4.29------------------------------------------------------------------------------------------------------------------

->4.31 do livro

module fig3\_40

(

input clk, a, b, c, d, SW[4:0]

output reg x, y LEDG[1:0]

);

wire n1, n2;

reg areg, breg, creg, dreg;

always @ (posedge SW[4])

begin

areg <= SW[3];

breg <= SW[2];

creg <= SW[1];

dreg <= SW[0];

LEDG[1] <= n2;

LEDG[0] <= ~(dreg | n2);

end

assign n1 = areg & breg;

assign n2 = n1 | creg;

endmodule

4.31------------------------------------------------------------------------------------------------------------------

->4.33 do livro (APARENTEMENTE O LIVRO EH 2 A FRENTE DOS EXERCICIO)

module fig3\_66

(

input SW[3:0],

output reg LEDG[0:0]

);

reg [1:0] state, nextstate;

parameter S0 = 2'b00;

parameter S1 = 2'b01;

parameter S2 = 2'b10;

always @(posedge SW[3], posedge SW[2])

if (SW[2]) state <= S0;

else state <= nextstate;

always @ (\*)

case (state)

S0: if (SW[1]) nextstate = S1;

else nextstate = S0;

S1: if (SW[0]) nextstate = S2;

else nextstate = S0;

S2: if (SW[1] & SW[0]) nextstate = S2;

else nextstate = S0;

default: nextstate = S0;

endcase

always @ (\*)

case (state) S0: LEDG[0] = 0;

S1: LEDG[0] = 0;

S2: if (SW[1] & SW[0]) LEDG[0] = 1;

else LEDG[0] = 0;

default: LEDG[0] = 0;

endcase

endmodule

4.33------------------------------------------------------------------------------------------------------------------

->4.35 do livro

module daughterfsm

(

input SW[2:0],

output LEDG[0:0]

);

reg [2:0] state, nextstate;

parameter S0 = 3'b000;

parameter S1 = 3'b001;

parameter S2 = 3'b010;

parameter S3 = 3'b011;

parameter S4 = 3'b100;

always @(posedge SW[2], posedge SW[1])

if (SW[1]) state <= S0;

else state <= nextstate;

always @ (\*)

case (state)

S0: if (SW[0]) nextstate = S1;

else nextstate = S0;

S1: if (SW[0]) nextstate = S2;

else nextstate = S0;

S2: if (SW[0]) nextstate = S4;

else nextstate = S3;

S3: if (SW[0]) nextstate = S1;

else nextstate = S0;

S4: if (SW[0]) nextstate = S4;

else nextstate = S3;

default: nextstate = S0;

endcase

assign LEDG[0] = ((state == S3) & SW[0]) |

((state == S4) & ~SW[0]);

endmodule

4.35------------------------------------------------------------------------------------------------------------------

->4.37 do livro, possivelmente apresentavel

module ex4\_35

(

input SW[1:0],

output LEDG[2:0]

);

reg [2:0] state, nextstate;

parameter S0 = 3'b000;

parameter S1 = 3'b001;

parameter S2 = 3'b011;

parameter S3 = 3'b010;

parameter S4 = 3'b110;

parameter S5 = 3'b111;

parameter S6 = 3'b101;

parameter S7 = 3'b100;

always @(posedge SW[1], posedge SW[0])

if (SW[0]) state <= S0;

else state <= nextstate;

always @ (\*)

case (state)

S0: nextstate = S1;

S1: nextstate = S2;

S2: nextstate = S3;

S3: nextstate = S4;

S4: nextstate = S5;

S5: nextstate = S6;

S6: nextstate = S7;

S7: nextstate = S0;

endcase

assign LEDG[2:0] = state;

endmodule

4.37------------------------------------------------------------------------------------------------------------------

->4.39 do livro

module ex4\_37

(

input SW[3:0],

output reg LEDG[0:0]

);

reg [1:0] state, nextstate;

parameter S0 = 2'b00;

parameter S1 = 2'b01;

parameter S2 = 2'b10;

parameter S3 = 2'b11;

always @(posedge SW[3], posedge SW[2])

if (SW[2]) state <= S0;

else state <= nextstate;

always @ (\*)

case (state)

S0: case ({SW[0],SW[1]})

2'b00: nextstate = S0;

2'b01: nextstate = S3;

2'b10: nextstate = S0;

2'b11: nextstate = S1;

endcase

S1: case ({SW[0],SW[1]})

2'b00: nextstate = S0;

