Projeto de curso – Valor: 20 pontos

ISL: Introdução a Sistemas Lógicos
Atividade em Verilog: registradores de deslocamento com feedback linear LFSR (linear feedback shift registers).

Introdução

A cifragem de mensagens pode ser realizada no modo stream-cypher. Já fizemos um trabalho assim no primeiro trabalho. A este cifrador se dá o nome de Vernam Cypher, onde Gilbert Vernam do AT&T Bell Labs inventou em 1917.

Claude Shannon, também da Bell Labs, provou que o one time pad, devidamente implementado, é inquebrável em sua pesquisa realizada durante a Segunda Guerra Mundial, publicada posteriormente em outubro de 1949. Ele também provou que qualquer sistema inquebrável deve ter essencialmente as mesmas características do one-time pad: a chave deve ser verdadeiramente aleatória, tão grande quanto o texto simples, nunca reutilizada no todo ou em parte e mantida em segredo.

Em sua forma original, o sistema de Vernam era vulnerável porque a fita principal era um loop, que era reutilizado sempre que o loop fazia um ciclo completo.

Atividade 1:

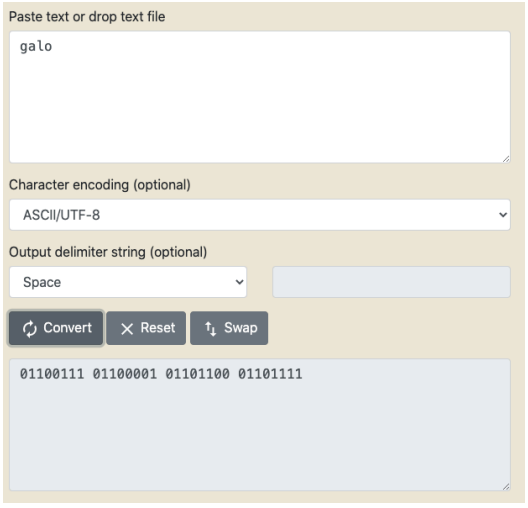
1) Gerar uma OTP: One Time Pad – uma sequência de 32 bits verdadeiramente aleatórios para a geração da chave. Vide, por exemplo: https://www.random.org/randomness/ (excelente revisão, e lá voces encontrarão as diferenças entre pseudo aleatório, verdadeiramente aleatório, as vantagens de cada um).
   Por exemplo, entre no link: https://www.random.org/bytes/

