Alarm Clock Project

Submitted to

Maven Silicon

(Submitted for course

Foundation – VLSI Design - VIT)

July 13, 2024

By

R. Visnu

Registration number: MS/VIT/2024-25/351

Mail: visnuram2004@gmail.com

Alarm Clock Project

Submitted to

Maven Silicon

(Submitted for course

Foundation – VLSI Design - VIT)

July 13, 2024

By

R. Visnu

Registration number: MS/VIT/2024-25/351

Mail: visnuram2004@gmail.com

ABSTRACT

This project presents the design and implementation of a digital alarm clock using Verilog HDL. The alarm clock features core functionalities such as time display, alarm setting, and key-based input handling. The system is composed of several interconnected modules that perform specific roles, ensuring modularity and clarity in design.

CONTENTS

INTRODUCTION

This implementation of alarm clock in RTL is split up into multiple sub modules for each of development and testing. The top module is called `alarm_clk_rtl`.

The key modules include:

1. `aclk_timegen`: This module generates precise one-second and one-minute pulses, ensuring accurate timekeeping. It can operate in normal and fast-watch modes.

2. `aclk_counter`: Responsible for maintaining the current time, this module increments the time by one minute upon receiving the one-minute pulse. It can also load new current time values provided by user input.

3. `aclk_areg`: This module stores the alarm time, allowing users to set and update the alarm time as needed.

4. `aclk_keyreg`: Handles key inputs from the user, buffering the values for setting the current time and alarm time.

5. `aclk_controller`: The central control unit orchestrates the overall operation, managing state transitions and coordinating between different modules based on user inputs and internal signals.

6. `aclk_lcd_display`: This module drives the LCD display, showing either the current time or the alarm time based on the system state. It also handles the activation of the alarm sound when the current time matches the alarm time.

The design was implemented and verified using a comprehensive testbench for each module, ensuring each component operates correctly in isolation before integrating into the full system. The top-level module, `alarm_clk_rtl`, integrates all the individual modules, and its functionality was validated through an extensive testbench simulating typical user interactions, including setting the time and alarm, and observing the display and alarm sound outputs.

This modular approach not only simplifies the design process but also facilitates easier debugging and future enhancements. The project demonstrates effective use of hardware description languages for designing complex digital systems, showcasing a practical application in the form of a digital alarm clock.

RTL CODE

1. aclk_timegen.v

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

      input clk, reset, reset_count, fast_watch;

      output one_minute, one_second;


      reg [13:0] counter;

      reg one_second;

      reg one_minute_reg;

      reg one_minute;



      //one minute

      always @ (posedge clk or posedge reset)

            begin

                  if (reset)

                        begin

                              counter <= 14'b0;

                              one_minute_reg <= 0;

                        end

                  else if (reset_count)

                        begin

                              counter <= 14'b0;

                              one_minute_reg <= 1'b0;

                        end

                  else if (counter[13:0] == 14'd15359)

                        begin

                        counter <= 14'b0;

                        one_minute_reg <= 1'b1;

                        end

                  else
```

```
                begin
                        counter <= counter +  1'b1;
                        one_minute_reg <= 1'b0;
                end
        end

//one second
always @ (posedge clk or posedge reset)
        begin
                if (reset)
                        begin
                                one_second <= 0;
                        end
                else if (reset_count)
                        begin
                                one_second <= 1'b0;
                        end
                else if (counter[7:0] == 8'd255)
                        begin
                        one_second <= 1'b1;
                        end
                else
                        begin
                                one_second <= 1'b0;
                        end
        end

//fast watch
always @ (*)
        begin
                if (fast_watch)
                        one_minute = one_second;
                else one_minute = one_minute_reg;
```

```
            end


endmodule
```

2. aclk_counter.v

```
module aclk_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);


      input clk, reset, one_minute, load_new_c;


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


      output [3:0] current_time_ms_hr, current_time_ls_hr,
current_time_ms_min, current_time_ls_min;


      reg [3:0] current_time_ms_hr, current_time_ls_hr,
current_time_ms_min, current_time_ls_min;


      always @ (posedge clk or posedge reset)
            begin
                  if (reset)
                        begin
                              current_time_ms_hr <= 4'd0;
                              current_time_ls_hr <= 4'd0;
                              current_time_ms_min <= 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_ls_hr <= new_current_time_ls_hr;
                              current_time_ms_min <=
new_current_time_ms_min;
```

