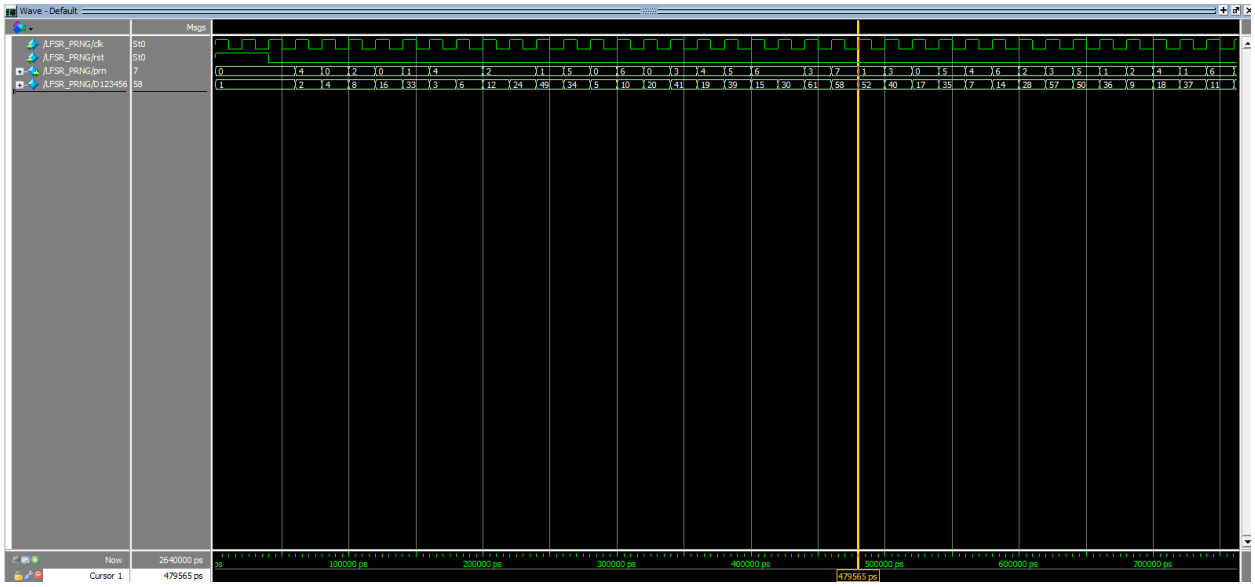
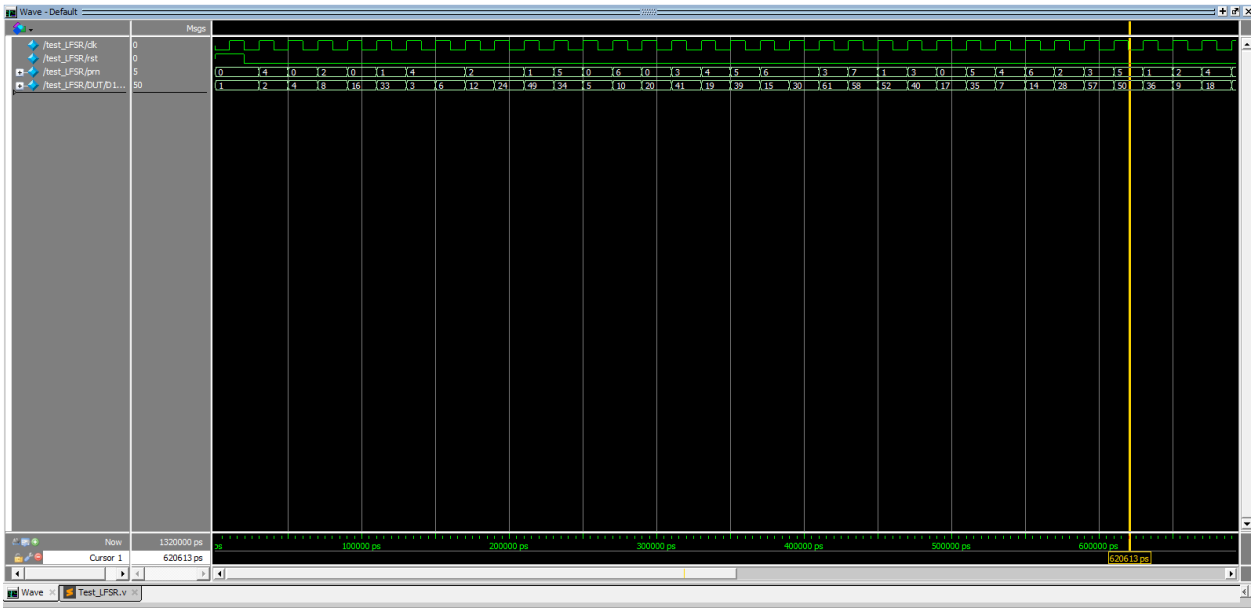
FPGA Assignment 1 Report