selecione : Get bytes, e:

```
11100010 11011001 11101001 01010001
```

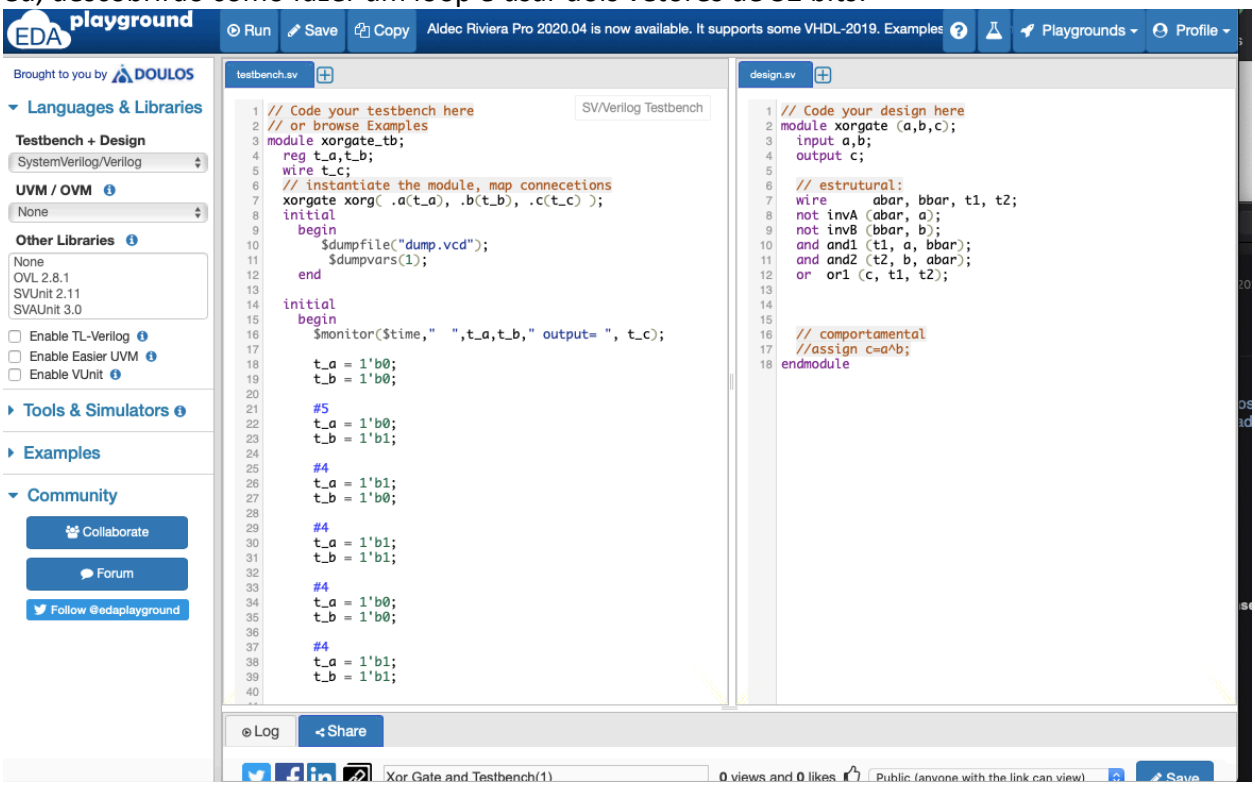
Acima é o seu OTP.

2) Agora cifre uma mensagem, bit a bit. Use, por exemplo, o site (ou qualquer outro que lhe for conveniente): https://www.rapidtables.com/convert/number/ascii-to-binary.html



3) Pronto, a mensagem que voce quer cifrar é: 01100111 01100001 01101100 01101111
4) Crie em seu testbench a sequência de números aleatórios (ie., seu OTP) com o qual voce fará o XOR com a mensagem acima.

5) Gere a mensagem cifrada, como saída do EDAPlayground. Exemplo abaixo, observe as entradas a e b, para cada instante de tempo (que voce sincroniza com o #5, se for 5 ns, ou seja, #<tempo>). Entao, teremos 32 linhas, ou se forem mais espertos que eu, descobrirão como fazer um loop e usar dois vetores de 32 bits:



6) Converta a mensagem para ASCII de volta, e veja como ela ficou cifrada.
7) Faça o caminho inverso e decifre a mensagem.

Linear Feedback Shift Register - LFSR: (aula)

Em computação, um registrador de deslocamento de feedback linear (LFSR) é um registrador de deslocamento cujo bit de entrada é uma função linear de seu estado anterior.

A função linear mais comumente usada de bits únicos é a ou exclusiva (XOR). Assim, um LFSR é mais freqüentemente um registrador de deslocamento cujo bit de entrada é dirigido pelo XOR de alguns bits do valor geral do registrador de deslocamento.

O valor inicial do LFSR é chamado de **seed** (ou semente), e como a operação do registrador é determinística, o fluxo de valores produzido pelo registrador é completamente determinado por seu estado atual (ou anterior). Da mesma forma, como o registrador tem um número finito de estados possíveis, ele deve eventualmente entrar em um ciclo de repetição. No entanto, um LFSR com uma função de feedback bem escolhida pode produzir uma sequência de bits que parece aleatória e tem um ciclo muito longo.

As aplicações dos LFSRs incluem a geração de números pseudo-aleatórios, sequências de pseudo-ruído, contadores digitais rápidos e sequências de clareamento. As implementações de hardware e software de LFSRs são comuns.

