UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Universidade Federal de Minas Gerais

Turma: TZ **Prof.:** Gilberto

Grupo: José Eduardo Duarte Massucato
Rubia Alice Moreira de Souza
Victor Yuji Yano

Relatório

Belo Horizonte
2023

1. INTRODUÇÃO

1.1. Contexto Histórico

A história dos algoritmos aplicados à segurança de dados remonta a séculos atrás, com o desenvolvimento de métodos criptográficos para proteger dados confidenciais. Exemplo disso foi o método da Cifra de César, utilizado pelo imperador romano Júlio César, baseado no deslocamento de caracteres.

Ao longo da história, novos algoritmos surgiram, suprindo as limitações dos métodos clássicos. Em 1917, o método Vernam Cipher (ou One-Time Pad) foi inventado pelo engenheiro norte-americano Gilbert Vernam, baseado no conceito de cifra de substituição, que funciona de maneira parecida com o método de deslocamento. No entanto, o método de Vernam foi além e introduziu um elemento-chave para garantir a segurança: o uso de uma chave secreta aleatória com o mesmo tamanho da mensagem.

Outro marco importante foi a invenção da máquina Enigma pelos nazistas durante a Segunda Guerra Mundial, uma máquina mecânica de criptografia para o envio de mensagens militares. Esse método de criptografia foi superado pelos Aliados, graças aos esforços e à invenção de computadores, com destaque à Alan Turing. Esse episódio demonstrou a necessidade de algoritmos mais robustos e sofisticados.

Assim, à medida que a computação e os computadores se desenvolveram, métodos criptográficos mais poderosos foram inventados. Surgiram então os algoritmos de chave assimétrica (ou criptografia de chave pública). Um dos mais conhecidos é o RSA (Rivest-Shamir-Adleman), que utiliza operações matemáticas com números primos para criptografar e descriptografar informações.

No contexto da década de 1960, foi inventado por Solomon W. Golomb, o algoritmo LFSR, um algoritmo baseado em registros de deslocamento para geração de sequências pseudo-aleatórias. Sua simplicidade e eficiência tornam-no uma escolha popular em várias áreas, como criptografia, comunicações digitais, geração de números aleatórios, testes de circuitos, etc.

O cenário atual apresenta desafios cada vez maiores, com ameaças cibernéticas sofisticadas e avanços na computação quântica. A segurança dos dados tornou-se uma preocupação central, levando a novas pesquisas e desenvolvimento de algoritmos criptográficos que possam resistir a essas ameaças emergentes.

1.2. Geração de Números Aleatórios

Diante da perspectiva histórica apresentada, tornou-se evidente em várias aplicações da computação a necessidade de gerar valores que sejam verdadeiramente aleatórios, ou seja, que não possam ser previstos com base em uma equação matemática fixa. Para atender a essa demanda, foram implementadas diversas estratégias que compartilham de uma técnica em comum: a detecção de pequenas e imprevisíveis mudanças em um conjunto de dados, que muitas vezes são de natureza física.

Um exemplo notável é a abordagem utilizada pelo RANDOM.ORG. Esse site captura variações na amplitude do ruído atmosférico para gerar seus números aleatórios. O ruído atmosférico é um fenômeno físico que ocorre naturalmente e é altamente imprevisível. A partir disso, o RANDOM.ORG assegura uma fonte de aleatoriedade confiável e não determinística. Outra estratégia comumente utilizada é baseada no decaimento radioativo.

Nesse caso, um dispositivo detecta os momentos em que uma fonte radioativa decai, um evento físico que é intrinsecamente imprevisível.

Essas abordagens físicas para a geração de números aleatórios têm a vantagem de fornecer uma fonte de aleatoriedade verdadeira, uma vez que são baseadas em fenômenos naturais imprevisíveis. Entretanto, essa concepção acaba sendo muito custosa em termos de recursos computacionais, o que justifica a não utilização generalizada de geração de valores verdadeiramente aleatórios na maioria dos algoritmos de criptografia amplamente utilizados na contemporaneidade.

1.3. Papel do Hardware

O hardware desempenha um papel fundamental nos processos de cifragem, permitindo a realização eficiente de operações lógicas sobre as mensagens a serem cifradas. Uma aplicação notável é o One-Time Pad (OTP), que requer uma chave gerada de forma verdadeiramente aleatória. Nesse método, é realizada a operação XOR entre a mensagem original e a chave para obter a mensagem cifrada. Para decifrar a mensagem, é necessário aplicar novamente a operação XOR entre a mensagem cifrada e a chave utilizada na cifragem, recuperando assim a mensagem original.

No contexto do hardware, a implementação de um cifrador OTP envolve a utilização de componentes dedicados, como os circuitos lógicos para realizar as operações XOR. É importante ressaltar que o OTP exige o armazenamento seguro e o gerenciamento adequado das chaves, uma vez que elas devem ser utilizadas apenas uma vez e mantidas sempre em sigilo. Além disso, a geração verdadeiramente aleatória das chaves é crucial para garantir a segurança do processo de cifragem.

No nosso trabalho, o OTP foi criado utilizando esta cifra gerada no site RANDOM.ORG: 01011110_01000011_00101111_01111100. Os resultados obtidos foram os seguintes:

Mensagem	**Binário**	**Texto**
Original	01100111011000010110110001101111	Galo
Cifrada	00111001001000100100001100010011	“9"Cnull”
Decifrada	01100111011000010110110001101111	Galo

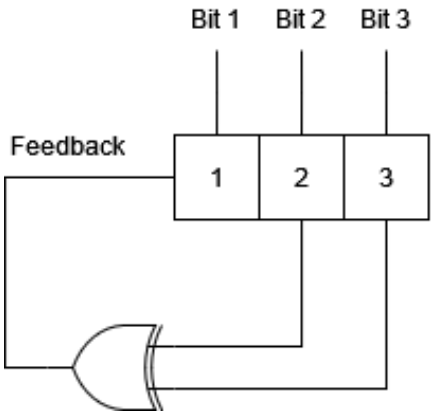
2. METODOLOGIA

Um LFSR (Linear-Feedback Shift Register) é um tipo de registrador de deslocamento que utiliza retroalimentação linear, através dos chamados “taps”, para gerar uma sequência de valores pseudo-aleatórios. Essa retroalimentação é realizada unindo certos bits do registrador por meio de uma porta lógica XOR, e o resultado é realimentado para a entrada do circuito. A escolha dos bits que irão retornar para o início do circuito é definida através dos polinômios de feedback, que são previamente escolhidos para garantir que o LFSR produza uma sequência com o maior ciclo de repetição possível, isto é, de comprimento igual a 2^n - 1, onde n é o número de bits total do circuito.

No contexto do nosso trabalho, implementamos LFSRs com 3, 4 e 8 bits, a fim de analisar os ciclos gerados.

2.1. LFSR de 3 bits

Aqui, o polinômio de feedback correspondente é o $X^3 + X^2 + 1$, ou seja, os “taps” são colocados no bit 2 e no bit 3. Diante disso, um ciclo de comprimento 7 é gerado, como demonstrado na imagem abaixo.