A. Deliverables

1. Simulation of LFSR_PRNG.v



2. Simulation of Test_LFSR.v

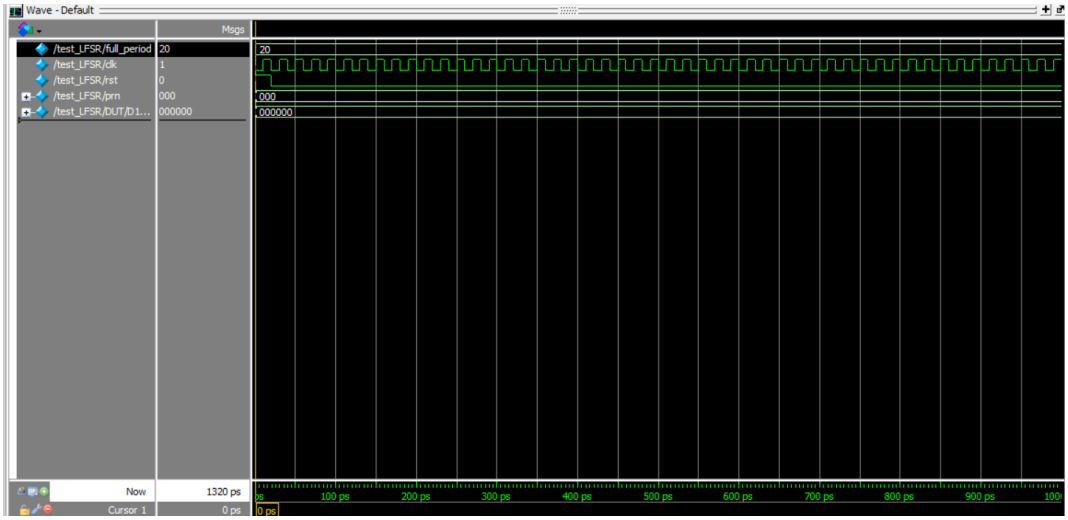


3. Pin Assignments (De0-Nano-SoC)

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate
clk	Input	PIN_V11	3B	B3B_N0	2.5 V		12mA (default)	
led	Output	PIN_V15	5A	B5A_N0	2.5 V		12mA (default)	1 (default)
prn[2]	Output	PIN_V16	5A	B5A_N0	2.5 V		12mA (default)	1 (default)
prn[1]	Output	PIN_AA24	5A	B5A_N0	2.5 V		12mA (default)	1 (default)
prn[0]	Output	PIN_W15	5A	B5A_N0	2.5 V		12mA (default)	1 (default)
rst	Input	PIN_AH17	4A	B4A_N0	2.5 V		12mA (default)	
sw	Input	PIN_AH16	4A	B4A_N0	2.5 V		12mA (default)	

- Describe the scenario when the initial value of the LFSR (seed) is 0x0.

