



## Introdução a Sistemas Lógicos - Trabalho Prático 3:

Arthur Pontes Nader - 2019022294

Rodrigo Ferreira Araújo - 2020006990

Thales Henrique Silva - 2020007040

### 1) Introdução


Um registrador de deslocamento com feedback linear (LFSR) é um registrador em que o bit de entrada depende de uma função linear do seu estado anterior. Geralmente, utiliza-se uma porta ou mais portas XOR para gerar esse bit de entrada. A principal aplicação de um LFSR ocorre na geração de números pseudo-aleatórios

Assim, esse trabalho teve como principal objetivo a implementação de registradores de deslocamento com feedback linear utilizando a linguagem de descrição de hardware Verilog.

### 2) Atividades

#### 1. Implementação de cada LFSR

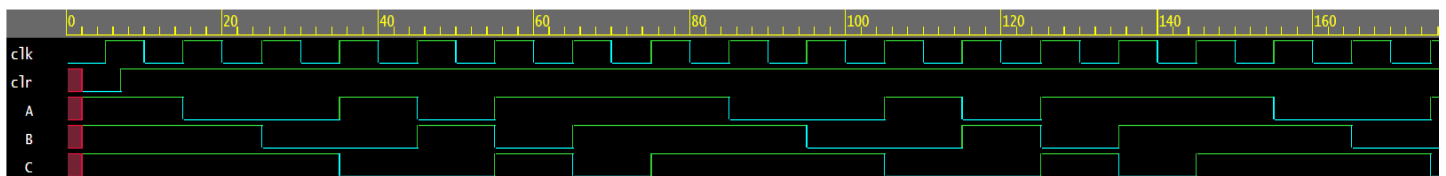
- LFSR para  $n = 3$

```
design.sv   
1 //Design  
2 module LFSR_3(clock,clear,ff_states);  
3     input clock,clear;  
4     output reg [2:0] ff_states= 3'b111;  
5     reg A,B,C;  
6     always @(posedge clock or negedge clear)  
7         begin  
8             //inicializacao dos flip flops  
9             if(!clear) begin A<=1;B<=1;C<=1; end  
10            //bit de entrada -> funcao linear do estado anterior  
11            else begin  
12                A<=C^B;  
13                B<=A;  
14                C<=B;  
15                ff_states[2] <= A;  
16                ff_states[1] <= B;  
17                ff_states[0] <= C;  
18            end  
19        end  
20 endmodule
```

```

testbench.sv
1 // Testbench
2 module Bancada_Teste;
3 //declaracao das variaveis
4 reg clk,clr;
5 wire [2:0] out;
6
7 LFSR_3 LF3(clk,clr,out);
8
9 //frequencia do clock e inicializacao do clear
10 initial
11     begin clk=1'b0;#2 clr=0;#5 clr=1; end
12 always #5 clk=~clk;
13
14 //exibicao das saidas
15 initial
16     begin
17         $monitor("out = %b", out);
18         #200 $finish;
19     end
20
21 initial
22     begin
23         $dumpfile("LFSR_3.vcd");
24         $dumpvars(0,Bancada_Teste);
25         #200 $finish;
26     end
27 endmodule

```



- LFSR para n = 4

```

design.sv
1 //Design
2 module LFSR_4(clock,clear,ff_states);
3     input clock,clear;
4     output reg [3:0] ff_states= 4'b1111;
5     reg A,B,C,D;
6     always @(posedge clock or negedge clear)
7         begin
8             //inicializacao dos flip flops
9             if(!clear) begin A<=1;B<=1;C<=1;D<=1; end
10            //bit de entrada -> funcao linear do estado anterior
11            else begin
12                A<=D^C;
13                B<=A;
14                C<=B;
15                D<=C;
16                ff_states[3] <= A;
17                ff_states[2] <= B;
18                ff_states[1] <= C;
19                ff_states[0] <= D;
20            end
21        end
22 endmodule

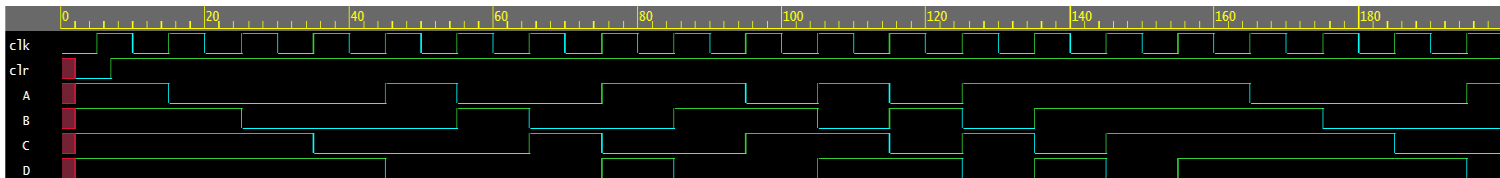
```

