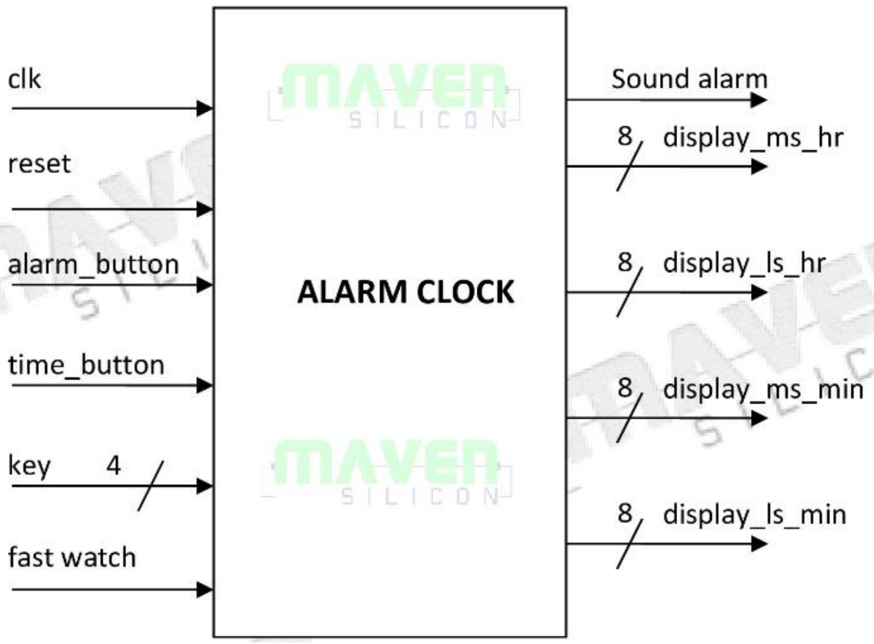
“<u>Name</u>”:-Seelam Chinna Venkata Narayana Reddy

“VIT-DM+DI RISCV Assignment-2”

Alarm Clock

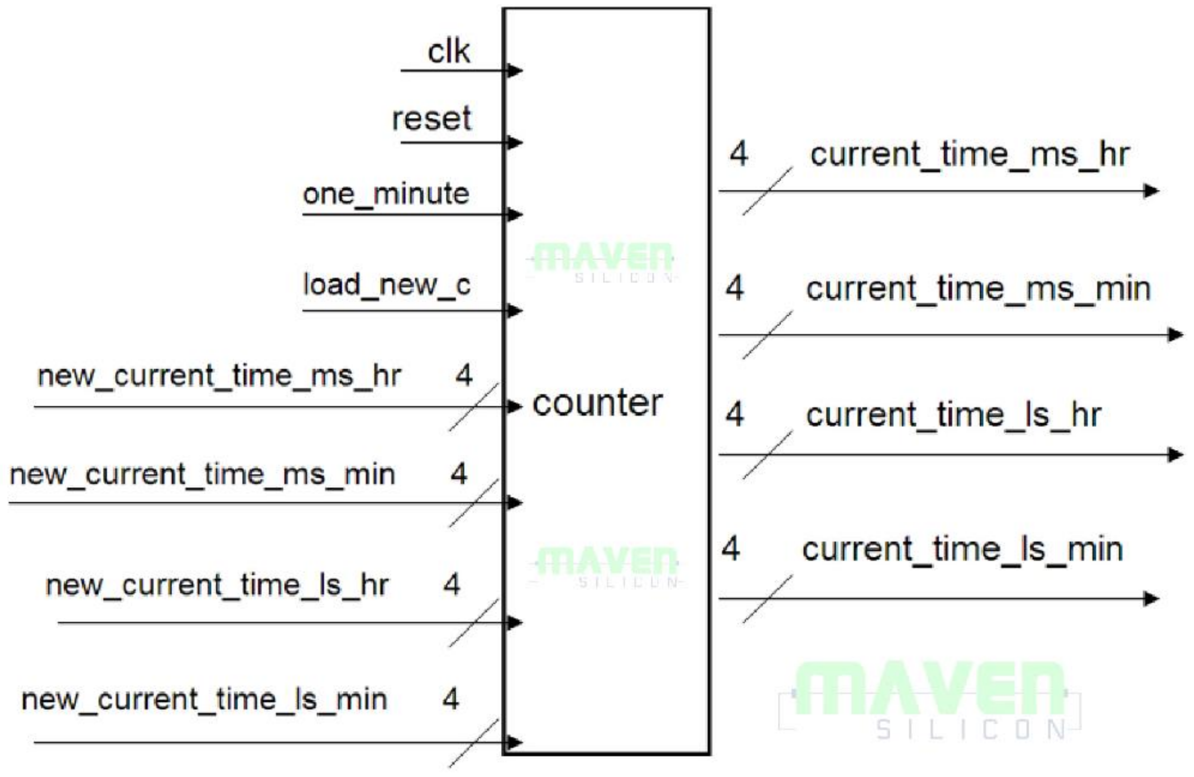
A digital alarm clock displays time in the LCD display format.

