Relatório de Labolatório de Sistemas Digitais Avançados

Prática 10

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 9 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 10

## Códigos Auxiliares

### Módulo do processador

module proc (DIN, Resetn, Clock, Run, Done, BusWires, ADDR, DOUT, W, Tstep\_Q, R0);

input [15:0] DIN;

input Resetn, Clock, Run;

output reg Done;

output reg [15:0] BusWires;

output [15:0] ADDR, DOUT;

output W;

output [2:0] Tstep\_Q;

output [15:0] R0;

//declare variables

reg IRin, DINout, Ain, Gout, Gin, AddSub, incr\_pc, ADDRin, DOUTin, W\_D;

reg [7:0] Rout, Rin;

wire [7:0] Xreg, Yreg;

wire [1:9] IR;

wire [1:3] I;

reg [9:0] MUXsel;

wire [15:0] R0, R1, R2, R3, R4, R5, R6, R7, result;

wire [15:0] A, G;

wire [2:0] Tstep\_Q;

wire Clear = Done | ~Resetn;

upcount Tstep (Clear, Clock, Tstep\_Q);

assign I = IR[1:3];

dec3to8 decX (IR[4:6], 1'b1, Xreg);

dec3to8 decY (IR[7:9], 1'b1, Yreg);

always @(Tstep\_Q or I or Xreg or Yreg)

begin

//specify initial values

IRin = 1'b0;

Rout[7:0] = 8'b00000000;

Rin[7:0] = 8'b00000000;

DINout = 1'b0;

Ain = 1'b0;

Gout = 1'b0;

Gin = 1'b0;

AddSub = 1'b0;

DOUTin = 1'b0;

ADDRin = 1'b0;

W\_D = 1'b0;

incr\_pc = 1'b0;

Done = 1'b0;

case (Tstep\_Q)

3'b000: // load next instruction in time step 0

begin

Rout = 8'b00000001;

ADDRin = 1'b1;

incr\_pc = 1'b1;

end

3'b001: // store next instruction in time step 1

begin

IRin = 1'b1; // should this be ANDed with Run?

ADDRin = 1'b1;

end

3'b010: //define signals in time step 1

case (I)

3'b000: // mv

begin

Rout = Yreg;

Rin = Xreg;

Done = 1'b1;

end

3'b001: // mvi

begin

DINout = 1'b1;

Rin = Xreg;

Done = 1'b1;

incr\_pc = 1'b1;

end

3'b010: // add

begin

Rout = Xreg;

Ain = 1'b1;

end

3'b011: // sub

begin

Rout = Xreg;

Ain = 1'b1;

end

3'b100: // ld

begin

Rout = Yreg;

ADDRin = 1'b1;

end

3'b101: // st

begin

Rout = Xreg;

DOUTin = 1'b1;

end

3'b110: // mvnz

begin

if (G != 0) begin

Rout = Yreg;

Rin = Xreg;

end

Done = 1'b1;

end

endcase

3'b011: //define signals in time step 2

case (I)

3'b010: // add

begin

Rout = Yreg;

Gin = 1'b1;

end

3'b011: // sub

begin

Rout = Yreg;

Gin = 1'b1;

AddSub = 1'b1;

end

3'b100: // ld

begin

DINout = 1'b1;

Rin = Xreg;

Done = 1'b1;

end

3'b101: // st

begin

Rout = Yreg;

ADDRin = 1'b1;

W\_D = 1'b1;

end

endcase

3'b100: //define signals in time step 3

case (I)

3'b010: // add

begin

Gout = 1'b1;

Rin = Xreg;

Done = 1'b1;

end

3'b011: // sub

begin

Gout = 1'b1;

Rin = Xreg;

