Trabalho Prático de Introdução aos Sistemas Lógicos

Raphaela Maria Costa e Silva - 2020006973
Alexander Martins Cavalieri - 2020065244
Vinicius Silva Gomes - 2021421869

Departamento de Ciência da Computação (DCC)
Universidade Federal de Minas Gerais (UFMG)
DCC114 - Introdução aos Sistemas Lógicos

Fevereiro de 2022

1) Para resolução dessa questão, foi implementado um código em Verilog que, primeiramente, criptografa o texto previamente escolhido e depois descriptografa esse texto, imprimindo no terminal o binário final em cada um dos passos.

A mensagem escolhida foi a palavra ‘code’ e o One-time pad foi gerado de maneira aleatória no site https://www.random.org/bytes/. Para conversão de ASCII para binário (e vice-versa), foi usado o site https://www.rapidtables.com/convert/number/ascii-to-binary.html.

‘code’: 01100011 01101111 01100100 01100101
One-time pad: 10001110 01101101 10001100 11000110

Para realizar a tarefa, uma porta XOR foi implementada e utilizada para criptografar e descriptografar a mensagem. No primeiro passo, foi realizada a operação XOR entre a mensagem e o One-time pad, bit a bit. O resultado da operação é salvo num outro registrador, que armazena a saída criptografada. Após realizar a operação XOR com cada um dos 32 bits, o conteúdo do vetor é exibido na tela.

De maneira similar, o processo inverso acontece ao realizar a operação XOR entre a mensagem criptografada e o One-time pad, bit a bit também. O resultado dessa operação também é salvo em outro registrador, que armazena a mensagem descriptografada. Com isso, será possível observar o conteúdo original contido na mensagem. Após o processo com os 32 bits, a mensagem descriptografada é exibida na tela.

```
Palavra escolhida: 'code'.

'code': 01100011 01101111 01100100 01100101
OTP:    10001110 01101101 10001100 11000110

Mensagem criptografada:
11101101 00000010 11101000 10100011
Mensagem descriptografada:
01100011 01101111 01100100 01100101
```

Exemplo das saídas do código.

Ao usar algum site para converter a resposta de binário para ASCII (foi usado o https://www.rapidtables.com/convert/number/binary-to-ascii.html) é possível visualizar que a mensagem descriptografada é igual a mensagem original. Dessa forma, o código é capaz de criptografar e descriptografar uma mensagem de 32 bits corretamente.

O código desenvolvido no EDAPlayground pode ser acessado através do link: https://www.edaplayground.com/x/qSeJ.

2) Os registradores de deslocamento com feedback linear que foram implementados possuem, respectivamente, os polinômios x^5+x^3+1 e x^3+x^2+1, sendo o primeiro de um LFSR de 5 bits e o segundo um LFSR de 3 bits.