```
                              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_ls_hr == 4'd3 && current_time_ms_min == 4'd5 &&
current_time_ls_min == 4'd9)

                              //23:59

                                    begin

                                          current_time_ms_hr <= 4'd0;

                                          current_time_ls_hr <= 4'd0;

                                          current_time_ms_min <= 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)

                              //x9:59

                                    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)

                              //xx:59

                                    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)

                              //xx:x9

                                    begin
```

```
                                        current_time_ms_min <=
current_time_ms_min + 1'd1;
                                        current_time_ls_min <= 4'd0;
                                  end
                            else current_time_ls_min <=
current_time_ls_min + 1'd1;
                      end
            end
endmodule
```

3. aclk_areg.v

```
module aclk_areg(clk, reset, load_new_a, new_alarm_ms_hr, new_alarm_ls_hr,
new_alarm_ms_min, new_alarm_ls_min, alarm_time_ms_hr, alarm_time_ls_hr,
alarm_time_ms_min, alarm_time_ls_min);


      input clk, reset, load_new_a;
      input [3:0] new_alarm_ms_hr, new_alarm_ls_hr, new_alarm_ms_min,
new_alarm_ls_min;
      output reg [3:0] alarm_time_ms_hr, alarm_time_ls_hr,
alarm_time_ms_min, alarm_time_ls_min;


      always @ (posedge clk or posedge reset)
            begin
                  if (reset)
                        begin
                              alarm_time_ms_hr <= 4'd0;
                              alarm_time_ls_hr <= 4'd0;
                              alarm_time_ms_min <= 4'd0;
                              alarm_time_ls_min <= 4'd0;
                        end
                  else if (load_new_a)
                        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
```

4. aclk_keyreg.v

```
module aclk_keyreg(clock, shift, reset, key, key_ms_hr, key_ls_hr,
key_ms_min, key_ls_min);
      input clock, shift, reset;
      input [3:0] key;
      output reg [3:0] key_ms_hr, key_ls_hr, key_ms_min, key_ls_min;

      always @(posedge clock or posedge reset)
            begin
                  if (reset)
                        begin
                              key_ms_hr <= 0;
                              key_ls_hr <= 0;
                              key_ms_min <= 0;
                              key_ls_min <= 0;
                        end
                  else if (shift == 1)
                        begin
                              key_ms_hr <= key_ls_hr;
                              key_ls_hr <= key_ms_min;
                              key_ms_min <= key_ls_min;
                              key_ls_min <= key;
                        end
            end
endmodule
```

5. aclk_controller.v

```
module aclk_controller(clk, rst, one_second, alarm_button, time_button,
key, reset_count, load_new_c, show_new_time, show_a, load_new_a, shift);
```

```
	input clk, rst, one_second, alarm_button, time_button;
	input [3:0] key;

	output reset_count, load_new_c, show_new_time, show_a, load_new_a,
shift;

	reg [2:0] state, next_state;
	wire time_out;
	reg [3:0] count0, count1;

	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;

	//key entry 10s timer
	always @ (posedge clk or posedge rst)
		begin
			if (rst)
				count0 <= 4'd0;
			else if (state != KEY_ENTRY)
				count0<= 4'd0;
			else if (count0 == 9)
				count0 <= 4'd0;
			else if (one_second)
				count0 <= count0 + 1'b1;
		end

	//key waited 10s timer
	always @ (posedge clk or posedge rst)
		begin
```

```
                if (rst)
                        count1 <= 4'd0;
                else if (state != KEY_WAITED)
                        count1 <= 4'd0;
                else if (count1 == 9)
                        count1 <= 4'd0;
                else if (one_second)
                        count1 <= count1 + 1'b1;
        end

    assign time_out = ((count0 == 9) || (count1 == 9)) ? 0 : 1;

    //current state sequential logic
    always @ (posedge clk or posedge rst)
        begin
                if (rst)
                        state <= SHOW_TIME;
                else
                        state <= next_state;
        end