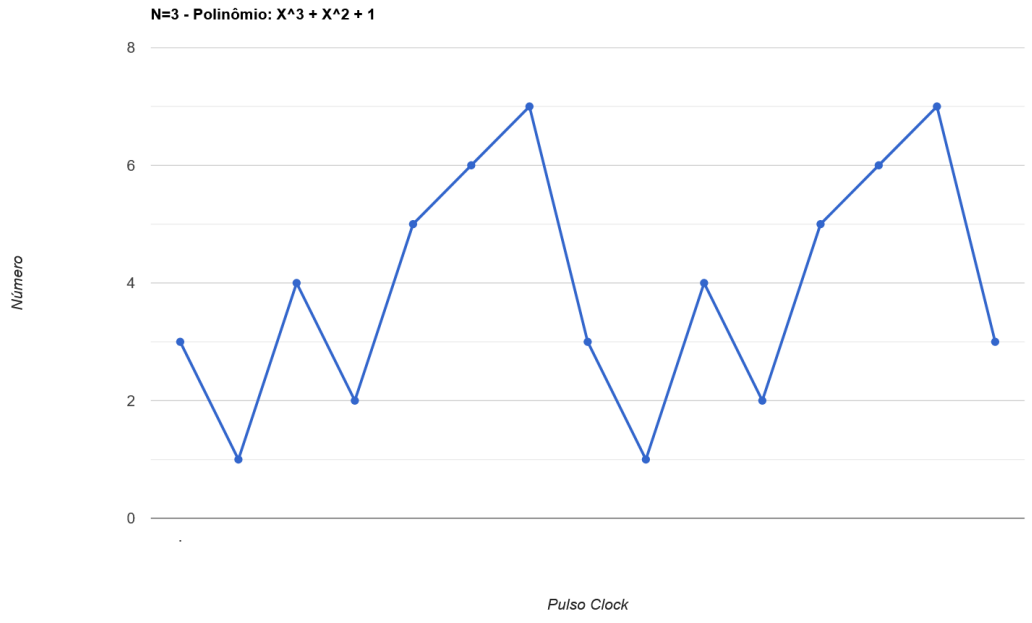
011
001
100
010
101
110
111
011
001
100
010

Ciclo com 7 iterações

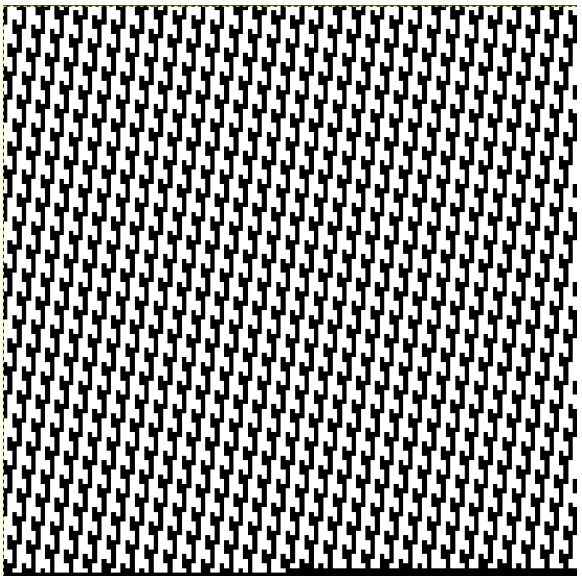
Neste caso, percebe-se que ele segue a seguinte sequência 011 → 001 → 100 → 010 → 101 → 110 → 111

Índice	1	2	3	4	5	6	7
Binário	011	001	100	010	101	110	111
Decimal	3	1	4	2	5	6	7

O gráfico abaixo representa dois ciclos gerados por este registrador, denotados a cada conjunto de 7 pontos.

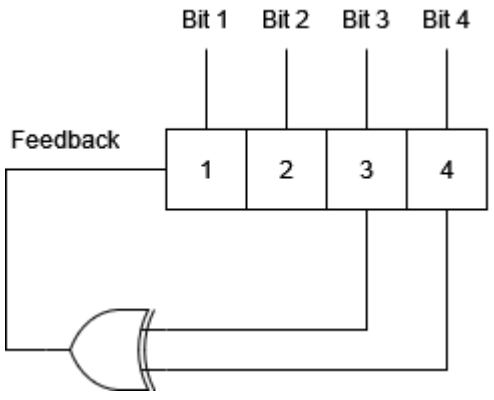


A imagem bitmap abaixo foi gerada para o LFSR de 3 bits, com 4900 iterações. Nela, é possível notar os ciclos de repetição dos valores advindos do registrador.



2.2. LFSR de 4 bits

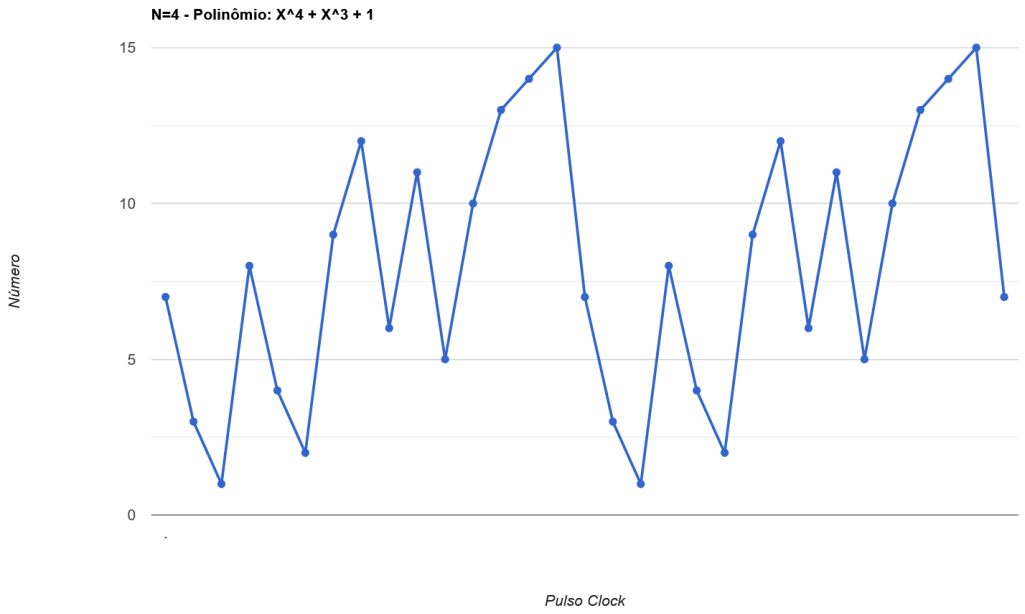
De maneira análoga, no caso do LFSR de 4 bits, o polinômio de feedback correspondente é o $X^4 + X^3 + 1$, ou seja, os “taps” são colocados no bit 3 e no bit 4. Assim, um ciclo de comprimento 15 é gerado, como demonstrado na imagem abaixo.



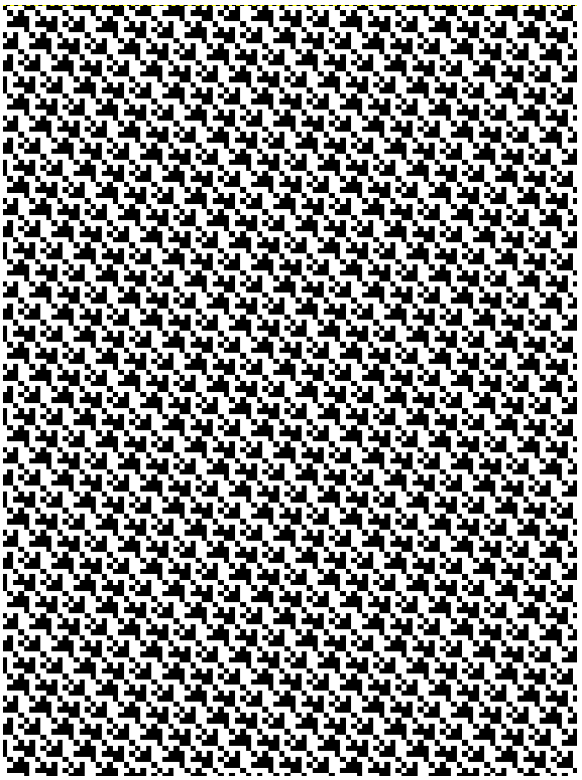
0111	
0011	1010
0001	1101
1000	1110
0100	1111
0010	0111
1001	0011
1100	0001
0110	1000
1011	0100
0101	0010

Ciclo com 15 valores diferentes

O gráfico abaixo representa dois ciclos gerados por este registrador, denotados a cada conjunto de 15 pontos.

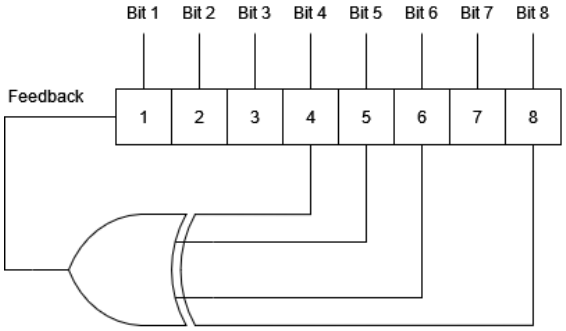


A imagem bitmap abaixo foi gerada para o LFSR de 4 bits, após 4900 iterações. Nela, também é possível notar claramente os ciclos de repetição dos valores produzidos pelo registrador.



