Solução Atividades das Aulas 12e13

Curso: Especialização em Microeletrônica para Front-End Digital/PECS

Matrícula: 20252009299

Aluno: Manoel Felipe Costa Furtado

CPF: 113.279.707-16

Tema: Circuitos Digitais II – Aulas 12e13 – A-203 Contadores Assíncronos

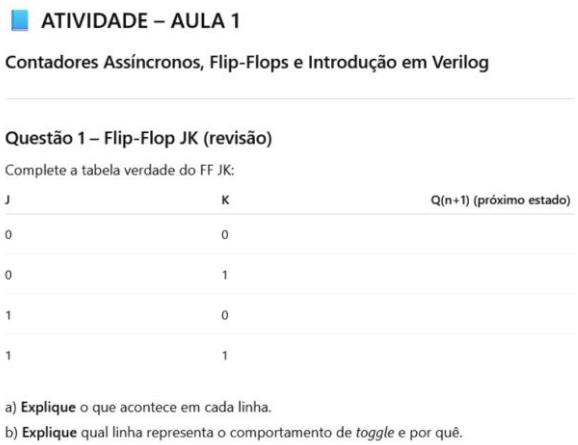
Prazo: 28/11/2025

Compilar no software Quartus e Simular no software Questa.

Link dos Códigos:

Link GitHub: https://github.com/ManoelFelipe/CI_DITAL_TIC_41

Atividade 01:



📘 ATIVIDADE – AULA 1

Contadores Assíncronos, Flip-Flops e Introdução em Verilog

Questão 1 – Flip-Flop JK (revisão)

Complete a tabela verdade do FF JK:

J	K	Q(n+1) (próximo estado)
0	0	
0	1	
1	0	
1	1	

a) **Explique** o que acontece em cada linha.

b) **Explique** qual linha representa o comportamento de *toggle* e por quê.

a) Explique o que acontece em cada linha:

J	K	Q(n+1) (próximo estado)
0	0	**Q(n)** (Mantém o estado atual)
0	1	**0** (Reset)
1	0	**1** (Set)
1	1	**Q(n)** (Toggle / Inverte)

- **Linha 1 (J=0, K=0):** O Flip-Flop está em estado de **memória (hold)**. O valor da saída Q não se altera, permanecendo igual ao estado anterior Q(n).
- **Linha 2 (J=0, K=1):** O Flip-Flop entra em estado de **Reset**. A saída Q é forçada para o nível lógico **0**, independentemente do estado anterior.
- **Linha 3 (J=1, K=0):** O Flip-Flop entra em estado de **Set**. A saída Q é forçada para o nível lógico **1**, independentemente do estado anterior.

- **Linha 4 (J=1, K=1):** O Flip-Flop entra em estado de **Toggle (comutação)**. A saída inverte seu valor; se estava em 0 vai para 1, e se estava em 1 vai para 0 (Q(n+1) = ‘Q(n).

b) Explique qual linha representa o comportamento de *toggle* e por quê:

A linha que representa o comportamento de *toggle* é a **última linha**, onde **J = 1 e K = 1**.

Por que: O termo "toggle" significa alternar ou comutar. Nesta configuração, a cada pulso de *clock*, o Flip-Flop olha para o seu estado atual e muda para o estado oposto (complemento). É o princípio básico utilizado para criar divisores de frequência e contadores binários.

Questão 2 – Flip-Flop D (revisão)

Complete:

D	Q(n+1)
0	
1	

b) **Explique** por que o FF D é adequado para registradores e contadores.

a) Tabela Verdade Preenchida:

O Flip-Flop D (do inglês *Data* ou *Delay*) é o mais simples e utilizado em projetos digitais modernos. A saída simplesmente copia a entrada na borda do *clock*.

D	Q(n+1) (próximo estado)
0	**0** (Reset / Nível Baixo)
1	**1** (Set / Nível Alto)

b) Explique por que o FF D é adequado para registradores e contadores:

O Flip-Flop tipo D é fundamental nesses componentes por dois motivos principais:

1. **Para Registradores (Armazenamento de Dados):**
    - A principal função de um registrador é armazenar um valor binário. Como o Flip-Flop D transfere o valor exato da entrada D para a saída Q quando o *clock* é acionado, ele é perfeito para "capturar" e manter dados. Para criar um registrador de 8 bits, por exemplo, basta alinhar 8 Flip-Flops tipo D em paralelo.

