Project 3: a study of PRPG pattern resistant faults

JIM PALOMO

In a circuit, faults may occur which results in incorrect output. It is not always certain that a fault may be found given a set of input; therefore, a simulation of test vectors is applied to attempt to propagate the fault at the output. The study of Pseudo-Random Pattern Generation (PRPG) pattern resistant faults is to capture and determine faults that are difficult to detect using PRPG. Through the use of simulating different PRPG patterns using a Linear-Feedback Shift Register (LFSR) based design, it is shown that faults that are hard to detect are often found at different levels of a circuit. The work that follows will show how the level of the circuit determines the difficulty of the fault being detected.

1 INTRODUCTION

The study of PRPG pattern-resistant faults was conducted through the use of replicating fault simulation using an LFSR. The data collection process was done in Python. Fault simulation of all faults with different LFSR designs (certain taps on) on circuit bench file c432.bench were done and data obtained is outputted as a CSV file. The CSV file was then imported to Google Sheets and graphed. At first, I was planning to use the module Matplotlib, a library to generate static or interactive graphs. However, due to the restrictions provided on Replit, an online IDE to share and run code, Matplotlib is unable to run successfully for peer review. Thus, I instead manually graphed the data that I extracted.

In regards to the initial Python setup, I used an Object-Oriented Programming (OOP) technique known as Composition which references other classes for a program to function. I did not incorporate any Inheritance techniques for this project, techniques used to develop sub-classes from an existing class. Therefore, I developed several Python classes: NodeList, Node, Gate, and Fault. Each of these classes contained certain key components for data collection to happen. For instance, the Node class contained details on the node: name, position, gate, input, operation, value, level, wire, fault, SA-fault value, n0, n1, c0, and c1. The Fault class contained the name of the fault and the number of times it is detected at the output when several test vectors are ran. The Gate class provided logical functions for normal circuit and controllability simulations. Finally, the main class, NodeList used a combination of all the classes to generate data for every node and fault.

Additionally, the Python modules that I used are OS, Logging, Copy, and Pickle. The OS module is used to determine if an input file exists within a provided work space. Logging was used for debugging, similar to print statements but loaded into a file, and outputting final results. The Copy module was used to provide a deep copy method to prevent Python's auto pass-by-reference. In other words, if you copy an object X from Y (X=Y) in Python without a deep copy, the newly copied object X is still intertwined with the old object, Y. Therefore, if object X is modified in any way, the modification is reflected onto object Y. Finally, the Pickle module is used to save variables which can be opened after the program has finished executing. Note that OpenCV was also implemented but replit does not support that so it was removed. A different UI from one in this report that uses OpenCV can be accessed in the replit link to the python code under the folder "codeWithOpenCV."

Furthermore, in my experiment, there are a few terms worth defining. First of all, the program that I developed uses the format of a good/bad circuit (g/b). The format refers to the output of a good, expected value of a non-faulty circuit, and bad, outcome a faulty circuit, at a given node or wire. Therefore, the idea of a fault being detected means that an undesired value at a node or wire is propagated to the output. Based on the g/b format, this would refer to a 1/0 (D) or 0/1 (D') at an output node.

Moreover, the Level metric is used to determine the depth of each node within a circuit. Starting at the input, the level is 0 and as a value propagates deeper into the circuit, the level increases. In my report, the level is separated by the operations of a gate. For example, given a newly made circuit, if the operation is A=AND(b,c) and with inputs b & c, b & c will have a level of 0. The value of A-b & A-c which are wires that carry the value of input b & c is inputted to a AND gate. The output of AND(b,c) is A and A will have a level of 1. Therefore, wires A-b & A-c are at level 0.

The Controllability metric determines the difficulty of controlling a value of 0 or 1 at a node. For instance, input nodes always start with a control value of (c0,c1)=(1,1). A node further in the circuit will have a value greater than 1. Therefore, the smaller the value for (c0,c1) the easier that value is to control.

Extra metrics that are not used (n0,n1) refer to the number of times a 0 or 1 has occurred at the output. This metric is not used for this report but was implemented in my program.