testbench.sv

```

1 // Testbench
2 module Bancada_Teste;
3     //declaracao das variaveis
4     reg clk,clr;
5     wire [3:0] out;
6
7     LFSR_4 LF4(clk,clr,out);
8
9     //frequencia do clock e inicializacao do clear
10    initial
11        begin clk=1'b0;#2 clr=0;#5 clr=1; end
12    always #5 clk=~clk;
13
14    //exibicao das saidas
15    initial
16        begin
17            $monitor("out = %b", out);
18            #200 $finish;
19        end
20
21    initial
22        begin
23            $dumpfile("LFSR_4.vcd");
24            $dumpvars(0,Bancada_Teste);
25            #200 $finish;
26        end
27 endmodule

```



- LFSR para  $n = 5$

design.sv



```

1 //Design
2 module LFSR_5(clock,clear,ff_states);
3   input clock,clear;
4   output reg [4:0] ff_states= 5'b11111;
5   reg A,B,C,D,E;
6   always @(posedge clock or negedge clear)
7     begin
8       //inicializacao dos flip flops
9       if(!clear) begin A<=1;B<=1;C<=1;D<=1; E<=1; end
10      //bit de entrada -> funcao linear do estado anterior
11      else begin
12        A<=E^C;
13        B<=A;
14        C<=B;
15        D<=C;
16        E<=D;
17        ff_states[4] <= A;
18        ff_states[3] <= B;
19        ff_states[2] <= C;
20        ff_states[1] <= D;
21        ff_states[0] <= E;
22      end
23    end
24 endmodule

```

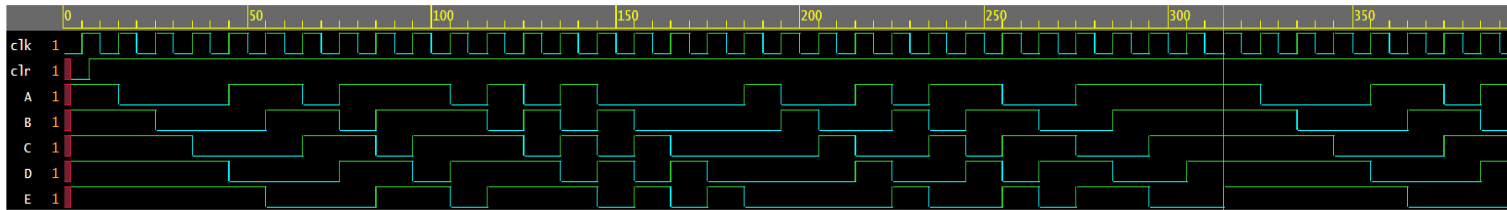
testbench.sv



```

1 // Testbench
2 module Bancada_Teste;
3   //declaracao das variaveis
4   reg clk,clr;
5   wire [4:0] out;
6
7   LFSR_5 LF5(clk,clr,out);
8
9   //frequencia do clock e inicializacao do clear
10  initial
11    begin clk=1'b0;#2 clr=0;#5 clr=1; end
12  always #5 clk=~clk;
13
14  //exibicao das saidas
15  initial
16    begin
17      $monitor("out = %b", out);
18      #200 $finish;
19    end
20
21  initial
22    begin
23      $dumpfile("LFSR_5.vcd");
24      $dumpvars(0,Bancada_Teste);
25      #200 $finish;
26    end
27 endmodule

