UIC Fall 2020 ECE 464 Project 3: Are LFSRs good for covering multi-faults?

Fardeen Hasan
Electrical and Computer Engineering (UIC)
University of Illinois
Chicago, U.S.A
Fhasan7@uic.edu

Sig Somsavath
Electrical and Computer Engineering (UIC)
University of Illinois
Chicago, U.S.A
Ssomsa2@uic.edu

I. Introduction

Linear-feedback shift register (LFSR)- is a shift register which has it's initial value set and is called as seed the next bit is a linear function of its previous state(taps), most commonly used linear function is exclusive-or(XOR).Any register should have a finite number of possible states and is periodic but with a seed which is uniquely chosen and varied taps LFSR can produce sequence of bits which are random and with a large period. Although LFSR was widely used in 1997 it had many advancements, for instance some of the XORs should be replaced with XNORs to avoid LFSR from producing all zero obfuscation Key in a locked up state and MISR a which is a type of LFSR. After introduction of MISR and some of it's advancements [1], Traditional LFSRs became obsolete but are necessary to understand all other BIST techniques which are types of LFSRs.

We see that LFSRs with varied taps perform better with a single stuck at fault, **But are LFSRs good for covering multi-faults ?**

II. Python simulator functionalities

A. LFSR

We use the LFSR from project 2 and simplicity in comparing we fix the seed for all the simulation as 0x123456789abcdef0 and add multi faults to the previous code, Since we ran into size issues we only have 2 faults as multi-faults and we have 3 faults for 2 smaller benches. The lfsr gives us the test vector seed to perform propagation's to find detection in faults

This is how we achieve the multi-faults:

```
def lfsr(seed, taps):
    a = []
    temp = []
    shiftRight = []
    temp = Convert(seed)
    a.append(seed)
    #print (temp[0])
    while shiftRight != seed:
    shiftRight = Convert(temp)
    for t in taps:
    if t > (len(temp)-1):
    continue
    else:
    shiftRight[t] = str(int(temp[t-1]) ^ int(temp[7]))
    for r in range(0,len(shiftRight)):
    if r in taps:
    continue
    elif r==0:
    (shiftRight[r]) = str(temp[7])
    else:
    (shiftRight[r]) = str(temp[r-1])
    temp = shiftRight
    shiftRightstr = listToString(shiftRight)
    a.append(shiftRightstr)
    if len(a) == 100:
    break
    return a
```

We only take 1000 faults with a combination of 2 faults and 40 faults with a combination of 3 faults,The test vector size is taken as 100; we have to choose these because the computation power of Python Sim did not let us go any higher, This code needs 64 bit Python to run, most Python Sims are 32bit. The reason we do this is because the circuits is really big and computational time would be very long

This is the code for 1000 faults with combination of 2 and 40 faults with combination of 3.

```
whichCombo = input(
  "Which one do you want to continue with?\n")

if whichCombo ==’1’:
 multi_list = []
 temp_list=[]
 temp_list.extend(combinations(node_list,2))
 multi_list.extend(random.sample(temp_list,250))
elif whichCombo ==’2’:
 multi_list = []
 temp_list=[]
 temp_list.extend(combinations(node_list,3))
multi_list.extend(random.sample(temp_list,5))
```

Since we need to test for different taps we keep the fault-list same once we test it for the first tap.To test it out first run the code and copy the Fault list that is generated for any tap you chose, then when you run the code for the next tap insert into [static-list] and run the code.

This is the variable where you need to add the Fault list

```
x = input("Use a static sample list(yes or no)")
if (x==’yes’):
    static_list = input("Enter a static Sample list")
    total_multipleFault = static_list
```

B. Combination of 2,3 Faults

To get the combinations of faults we use combinations function which is a readily available library.

```
import random
from itertools import combinations

temp_list.extend(combinations(node_list,2))
temp_list.extend(combinations(node_list,3))
```

C. Fault Coverage of a list of test vectors

```
def cal_multi_list():
  for i in multi_list:
    faultval = 2**len(i)
    temp=0
    while temp<faultval:
      z = []
      index= 0
      x=bin(temp)[2:].zfill(len(i))
      for item in i:
        if index > len(x)-1:
          break
        z.append(item.name+"-"+x[index])
        index = index + 1
      total_multipleFault.append(z)
      temp+=1
```

This function will create a list that contains multiple faults of the circuit. We use the LFSR given test vectors to propagate and detect and furthers faults

D. Testing

Some of the Testing parameters we chose

1) ***SEED****:* We use a fixed seed as `0x123456789abcdef0`, for all the taps and circuit benches, we do this so that we have a better comparison between different benches.