    //next state decoder

    always @ (state or key or alarm_button or time_button or time_out)
        begin
                case (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: begin
                              if (!alarm_button) next_state <= SHOW_TIME;
                              else next_state <= SHOW_ALARM;
                        end
                        SET_ALARM_TIME: next_state <= SHOW_TIME;
                        SET_CURRENT_TIME: next_state <=  SHOW_TIME;
                        default: next_state <= SHOW_TIME;
                  endcase
            end

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

endmodule
```

6. aclk_lcd_display.v

```
module aclk_lcd_display(
                        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,
                        key_ms_hr,
                        key_ls_hr,
                        key_ms_min,
                        key_ls_min,
                        show_new_time,
                        show_a,
                        sound_alarm,
                        display_ms_hr,
                        display_ls_hr,
                        display_ms_min,
                        display_ls_min
                              );

      input [3:0] 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,
```

```
                        key_ms_hr,
                        key_ls_hr,
                        key_ms_min,
                        key_ls_min;
      input show_new_time, show_a;


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


      wire sound_a1, sound_a2, sound_a3, sound_a4;
      assign sound_alarm = sound_a1 & sound_a2 & sound_a3 & sound_a4;


      aclk_lcd_driver MS_HR (alarm_time_ms_hr, current_time_ms_hr,
key_ms_hr, show_a, show_new_time, display_ms_hr, sound_a1);
      aclk_lcd_driver LS_HR (alarm_time_ls_hr, current_time_ls_hr,
key_ls_hr, show_a, show_new_time, display_ls_hr, sound_a2);
      aclk_lcd_driver MS_MIN (alarm_time_ms_min, current_time_ms_min,
key_ms_min, show_a, show_new_time, display_ms_min, sound_a3);
      aclk_lcd_driver LS_MIN (alarm_time_ls_min, current_time_ls_min,
key_ls_min, show_a, show_new_time, display_ls_min, sound_a4);


endmodule
```

7. aclk_lcd_driver.v

```
module aclk_lcd_driver(alarm_time, current_time, key, show_alarm,
show_new_time, display_time, sound_alarm);


      input [3:0] alarm_time, current_time, key;
      input show_alarm, show_new_time;


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

```
	reg [3:0] display_value;

	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;

	always @ (alarm_time or current_time or key 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;

			if (current_time == alarm_time) sound_alarm = 1'b1;
			else sound_alarm = 1'b0;
		end

	//decoder

	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
```

8. alarm_clk_rtl.v

```
module alarm_clk_rtl(clk, reset, alarm_button, time_button, key,
fast_watch, sound_alarm, display_ms_hr, display_ls_hr, display_ms_min,
display_ls_min);


      input clk, reset, alarm_button, time_button, fast_watch;
      input [3:0] key;


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


      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;


      aclk_timegen timegen (clk, reset, reset_count, fast_watch,
one_minute, one_second);
      aclk_counter counter (clk, reset, one_minute, load_new_c,
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);
```

```
      aclk_areg areg(clk, reset, load_new_a, key_buffer_ms_hr,
key_buffer_ls_hr, key_buffer_ms_min, key_buffer_ls_min, alarm_time_ms_hr,
alarm_time_ls_hr, alarm_time_ms_min, alarm_time_ls_min);

      aclk_keyreg keyreg(clk, shift, reset, key, key_buffer_ms_hr,
key_buffer_ls_hr, key_buffer_ms_min, key_buffer_ls_min);

      aclk_controller controller(clk, reset, one_second, alarm_button,
time_button, key, reset_count, load_new_c, show_current_time, show_a,
load_new_a, shift);

      aclk_lcd_display lcd_display(

                        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,

                        key_buffer_ms_hr,

                        key_buffer_ls_hr,

                        key_buffer_ms_min,

                        key_buffer_ls_min,

                        show_current_time,

                        show_a,

                        sound_alarm,

                        display_ms_hr,

                        display_ls_hr,

                        display_ms_min,

                        display_ls_min

                              );


Endmodule
```

TESTBENCH CODE

1. tb_aclk_timegen.v

```
module tb_aclk_timegen;

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

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

    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        reset = 0;
        reset_count = 0;
        fast_watch = 1;
        #10 reset = 0;
        #100000;
        reset_count = 1;
```

```
        #10 reset_count = 0;
        #100000;
        fast_watch = 1;
        #100000;
        fast_watch = 0;

    end

    initial begin
        $monitor("At time %t, reset = %b, reset_count = %b, fast_watch =
%b, one_minute = %b, one_second = %b",
                 $time, reset, reset_count, fast_watch, one_minute,
one_second);
    end

endmodule
```