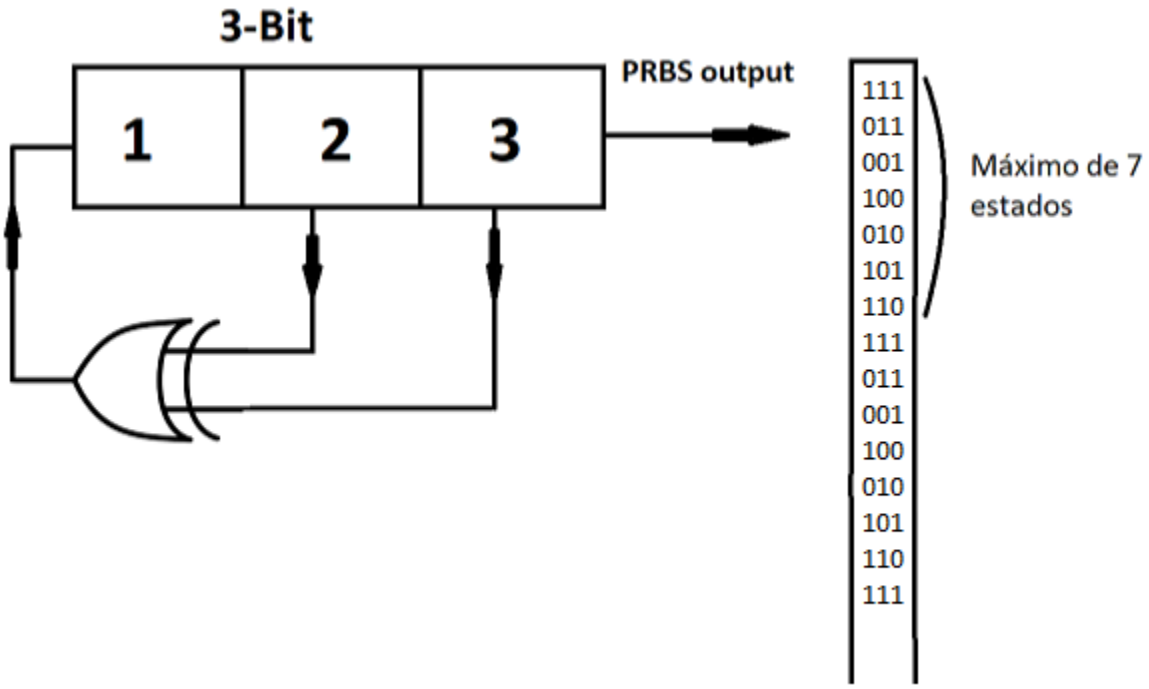
```
1  module LFSR3 (
2    input clk,
3    output reg[2:0] LFSR = 7
4  );
5
6    wire feedback = LFSR[2];
7
8    always @(posedge clk)
9      begin
10       LFSR[1] <= LFSR[0];
11       LFSR[2] <= LFSR[1];
12       LFSR[0] <= (LFSR[1] ^ feedback);
13     end
14 endmodule
```

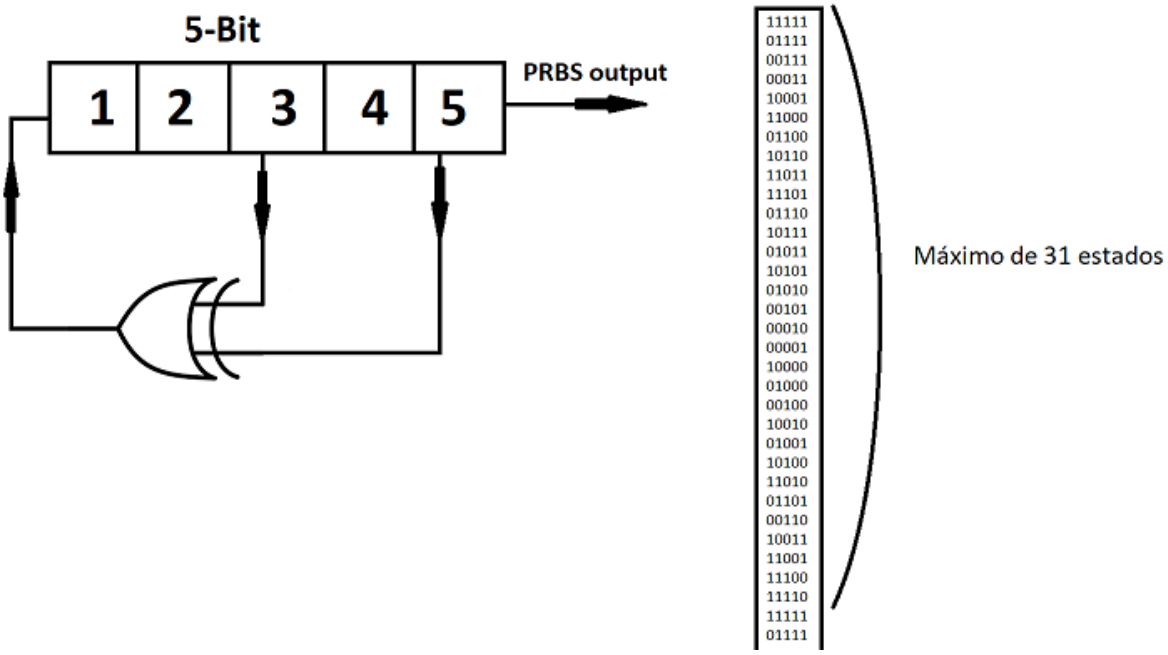
LFSR 3 bits com estado inicial em 111

```
1
2  module LFSR5 (
3    input clk,
4    output reg[4:0] LFSR = 31
5
6  );
7
8    wire feedback = LFSR[4];
9
10   always @(posedge clk)
11     begin
12       LFSR[1] <= LFSR[0];
13       LFSR[2] <= LFSR[1];
14       LFSR[3] <= LFSR[2];
15       LFSR[4] <= LFSR[3];
16       LFSR[0] <= (LFSR[2] ^ feedback);
17     end
18 endmodule
```

