UFMG

Introdução a Sistemas Lógicos - Trabalho Prático 4

Arthur Pontes Nader - 2019022294

Rodrigo Ferreira Araújo - 2020006990

Thales Henrique Silva - 2020007040

1) Introdução

No Trabalho Prático 1, aprendeu-se como cifrar uma mensagem utilizando-se uma One Time Pad (OTP) e a função lógica XOR. Em seguida, compreendeu-se como gerar bits pseudo-aleatórios utilizando registradores de deslocamento com feedback linear (LFSR). Dessa forma, o principal objetivo deste trabalho prático foi unir esses conhecimentos, utilizando a sequência de bits gerada por um LFSR ao invés do OTP para cifrar uma imagem, que no caso foi o brasão da UFMG.

Ao final, há um relatório conclusivo que aborda sobre os conceitos aprendidos ao longo da disciplina. Também é comentado sobre cada trabalho prático desenvolvido.

2) Funções criadas

Utilizou-se a linguagem Python e as bibliotecas Numpy e Matplotlib para realização da atividade.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
brasao = plt.imread('brasaoUFMG.jpg')
print(brasao.shape)
```

```
(197, 200, 3)
```

A seguir, é mostrado algumas das funções criadas para a cifragem da imagem, cada uma relacionada a uma determinada etapa do processo.

- **Processamento**

Primeiramente, é necessário transformar a imagem do brasão da UFMG em uma matriz binária para realização da cifragem. A função binarizar_imagem(img) faz exatamente isso.

```python
def binarizar_imagem(img):
    img_bin = np.zeros((img.shape[0], img.shape[1]))

    threshold = 190

    for linha in range (img.shape[0]):
        for pixel in range (img.shape[1]):

            valor = 0
            for cor in range(img.shape[2]):
                valor += img[linha][pixel][cor]
            valor = valor/3

            if ( valor > threshold):
                img_bin[linha][pixel] = 255


    return img_bin
```

- **Exibição**

A função mostrar_imagem(imagem) utiliza funções da biblioteca Matplotlib para exibir a imagem. A seguir, observa-se a exibição da imagem após realização do processo de binarização.

```python
def mostrar_imagem(imagem):
    plt.imshow(imagem, cmap = 'gray')
    plt.axis('off')
    plt.show()
```

```python
brasao_binario = binarizar_imagem(brasao)
mostrar_imagem(brasao_binario)
```

- **Transformação da sequência**

O código em Verilog gera uma sequência de 0's e 1's. Para realização da cifragem, deve-se converter os 1's obtidos em 255, que representará a cor branca. Os 0's não precisam ser alterados porque já representam a cor preta. Essa substituição é feita pela função tranforma_vetor(vetor), mostrada a seguir:

```
def transformar_vetor(vetor):
    vetor_auxiliar = []

    for bit in vetor:
        if bit == 0:
            vetor_auxiliar.append(0)
        else:
            vetor_auxiliar.append(255)

    return vetor_auxiliar
```

- **Função XOR**

Para realização da operação XOR, criou-se a seguinte função adaptada para o xor de pixels de uma imagem binária:

```
def xor(bit1, bit2):

    if bit1 == bit2:
        resultado = 0
    else:
        resultado = 255

    return resultado
```

- **Cifragem da imagem**

A função cifragem_imagem(img, seq_lfsr) mostrada a seguir, utiliza a função xor(bit1, bit2) e a sequência pseudo-aleatória gerada pelo LFSR para cifrar a imagem.