```

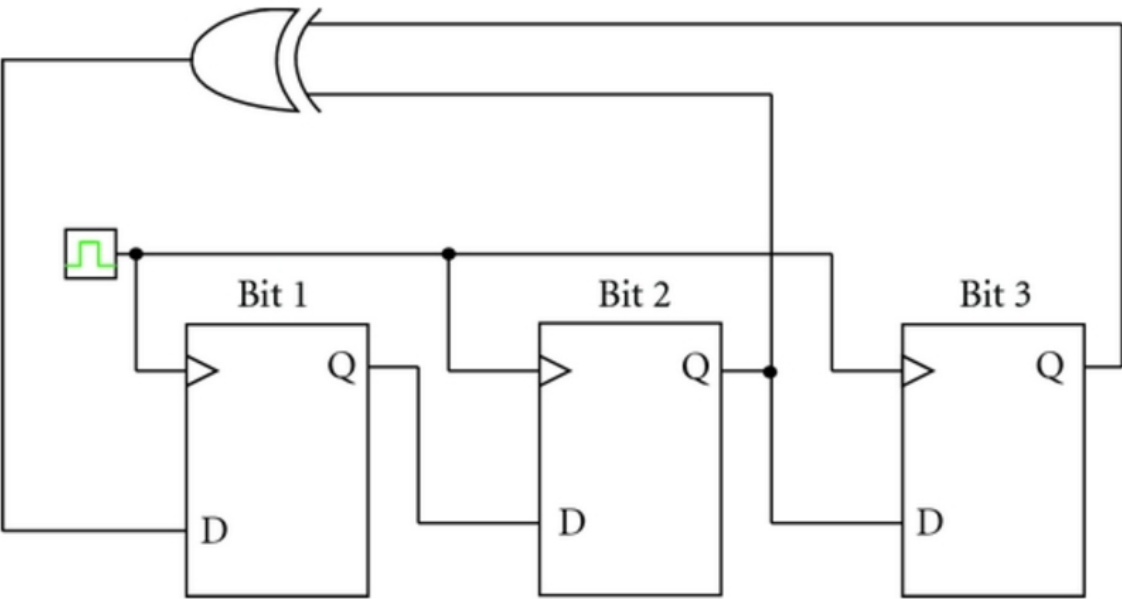


## 2. Apresentação dos resultados

Para cada LFSR implementado, é mostrado a seguir o circuito que o representa, o conteúdo do registrador em cada estado até a ocorrência de um novo ciclo e um gráfico com a representação decimal de acordo com o tempo. Os gráficos foram gerados utilizando a biblioteca Matplotlib da linguagem Python.

- **LFSR para  $n = 3$**

Circuito:

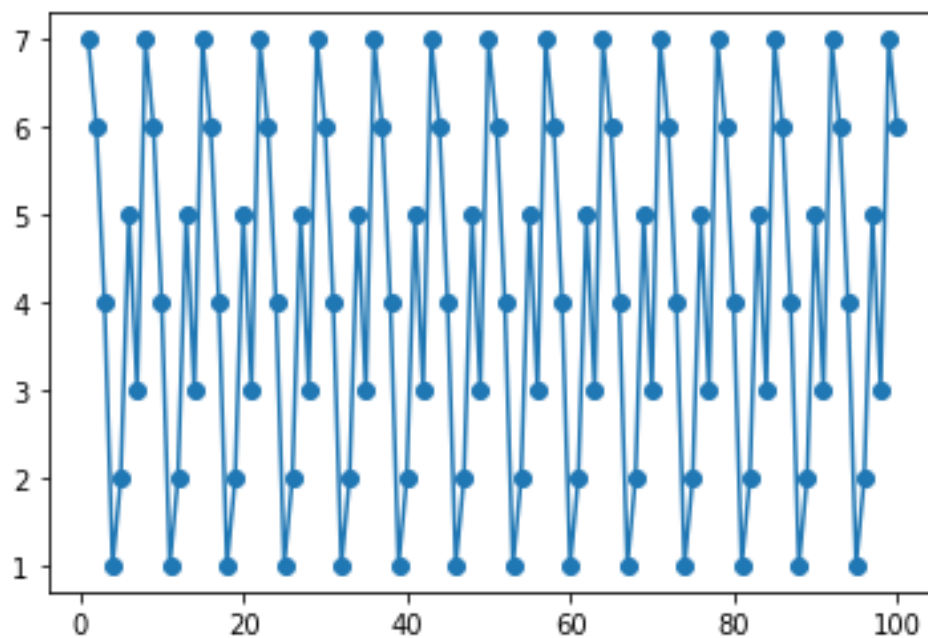


Estados:

```
Run Save* Copy* If this page reloads when you click "Run" please read this.
testbench.vv
Log Share
[2021-08-27 13:40:18 EDT] iverilog '-wall' design.vv testbench.vv && unbuffer vvp a.out
VCD info: dumpfile LFSR_3.vcd opened for output.
out = 111
out = 011
out = 001
out = 100
out = 010
out = 101
out = 110
out = 111
Finding VCD file...
./LFSR_3.vcd
[2021-08-27 13:40:18 EDT] Opening EPWave...
Done
```

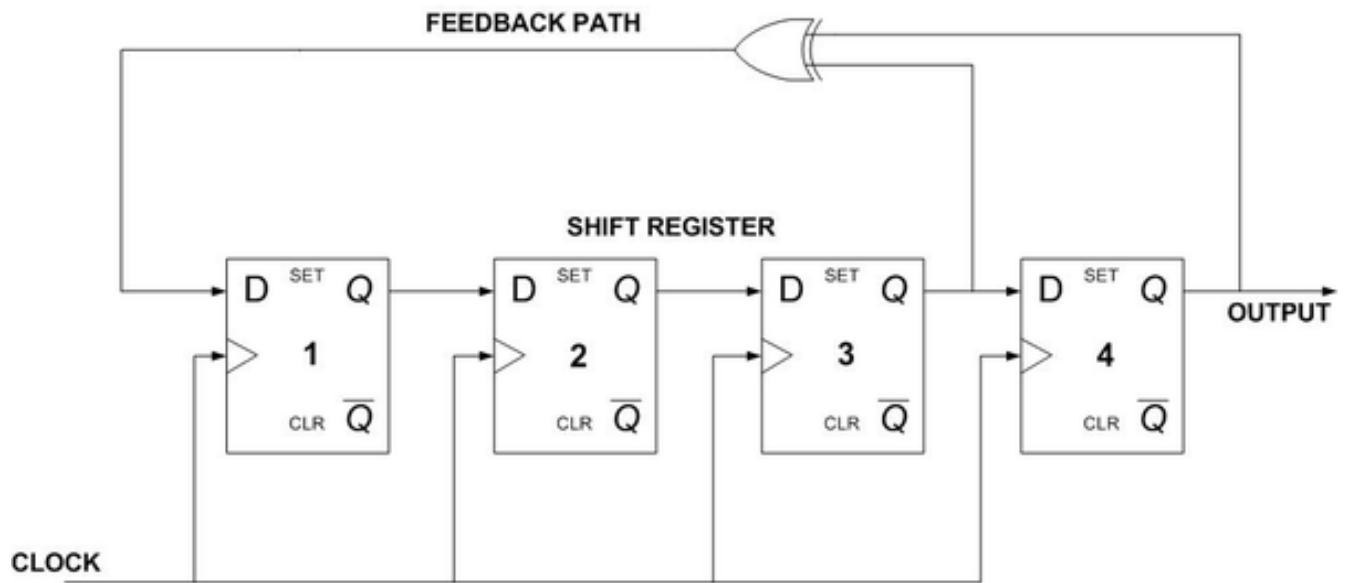
Gráfico:

$$N = 3; x^3 + x^2 + 1$$



- LFSR para  $n = 4$

Circuito:



Estados:

Run

Save\*

Copy\*

If this page reloads when you click "Run" please read [this](#).

