Design of an 8-bit LFSR

Linear Feedback Shift Registers (LFSRs) are not complete random number generators. They commonly called pseudo-random number generators (PRNGs). However, due to simple architecture and low-cost implementation, they are used in many applications.

In this workshop, we will design and simulate an 8-bit LFSR. We will use Intel's Quartus II design tool software to design our LFSR project and ModelSim to perform simulations of the design.

Let us have a look at the LFSR architecture first. Figure 1, depicts the architecture of our an 8-bit LFSR. To have a statistically better output, I considered a 10-bit register; the output of the LFSR is derived from the lower 8 bits of this 10-bit register. The feedback bit is generated by XORing of bit numbers 8, 7, 4, 2, and 0 and is fed to the MSB (bit 9). As you can see, the 10-bit register is seeded to 1A6H; i.e. when it resets, the output of the LFSR is A6H and the feedback bit is one, as shown in Figure 2 (top).

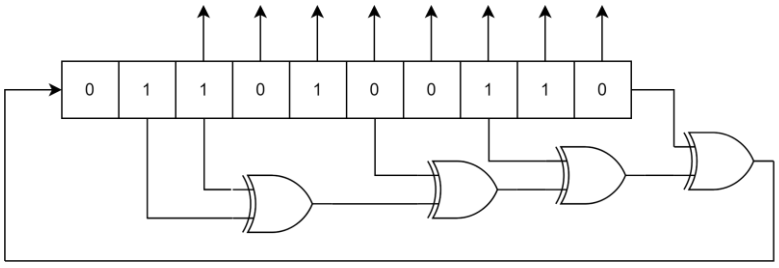


Figure 1: Architecture of our 8-bit LFSR

At the rising edge of the first clock after reset, the feedback bit will be placed in the bit 9 of the register and all bits will be shifted to the right. Therefore, the output will be D3H and feedback bit will be one. You can see this state in figure 2 (bottom).

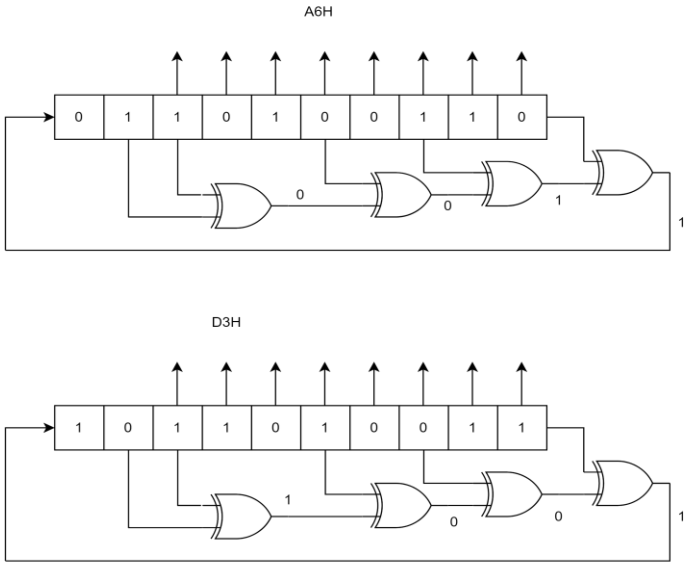


Figure 2: (Top) the LFSR in reset state. (Bottom) the LFSR after first clock after reset

After rising edge of the second and third clock pulse, the output of the LFSR will be 69H and B4H, respectively. Those states are depicted in Figure 3.