2. tb_aclk_counter.v

```
module tb_aclk_counter;

    reg clk;
    reg reset;
    reg one_minute;
    reg load_new_c;
    reg [3:0] new_current_time_ms_hr;
    reg [3:0] new_current_time_ls_hr;
    reg [3:0] new_current_time_ms_min;
    reg [3:0] new_current_time_ls_min;

    wire [3:0] current_time_ms_hr;
    wire [3:0] current_time_ls_hr;
    wire [3:0] current_time_ms_min;
    wire [3:0] current_time_ls_min;

    aclk_counter uut (
```

```
    .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_ls_hr(new_current_time_ls_hr),
    .new_current_time_ms_min(new_current_time_ms_min),
    .new_current_time_ls_min(new_current_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)
);

initial begin
    clk = 0;
    forever #5 clk = ~clk;
end

initial begin
    reset = 1;
    one_minute = 0;
    load_new_c = 0;
    new_current_time_ms_hr = 4'd0;
    new_current_time_ls_hr = 4'd0;
    new_current_time_ms_min = 4'd0;
    new_current_time_ls_min = 4'd0;

    #10 reset = 0;

    #10 load_new_c = 1;
    new_current_time_ms_hr = 4'd1;
    new_current_time_ls_hr = 4'd2;
    new_current_time_ms_min = 4'd3;
```

```
        new_current_time_ls_min = 4'd4;
        #10 load_new_c = 0;

        #50 one_minute = 1;
        #10 one_minute = 0;

        #500000 one_minute = 1;
        #10 one_minute = 0;

        #500000 $finish;
    end

    initial begin
        $monitor("At time %t, reset = %b, one_minute = %b, load_new_c = %b,
current_time = %d%d:%d%d",
                 $time, reset, one_minute, load_new_c, current_time_ms_hr,
current_time_ls_hr, current_time_ms_min, current_time_ls_min);
    end

endmodule
```

3. tb_aclk_areg.v

```
module tb_aclk_areg;

    reg clk;
    reg reset;
    reg load_new_a;
    reg [3:0] new_alarm_ms_hr;
    reg [3:0] new_alarm_ls_hr;
    reg [3:0] new_alarm_ms_min;
    reg [3:0] new_alarm_ls_min;
    wire [3:0] alarm_time_ms_hr;
    wire [3:0] alarm_time_ls_hr;
    wire [3:0] alarm_time_ms_min;
    wire [3:0] alarm_time_ls_min;
```

```
aclk_areg uut (
    .clk(clk),
    .reset(reset),
    .load_new_a(load_new_a),
    .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),
    .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
    clk = 0;
    forever #5 clk = ~clk;
end

initial begin
    reset = 1;
    load_new_a = 0;
    new_alarm_ms_hr = 4'd0;
    new_alarm_ls_hr = 4'd0;
    new_alarm_ms_min = 4'd0;
    new_alarm_ls_min = 4'd0;
    #10 reset = 0;

    #10 load_new_a = 1;
    new_alarm_ms_hr = 4'd1;
    new_alarm_ls_hr = 4'd2;
    new_alarm_ms_min = 4'd3;
    new_alarm_ls_min = 4'd4;
```

```
        #10 load_new_a = 0;


        #50 load_new_a = 1;
        new_alarm_ms_hr = 4'd2;
        new_alarm_ls_hr = 4'd3;
        new_alarm_ms_min = 4'd4;
        new_alarm_ls_min = 4'd5;
        #10 load_new_a = 0;


        #50 $finish;
    end


    initial begin
        $monitor("At time %t, reset = %b, load_new_a = %b, alarm_time =
%d%d:%d%d",
                 $time, reset, load_new_a, alarm_time_ms_hr,
alarm_time_ls_hr, alarm_time_ms_min, alarm_time_ls_min);
    end


endmodule
```

