Relatório de Labolatório de Sistemas Digitais Avançados

Alunos:

Judah Holanda Correia Lima – Matricula: 1010500/5

José David Sousa de Araújo– Matricula: 1020678

José Kilson – Matricula: 0810847

Sumário

[Introdução 3](#_Toc382666853)

[Material Utilizado 4](#_Toc382666854)

[Prática 3- Latches, Flip-flops and Registers 5](#_Toc382666855)

[Objetivo 5](#_Toc382666856)

[Parte 1 6](#_Toc382666857)

[Parte 2 7](#_Toc382666858)

[Parte 3 8](#_Toc382666859)

[Parte 4 9](#_Toc382666860)

[Parte 5 11](#_Toc382666861)

[Conclusão 13](#_Toc382666862)

# Introdução

Esta prática tem como finalidade a resolução do exercício 3 que impõem o projeto de Flip Flop e registradores. Na sua descrição de hardware foi utilizado verilog 2001, utilizando portas lógicas e lógica booleana, sendo realizado a descrição do hardware em módulos, tornando-os reaproveitáveis para as outras práticas.

# Objetivo

O propósito deste exercício é desenvolver e fornecer uma introdução em verilog de latches, flip-flops e registers para serem usados nos próximos exercícios.

# Material Utilizado

* Kit de desenvolvimento DE0
* Software Quartus fornecido pelo fabricante Altera

# Prática 3- Latches, Flip-flops and Registers

## Parte 1

Esta prática tem como finalidade criar um circuito RS Latch:

//Módulo principal

module part1(SW,LEDG);

//Sinal de Entrada

input [9:0] SW; //Chaves

output [9:0] LEDG; //Leds

//Chamada do modulo do Flip Flop SR

FlipFlopSR\_1 SR1(SW[0],SW[1],SW[2],LEDG[0],LEDG[1]);

// FlipFlopSR\_2 SR2(R,S,Clk,Q1,Q2);

endmodule

//Módulo Flip Flop SR usando portas lógicas

module FlipFlopSR\_1(Clk,S,R,Q1,Q2);

input R,S,Clk;

output Q1,Q2;

wire R\_g, S\_g, Qa, Qb;

and (R\_g, R, Clk);

and (S\_g, S, Clk);

nor (Qa, R\_g, Qb);

nor (Qb, S\_g, Qa);

assign Q1 = Qa;

assign Q2 = Qb;

endmodule

//Módulo Flip Flop SR usando lógica booleana

module FlipFlopSR\_2(Clk,S,R,Q1,Q2);

input R,S,Clk;

output Q1,Q2;

wire R\_g, S\_g, Qa, Qb;

assign R\_g = R & Clk;

assign S\_g = S & Clk;

assign Qa = ~(R\_g | Qb);

assign Qb = ~(S\_g | Qa);

assign Q1 = Qa;

assign Q2 = Qb

endmodule

## Parte 2

Esta prática tem como finalidade criar um circuito gated D Latch:

//Módulo Principal

module part2(SW,LEDG);

//Sinais de entrada

input [9:0] SW; //chave

output [9:0] LEDG; //Leds

//Chamada do modulo Flip Flop D

FlipFlopD D1(SW[1],SW[0],LEDG[0],LEDG[1]);

endmodule

//Módulo Flip Flop D, Q1 saída normal e Q2 saída invertida

module FlipFlopD(Clk,D,Q1,Q2);

input Clk,D;

output Q1,Q2;

wire S,R;

assign S = D;

assign R = ~D;

//chamada do módulo Flip Flop SR á implementado, usado para fazer o Flip Flop D.

FlipFlopSR\_1 SR2(Clk,S,R,Q1,Q2);

endmodule

## Parte 3

Esta prática tem como finalidade criar um circuito master-slave D Flip-flop:

//Módulo principal

module part3(SW, LEDG);

//Sinais de entrada

input [9:0] SW; //chave

output [9:0]LEDG; //Leds

//Chamada do módulo

FlipFlopMasterSlaveD D1(SW[1], SW[0], LEDG[0], LEDG[1]);

endmodule

//Módulo do Flip Flop Master Slave

module FlipFlopMasterSlaveD(Clk, D, Q1, Q2);

input Clk, D;

output Q1, Q2;

wire Qm1, Qm2, Clk1;

assign Clk1 = ~Clk;

//Chamada do modulo Flip Flop D

FlipFlopD D1(Clk1, D,Qm1, Qm2);

//O Q do D1 é a entrada do D2

FlipFlopD D2(Clk, Qm1, Q1, Q2);

endmodule

## Parte 4

Esta prática tem como finalidade criar um circuito gated D latch, um positive-edge triggered D flip-flop e um negative-edge triggered D flip-flop, em paralelo:

Esta prática tem como finalidade fazer um somador de dois BCDs mais carry:

Decodificador BCD de 0 a 9 para cada display, depois a soma destes e ao final coloca-se nos dois outros displays:

//Módulo principal

module part4(LEDG, SW);

//Sinais de entrada

input [9:0] SW; //Chave

output [9:0] LEDG; //Leds

//Chamada dos módulos

FlipFlopDLatch D0 (SW[0], SW[1], LEDG[0], LEDG[1]);

FlipFlopDPositiveEdgeTriggered D1 (SW[0], SW[1], LEDG[2], LEDG[3]);

FlipFlopDNegativeEdgeTriggered D2 (SW[0], SW[1], LEDG[4], LEDG[5]);

endmodule

//Módulo Flip Flop D como chave

module FlipFlopDLatch (D, Clk, Q, nQ);

input D, Clk;

output reg Q, nQ;

always @ (Clk or D)begin

if (Clk)

Q <= D;

nQ <=~Q;

end

endmodule

//Módulo Flip Flop D aciona na borda de subida

module FlipFlopDPositiveEdgeTriggered (D, Clk, Q, nQ);

input D, Clk;

output reg Q, nQ;

always @ (posedge Clk)begin

Q <= D;

nQ <=~D;

end

endmodule

//Módulo Flip Flip D aciona na borda de descina

module FlipFlopDNegativeEdgeTriggered (D, Clk, Q, nQ);

input D, Clk;

output reg Q, nQ;

always @ (negedge Clk)begin

Q <= D;

nQ <=~D;

end

endmodule

## Parte 5

Esta prática tem como finalidade criar um circuito onde sejam mostrados dois números de 8 bits, A e B cada um sendo mostrado em dois displays de 7 segmentos, em hexadecmal.