LFSR 5 bits com estado inicial em 11111

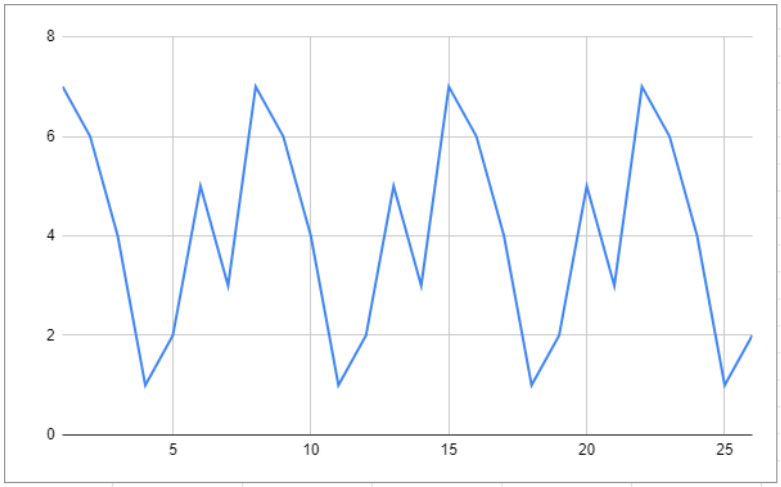
Dependências e tabela de ciclos:



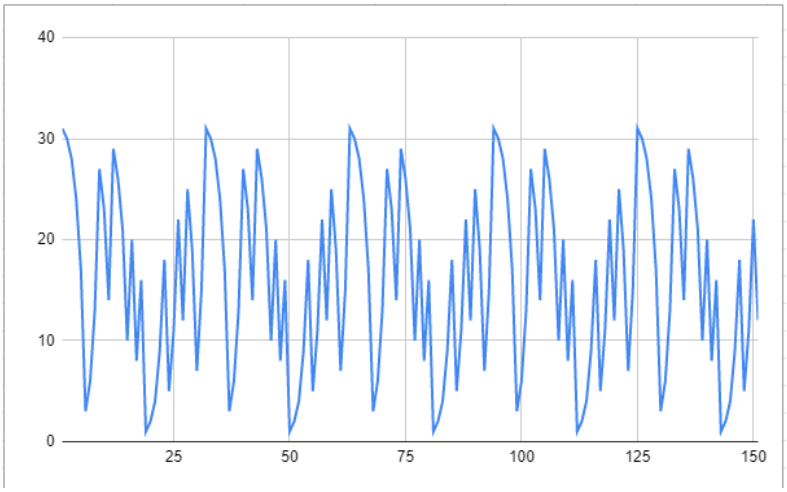


Representações decimais por ciclo:

N=3; $\mathbf{x^3+x^2+1}$



N=5; $\mathbf{x^5+x^3+1}$



Gráficos 2D:

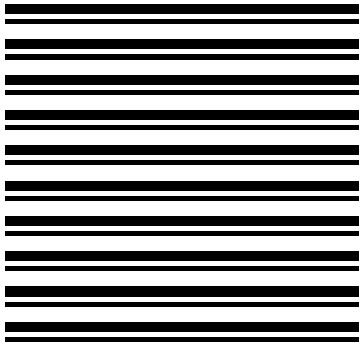


Gráfico 2d do LFSR de 3-Bits

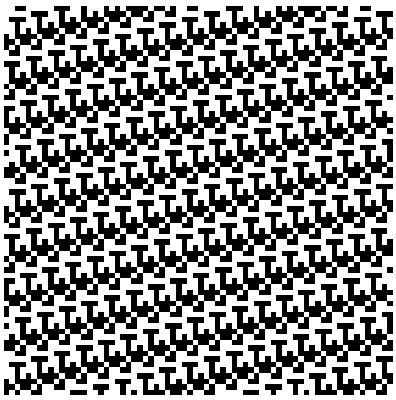


Gráfico 2d do LFSR de 5-Bits

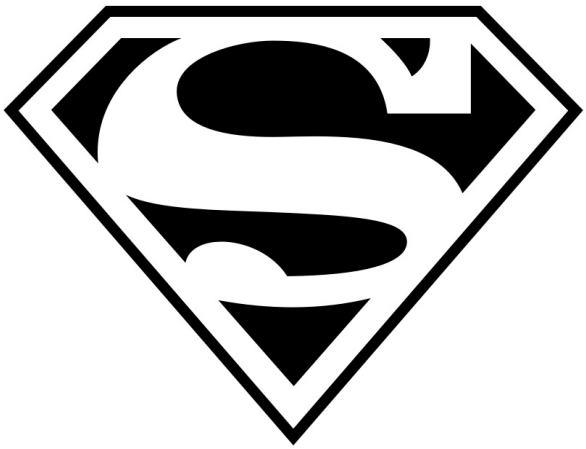
Link para os códigos no EDA playground:

- https://www.edaplayground.com/x/VNa9 LFSR 5 bits
- https://www.edaplayground.com/x/ZgpE LFSR 3 bits

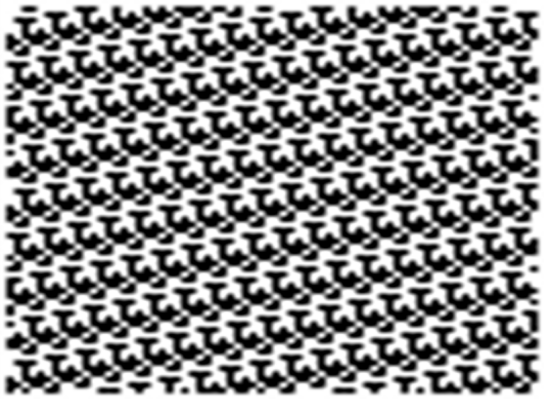
3) Foi feito a cifragem para as imagens de 3-Bits e de 5-Bits no link a seguir:

https://colab.research.google.com/drive/1Z7uNcyAwboNfR7J0OKEJz_hnZJjm1Aqh?usp=sharing

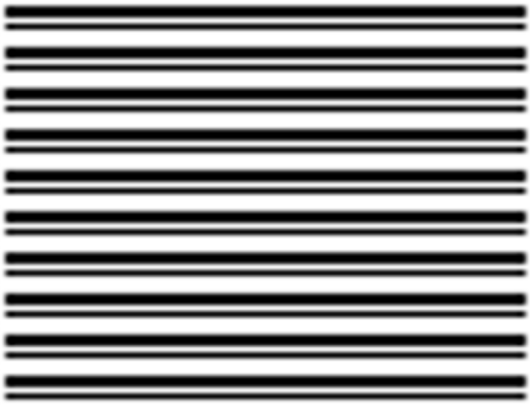
```
import numpy as np
from PIL import Image
import cv2

from google.colab.patches import cv2_imshow

IMAGEM_BITMAP = cv2.imread('IMAGEM_BITMAP.png')
CHAVE_3 = cv2.imread('CHAVE_LFRS_3.png')
CHAVE_5 = cv2.imread('CHAVE_LFRS_5.png')

cv2_imshow(IMAGEM_BITMAP)
cv2_imshow(CHAVE_3)
cv2_imshow(CHAVE_5)
```



IMAGEM_BITMAP(Original)