4. tb_keyreg.v

```
module tb_aclk_keyreg;
    reg clock;
    reg shift;
    reg reset;
    reg [3:0] key;
    wire [3:0] key_ms_hr;
    wire [3:0] key_ls_hr;
    wire [3:0] key_ms_min;
    wire [3:0] key_ls_min;


    aclk_keyreg uut (
        .clock(clock),
        .shift(shift),
```

```
    .reset(reset),
    .key(key),
    .key_ms_hr(key_ms_hr),
    .key_ls_hr(key_ls_hr),
    .key_ms_min(key_ms_min),
    .key_ls_min(key_ls_min)
);

initial begin
    clock = 0;
    forever #5 clock = ~clock;
end

initial begin
    reset = 1;
    shift = 0;
    key = 4'b0000;

    #10 reset = 0;

    #10 shift = 1; key = 4'b0001;
    #10 shift = 0;
    #10 shift = 1; key = 4'b0010;
    #10 shift = 0;
    #10 shift = 1; key = 4'b0011;
    #10 shift = 0;
    #10 shift = 1; key = 4'b0100;
    #10 shift = 0;

    #10 reset = 1;
    #10 reset = 0;

    #10 shift = 1; key = 4'b0101;  // Shift in 5
    #10 shift = 0;
```

```
        #10 shift = 1; key = 4'b0110;  // Shift in 6
        #10 shift = 0;
        #10 shift = 1; key = 4'b0111;  // Shift in 7
        #10 shift = 0;
        #10 shift = 1; key = 4'b1000;  // Shift in 8
        #10 shift = 0;

        #20 $finish;
    end

    initial begin
        $monitor("At time %t, reset = %b, shift = %b, key = %b, key_ms_hr =
%b, key_ls_hr = %b, key_ms_min = %b, key_ls_min = %b",
                 $time, reset, shift, key, key_ms_hr, key_ls_hr,
key_ms_min, key_ls_min);
    end
endmodule
```

5. tb_aclk_controller.v

```
module tb_aclk_controller;
    reg clk;
    reg rst;
    reg one_second;
    reg alarm_button;
    reg time_button;
    reg [3:0] key;

    wire reset_count;
    wire load_new_c;
    wire show_new_time;
    wire show_a;
    wire load_new_a;
    wire shift;

    aclk_controller uut (
```

```
    .clk(clk),
    .rst(rst),
    .one_second(one_second),
    .alarm_button(alarm_button),
    .time_button(time_button),
    .key(key),
    .reset_count(reset_count),
    .load_new_c(load_new_c),
    .show_new_time(show_new_time),
    .show_a(show_a),
    .load_new_a(load_new_a),
    .shift(shift)
);

initial begin
    clk = 0;
    forever #5 clk = ~clk;
end

initial begin
    rst = 1;
    one_second = 0;
    alarm_button = 0;
    time_button = 0;
    key = 4'b1010;

    #10 rst = 0;

    #10 key = 4'b0000;
    #10 key = 4'b1010;

    #10 one_second = 1;
    #10 one_second = 0;
```

```
        #10 key = 4'b0001;
        #10 key = 4'b1010;
        #10 one_second = 1;
        #10 one_second = 0;

        #10 time_button = 1;
        #10 time_button = 0;

        #10 alarm_button = 1;
        #10 alarm_button = 0;

        #10 alarm_button = 1;
        #10 alarm_button = 0;

        #10 time_button = 1;
        #10 time_button = 0;

        #100 $finish;
    end

    initial begin
        $monitor("At time %t, rst = %b, one_second = %b, alarm_button = %b,
time_button = %b, key = %b, reset_count = %b, load_new_c = %b,
show_new_time = %b, show_a = %b, load_new_a = %b, shift = %b",
                 $time, rst, one_second, alarm_button, time_button, key,
reset_count, load_new_c, show_new_time, show_a, load_new_a, shift);
    end
endmodule
```

