A close-up of a paper

Description automatically generated

A computer screen with text and diagrams

Description automatically generated

**Watch Main:**

module watch\_main(

input clk,

input reset,

input [7:0] sw,

input [1:0] mode,

output [3:0] an,

output [6:0] sseg,

output dp,

input stop\_start

);

localparam [6:0] zero = 7'b0000001;

wire[6:0] in0, in1, in2, in3;

wire[6:0] iu0, iu1, iu2, iu3;

wire[6:0] id0, id1, id2, id3;

wire[3:0] su0, su1, su2, su3;

wire[3:0] sd0, sd1, sd2, sd3;

wire tick;

wire slow\_clk, faster\_clk;

clkdiv cd1(.clk(clk), .reset(reset), .slow\_clk(slow\_clk), .tick(tick), .faster\_clk(faster\_clk));

up\_count uc1(.clk(clk), .reset(reset), .tick(tick), .stop\_start(stop\_start), .mode(mode), .sw(sw), .seg0(su0), .seg1(su1), .seg2(su2), .seg3(su3));

//down\_count dc1(); //to be implemented

hex\_7seg h1(.x(su0), .r(id0));

hex\_7seg h2(.x(su1), .r(id1));

hex\_7seg h3(.x(su2), .r(id2));

hex\_7seg h4(.x(su3), .r(id3));

hex\_7seg h5(.x(su0), .r(iu0));

hex\_7seg h6(.x(su1), .r(iu1));

hex\_7seg h7(.x(su2), .r(iu2));

hex\_7seg h8(.x(su3), .r(iu3));

assign in0 = ((mode == 2'b00 || mode == 2'b01)?id0:iu0);

assign in1 = ((mode == 2'b00 || mode == 2'b01)?id1:iu1);

assign in2 = ((mode == 2'b00 || mode == 2'b01)?id2:iu2);

assign in3 = ((mode == 2'b00 || mode == 2'b01)?id3:iu3);

time\_mux t1(.clk(faster\_clk), .reset(reset), .in0(in0), .in1(in1), .in2(in2), .in3(in3), .sseg(sseg), .an(an), .dp(dp));

endmodule

**Clock Divider**

module clkdiv(

input clk,

input reset,

output slow\_clk,

output faster\_clk,

output tick

);

reg[19:0] COUNT;

assign faster\_clk = COUNT[14];

assign slow\_clk = COUNT[19];

always @(posedge clk or posedge reset)

begin

if(reset)

begin

COUNT <= 0;

end

else

COUNT <= COUNT + 1;

end

assign tick = ((COUNT == 1000000)?1'b1:1'b0);

endmodule

**Up/Down Count Logic**

module up\_count(

input clk,

input reset,

input tick,

input stop\_start,

input [1:0] mode,

input [7:0] sw,

output reg[3:0] seg0,

output reg[3:0] seg1,

output reg[3:0] seg2,

output reg[3:0] seg3

);

reg stop = 0;

always@(posedge clk or posedge reset)

begin

if(reset) //if reset button is pressed,

begin

case(mode)

2'b00:begin //mode 1, count up from 0

seg0<=0;

seg1<=0;

seg2<=0;

seg3<=0;

stop<=0;

end

2'b01: begin //mode 2, count up from loaded val

seg0<=0;

seg1<=0;

seg2<=sw[3:0]; //second digit of seconds

seg3<=sw[7:4]; //first digit of seconds

stop<=0;

end

3'b10: begin

seg0<=9;

seg1<=9;

seg2<=9;

seg3<=9;

stop<=0;

end

2'b11: begin //mode 2, count up from loaded val

seg0<=0;

seg1<=0;

seg2<=sw[3:0]; //second digit of seconds

seg3<=sw[7:4]; //first digit of seconds

stop<=0;

end

default: stop<=0;

endcase

end

else if(stop\_start) //if the stop/start button is pressed

begin //retain current value

if(stop == 0)

stop<=1;

else if(stop == 1)

stop<=0;

end

else if((tick && stop) &&(mode == 0 || mode==1))

begin

if(seg0==9)

begin

if(seg3==9 && seg2==9 && seg1 == 9) //all segments equal to max

seg0<=0;

else

seg0<=0;

if(seg1==9)

begin

if(seg3==9 && seg2==9)

seg1<=9;

else

seg1<=0;

if(seg2==9 && seg3!=9)

begin

seg2<=0;

if(seg3>=9)

seg3=9;

else if(seg3!=9)

seg3<=seg3+1;

end

else if(seg2!=9)

seg2<=seg2+1;

end

else if(seg1!=9)

seg1<=seg1+1;

end

else if(seg0!=9)

seg0<=seg0+1;

end

else if ((tick && stop) && (mode == 2 || mode==3))

begin

if (seg0 == 0) // if seg0 reaches 0, decrement other segments

begin

if (seg1 == 0)

begin

if (seg2 == 0)

begin

if (seg3 == 0)

stop <= 1; // Stop counting when all segments reach 0

else

seg3 <= seg3 - 1;

end

else

seg2 <= seg2 - 1;

end

else

seg1 <= seg1 - 1;

seg0 <= 9;

end

else

seg0 <= seg0 - 1;

end

end

endmodule

**Hex-to-7 segment decoder**

module hex\_7seg(

input[3:0]x,

output reg[6:0]r

);

always@(\*)

case(x)

4'b0000 : r = 7'b0000001;

4'b0001 : r = 7'b1001111;

4'b0010 : r = 7'b0010010;

4'b0011 : r = 7'b0000110;

4'b0100 : r = 7'b1001100;

4'b0101 : r = 7'b0100100;

4'b0110 : r = 7'b0100000;

4'b0111 : r = 7'b0001111;

4'b1000 : r = 7'b0000000;

4'b1001 : r = 7'b0000100;

4'b1010 : r = 7'b0001000;

4'b1011 : r = 7'b1100000;

4'b1100 : r = 7'b0110001;

4'b1101 : r = 7'b1000010;

4'b1110 : r = 7'b0110000;

4'b1111 : r = 7'b0111000;

endcase

endmodule

**Time Multiplexer (FSM Controller)**

module time\_mux(

input clk,

input reset,

input [6:0] in0,

input [6:0] in1,

input [6:0] in2,

input [6:0] in3,

output reg[6:0] sseg,

output reg[3:0] an,

output reg dp

);

reg[1:0] state;

reg[1:0] next\_state;

localparam [6:0] zero = 7'b0000001;

localparam [6:0] nine = 7'b0001100;

always@(\*)begin

case(state)

2'b00: next\_state = 2'b01;

2'b01: next\_state = 2'b10;

2'b10: next\_state = 2'b11;

2'b11: next\_state = 2'b00;

endcase

end

always@(\*)begin

case(state)

2'b00: sseg = in0;

2'b01: sseg = in1;

2'b10: sseg = in2;

2'b11: sseg = in3;

endcase

case(state)

2'b00:

begin

an = 4'b1110;

dp = 1'b1;

end

2'b01:

begin

an = 4'b1101;

dp = 1'b1;

end

2'b10:

begin

an = 4'b1011;

dp = 1'b0;

end

2'b11:

begin

an = 4'b0111;

dp = 1'b1;

end

endcase

end

always@(posedge clk or posedge reset)

begin

if(reset)

begin

state<=2'b00;

end

else

state<=next\_state;

end

endmodule