```
def cifragem_imagem(img, seq_lfsr):

    contador = 0

    img_cifrada = np.zeros((img.shape[0], img.shape[1]))

    for i in range (img.shape[0]):
        for k in range (img.shape[1]):
            img_cifrada[i][k] = xor(img[i][k], seq_lfsr[contador % len(seq_lfsr)])
            contador += 1


    return img_cifrada
```

3) Cifragem com polinômio de período pequeno

Utilizou-se o seguinte polinômio para a primeira cifragem:

$$x^3 + x^2 + x^1 + 1$$

A seed utilizada foi “110”, sendo que assim obteve-se ao final um período igual a 4. O código em Verilog e os 4 diferentes estados obtidos são mostrados a seguir:

design.sv

```
//Design
module LFSR_3(clock,clear,ff_states);
  input clock,clear;
  output reg [2:0] ff_states= 3'b110;
  reg A,B,C;
  always @(posedge clock or negedge clear)
    begin
      //inicializacao dos flip flops
      if(!clear) begin A<=1;B<=1;C<=0; end
      //bit de entrada -> funcao linear do estado anterior
      else begin
        A<=C^B^A;
        B<=A;
        C<=B;
        ff_states[2] <= A;
        ff_states[1] <= B;
        ff_states[0] <= C;
      end
    end
endmodule
```

testbench.sv

```
// Testbench
module Bancada_Teste;
  //declaracao das variaveis
  reg clk,clr;
  wire [2:0] out;

  LFSR_3 LF3(clk,clr,out);

  //frequencia do clock e inicializacao do clear
  initial
    begin clk=1'b0;#2 clr=0;#5 clr=1; end
  always #5 clk=~clk;

  //exibicao das saidas
  initial
    begin
      $monitor("out = %b", out);
      #60 $finish;
    end

  initial
    begin
      $dumpfile("LFSR_3.vcd");
      $dumpvars(0,Bancada_Teste);
      #60 $finish;
    end
endmodule
```

```
Log  Share
[2021-09-03 12:32:03 EDT] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
VCD info: dumpfile LFSR_3.vcd opened for output.
out = 110
out = 011
out = 001
out = 100
out = 110
Finding VCD file...
./LFSR_3.vcd
[2021-09-03 12:32:03 EDT] Opening EPWave...
Done
```

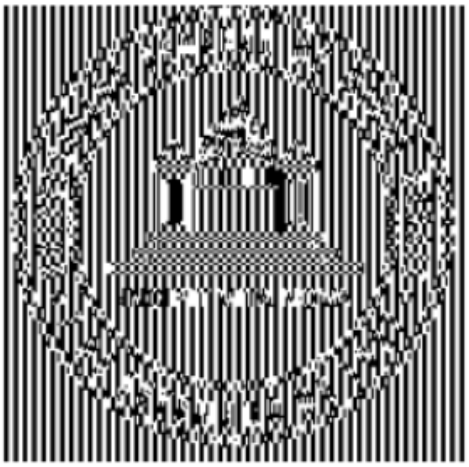
Extraindo se o último bit de cada estado e utilizando as funções criadas, cifrou-se o brasão da UFMG, obtendo-se o seguinte resultado:

```
lfsr_curto = [0,1,1,0]
lfsr_curto_imagem = transformar_vetor(lfsr_curto)
print(lfsr_curto_imagem)
```

```
[0, 255, 255, 0]
```

```
imagem_cifrada = cifragem_imagem(brasao_binario, lfsr_curto_imagem)
```

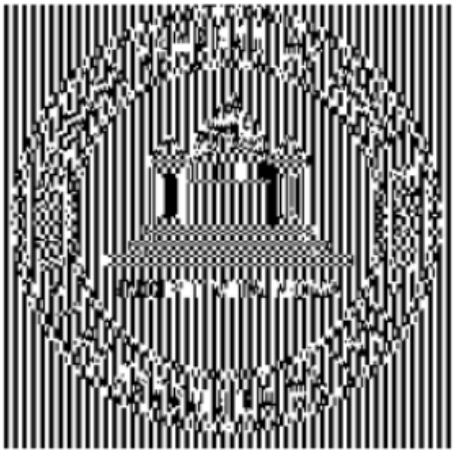
```
mostrar_imagem(brasao_binario)
mostrar_imagem(imagem_cifrada)
```



Ao realizar o processo novamente, mas passando como parâmetro para a função a imagem cifrada, é possível decifrá-la, tal qual mostrado a seguir:

```
imagem_decifrada = cifragem_imagem(imagem_cifrada, lfsr_curto_imagem)

mostrar_imagem(imagem_cifrada)
mostrar_imagem(imagem_decifrada)
```



4) Cifragem com polinômio de período longo

Agora, para a segunda cifragem da imagem do brasão da UFMG, utilizou-se o seguinte polinômio:

$$x^8 + x^6 + x^5 + x^4 + 1$$

Dessa vez, a seed utilizada foi “11111111”, obtendo assim um período igual a 255. O código em Verilog desenvolvido é mostrado a seguir. Como muitos estados foram gerados, apenas os primeiros foram mostrados.

design.sv

```
//Design
module LFSR_8(clock,clear,ff_states);
  input clock,clear;
  output reg [7:0] ff_states= 8'b11111111;
  reg A,B,C,D,E,F,G,H;
  always @(posedge clock or negedge clear)
    begin
      //inicializacao dos flip flops
      if(!clear) begin A<=1;B<=1;C<=1;D<=1;E<=1;F<=1;G<=1;H<=1; end
      //bit de entrada -> funcao linear do estado anterior
      else begin
        A<=H^F^E^D;
        B<=A;
        C<=B;
        D<=C;
        E<=D;
        F<=E;
        G<=F;
        H<=G;
        ff_states[7] <= A;
        ff_states[6] <= B;
        ff_states[5] <= C;
        ff_states[4] <= D;
        ff_states[3] <= E;
        ff_states[2] <= F;
        ff_states[1] <= G;
        ff_states[0] <= H;
      end
    end
endmodule
```

testbench.sv

```
1  // Testbench
2  module Bancada_Teste;
3    //declaracao das variaveis
4    reg clk,clr;
5    wire [7:0] out;
6
7    LFSR_8 LF8(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       #400 $finish;
19     end
20
21   initial
22     begin
23       $dumpfile("LFSR_8.vcd");
24       $dumpvars(0,Bancada_Teste);
25       #400 $finish;
26     end
27 endmodule
```

Log | Share

```
[2021-09-03 14:00:36 EDT] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
VCD info: dumpfile LFSR_8.vcd opened for output.
out = 11111111
out = 01111111
out = 00111111
out = 00011111
out = 00001111
out = 10000111
out = 01000011
out = 10100001
out = 11010000
out = 11101000
out = 11110100
out = 01111010
out = 00111101
out = 00011110
out = 10001111
out = 11000111
out = 01100011
out = 10110001
out = 01011000
out = 00101100
out = 00010110
out = 00001011
out = 00000101
out = 00000010
out = 00000001
out = 10000000
out = 01000000
out = 00100000
out = 00010000
out = 10001000
out = 11000100
```

Novamente, extraiu-se o último bit de cada estado e transformando-se o vetor gerado, obteve-se o seguinte resultado:

```
lfsr_8 = "1111111100001011110001101000000010001110001001011100000011001001001101110010000010101101101011001011000011111011011110
lfsr_8 = list(lfsr_8)
print(len(lfsr_8))
```

```
lfsr_longo = [int(x) for x in lfsr_8]
print(lfsr_longo)
```

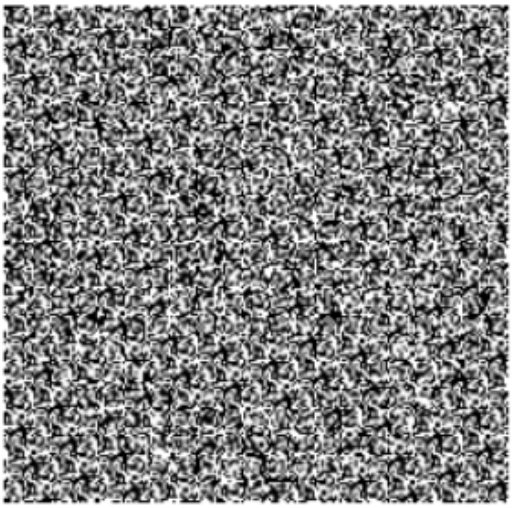
```
lfsr_longo_imagem = transformar_vetor(lfsr_longo)
print(lfsr_longo_imagem)
```