6. tb_aclk_lcd_display.v

```
module tb_aclk_lcd_display;

    reg [3:0] current_time_ms_hr;
    reg [3:0] current_time_ls_hr;
    reg [3:0] current_time_ms_min;
    reg [3:0] current_time_ls_min;
```

```
reg [3:0] alarm_time_ms_hr;
reg [3:0] alarm_time_ls_hr;
reg [3:0] alarm_time_ms_min;
reg [3:0] alarm_time_ls_min;
reg [3:0] key_ms_hr;
reg [3:0] key_ls_hr;
reg [3:0] key_ms_min;
reg [3:0] key_ls_min;
reg show_new_time;
reg show_a;
reg clk;

wire [7:0] display_ms_hr;
wire [7:0] display_ls_hr;
wire [7:0] display_ms_min;
wire [7:0] display_ls_min;
wire sound_alarm;

// Instantiate the aclk_lcd_display module
aclk_lcd_display uut (
    .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),
    .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),
    .key_ms_hr(key_ms_hr),
    .key_ls_hr(key_ls_hr),
    .key_ms_min(key_ms_min),
    .key_ls_min(key_ls_min),
    .show_new_time(show_new_time),
    .show_a(show_a),
```

```
    .sound_alarm(sound_alarm),
    .display_ms_hr(display_ms_hr),
    .display_ls_hr(display_ls_hr),
    .display_ms_min(display_ms_min),
    .display_ls_min(display_ls_min)
);

// Clock generation
initial begin
    clk = 0;
    forever #5 clk = ~clk;  // 10 time units period clock
end

// Test procedure
initial begin
    // Initialize signals
    show_new_time = 0;
    show_a = 0;
    current_time_ms_hr = 4'd0;
    current_time_ls_hr = 4'd0;
    current_time_ms_min = 4'd0;
    current_time_ls_min = 4'd0;
    alarm_time_ms_hr = 4'd0;
    alarm_time_ls_hr = 4'd0;
    alarm_time_ms_min = 4'd0;
    alarm_time_ls_min = 4'd0;
    key_ms_hr = 4'd0;
    key_ls_hr = 4'd0;
    key_ms_min = 4'd0;
    key_ls_min = 4'd0;

    // Simulate operation
    // Show current time
    #10;
```

```
        show_new_time = 1;
        show_a = 0;
        current_time_ms_hr = 4'd1;
        current_time_ls_hr = 4'd2;
        current_time_ms_min = 4'd3;
        current_time_ls_min = 4'd4;
        #100;

        // Show alarm time
        show_new_time = 0;
        show_a = 1;
        alarm_time_ms_hr = 4'd5;
        alarm_time_ls_hr = 4'd6;
        alarm_time_ms_min = 4'd7;
        alarm_time_ls_min = 4'd8;
        #100;

        // Finish simulation
        #100 $finish;
    end

    // Monitor outputs
    initial begin
        $monitor("At time %t: show_new_time = %b, show_a = %b, sound_alarm
= %b, display_ms_hr = %h, display_ls_hr = %h, display_ms_min = %h,
display_ls_min = %h",
                 $time, show_new_time, show_a, sound_alarm, display_ms_hr,
display_ls_hr, display_ms_min, display_ls_min);
    end

endmodule
```

7. tb_aclk_lcd_driver.v

```
module tb_aclk_lcd_driver;

    reg [3:0] alarm_time;
```

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

wire [7:0] display_time;
wire sound_alarm;

aclk_lcd_driver uut (
    .alarm_time(alarm_time),
    .current_time(current_time),
    .key(key),
    .show_alarm(show_alarm),
    .show_new_time(show_new_time),
    .display_time(display_time),
    .sound_alarm(sound_alarm)
);

initial begin
    show_new_time = 0;
    show_alarm = 0;
    current_time = 4'd0;
    alarm_time = 4'd0;
    key = 4'd0;

    #10;
    show_new_time = 1;
    show_alarm = 0;
    key = 4'd5;
    #100;

    show_new_time = 0;
    show_alarm = 1;
    alarm_time = 4'd9;
```

```
        #100;


        #10;
        current_time = 4'd9;
        alarm_time = 4'd9;
        #100;