Nesta terceira parte de nosso trabalho, iremos implementar um LFSR para geração de números pseudo-aleatórios para a contrução de um “stream-cypher”, onde a mensagem é cifrada bit a bit com o LFSR. Para a implementação do LFSR, utilizaremos um LFSR Fibonacci.

LFSR Fibonacci (adaptado de Wikipedia, LFSR)

As posições de bit que afetam o próximo estado são chamadas de taps. No diagrama na fig.1, as derivações são [16,14,13,11]. O bit mais à direita do LFSR é chamado de bit de saída. Os taps são submetidos a XOR sequencialmente com o bit de saída e, em seguida, realimentados com o bit mais à esquerda. A sequência de bits na posição mais à direita é chamada de fluxo de saída. Este será o ponto de entrada para o stream cypher, em uma das entradas de uma porta XOR responsável pelo processo de cifra bit a bit.

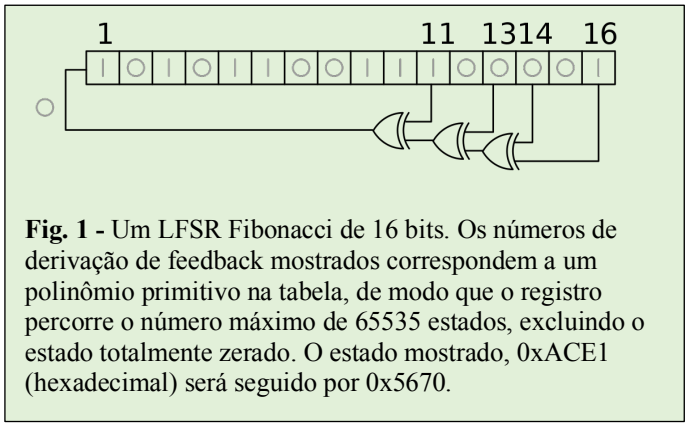


Fig. 1 - Um LFSR Fibonacci de 16 bits. Os números de derivação de feedback mostrados correspondem a um polinômio primitivo na tabela, de modo que o registro percorre o número máximo de 65535 estados, excluindo o estado totalmente zerado. O estado mostrado, 0xACE1 (hexadecimal) será seguido por 0x5670.

Os bits no estado LFSR que influenciam a entrada são chamados de taps.
Um LFSR de comprimento máximo produz uma sequência m (ou seja, ele percorre todos os estados 2m - 1 possíveis dentro do registrador de deslocamento, exceto o estado onde todos os bits são zero), a menos que contenha todos os zeros, caso em que nunca mudará.
Este estado é considerado ilegal porque o contador permaneceria "travado" neste estado. Este método pode ser vantajoso em LFSRs de hardware usando flip-flops que começam em um estado zero, já que não inicia em um estado de travamento, o que significa que o registrador não precisa ser propagado para iniciar a operação. A sequência de números gerada por um LFSR pode ser considerada um sistema numeral binário tão válido quanto o código Gray ou o código binário natural.

O arranjo de derivações para feedback em um LFSR pode ser expresso em aritmética de campo finito como um polinômio mod 2. Isso significa que os coeficientes do polinômio devem ser 1s ou 0s. Isso é chamado de polinômio de feedback ou polinômio de característica recíproca. Por exemplo, se os taps estão no 16º, 14º, 13º e 11º bits (como mostrado), o polinômio de feedback é

$$x^{16}+x^{14}+x^{13}+x^{11}+1$$

O "um" no polinômio não corresponde a um tap - ele corresponde à entrada para o primeiro bit (ou seja, x^0, que é equivalente a 1). As potências dos termos representam os bits oriundos das derivações, contando a partir da esquerda. O primeiro e o último bits são sempre conectados como um tap de entrada e saída, respectivamente.