testbench.sv

Log

Share

```

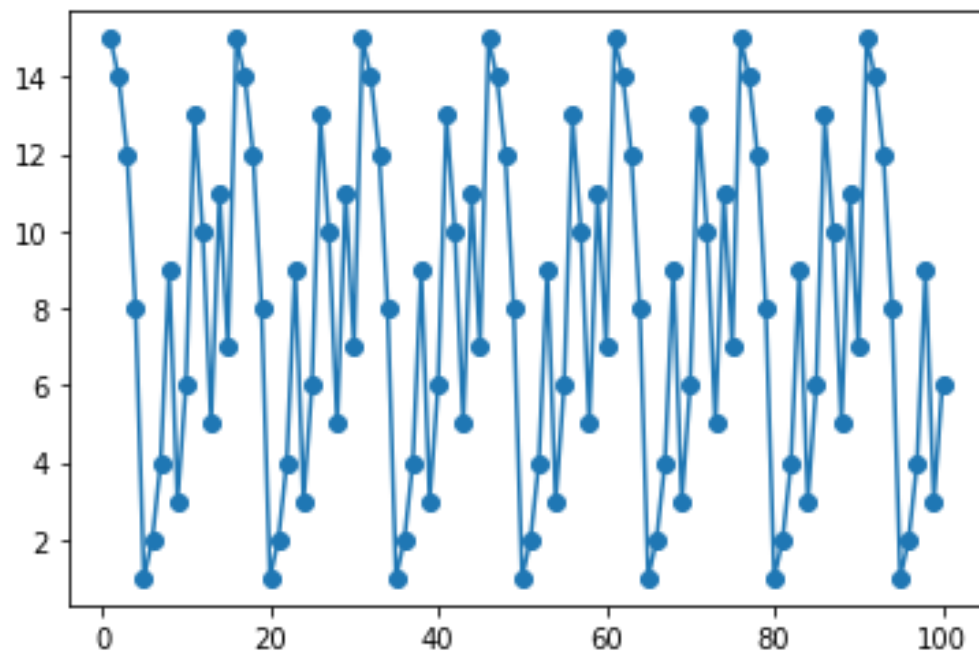
[2021-08-27 13:18:27 EDT] iverilog '-wall' design.sv testbench.sv && unbuffer vvp a.out
VCD info: dumpfile LFSR_4.vcd opened for output.
out = 1111
out = 0111
out = 0011
out = 0001
out = 1000
out = 0100
out = 0010
out = 1001
out = 1100
out = 0110
out = 1011
out = 0101
out = 1010
out = 1101
out = 1110
out = 1111

```

Done

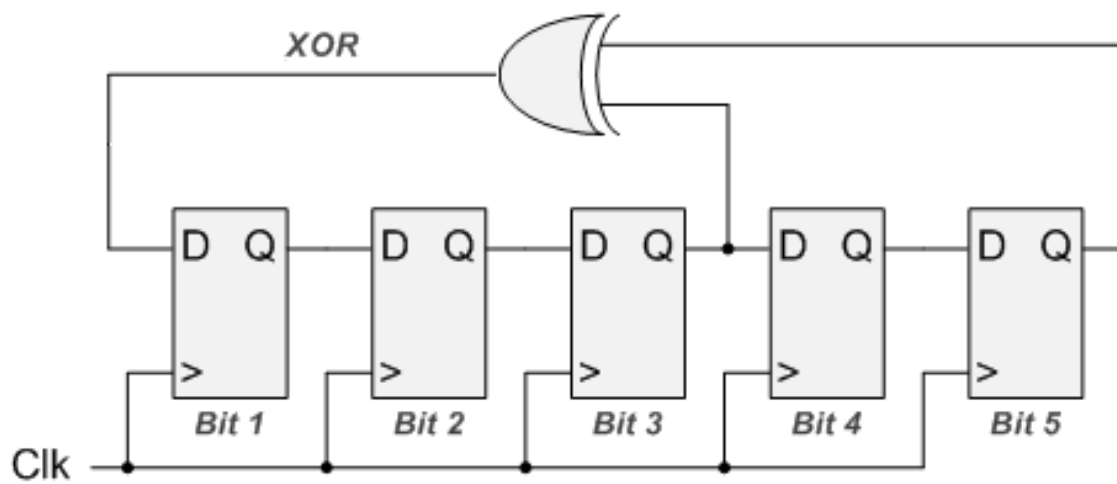
Gráfico:

$$N = 4; x^4 + x^3 + 1$$



- **LFSR para  $n = 5$**

Circuito:





Estados:

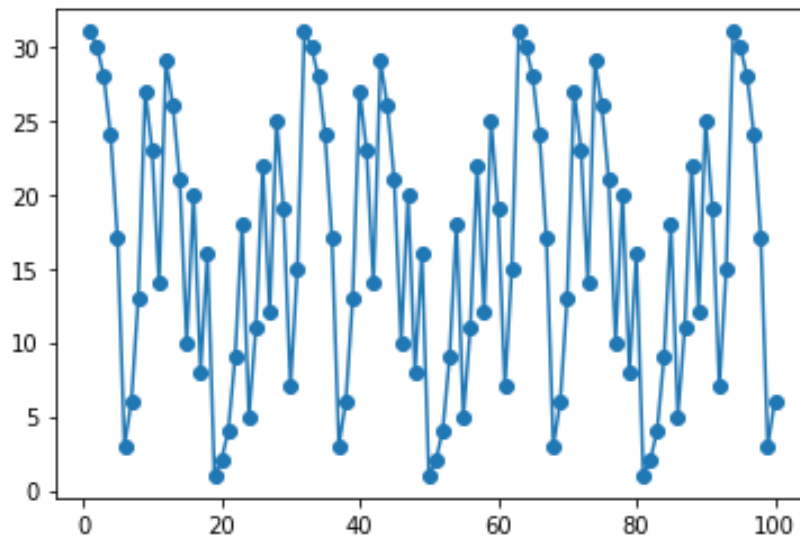
Log

Share

```
[2021-08-27 14:25:53 EDT] iverilog '-Wall' design.sv testbench.sv && unbuffer vvp a.out
VCD info: dumpfile LFSR_5.vcd opened for output.
out = 11111
out = 01111
out = 00111
out = 00011
out = 10001
out = 11000
out = 01100
out = 10110
out = 11011
out = 11101
out = 01110
out = 10111
out = 01011
out = 10101
out = 01010
out = 00101
out = 00010
out = 00001
out = 10000
out = 01000
out = 00100
out = 10010
out = 01001
out = 10100
out = 11010
out = 01101
out = 00110
out = 10011
out = 11001
out = 11100
out = 11110
out = 11111
Finding VCD file...
./LFSR_5.vcd
[2021-08-27 14:25:53 EDT] Opening EPWave...
Done
```

**Gráfico:**

$$N = 5; x^5 + x^3 + 1$$



### 3. Geração de um bitmap

O seguinte código em Verilog implementa um LFSR para  $n = 9$ , em que o período é 255, sendo assim difícil de observar o padrão.

```
design1.vv
1 //Design
2 module LFSR_5(clock,clear,ff_states);
3   input clock,clear;
4   output reg [7:0] ff_states= 8'b11111111;
5   reg A,B,C,D,E,F,G,H;
6   always @(posedge clock or negedge clear)
7     begin
8       //inicializacao dos flip flops
9       if(!clear) begin A<=1;B<=1;C<=1;D<=1; E<=1;F<=1;G<=1;H<=1; end
10      //bit de entrada -> funcao linear do estado anterior
11      else begin
12        A<=H^F^E^D;
13        B<=A;
14        C<=B;
15        D<=C;
16        E<=D;
17        F<=E;
18        G<=F;
19        H<=G;
20        ff_states[7] <= A;
21        ff_states[6] <= B;
22        ff_states[5] <= C;
23        ff_states[4] <= D;
24        ff_states[3] <= E;
25        ff_states[2] <= F;
26        ff_states[1] <= G;
27        ff_states[0] <= H;
28      end
29    end
30 endmodule
```

```

1 // Testbench
2 module Bancada_Teste;
3     //declaracao das variaveis
4     reg clk,clr;
5     wire [7:0] out;
6
7     LFSR_5 LF5(clk,clr,out);
8
9     //frequencia do clock e inicializacao do clear
10    initial
11        begin clk=1'b0;#2 clr=0;#5 clr=1; end
12        always #5 clk=~clk;
13
14    //exibicao das saidas
15    initial
16        begin
17            $monitor("out = %b", out);
18            #4000 $finish;
19        end
20
21    initial
22        begin
23            $dumpfile("LFSR_5.vcd");
24            $dumpvars(0,Bancada_Teste);
25            #4000 $finish;
26        end
27 endmodule

