APOSTILA DE COMANDOS

Máquinas de Estado Finito - Prática Progressiva

Disciplina: Organização de Computadores
Foco: Interpretação de comandos → Tabela de Estados → Código Verilog

---

INSTRUÇÕES GERAIS

Para cada questão, você deve:

1. **Interpretar o comando** e identificar entradas, saídas e comportamentos
2. **Criar a tabela de estados** completa com todas as transições
3. **Implementar o código Verilog estrutural** com flip-flops

Dica: Comece sempre identificando quantos estados você precisa, depois pense nas transições.

QUESTÃO 1 - Nível Básico

Comando:

Crie uma tabela verdade para um sistema com 2 entradas (START, STOP) e 1 saída (MOTOR), com as seguintes regras:

1. **MOTOR** só liga quando **START = 1**.
2. **MOTOR** permanece ligado até **STOP = 1**.
3. Se **START = 1** e **STOP = 1** simultaneamente, **STOP** tem prioridade.
4. Quando **START = 0** e **STOP = 0**, o sistema mantém o estado atual.

Sua tarefa:

- Monte a tabela de estados
- Implemente o código Verilog estrutural

QUESTÃO 2 - Nível Básico+

Comando:

Crie uma tabela verdade para um sistema com 2 entradas (COIN, RESET) e 2 saídas (LED1, LED2), com as seguintes regras:

1. **LED1** liga no primeiro pulso de **COIN = 1**.
2. **LED2** liga no segundo pulso de **COIN = 1**.
3. No terceiro pulso de **COIN = 1**, ambos LEDs desligam e o sistema volta ao estado inicial.
4. **RESET = 1** desliga tudo e volta ao estado inicial a qualquer momento.
5. Considere que **COIN** gera pulsos (0→1→0).

Sua tarefa:

- Monte a tabela de estados
- Implemente o código Verilog estrutural

QUESTÃO 3 - Nível Intermediário

Comando:

Crie uma tabela verdade para um sistema com 3 entradas (A, B, ENABLE) e 3 saídas (X, Y, Z), com as seguintes regras:

1. **X** liga quando **ENABLE = 1** e **A = 1**.
2. **Y** liga automaticamente sempre que **X = 1**, mas se **B = 0**, **Y** deve desligar imediatamente.
3. **Z** alterna seu estado (0→1 ou 1→0) sempre que **A = 1** e **B = 1** simultaneamente.
4. Se **ENABLE = 0**, todas as saídas ficam desligadas, independentemente de A e B.
5. **Z** mantém seu último valor quando não há alternância.

Sua tarefa:

- Monte a tabela de estados
- Implemente o código Verilog estrutural

QUESTÃO 4 - Nível Intermediário+

Comando:

Crie uma tabela verdade para um sistema de semáforo com 2 entradas (SENSOR_CARRO, BOTAO_PEDESTRE) e 3 saídas (VERDE, AMARELO, VERMELHO), com as seguintes regras:

1. **VERDE** fica ligado por padrão (estado inicial).
2. Quando **BOTAO_PEDESTRE = 1**, após 2 ciclos de clock, passa para **AMARELO**.
3. **AMARELO** fica ligado por exatamente 1 ciclo, depois passa para **VERMELHO**.
4. **VERMELHO** fica ligado por 3 ciclos, depois volta para **VERDE**.

5. Se **SENSOR_CARRO = 0** durante **VERMELHO**, prolonga o vermelho por mais 2 ciclos.
6. Apenas uma saída pode estar ativa por vez.

Sua tarefa:

- Monte a tabela de estados
- Implemente o código Verilog estrutural

QUESTÃO 5 - Nível Intermediário++

Comando:

Crie uma tabela verdade para um sistema com 2 entradas (CLOCK_EXTERNO, MODE) e 4 saídas (OUT1, OUT2, OUT3, OUT4), com as seguintes regras:

1. As saídas funcionam como um **contador em anel** (OUT1→OUT2→OUT3→OUT4→OUT1...).
2. A transição só acontece na **borda de subida** de **CLOCK_EXTERNO**.
3. Se **MODE = 0**: sequência normal (OUT1→OUT2→OUT3→OUT4).
4. Se **MODE = 1**: sequência reversa (OUT4→OUT3→OUT2→OUT1).
5. **MODE** pode mudar a qualquer momento, alterando imediatamente a direção da sequência.
6. Estado inicial: **OUT1 = 1**, demais saídas = 0.