```
[255, 255, 255, 255, 255, 255, 255, 255, 0, 0, 0, 0, 255, 0, 255, 255, 255, 255, 0, 0, 0, 255, 255, 0, 255, 0, 0, 0, 0, 0, 0,
0, 255, 0, 0, 0, 255, 255, 255, 0, 0, 0, 255, 0, 0, 255, 0, 255, 255, 255, 0, 0, 0, 0, 0, 255, 255, 0, 0, 255, 0, 0, 255, 0,
0, 255, 255, 0, 255, 255, 255, 0, 0, 255, 0, 0, 0, 0, 0, 255, 0, 255, 0, 255, 255, 0, 255, 255, 0, 255, 0, 255, 255, 0, 0, 255,
0, 255, 255, 0, 0, 0, 0, 255, 255, 255, 255, 255, 0, 255, 255, 0, 255, 255, 255, 255, 0, 255, 0, 255, 255, 255, 0, 255, 0, 0,
0, 255, 0, 0, 0, 0, 255, 255, 0, 255, 255, 0, 0, 0, 255, 255, 255, 255, 0, 0, 255, 255, 255, 0, 0, 255, 255, 0, 0, 0, 255, 0, 2
55, 255, 0, 255, 0, 0, 255, 0, 0, 0, 255, 0, 255, 0, 0, 255, 0, 255, 0, 255, 0, 0, 255, 255, 255, 0, 255, 255, 255, 0, 255, 25
5, 0, 0, 255, 255, 255, 255, 0, 255, 255, 255, 255, 255, 255, 0, 255, 0, 0, 255, 255, 0, 0, 255, 255, 0, 255, 0, 255, 0, 0, 0,
255, 255, 0, 0, 0, 0, 0, 255, 255, 255, 0, 255, 0, 255, 0, 255, 0, 255, 255, 255, 255, 255, 0, 0, 255, 0, 255, 0, 0, 0, 0, 255,
0, 0]
```

Em seguida, utilizou-se a função criada para cifrar a imagem. Os resultados obtidos são mostrados a seguir:

```
imagem_cifrada = cifragem_imagem(brasao_binario, lfsr_longo_imagem)
```

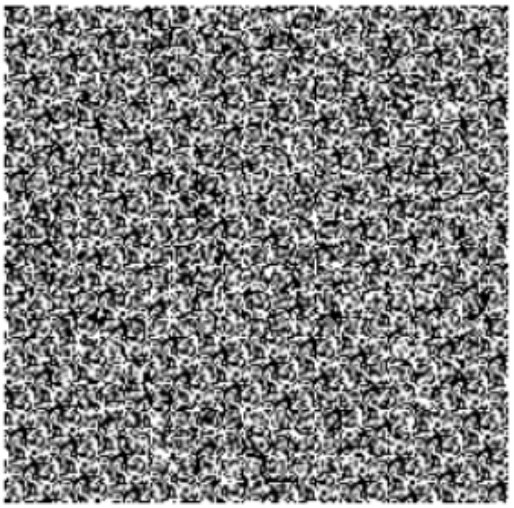
```
mostrar_imagem(brasao_binario)
mostrar_imagem(imagem_cifrada)
```



Após a decifragem, obteve-se:

```
imagem_decifrada = cifragem_imagem(imagem_cifrada, lfsr_longo_imagem)

mostrar_imagem(imagem_cifrada)
mostrar_imagem(imagem_decifrada)
```

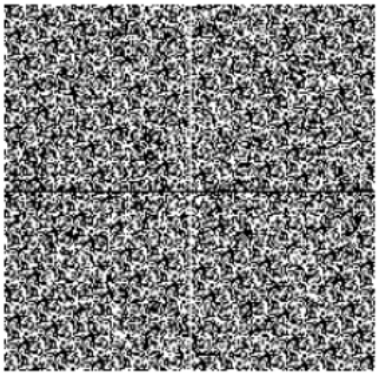


5) Decifragem com um “erro” em parte da sequência

Observa-se que a decifragem com certas linhas e colunas da imagem cifrada alteradas não recupera a imagem inicial. Esse erro é comumente associado a falhas de transmissão.

```
imagem_cifrada[:,100] = 255
imagem_cifrada[100,:] = 0
imagem_decifrada = cifragem_imagem(imagem_cifrada, lfsr_longo_imagem)

mostrar_imagem(imagem_cifrada)
mostrar_imagem(imagem_decifrada)
```



6) Relatórios finais

Relatório Sobre os Trabalhos Práticos

- TP1: A partir desse trabalho individual, conseguimos visualizar e analisar a fundo, com a linguagem de descrição de hardware Verilog, o funcionamento de um mecanismo simples de criptografia: o One Time Pad (OTP). Nesse sentido, conseguimos cifrar e decifrar uma sequência de caracteres fazendo XOR bit-a-bit entre a mensagem original e uma chave (pseudo ou realmente aleatória) para cifrá-la, e XOR bit-a-bit entre a mensagem cifrada e a mesma chave para decifrá-la.
- TP2: Neste trabalho foram aplicados conceitos sobre registradores de deslocamento e contadores na linguagem Verilog, mais especificamente: registrador para esquerda, registrador com feedback e contador Johnson. Além disso, analisamos os diagramas de onda obtidos para observar a saída dos circuitos com o pulsar do clock.
- TP3: Aprendemos conceitos sobre LFSR, que é um tipo de registrador capaz de gerar números pseudo-aleatórios. Eles não são verdadeiramente aleatórios pois a sequência depende do estado inicial (seed) e do polinômio escolhido. Além do mais, existe um período a partir do qual os números voltam a se repetir. Quando a sequência é muito extensa, é possível usar uma imagem bitmap para observar as repetições. Deseja-se, então, que a figura obtida seja o mais "caótica" possível, indicando a ausência de padrões.