  When I changed the initial value of LFSR to zero, the output D123456 and the prn all appear as zero. Basically, when all the bits are initially zeros, all the outputs of the xor operations are zero and no random numbers will be generated. (0 ^ 0 = 0)

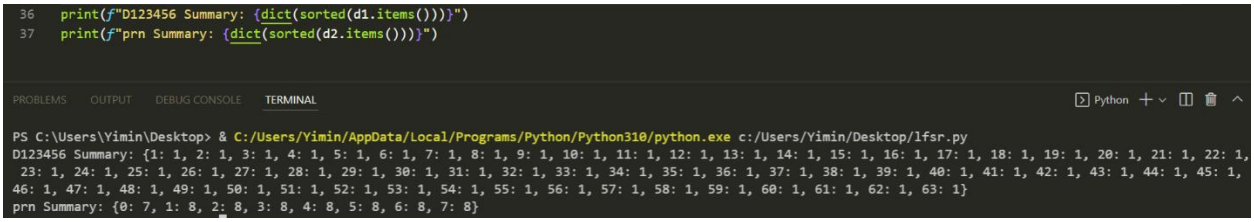


- Write down a table with the first 63 values generated by the LFSR in signals *D123456* and *prn*. During a "period" (i.e. the first *(2^6)-1* values), how many times does each value occur in each of the signals? Is there a bias (i.e. is any value occurring more often/less)?

Iteration	D123456	ptr
1	1	0
2	2	4
3	4	0
4	8	2
5	16	0
6	33	1

7	3	4
8	6	4
9	12	2
10	24	2
11	49	1
12	34	5
13	5	0
14	10	6
15	20	0
16	41	3
17	19	4
18	39	5
19	15	6
20	30	6
21	61	3
22	58	7
23	52	1
24	40	3
25	17	0
26	35	5
27	7	4
28	14	6
29	28	2
30	57	3
31	50	5
32	36	1
33	9	2
34	18	4
35	37	1
36	11	6
37	22	4
38	45	3
39	27	6
40	55	5
41	46	7
42	29	2
43	59	7
44	54	5
45	44	3
46	25	2
47	51	5

48	38	5
49	13	2
50	26	6
51	53	1
52	42	7
53	21	0
54	43	7
55	23	4
56	47	7
57	31	6
58	63	7
59	62	7
60	60	3
61	56	3
62	48	1
63	32	1



(Data and statistics generated by python.)

For D123456, each integer from 1 to 63 appears once. For prn, each integer from 1 to 7 appears eight times while 0 appears 7 times in the period. Since the initial value of prn is zero that it counts to 8 as well, there is no bias that each signal appears with the same number of times.

- When is it "safe" to define logic asynchronously, outside a process, and when is it not?

It is safe to define logic asynchronously only when you want totally combinational logics, and when you want an instantaneous output due to the change of the input.

However, for a system that we want to design sequential logics to store some data for future usage, such as a feedback system like the LFSR in this lab, it is not safe to define logic asynchronously since that will damage the datapath and make things confusing. We should stick with synchronous logic to have the data flowed with a fixed time(rising_edge of a clock) to easily manage the data flow of the design.

- What is a testbench? In your opinion, what are the pros/cons of using ModelSim *Force* and *Clock* versus using a testbench?

  A testbench is a non-synthesizable file that describes the details of a behavior test, including clock period, the status of all inputs at various time. It is a file that can be compiled by the ModelSim for simulation. We can do the same thing by hand, but with more complicated design and repetitive tests needed, a testbench makes the test more accurate and saves a huge amount of time.

  ModelSim Force and *Clock*:

  - Pros: It is more flexible that I could easily change the input signals by hand whenever I want during the simulation for each test.
  - Cons: It is time-consuming to force every input signal and simulate by hand when the design is complicated, it is also tedious when we need to test the same module for a lot of times.