Eles devem ser configurados a partir de um clock (gerado por um switch) e um reset (também gerado por um switch), ao configurar um número o outro deve ser armazenado:

//Instanciação e execução dos módulos.

module part5(SW, LEDG, HEX0, HEX1, HEX2, HEX3);

input [9:0] SW; //Chave

output [0:6] HEX0, HEX1, HEX2, HEX3; //7-Segmentos

output [9:0] LEDG; //Leds

wire [7:0] A;

wire [7:0] B;

//Chamada dos módulos

counter counter0(SW[8],SW[9],A,B,SW[7:0]); //SW[8]=clock, SW[9]=reset

segmentDecoder7 dec0(HEX0,HEX1,A);

segmentDecoder7 dec1(HEX2,HEX3,B);

endmodule

//Módulo com o contador para a configuração dos números de 8bits, com o reset para o contador.

module counter(clock,reset,A,B,C);

input [7:0] C;

input clock, reset;

output reg[7:0] A;

output reg[7:0] B;

reg [4:0] count;

always @(posedge clock or negedge reset) begin

if (~reset) begin

count <= 'd0;

end

else begin

if (clock)

count <= count + 1;

end

case (count)

0 : begin

A<=0;

end

1 : begin

B<=0;

end

2 : begin

A<=C;

end

3 : begin

B<=C;

end

default : count <= 'd2;

endcase

end

endmodule

//Decodificador para 2 displays de 7 segmentos no formato de hexadecimal

module segmentDecoder7(HEX0,HEX1,A);

input [0:7] A;

output [0:6] HEX0,HEX1;

assign HEX0[0] = ~(~A[2]&~A[0] | ~A[3]&A[1] | A[2]&A[1] | A[3]&~A[0] | ~A[3]&A[2]&A[0] | A[3]&~A[2]&~A[1]);

assign HEX0[1] = ~(~A[2]&~A[0] | ~A[2]&~A[1] | ~A[3]&~A[1]&~A[0] | A[3]&~A[1]&A[0] | ~A[3]&A[1]&A[0]);

assign HEX0[2] = ~(A[3]&~A[2] | ~A[1]&A[0] | ~A[2]&~A[1] | ~A[3]&A[0] | ~A[3]&A[2]);

assign HEX0[3] = ~(~A[3]&~A[2]&~A[0] | ~A[2]&A[1]&A[0] | A[2]&~A[1]&A[0] | A[3]&~A[1] | A[2]&A[1]&~A[0]);

assign HEX0[4] = ~(~A[2]&~A[0] | A[3]&A[2] | A[1]&~A[0] | A[3]&A[1]);

assign HEX0[5] = ~(~A[1]&~A[0] | A[3]&~A[2] | A[2]&~A[0] | A[3]&A[1] | ~A[3]&A[2]);

assign HEX0[6] = ~(A[3]&~A[2] | A[1]&~A[0] | A[3]&A[0] | ~A[3]&A[2]&~A[1] | ~A[2]&A[1]);

assign HEX1[0] = ~(~A[6]&~A[4] | ~A[7]&A[5] | A[6]&A[5] | A[7]&~A[4] | ~A[7]&A[6]&A[4] | A[7]&~A[6]&~A[5]);

assign HEX1[1] = ~(~A[6]&~A[4] | ~A[6]&~A[5] | ~A[7]&~A[5]&~A[4] | A[7]&~A[5]&A[4] | ~A[7]&A[5]&A[4]);

assign HEX1[2] = ~(A[7]&~A[6] | ~A[5]&A[4] | ~A[6]&~A[5] | ~A[7]&A[4] | ~A[7]&A[6]);

assign HEX1[3] = ~(~A[7]&~A[6]&~A[4] | ~A[6]&A[5]&A[4] | A[6]&~A[5]&A[4] | A[7]&~A[5] | A[6]&A[5]&~A[4]);

assign HEX1[4] = ~(~A[6]&~A[4] | A[7]&A[6] | A[5]&~A[4] | A[7]&A[5]);

assign HEX1[5] = ~(~A[5]&~A[4] | A[7]&~A[6] | A[6]&~A[4] | A[7]&A[5] | ~A[7]&A[6]);

assign HEX1[6] = ~(A[7]&~A[6] | A[5]&~A[4] | A[7]&A[4] | ~A[7]&A[6]&~A[5] | ~A[6]&A[5]);

//assign Display[7] = 1; // 3-to-1 multiplexer output

endmodule

# Conclusão

Nesta prática foi introduzido o uso de portas lógicas (Parte 1) e do “reg” que facilitar a geração de lógica combinatória, embora continue o uso de lógica booleana.

A maior parte da implementação é modularizado, apresentando os resultados esperados da prática, sendo todas as tarefas executadas perfeitamente. Foi colocado algumas saídas adicionais para os Flip Flop que o exercício não pedia, como o Q invertido.