- TP4: No último Trabalho Prático, implementamos uma solução à uma aplicação possível de cifragem de imagens utilizando bits do período de sequências gerados tanto por um LFSR de período longo tanto para um de período curto como chaves para a cifragem, semelhante ao processo empregado no TP1.

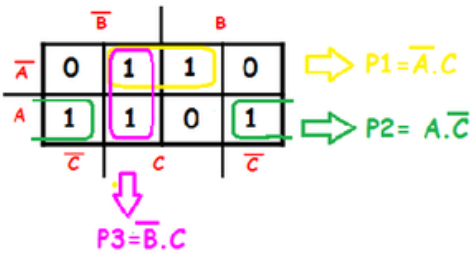
Relatório Sobre o Aprendizado

Os principais conceitos aprendidos com relação à lógica combinacional foram:

- leis e teoremas da Álgebra booleana;
- A dependência exclusiva da saída do circuito para com a entrada;
- A quantidade de possibilidades de se arranjar portas lógicas com o fim de economizar transistores (conversão para redes NAND-NAND e NOR-NOR);
- Múltiplas abordagens para simplificação de equações características de circuitos, desde as mais simples (dedução à partir do circuito CMOS, expansão de minterms/maxterms, simplificação algébrica a partir da SoP/PoS), até as mais complexas (cubos booleanos, mapas de Karnaugh, algoritmos para simplificação de 2 níveis, Quine-McCluskey);
- Múltiplas tecnologias para se abordar circuitos complexos em lógica combinacional, como: (De)multiplexadores, a programabilidade dos PLAs e a rapidez dos PALs, ROM (Read-Only Memory);

Exemplos de aplicações:

- Mapa de Karnaugh:

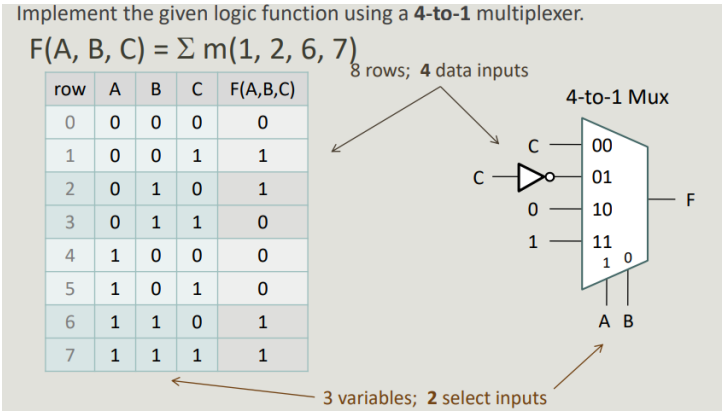


Observe que as adjacências podem ser usadas para simplificar as expressões.

- Tabela com os Minterms e Maxterms de 3 bits:

Row number	x_1	x_2	x_3	Minterm	Maxterm
0	0	0	0	$m_0 = \bar{x}_1\bar{x}_2\bar{x}_3$	$M_0 = x_1 + x_2 + x_3$
1	0	0	1	$m_1 = \bar{x}_1\bar{x}_2x_3$	$M_1 = x_1 + x_2 + \bar{x}_3$
2	0	1	0	$m_2 = \bar{x}_1x_2\bar{x}_3$	$M_2 = x_1 + \bar{x}_2 + x_3$
3	0	1	1	$m_3 = \bar{x}_1x_2x_3$	$M_3 = x_1 + \bar{x}_2 + \bar{x}_3$
4	1	0	0	$m_4 = x_1\bar{x}_2\bar{x}_3$	$M_4 = \bar{x}_1 + x_2 + x_3$