Sua tarefa:

- Monte a tabela de estados
- Implemente o código Verilog estrutural

QUESTÃO 6 - Nível Avançado

Comando:

Crie uma tabela verdade para um sistema de controle com 3 entradas (TEMP_ALTA, PRESSAO_ALTA, EMERGENCIA) e 4 saídas (BOMBA, VALVULA, ALARME, STATUS[1:0]), com as seguintes regras:

1. **BOMBA** liga quando **TEMP_ALTA = 1** e permanece ligada por 4 ciclos.
2. **VALVULA** liga quando **PRESSAO_ALTA = 1**, mas só se **BOMBA** não estiver ativa.
3. **ALARME** liga quando **BOMBA** e **VALVULA** estão simultaneamente ativas.
4. **STATUS[1:0]** indica: 00=Normal, 01=Bomba ativa, 10=Válvula ativa, 11=Alarme.
5. **EMERGENCIA = 1** desliga tudo imediatamente e ativa **ALARME**.

6. Após **EMERGENCIA**, o sistema só volta ao normal quando todas as entradas estiverem em 0 por 2 ciclos consecutivos.

Sua tarefa:

- Monte a tabela de estados
- Implemente o código Verilog estrutural

QUESTÃO 7 - Nível Avançado+

Comando:

Crie uma tabela verdade para um sistema de máquina de estados com 3 entradas (DATA_IN, VALID, RESET) e 5 saídas (READY, BUSY, ERROR, DATA_OUT[1:0], COUNTER[2:0]), com as seguintes regras:

1. **READY = 1** no estado inicial, indicando que pode receber dados.
2. Quando **VALID = 1** e **READY = 1**, o sistema captura **DATA_IN** e vai para estado **BUSY**.
3. **BUSY** permanece ativo por 3 ciclos, processando os dados.
4. **DATA_OUT** recebe o valor de **DATA_IN** + 1 (módulo 4) após o processamento.
5. **COUNTER** conta de 0 a 7 continuamente, independente dos outros sinais.
6. **ERROR = 1** se **VALID = 1** quando o sistema já está **BUSY**.
7. **RESET = 1** limpa tudo: volta ao estado inicial, **COUNTER = 0**, **ERROR = 0**.
8. Após **BUSY**, o sistema volta automaticamente para **READY**.

Sua tarefa:

- Monte a tabela de estados
- Implemente o código Verilog estrutural

OBSERVAÇÕES IMPORTANTES

Dicas para Resolução:

1. **Identifique os estados necessários** antes de começar a tabela
2. **Pense nos contadores** quando há referência a "ciclos"
3. **Prioridades** devem ser respeitadas quando há conflitos
4. **Estados intermediários** podem ser necessários para temporização
5. **Reset/Emergência** geralmente têm prioridade máxima

Estrutura do Código Verilog:

```verilog
module nome(input clk, input res, input ..., output ...);
    // Declarações de fios e registradores
    wire [n:0] entradas;
    wire [m:0] prox_estado;
    reg [m:0] estado_atual;

    // Lógica combinacional para próximo estado
    assign prox_estado = ...;

    // Lógica das saídas
    assign saida = ...;

    // Flip-flop para armazenar estado
    always @(posedge clk or posedge res) begin
        if(res)
            estado_atual <= valor_inicial;
        else
            estado_atual <= prox_estado;
    end
endmodule
```

Verificação Final:

- Todas as combinações de entrada foram consideradas?
- As transições respeitam as regras de prioridade?
- O código implementa corretamente a tabela?
- Os tipos de dados (wire/reg) estão corretos?

BOA PRÁTICA E SUCESSOS NOS ESTUDOS!