2. **Para Contadores (Simplicidade Síncrona):**
    - Em projetos síncronos (como dentro de FPGAs), os contadores são feitos calculando o "próximo número" através de lógica combinacional e jogando esse resultado na entrada D.
    - Diferente do Flip-Flop SR, o tipo D **não possui estado proibido/indeterminado**. Isso garante que o circuito seja estável e previsível, o que é crucial para contadores que não podem "se perder" na contagem.

```verilog
// Exemplo de Flip-Flop D em Verilog
module d_flip_flop (
    input clk,
    input d,
    output reg q
);

    always @(posedge clk) begin
        q <= d; // A saída Q recebe D na borda de subida
    end

endmodule
```

Questão 3 – Conceitos de contadores

a) **Explique** o que é um contador digital.

b) **Explique** o que significa um contador de módulo 8.

c) **Explique** quantos estados possíveis tem um contador com:

- 3 flip-flops
- 4 flip-flops

d) **Explique** a diferença entre:

- contador síncrono
- contador assíncrono.

a) Explique o que é um contador digital.

Um contador digital é um **circuito sequencial** construído (geralmente) a partir de Flip-Flops conectados em cascata. Sua função principal é **contar pulsos de clock**. Ele percorre uma sequência pré-determinada de estados binários (ex: 00, 01, 10, 11) a cada vez que recebe um sinal de entrada (borda de clock). Eles são usados para medir tempo, contar eventos, dividir frequências e endereçar memórias.

b) Explique o que significa um contador de módulo 8.

O termo "Módulo" (ou MOD) refere-se ao **número total de estados únicos** que o contador assume antes de repetir a sequência.

- Um contador **Módulo 8** possui **8 estados diferentes**.
- Ele conta de 0 a 7 (em binário: de 000 a 111).

- Após chegar ao 7 (111), no próximo pulso de clock, ele reinicia (volta) para o 0 (000).

c) Explique quantos estados possíveis tem um contador com:

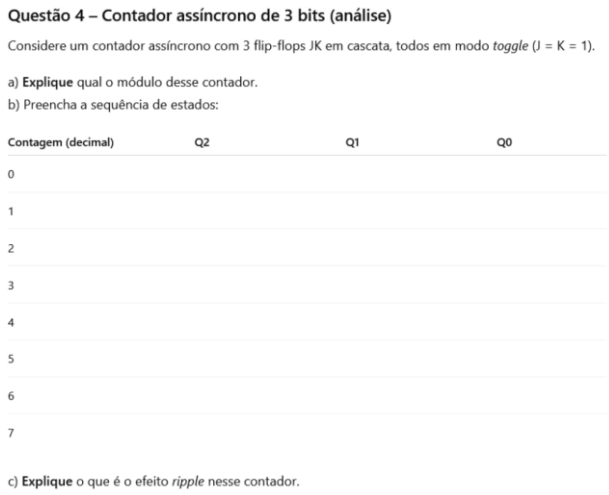
O número de estados é calculado pela fórmula **2^n**, onde n é o número de Flip-Flops.

- 3 flip-flops: 2^3 = 8 estados (Conta de 0 a 7).
- 4 flip-flops: 2^4 = 16 estados (Conta de 0 a 15).

d) Explique a diferença entre contador síncrono e contador assíncrono.

Esta é a distinção mais importante para projetos em FPGA e Verilog:

Característica	Contador Assíncrono (Ripple)	Contador Síncrono (Paralelo)
Clock	O clock é aplicado apenas no primeiro Flip-Flop. A saída de um FF serve de clock para o próximo.	O **mesmo sinal de clock** é conectado a **todos** os Flip-Flops simultaneamente.
Velocidade	**Mais lento.** O atraso de propagação se acumula (efeito cascata), limitando a frequência máxima.	**Mais rápido.** O atraso é apenas o tempo de um único FF mais as portas lógicas.
Complexidade	Simples (poucas conexões).	Mais complexo (exige lógica combinacional extra entre os estágios).
Uso em FPGA	**Evitado** (pode causar falhas/glitches temporais).	**Padrão ouro** (ideal para design robusto em Verilog/VHDL).



Questão 4 – Contador assíncrono de 3 bits (análise)

Considere um contador assíncrono com 3 flip-flops JK em cascata, todos em modo *toggle* (J = K = 1).

a) **Explique** qual o módulo desse contador.
b) Preencha a sequência de estados:

Contagem (decimal)	Q2	Q1	Q0
0			
1			
2			
3			
4			
5			
6			
7			

c) **Explique** o que é o efeito *ripple* nesse contador.

Cenário: 3 Flip-Flops JK em cascata, modo *toggle* (J=K=1).

a) Explique qual o módulo desse contador.

O módulo é **8**.

