.txt** : Shows a list of all possible detectable faults in the circuit for the test vector passed via the command line.



Line	Net	Stuck-at
1	25	1
2	51	0
3	1	1
4	174	1
5	175	1
6	3	1
7	181	0
8	176	1
9	16	1
10	26	0
11	52	1
12	2	0
13	178	0
14	177	1
15	182	0
16	27	0
17	53	1

Figure 3—Format of the output_net_faults.txt

2. **sorted_output_net_faults.txt** : Shows a list of all possible detectable faults in the circuit for the test vector passed via the command line, sorted in ascending order of the nets.



```

1 1 stuck at 1
2 2 stuck at 0
3 3 stuck at 1
4 4 stuck at 1
5 5 stuck at 1
6 6 stuck at 1
7 7 stuck at 0
8 8 stuck at 1
9 9 stuck at 1
10 10 stuck at 1
11 11 stuck at 1
12 12 stuck at 1
13 13 stuck at 1
14 14 stuck at 0
15 15 stuck at 1
16 16 stuck at 1
17 25 stuck at 1
18 26 stuck at 0
19 27 stuck at 0
20 28 stuck at 1

```

Figure 4—Format of the sorted_output_net_faults.txt

4 USER MANUAL: PODEM

This codebase is written using Python 3 in PyCharm installed in a Linux machine in VMware Workstation (Virtual Machine) and utilises the following libraries as part of Python's standard library - random, argparse, re and sys. Ensure these packages are installed in your Python environment to run the code successfully. A basic setup of the Python environment is sufficient to run this code.

4.1 README

Unzip the zip file in a clean Linux workspace, following the steps enumerated in Section 3.1 under **Unzip the Zip File in Linux:**.

Running the PODEM Simulator:

Running the PODEM program requires two input files - <circuit>.txt and <circuit_fault>.txt. For explanation, I have used the s344f_2 circuit here.

The s344f_2.txt lists all the logic gates in the digital circuit and their corresponding input and output nets. For example: INV 6 151 implies that an inverter gate exists in the circuit with net 6 as input and net 151 as output. The last 2 lines in the file (INPUT and OUTPUT as seen in the image below) consolidates the primary inputs and primary outputs for the entire digital circuit.

```

143 INV 10 149
144 INV 6 151
145 INV 94 153
146 INV 103 154
147 INV 6 156
148 BUF 185 157
149 INV 5 159
150 INV 117 161
151 INV 5 163
152 BUF 186 164
153 INV 120 165
154 INV 4 167
155 INV 123 169
156 INV 4 171
157 BUF 187 172
158 INV 126 173
159 INV 131 174
160 INV 134 175
161 BUF 1 177
162 INV 91 178
163 INV 1 182
164 BUF 91 183
165 BUF 2 189
166 INV 3 190
167 INPUT 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 -1
168 OUTPUT 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 -1

```

Figure 5—Format of the s344f_2.txt

The s344f_2_fault.txt contains faults for which you want to find a test vector. For example: 166 0 (first line in the file) implies that net 16 in the s344f_2 circuit is stuck at 0. You can add or delete any of the lines in this file depending on which fault you want to find a test vector.

```

1 166 0
2 71 1
3 16 0
4 91 1
5 38 0
6 5 1
7 138 0
8 91 0

```

Figure 6—Format of the s344f_2_fault.txt

In the path where you unzipped the zip file, run the following command.

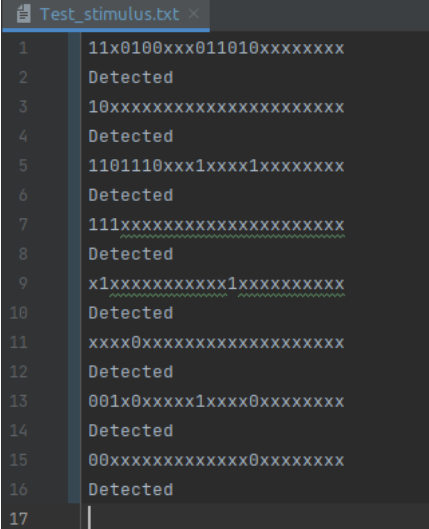
```
python PODEM/PODEM.py <circuit>.txt <circuit_fault>.txt
```

For example:

```
python PODEM/PODEM.py s344f_2.txt s344f_2_fault.txt
```

Two output files are generated after running this command.

1. **Test_stimulus.txt** : Contains PI net assignments that can reveal the fault. Only those PIs that can reveal the fault are set. The other PIs are denoted with an x. Whether each fault is detected or not is also specified for each fault.



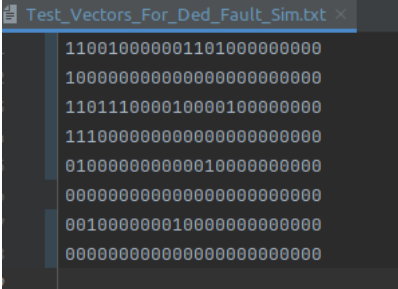
```

1 11x0100xxx011010xxxxxxx
2 Detected
3 10xxxxxxxxxxxxxxxxxxxx
4 Detected
5 1101110xxx1xxx1xxxxxxxx
6 Detected
7 111xxxxxxxxxxxxxxxxxxxx
8 Detected
9 x1xxxxxxxxx1xxxxxxxx
10 Detected
11 xxx0xxxxxxxxxxxxxxxx
12 Detected
13 001x0xxx1xxx0xxxx
14 Detected
15 00xxxxxxxx0xxxx
16 Detected
17

```

Figure 7—Snapshot of the file for the s344f_2 circuit

2. **Test_Vectors_For_Ded_Fault_Sim.txt** : Contains test vectors (all the x's in Test_stimulus.txt are replaced with a 0) that can be used in turn in the Deductive Fault Simulator to cross-verify if the fault is detected by the test vector.



```

1 110010000001101000000000
2 100000000000000000000000
3 110111000010000100000000
4 111000000000000000000000
5 010000000000100000000000
6 000000000000000000000000
7 001000000010000000000000
8 
9 
10

```

Figure 8—Snapshot of the file for the s344f_2 circuit