2.3. LFSR de 8 bits

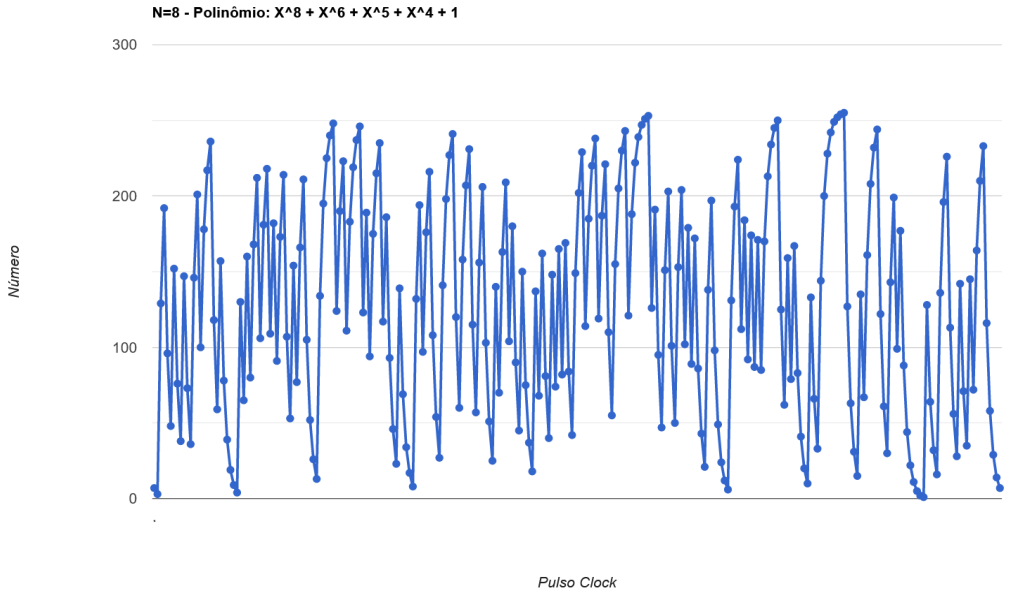
Por fim, no caso do LFSR de 8 bits, cujo polinômio de feedback correspondente é o $X^8 + X^6 + X^5 + X^4 + 1$, temos que os “taps” são colocados no bit 4, 5, 6 e no 8. Assim, um ciclo de comprimento 255 é gerado, como demonstrado na imagem abaixo.



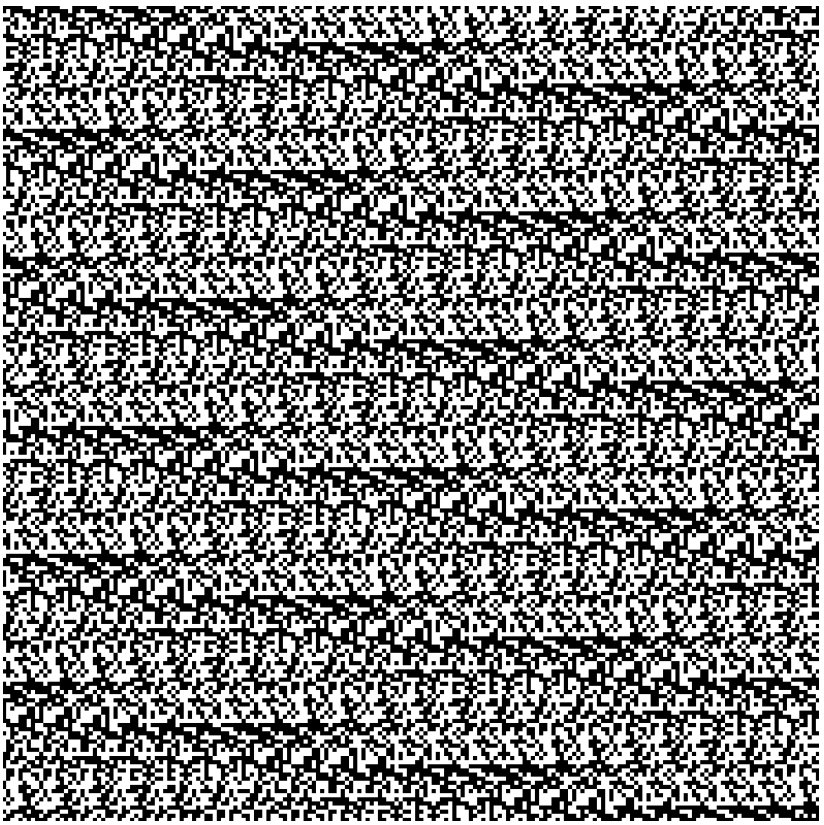
00000111	…
00000011	01110001
10000001	00111000
11000000	00011100
01100000	10001110
00110000	01000111
10011000	00100011
01001100	10010001
00100110	01001000
10010011	10100100
01001001	11010010
00100100	11101001
10010010	01110100
10001000	00111010
11000100	00011101
11100010	00001110
…	00000111

Ciclo com 255 valores diferentes

O gráfico abaixo representa um único ciclo gerado por este registrador, denotado a cada conjunto de 255 pontos.



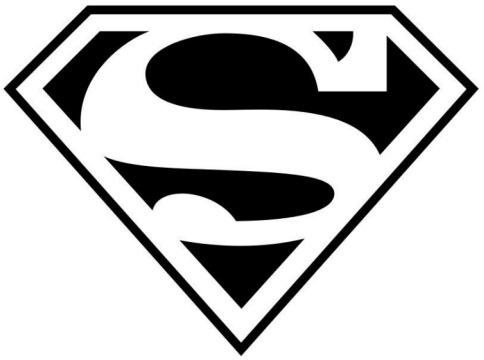
A imagem bitmap abaixo foi gerada para o LFSR de 8 bits, após 4900 iterações. Nela, também é possível observar a formação de padrões na saída do registrador, embora seja menos evidente em comparação com as imagens anteriores.



Observação: todas as imagens bitmap foram geradas utilizando o seguinte código em C++: Basico Verilog. Ela foi criada em um arquivo PGM (portable graymap format) com 39.200 pixels.

3. RESULTADOS OBTIDOS

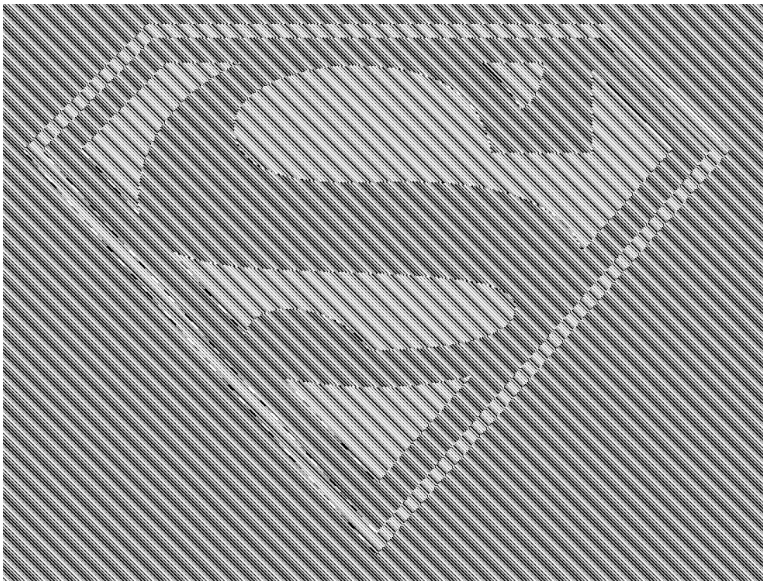
A imagem original a ser cifrada foi a disponibilizada no enunciado do Trabalho Prático:



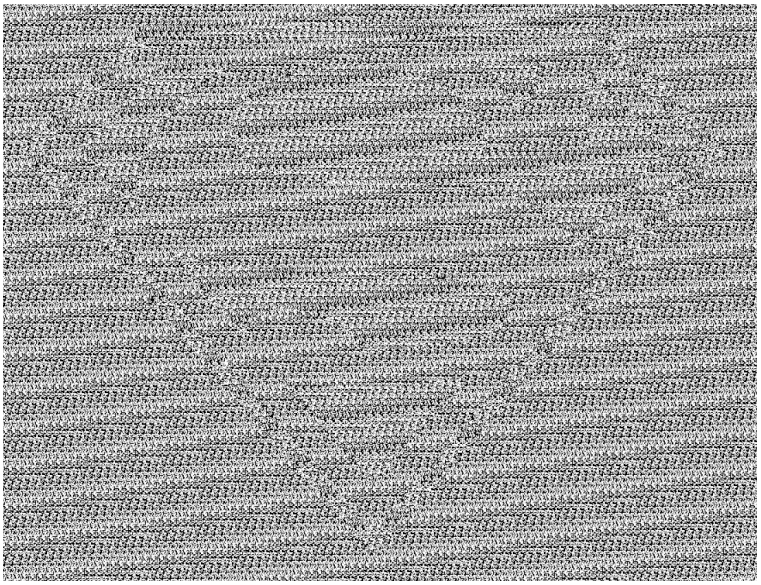
Inicialmente, a imagem foi convertida para o formato PGM, onde os valores da imagem que fossem maiores que 200 eram marcados como 1 e os demais como 0, para garantir que a imagem fosse totalmente preta e branca. Após esse processo, o programa cifra a imagem aplicando um XOR entre os bits da imagem original e os valores pseudo-aleatórios gerados pelo LFSR explicitado anteriormente. Para que fosse necessário

armazenar apenas uma vez a sequência gerada pelo LFSR, a leitura do polinômio foi feita de forma circular.

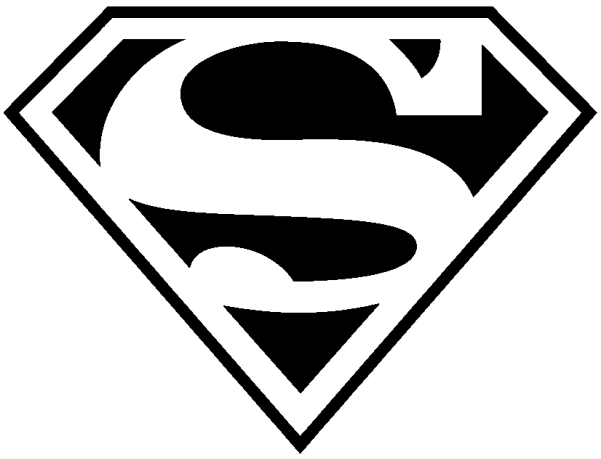
Após utilizar a sequência pseudo-aleatória de 3 bits do LFSR, a imagem cifrada obtida foi:



Nela, ainda é possível perceber o símbolo da imagem original, muito pelo fato de o ciclo de repetição ser muito pequeno, ou seja, o padrão torna-se muito rapidamente reconhecível. Por outro lado, após utilizar a sequência pseudo-aleatória de 8 bits do LFSR, a imagem cifrada obtida foi:



Nela, o símbolo da imagem original já é praticamente irreconhecível. Ao decifrar a imagem, obtivemos o seguinte resultado:



Observação: Todos os códigos para manipulação de imagens estão disponíveis no link: Rubia Souza - BasicoVerilog/Manipulacao-Imagens.

4. CONCLUSÕES

Durante o curso, tivemos a oportunidade de estudar diversos tópicos, desde CMOS, circuitos combinacionais e sequenciais, até registradores e contadores. Essa abordagem nos permitiu compreender tanto os aspectos mais internos quanto os mais externos do hardware. Esse processo foi fundamental para uma compreensão completa do curso, pois pudemos estabelecer conexões entre os conceitos que estávamos aprendendo e o conhecimento prévio adquirido.

No escopo do nosso trabalho, aplicamos vários temas que aprendemos ao longo do curso. Em particular, utilizamos o conceito de registradores de deslocamento e sua implementação na linguagem Verilog. Essa aplicação foi extremamente valiosa, pois nos permitiu explorar e aprofundar nosso conhecimento de forma prática.

Nosso grupo concorda que essa aplicação prática foi uma experiência enriquecedora e altamente relevante para a nossa formação. Foi gratificante ver como o conhecimento adquirido durante a disciplina pôde ser aplicado em um projeto real. Isso fortaleceu nossa compreensão dos conceitos estudados e nos proporcionou uma visão mais abrangente sobre a utilidade e as aplicações desses conceitos no mundo real.

5. REFERÊNCIAS

AMD. Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators. Disponível em: <https://docs.xilinx.com/v/u/en-US/xapp052>. Acesso em: 07/06/2023.

Gilberto Medeiros Ribeiro. Slides virtuais da disciplina de Introdução aos Sistemas Lógicos. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

Khan Academy. A cifra de César. Disponível em: <https://pt.khanacademy.org/computing/computer-science/cryptography/crypt/v/caesar-cipher>. Acesso em: 12/06/2023

Khan Academy. O que é criptografia?. Disponível em: <https://pt.khanacademy.org/computing/computer-science/cryptography/crypt/v/intro-to-cryptography>. Acesso em: 12/06/2023.

Random.org. Randomness. Disponível em: <https://www.random.org/randomness/>. Acesso em: 12/06/2023.

Wikipedia. Gilbert Vernam. Disponível em: <https://en.wikipedia.org/wiki/Gilbert_Vernam>. Acesso em: 14/06/2023.

Wikipedia. One Time Pad. Disponível em: <https://pt.wikipedia.org/wiki/One-time_pad>. Acesso em: 14/06/2023.

Wikipedia. Linear Feedback Shift Register. Disponível em: <https://en.wikipedia.org/wiki/Linear-feedback_shift_register>. Acesso em: 14/06/2023.

Anexos Atividades

---

1. Atividade: Gerar uma OTP

Implementação em código Verilog disponível em:

- Código EDA Workbench: One Time Pad
- Simulação Digital JS: Digital JS - Simulação One Time Pad

O One Time Pad foi criado utilizando esta cifra gerada no site Random.org: 01011110_01000011_00101111_01111100.

Os resultados obtidos foram os seguintes:

Mensagem	**Binário**	**Texto**
Original	01100111011000010110110001101111	Galo
Cifrada	00111001001000100100001100010011	“9"Cnull”
Decifrada	01100111011000010110110001101111	Galo

2. Atividade: Geração de sequência pseudo-aleatória

2.1. Parte 1: Código Verilog

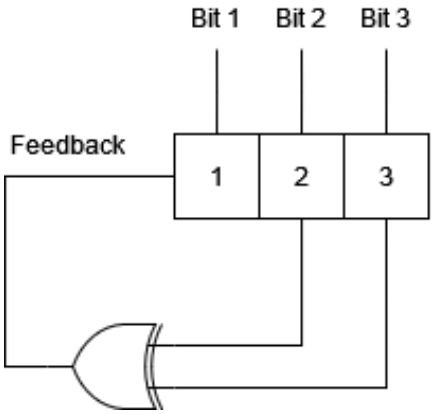
Implementação do código Verilog disponíveis em:

Quantidade Bits LFSR	Simulações	
	EDA Workbanch	Digital JS
3bits	Linear Feedback Shift Register - 3 bits	Digital JS - Simulação LFSR 3 Bits
4bits	Linear Feedback Shift Register - 4 bits	Digital JS - Simulação LFSR 4 Bits
8bits	Linear Feedback Shift Register - 8 bits	Digital JS - Simulação LFSR 8 Bits