Block Diagram



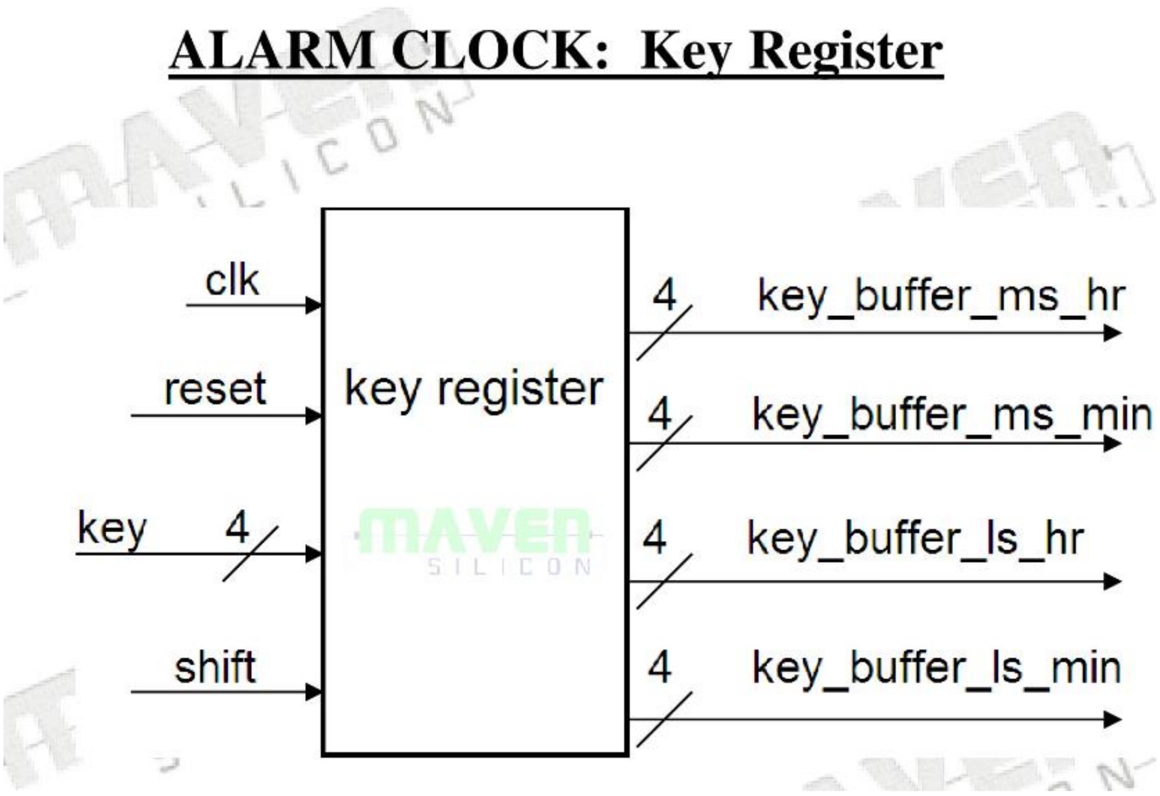
```
module alarm_clock_top(clock, key, reset, time_button, alarm_button, fastwatch, ms_hour, ls_hour, ms_minute, ls_minute, alarm_sound);
  input clock, reset, time_button, alarm_button, fastwatch;
  input [3:0] key;
  output [7:0] ms_hour, ls_hour, ms_minute, ls_minute;
  output alarm_sound;
  wire one_second, one_minute, load_new_c, load_new_a, show_current_time, show_a, shift, reset_count;
  wire [3:0] key_buffer_ms_hr, key_buffer_ls_hr, key_buffer_ms_min, key_buffer_ls_min, current_time_ms_hr, current_time_ls_hr,
  current_time_ms_min, current_time_ls_min, alarm_time_ms_hr, alarm_time_ls_hr, alarm_time_ms_min, alarm_time_ls_min;

  timegen tgen1 (.clock(clock), .reset(reset), .fastwatch(fastwatch), .one_second(one_second), .one_minute(one_minute), .reset_count(reset_count));

  counter count1 (.one_minute(one_minute), .new_current_time_ms_min(key_buffer_ms_min), .new_current_time_ls_min(key_buffer_ls_min),
  .new_current_time_ms_hr(key_buffer_ms_hr), .new_current_time_ls_hr(key_buffer_ls_hr), .load_new_c(load_new_c), .clk(clock), .reset(reset),
  .current_time_ms_min(current_time_ms_min), .current_time_ls_min(current_time_ls_min),.current_time_ms_hr(current_time_ms_hr),
  .current_time_ls_hr(current_time_ls_hr));

  alarm_reg alreg1 (.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));

  keyreg keyreg1 (.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));

  fsm fsm1 (.clock(clock), .reset(reset), .one_second(one_second), .time_button(time_button), .alarm_button(alarm_button), .key(key),
  .load_new_a(load_new_a), .show_a(show_a), .reset_count(reset_count), .show_new_time(show_current_time), .load_new_c(load_new_c), .shift(shift));

  lcd_driver_4 lcd_disp (.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_current_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
```