        #100 $finish;
    end


    initial begin
        $monitor("At time %t: show_new_time = %b, show_alarm = %b,
sound_alarm = %b, display_time = %h",
                 $time, show_new_time, show_alarm, sound_alarm,
display_time);
    end


endmodule
```

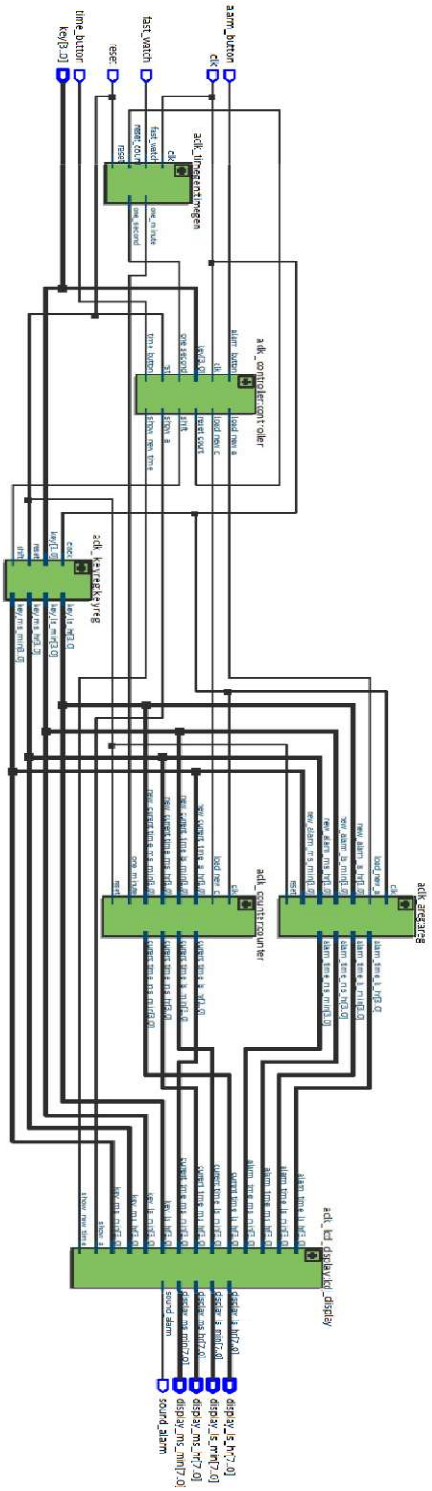
8. tb_alarm_clk_rtl.v

```
module tb_alarm_clk_rtl;


    reg clk;
    reg reset;
    reg alarm_button;
    reg time_button;
    reg [3:0] key;
    reg fast_watch;