2 DATA COLLECTION AND PRESENTATION

Data collection was done by running fault simulation on the circuit bench file c432.bench. The bench file is first loaded, sorted, and extracted into a NodeList. The NodeList contains the nodes which are provided from the bench file and in this report, I incorporated intermediate wires as nodes. Before the circuit is simulated, input nodes are placed into a knownList, a list of values that are known, and the rest of the nodes are inserted into an unknownList. When a circuit is simulated, the node in operation (e.g. A=NAND(b,c)) must have input values b & c before it can provide output A. My program views the inputs as wires so A-b & A-c must be present. Once the value of the node is available, it is then placed into the knownList. For a wire such as A-b, the value is provided from input b unless a fault occurs at the provided wire. A fault can be determined at a node or wire if the node properties contain a fault=True and sa-fault="value" where the value is 0 or 1. The iteration process is continued until unknownList is empty referring to all nodes and wires now having a value. A fault is determined to be detected in any of the output nodes that are considered to contain a D (1/0) or D' (0/1).

Moreover, before fully simulating a circuit, a full fault list is then generated so that simulation for all faults will be done and incorporated in faultList. The final setup pertains to different LFSR designs. The designs for each LFSR are differentiated by activating different taps. The different LFSR taps that are available are h=1, h=1,3,5, h=2,4,6, and h=6,7,8. The selection of taps was determined to fulfill different arrangements. For instance, h=1 provides one tap to catch as many faults as possible. On the other hand, I configured h=1,3,5 and h=2,3,5 as odd and even taps. Finally, I added tap h=6,7,8 which are consecutive taps. From the given LFSR tap designs, 10 test vectors are generated which each test vector is size 36. Do note that there are even more LFSR designs that can be implemented, the designs that I have chosen are not all of them. After this setup, all faults within the faultList, a list containing all faults, totaling 1064 faults, is then simulated with the selection of different LFSR tap designs. An insight of how much information is generated is seen by the following: 10 test vectors of size 36 (due to inputs), 1064 faults within the full fault list, 4 different LFSR tap designs. In other words, 10 test vectors * 4 LFSR tap designs * 1064 faults leading to a total of 42560 fault simulations or 40 simulations per fault.

2.1 Simulator Software UI and Example of program output

```
python3 main.py
The bench file used in this project is c432.bench

Testing bench/c432.bench...

Generating full fault list...

Simulating control...

Select LFSR tap configuration:
 a: h=1
 b: h=1,3,5
 c: h=2,4,6
 d: h=6,7,8
> c

Selected tap configuration: c

Simulating all faults in fault list...
```

Fault	No. detected	Level	c0	c1	n0	n1
1-0	6	0	1	1	0	10640
1-1	0	0	1	1	0	10640
4-0	5	0	1	1	0	10640
4-1	0	0	1	1	0	10640
8-0	0	0	1	1	10640	0
8-1	5	0	1	1	10640	0
11-0	6	0	1	1	9576	1064
11-1	16	0	1	1	9576	1064
14-0	0	0	1	1	1064	9576
14-1	4	0	1	1	1064	9576
17-0	34	0	1	1	1064	9576
17-1	1	0	1	1	1064	9576
21-0	0	0	1	1	9576	1064
21-1	10	0	1	1	9576	1064
24-0	6	0	1	1	8512	2128
24-1	5	0	1	1	8512	2128
27-0	4	0	1	1	7448	3192
27-1	7	0	1	1	7448	3192
30-0	5	0	1	1	6384	4256
30-1	0	0	1	1	6384	4256
34-0	1	0	1	1	5320	5320
34-1	0	0	1	1	5320	5320
37-0	6	0	1	1	4256	6384
37-1	0	0	1	1	4256	6384
40-0	1	0	1	1	3192	7448
40-1	0	0	1	1	3192	7448
43-0	0	0	1	1	2128	8512
43-1	0	0	1	1	2128	8512
47-0	0	0	1	1	1064	9576
47-1	0	0	1	1	1064	9576
50-0	5	0	1	1	1064	9576
50-1	0	0	1	1	1064	9576
53-0	0	0	1	1	0	10640