ALARM CLOCK: Counter



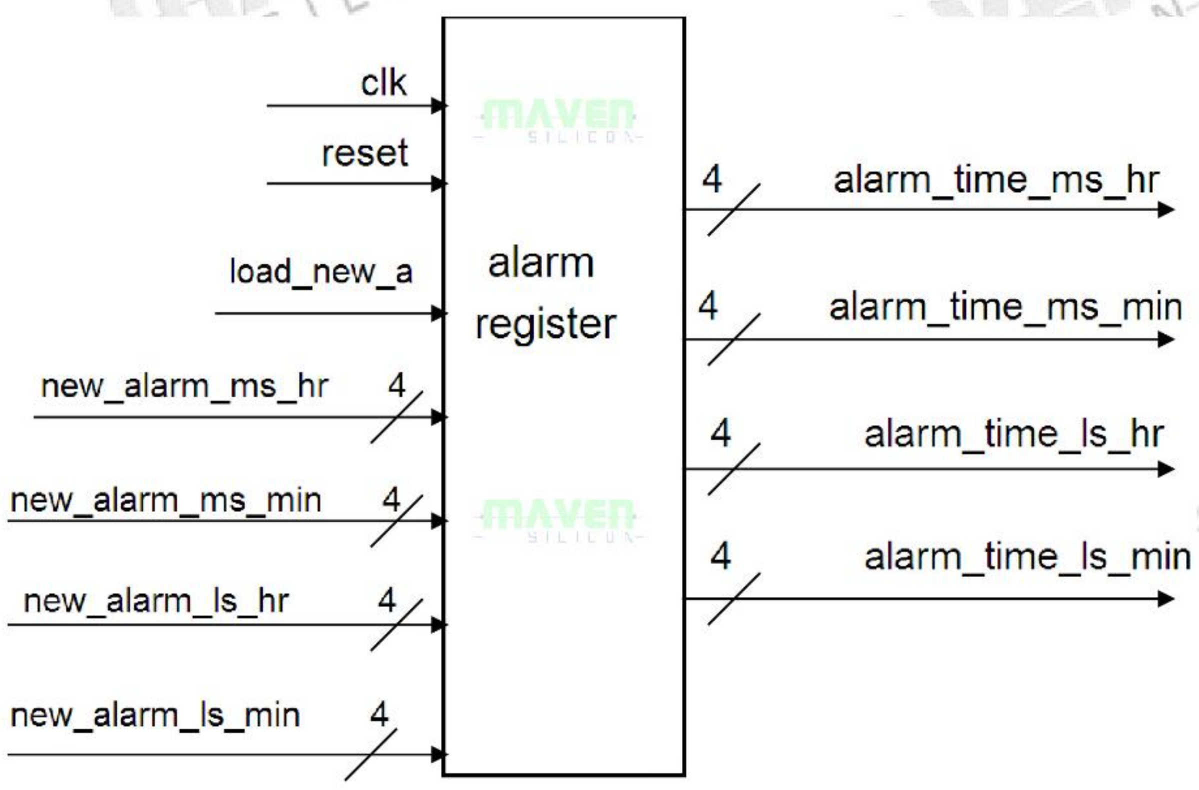
```verilog
module counter (input clk,input reset,input one_minute,input load_new_c,input [3:0] new_current_time_ms_hr,input [3:0] new_current_time_ms_min,
input [3:0] new_current_time_ls_hr,input [3:0] new_current_time_ls_min,output reg [3:0] current_time_ms_hr,output reg [3:0] current_time_ms_min,
  output reg [3:0] current_time_ls_hr,output reg [3:0] current_time_ls_min
);
  always @(posedge clk or posedge reset) begin
    if (reset) begin current_time_ms_hr <= 4'd0; current_time_ms_min <= 4'd0; current_time_ls_hr <= 4'd0; current_time_ls_min <= 4'd0;end
    else if (load_new_c) begin current_time_ms_hr <= new_current_time_ms_hr; current_time_ms_min <= new_current_time_ms_min;
      current_time_ls_hr <= new_current_time_ls_hr; current_time_ls_min <= new_current_time_ls_min;
    end
    else if (one_minute == 1) begin
      if (current_time_ms_hr == 4'd2 && current_time_ms_min == 4'd5 && current_time_ls_hr == 4'd3 && current_time_ls_min == 4'd9) begin
        current_time_ms_hr <= 4'd0; current_time_ms_min <= 4'd0; current_time_ls_hr <= 4'd0; current_time_ls_min <= 4'd0;
      end
      else if (current_time_ls_hr == 4'd9 && current_time_ms_min == 4'd5 && current_time_ls_min == 4'd9) begin
        current_time_ms_hr <= current_time_ms_hr + 1'd1; current_time_ls_hr <= 4'd0;current_time_ms_min <= 4'd0; current_time_ls_min <= 4'd0;
      end
      else if (current_time_ms_min == 4'd5 && current_time_ls_min == 4'd9) begin
        current_time_ls_hr <= current_time_ls_hr + 1'd1; current_time_ms_min <= 4'd0;current_time_ls_min <= 4'd0;
      end
      else if (current_time_ls_min == 4'd9) begin
        current_time_ms_min <= current_time_ms_min + 1'd1;current_time_ls_min <= 4'd0;
      end
      else begin
        current_time_ls_min <= current_time_ls_min + 1'b1;
      end
    end
  end
endmodule
```

