ALARM CLOCK PROJECT

Write RTL and verify components, integrate and test the Alarm Clock specified.

RTL-aclk_controller

```
module aclk_controller(input clk,reset,one_second,alarm_button,time_button,
                input [3:0]key,
                output
load_new_c,show_new_time,show_a,load_new_a,shift,reset_count);

    parameter SHOW_TIME=3'd0,
          KEY_STORED=3'd1,
          KEY_WAITED=3'd2,
          KEY_ENTRY=3'd3,
          SET_ALARM_TIME=3'd4,
          SET_CURRENT_TIME=3'd5,
          SHOW_ALARM=3'd6;

reg [2:0]pstate,next_state;

reg [3:0]count1,count2;

reg timeout;

//reg load_new_c_reg,show_new_time_reg,show_a_reg,load_new_a_reg,shift_reg;



assign load_new_c = (pstate==SET_CURRENT_TIME)?1:0;
assign load_new_a = (pstate== SET_ALARM_TIME)?1:0;

assign show_new_time= (pstate==KEY_STORED || pstate==KEY_WAITED ||
pstate==KEY_ENTRY || pstate==SET_ALARM_TIME || pstate==SET_CURRENT_TIME
)?1:0;
assign show_a=(pstate==SHOW_ALARM)?1:0;
assign reset_count=0;
assign shift= (pstate==KEY_STORED)?1:0;



//SEQUENTIAL
always @(posedge clk or posedge reset)
    begin

    if(reset==1)
        {pstate,next_state}=0;


    else
        pstate<=next_state;

    end
```

```
//10 Second TIMER


always @(posedge one_second or posedge reset)
    begin
        if(reset==1)
            count1=0;

        else if(pstate!=KEY_WAITED)
            count1=0;

        else if(count1==9)
            count1=0;
        else
            count1=count1+1;
    end


always @(posedge one_second or posedge reset)
    begin
        if(reset==1)
            count2=0;

        else if(pstate!=KEY_ENTRY)
            count2=0;

        else if(count2==9)
            count2=0;
        else
            count2=count2+1;
    end


assign timeout = (count1==9 || count2==9)?0:1;

//NEXT_STATE LOGIC
always @(pstate,one_second,alarm_button,time_button,key,timeout)
    begin

    case (pstate)

    SHOW_TIME:
        if(key!=10)
            next_state=KEY_STORED;

        else if(alarm_button==1)
            next_state=SHOW_ALARM;
        else
            next_state=SHOW_TIME;



    KEY_STORED:
        if(key!=10)
            next_state=KEY_WAITED;
```

```
        else
            next_state=SHOW_TIME;

    KEY_WAITED:
        if(key==10)
            next_state=KEY_ENTRY;
        else if(timeout==0)
            next_state=SHOW_TIME;
        else
            next_state=KEY_WAITED;

    KEY_ENTRY:
        if(key!=10)
            next_state=KEY_STORED;

        else if(alarm_button==1)
            next_state=SET_ALARM_TIME;

        else if(time_button==1)
            next_state=SET_CURRENT_TIME;

        else if(timeout==0)
            next_state=SHOW_TIME;

        else
            next_state=KEY_ENTRY;
    SET_ALARM_TIME:
        next_state=SHOW_TIME;

    SET_CURRENT_TIME:
        next_state=SHOW_TIME;


    SHOW_ALARM:
        if(alarm_button==1)
            next_state=SHOW_ALARM;
        else
            next_state=SHOW_TIME;
    endcase


    end


endmodule
```