5	1	0	1	$m_5 = x_1\bar{x}_2x_3$	$M_5 = \bar{x}_1 + x_2 + \bar{x}_3$
6	1	1	0	$m_6 = x_1x_2\bar{x}_3$	$M_6 = \bar{x}_1 + \bar{x}_2 + x_3$
7	1	1	1	$m_7 = x_1x_2x_3$	$M_7 = \bar{x}_1 + \bar{x}_2 + \bar{x}_3$

- Implementando uma função com MUX:



Os principais conceitos aprendidos com relação à lógica sequencial foram:

- Latches: são circuitos com retroalimentação implementados com duas portas NAND ou NOR. A depender das entradas, as saídas podem mudar ou manter o valor anterior.

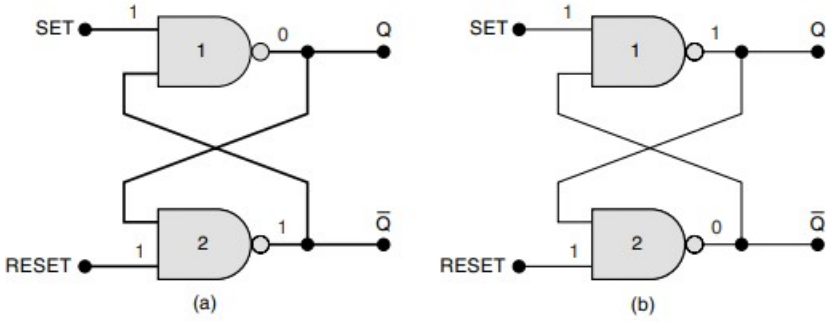


FIGURA 5.3 Um latch com portas NAND tem dois estados de repouso possíveis quando SET = RESET = 1.

- Flip-flops: são circuitos capazes de armazenar um bit de informação. Geralmente, a sua saída só pode mudar de acordo com as transições de subida ou descida do clock. Isso tem a vantagem de criar um sistema com comportamento mais previsível e fácil de controlar. Há vários tipos de flip-flops, tais como J-K, tipo D, etc. Os FFs são o principal fundamento dos contadores, registradores e máquinas de estado finitas.

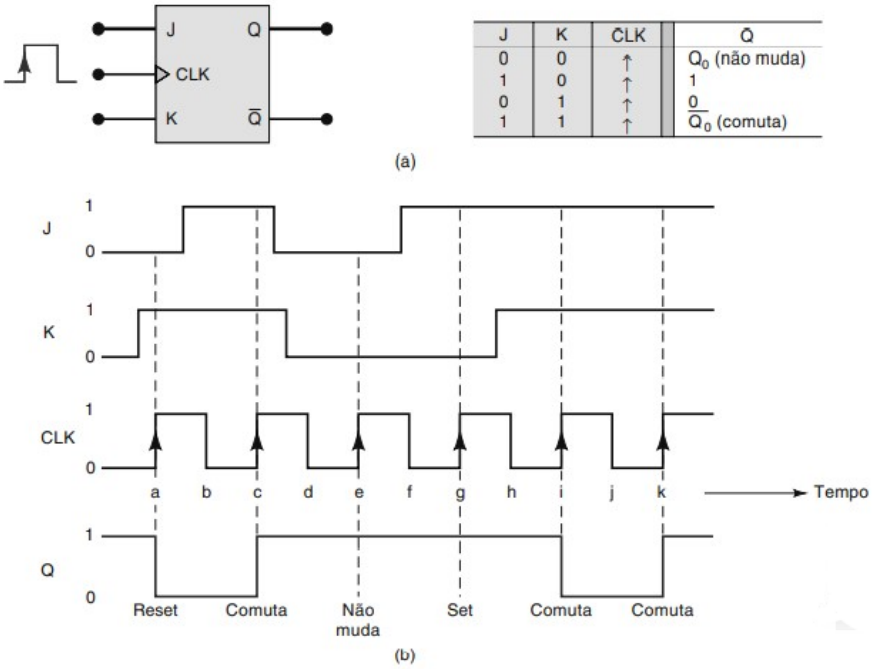
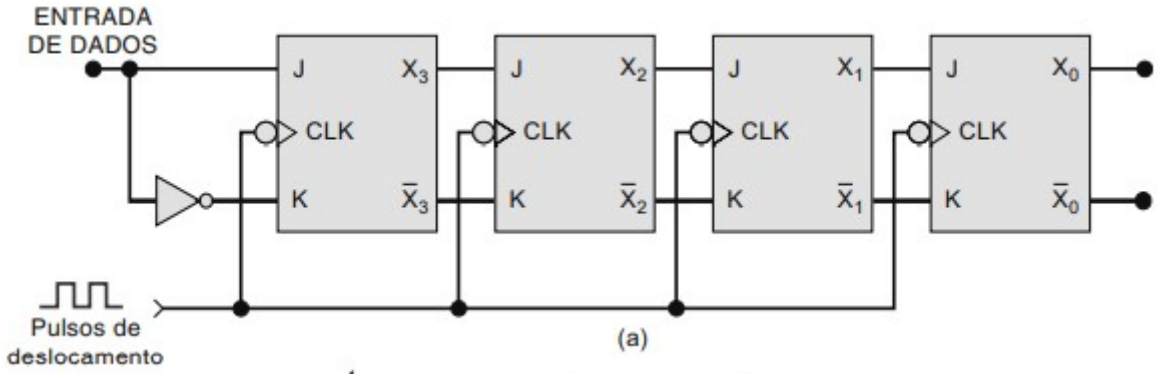


FIGURA 5.23 (a) Flip-flop *J-K* com clock que responde apenas às bordas positivas do clock; (b) formas de ondas.

Exemplo de diagrama de onda de um FF tipo J-K.

Os flip-flops têm inúmeras aplicações, incluindo armazenamento e transferência de dados, contagem, etc.

- Registradores de deslocamento: consistem em um grupo de flip-flops em cascata, de tal forma que o bit de um deles pode ser transferido para o próximo.



Há vários tipos de registradores, que podem ter saídas/entradas serial ou paralela. Naturalmente, eles são amplamente utilizados nos circuitos modernos, tais como CPU's, memória ROM, etc.

- LFSR (Linear Feedback Shift Register): São registradores de deslocamento com uma propriedade especial: possuem feedback fazendo XOR de bits específicos da cadeia gerada, determinado por um polinômio característico, por exemplo: $X^5 + X^3 + 1$ é um