ALARM CLOCK: Key Register



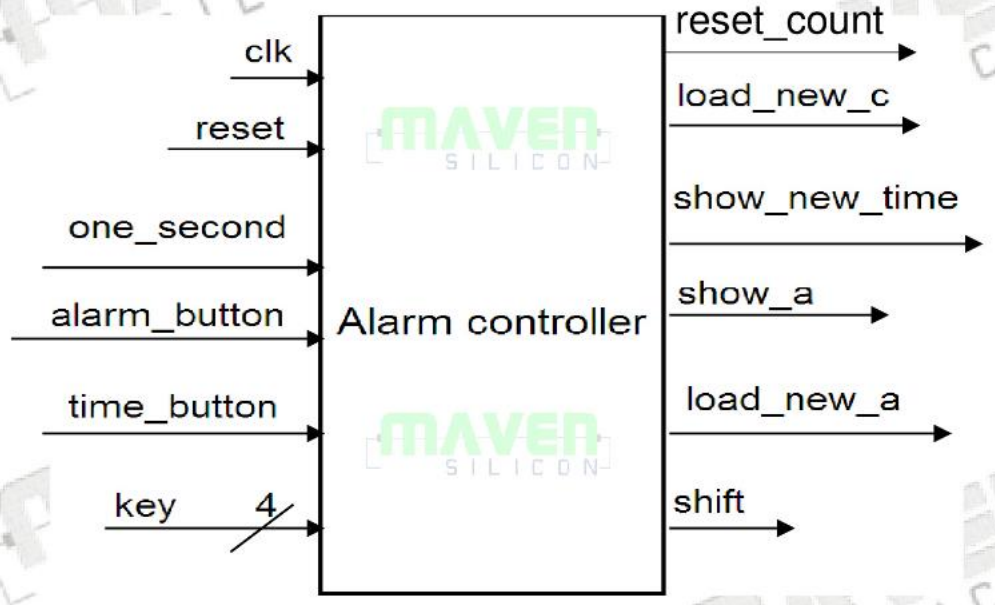
```
module keyreg(
  input reset,
  clock,
  shift,
  input [3:0] key,
  output reg [3:0] key_buffer_ls_min,
  key_buffer_ms_min,
  key_buffer_ls_hr,
  key_buffer_ms_hr
);

always @(posedge clock or posedge reset) begin
  if (reset) begin
    key_buffer_ls_min <= 0;
    key_buffer_ms_min <= 0;
    key_buffer_ls_hr <= 0;
    key_buffer_ms_hr <= 0;
  end else if (shift == 1) begin
    key_buffer_ms_hr <= key_buffer_ls_hr;
    key_buffer_ls_hr <= key_buffer_ms_min;
    key_buffer_ms_min <= key_buffer_ls_min;
    key_buffer_ls_min <= key;
  end
end
endmodule
```

ALARM CLOCK: The Alarm Register



```
module alarm_reg (
  input [3:0] new_alarm_ms_hr,
  input [3:0] new_alarm_ls_hr,
  input [3:0] new_alarm_ms_min,
  input [3:0] new_alarm_ls_min,
  input load_new_alarm,
  input clock,
  input reset,
  output reg [3:0] alarm_time_ms_hr,
  output reg [3:0] alarm_time_ls_hr,
  output reg [3:0] alarm_time_ms_min,
  output reg [3:0] alarm_time_ls_min
);
  always @(posedge clock or posedge reset) begin
    if (reset) begin
      alarm_time_ms_hr <= 4'b0;
      alarm_time_ls_hr <= 4'b0;
      alarm_time_ms_min <= 4'b0;
      alarm_time_ls_min <= 4'b0;
    end
    else if (load_new_alarm) 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
```