TB-aclk_controller

```
module tb_aclk_controller();

    reg clk,reset,one_second,alarm_button,time_button;
    reg [3:0]key;
    wire load_new_c,show_new_time,show_a,load_new_a,shift,reset_count;
```

```
    wire [3:0] key_buffer_ls_min,
               key_buffer_ms_min,
               key_buffer_ls_hr,
               key_buffer_ms_hr;

aclk_controller
DUT(.clk(clk),.reset(reset),.one_second(one_second),.alarm_button(alarm_butto
n),.time_button(time_button),

.reset_count(reset_count),.key(key),.load_new_c(load_new_c),.show_new_time(sh
ow_new_time),.show_a(show_a),
            .load_new_a(load_new_a),.shift(shift) );


keyreg DUT2(
            .reset(reset),
                .clock(clk),
                .shift(shift),
                .key(key),
        .key_buffer_ls_min(key_buffer_ls_min),
            .key_buffer_ms_min(key_buffer_ms_min),
                .key_buffer_ls_hr(key_buffer_ls_hr),
                .key_buffer_ms_hr(key_buffer_ms_hr)

        );




initial
clk=0;
always #5 clk=~clk;

initial
one_second=0;
always #20 one_second=~one_second;

task initialize;
    begin
    {one_second,alarm_button,time_button}=0;
    key=10;
    end
endtask

initial
    begin


        @(negedge clk);
        initialize;
        @(negedge clk);
        reset=1;
        @(negedge clk);
        reset=0;


//FIRST
```

```
        @(negedge clk);
        key=1;      //PRESS
        //KEY_STORED
        @(negedge clk);

        ////KEY_WAITED

        #480 //CHECKING KEY_WAITEED TIMEOUT


        @(negedge clk);
        reset=1;
        @(negedge clk);
        reset=0;

/*-------------------------------------------------------------------------*/
//FIRST
        @(negedge clk);
        key=1;      //PRESS
        //KEY_STORED
        @(negedge clk);

        ////KEY_WAITED


        @(negedge clk);
        key=10;          //RELEASE


        //KEY_ENTRY


        #480 //CHECKING KEY ENTRY TIMEOUT


        @(negedge clk);
        reset=1;
        @(negedge clk);
        reset=0;
/*-------------------------------------------------------------------------*/
//FIRST
        @(negedge clk);
        key=1;      //PRESS
        //KEY_STORED
        @(negedge clk);

        ////KEY_WAITED

        @(negedge clk);
        key=10;          //RELEASE


        //KEY_ENTRY


        @(negedge clk);
        key=7;
```

```
        //KEY_STORED


        @(negedge clk);

        @(negedge clk);
        key=10;          //RELEASE

//SECOND

        @(negedge clk);
        key=3;      //PRESS
        //KEY_STORED
        @(negedge clk);

        ////KEY_WAITED

        @(negedge clk);
        key=10;     //RELEASE


        //KEY_ENTRY

        @(negedge clk);
        key=2;

        //KEYSTORED

        @(negedge clk);

        @(negedge clk);
        key=10;          //FINAL RELEASE


        #50;

        @(negedge clk);
        time_button=1;

        @(negedge clk);
        time_button=0;

        @(negedge clk);
        reset=1;
        @(negedge clk);
        reset=0;
        #100;



    end

endmodule
```

RTL-LCD DRIVER

```
module lcd_driver (alarm_time,
                   current_time,
                   show_alarm,
                   show_new_time,
                   key,display_time,
                   sound_alarm);
//Define input and output ports direction
input [3:0] key;
input [3:0]alarm_time;
input [3:0]current_time;
input show_alarm;
input show_new_time;


output reg [7:0]display_time;
output reg sound_alarm;

//Define the internal signals
reg [3:0]display_value ;

//Define the Parameter constants to represent LCD numbers
parameter ZERO   = 8'h30;
parameter ONE    = 8'h31;
parameter TWO    = 8'h32;
parameter THREE  = 8'h33;
parameter FOUR   = 8'h34;
parameter FIVE   = 8'h35;
parameter SIX    = 8'h36;
parameter SEVEN  = 8'h37;
parameter EIGHT  = 8'h38;
parameter NINE   = 8'h39;
parameter ERROR  = 8'h3A;


 assign sound_alarm=(alarm_time==current_time)?1:0;

always @ (alarm_time or current_time or show_alarm or show_new_time or key)
  begin
    //Displays the key_time,alarm_time or current_time as per the control
signals

    if(show_alarm==1 && show_new_time==0)
        display_value=alarm_time;

    else if(show_alarm==0 && show_new_time==1)
        display_value=key;

    else if(show_alarm==0 && show_new_time==0)
        display_value=current_time;
    else
```

```
        display_value=current_time;



  end

//Decoder logic
always @ (display_value)
  begin
    // For number stored in display_value register, load display_time
register with LCD equivalent
    case (display_value)
    0: display_time=ZERO;
    1: display_time=ONE;
    2: display_time=TWO;
    3: display_time=THREE;
    4: display_time=FOUR;
    5: display_time=FIVE;
    6: display_time=SIX;
    7: display_time=SEVEN;
    8: display_time=EIGHT;
    9: display_time=NINE;

       default : display_time = ERROR;
    endcase
   end


endmodule
```