polinômio que indica que os bits 5 e 3 da cadeia devem entrar no XOR e o bit de resultado deve ser enviado na realimentação do registrador (feedback). Nesse sentido, os LFSRs produzem períodos de cadeias de bits, que podem ser utilizados para gerar um bitmap (retirando-se o último bit de cada cadeia) para observar o padrão do período em uma imagem, bem como pode ser usado como chave para cifragem de uma sequência de bits, como uma imagem vista no TP4. Desse modo, deve-se tomar cuidado ao se escolher o polinômio característico, uma vez que, a partir de um período pequeno, é viável realizar engenharia reversa na cadeia cifrada e revelar a imagem, testando-se múltiplas combinações de cadeias de LFSR de períodos pequenos, ou, também, caso seja uma imagem, o conteúdo original pode estar parcialmente compreensível (bits gerados pelo LFSR são pseudo-aleatórios) mesmo após a cifragem, o que não é desejável.

- Máquinas de estado: refere-se a um tipo de circuito com um conjunto de estados predeterminados. Talvez o exemplo mais conhecido seja o dos contadores, os quais seguem uma sequência bem definida. Para tais circuitos o uso de diagramas de estado é uma ferramenta didática para auxiliar o seu entendimento.

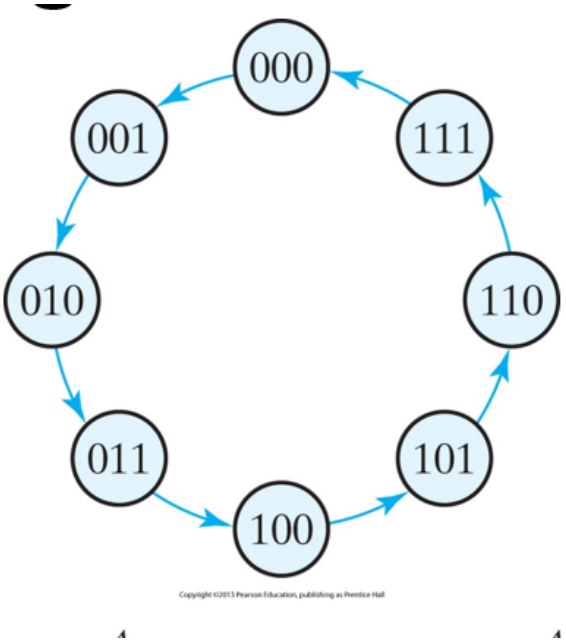


Diagrama de um contador de 3 bits

  - Aplicação de mecanismos da lógica combinacional e sequencial para construção de máquinas de comportamento previsível e circular, como contadores de n bits que retornam a 0 após esgotar os estados possíveis;

7) Referências:

- Slides e aulas da disciplina;
- TOCCI, Ronald; L. MOSS, Gregory ; S. WIDMER, Neal. **Sistemas Digitais. Princípios e Aplicações**. 11 ed.
- Matplotlib imread em Python | Delft Stack
- Display an Image in Grayscale in Matplotlib | Delft Stack