Pontos chave sobre LFSR a serem observados:

- Um registrador de deslocamento com feedback linear é um registrador de deslocamento com lógica combinacional que faz com que ele percorra pseudo-aleatoriamente uma sequência de valores binários.
    - É "aleatório" no sentido de que o valor de um elemento da sequência é independente dos valores de qualquer um dos outros elementos.
    - É 'pseudo' porque é determinístico e após N elementos que começa a se repetir, ao contrário das sequências reais aleatórias
    - ... Se alguém souber o "estado atual", bem como as posições das portas XOR no LFSR, pode-se prever o “próximo estado". Isso não é possível com eventos verdadeiramente aleatórios.
    - … O fluxo de saída é reversível, um LFSR com taps espelhado percorrerão a sequência de saída na ordem inversa
- O feedback sobre o registrador de deslocamento de um LFSR vem de um seleção de pontos (taps) na cadeia de FlipFlops do registrador e fazendo XOR destes taps de volta para o registrador.
- Bits do registrador bits que não precisam de um tap de entrada, operam como um registrador de deslocamento padrão. É esse feedback que causa o registrador percorrer sequências repetitivas de valores pseudo-aleatório.

Questões práticas para a construção de LFSRs:

- A escolha dos taps determina quantos valores existem estão em uma determinada sequência antes que a sequência se repita.
- Um LFSR irá produzir uma sequência pseudo-aleatória de comprimento (2n - 1) estados (onde n é o número de estágios) se o LFSR é de comprimento máximo (ou seja, todos os valores possíveis (2n) menos o “estado zero”).
- A sequência será então repetida a partir do estado inicial para quando contanto que o LFSR seja cronometrado.
- Um LFSR é de comprimento 'máximo' quando a sequência gera passagens por todos os valores 2n-1 possíveis.
- Pode haver mais de uma combinação de taps que dão comprimento máximo para cada LFSR.

IMPORTANTE !

- A sequência LFSR depende do valor da semente, das posições de tap e o tipo de feedback.

Polinômios:

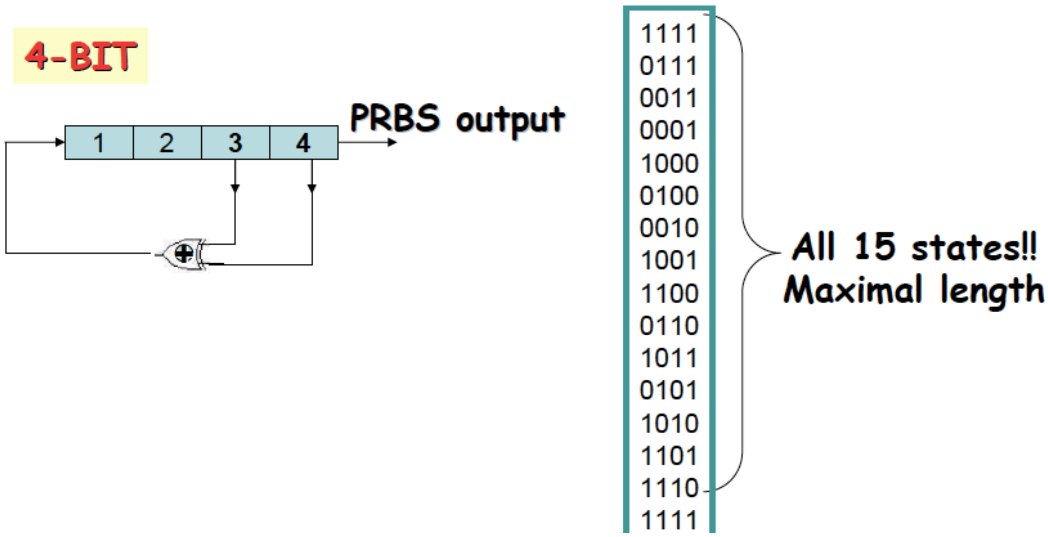
Na figura 2 ao lado uma lista de polinômios que podem ser usados para se gerar uma sequência pseudo-aleatória de comprimento máximo (isto é, 2^n-1, onde n é o número de flip-flops).

