Relatório de Labolatório de Sistemas Digitais Avançados

Prática 7 e 8

Alunos:

José David Sousa de Araújo– Matricula: 1020678

Judah Holanda Correia Lima – Matricula: 1010500/5

José Kilson de Souza Oliveira – Matricula: 0810847/1

Sumário

[Material Utilizado 3](#_Toc387753691)

[Prática 7 e 8 4](#_Toc387753692)

[INTRODUÇÃO 4](#_Toc387753693)

[Códigos Auxiliares 4](#_Toc387753694)

[Módulo Removedor de Bounce 4](#_Toc387753695)

[Módulo Codificador para o Display de 7 Segmentos BCD (B2D) 5](#_Toc387753696)

[Parte 1 (Exercício 7) 5](#_Toc387753697)

[Módulo Principal do Projeto 6](#_Toc387753698)

[Parte 2 (Exercício 7) 7](#_Toc387753699)

[Módulo Principal do Projeto 7](#_Toc387753700)

[Parte 1 (Exercício 8) 9](#_Toc387753701)

[Módulo Principal do Projeto 9](#_Toc387753702)

[Parte 2 (Exercício 8) 11](#_Toc387753703)

[Módulo Principal do Projeto 11](#_Toc387753704)

[Conclusão 12](#_Toc387753705)

# Material Utilizado

* Kit de desenvolvimento DE0
* Software Quartus2 13.1 fornecido pelo fabricante Altera
* Sistema operacional: Windows 7 64 bits Serve Pack 1 e Windows 8.1 64 bits Serve Pack 1
* Estação de trabalho: Notebooks Core i7 Segunda Geração 4 Gb de Ram (Windows 7) e Core i7 Segunda Geração 8 Gb de Ram (Windows 8.1)

# Prática 7 e 8

## INTRODUÇÃO

A tarefas executadas em um computador precisam receber entradas e fornecer saídas de acordo com a entrada fornecida, contudo o computador atualmente não suporta qualquer entrada, elas são definidas no momento do planejamento da arquitetura.

O projeto deve prevê os sinais a serem suportados e os estados a serem estacionados, para isso deve ser implementada uma lógica sequencial a partir dos sinais recebidos, geralmente isso constitui a implementação de uma máquina de estado. Então esta máquina deve armazenar seu estado e sinais de entrada para poder executar suas operações, para isso é necessário a incorporação de uma memória, dependendo da tecnologia escolhida a partir da necessidade do projeto.

Para esta prática foram propostos os exercício 7 e 8 fornecidos pela altera, mas especificamente os dois primeiros itens de cada um. Estas práticas tem como finalidade o estudo e implementação de uma memória convencional, baseada na tecnologia da Altera.

A descrição de hardware desta prática foi feita em Verilog no Kit de desenvolvimento da empresa DE0, fornecido pela Unifor.

## Códigos Auxiliares

### Módulo Removedor de Bounce

Esse módulo tem como finalidade a retirada do bounce de alguma chave do circuito.

//Registrador para tratar o Bounce

module DeBounce (En, Clk, Clr, Q);

//Sinais de entrada: Habilitador, clock,clear

input En, Clk, Clr;

//registrador

output reg Q;

//Aciona pela borda de subida do clock e

//verifica se está habilitado para setar a saída em 1

//ou clear em 0 para setar a saída em 0

always @ (posedge Clk)

begin

if (~Clr)

Q = 0;

else if (En)

Q = ~Q;

end

endmodule

### Módulo Codificador para o Display de 7 Segmentos BCD (B2D)

module b2d\_ssd (X, SSD);

input [4:0] X; //bits de entrada

output reg [0:6] SSD; //bits de saída para o display

//Verifica o sinais de entrada e

//seta a saída de acordo com o BCD.

always begin

case(X)

0:SSD=7'b0000001;

1:SSD=7'b1001111;

2:SSD=7'b0010010;

3:SSD=7'b0000110;

4:SSD=7'b1001100;

5:SSD=7'b0100100;

6:SSD=7'b0100000;

7:SSD=7'b0001111;

8:SSD=7'b0000000;

9:SSD=7'b0001100;

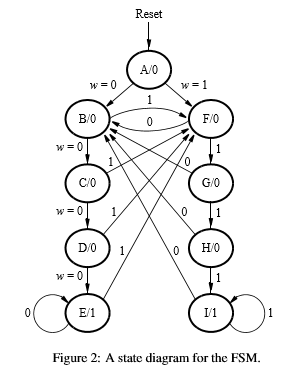
endcase

end

endmodule

## Parte 1 (Exercício 7)

Este exercício tem como finalidade a implementação de uma máquina de estado de acordo com a tabela de sinais especificados no exercício fornecido pela Altera, o qual é mostrado abaixo:



### Módulo Principal do Projeto

//Definição do módulo principal//

module part1 (SW, LEDG);

// Sinais de entrada e saída em que o SW representas as chaves e o LEDG os leds

input [9:0] SW;

output [9:0] LEDG;

//Sinais auxiliares

wire [8:0] D, Y;

wire w;

assign w = SW[0];

//chamada do modulo da máquina, os sinais de entrada e saída são passados através de parâmetros

my\_lpm\_ff FF (~SW[2], SW[1], D[8:0], Y[8:0]);

//Atribui do sinais de saída da máquina de estado ao auxiliar W

assign D[0] = ~(~(~Y[0] | Y[1] | Y[2] | Y[3] | Y[4] | Y[5] | Y[6] | Y[7] | Y[8]) | ~(D[1] | D[2] | D[3] | D[4] | D[5] | D[6] | D[7] | D[8]));

assign D[1] = (~Y[0] | Y[5] | Y[6] | Y[7] | Y[8]) & ~w; // B

assign D[2] = Y[1] & ~w; // C

assign D[3] = Y[2] & ~w; // D

assign D[4] = (Y[3] | Y[4]) & ~w; // E

assign D[5] = (~Y[0] | Y[1] | Y[2] | Y[3] | Y[4]) & w; // F

assign D[6] = Y[5] & w; // G

assign D[7] = Y[6] & w; // H

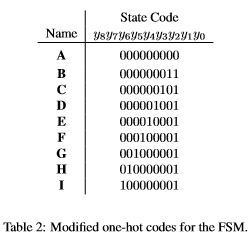
assign D[8] = (Y[7] | Y[8]) & w; // I

//Seta do leds com o valor da máquina de estado

assign LEDG[9] = Y[0];

assign LEDG[8:0] = D;

endmodule

Os testes foram realizados por sinais produzidos pelas chaves e as saídas verificadas através da tabela fornecida na prática, o qual é mostrado abaixo:

Quando aplicado o referido sinal a saída apresentou-se como esperado, mesmo quando um sinal diferente é aplicada no decorrido estado, então pode-se concluir que a implementação foi bem sucedida.

## Parte 2 (Exercício 7)

Este exercício tem como finalidade a implementação de uma máquina de estado de acordo com a tabela de sinais especificados anteriormente discutida.

Neste exercício foram utilizados os ”statements”: “case” e “always”.

### Módulo Principal do Projeto

Em “module **part2** ();” é feita a declaração do módulo da lógica do circuito, como os sinais nele existentes.

module part2 (SW, LEDG);

// Sinais de entrada e saída em que o SW representas as chaves e o LEDG os leds

input [9:0] SW;

output [9:0] LEDG;

//sinal auxiliar para o clock

wire w;

assign w = SW[1];

//sinal auxiliar para o clock

wire Clock;

assign Clock = SW[0];

//registrador

reg z;

//y\_Q representa o estado atual enquanto Y\_D representa o próximo estado

reg [3:0] y\_Q, Y\_D;

parameter A = 4'b0000, B = 4'b0001, C = 4'b0010, D = 4'b0011, E = 4'b0100, F = 4'b0101, G = 4'b0110, H = 4'b0111, I = 4'b1000;

//Verifica o sinais de entrada e seta as saída de acordo com a tabela fornecida na prática

always @(w, y\_Q)

begin: state\_table

case (y\_Q)

A: if (!w) Y\_D = B;

else Y\_D = F;

B: if (!w) Y\_D = C;

else Y\_D = F;

C: if (!w) Y\_D = D;

else Y\_D = F;

D: if (!w) Y\_D = E;

else Y\_D = F;

E: if (!w) Y\_D = E;

else Y\_D = F;

F: if (w) Y\_D = G;

else Y\_D = B;

G: if (w) Y\_D = H;

else Y\_D = B;

H: if (w) Y\_D = I;

else Y\_D = B;

I: if (w) Y\_D = I;

else Y\_D = B;

default: Y\_D = 4'bxxxx;

endcase

end

always @(posedge Clock)

begin: state\_FFs

y\_Q = Y\_D;

end // state\_FFS

//Verifica o estado atual para produzir novos sinais

always

begin: zset

case (y\_Q)

E: z = 1;

I: z = 1;

default: z = 0;

endcase

end

//Seta os LED’s com estados salvos

assign LEDG[3:0] = y\_Q;

assign LEDG[7:4] = Y\_D;

assign LEDG[9] = z;

test t1 (SW[4], SW[3], LEDG[8]);

endmodule

//Módulo de teste para os estados

module test (A, B, Out);

input A, B;

output reg Out;

always

case (A)

0: Out = 0;

1: Out = B;

endcase

endmodule

Os testes foram realizados por sinais produzidos pelas chaves e as saídas verificadas através da tabela fornecida na prática, quando aplicado o referido sinal a saída apresentou-se como esperado, mesmo quando um sinal diferente é aplicada no decorrido estado, então pode-se concluir que a implementação foi bem sucedida.