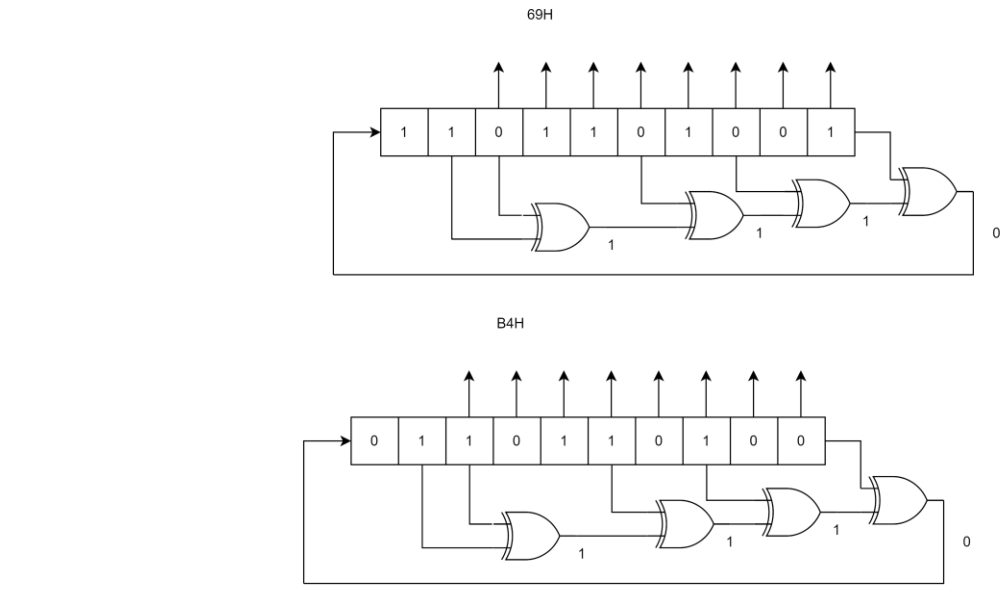


Figure 3: The LFSR after second (Top) and third (Bottom) states after reset.

Figure 4, illustrates the gate-level design of our LFSR which consists of 10 D-type flip-flops and four 2-in XOR gates. The D-flip-flops (DFFs) are tagged as u1-u10. To implement the seed the flip-flops must output 0 or 1 at reset. Therefore, we designed two types of flip-flops; i.e. reset to zero or dff0 and reset to one or dff1. The Verilog code for dff0, dff1 and the 2-in XOR gate is shown in blocks 1-3.

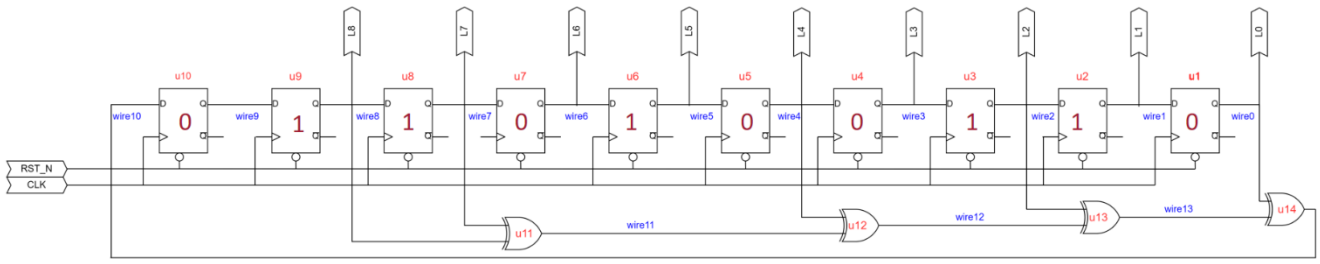


Figure 4: Gate-level design of the 8-bit LFSR

Open Quartus II software and add a new project. Locate the project on your hard drive and name the project and top-level design entity as 'lfsr'.

Use FPGA 5CSEBA6U23I7.
Add dff0, dff1, xor2in, and lfsr Verilog codes from Blocks 1-4 to your design.
Add lfsr_tb Verilog code (Block 6) to your design.

Block 1: Verilog code for dff0

```
module dff0
(
 input   i_clk,
 input   i_rst_n,
 input   i_d,
 output  o_q
);
reg r_q;
assign  o_q = r_q;
always @(posedge i_clk or negedge i_rst_n)
begin
  if(!i_rst_n)
     r_q <= 1'b0;
  else
     r_q <= i_d;
end
endmodule
```

Block 2: Verilog code for dff1

```
module dff1
(
 input   i_clk,
 input   i_rst_n,
 input   i_d,
 output  o_q
);
reg r_q;
assign  o_q = r_q;
always @(posedge i_clk or negedge i_rst_n)
begin
  if(!i_rst_n)
    r_q <= 1'b1;
  else
    r_q <= i_d;
end
endmodule
```

Block 3: Verilog code for 2-in xor

```
module xor2in
 (
 input   i_a,
 input   i_b,
 output  o_c
);
assign o_c = i_a ^ i_b;
endmodule
```

