SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE VIÇOSA · UFV
CAMPUS FLORESTAL

Trabalho 2 - ISL

Máquina de Estado Finito

INGRED FONSECA DE ALMEIDA [EF 04691]
MARIA CLARA BRAGA ARAÚJO VIANA [EF04667]
MIGUEL RIBEIRO [EF04680]
RAFAELA TOLEDO DOLABELLA [EF04665]

Florestal - MG
2022

Sumário

1. Introdução **3**

2. Desenvolvimento **6**

2.1 tp.v 6

2.2 README.txt 4

3. Resultados e Conclusão **12**

1. Introdução

O projeto consiste em descrever um hardware usando **Verilog** de uma Máquina de Estados de Moore. A máquina terá entradas de 8 bits, como mostra a imagem a seguir:

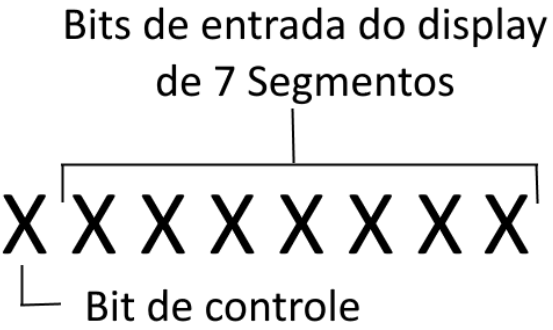

Figura 1 - Detalhamento da entrada.

Um bit de controle, que caso 0 não envia a informação para frente e caso 1 siga para o próximo estado e os outros 7 bits são uma decodificação de um diplay de 7 segmentos (Figura 2)

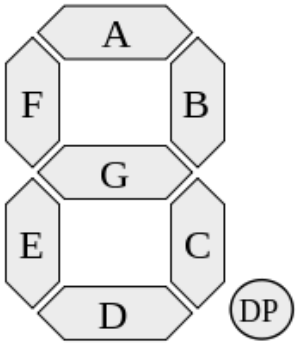

Figura 2 - Modelo display de 7 segmentos.

Assim, cada grupo recebeu uma tabela com 8 entradas (Figura 3). Sendo as 5 primeiras de estados intermediários da nossa máquina de estado e as outras 3 de estados finais.

Grupo	C1	C2	C3	C4	C5	C6	C7	C8
3								
	0100000	0000100	0111100	0011010	0101110	0001001	0110101	0010011

Figura 3 - Decodificação das entradas para cada estado do display para binário.

Foi descrito que o bloco inicial poderia seguir para qualquer estado intermediário, dado na primeira entrada. Essa entrada poderia ser corrigida, porém esses estados intermediários só poderiam passar para o estado anterior ou próximo (caso existam), caso contrário seguiria para o estado final de saída inválida. A segunda entrada deveria ser um dos estados finais, os três primeiros estados só poderiam seguir para a saída 1 e os dois últimos para a saída 2, e todos para a saída inválida, seja pela entrada sendo a decodificação do display dessa saída ou com entradas descritas anteriormente. Para

visualização desse processo foi feito um diagrama (Figura 4).