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

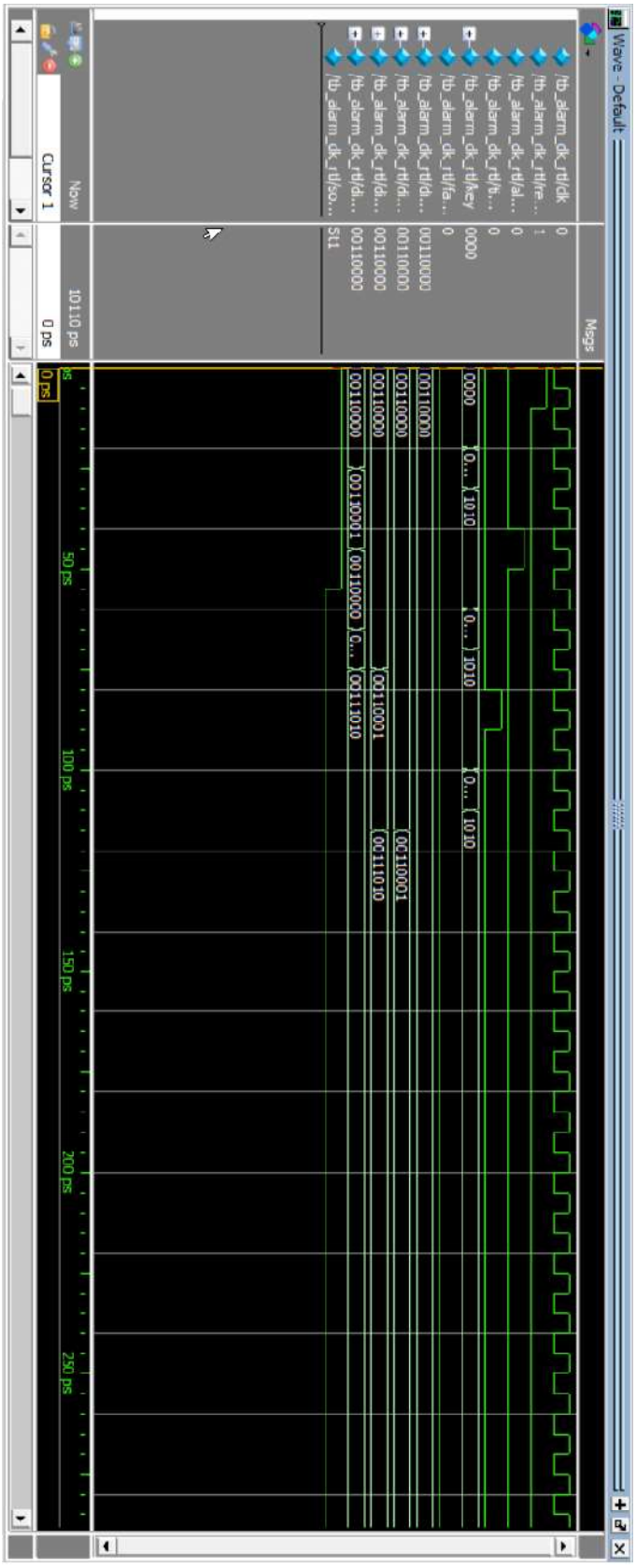
```
// Instantiate the alarm_clk_rtl module
alarm_clk_rtl uut (
    .clk(clk),
    .reset(reset),
    .alarm_button(alarm_button),
    .time_button(time_button),
    .key(key),
    .fast_watch(fast_watch),
    .display_ms_hr(display_ms_hr),
    .display_ls_hr(display_ls_hr),
    .display_ms_min(display_ms_min),
    .display_ls_min(display_ls_min),
    .sound_alarm(sound_alarm)
);

// Clock generation
initial begin
    clk = 0;
    forever #5 clk = ~clk;  // 10 time units period clock
end

// Test procedure
initial begin
    // Initialize signals
    reset = 1;
    alarm_button = 0;
    time_button = 0;
    fast_watch = 0;
    key = 4'b0000;  // NOKEY value (0)

    // Release reset
    #10 reset = 0;

    // Simulate normal operation
```

```
        // Example sequence:
        #10 key = 4'b0001;  // Press key 1
        #10 key = 4'b1010;  // Release key

        // Set alarm time example
        #10 alarm_button = 1;
        #10 alarm_button = 0;
        #10 key = 4'b0001;  // Press key 1
        #10 key = 4'b1010;  // Release key

        // Set current time example
        #10 time_button = 1;
        #10 time_button = 0;
        #10 key = 4'b0010;  // Press key 2
        #10 key = 4'b1010;  // Release key

        // Finish simulation
        #10000 $finish;
    end

    // Monitor outputs
    initial begin
        $monitor("At time %t: reset = %b, alarm_button = %b, time_button =
%b, key = %b, fast_watch = %b, sound_alarm = %b, display_ms_hr = %h,
display_ls_hr = %h, display_ms_min = %h, display_ls_min = %h",
                 $time, reset, alarm_button, time_button, key, fast_watch,
sound_alarm, display_ms_hr, display_ls_hr, display_ms_min, display_ls_min);
    end

endmodule
```

SYNTHESISED RTL



OUTPUT WAVEFORM



CONCLUSION

The digital alarm clock project successfully demonstrates the design and implementation of a functional timekeeping device using Verilog HDL. The system's modular components effectively manage time generation, user input, control logic, and display functions. The `aclk_timegen` module ensures accurate time pulses, while the `aclk_counter` and `` `aclk_areg` `` modules handle time settings efficiently. The `aclk_keyreg` processes user inputs, and the `aclk_controller` ensures smooth operation. The `aclk_lcd_display` provides clear time display and accurate alarm indication. Finally, the `alarm_clk_rtl` acts as the top level module and brings all the sub modules together.

Thorough testing validated the design, confirming reliable and correct operation. This project showcases the effectiveness of hardware description languages in designing complex digital systems and demonstrates the benefits of a modular approach for ease of design, debugging, and future enhancements. Potential improvements could include additional features like multiple alarms, snooze functionality, and external interface integration.

Overall, this project highlights the practicality and power of Verilog HDL in real-world applications, offering valuable insights into digital system design.