## Parte 1 (Exercício 8)

Esta prática tem como finalidade a implementação de uma API de memória utilizando a megawizard function da Altera.

### Módulo Principal do Projeto

//Módulo produzido pela megawizard da altera

//Apresenta sinais: Endereço, clock, dados, habilita escrita e leitura, saída

module ramlpm16bits4 (

address,

clock,

data,

wren,

q);

input [3:0] address;

input clock;

input [15:0] data;

input wren;

output [15:0] q;

`ifndef ALTERA\_RESERVED\_QIS

// synopsys translate\_off

`endif

tri1 clock;

`ifndef ALTERA\_RESERVED\_QIS

// synopsys translate\_on

`endif

wire [15:0] sub\_wire0;

wire [15:0] q = sub\_wire0[15:0];

altsyncram altsyncram\_component (

.address\_a (address),

.clock0 (clock),

.data\_a (data),

.wren\_a (wren),

.q\_a (sub\_wire0),

.aclr0 (1'b0),

.aclr1 (1'b0),

.address\_b (1'b1),

.addressstall\_a (1'b0),

.addressstall\_b (1'b0),

.byteena\_a (1'b1),

.byteena\_b (1'b1),

.clock1 (1'b1),

.clocken0 (1'b1),

.clocken1 (1'b1),

.clocken2 (1'b1),

.clocken3 (1'b1),

.data\_b (1'b1),

.eccstatus (),

.q\_b (),

.rden\_a (1'b1),

.rden\_b (1'b1),

.wren\_b (1'b0));

defparam

altsyncram\_component.clock\_enable\_input\_a = "BYPASS",

altsyncram\_component.clock\_enable\_output\_a = "BYPASS",

altsyncram\_component.intended\_device\_family = "Cyclone III",

altsyncram\_component.lpm\_hint = "ENABLE\_RUNTIME\_MOD=NO",

altsyncram\_component.lpm\_type = "altsyncram",

altsyncram\_component.numwords\_a = 16,

altsyncram\_component.operation\_mode = "SINGLE\_PORT",

altsyncram\_component.outdata\_aclr\_a = "NONE",

altsyncram\_component.outdata\_reg\_a = "UNREGISTERED",

altsyncram\_component.power\_up\_uninitialized = "FALSE",

altsyncram\_component.ram\_block\_type = "M9K",

altsyncram\_component.read\_during\_write\_mode\_port\_a = "NEW\_DATA\_NO\_NBE\_READ",

altsyncram\_component.widthad\_a = 4,

altsyncram\_component.width\_a = 16,

altsyncram\_component.width\_byteena\_a = 1;

endmodule

Não foram realizados testes, pois este módulo é utilizado na próxima prática.

## Parte 2 (Exercício 8)

Esta prática tem como finalidade, utilizar a memória produzida pela exercício 8 prática 1.

### Módulo Principal do Projeto

Em “module **part2** ()é feita a declaração do módulo da lógica do circuito, como os sinais nele existentes.

Os conteúdos do código Morse vão ser exibidos no LED verde.

module part2 (SW, LEDG, HEX0);

//Sinais de entrada e saída em que o SW representas as chaves e o LEDG os leds

input [9:0] SW;

output [9:0] LEDG;

output [6:0] HEX0;

//Sinais auxiliaries: Endereço, clock, dados, habilita escrita e leitura, saída.

wire [9:0] din, dout;

wire [2:0] addr;

wire wren, clock;

//sinal auxiliar de saída de bounce

wire Bt;

//chama modulo do bounce

DeBounce D0(1, SW[9], SW[8], Bt);

assign addr = SW[6:3];

assign clock = Bt;

assign din = SW[2:0];

assign wren = SW[7];

//assign LEDG[9:0] = dout[9:0];

//chamada da api da memória

ramlpm16bits4 R0 (addr, clock, din, wren, dout);

//chamada do display

hexD H0(dout, HEX0);

endmodule

Os teste básicos foram realizados aplicando os sinais para o endereço e dados, depois de estados, foi dando um pulso pelo clock, a saída da memória apresentou-se com o mesmo sinal de entrada de dadas, fornecendo resultados satisfatórios.

# Conclusão

Analisando os resultados descritos em cada prática, percebe-se que as práticas foram feitas de acordo com as restrições presentes em sua descrição, obtendo as respostas esperadas dentro de seus parâmetros. As práticas fornecerem um aprendizado básico de implementação em Verilog de máquinas de estado e memória que serão aplicas num contexto mais prático em práticas futuras.