-

Block 4: Verilog code for Gate-level design of the LFSR

```
module lfsr(
input            i_clk,
input            i_rst_n,
output [7:0]  o_lfsr);
wire [13:0]  w_wire;
assign o_lfsr[7:0] = w_wire[7:0];
// Gate Level design
dff0  u1(.i_clk(i_clk),
         .i_rst_n(i_rst_n),
         .i_d(w_wire[1]),
         .o_q(w_wire[0]));
dff1  u2(.i_clk(i_clk),
         .i_rst_n(i_rst_n),
         .i_d(w_wire[2]),
         .o_q(w_wire[1]));
dff1  u3(.i_clk(i_clk),
         .i_rst_n(i_rst_n),
         .i_d(w_wire[3]),
         .o_q(w_wire[2]));
dff0  u4(.i_clk(i_clk),
         .i_rst_n(i_rst_n),
         .i_d(w_wire[4]),
         .o_q(w_wire[3]));
dff0  u5(.i_clk(i_clk),
         .i_rst_n(i_rst_n),
         .i_d(w_wire[5]),
         .o_q(w_wire[4]));
dff1  u6(.i_clk(i_clk),
         .i_rst_n(i_rst_n),
         .i_d(w_wire[6]),
         .o_q(w_wire[5]));
dff0  u7(.i_clk(i_clk),
         .i_rst_n(i_rst_n),
         .i_d(w_wire[7]),
         .o_q(w_wire[6]));
dff1  u8(.i_clk(i_clk),
         .i_rst_n(i_rst_n),
         .i_d(w_wire[8]),
         .o_q(w_wire[7]));
```

```
dff1 u9(.i_clk(i_clk),
        .i_rst_n(i_rst_n),
        .i_d(w_wire[9]),
        .o_q(w_wire[8]));
dff0  u10(.i_clk(i_clk),
          .i_rst_n(i_rst_n),
          .i_d(w_wire[10]),
          .o_q(w_wire[9]));
xor2in  u11(.i_a(w_wire[7]),
            .i_b(w_wire[8]),
            .o_c(w_wire[11]));
xor2in  u12(.i_a(w_wire[4]),
            .i_b(w_wire[11]),
            .o_c(w_wire[12]));
xor2in  u13(.i_a(w_wire[2]),
            .i_b(w_wire[12]),
            .o_c(w_wire[13]));
xor2in  u14(.i_a(w_wire[0]),
            .i_b(w_wire[13]),
            .o_c(w_wire[10]));
endmodule
```

From Quartus II menu bar, got to Assignments -> settings -> EDA tools settings -> simulations