```

Ⓜ Log

```
[2021-08-27 15:35:33 EDT] iverilog '-wall' design.sv testbench.sv && unbuffer vvp a.out
```

```
out = 11111111
out = 01111111
out = 00111111
out = 00011111
out = 00001111
out = 10000111
out = 01000011
out = 10100001
```

Utilizou-se a linguagem Python para extrair o último bit de cada linha da saída do código Verilog:

```
[23] bits = bits.split("\n")
```

```
[26] bits_final = []
      for ele in bits:
          bits_final.append(int(ele[-1]))
      final += ele[-1]
```

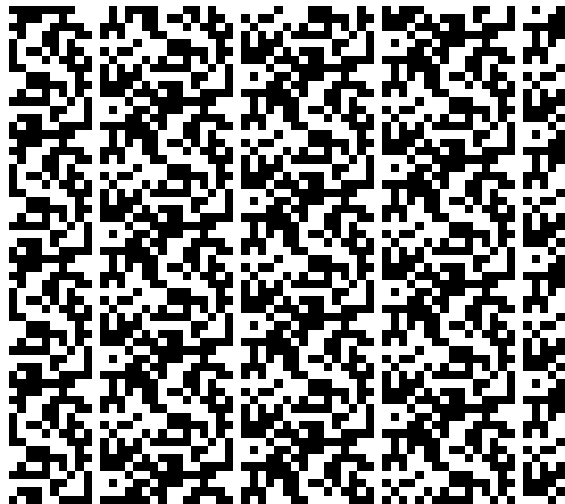
```
[27] print(bits_final)
```

[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0]

A sequência de 255 bits obtida foi:

```
"111111110000101111000110100000001000111000100101110000001100100100110  
1110010000010101101101011001011000011111011011110101110100010000110110  
0011110011100110001011010010001010010101001110111011001111011111101001  
1001101010001100000111010101011111001010000100"
```

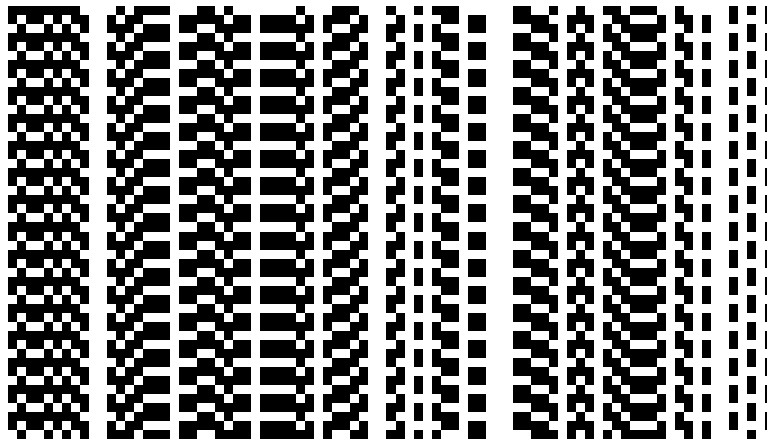
E usando um gerador de imagem a partir de dados binários, gerou-se as seguintes imagens, cada uma com uma determinada resolução para diferentes visualizações do padrão :



Razão: 68x60



Razão: 80x51



Razão 85x48

### 3) Conclusões:

As implementações de cada LFSR em Verilog geraram os resultados esperados. É possível notar que, como previsto, quanto mais flip-flops utilizados, maior fica o período até a repetição do valor inicial.

Para valores de “n” grandes, percebemos que é mais adequado utilizar bitmaps para visualizar a repetição das sequências. Além disso, nota-se que diferentes resoluções (razões) de imagens resultaram em padrões distintos, mas em todos os casos foi observada alguma periodicidade.

### 4) Referências

- [Linear Feedback Shift Register for FPGA \(nandland.com\)](http://nandland.com)
- [Schematic of a maximal length 3-bit linear feedback shift register. | Download Scientific Diagram \(researchgate.net\)](http://researchgate.net)
- [Why are primitive polynomial equations used in linear feedback shift registers? - Electrical Engineering Stack Exchange](http://www.electrical-engineering-stack-exchange.com)
- [Image to Binary Converter - Online Picture to Array \(dcode.fr\)](http://dcode.fr)