  Testbench:

  - Pros: It reduces the effort for repetitive works if we need to run simulation to test a module several times with the same test vectors. With the help of testbench to help automate the simulation, it could cost a lot less time.
  - Cons: Since all the changes of the inputs are set in advance in the testbench file, we cannot change the signal during the test. It is more difficult to change some specific test vectors. We have to go back into the testbench, modify the code to change some input vectors.

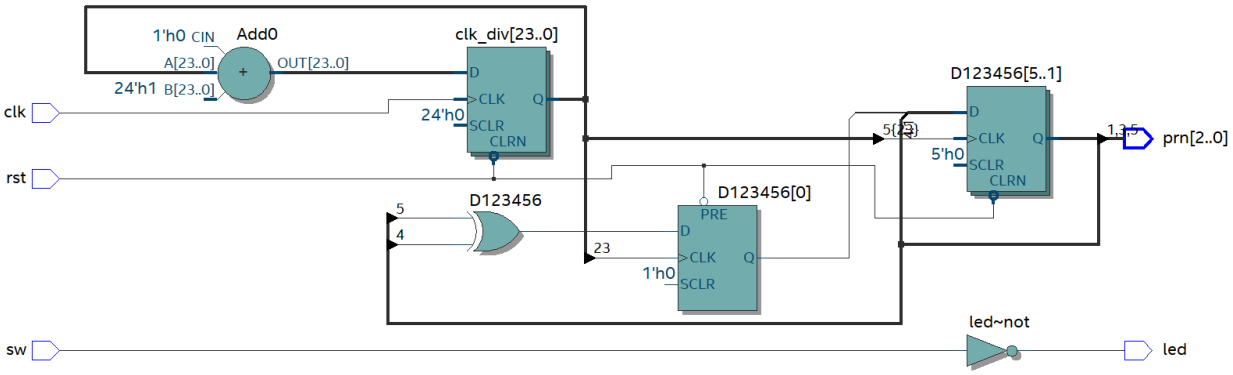
- If the board clock frequency is 50 MHz, what is the frequency of "*clk_slow*"?

  Since the clk_slow switches to 1 every 2^24 increments of clk (if we observe the time slice between the points when clk_slow turns from 1 to 0, the period is 2^24 of clk cycle), we can get:

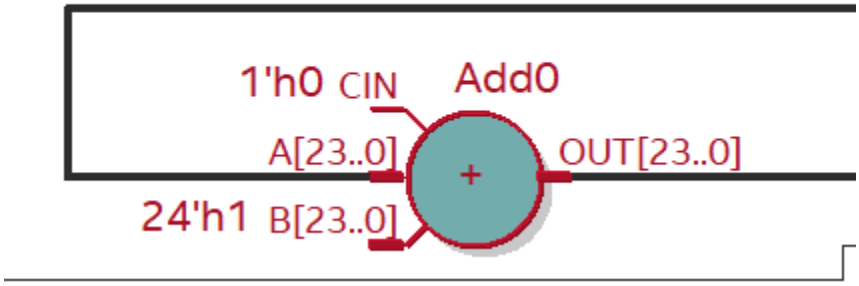
  fclk_slow = fclk/(2^24) = 2.98 Hz

- In Quartus Prime, go to *Tools->Netlist Viewers* and explore the generated schematics. Describe what each of the viewers attempts to display. For one of them, describe the blocks and wiring comprising your design (please include a screen snip of the schematics you are describing).

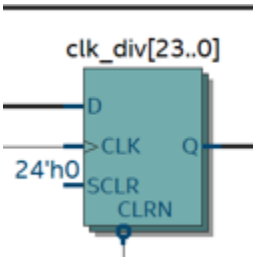
Overall schematic:



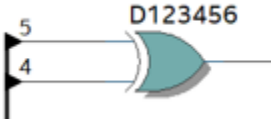
1. This is an adder that increments the value of the counter by one. The input is the output of the counter while the output is an input of the counter.