Bits *n*	Feedback polynomial	Period $2^n - 1$
2	$x^2 + x + 1$	3
3	$x^3 + x^2 + 1$	7
4	$x^4 + x^3 + 1$	15
5	$x^5 + x^3 + 1$	31
6	$x^6 + x^5 + 1$	63
7	$x^7 + x^6 + 1$	127
8	$x^8 + x^6 + x^5 + x^4 + 1$	255
9	$x^9 + x^5 + 1$	511
10	$x^{10} + x^7 + 1$	1023
11	$x^{11} + x^9 + 1$	2047
12	$x^{12} + x^{11} + x^{10} + x^4 + 1$	4095
13	$x^{13} + x^{12} + x^{11} + x^8 + 1$	8191
14	$x^{14} + x^{13} + x^{12} + x^2 + 1$	16383
15	$x^{15} + x^{14} + 1$	32767
16	$x^{16} + x^{14} + x^{13} + x^{11} + 1$	65535

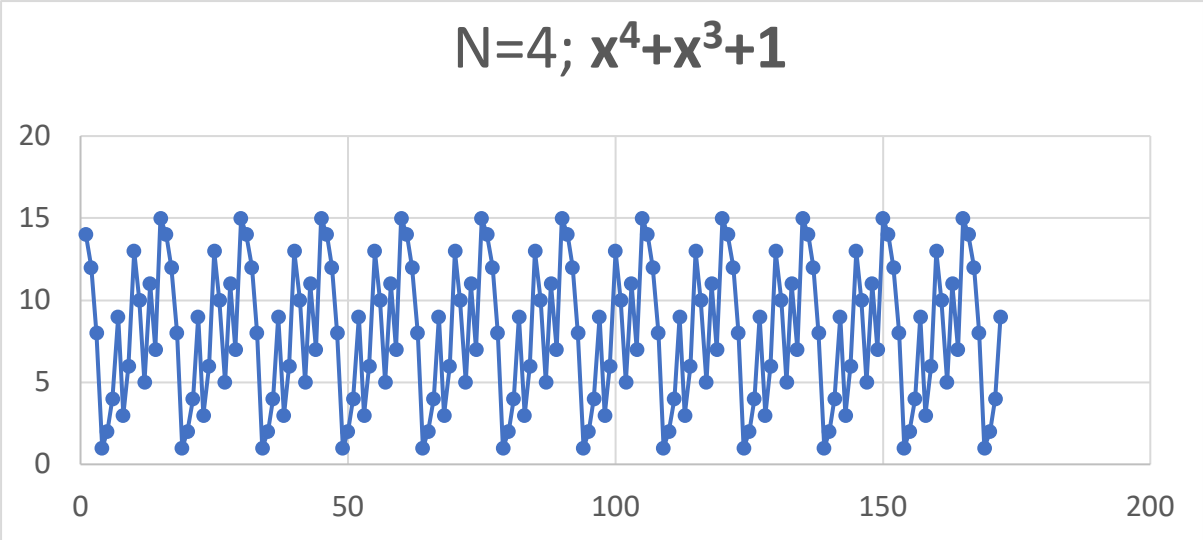
Fig. 2 – Polinômios de feedback.

Atividade 2: Geração de sequência pseudo-aleatória

1. Implemente um registrador de deslocamento com feedback linear em Verilog. Escolha três polinômios (pelo menos um com n< 4) e implemente os registradores. Apresente o Código em Verilog.
2. Represente a dependência com o número de ciclos do conteúdo dos registradores. Por exemplo, tal qual a representação na figura 3 abaixo.