ALARM CLOCK: Controller Unit

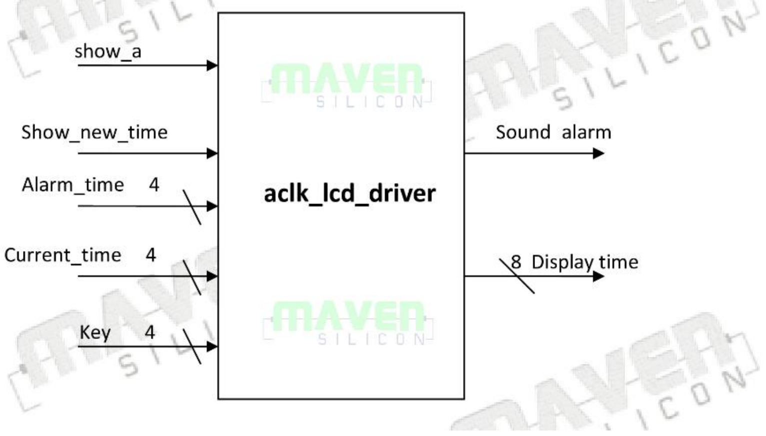


```verilog
module fsm (clock,reset,one_second,time_button,alarm_button,key,reset_count,load_new_a,show_a,show_new_time,load_new_c,shift);
input clock,reset, one_second,time_button,alarm_button;
input [3:0] key;
output load_new_a,show_a,show_new_time,load_new_c,shift,reset_count;
reg [2:0] pre_state,next_state;
wire time_out;
reg [3:0] count1,count2;
parameter SHOW_TIME         = 3'b000;
parameter KEY_ENTRY         = 3'b001;
parameter KEY_STORED        = 3'b010;
parameter SHOW_ALARM        = 3'b011;
parameter SET_ALARM_TIME    = 3'b100;
parameter SET_CURRENT_TIME  = 3'b101;
parameter KEY_WAITED        = 3'b110;
parameter NOKEY             = 10;
always @ (posedge clock or posedge reset)
begin
  if(reset) count1 <= 4'd0;
  else if(!(pre_state == KEY_ENTRY)) count1 <= 4'd0;
  else if (count1==9) count1 <= 4'd0;
  else if(one_second) count1 <= count1 + 1'b1;
end
always @ (posedge clock or posedge reset)
begin
  if(reset)
     count2 <= 4'd0;
     else if(!(pre_state == KEY_WAITED))  count2 <= 4'd0;
     else if (count2==9) count2 <= 4'd0;
  else if(one_second)
     count2 <= count2 + 1'b1;
end
assign time_out=((count1==9) || (count2==9)) ? 0 : 1;
```

```verilog
assign time_out=((count1==9) || (count2==9)) ? 0 : 1;
always @ (posedge clock or posedge reset)
begin
  if(reset) pre_state <= SHOW_TIME;
  else
     pre_state <= next_state;
end
always @(pre_state or key or alarm_button or time_button or time_out) begin
  case(pre_state)
    SHOW_TIME: begin
      if (alarm_button) next_state = SHOW_ALARM;
      else if (key != NOKEY) next_state = KEY_STORED;
      else next_state = SHOW_TIME;
    end
    KEY_STORED: next_state = KEY_WAITED;
    KEY_WAITED: begin
      if (key == NOKEY) next_state = KEY_ENTRY;
      else if (time_out == 0) next_state = SHOW_TIME;
      else next_state = KEY_WAITED;
    end
    KEY_ENTRY: begin
      if (alarm_button) next_state = SET_ALARM_TIME;
      else if (time_button) next_state = SET_CURRENT_TIME;
      else if (time_out == 0) next_state = SHOW_TIME;
      else if (key != NOKEY) next_state = KEY_STORED;
      else next_state = KEY_ENTRY;
    end
    SHOW_ALARM: next_state = alarm_button ? SHOW_ALARM : SHOW_TIME;
    SET_ALARM_TIME, SET_CURRENT_TIME: next_state = SHOW_TIME;
    default: next_state = SHOW_TIME;
  endcase
end
```

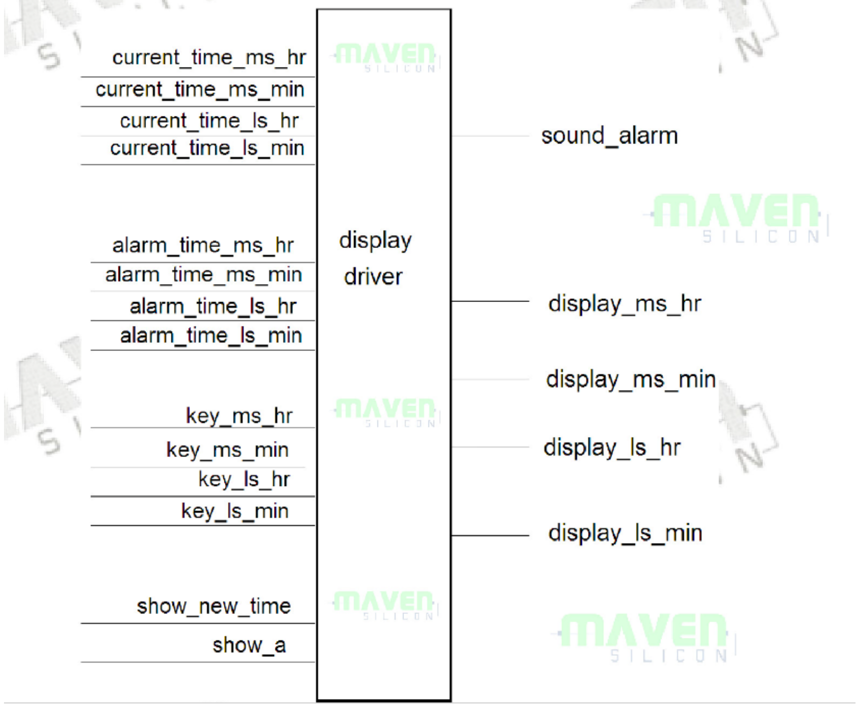
```
always @(posedge clock or posedge reset) begin
  if (reset) begin
    count1 <= 4'd0;
    count2 <= 4'd0;
  end
  else if (!(pre_state == KEY_ENTRY)) count1 <= 4'd0;
  else if (count1 == 9) count1 <= 4'd0;
  else if (one_second) count1 <= count1 + 1'b1;
end

always @(posedge clock or posedge reset) begin
  if (reset) begin
    count2 <= 4'd0;
  end
  else if (!(pre_state == KEY_WAITED)) count2 <= 4'd0;
  else if (count2 == 9) count2 <= 4'd0;
  else if (one_second) count2 <= count2 + 1'b1;
end

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

assign show_new_time = (pre_state == KEY_ENTRY || pre_state == KEY_STORED || pre_state == KEY_WAITED) ? 1 : 0;
assign show_a = (pre_state == SHOW_ALARM) ? 1 : 0;
assign load_new_a = (pre_state == SET_ALARM_TIME) ? 1 : 0;
assign load_new_c = (pre_state == SET_CURRENT_TIME) ? 1 : 0;
assign reset_count = (pre_state == SET_CURRENT_TIME) ? 1 : 0;
assign shift = (pre_state == KEY_STORED) ? 1 : 0;
endmodule
```