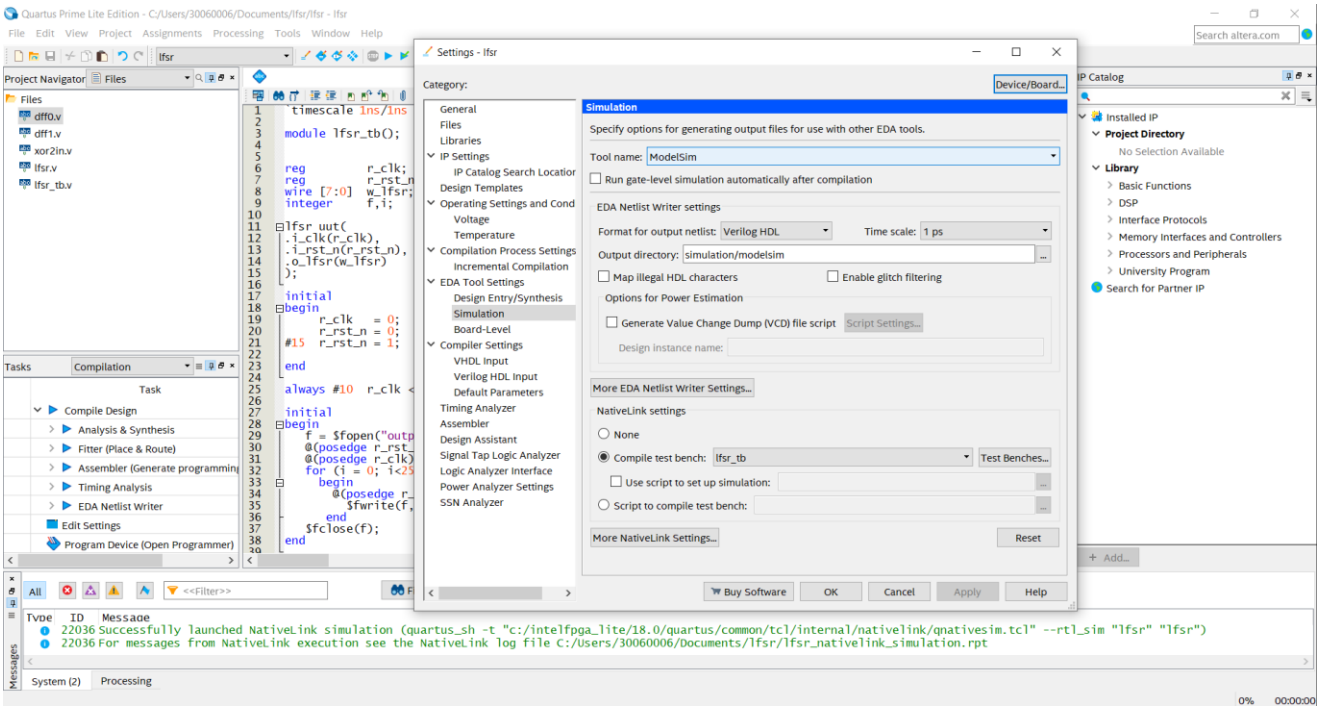
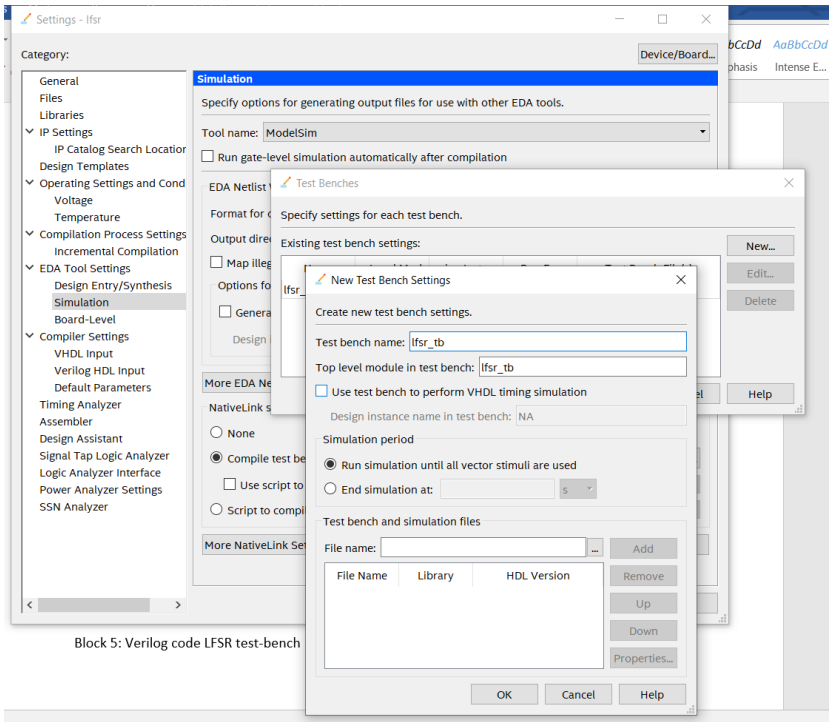


Figure 5: Assigning simulation tools to Quartus

Select tool name = ModelSim or ModelSim-Altera; whichever you use on your computer.
Select Compile test bench and click on Test Benches … button.
On the new opened window click on New.



Block 5: Verilog code LFSR test-bench

Figure 6: Introducing the test bench file to Quartus

Name the test bench lfsr_tb, as well as test bench top level.
On file name click on … and locate the lfsr_tb.v file you already added to your project.
Click on Ok on this and the former window. Finally click on Apply button on the setting window.

On the Quartus II main window, click on start Analysis and synthesis (Ctrl +K).