TB-LCD DRIVER

```
module lcd_driver_tb();

reg [3:0] key;
reg [3:0]alarm_time;
reg [3:0]current_time;
reg show_alarm;
reg show_new_time;

wire [7:0]display_time;
wire sound_alarm;



    lcd_driver DUT(
           .alarm_time(alarm_time),
                   .current_time(current_time),
                   .show_alarm(show_alarm),
                   .show_new_time(show_new_time),
                   .key(key),
           .display_time(display_time),
                   .sound_alarm(sound_alarm)  );
```

```
initial
begin
    {key,alarm_time,current_time,show_alarm,show_new_time}=0;

#5 current_time=4'b0011;
#5 key=4'b0101; show_new_time=1;
#5 alarm_time=4'b0011; show_alarm=1;show_new_time=0;
#5 show_alarm=0;


end

initial
    $monitor("Inputs
key=%b,alarm_time=%b,current_time=%b,show_alarm=%b,show_new=%b --- Outputs
display_time=%b,sound_a=%b",

key,alarm_time,current_time,show_alarm,show_new_time,display_time,sound_alarm
);
endmodule
```

RTL - LCD DRIVER 4LCDS

```
module lcd_driver_4 ( alarm_time_ms_hr,
                      alarm_time_ls_hr,
                      alarm_time_ms_min,
                      alarm_time_ls_min,
                      current_time_ms_hr,
                      current_time_ls_hr,
                      current_time_ms_min,
                      current_time_ls_min,
                      key_ms_hr,
                      key_ls_hr,
                      key_ms_min,
                      key_ls_min,
                      show_a,
                      show_current_time,
                      display_ms_hr,
                      display_ls_hr,
                      display_ms_min,
                      display_ls_min,
                      sound_a);
// Define input and output port directions
input [3:0] alarm_time_ms_hr,
            alarm_time_ls_hr,
            alarm_time_ms_min,
            alarm_time_ls_min,
            current_time_ms_hr,
            current_time_ls_hr,
            current_time_ms_min,
            current_time_ls_min,
            key_ms_hr,
            key_ls_hr,
```

```
            key_ms_min,
            key_ls_min;

output [7:0] display_ms_hr,
             display_ls_hr,
             display_ms_min,
             display_ls_min;

input show_a,show_current_time;
output sound_a;

wire sound_alarm1,sound_alarm2,sound_alarm3,sound_alarm4;
// Assert sound_a when all 4 digits matches

//assign sound_alarm1=(alarm_time_ms_hr==current_time_ms_hr)?1:0;
//assign sound_alarm2=(alarm_time_ls_hr==current_time_ls_hr)?1:0;
//assign sound_alarm3=(alarm_time_ms_min==current_time_ms_min)?1:0;
//assign sound_alarm4=(alarm_time_ls_min==current_time_ls_min)?1:0;

//assign sound_a=(sound_alarm1 && sound_alarm2 && sound_alarm3 &&
sound_alarm4)?1:0;

//Instantiate lcd_driver as MS_HR_display
lcd_driver MS_HR (.alarm_time(alarm_time_ms_hr),
                  .current_time(current_time_ms_hr),
                  .key(key_ms_hr),
                  .show_alarm(show_a),
                  .show_new_time(show_current_time),
                  .display_time(display_ms_hr),
                  .sound_alarm(sound_alarm1));

//Instantiate lcd_driver as LS_HR_display
lcd_driver LS_HR (.alarm_time(alarm_time_ls_hr),
                  .current_time(current_time_ls_hr),
                  .key(key_ls_hr),
                  .show_alarm(show_a),
                  .show_new_time(show_current_time),
                  .display_time(display_ls_hr),
                  .sound_alarm(sound_alarm2));

//Instantiate lcd_driver as MS_MIN_display
lcd_driver MS_MIN (.alarm_time(alarm_time_ms_min),
                  .current_time(current_time_ms_min),
                  .key(key_ms_min),
                  .show_alarm(show_a),
                  .show_new_time(show_current_time),
                  .display_time(display_ms_min),
                  .sound_alarm(sound_alarm3));


//Instantiate lcd_driver as LS_MIN_display
lcd_driver LS_MIN (.alarm_time(alarm_time_ls_min),
                  .current_time(current_time_ls_min),
                  .key(key_ls_min),
                  .show_alarm(show_a),
                  .show_new_time(show_current_time),
                  .display_time(display_ls_min),
```

```
                .sound_alarm(sound_alarm4));

assign sound_a=(sound_alarm1 && sound_alarm2 && sound_alarm3 &&
sound_alarm4)?1:0;


endmodule
```