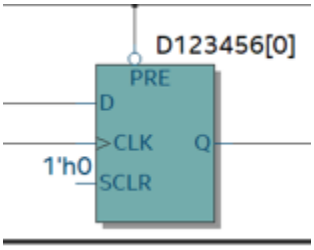
2. This is a register that stores the current value of the counter (The counter increments by one with the original clk frequency) triggered by the rising edge of the board clk. It has a 24-bit input and a 24-bit output, it also has an asynchronous reset. The most significant bit of the output Q is the clk_slow signal which is a slower frequency that we can observe.



3. This is a xor gate which generates the bit 0 of the signal D123456.



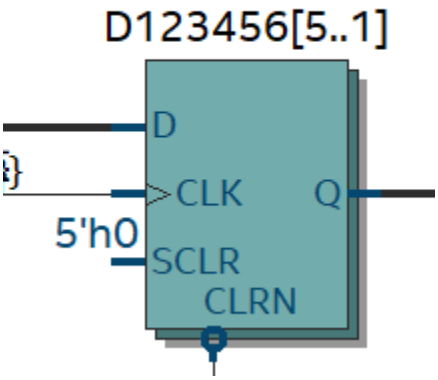
4. This is a register that stores the least bit of the signal D123456, with the frequency of clk_slow (a frequency that we can observe the change of LED). The input signal is the result of XOR and the output links to the D123456[1] as its input for next clk_slow cycle. The register also has an asynchronous reset.



5. This is a register that stores bit 1 to bit 5 of D123456 with frequency of clk_slow, which is the result of a shifting operation. The output bits D123456[5], D123456[3], D123456[1] link to prn[2:0].

   The wiring structure of the register is:

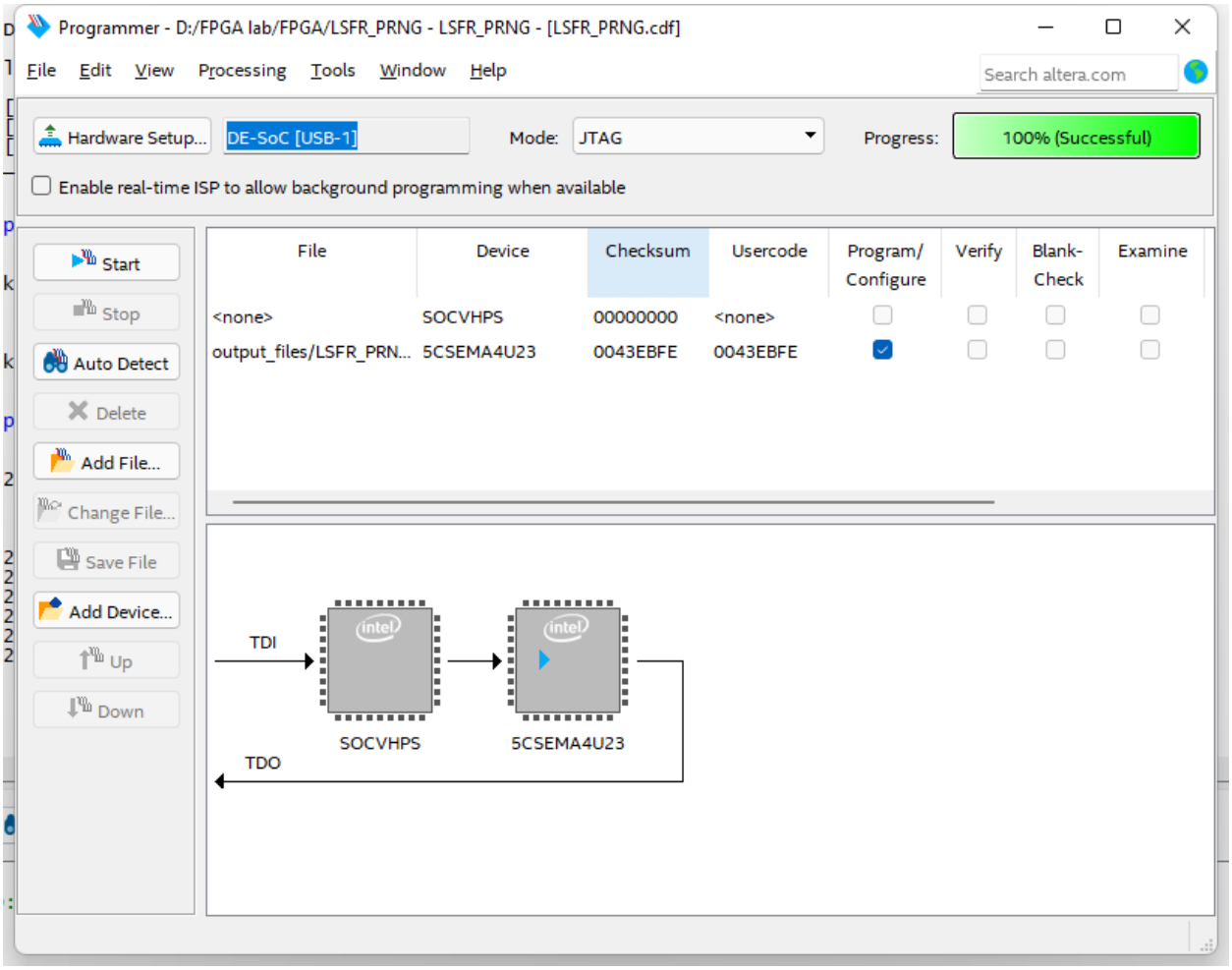
Reg Q ports	(wires to)	Reg D ports	XOR input ports
D123456[1]	=>	D123456[2]	
D123456[2]	=>	D123456[3]	
D123456[3]	=>	D123456[4]	
D123456[4]	=>	D123456[5],	XOR INPUT 1
D123456[5]	=>		XOR INPUT 2
D123456[0]	=>	D123456[1]	