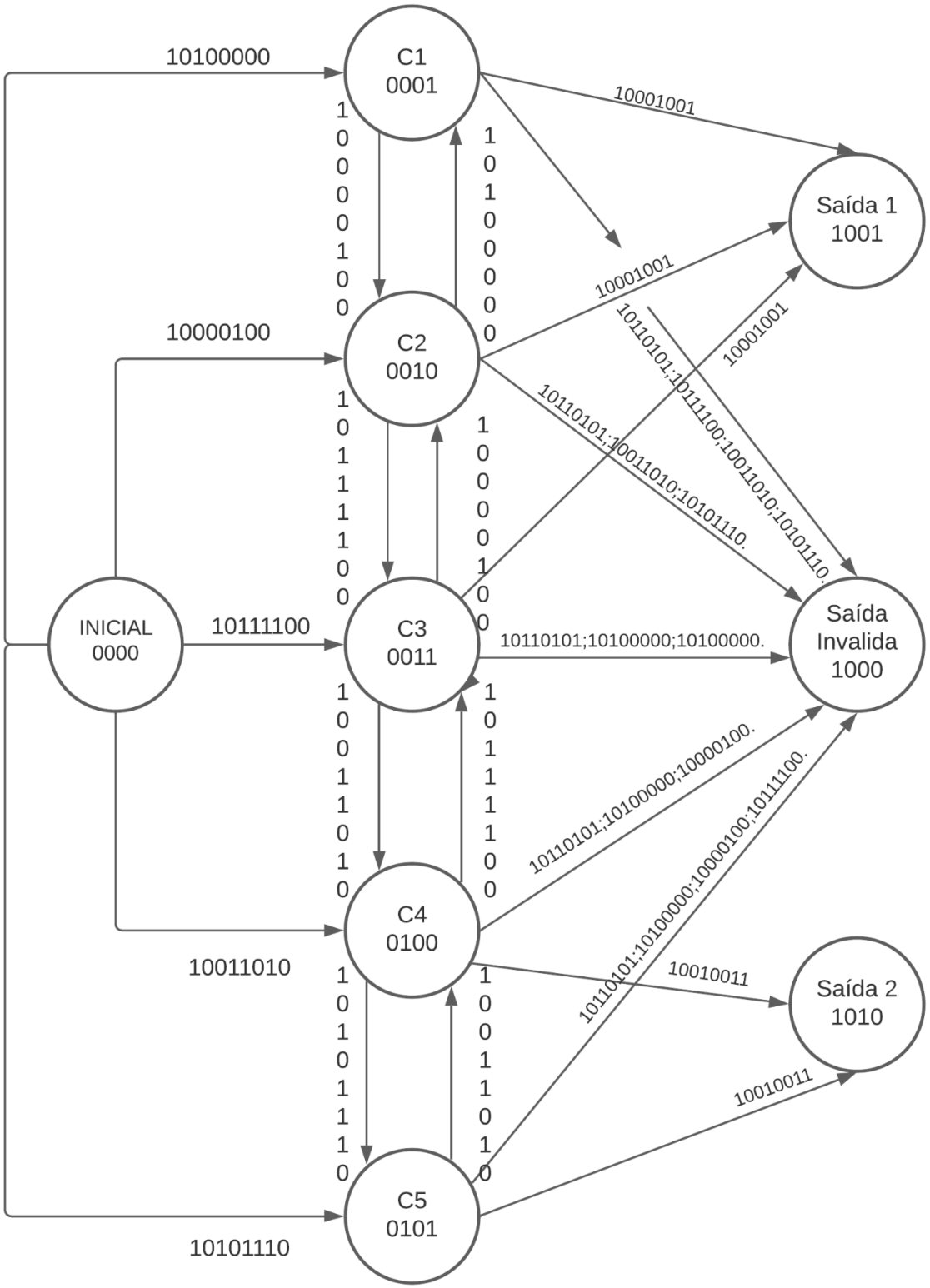

Figura 4 - Diagrama da máquina de estados (feito pela plataforma Lucid).

A cada passo da máquina deverá ter uma saída no terminal seguindo a Tabela 1.

Saída	Estado
0000	Estado inicial
0001	Estado 1 (Letra 1)
0010	Estado 2 (Letra 2)
0011	Estado 3 (Letra 3)
0100	Estado 4 (Letra 4)
0101	Estado 5 (Letra 5)
1000	Mensagem Falsa, Invalido
1001	Saída 1, Ação
1010	Saída 2, Horário

Tabela 1 - Valores de saída

2. Desenvolvimento

2.1 tp.v

Primeiramente, é iniciado o módulo nomeado Maquina_estados, onde declaramos as variáveis entrada, clock, reset e saída. Em seguida, define-se os outputs e inputs com as variáveis anteriormente declaradas. Clock, reset e entrada são inputs, sendo a entrada de 8bits, enquanto a saída é um output de 4bits. Abaixo, são declarados os parâmetros onde se atribui o valor dos estados, sendo eles: inicial, C1, C2, C3, C4, C5, saida1, saida2 e saida_invalida, depois setamos o estado atual e o proximo_estado como sendo dois registradores de 8bits.

```
module Maquina_estados(entrada,clock,reset,saida);
    //Inputs e Outputs
    input clock;
    input reset;
    input[7:0] entrada;
    output reg[3:0] saida;

    //Valor dos estados
    parameter
            Inicial =        0,
            C1 =             1,
            C2 =             2,
            C3 =             3,
            C4 =             4,
            C5 =             5,
            Saida1 =         6,
            Saida2 =         8,
            Saida_invalida = 7;

    //Declara o estado atual e próximo
    reg  [7:0] estado_atual;
    reg  [7:0] proximo_estado;
```

Figura 5 - Parte 1 do código: Declaração de variáveis.