É importante salientar que na figura acima, a ordem dos bits representa o conteúdo dos flip-flops, ou seja, o bit mais a esquerda é o bit 1, e o bit mais a direita, o bit 4. Portanto, quando voces rodarem o Código em Verilog e gerarem a sequência acima, os bits aparecerão invertidos, tal qual apresentado na aula.
A melhor forma de voces apresentarem estes dados é através de plotar o número em representação decimal com o tempo, conforme figura abaixo:



Observe que o período de repetição do padrão é muito claro e é máximo, igual a $T=2^4-1=15$ ciclos.
Tarefa: gere um gráfico como o apresentado acima para todos os casos estudados.

3. Quando o polinômio é muito grande, por exemplo de 16 Flip-Flops (FF), “enxergar” o padrão de repetição não é simples. Fica muito difícil por inspeção visual se a

sequência é ou não aleatória. Uma forma visual é através da representação em imagem bitmap da sequência de bits. No link https://boallen.com/random-numbers.html você encontrará um código em php. A sequência acima, agora extraíndo apenas o bit de saída do ultimo FF, você terá uma sequência de bits (0 ou 1), preto ou branco, que podem ser usados para gerar uma imagem.

I. Como gerar arquivos através do EDAPlayground: https://www.youtube.com/watch?v=MZh0-Bz5TwI
Caso tenham problemas na extração dos bits, o Icarus provavelmente será melhor do que o ambiente EDAPlayground. Na pior das hipoteses, voces podem copier da saida… mas é muito dado!

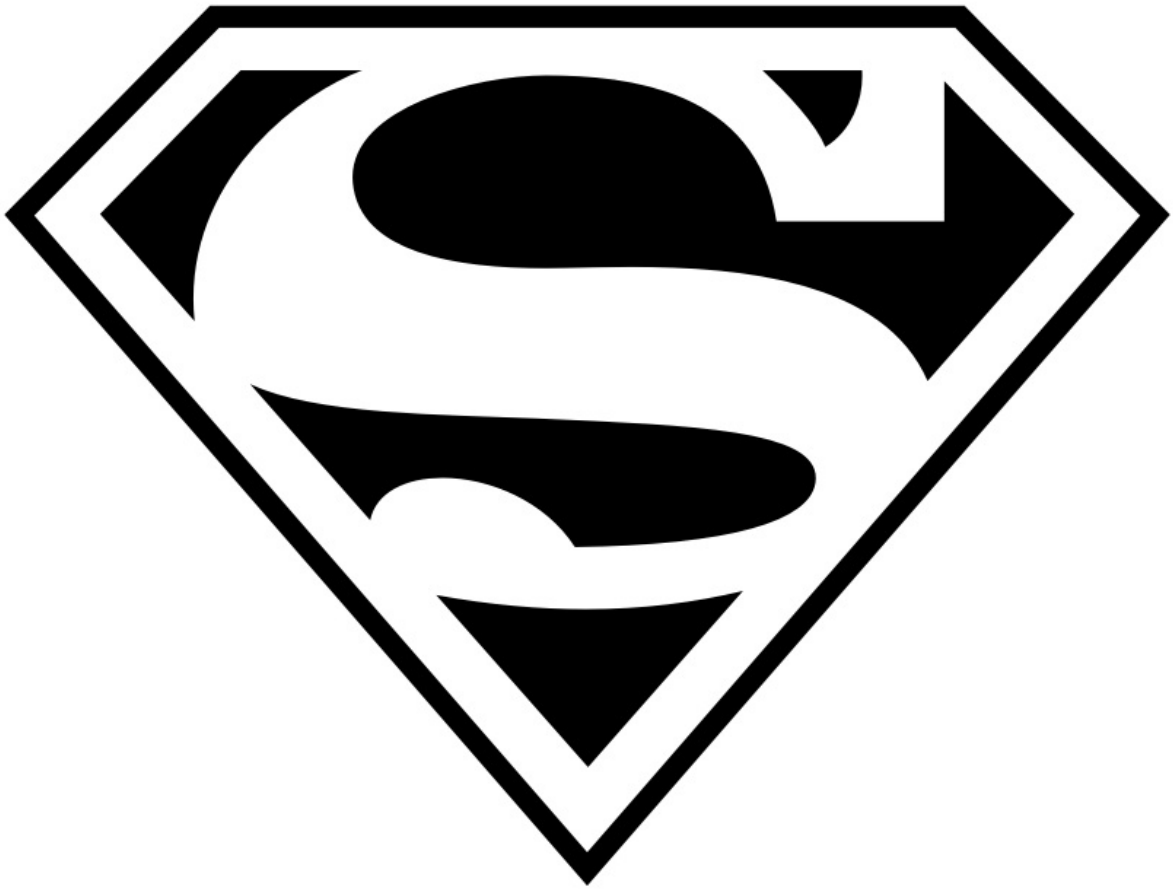
II. Extraia a sequência dos bits. Para a geração de imagens, voces podem assumer que bit0=0, e bit=1 seja por exemplo 255 (no caso de um arquivo de 8 bits). 0 é codificado em preo, e 255 em branco.