RTL - ACLK-TIMING GENERTAOR

```
module aclk_timegen(clk,reset,reset_count,fast_watch,one_minute,one_second);

    input clk,reset,fast_watch,reset_count;
    output one_minute,one_second;

    reg [15:0]count_reg;

    reg one_minute_reg,one_second_reg;



    //one second
    always @(posedge clk or posedge reset)
    begin
        if(reset==1)
            begin
            count_reg<=0;
            one_second_reg<=0;
            end
        else if(reset_count==1)
            begin
            count_reg<=0;
            one_second_reg<=0;
            end


        else if(count_reg%256==0 && count_reg!=0)
            begin
            one_second_reg<=1;

            if(count_reg!=15360)
                count_reg<=count_reg+1;

            end
        else
            begin
            count_reg<=count_reg+1;
            one_second_reg<=0;
            end

    end

    //one minute
    always @(posedge clk or posedge reset)
```

```
    begin
        if(reset==1 || reset_count==1)
            begin
            count_reg<=0;
            one_minute_reg<=0;
            end

        else if(reset_count==1)
            begin
            count_reg<=0;
            one_minute_reg<=0;
            end

        else if(count_reg==15360)
            begin
            one_minute_reg<=1;
            count_reg<=0;
            end
        else
            one_minute_reg<=0;

    end


    assign one_second=one_second_reg;
    assign one_minute=(fast_watch==1)?one_second_reg:one_minute_reg;


endmodule
```

TB - ACLK-TIMING GENERTAOR

```
module tb_aclk_timegen();

    reg clk,reset,fast_watch,reset_count;
    wire one_minute,one_second;

initial
clk=0;
always #5 clk=~clk;

aclk_timegen
DUT(.clk(clk),.reset(reset),.reset_count(reset_count),.fast_watch(fast_watch)
,.one_minute(one_minute),.one_second(one_second));


    task initialize;
        {reset,fast_watch,reset_count}=0;

    endtask


initial
    begin
```

```
    initialize;

    @(negedge clk);
        reset=1;
    @(negedge clk);
        reset=0;
    #16000;
/*
    @(negedge clk);
        fast_watch=1;
    #1000;
    @(negedge clk);
        fast_watch=0;
    #100;
*/
    end


endmodule;
```

RTL – KEY REG

```
module keyreg(reset,
              clock,
              shift,
              key,
              key_buffer_ls_min,
              key_buffer_ms_min,
              key_buffer_ls_hr,
              key_buffer_ms_hr);
// Define input and output port direction

input reset,clock,shift;
input [3:0]key;

output [3:0]  key_buffer_ls_min,
              key_buffer_ms_min,
              key_buffer_ls_hr,
              key_buffer_ms_hr;


reg [3:0]     key_buffer_ls_min_reg,
              key_buffer_ms_min_reg,
              key_buffer_ls_hr_reg,
              key_buffer_ms_hr_reg;


always @(posedge clock or posedge reset)
begin
  // For asynchronous reset, reset the key_buffer output register to 1'b0
    if(reset==1)
    begin
          key_buffer_ls_min_reg<=0;
              key_buffer_ms_min_reg<=0;
```

```
            key_buffer_ls_hr_reg<=0;
            key_buffer_ms_hr_reg<=0;
    end

    else if(shift==1 && key!=10)
    begin
          key_buffer_ls_min_reg<=key;
              key_buffer_ms_min_reg<=key_buffer_ls_min_reg;
              key_buffer_ls_hr_reg<=key_buffer_ms_min_reg;
              key_buffer_ms_hr_reg<=key_buffer_ls_hr_reg;
    end


  // Else if there is a shift, perform left shift from LS_MIN to MS_HR

end


assign key_buffer_ls_min=key_buffer_ls_min_reg;
assign key_buffer_ms_min=key_buffer_ms_min_reg;
assign key_buffer_ls_hr=key_buffer_ls_hr_reg;
assign key_buffer_ms_hr=key_buffer_ms_hr_reg;

endmodule
```