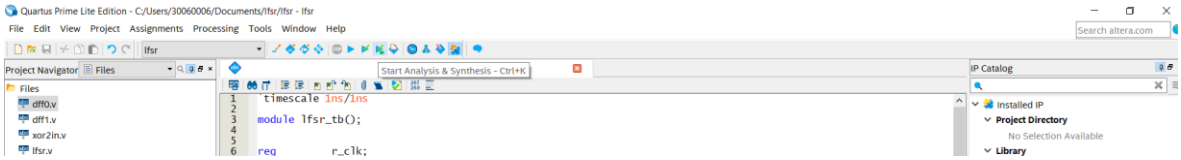


Figure 8: starting Analysis and Synthesis

Once the synthesis process complete, Go to Tools -> Run Simulation Tool -> RTL simulation

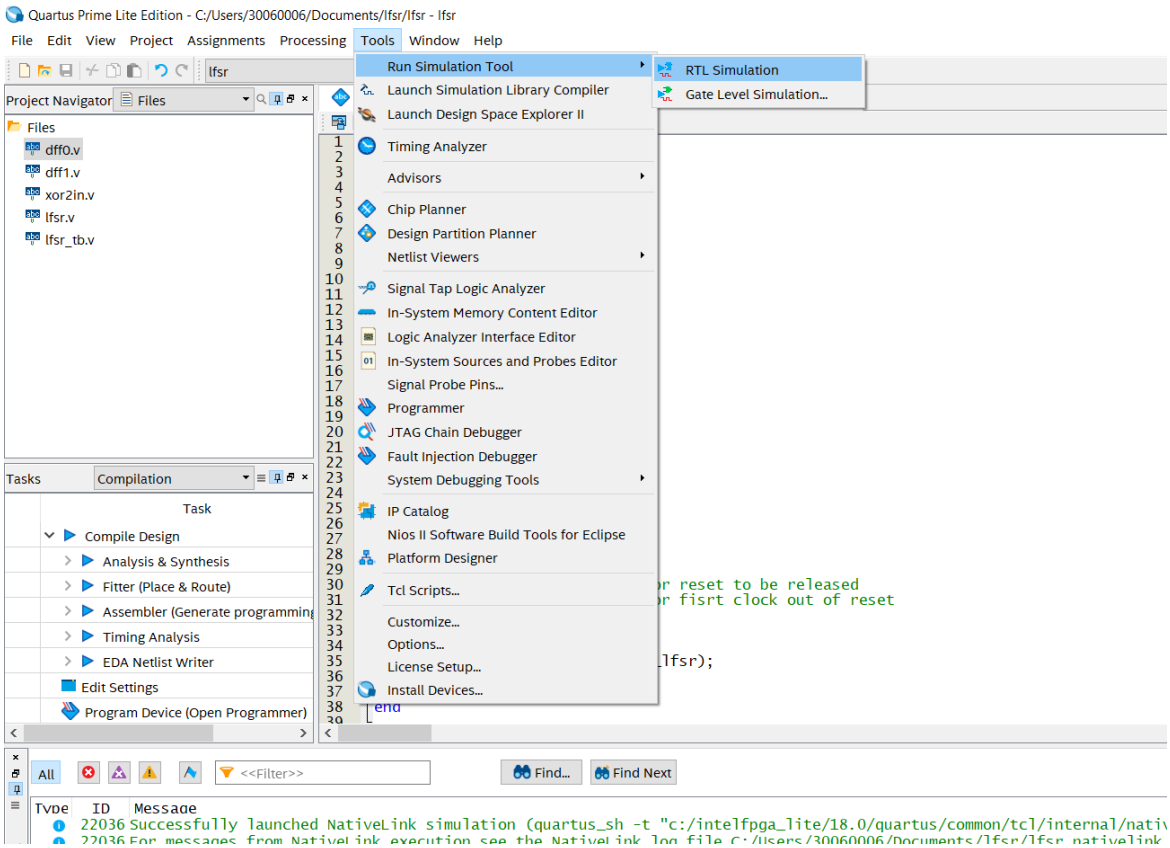


Figure 8: Running the ModelSim simulator from Quartus

Block 5: Verilog code LFSR test-bench

```
`timescale 1ns/1ns
module lfsr_tb();
reg        r_clk;
reg        r_rst_n;
wire [7:0]  w_lfsr;
integer     f,i;

lfsr uut
(
  .i_clk(r_clk),
  .i_rst_n(r_rst_n),
  .o_lfsr(w_lfsr)
);

initial
begin
         r_clk     = 0;
         r_rst_n = 0;
#15    r_rst_n = 1;
End

always #10  r_clk <= ~r_clk;

initial
begin
   f = $fopen("output.csv","w");
   @(posedge r_rst_n); //Wait for reset to be released
   @(posedge r_clk);   //Wait for first clock out of reset
   for (i = 0; i<2550; i=i+1)
     begin
       @(posedge r_clk);
        $fwrite(f,"%d,\n",  w_lfsr);
      end
   $fclose(f);
end
endmodule
```