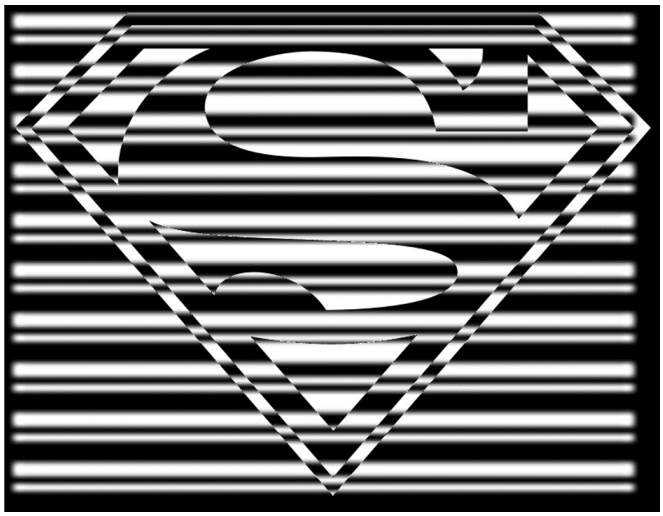
LFRS_5



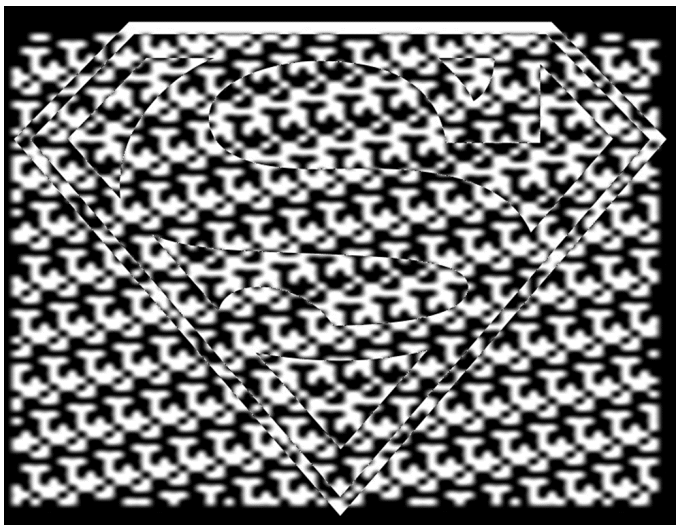
LFRS_3

Nessa parte do código foram impressas as imagens. Todas as imagens têm dimensão 825x626 e 24 intensidade de bits .

```
xor_3 = cv2.bitwise_xor(CHAVE_3, IMAGEM_BITMAP)
cv2_imshow(xor_3)

xor_5 = cv2.bitwise_xor(CHAVE_5, IMAGEM_BITMAP)
cv2_imshow(xor_5)
```



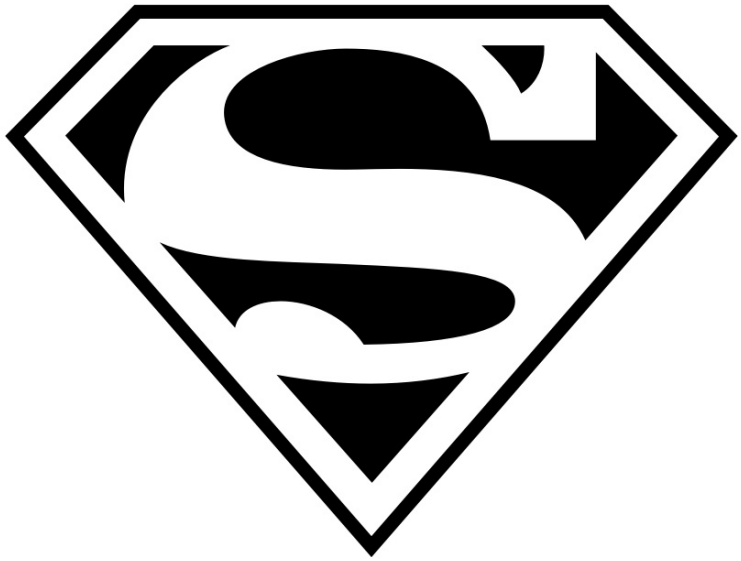
LRS_CHAVE_3 XOR IMAGEM_BITMAP



LRS_CHAVE_5 XOR IMAGEM_BITMAP

Já nessa parte do código foram geradas duas imagens cifradas, uma com o LFRS bit 3 XOR IMAGEM_BITMAP (original), a outra com LFRS bit 5 XOR IMAGEM_BITMAP (original).

```
xor_3_volta = cv2.bitwise_xor(xor_3, CHAVE_3)
cv2_imshow(xor_3_volta)

xor_5_volta = cv2.bitwise_xor(xor_5, CHAVE_5)
cv2_imshow(xor_5_volta)
```



XOR_3 XOR CHAVE_3 ou XOR_3 XOR CHAVE_3

Nessa última parte foi feito processo contrário. Foram geradas duas imagens iguais à imagem original. Foi feito o XOR das imagens cifradas com suas respectivas chaves LFRS.