RTL - COUNTER

```
module counter (clk,
            reset,
        one_minute,
        load_new_c,
        new_current_time_ms_hr,
        new_current_time_ms_min,
        new_current_time_ls_hr,
        new_current_time_ls_min,
        current_time_ms_hr,
        current_time_ms_min,
        current_time_ls_hr,
        current_time_ls_min);
// Define input and output port directions
input clk,reset,one_minute,load_new_c;

input  [3:0] new_current_time_ms_hr,
         new_current_time_ms_min,
         new_current_time_ls_hr,
         new_current_time_ls_min;

output [3:0]    current_time_ms_hr,
        current_time_ms_min,
        current_time_ls_hr,
        current_time_ls_min;
// Define register to store current time

reg [3:0]   current_time_ms_hr_reg,
```

```
        current_time_ms_min_reg,
        current_time_ls_hr_reg,
        current_time_ls_min_reg;


// Lodable Binary up synchronous Counter logic
/*****************************

// Write an always block with asynchronous reset
always@( posedge clk or posedge reset)
 begin
    // Check for reset signal and upon reset load the current_time register
with default value (1'b0)
    if(reset==1)
    {current_time_ms_hr_reg,
        current_time_ms_min_reg,
        current_time_ls_hr_reg,
        current_time_ls_min_reg}=0;

    // Else if there is no reset, then check for load_new_c signal and load
new_current_time to current_time register
      else if(load_new_c==1)
    begin
        current_time_ms_hr_reg<=new_current_time_ms_hr;
        current_time_ms_min_reg<=new_current_time_ms_min;
        current_time_ls_hr_reg<=new_current_time_ls_hr;
        current_time_ls_min_reg<=new_current_time_ls_min;
    end     //              0       0       0       9  -> 00:10
    // Else if there is no load_new_c signal, then check for one_minute
signal and implement the counting algorithm
    else if(one_minute==1)
    begin

        if(current_time_ms_hr_reg==2 && current_time_ls_hr_reg==3 &&
           current_time_ms_min_reg==5 && current_time_ls_min_reg==9)

            begin

            current_time_ms_hr_reg<=0;
            current_time_ls_hr_reg<=0;
            current_time_ms_min_reg<=0;
            current_time_ls_min_reg<=0;

            end

        else if(current_time_ms_min_reg==5 && current_time_ls_min_reg==9)
            begin

            if(current_time_ls_hr_reg==9)
                begin

                current_time_ms_hr_reg<=current_time_ms_hr_reg+1;
                current_time_ls_hr_reg<=0;
                current_time_ms_min_reg<=0;
                current_time_ls_min_reg<=0;
```

```
                end
            else
                begin
                current_time_ls_hr_reg<=current_time_ls_hr_reg+1;
                current_time_ms_min_reg<=0;
                current_time_ls_min_reg<=0;
                end

            end

        else if(current_time_ls_min_reg==9)
            begin

                current_time_ms_min_reg<=current_time_ms_min_reg+1;
                current_time_ls_min_reg<=0;
            end

        else
            current_time_ls_min_reg<=current_time_ls_min_reg+1;

    end
    // Check for the corner case
    // If the current_time is 23:59, then the next current_time will be 00:00

    // Else check if the current_time is 09:59, then the next current_time
will be 10:00

    // Else check if the time represented by minutes is 59, Increment the
LS_HR by 1 and set MS_MIN and LS_MIN to 1'b0

    // Else check if the LS_MIN is equal to 9, Increment the MS_MIN by 1 and
set MS_MIN to 1'b0

    // Else just increment the LS_MIN by 1

    end

assign   current_time_ms_hr=current_time_ms_hr_reg;
assign   current_time_ms_min=current_time_ms_min_reg;
assign   current_time_ls_hr=current_time_ls_hr_reg;
assign   current_time_ls_min=current_time_ls_min_reg;

endmodule
```

TB – COUNTER

```
module tb_counter();

reg clk,reset,one_minute,load_new_c;

reg  [3:0]   new_current_time_ms_hr,
         new_current_time_ms_min,
         new_current_time_ls_hr,
         new_current_time_ls_min;
```

```
wire [3:0]  current_time_ms_hr,
        current_time_ms_min,
        current_time_ls_hr,
        current_time_ls_min;
initial
clk=0;
always #5 clk=~clk;


counter DUT(    .clk(clk),
            .reset(reset),
        .one_minute(one_minute),
        .load_new_c(load_new_c),
        .new_current_time_ms_hr(new_current_time_ms_hr),
        .new_current_time_ms_min(new_current_time_ms_min),
        .new_current_time_ls_hr(new_current_time_ls_hr),
        .new_current_time_ls_min(new_current_time_ls_min),
        .current_time_ms_hr(current_time_ms_hr),
        .current_time_ms_min(current_time_ms_min),
        .current_time_ls_hr(current_time_ls_hr),
        .current_time_ls_min(current_time_ls_min)
       );
task initialize();
    begin

    {reset,one_minute,load_new_c}=0;

    {new_current_time_ms_hr,
         new_current_time_ms_min,
         new_current_time_ls_hr,
         new_current_time_ls_min}=0;

    end
endtask

initial
    begin
        initialize;
        @(negedge clk);
        reset=1;
        @(negedge clk);
        reset=0;load_new_c=1;one_minute=1;
        new_current_time_ms_hr=2;
            new_current_time_ls_hr=3;
            new_current_time_ms_min=4;
            new_current_time_ls_min=5;
        @(negedge clk);
        load_new_c=0;

        #10000;


    end


endmodule
```