6. Here is a simple NOT gate. The input is the switch signal and the output links to the signal led. It is a combinational logic that the led turns on whenever the button sw is pressed.

C. Comments:

- I learned how to test the design using ModelSim, and implement it on an FPGA. I am using De0-Nano-SoC. So what I have done differently from the wiki is the pin assignment, I checked the manual of the board and assigned the pins successfully.
- One thing that I got stuck for a while is how to program the board through JTAG. Since there are two devices (FPGA and HPS) on the JTAG chain for De0-Nano-SoC, I have to auto-connect the two devices, select the device 5CMESA4 and add the bitstream file to it to load the bitstream. At first, I only added the 5CSEMA4U device by hand without having the HPS and tried to program it, and it failed. I figured it out after looking into the manual of the board.



- It is interesting to get FPGA working as I desired. I can't wait to implement some more interesting designs on the board!

C. Here is a python script I used to generate data for question 2, which outputs the same results as the simulation in Modelsim:

```
d = [0, 0, 0, 0, 0, 1]
D123456 = []
d1 = dict()
prn = []
d2 = dict()

for i in range(63):
    x = d[0]
    d.pop(0)
    d.append(x^d[0])
    x = 2**5 * d[0] + 2**4 * d[1] + 2**3 * d[2] + 2**2 * d[3] + 2 * d[4] + d[5]
    y = 2**2 * d[4] +2 * d[2] + d[0]
    D123456.append(x)
    prn.append(y)

print("----------D123456 Data----------")
for r in D123456:
    print(r)

print("----------prn Data----------")
for r in prn:
    print(r)

for r in D123456:
    if r in d1.keys():
        d1[r] += 1
    else:
        d1[r] = 1


for r in prn:
    if r in d2.keys():
        d2[r] += 1
    else:
        d2[r] = 1


print("----------D123456 Dict----------")
print(f"D123456 Summary: {dict(sorted(d1.items()))}")
print("----------prn Dict----------")
print(f"prn Summary: {dict(sorted(d2.items()))}")
```