Done = 1'b1;

end

endcase

endcase

end

//instantiate registers and the adder/subtracter unit

//regn reg\_0 (BusWires, Rin[0], Clock, R0);

counterlpm reg\_0 (1'b1, Clock, incr\_pc, BusWires, ~Resetn, Rin[0], R0);

regn reg\_1 (BusWires, Rin[1], Clock, R1);

regn reg\_2 (BusWires, Rin[2], Clock, R2);

regn reg\_3 (BusWires, Rin[3], Clock, R3);

regn reg\_4 (BusWires, Rin[4], Clock, R4);

regn reg\_5 (BusWires, Rin[5], Clock, R5);

regn reg\_6 (BusWires, Rin[6], Clock, R6);

regn reg\_7 (BusWires, Rin[7], Clock, R7);

regn reg\_IR (DIN, IRin, Clock, IR);

defparam reg\_IR.n = 9;

regn reg\_A (BusWires, Ain, Clock, A);

regn reg\_G (result, Gin, Clock, G);

regn reg\_ADDR (BusWires, ADDRin, Clock, ADDR);

regn reg\_DOUT (BusWires, DOUTin, Clock, DOUT);

regn reg\_W (W\_D, 1'b1, Clock, W);

defparam reg\_W.n = 1;

addsub AS (~AddSub, A, BusWires, result);

//define the bus

always @ (MUXsel or Rout or Gout or DINout)

begin

MUXsel[9:2] = Rout;

MUXsel[1] = Gout;

MUXsel[0] = DINout;

case (MUXsel)

10'b0000000001: BusWires = DIN;

10'b0000000010: BusWires = G;

10'b0000000100: BusWires = R0;

10'b0000001000: BusWires = R1;

10'b0000010000: BusWires = R2;

10'b0000100000: BusWires = R3;

10'b0001000000: BusWires = R4;

10'b0010000000: BusWires = R5;

10'b0100000000: BusWires = R6;

10'b1000000000: BusWires = R7;

endcase

end

endmodule

module upcount(Clear, Clock, Q);

input Clear, Clock;

output [2:0] Q;

reg [2:0] Q;

always @(posedge Clock)

if (Clear)

Q <= 3'b0;

else

Q <= Q + 1'b1;

endmodule

module dec3to8(W, En, Y);

input [2:0] W;

input En;

output [0:7] Y;

reg [0:7] Y;

always @(W or En)

begin

if (En == 1)

case (W)

3'b000: Y = 8'b10000000;

3'b001: Y = 8'b01000000;

3'b010: Y = 8'b00100000;

3'b011: Y = 8'b00010000;

3'b100: Y = 8'b00001000;

3'b101: Y = 8'b00000100;

3'b110: Y = 8'b00000010;

3'b111: Y = 8'b00000001;

endcase

else

Y = 8'b00000000;

end

endmodule

### Módulo da memória rom

module Rom1Port (

address,

clock,

q);

input [4:0] address;

input clock;

output [8:0] q;

`ifndef ALTERA\_RESERVED\_QIS

// synopsys translate\_off

`endif

tri1 clock;

`ifndef ALTERA\_RESERVED\_QIS

// synopsys translate\_on

`endif

wire [8:0] sub\_wire0;

wire [8:0] q = sub\_wire0[8:0];

altsyncram altsyncram\_component (

.address\_a (address),

.clock0 (clock),

.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\_a ({9{1'b1}}),

.data\_b (1'b1),

.eccstatus (),

.q\_b (),

.rden\_a (1'b1),

.rden\_b (1'b1),

.wren\_a (1'b0),

.wren\_b (1'b0));

defparam

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

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

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

altsyncram\_component.init\_file = "inst\_mem.mif",

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

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

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

altsyncram\_component.numwords\_a = 32,

altsyncram\_component.operation\_mode = "ROM",

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

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

altsyncram\_component.widthad\_a = 5,

altsyncram\_component.width\_a = 9,

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

endmodule

### counter\_modk

module counter\_modk(clock, reset\_n, Q);

parameter n = 4;

parameter k = 16;

input clock, reset\_n;

output [n-1:0] Q;

reg [n-1:0] Q;

always @(posedge clock or negedge reset\_n)

begin

if (~reset\_n)

Q <= 'd0;

else begin

Q <= Q + 1'b1;

if (Q == k-1)