- **Por quê?** O módulo de um contador binário é determinado pela fórmula 2^n, onde n é o número de Flip-Flops.
- Como temos n=3, então 2^3 = 8. Isso significa que o contador possui **8 estados distintos** (contando de 0 a 7) antes de reiniciar a sequência.

b) Preencha a sequência de estados:

Aqui temos a contagem binária padrão de 3 bits. O bit Q_0 é o menos significativo (LSB) e muda a cada pulso, enquanto Q_2 é o mais significativo (MSB).

Contagem (decimal)	Q2 (MSB)	Q1	Q0 (LSB)
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

(Após o 7, o próximo estado volta a ser 000).

c) Explique o que é o efeito *ripple* nesse contador.

O termo *"Ripple"* (ondulação) refere-se ao **atraso de propagação acumulado** que ocorre nos contadores assíncronos.

1. **A Causa:** Em um contador assíncrono, o *clock* externo aciona apenas o primeiro Flip-Flop (Q_0). O segundo Flip-Flop (Q_1) não usa o *clock* principal; ele usa a **saída** de Q_0 como se fosse seu *clock*. O terceiro (Q_2) usa a saída de Q_1, e assim por diante.
2. **O Efeito:** Quando ocorre uma mudança de estado (por exemplo, de 011 para 100), o sinal tem que "viajar" através de toda a cadeia:
    - Q_0 muda primeiro...
    - ...que faz Q_1 mudar um pouco depois...
    - ...que faz Q_2 mudar um pouco depois.
3. **Consequência:** Os bits não mudam todos ao mesmo tempo. Há um instante transitório onde a saída pode apresentar valores incorretos (glitches) antes de estabilizar. É como um efeito dominó.

```
module contador_sincrono (
    input clk,
    input reset,
    output reg [3:0] q // 4 bits = 16 estados
);

    always @(posedge clk or posedge reset) begin
        if (reset)
            q <= 4'b0000;
        else
            q <= q + 1; // Todos os bits mudam sincronizados pelo clk
    end

endmodule
```

Questão 5 – Vantagens e desvantagens (Contadores Assíncronos)

Esta questão foca na análise crítica dos contadores assíncronos (Ripple Counters).

a) Explique duas vantagens dos contadores assíncronos.

1. **Simplicidade de Hardware (Economia):** Eles utilizam **menos portas lógicas**. Ao contrário dos contadores síncronos, que precisam de lógica combinacional extra para calcular o próximo estado de cada bit, o assíncrono só precisa dos fios conectando a saída de um Flip-Flop ao clock do próximo.
2. **Facilidade de Projeto:** Para circuitos simples e contadores pequenos, o design é quase imediato (apenas cascatear Flip-Flops), sem necessidade de desenhar mapas de Karnaugh ou tabelas de excitação complexas.

b) Explique três desvantagens.

1. **Velocidade Limitada (Lentidão):** Devido ao efeito *ripple* (propagação em cascata), o atraso total é a soma dos atrasos de todos os Flip-Flops. Isso torna o contador muito mais lento que o síncrono à medida que o número de bits aumenta.
2. **Glitches na Decodificação (Estados Espúrios):** Durante a transição dos bits (enquanto o sinal viaja do LSB para o MSB), o contador passa brevemente por números "falsos". Se você ligar um decodificador na saída, ele pode detectar esses valores intermediários e acionar circuitos errados.
3. **Dificuldade de Análise Temporal:** Em sistemas digitais complexos (como FPGAs), ferramentas de *Timing Analysis* têm dificuldade em garantir a estabilidade do circuito, pois não há um clock único e global sincronizando todas as mudanças.

c) Explique por que eles não são recomendados para altas frequências de clock.

Eles não são recomendados porque o **período do clock (T) deve ser maior que o atraso total acumulado** da cadeia de Flip-Flops.

Se você aumentar muito a frequência (diminuir o período T), o sinal de mudança gerado pelo primeiro Flip-Flop pode não ter tempo suficiente para chegar até o último Flip-Flop antes que o próximo pulso de clock entre.

- **Matematicamente:** Se cada Flip-Flop tem um atraso t_pd e você tem N bits, o contador falhará se o período do clock for menor que N X t_pd.

- **Resultado:** O contador começa a "pular" contagens ou apresentar valores aleatórios nos bits mais significativos.

```
// SEMPRE PREFIRA ISSO (Síncrono)
always @(posedge clk) count <= count + 1;

// AO INVÉS DISSO (Assíncrono / Ripple - Má prática em FPGA)
always @(posedge q0) q1 <= ~q1;
```

Isso garante que o *synthesizer* da ferramenta (Quartus, Vivado) consiga otimizar as rotas e garantir que seu circuito funcione na frequência desejada sem erros de *setup/hold time*.