ALARM CLOCK: LCD Display Driver



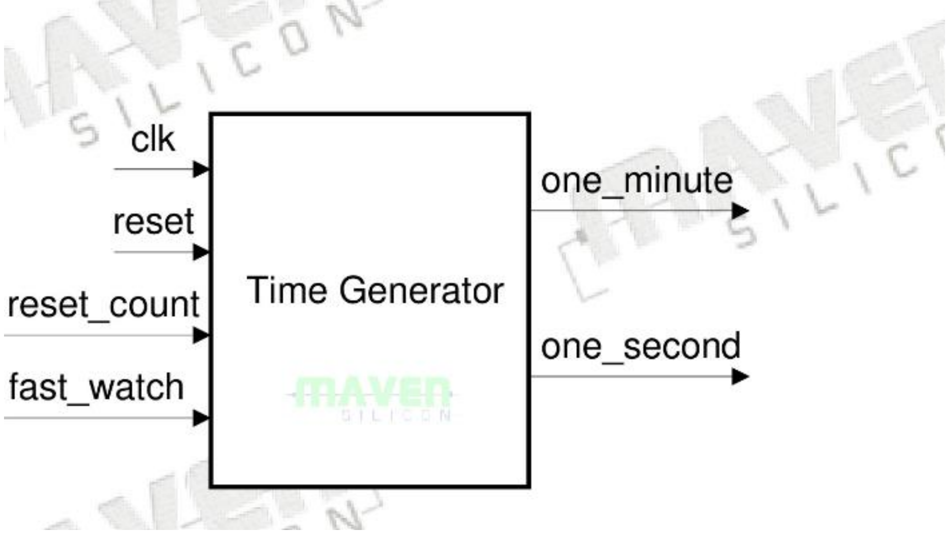
```
module lcd_driver(alarm_time, current_time, show_alarm, show_new_time, key, display_time, sound_alarm);
input [3:0] key;input [3:0] alarm_time;input [3:0] current_time;input show_alarm;input show_new_time;
output reg [7:0] display_time;output reg sound_alarm;
reg [3:0] display_value;
parameter ZERO = 8'h30, ONE = 8'h31, TWO = 8'h32, THREE = 8'h33, FOUR = 8'h34, FIVE = 8'h35, SIX = 8'h36,
 SEVEN = 8'h37, EIGHT = 8'h38, NINE = 8'h39, ERROR = 8'h3A;
always @(alarm_time or current_time or show_alarm or show_new_time or key) begin
  if (show_new_time) display_value = key;
  else if (show_alarm) display_value = alarm_time;
  else display_value = current_time;
  sound_alarm = (current_time == alarm_time) ? 1'b1 : 1'b0;
end
always @(display_value) begin
  case (display_value)
    4'd0 : display_time = ZERO;
    4'd1 : display_time = ONE;
    4'd2 : display_time = TWO;
    4'd3 : display_time = THREE;
    4'd4 : display_time = FOUR;
    4'd5 : display_time = FIVE;
    4'd6 : display_time = SIX;
    4'd7 : display_time = SEVEN;
    4'd8 : display_time = EIGHT;
    4'd9 : display_time = NINE;
    default : display_time = ERROR;
  endcase
end
endmodule
```

