Apostila de Exercícios - Máquinas de Estados Finitos (FSM)

Seção 1: Descrição → Tabela → Diagrama → Verilog

Questão 1.1

Uma máquina de controle para semáforo tem duas entradas: **SENSOR** (detecta veículos) e **TIMER** (sinaliza tempo decorrido). As saídas são **VERDE**, **AMARELO** e **VERMELHO**.

O comportamento é:

- Quando não há veículos (SENSOR=0), o semáforo permanece verde indefinidamente
- Quando há veículo detectado (SENSOR=1) e timer ativo (TIMER=1), alterna: verde → amarelo → vermelho → verde
- Com veículo detectado mas timer inativo, mantém estado atual
- Apenas uma saída pode estar ativa por vez

Fazer: diagrama de estados, tabela de transições e código Verilog.

Questão 1.2

Um controlador de acesso tem entradas **CARTAO** (cartão válido) e **SENSOR_PORTA** (porta aberta/fechada). As saídas são **TRAVA** (destrava=1) e **ALARME**.

Comportamento:

- Estado inicial: porta travada, sem alarme
- Cartão válido: destrava por 2 ciclos de clock, depois trava automaticamente
- Se porta não fechar em 2 ciclos após destravar: ativa alarme
- Alarme permanece até próximo cartão válido
- Sensor_porta=1 indica porta aberta

Fazer: diagrama de estados, tabela de transições e código Verilog.

Questão 1.3

Uma máquina de estados controla LEDs de um painel com entrada **MODO** (0=automático, 1=manual) e **BOTAO** (pulso do usuário). Saídas: **LED_A**, **LED_B**, **LED_C**.

Funcionamento:

- Modo automático: LEDs piscam em sequência A→B→C→A continuamente

- Modo manual: LEDs param na posição atual, botão avança um LED por vez na sequência
- Transição entre modos preserva estado atual dos LEDs
- Apenas um LED aceso por vez

Fazer: diagrama de estados, tabela de transições e código Verilog.

Questão 1.4

Um sistema de irrigação tem entrada **UMIDADE** (solo seco=0, úmido=1) e **ENABLE** (sistema ligado). Saídas: **BOMBA**, **ALERTA** e **STATUS**.

Lógica de operação:

- Sistema desabilitado: bomba desligada, status=0
- Solo úmido com sistema habilitado: bomba desligada, status=1
- Solo seco com sistema habilitado: bomba liga por 3 ciclos, depois para
- Se solo continuar seco após irrigação: ativa alerta permanente
- Alerta só desativa quando solo ficar úmido

Fazer: diagrama de estados, tabela de transições e código Verilog.

Questão 1.5

Um detector de sequência identifica a sequência "101" nas entradas **DATA** e **CLOCK_EN**. Saídas: **DETECTADO** e **RESET_OUT**.

Operação:

- Analisa entrada DATA apenas quando CLOCK_EN=1
- Ao detectar "101": ativa DETECTADO por 1 ciclo e gera RESET_OUT
- Sequências sobrepostas são válidas (ex: "1011" detecta em posições 0-2 e 1-3)
- RESET_OUT permanece ativo até próxima sequência válida
- Reinicia detecção após encontrar sequência

Fazer: diagrama de estados, tabela de transições e código Verilog.

Seção 2: Tabela → Diagrama → Verilog

Questão 2.1

Dada a tabela de transições abaixo, fazer o diagrama de estados e código Verilog:

Estado Atual	X Y	Próximo Estado	A B
00	00	00	0 1
00	01	01	0 1
00	10	10	0 1
00	11	00	0 1
01	00	01	1 0
01	01	10	1 0
01	10	00	1 0
01	11	01	1 0
10	00	10	0 0
10	01	00	0 0
10	10	01	0 0
10	11	10	0 0

Questão 2.2

Para a tabela de estados a seguir, elaborar diagrama e implementação Verilog:

Estado	P Q	Próximo	OUT1 OUT2 OUT3
S0	00	S0	1 0 0
S0	01	S1	1 0 0
S0	10	S2	1 0 0
S0	11	S0	1 0 0
S1	00	S2	0 1 0
S1	01	S1	0 1 0
S1	10	S0	0 1 0
S1	11	S3	0 1 0
S2	00	S0	0 0 1
S2	01	S3	0 0 1
S2	10	S2	0 0 1
S2	11	S1	0 0 1
S3	00	S3	1 1 1
S3	01	S0	1 1 1
S3	10	S1	1 1 1
S3	11	S2	1 1 1

Questão 2.3

Construir diagrama de estados e código Verilog para:

Estado Atual	EN CLK	Estado Seguinte	MOTOR FAN
IDLE	00	IDLE	0 0
IDLE	01	IDLE	0 0
IDLE	10	START	0 0
IDLE	11	START	0 0
START	00	IDLE	1 0
START	01	IDLE	1 0
START	10	RUN	1 0
START	11	RUN	1 0
RUN	00	STOP	1 1
RUN	01	STOP	1 1
RUN	10	RUN	1 1
RUN	11	RUN	1 1
STOP	00	IDLE	0 1
STOP	01	IDLE	0 1
STOP	10	IDLE	0 1
STOP	11	START	0 1

Questão 2.4

Implementar FSM baseada na tabela:

Current	A B	Next	LED1 LED2 BUZZ
Q0	00	Q0	0 0 0
Q0	01	Q1	0 0 0
Q0	10	Q2	0 0 0
Q0	11	Q1	0 0 0
Q1	00	Q2	1 0 1
Q1	01	Q1	1 0 1
Q1	10	Q0	1 0 1
Q1	11	Q2	1 0 1
Q2	00	Q0	0 1 0
Q2	01	Q0	0 1 0
Q2	10	Q1	0 1 0
Q2	11	Q2	0 1 0

Questão 2.5

Desenvolver diagrama e Verilog para:

Estado	IN1 IN2	Próximo	VALVE PUMP WARN
WAIT	00	WAIT	0 0 0
WAIT	01	FILL	0 0 0
WAIT	10	ERROR	0 0 0
WAIT	11	FILL	0 0 0
FILL	00	WAIT	1 1 0
FILL	01	FULL	1 1 0
FILL	10	ERROR	1 1 0
FILL	11	FULL	1 1 0
FULL	00	DRAIN	0 0 0
FULL	01	FULL	0 0 0
FULL	10	ERROR	0 0 0
FULL	11	DRAIN	0 0 0
DRAIN	00	WAIT	0 1 0
DRAIN	01	DRAIN	0 1 0
DRAIN	10	ERROR	0 1 0
DRAIN	11	DRAIN	0 1 0
ERROR	00	WAIT	0 0 1
ERROR	01	ERROR	0 0 1
ERROR	10	ERROR	0 0 1
ERROR	11	ERROR	0 0 1

Seção 3: Verilog → Tabela → Diagrama

Questão 3.1

Analisar o código Verilog abaixo e construir a tabela de transições e diagrama de estados:

```verilog
```

```
module fsm_control(input clk, input rst, input enable, input mode, output reg lamp, output reg heater);
    wire [1:0] state;
    assign state = {enable, mode};
    assign lamp = enable & mode;
    assign heater = enable & ~mode;

    reg [1:0] next_state;
    always @(posedge clk or posedge rst) begin
        if (rst)
            next_state <= 2'b00;
        else
            next_state <= state;
    end
endmodule
```

Questão 3.2

Para o código seguinte, elaborar tabela de estados e diagrama:

verilog

```
module sequence_det(input clk, input rst, input data_in, output reg found, output reg active);
    reg [1:0] current_state, next_state;
    parameter S0=2'b00, S1=2'b01, S2=2'b10, S3=2'b11;

    always @(posedge clk or posedge rst) begin
        if (rst) current_state <= S0;
        else current_state <= next_state;
    end

    always @(*) begin
        case (current_state)
            S0: if (data_in) next_state = S1; else next_state = S0;
            S1: if (data_in) next_state = S2; else next_state = S0;
            S2: if (~data_in) next_state = S3; else next_state = S2;
            S3: if (~data_in) next_state = S0; else next_state = S1;
        endcase
    end

    assign found = (current_state == S3);
    assign active = (current_state != S0);
endmodule
```

Questão 3.3

Extrair tabela e diagrama do código Verilog:

verilog

```verilog
module traffic_light(input clk, input rst, input sensor, output reg [1:0] lights);
    reg [1:0] state;
    parameter GREEN=2'b00, YELLOW=2'b01, RED=2'b10, FLASH=2'b11;

    always @(posedge clk or posedge rst) begin
        if (rst)
            state <= GREEN;
        else begin
            case (state)
                GREEN: if (sensor) state <= YELLOW; else state <= GREEN;
                YELLOW: state <= RED;
                RED: if (~sensor) state <= FLASH; else state <= RED;
                FLASH: state <= GREEN;
            endcase
        end
    end

    always @(*) begin
        case (state)
            GREEN: lights = 2'b01;
            YELLOW: lights = 2'b10;
            RED: lights = 2'b11;
            FLASH: lights = 2'b00;
        endcase
    end
endmodule
```

Questão 3.4

Construir tabela de transições e diagrama para:

verilog

```verilog
module door_controller(input clk, input rst, input key_valid, input door_sensor, output reg lock, output reg alarm);
    reg [2:0] state, next_state;
    parameter LOCKED=3'b000, UNLOCK1=3'b001, UNLOCK2=3'b010, OPEN=3'b011, ALARM=3'b100;

    always @(posedge clk or posedge rst) begin
        if (rst) state <= LOCKED;
        else state <= next_state;
    end

    always @(*) begin
        case (state)
            LOCKED: if (key_valid) next_state = UNLOCK1; else next_state = LOCKED;
            UNLOCK1: next_state = UNLOCK2;
            UNLOCK2: if (door_sensor) next_state = OPEN; else next_state = ALARM;
            OPEN: if (~door_sensor) next_state = LOCKED; else next_state = OPEN;
            ALARM: if (key_valid) next_state = LOCKED; else next_state = ALARM;
            default: next_state = LOCKED;
        endcase
    end

    assign lock = (state == LOCKED);
    assign alarm = (state == ALARM);
endmodule
```

Questão 3.5

Analisar e extrair comportamento do código:

verilog

```
module counter_fsm(input clk, input rst, input up_down, input enable, output reg [1:0] count, output reg overflow);
    reg [1:0] state;

    always @(posedge clk or posedge rst) begin
        if (rst)
            state <= 2'b00;
        else if (enable) begin
            if (up_down) begin
                if (state == 2'b11) state <= 2'b00;
                else state <= state + 1;
            end else begin
                if (state == 2'b00) state <= 2'b11;
                else state <= state - 1;
            end
        end
    end

    assign count = state;
    assign overflow = enable & ((up_down & state == 2'b11) | (~up_down & state == 2'b00));
endmodule
```