2) ***Faults****:* Since the enormous combinations of faults we only consider 1000 faults at random for a combination of 2 and 40 faults with a combination of 3 faults; To test out combination of 3 faults a 64 bit Python Sim is required.

3) ***Taps****:* We test on 4 taps configuration- no taps, taps at (2,4,5), taps at (2,3,4), taps at (3,5,7). Since the Faults are selected at random when we run it for the first tap configuration we copy the Fault list generated and use it again by adding it to a static variable explained above at the end of (A)

III. Data and Results

A. UI and Functionality

We keep the UI rather simple **Input:** your program should allow the user to specify:

1) Which circuit bench to run.
2) A SEED in hex.
3) The selection of Taps.

Output: Shows which tv is being applied to the code and how many faults are undetected before going over the next tv; at the end it gives the Detection rate.

B. Fault coverage comparison plot

We plot the testing for all the benches with 1000 faults with a combination of 2 faults with different taps which gives us 4 plots

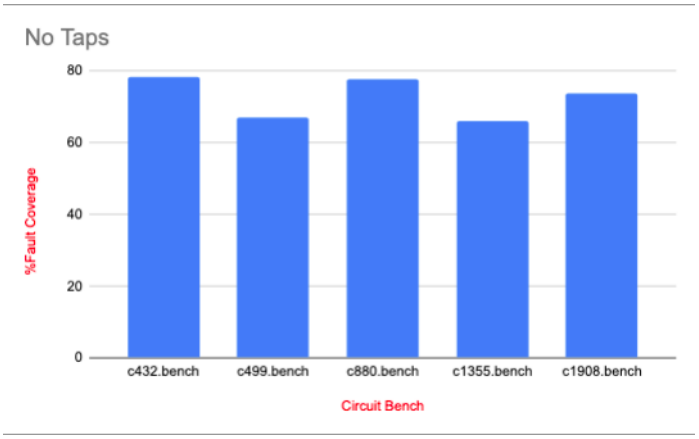

Fig. 1. This shows the result of No Taps with different circuit benches.

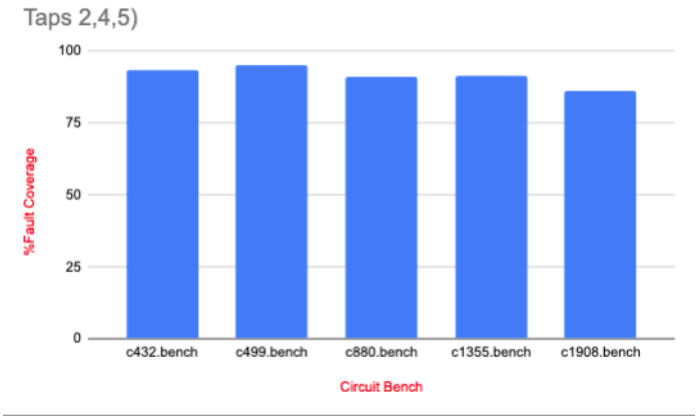

Fig. 2. This shows the result of Taps at (2,4,5) with different circuit benches.