ALARM CLOCK: LCD Display Unit



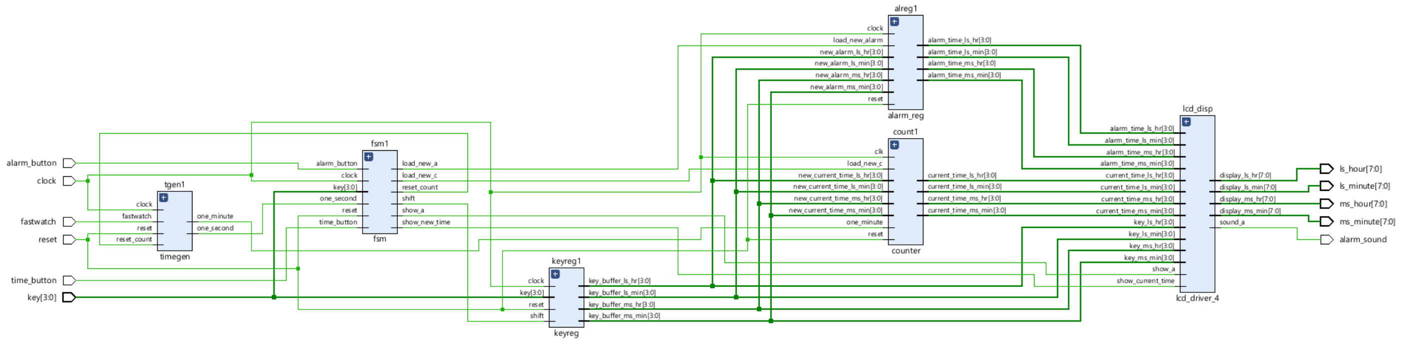
```
module lcd_driver_4 ( input [3:0] alarm_time_ms_hr,alarm_time_ls_hr,alarm_time_ms_min,alarm_time_ls_min,current_time_ms_hr,current_time_ls_hr,
            current_time_ms_min,current_time_ls_min,key_ms_hr,key_ls_hr,key_ms_min,key_ls_min,input show_a,input show_a,show_current_time,
            output [7:0] display_ms_hr,display_ls_hr,display_ms_min,display_ls_min);


wire sound_alarm1, sound_alarm2, sound_alarm3, sound_alarm4;
assign sound_a = sound_alarm1 & sound_alarm2 & sound_alarm3 & sound_alarm4;

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));

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));

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));

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));
endmodule
```

ALARM CLOCK: Time Generator



```
module timegen(input clock,reset,reset_count,fastwatch,output one_second,one_minute);
  reg [13:0] count; reg one_second;reg one_minute_reg;reg one_minute;
  reg [13:0] count;reg one_second;reg one_minute_reg;reg one_minute;
always@(posedge clock or posedge reset)
begin
  if (reset) begin count <= 14'b0; one_minute_reg <= 0;
  end
  else if (reset_count) begin count <= 14'b0; one_minute_reg <= 1'b0;
  end
  else if (count[13:0] == 14'd15359) begin count <= 14'b0; one_minute_reg <= 1'b1;
  end
  else begin
    count <= count + 1'b1; one_minute_reg <= 1'b0;
  end
end
always @(posedge clock or posedge reset) begin
  if (reset) begin one_second <= 1'b0;
  end
  else if (reset_count) begin one_second <= 1'b0;
  end
  else if (count[7:0] == 8'd255) begin one_second <= 1'b1;
  end
  else begin one_second <= 1'b0;
  end
end
always @(*) begin
  if (fastwatch) begin one_minute = one_second ; end
  else begin one_minute = one_minute_reg;
  end end
endmodule
```