RTL – ALARM REG

```
module alarm_reg (new_alarm_ms_hr,

              new_alarm_ls_hr,
              new_alarm_ms_min,
              new_alarm_ls_min,
              load_new_alarm,
              clock,
              reset,
              alarm_time_ms_hr,
              alarm_time_ls_hr,
              alarm_time_ms_min,
              alarm_time_ls_min );

// Define input and output port directions
input   [3:0]new_alarm_ms_hr,
             new_alarm_ls_hr,
             new_alarm_ms_min,
             new_alarm_ls_min;

input load_new_alarm,clock,reset;

output reg  [3:0]alarm_time_ms_hr,
                     alarm_time_ls_hr,
                     alarm_time_ms_min,
                     alarm_time_ls_min;


always @ (posedge clock or posedge reset)

begin
  // Upon reset, store reset value(1'b0) to the alarm_time registers
    if(reset==1)
        begin
            {alarm_time_ms_hr,
                     alarm_time_ls_hr,
                     alarm_time_ms_min,
                     alarm_time_ls_min}=0;

        end
  // Else if no reset, check for load_new_alarm signal and load new_alarm
time to alarm_time registers
    else if(load_new_alarm==1)
        begin
             alarm_time_ms_hr<=new_alarm_ms_hr;
                     alarm_time_ls_hr<=new_alarm_ls_hr;
                     alarm_time_ms_min<=new_alarm_ms_min;
                     alarm_time_ls_min<=new_alarm_ls_min;
        end


end

endmodule
```

TB - ALARM REG

```
module alarm_reg_tb ();

// Define input and output port directions
reg     [3:0]new_alarm_ms_hr,
             new_alarm_ls_hr,
             new_alarm_ms_min,
             new_alarm_ls_min;

reg load_new_alarm,clock,reset;

wire        [3:0]alarm_time_ms_hr,
                 alarm_time_ls_hr,
                 alarm_time_ms_min,
             alarm_time_ls_min;

initial
clock=0;
always #5 clock=~clock;




alarm_reg DUT(.new_alarm_ms_hr(new_alarm_ms_hr),
              .new_alarm_ls_hr(new_alarm_ls_hr),
              .new_alarm_ms_min(new_alarm_ms_min),
              .new_alarm_ls_min(new_alarm_ls_min),
              .load_new_alarm(load_new_alarm),
              .clock(clock),
              .reset(reset),
              .alarm_time_ms_hr(alarm_time_ms_hr),
              .alarm_time_ls_hr(alarm_time_ls_hr),
              .alarm_time_ms_min(alarm_time_ms_min),
              .alarm_time_ls_min(alarm_time_ls_min));

initial
begin

@(negedge clock);
load_new_alarm=0;reset=1;
@(negedge clock);
load_new_alarm=1;reset=0;
        new_alarm_ms_hr=4'b0000;
        new_alarm_ls_hr=4'b0101;
                new_alarm_ms_min=4'b0011;
                new_alarm_ls_min=4'b0010;
@(negedge clock);
load_new_alarm=0;reset=0;
@(negedge clock);
```

```
end

initial
    $monitor("TIME IS %d%d:%d%d
HH:MM",new_alarm_ms_hr,new_alarm_ls_hr,new_alarm_ms_min,new_alarm_ls_min);


endmodule
```