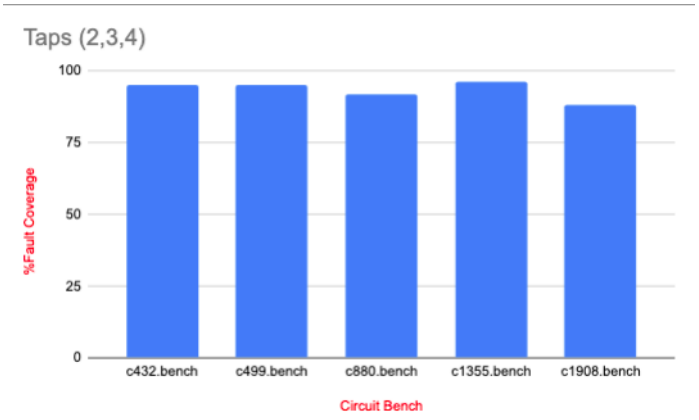

Fig. 3. This shows the result of Taps at (2,3,4) with different circuit benches.

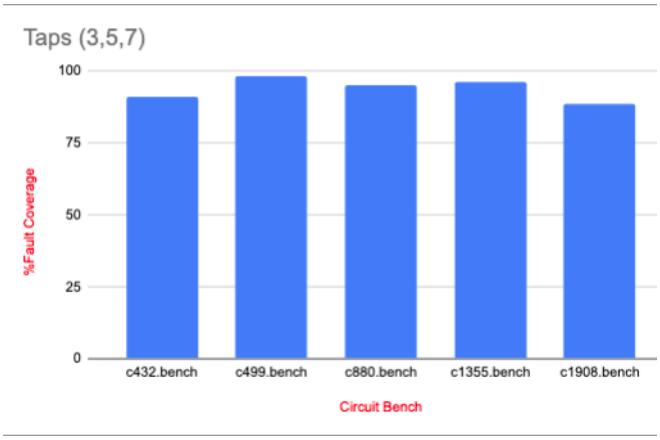

Fig. 4. This shows the result of Taps at (3,5,7) with different circuit benches.

We plot the testing for all benches c432 and c880 with 40 faults with combination of 3 faults

.

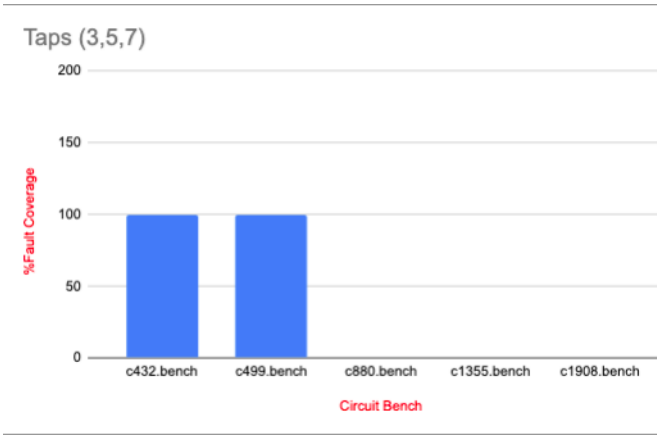

Fig. 5. This shows the result of Taps at (3,5,7) with different circuit benches For a combination of 3 Faults

IV. Discussion of Results

A. *1000 faults with a combination of 2*

As the benches get bigger LFSR performance deteriorates, Taps (3,5,7) gives better fault coverage; Taps give a better fault coverage than No taps We observe that LFSRs give a better fault coverage when compared to single faults

B. *40 faults with a combination of 3*

We could not run bigger benches as our Python Sim was not strong enough, we managed to run C432.bench and c499.bench we get full 100 percent fault coverage.

C. *Conclusion*

1) LFSRs Performs better for multi-faults when compared to single faults, this is because some faults cancel each other and it is more likely that when a TV covers a fault it also covers the other fault
2) We still confirm that having Taps makes LFSRs more random thus better coverage of faults is achieved
3) As the no. of combinations increase that is 2 faults to 3 we see the fault coverage increases as we achieved 100 percent coverage for c432, c499 and would have achieved similar results with other larger benches.

D. *References*

1) F. Elguibaly and M. W. El-Kharashi, ”Multiple-input signature registers: an improved design,” 1997 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, PACRIM. 10 Years Networking the Pacific Rim, 1987-1997, Victoria, BC, Canada, 1997, pp. 519-522 vol.2, doi: 10.1109/PACRIM.1997.620315.
2) https://www.sciencedirect.com/topics/mathematic/linear-feedback-shift-register