Fig. 1. Terminal output

```
|422-417-0  |        3        |     15     |   3   |   9   | 10455 |   0   |
|422-417-1  |        0        |     15     |   3   |   9   | 10455 |   0   |
|425-386-0  |        3        |     14     |   8   |   2   |  893  | 9441  |
|425-386-1  |        0        |     14     |   8   |   2   |  893  | 9441  |
|425-393-0  |        3        |     14     |   8   |   2   |   0   | 10475 |
|425-393-1  |        0        |     14     |   8   |   2   |   0   | 10475 |
|425-418-0  |        3        |     15     |   3   |   9   | 10551 |   0   |
|425-418-1  |        0        |     15     |   3   |   9   | 10551 |   0   |
|425-399-0  |        3        |     14     |   8   |   2   |   0   | 10549 |
|425-399-1  |        0        |     14     |   8   |   2   |   0   | 10549 |
|428-399-0  |        1        |     14     |   8   |   2   |   0   | 10549 |
|428-399-1  |        0        |     14     |   8   |   2   |   0   | 10549 |
|428-393-0  |        1        |     14     |   8   |   2   |   0   | 10475 |
|428-393-1  |        0        |     14     |   8   |   2   |   0   | 10475 |
|428-419-0  |        1        |     15     |   3   |   9   | 10551 |   0   |
|428-419-1  |        0        |     15     |   3   |   9   | 10551 |   0   |
|429-386-0  |        2        |     14     |   8   |   2   |  893  | 9441  |
|429-386-1  |        0        |     14     |   8   |   2   |  893  | 9441  |
|429-393-0  |        2        |     14     |   8   |   2   |   0   | 10475 |
|429-393-1  |        0        |     14     |   8   |   2   |   0   | 10475 |
|429-407-0  |        2        |     14     |   8   |   2   |   0   | 10571 |
|429-407-1  |        0        |     14     |   8   |   2   |   0   | 10571 |
|429-420-0  |        2        |     15     |   3   |   9   | 10551 |   0   |
|429-420-1  |        0        |     15     |   3   |   9   | 10551 |   0   |

1064 faults have been simulated
```

Fig. 2. Terminal output cont.

2.2 Main results

The main findings that were accumulated from the fault simulation data are average faults detected for circuit c432.bench with LFSR taps, the Number of Non-Detected Faults for circuit c432.bench with LFSR taps, and the Maximum Controllability value of c432.bench at given circuit levels. Levels will be divided into 3 sections: the beginning of the circuit (levels 0-4), the middle of the circuit (levels 5-11), and the end of the circuit (levels 12-17).

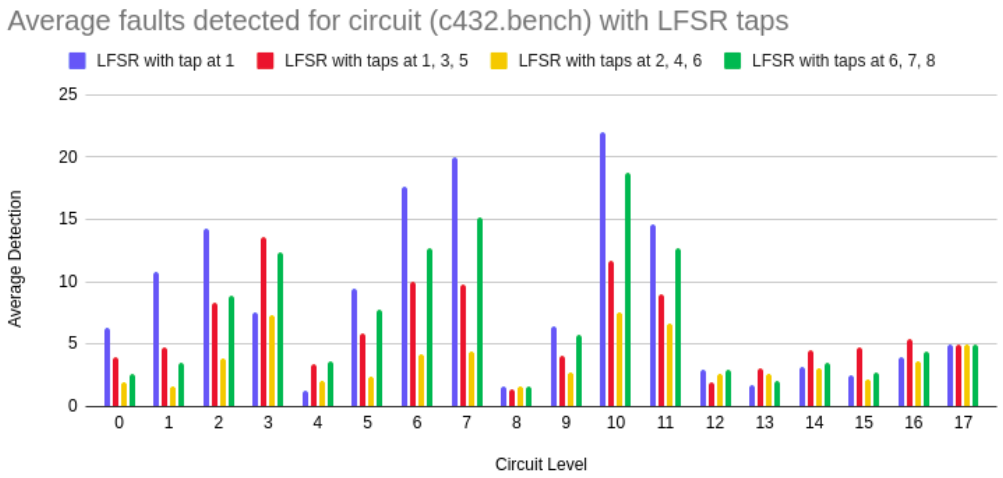

Fig. 3. Average number of faults detected given the level within the circuit (c432.bench) with different taps