Questão 6 – Interpretando Verilog de um FF JK

a) Explique o que cada linha marcada (1), (2), (3) e (4) faz com a saída q.

Estas linhas representam a lógica de transição de estados baseada nas entradas {j, k} concatenadas:

- **(1) 2'b00: q <= q;**
  - **Ação: Memória (Hold)**.
  - **Explicação:** O Flip-Flop mantém o estado atual inalterado. Se estava 0, fica 0; se estava 1, fica 1.
- **(2) 2'b01: q <= 1'b0;**
  - **Ação: Reset**.
  - **Explicação:** Força a saída q para o nível lógico baixo (**0**), independentemente do valor anterior.
- **(3) 2'b10: q <= 1'b1;**
  - **Ação: Set**.
  - **Explicação:** Força a saída q para o nível lógico alto (**1**), independentemente do valor anterior.
- **(4) 2'b11: q <= ~q;**
  - **Ação: Toggle (Comutação)**.
  - **Explicação:** Inverte o estado atual da saída. Se q era 0, torna-se 1; se q era 1, torna-se 0.

b) Explique qual será o novo valor de q se q = 0, j = 1 e k = 0 em uma borda de subida.

- **Análise:** Com j=1 e k=0, a concatenação {j, k} resulta em 2'b10.
- **Código:** O case executa a linha (3): q <= 1'b1.
- **Resultado:** O novo valor de q será **1** (estado de Set).

c) Explique o que acontece com q se q = 1, j = 1 e k = 1.

- **Análise:** Com j=1 e k=1, a concatenação {j, k} resulta em 2'b11.
- **Código:** O case executa a linha (4): q <= ~q.
- **Cálculo:** O operador ~ é a negação bit a bit. Como o estado atual q é **1**, a negação será **0**.
- **Resultado:** O valor de q muda para **0** (acontece o *toggle*).

d) Explique o que garante que o reset é assíncrono neste código.

O que garante o comportamento assíncrono é a **lista de sensibilidade** do bloco always:

always @(posedge clk or posedge rst)

1. **Sensibilidade ao Reset:** O termo or posedge rst diz ao simulador (e à ferramenta de síntese) que este bloco de código deve ser executado **imediatamente** quando o sinal rst tiver uma borda de subida, **sem precisar esperar** pela borda do clk.
2. **Prioridade na Lógica:** Dentro do bloco, a condição if (rst) é verificada antes de qualquer outra coisa. Se rst for verdadeiro, ele zera a saída e ignora o restante do código (incluindo o clock e as entradas J/K).

Nota-se que o código usa atribuição não-bloqueante (<=)?

- q <= ... Isso é crucial em Verilog para lógica sequencial (Flip-Flops). Se o autor do código tivesse usado atribuição bloqueante (=), isso poderia criar *race conditions* (condições de corrida) na simulação e gerar hardware incorreto na síntese do FPGA. O código da questão está seguindo as boas práticas de RTL.

```
// Definição do Módulo: Flip-Flop JK com Reset Assíncrono
module jk_ff (
    input  wire clk, // Sinal de Clock
    input  wire j,   // Entrada J
    input  wire k,   // Entrada K
    input  wire rst, // Sinal de Reset (Reinicialização)
    output reg  q    // Saída Q (tipo 'reg' pois recebe valor dentro de um 'always')
);

    // Bloco Always: Define o comportamento sequencial
    // A lista de sensibilidade inclui "posedge rst", tornando o reset ASSÍNCRONO.
    // O bloco é ativado na subida do clock OU na subida do reset.
    always @(posedge clk or posedge rst) begin

        // 1. Verificação do Reset (Prioridade Máxima)
        if (rst) begin
            // Se o reset for alto, a saída vai para 0 IMEDIATAMENTE,
            // sem esperar o clock e ignorando J e K.
            q <= 1'b0;
        end

        // 2. Comportamento Síncrono (Funcionamento Normal)
        else begin
            // O comando 'case' analisa a concatenação das entradas J e K.
            // {j, k} cria um vetor de 2 bits. Ex: se j=1 e k=0, vira 2'b10.
            case ({j, k})

                // Caso J=0, K=0 -> Hold (Memória)
                2'b00: q <= q;    // Mantém o valor atual

                // Caso J=0, K=1 -> Reset
                2'b01: q <= 1'b0; // Força a saída para 0

                // Caso J=1, K=0 -> Set
                2'b10: q <= 1'b1; // Força a saída para 1

                // Caso J=1, K=1 -> Toggle (Comutação)
                2'b11: q <= ~q;   // Inverte o valor atual (0 vira 1, 1 vira 0)

            endcase
        end
    end

endmodule
```