RTL – ALARM CLOCK TOP MODULE

```
module alarm_clock_top(clock,
                       key,
                reset,
                time_button,
                alarm_button,
                fast_watch,
                ms_hour,
                ls_hour,
                ms_minute,
                ls_minute,
                alarm_sound);
// Define port directions for the signals
input clock,reset,time_button,alarm_button,fast_watch;
input [3:0]key;

output alarm_sound;

output [7:0] ms_hour,
         ls_hour,
         ms_minute,
         ls_minute;

//Define the Interconnecting internal wires
    //TIMING GENERTOR
wire one_minute,one_second,reset_count;
    //ALARM CONTROLLER
wire load_new_c,show_new_time,show_a,load_new_a,shift;

    //KEY REG
wire [3:0]    key_buffer_ls_min,
              key_buffer_ms_min,
              key_buffer_ls_hr,
              key_buffer_ms_hr;

    //ALARM,CURRENT TIME REGS
wire [3:0] alarm_time_ms_hr,
            alarm_time_ls_hr,
            alarm_time_ms_min,
            alarm_time_ls_min,
            current_time_ms_hr,
            current_time_ls_hr,
            current_time_ms_min,
```

```
            current_time_ls_min;



//Instantiate lower sub-modules. Use interconnect(Internal) signals for
connecting the sub modules
// Instantiate the timing generator module

aclk_timegen
TG(.clk(clock),.reset(reset),.reset_count(reset_count),.fast_watch(fast_watch
),.one_minute(one_minute),.one_second(one_second));




//FSM

aclk_controller
CTRL(.clk(clock),.reset(reset),.one_second(one_second),.alarm_button(alarm_bu
tton),.time_button(time_button),

.key(key),.load_new_c(load_new_c),.show_new_time(show_new_time),.show_a(show_
a),.load_new_a(load_new_a),
            .shift(shift),.reset_count(reset_count) );

// Instantiate the counter module
counter CNT(    .clk(clock),
            .reset(reset),
        .one_minute(one_minute),
        .load_new_c(load_new_c),
        .new_current_time_ms_hr(key_buffer_ms_hr),
        .new_current_time_ms_min(key_buffer_ms_min),
        .new_current_time_ls_hr(key_buffer_ls_hr),
        .new_current_time_ls_min(key_buffer_ls_min),
        .current_time_ms_hr(current_time_ms_hr),
        .current_time_ms_min(current_time_ms_min),
        .current_time_ls_hr(current_time_ls_hr),
        .current_time_ls_min(current_time_ls_min) );

// Instantiate the key register module

keyreg KEYREG(.reset(reset),
              .clock(clock),
              .shift(shift),
              .key(key),
              .key_buffer_ls_min(key_buffer_ls_min),
              .key_buffer_ms_min(key_buffer_ms_min),
              .key_buffer_ls_hr(key_buffer_ls_hr),
              .key_buffer_ms_hr(key_buffer_ms_hr) );


// Instantiate the alarm register module
alarm_reg A_REG(.new_alarm_ms_hr(key_buffer_ms_hr),
              .new_alarm_ls_hr(key_buffer_ls_hr),
```

```
                .new_alarm_ms_min(key_buffer_ms_min),
                .new_alarm_ls_min(key_buffer_ls_min),
                .load_new_alarm(load_new_a),
                .clock(clock),
                .reset(reset),
                .alarm_time_ms_hr(alarm_time_ms_hr),
                .alarm_time_ls_hr(alarm_time_ls_hr),
                .alarm_time_ms_min(alarm_time_ms_min),
                .alarm_time_ls_min(alarm_time_ls_min) );


// Instantiate the lcd_driver_4 module
lcd_driver_4 LCD(     .alarm_time_ms_hr(alarm_time_ms_hr),
                      .alarm_time_ls_hr(alarm_time_ls_hr),
                      .alarm_time_ms_min(alarm_time_ms_min),
                      .alarm_time_ls_min(alarm_time_ls_min),
                      .current_time_ms_hr(current_time_ms_hr),
                      .current_time_ls_hr(current_time_ls_hr),
                      .current_time_ms_min(current_time_ms_min),
                      .current_time_ls_min(current_time_ls_min),
                      .key_ms_hr(key_buffer_ms_hr),
                      .key_ls_hr(key_buffer_ls_hr),
                      .key_ms_min(key_buffer_ms_min),
                      .key_ls_min(key_buffer_ls_min),
                      .show_a(show_a),
                      .show_current_time(show_new_time),
                      .display_ms_hr(ms_hour),
                      .display_ls_hr(ls_hour),
                      .display_ms_min(ms_minute),
                      .display_ls_min(ls_minute),
                      .sound_a(alarm_sound)
        );

endmodule
```

TB – ALARM CLOCK TOP MODULE

```
`timescale 1ms/1ns

module tb_alarm_clock();

reg clk,
    reset,
    fast_watch,
    alarm_button,
    time_button;

reg [3:0] key;
```

```
wire [7:0] display_ms_hr,
           display_ms_min,
           display_ls_hr,
           display_ls_min;

wire sound_alarm;

parameter cycle = 3.90625;


alarm_clock_top DUV(.clock(clk),
                    .reset(reset),
                    .fast_watch(fast_watch),
                    .alarm_button(alarm_button),
                    .time_button(time_button),
                    .key(key),
                    .alarm_sound(sound_alarm),
                    .ms_hour(display_ms_hr),
                    .ls_hour(display_ls_hr),
                    .ms_minute(display_ms_min),
                    .ls_minute(display_ls_min));



task initialize;
    begin

    {alarm_button,time_button,key,fast_watch}=0;

    end
endtask

 //Clock generation logic
 initial
  begin
     clk = 1'b0;
     forever
     #(cycle/2) clk = ~clk;
   end

 //

 //Stimulus logic
 initial
  begin
    initialize;
   //Hard reset the design
    reset = 1;
    #10;
    reset = 0;
   //Set fastwatch to 1 to make counting faster
   fast_watch = 1;
   //Set  key time to current time :11:23
   key = 1;
   repeat(3)
   @(negedge clk);
```

```
    key = 10;
    @(negedge clk);
    key = 1;
    repeat(3)
    @(negedge clk);
    key = 10;
    @(negedge clk);
    key = 2;
    repeat(3)
    @(negedge clk);
    key = 10;
    @(negedge clk);
    key = 3;
    repeat(3)
    @(negedge clk);
    key = 10;
    @(negedge clk);
    time_button = 1;
    @(negedge clk);
    time_button = 0;

    //Set  key time to alarm time :11:30
    key = 1;
    repeat(3)
    @(negedge clk);
    key = 10;
    @(negedge clk);
    key = 1;
    repeat(3)
    @(negedge clk);
    key = 10;
    @(negedge clk);
    key = 3;
    repeat(3)
    @(negedge clk);
    key = 10;
    @(negedge clk);
    key = 0;
    repeat(3)
    @(negedge clk);
    key = 10;
    @(negedge clk);
    alarm_button = 1;
    @(negedge clk);
    alarm_button = 0;
    #(7*256*2);//7 -> 7minutes ->7seconds->7 *256 clock cycles ->7*256*2(Time
period of clock)
    //Time out for Alarm clock
     //key = 7;
     repeat(4*2564) //Wait for minimum 10second pulses i.e (10*256) clock
cycles
     @(negedge clk);
    $finish;
  end

 initial
```

```
  $monitor($time,"\-ns\t MAVEN SILICON : \tDISPLAY_MS_HR =%H >>>
DISPLAY_LS_HR =%H>>> DISPLAY_MS_MIN =%H>>>
DISPLAY_LS_MIN=%H",display_ms_hr[3:0],display_ls_hr[3:0],display_ms_min[3:0],
display_ls_min[3:0]);

endmodule
```