Em seguida, iniciamos um bloco always onde usamos a instrução case que verifica se a expressão fornecida corresponde a uma entre as outras expressões dentro da lista e das ramificações, e então especificamos todos os casos de entrada e verificamos se correspondem a expressão. Fazemos as verificações para cada possível caso de entrada, e para construir este case nos baseamos no diagrama (Figura 4), onde seguimos as linhas e analisamos os possíveis caminhos.

```
always @(estado_atual , entrada)
    begin
        proximo_estado = estado_atual;
    case(estado_atual)
    Inicial:begin
        if(entrada==8'b10100000)
            proximo_estado = C1;
        if(entrada==8'b10000100)
            proximo_estado = C2;
        if(entrada==8'b10111100)
            proximo_estado = C3;
        if(entrada==8'b10011010)
            proximo_estado = C4;
        if (entrada==8'b10101110)
            proximo_estado = C5;
    end
    C1:begin

        if(entrada==8'b10000100)
            proximo_estado = C2;
        if(entrada==8'b10111100)
            proximo_estado = Saida_invalida; //C3
        if(entrada==8'b10011010)
            proximo_estado = Saida_invalida; //C4
        if(entrada==8'b10101110)
            proximo_estado = Saida_invalida; //C5
        if(entrada==8'b10110101)
            proximo_estado = Saida_invalida;
        if(entrada==8'b10001001)
            proximo_estado = Saida1;
    end
    C2:begin
```

Figura 6 - Parte 2 do código: Bloco always (estados Inicial e C1).

```
C2:begin

    if(entrada==8'b10111100)
        proximo_estado = C3;
    if(entrada==8'b10100000)
        proximo_estado = C1;
    if(entrada==8'b10011010)
        proximo_estado = Saida_invalida; //C4
    if(entrada==8'b10101110)
        proximo_estado = Saida_invalida; //C5
    if(entrada==8'b10110101)
        proximo_estado = Saida_invalida;
    if(entrada==8'b10001001)
        proximo_estado = Saida1;
    end
C3:begin
    if (entrada==8'b10100000)
        proximo_estado = Saida_invalida; // C1
    if(entrada==8'b10000100)
        proximo_estado = C2;
    if(entrada==8'b10011010)
        proximo_estado=C4;
    if(entrada==8'b10101110)
        proximo_estado= Saida_invalida;  // C5
    if(entrada==8'b10001001)
        proximo_estado = Saida1;
    if(entrada==8'b10110101)
        proximo_estado=Saida_invalida;
    end
```

Figura 7 - Parte 3 do código: Bloco always (estados C2 e C3).

```
    C4:begin

        if(entrada==8'b10100000)
            proximo_estado=Saida_invalida; //C1
        if(entrada==8'b10000100)
            proximo_estado=Saida_invalida; //C2
        if(entrada==8'b10111100)
            proximo_estado=C3;
        if(entrada==8'b10101110)
            proximo_estado=C5;
        if(entrada==8'b10010011)
            proximo_estado=Saida2;
        if(entrada==8'b10110101)
            proximo_estado=Saida_invalida;
        end
    C5:begin

        if(entrada==8'b10100000)
            proximo_estado=Saida_invalida; //C1
        if(entrada==8'b10000100)
            proximo_estado=Saida_invalida; //C2
        if(entrada==8'b10111100)
            proximo_estado=Saida_invalida; //C3
        if(entrada==8'b10011010)
            proximo_estado=C4;
        if(entrada==8'b10010011)
            proximo_estado=Saida2;
        if(entrada==8'b10110101)
            proximo_estado=Saida_invalida;
        end
    endcase
end
```

Figura 8 - Parte 4 do código: Bloco always (estados C4 e C5).

Por fim, iniciamos o module do testbench, onde declaramos os registradores e fios usados no teste e chamamos a função Maquina_estados passando os parâmetros anteriormente declarados. Na imagem abaixo é possível ver que o bloco initial begin onde o programa percorre os estados. E por fim, mostramos no terminal os valores de entrada e saída de cada caso.

```
module testbench; //testbench
 // Inputs
 reg [7:0]entrada;
 reg clock;
 reg reset;
 // Outputs
 wire [3:0]saida;

 Maquina_estados uut (.entrada(entrada), .clock(clock), .reset(reset), .saida(saida)); //chamada da função

 initial begin //funcionamento do clock
    clock = 0;
    forever #10 clock = ~clock;
 end

 initial begin
    $display("\nInicial => C1 => C2 => C3 => C4 => C5 => C4 => C3 => C2 => C1 => Saida1"); //entradas
    entrada = 0;
    reset = 1; //reseta a máquina
    #10; reset = 0; //começa a percorrer os estados
    #20; entrada = 8'b10100000; //C1
    #20; entrada = 8'b10000100; //C2
    #20; entrada = 8'b10111100; //C3
    #20; entrada = 8'b10011010; //C4
    #20; entrada = 8'b10101110; //C5
    #20; entrada = 8'b10011010; //C4
    #20; entrada = 8'b10111100; //C3
    #20; entrada = 8'b10000100; //C2
    #20; entrada = 8'b10100000; //C1
    #20; entrada = 8'b10001001; //Saida1
    #20; $finish;
```