III. Importe para um program por exemplo como Python e gere a imagem. Abaixo um Código em python, mas pode ser qualquer codigo, ou qualquer programa (MatLab, mathematica, adobe photoshop, gimp, o que quiserem):

```
from  PIL  import  Image
with  open("file_with_1s_and_0s.txt") as f:
        lines = f.read().splitlines ()

        img = Image.open("grafico2d.png")
        width = img.size [0]
        height = img.size [1]

        for i in  range(0, width):
          for j in  range(0, height):
                data = img.getpixel ((i,j))

                if  lines[i*width+j] == "0":
                img.putpixel ((i,j) ,(0,0,0))

        img.show()
```

Atividade 3: Cifragem de Imagens

Utilizando python, ou qualquer linguagem de sua preferência, cifre a seguinte imagem bitmap (cada pixel pode ser 0 ou 1, preto ou branco):



1) Utilize dois polinômios, definidos previamente (um de grau 3, e um de grau >6), e **gere a imagem** CHAVE_LFSR XOR IMAGEM_BITMAP, onde CHAVE_LFSR é. Imagem que voce gerou na atividade 2, e IMAGEM_BITMAP é a imagem acima.
2) Faça o caminho contrário: Das imagens geradas no exercício anterior, faça um XOR com a CHAVE_LFSR para obter a **imagem original**. Ou voce pode fazer um XOR com uma outra chave, ou uma chave CHAVE_LFSR que contenha algum erro.

   O site abaixo mostra umas possibilidades:
   https://pyimagesearch.com/2021/01/19/opencv-bitwise-and-or-xor-and-not/

Observações

1) **Qualquer imagem preto e branco serve**
2) **Qualquer forma de fazer a operação bitwise serve**
3) **Imprimam os resultados (quero dizer, gerem o capture screen)**

TEMPLATE PARA PROJETO

1. **INTRODUÇÃO**
    - **a.** Descrição histórica, breve de algoritmos para a solução de desafios em segurança de dados;
    - **b.** Descrição de estratégias utilizadas para a geração de números aleatórios e as especificidades de cada estratégia.
    - **c.** Descrição de como implementar cifradores com hardware
2. **METODOLOGIA**
    - **a.** Descrição dos polinômios de Fibonacci, e como mapeá-los em circuitos;
    - **b.** Demonstração de comprimento de sequências para diversos polinômios
    - **c.** Elaboração do Código em Verilog
3. **RESULTADOS OBTIDOS**
    - **a.** Demonstração de cifragem de imagens com polinômios de diferentes comprimentos, explicitando o tamanho a partir do qual a imagem se torna aleatória;
    - **b.** Demonstração da “de-cifragem”, onde a partir da sequência se obtém a imagem original;
4. **CONCLUSÕES**
    - **a.** Discussão sobre o trabalho e o **curso** como um todo, e como os conhecimentos adquiridos lhe permitiram realizer o curso.
5. **REFERÊNCIAS**