Q <= 'd0;

end

end

endmodule

### hex\_ssd

module hex\_ssd (BIN, SSD);

input [3:0] BIN;

output reg [0:6] SSD;

always begin

case(BIN)

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;

10:SSD=7'b0001000;

11:SSD=7'b1100000;

12:SSD=7'b0110001;

13:SSD=7'b1000010;

14:SSD=7'b0110000;

15:SSD=7'b0111000;

endcase

end

endmodule

### port\_n

module port\_n (Clock, SW, wren, out);

input [15:0] SW;

input Clock;

output reg [15:0] out;

input wren;

always @ (posedge Clock) begin

out <= SW;

end

endmodule

### seg7\_scroll

module seg7\_scroll (ADDR, Clock, data, wren, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7);

input [15:0] ADDR;

input Clock;

input [15:0] data;

input wren;

output reg [0:6] HEX7, HEX6, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;

reg [0:6] H0, H1, H2, H3, H4, H5, H6, H7;

always @ (posedge Clock) begin

case (ADDR[2:0])

3'b000: HEX0 = data[6:0];

3'b001: HEX1 = data[6:0];

3'b010: HEX2 = data[6:0];

3'b011: HEX3 = data[6:0];

3'b100: HEX4 = data[6:0];

3'b101: HEX5 = data[6:0];

3'b110: HEX6 = data[6:0];

3'b111: HEX7 = data[6:0];

endcase

end

endmodule

### RAM e ROM

Ram e Rom geradas pelo wizard

## Principal

module part1 (CLOCK\_50, SW, KEY, LEDG, HEX3, HEX2, HEX1, HEX0);

input CLOCK\_50;

input [9:0] SW;

input [3:0] KEY;

output reg [9:0] LEDG;

output [0:6] HEX3, HEX2, HEX1, HEX0;

wire [15:0] BusWires, ADDR, DOUT, LEDsOUT, RAMOUT, PORTNOUT;

reg [15:0] DIN;

wire Resetn, Clock, Run, Done, W;

reg LEDen, MEMen, SSDen, PRTen;

//assign DIN = SW[15:0];

//assign LEDR[15:0] = BusWires;

//assign LEDR[15:0] = DIN;

wire [25:0] newclock;

counter\_modk C\_new (CLOCK\_50, 1, newclock);

defparam C\_new.n = 26;

defparam C\_new.k = 50000000;

//assign MClock = newclock[25];//CLOCK\_50; //KEY[1];

//assign PClock = ~newclock[25];//CLOCK\_50; //KEY[2];

assign MClock = KEY[1];

assign PClock = KEY[2];

assign Run = SW[9];

assign Resetn = KEY[0];

wire [15:0] R0;

always begin

if (SW[9])

LEDG[8:0] = DIN;

else

LEDG[8:0] = R0;

LEDG[9] = Done;

end

always

begin

//LEDen = W;

//MEMen = W;

MEMen = W & ~(ADDR[15] | ADDR[14] | ADDR[13] | ADDR[12]);

LEDen = W & ~(ADDR[15] | ADDR[14] | ADDR[13] | ~ADDR[12]);

SSDen = W & ~(ADDR[15] | ADDR[14] | ~ADDR[13] | ADDR[12]);

PRTen = ~(ADDR[15] | ADDR[14] | ~ADDR[13] | ~ADDR[12]);

case (ADDR[15:12])

4'b0000: DIN = RAMOUT;

4'b0011: DIN = PORTNOUT;

endcase

end

proc P0 (DIN, Resetn, PClock, Run, Done, BusWires, ADDR, DOUT, W, LEDG[2:0], R0);

regn LEDs (DOUT, LEDen, MClock, LEDsOUT);

ramlpm Memory (ADDR, MClock, DOUT, MEMen, RAMOUT);

port\_n (MClock, SW[9:0], PRTen, PORTNOUT);

hex\_ssd H0 (BusWires[3:0], HEX0);

hex\_ssd H1 (BusWires[7:4], HEX1);

hex\_ssd H2 (BusWires[11:8], HEX2);

hex\_ssd H3 (BusWires[15:12], HEX3);

endmodule

## Ondas