Figura 9 - Parte 5 do código: Testbech.

```
 initial begin
            $display("====================="); //imprime as saídas de cada estado no terminal
            $display("| ENTRADA  || SAIDA |");
            $monitor("| %8b ||  %4b |",entrada,saida);
    end
endmodule
```

Figura 10 - Parte 6 do código: Comando para saída.

2.2 README.txt

Neste arquivo de texto, especificamos os comandos necessários para compilar o arquivo tp1 e executar para que seja possível visualizar as saídas no terminal.

README.txt

tp02 > README.txt

```
1   Comando para compilar o arquivo:
2       iverilog -o tp2 tp2.v
3
4   Comando para rodar o arquivo:
5       vvp tp2
```

Figura 11- Comandos para compilação.

4. Resultados e Conclusão

Após rodar o código, com três versões de entradas diferentes, temos as tabelas a seguir (Figura 11, Figura 12 e Figura 13) que é printada no terminal a partir do código da Figura 10, que mostra o caminho de estados percorridos, a entrada e por fim a saída.

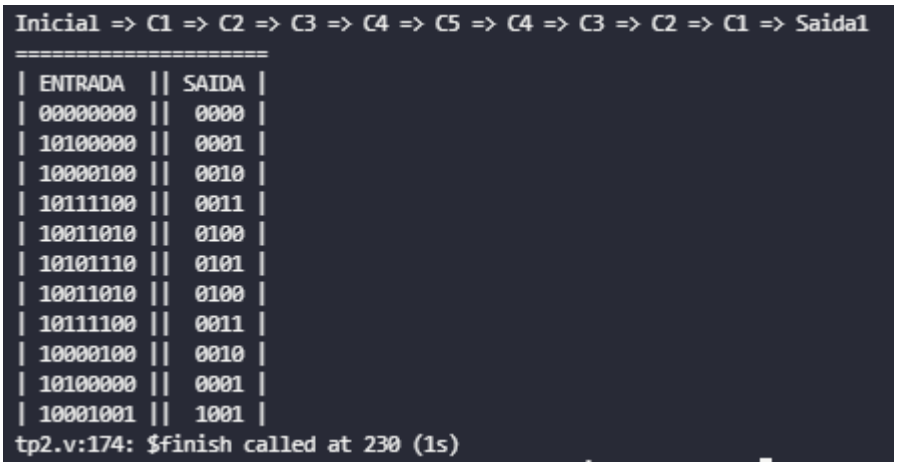

Figura 11- Saída terminal Saída1.

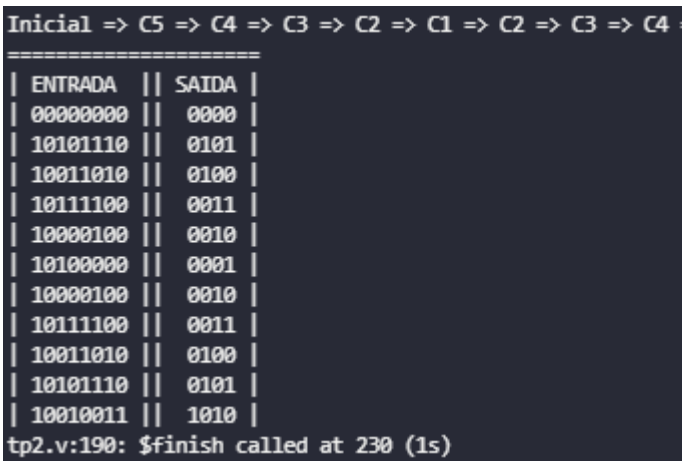

Figura 12- Saída terminal Saída2.

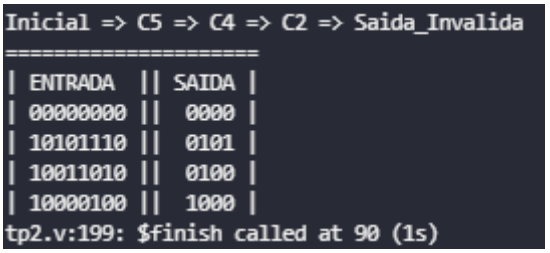

Figura 13- Saída terminal Saída_Invalida.