First of all, I obtained the average amount of faults detected for circuit c432.bench at each level. I used various taps: h=1, h=1,3,5, h=2,4,6, and h=6,7,8. In Layman's terms, I found all the faults detected at each level and divided that by all the faults present. Furthermore, the circuit c432.bench had levels 18 total levels, 0 to 17. Based on the simulations, it is evident that it is harder to detect faults near the end of the circuit which can be viewed by levels 12 through 17 containing about less than 6 faults detected for each tap tested. On the other hand, it is easier to detect faults at certain points compared to input faults. Therefore, the order of ease of fault detection is regarded, from easiest to hardest, as the middle (minus the outlier at level 8) of the circuit, beginning of the circuit, and then the output of the circuit. Going further in-depth with the results that are obtained, each LFSR tap design produced different results. Having a tap activated at the beginning of the LFSR (h=1) and consecutive taps (h=6, 7, 8) produced higher fault detection ratings compared to the other taps (h=1, 3, 5 and h=2, 4, 6) which are seen by the blue and green bars. Moreover, the tap at h=2, 4, 6 which are even taps covered the least amount of faults altogether. An interesting note is based on the findings is that at levels 4 and 8, it was harder to detect a fault even though the surrounding levels had high fault detection ranges.

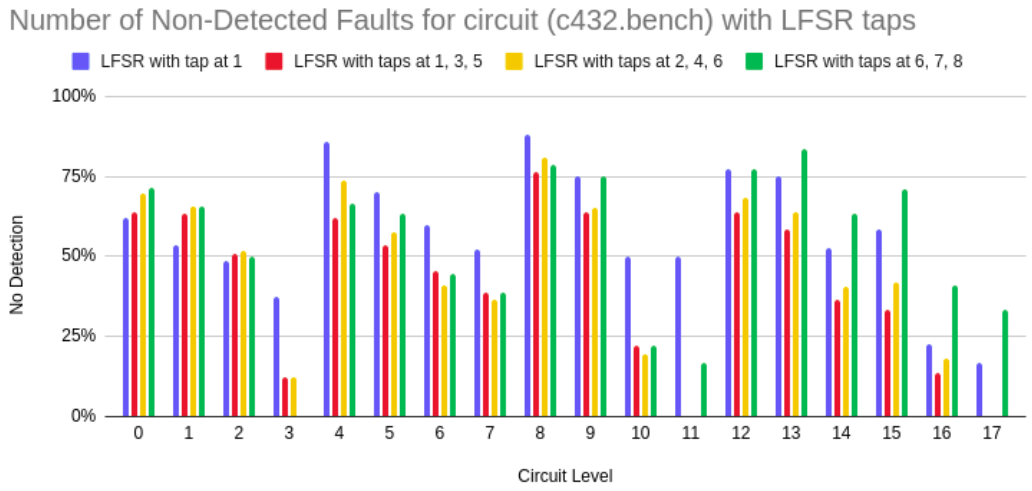

Fig. 4. Number of faults detected given the level within the circuit (c432.bench) with different taps

On the other hand, findings on the Number of Non-Detected Faults for each circuit level produced correlating and contradicting results with the Average faults detected at each level. For clarity, a non-detected fault is a fault that was not detected at output. An instance where values correlate is shown in locations near the output in the Number of Non-Detected Faults for circuit (c432.bench) graph. This visual matches with the fact that the average number of faults detected for the circuit is very minimal at the output on the average faults detected for the circuit. On the contrary, it is shown that there are a lot of non-detected faults in the middle levels of the circuit based even though the average for these faults at the given level is high (about 70%) compared to the other circuit levels (nearly 15 faults detected).