2'b01: nextstate = S3;

2'b10: nextstate = S2;

2'b11: nextstate = S1;

endcase

S2: case ({SW[0],SW[1]})

2'b00: nextstate = S0;

2'b01: nextstate = S3;

2'b10: nextstate = S2;

2'b11: nextstate = S1;

endcase

S3: case ({SW[0],SW[1]})

2'b00: nextstate = S0;

2'b01: nextstate = S3;

2'b10: nextstate = S2;

2'b11: nextstate = S1;

endcase

default: nextstate = S0;

endcase

always @ (\*)

case (state)

S0: LEDG[0] = SW[1] & SW[0];

S1: LEDG[0] = SW[1] | SW[0];

S2: LEDG[0] = SW[1] & SW[0];

S3: LEDG[0] = SW[1] | SW[0];

default: LEDG[0] = 1'b0;

endcase

endmodule

------------------------------------------------------------------------------------------------------------------------

OUTRA OPCAO

module ex4\_37

(

input SW[2:0],

output reg LEDG[0:0]

);

reg aprev;

always @(posedge SW[2])

aprev <= a;

assign LEDG[0] = SW[0] ? (aprev | SW[1]) : (aprev & SW[1]);

endmodule

4.39------------------------------------------------------------------------------------------------------------------

->4.41 do livro

module ex4\_39

(

input SW[2:0],

output LEDG[0:0]

);

reg [1:0] state, nextstate;

parameter S0 = 2'b00;

parameter S1 = 2'b01;

parameter S2 = 2'b10;

parameter S3 = 2'b11;

always @(posedge SW[2], posedge SW[1])

if (SW[1]) state <= S0;

else state <= nextstate;

always @(\*)

case (state)

S0: if (SW[0]) nextstate = S1;

else nextstate = S0;

S1: if (SW[0]) nextstate = S2;

else nextstate = S3;

S2: if (SW[0]) nextstate = S2;

else nextstate = S3;

S3: if (SW[0]) nextstate = S2;

else nextstate = S3;

endcase

assign LEDG[0] = state[0];

endmodule

4.41------------------------------------------------------------------------------------------------------------------

->4.43 do livro

module ex4\_41

(

input SW[2:0],

output LEDG[0:0]

);

reg [1:0] state, nextstate;

parameter S0 = 2'b00;

parameter S1 = 2'b01;

parameter S2 = 2'b10;

always @(posedge SW[2], posedge SW[1])

if (SW[1]) state <= S0;

else state <= nextstate;

always @(\*)

case (state)

S0: if (SW[0]) nextstate = S1;

else nextstate = S0;

S1: if (SW[0]) nextstate = S2;

else nextstate = S0;

S2: if (SW[0]) nextstate = S2;

else nextstate = S0;

default: nextstate = S0;

endcase

assign LEDG[0] = state[1];

endmodule

4.43------------------------------------------------------------------------------------------------------------------

->4.45 do livro

module ex4\_43

(

input SW[5:0],

output reg LEDG[1:0]

);

reg [1:0] areg, breg;

reg creg;

wire [1:0] sum;

wire cout;

always @(posedge SW[5])

{areg, breg, creg, LEDG} <= {SW[3:2], SW[1:0], SW[4], sum};

fulladder fulladd1(areg[0], breg[0], creg, sum[0], cout);

fulladder fulladd2(areg[1], breg[1], cout, sum[1], );

endmodule

module fulladder

(

input logic a, b, cin,

output logic s, cout

);

logic p, g;

assign p =a ^ b;

assign g =a & b;

assign s = p ^ cin;

assign cout = g | (p & cin);

endmodule

4.45------------------------------------------------------------------------------------------------------------------

->4.47 do livro

module syncbad

(

input SW[1:0],

output reg LEDG[0:0]

);

reg n1;

always @(posedge SW[1])

begin

LEDG[0] <= n1;

n1 <= SW[0];

end

endmodule

4.47------------------------------------------------------------------------------------------------------------------

->4.49 do livro

module code1

(

input logic SW[3:0],

output logic LEDG[0:0]

);

logic x;

always\_ff @(posedge SW[3])

begin

x = SW[2] & SW[1];

LEDG[0] = x | SW[0];

end

endmodule

module code2

(

input logic SW[3:0],

output logic LEDG[0:0]

);

logic x;

always\_ff @(posedge SW[3])

begin

LEDG[0] = x | SW[0];

x = SW[2] & SW[1];

end

endmodule