Terminal Output

```
#                    0-ns    MAVEN SILICON :    DISPLAY_MS_HR =0 >>>
DISPLAY_LS_HR =0>>> DISPLAY_MS_MIN =0>>> DISPLAY_LS_MIN=0
#                   18-ns    MAVEN SILICON :    DISPLAY_MS_HR =0 >>>
DISPLAY_LS_HR =0>>> DISPLAY_MS_MIN =0>>> DISPLAY_LS_MIN=1
#                   29-ns    MAVEN SILICON :    DISPLAY_MS_HR =0 >>>
DISPLAY_LS_HR =0>>> DISPLAY_MS_MIN =1>>> DISPLAY_LS_MIN=1
#                   45-ns    MAVEN SILICON :    DISPLAY_MS_HR =0 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =1>>> DISPLAY_LS_MIN=2
#                   61-ns    MAVEN SILICON :    DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =2>>> DISPLAY_LS_MIN=3
#                   84-ns    MAVEN SILICON :    DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =2>>> DISPLAY_MS_MIN =3>>> DISPLAY_LS_MIN=1
#                   96-ns    MAVEN SILICON :    DISPLAY_MS_HR =2 >>>
DISPLAY_LS_HR =3>>> DISPLAY_MS_MIN =1>>> DISPLAY_LS_MIN=1
#                  111-ns    MAVEN SILICON :    DISPLAY_MS_HR =3 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =1>>> DISPLAY_LS_MIN=3
#                  127-ns    MAVEN SILICON :    DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =3>>> DISPLAY_LS_MIN=0
#                  143-ns    MAVEN SILICON :    DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =2>>> DISPLAY_LS_MIN=3
run
#                 1018-ns    MAVEN SILICON :    DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =2>>> DISPLAY_LS_MIN=4
run
#                 2018-ns    MAVEN SILICON :    DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =2>>> DISPLAY_LS_MIN=5
run
#                 3018-ns    MAVEN SILICON :    DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =2>>> DISPLAY_LS_MIN=6
run
#                 4018-ns    MAVEN SILICON :    DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =2>>> DISPLAY_LS_MIN=7
run
#                 5018-ns    MAVEN SILICON :    DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =2>>> DISPLAY_LS_MIN=8
run
#                 6018-ns    MAVEN SILICON :    DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =2>>> DISPLAY_LS_MIN=9
run
#                 7018-ns    MAVEN SILICON :    DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =3>>> DISPLAY_LS_MIN=0
run
#                 8018-ns    MAVEN SILICON :    DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =3>>> DISPLAY_LS_MIN=1
run
#                 9018-ns    MAVEN SILICON :    DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =3>>> DISPLAY_LS_MIN=2
```

```
run
#                 10018-ns     MAVEN SILICON :     DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =3>>> DISPLAY_LS_MIN=3
run
#                 11018-ns     MAVEN SILICON :     DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =3>>> DISPLAY_LS_MIN=4
run
#                 12018-ns     MAVEN SILICON :     DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =3>>> DISPLAY_LS_MIN=5
run
#                 13018-ns     MAVEN SILICON :     DISPLAY_MS_HR =1 >>>
DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =3>>> DISPLAY_LS_MIN=6
```

Waveform