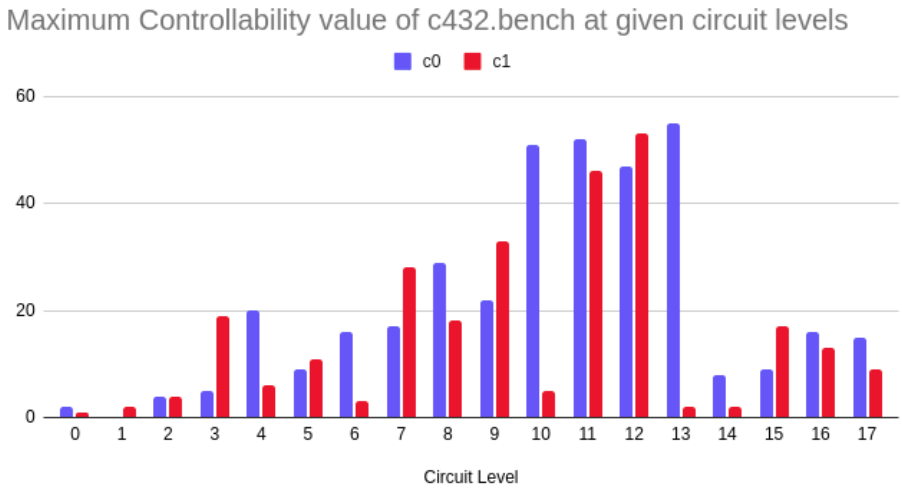

Fig. 5. Maximum Controllability value of bench file c432.bench at all given levels

The final graph portrays the maximum controllability values (c0,c1) of the circuit at each level. Note that a control value closer to 1 refers to the output being easier to control. According to the data provided, it is easiest to control values at input than that of the middle levels and near the output levels. At the input, the control values are roughly less than 5 while in the middle the control values reach up to levels of 40 or more.

3 DISCUSSION AND CONCLUSIONS

Provide your main conclusions here. Overall, it is seen that given the activated taps that I selected, it is easier to detect a fault near the beginning and middle of the circuit. However, in some cases such as levels 4 and 8 in Fig. 4, it is shown that detecting a fault at the beginning or middle may be harder than detecting a fault near the end of the circuit. In regards to the number of non-detected faults for c432.bench, the results show that there is no high detection rating for the taps that are used given the limited amount of test vectors ran for each fault (40 test vectors per fault). The no-fault detection rating lies among 50% and upwards to 75% with the selected taps. In terms of controllability, the results state that is easier to control values in the beginning, levels 0-4, and near the end with levels 14-17 than in the middle of the circuit. Obtaining the controllability of the circuit is to see whether or not SCOAP can be used as a metric to determine if a fault is easier to detect at different circuit levels. My circuit did not fully correlate with the SCOAP results but in Fig. 4 the input levels started to converge onto following the SCOAP metric. In the end, based on my data, SCOAP may be used to determine if controlling a value is easier to control (0 or 1); however, given the limited selected taps that I used and a minimum of 10 fault simulations per test vector, there was little to no correlation. Therefore, some suggested improvements will be stated.

My experimental approach did not fully lead to any definite conclusive statements but I was able to explain my findings with the selected LFSR design choices provided the data I used. However, further improvements to this experiment may be met. For instance, my LFSR was unable to obtain a lot of faults during the simulation. This is because I only simulated 10 test vectors per fault (1064 faults total) and each fault was simulated on 4 different taps; thus 40 simulations were conducted per fault. To increase the accuracy of experimental data in the future, it is suggested that I run more tests. Instead of running 10 test vectors per fault, I run up to 1000 which would lead to 4000 simulations per fault. This would lead to more faults being detected with the given taps. As expressed throughout the report and in this paragraph, testing is a very important process to determine how reliable something is. Increasing the test size would have led me to more accurate results that converge to a more defined solution, similar to limits in Calculus.

4 REFERENCE TO PYTHON, REPLIT

https://replit.com/@JimPalomo/P3-1