The test bench writes 2550 samples of the LFSR output to file “output.csv”.
Block 7 suggests a Python script to evaluate the LFSR output. Run the script and see the histogram of the output. Discuss how good our LFSR works.
For reference, compare this histogram with the results of Python’s randint() function.

Block 7: Python code to evaluate LFSR's outputs

```
#Python code to analyse LFSR's
import matplotlib.pyplot as plt
import pandas as pd
df = pd.read_csv(r'\output.csv')
df.columns = ['A','B']
df.drop(['B'], axis=1, inplace=True)
plt.hist(df, bins = 255)
plt.show()

import random
import numpy as np
x=np.array(np.zeros(1000))
for i in range(1000):
    x[i] = (random.randint(1,255))
plt.hist(x,bins=255)
plt.show()
```

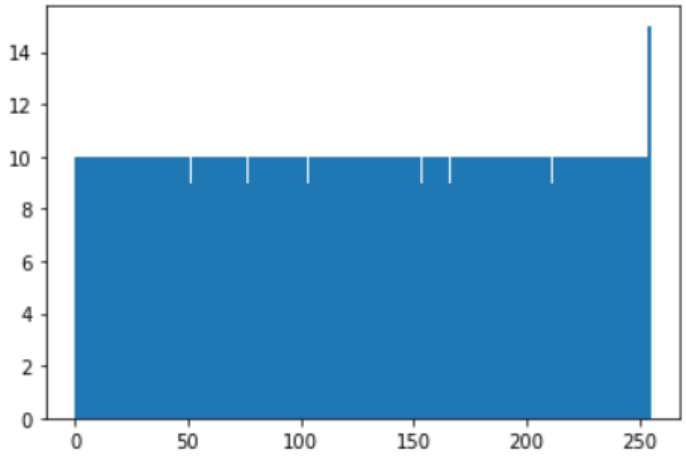


Figure : LFSR output histogram

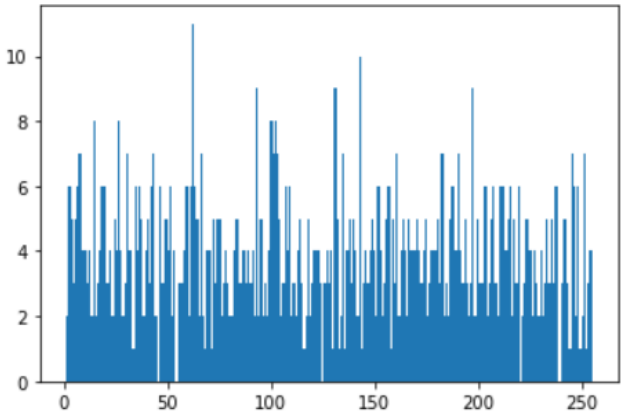


Figure: Python randint() histogram

You can design this LFSR at the RTL level. Block 6 shows the hardware description of the same LFSR in RTL level. Create a new project, use this code with the same test bench (Block 5), and compare the results of the two designs (Gate and RTL level).

Block 6: Verilog code for RTL-level design of the LFSR

```
module lfsr (
input            i_clk,
input            i_rst_n,
output [7:0]  o_lfsr
);
// RTL  design
reg  [9:0] r_lfsr ;
wire      w_feedback;
wire [7:0] w_lfsr;
assign w_feedback = r_lfsr[8]^r_lfsr[7]^r_lfsr[4]^r_lfsr[2]^r_lfsr[0];
assign w_lfsr = r_lfsr[7:0];
always @(posedge i_clk or negedge i_rst_n)
begin
   if(!i_rst_n)
      r_lfsr <= 10'b01_1010_0110;
   else
     begin
        r_lfsr[8:0] <= r_lfsr[9:1];
        r_lfsr[9] <= w_feedback;
     end
end
endmodule
```