2.2. Parte 2: Gráficos dos Períodos de Repetição das Sequências

2.2.1. Polinômio: $X^3 + X^2 + 1$

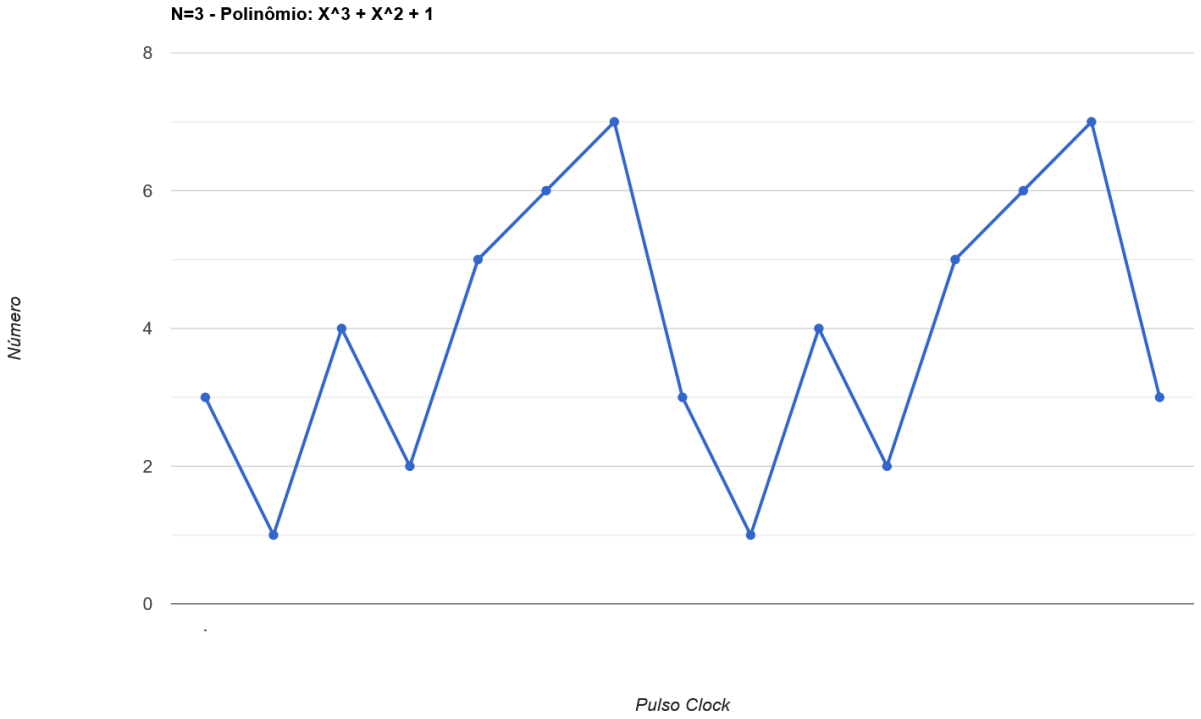
Para o polinômio X^3+X^2+1, temos o circuito da seguinte forma:



011
001
100
010
101
110
111
011
001
100
010

Ciclo com 7 iterações

Nele, é possível notar um ciclo a cada 7 valores (pontos), no gráfico abaixo:



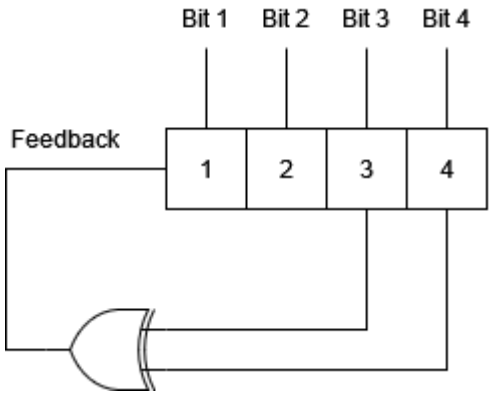
Neste caso, ele segue a seguinte sequência 3 → 1 → 4 → 2 → 5 → 6 → 7

Índice	1	2	3	4	5	6	7

Binário	011	001	100	010	101	110	111
Decimal	3	1	4	2	5	6	7

2.2.2. Polinômio: $X^4 + X^3 + 1$

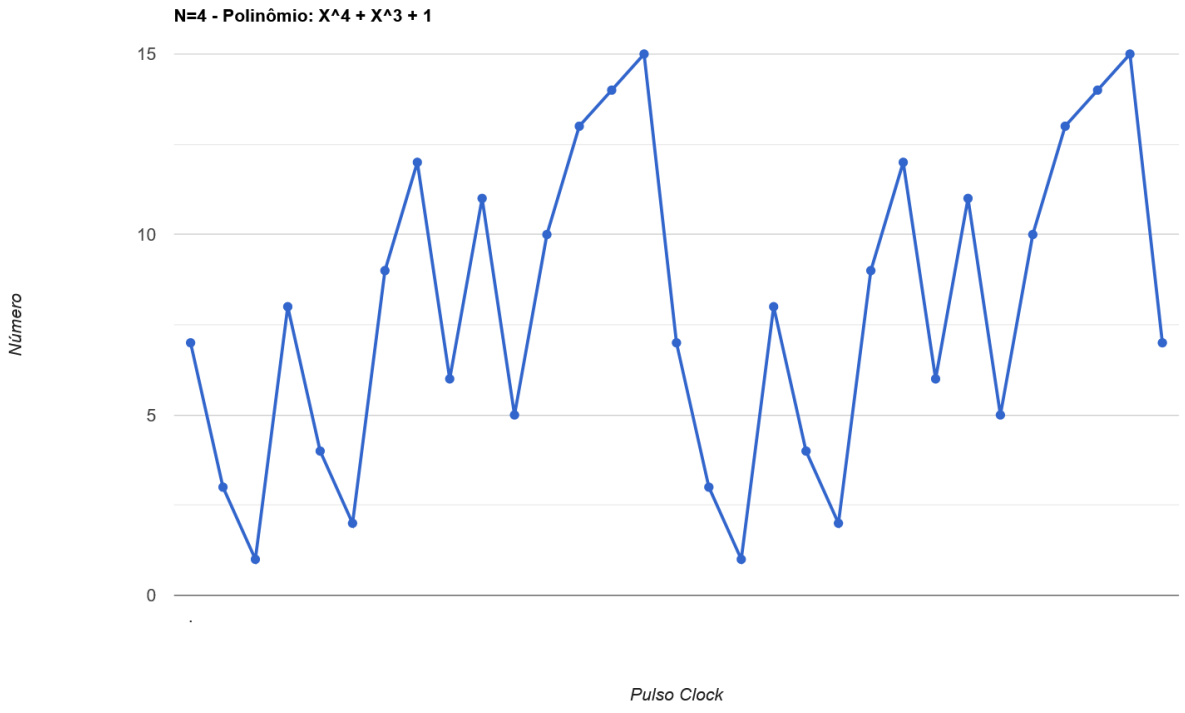
Já para o polinômio $X^4 + X^3 + 1$, temos:



0111	
0011	1010
0001	1101
1000	1110
0100	1111
0010	0111
1001	0011
1100	0001
0110	1000
1011	0100
0101	0010

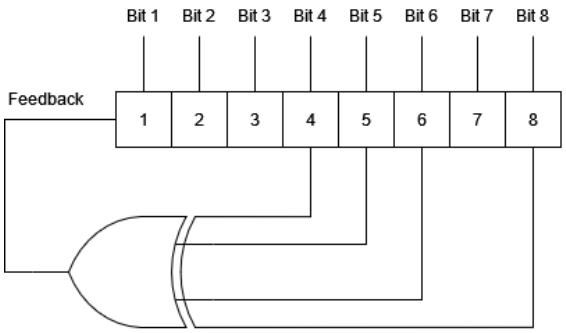
Ciclo com 15 valores diferentes

Ele segue uma sequência com 15 valores:



2.2.3. Polinômio: $X^8 + X^6 + X^5 + X^4 + 1$

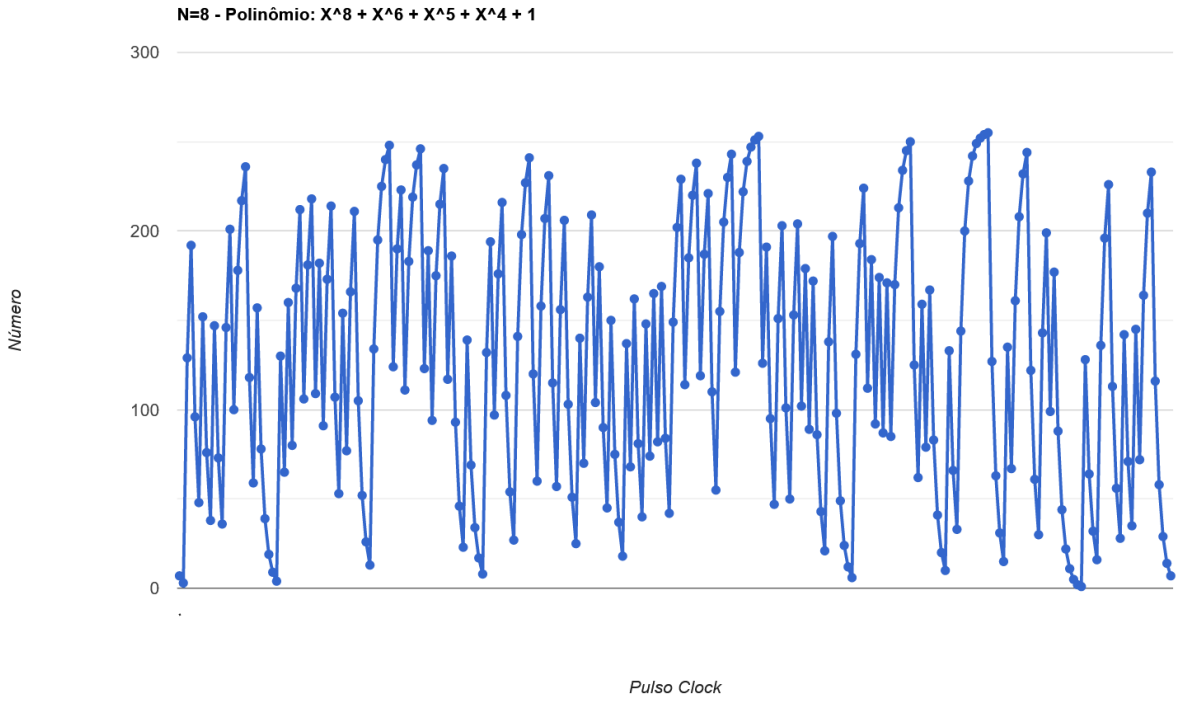
Por fim, temos o polinômio $X^8 + X^6 + X^5 + X^4 + 1$:



00000111	…
00000011	01110001
10000001	00111000
11000000	00011100
01100000	10001110
00110000	01000111
10011000	00100011
01001100	10010001
00100110	01001000
10010011	10100100
01001001	11010010
00100100	11101001
10010010	01110100
10001000	00111010
11000100	00011101
11100010	00001110
…	00000111

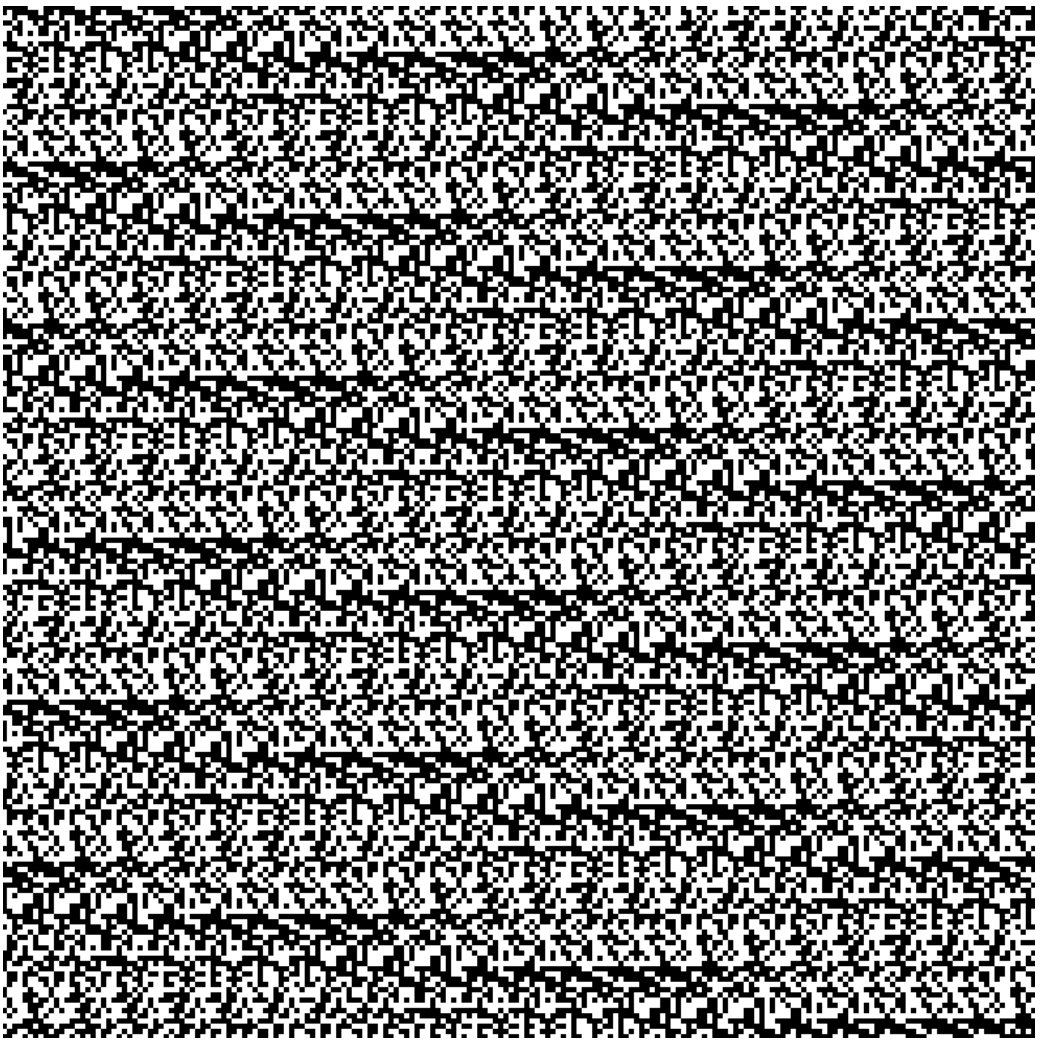
Ciclo com 255 valores diferentes

Ele possui um ciclo de 255 valores. Neste gráfico, eu representei um ciclo apenas do polinômio, para que seja visível a variação nele:



2.3. Imagem Gerada

A imagem foi gerada utilizando o seguinte código em C++: Basico Verilog. Ela foi criada em um arquivo PGM (portable graymap format) com 39.200 pixels. Ou seja, possui 4900 iterações do LFSR com 8 bits.



Também geramos uma imagem de um LFSR com 4 bits e 3 bits. Ambos com 4900 iterações:

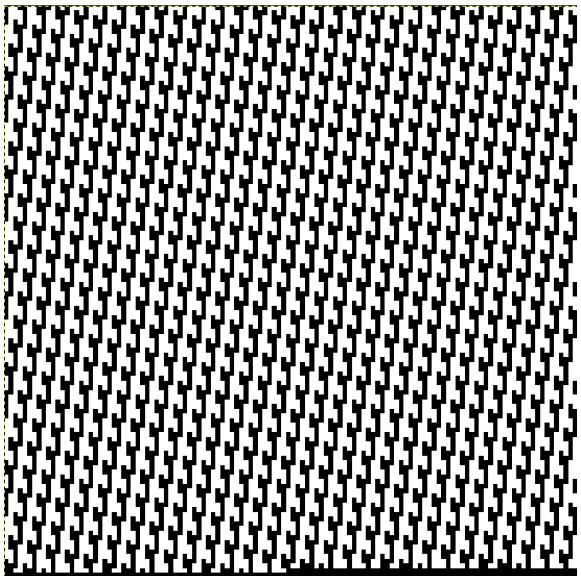


Imagem gerada a partir de um LFSR com 3 bits

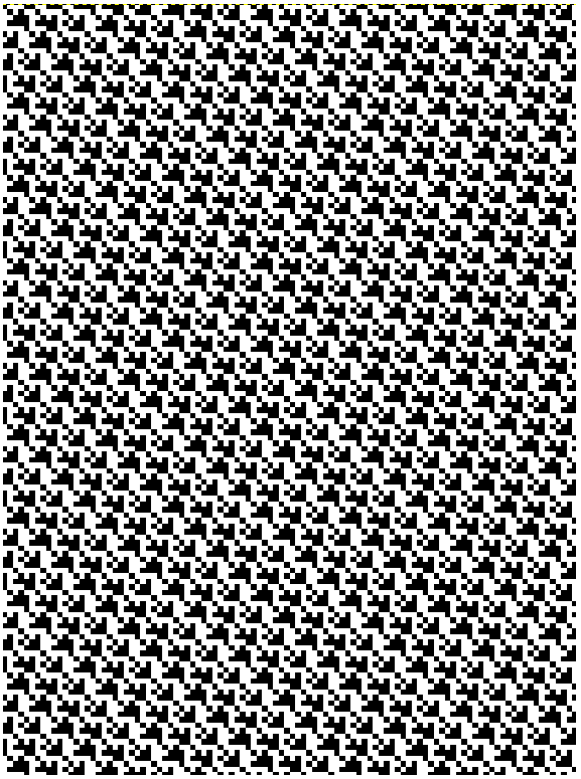


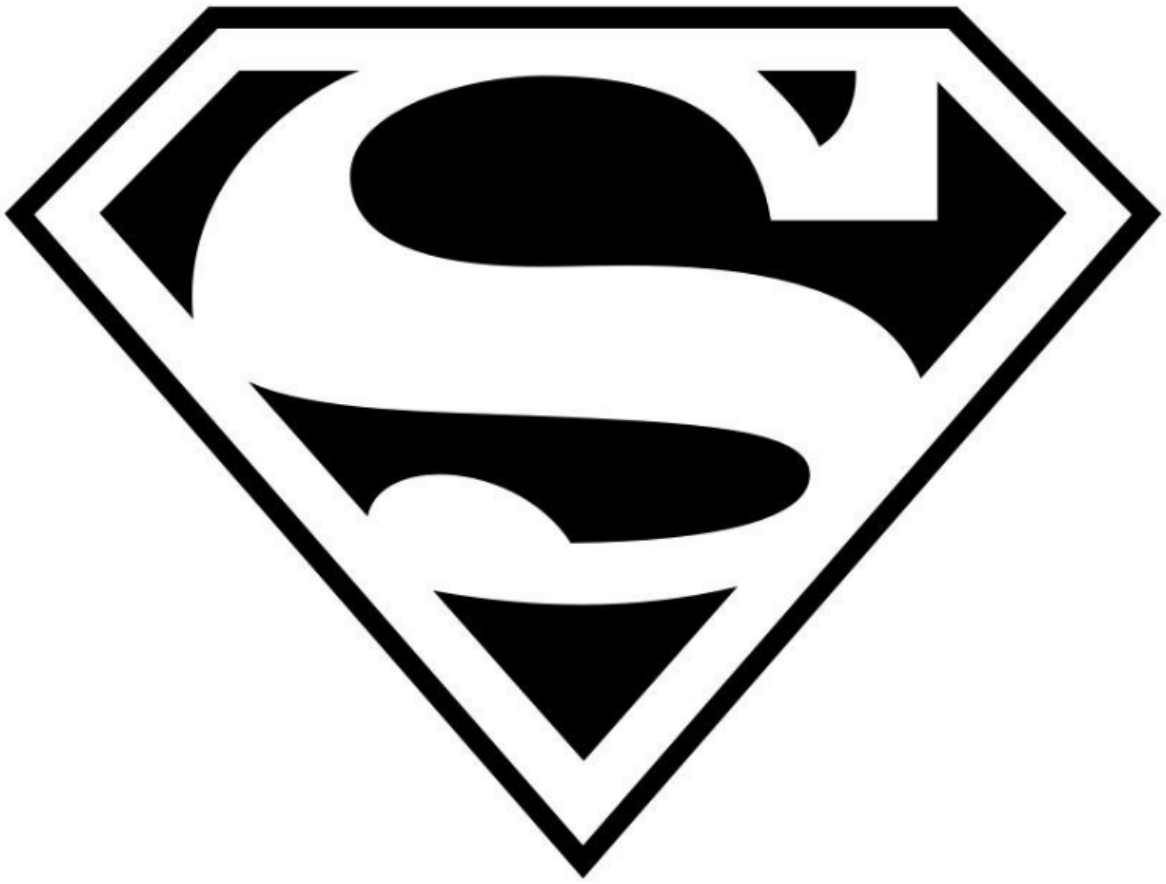
Imagem gerada a partir de um LFSR com 4 bits

2.4. Referências

Referência utilizada para gerar os polinômios com quantidade máxima de valores por ciclo foi a “Tabela 3 - Taps for Maximum-Length LFSR Counters”. Disponível neste link: <https://docs.xilinx.com/v/u/en-US/xapp052>.

3. Atividade: Cifragem de Imagens

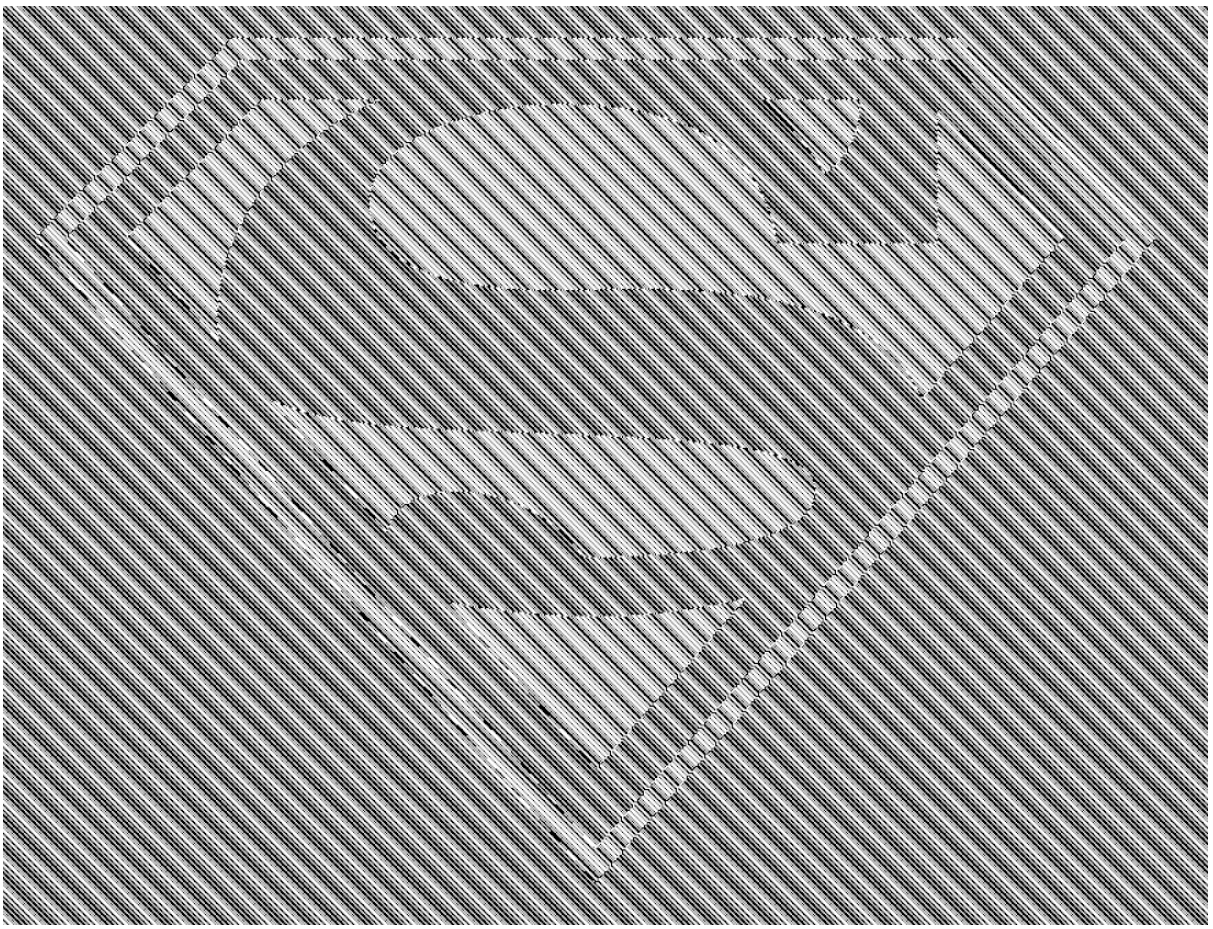
A imagem original que será cifrada, foi aquela fornecida no enunciado do trabalho:



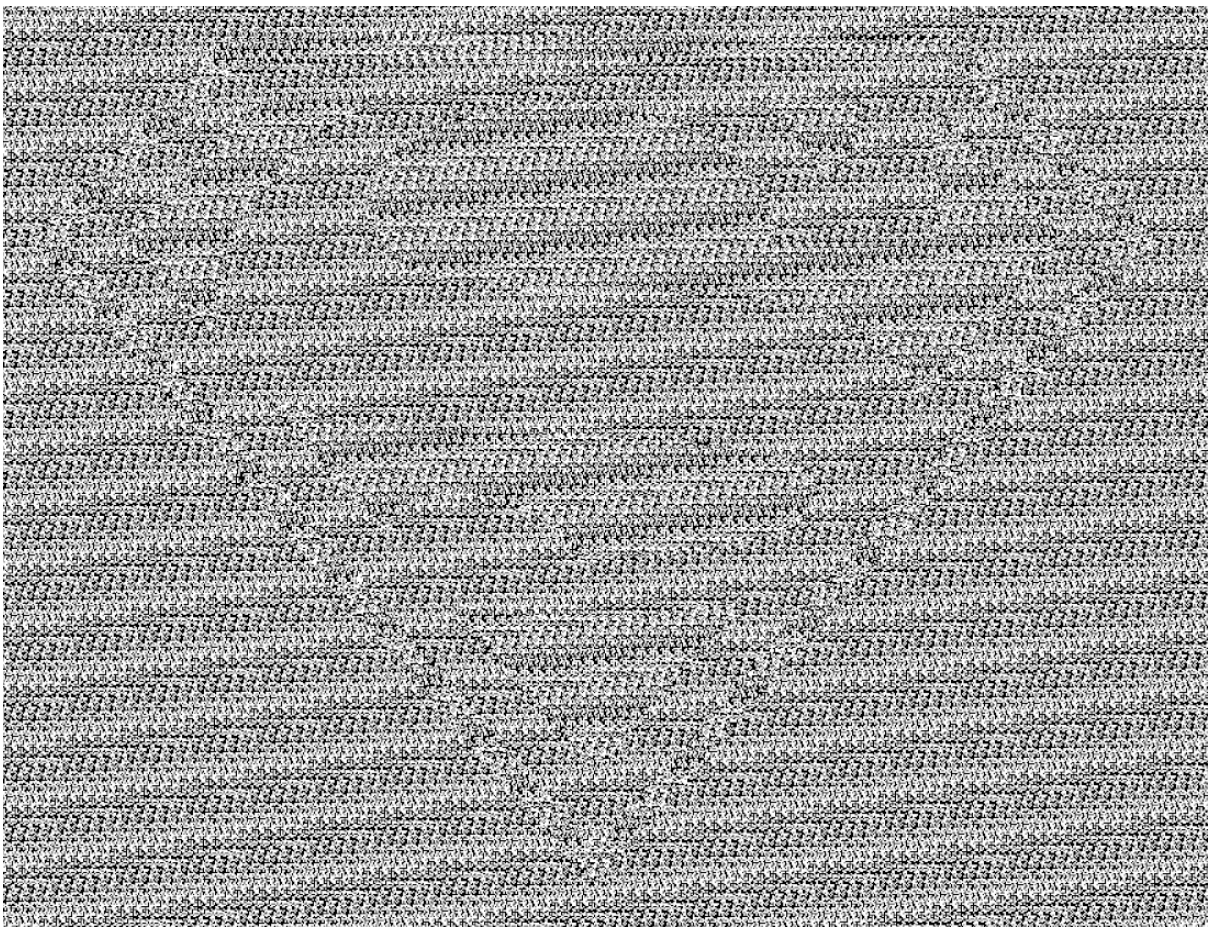
Inicialmente, criei um programa em C++ para convertê-la para PGM e junto com um arquivo binário, em que os valores maiores que 200 eram marcados como 1. Já os outros como 0.

Então, fiz um programa que utiliza uma cifra dos polinômios pseudo-aleatórios da questão anterior e aplicar um XOR aos bits da imagem original. A leitura do polinômio foi feita de forma circular para que fosse necessário armazenar apenas uma sequência.

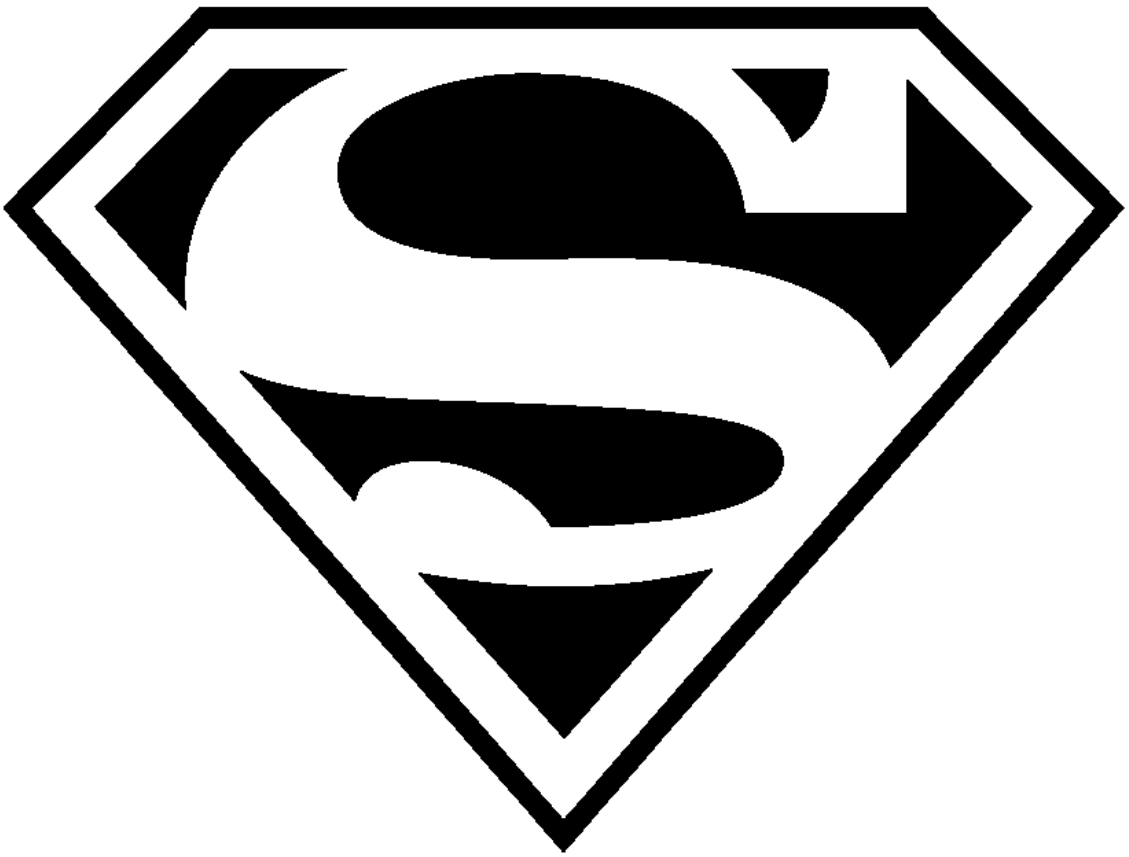
Utilizando a sequência pseudo aleatória de 3 bits do LFSR, obtive esta imagem ao cifrar a original:



Ainda é possível notar o símbolo original nela. No entanto, ao utilizar a cifra de 8 bits, o padrão fica menos reconhecível na imagem, que ficou da seguinte forma:



Ao reverter a imagem para o tipo original, obtivemos o seguinte resultado:



Todos os códigos para manipulação de imagens estão disponíveis no link: [Rubia Souza - BasicoVerilog/Manipulacao-Imagens]().