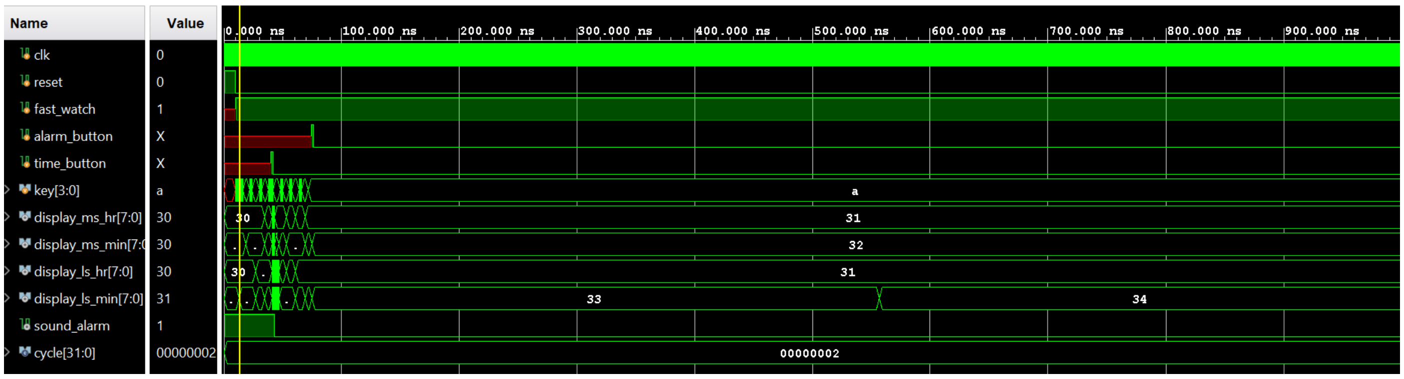
“Schematics”:-



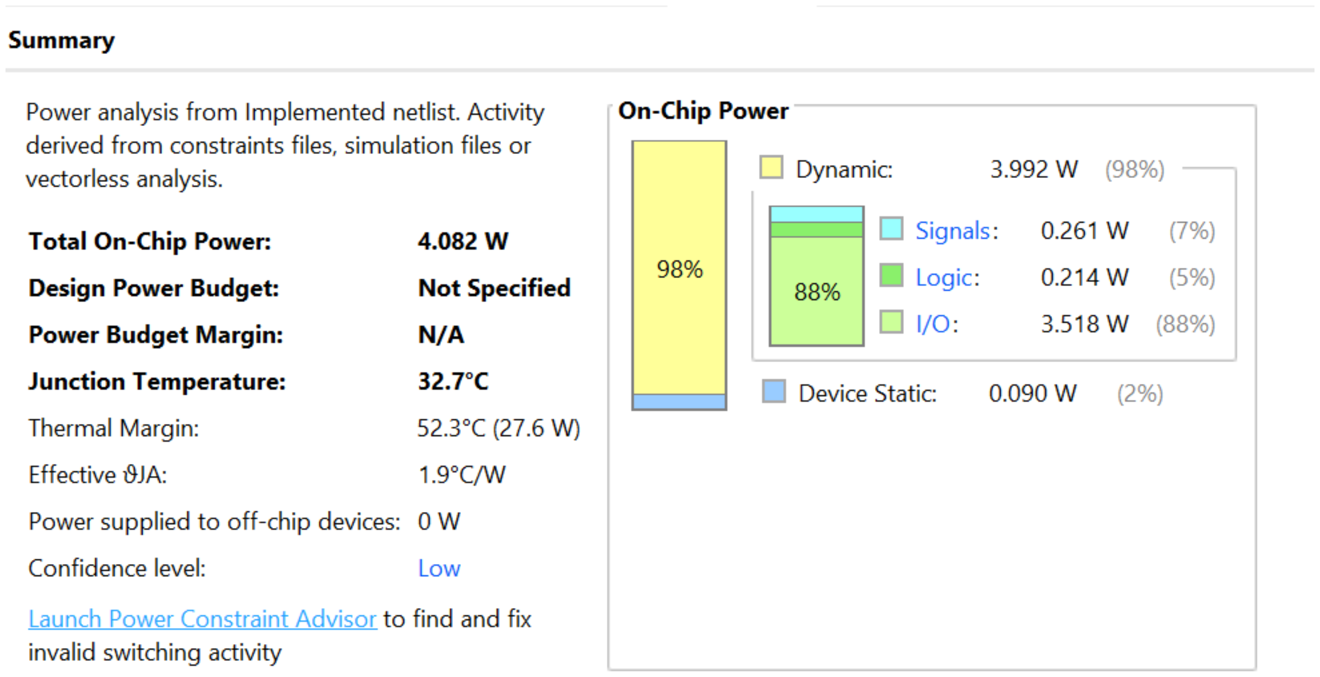
“Console”:-

```
  0-ns  MAVEN SILICON :    DISPLAY_MS_HR =0 >>> DISPLAY_LS_HR =0>>> DISPLAY_MS_MIN =0>>> DISPLAY_LS_MIN=0
 13-ns  MAVEN SILICON :    DISPLAY_MS_HR =0 >>> DISPLAY_LS_HR =0>>> DISPLAY_MS_MIN =0>>> DISPLAY_LS_MIN=1
 19-ns  MAVEN SILICON :    DISPLAY_MS_HR =0 >>> DISPLAY_LS_HR =0>>> DISPLAY_MS_MIN =1>>> DISPLAY_LS_MIN=1
 27-ns  MAVEN SILICON :    DISPLAY_MS_HR =0 >>> DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =1>>> DISPLAY_LS_MIN=2
 35-ns  MAVEN SILICON :    DISPLAY_MS_HR =1 >>> DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =2>>> DISPLAY_LS_MIN=3
 41-ns  MAVEN SILICON :    DISPLAY_MS_HR =0 >>> DISPLAY_LS_HR =0>>> DISPLAY_MS_MIN =0>>> DISPLAY_LS_MIN=0
 43-ns  MAVEN SILICON :    DISPLAY_MS_HR =1 >>> DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =2>>> DISPLAY_LS_MIN=3
 47-ns  MAVEN SILICON :    DISPLAY_MS_HR =1 >>> DISPLAY_LS_HR =2>>> DISPLAY_MS_MIN =3>>> DISPLAY_LS_MIN=1
 53-ns  MAVEN SILICON :    DISPLAY_MS_HR =2 >>> DISPLAY_LS_HR =3>>> DISPLAY_MS_MIN =1>>> DISPLAY_LS_MIN=1
 61-ns  MAVEN SILICON :    DISPLAY_MS_HR =3 >>> DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =1>>> DISPLAY_LS_MIN=3
 69-ns  MAVEN SILICON :    DISPLAY_MS_HR =1 >>> DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =3>>> DISPLAY_LS_MIN=0
 75-ns  MAVEN SILICON :    DISPLAY_MS_HR =1 >>> DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =2>>> DISPLAY_LS_MIN=3
557-ns  MAVEN SILICON :    DISPLAY_MS_HR =1 >>> DISPLAY_LS_HR =1>>> DISPLAY_MS_MIN =2>>> DISPLAY_LS_MIN=4
```

“Waveforms”:-



“Power Analysis”:-



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**4.082 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**32.7°C**
Thermal Margin:	52.3°C (27.6 W)
Effective ϑJA:	1.9°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:		3.992 W	(98%)
	Signals:	0.261 W	(7%)
	Logic:	0.214 W	(5%)
	I/O:	3.518 W	(88%)
Device Static:		0.090 W	(2%)