ALARM CLOCK

Introduction:

Modelled in Verilog, alarm clock synthesised is functional to be used as daily use alarm clock. User can set an alarm in this clock.

Explanation:

Alarm clock synthesised, takes multiple inputs such as input time, clock, load_time, load_alarm set_alarm, stop_alarm and an active low reset button. In output it shows current time, and alarm rings if alarm is set.

- Input time – user can input time in hh:mm format, this input time will be used to load time and load alarm to the module.
- Clock – we are using a clock of frequency 100Hz (10ms)
- load_time – when load_time is high, clock time is set to input time, ss being 00
- load_alarm – when load_alarm is high, alarm time is set to input time, ss being 00
- set_alarm – to set the alarm, when high alarm is set i.e., when current time reaches alarm time, alarm rings.
- stop_alarm – a button for user to stop the ringing alarm, active high.
- reset – resets the clock time to input time, alarm time to 00:00, and stops the alarm if ringing

It outputs the time in hh:mm:ss format, a 7 seg BCD can be used to display time, and a ringing equipment can be installed at alarm output.

Alarm Operation:

This clock is designed such that, if the set_alarm is high and current time reaches alarm time, alarm rings. Now, user can stop the alarm using stop_alarm button, but if user is not active and stop_alarm is not pressed, alarm will ring for one minute then it will stop automatically, and again ring after nine minutes. This process repeats three times, if user is inactive and after that it will stop ringing. User can get out of this process as soon as he pressed stop_alarm button.

Code Structure:

- Input and Output ports

```
module alarm_clock (
    rst_bar, clk, h1_in, h0_in, m1_in, m0_in, load_time, load_alarm, set_alarm, stop_alarm,
    alarm, h1_out, h0_out, m1_out, m0_out, s1_out, s0_out
);

    input   rst_bar,    // active low reset button, set alarm time to 00:00:00, and clock time to input time,
            clk,        // a clock of frequency 100Hz (10ms)
            load_time,  // when high, sets clock time to input time
            load_alarm, // when high, sets alaram time to input time
            set_alarm,  // when high, alarm rings when clock time reaches alarm time
            stop_alarm; // when high, stops the ringing alarm

    input [1:0] h1_in;  // inputs the MSB of hrs,  (0-2)
    input [3:0] h0_in;  // inofputs the LSB of hrs,  (0-9)
    input [2:0] m1_in;  // inputs the MSB  mins, (0-5)
    input [3:0] m0_in;  // inputs the LSB of mins, (0-9)

    output reg alarm;         // when high, alarm rings

    output reg [1:0] h1_out;  // outputs the MSB of hrs,  (0-2)
    output reg [3:0] h0_out;  // outputs the LSB of hrs,  (0-9)
    output reg [2:0] m1_out;  // outputs the MSB of mins, (0-5)
    output reg [3:0] m0_out;  // outputs the LSB of mins, (0-9)
    output reg [2:0] s1_out;  // outputs the MSB of secs, (0-5)
    output reg [3:0] s0_out;  // outputs the LSB of secs, (0-9)
```

- Registers to store alarm time and clock time

```
//  we need temperory clock time and alarm time for operations to be followed

reg [1:0] temp_c_h1, temp_a_h1;     //  MSB of hrs of clock and alarm    (0-2)
reg [3:0] temp_c_h0, temp_a_h0;     //  LSB of hrs of clock and alarm    (0-9)
reg [2:0] temp_c_m1, temp_a_m1;     //  MSB of mins of clock and alarm   (0-5)
reg [3:0] temp_c_m0, temp_a_m0;     //  LSB of mins of clock and alarm   (0-9)
reg [2:0] temp_c_s1, temp_a_s1;     //  MSB of secs of clock             (0-5)
reg [3:0] temp_c_s0, temp_a_s0;     //  LSB of secs of clock             (0-9)
```

- we have clock of 100 Hz which could be converted into clock of 1Hz, for ease in operations later

```
//  we have a clock of frequency 100 Hz but for alarm clock we need a clock of frequency 1 Hz

reg clk_1;  //  this will work as clock of frequency 1Hz
integer i;  //  integer value to be iterated

always @(posedge clk or negedge rst_bar) begin

    if (!rst_bar) begin         //  active low reset button
        i <= 0;                 //  both i and clk_1 to be
        clk_1 <= 1'b0;          //  set to 0, at reset
    end
    else begin
        if (i>=49) begin        //  since we have a clock of frequency 100 Hz
            i <= 0;             //  halfway is at 49, where out edge must come
            clk_1 <= ~clk_1;    //  negedge or posedge
        end
        else i<= i+1;           //  otherwise i will be incremented, until reaches halfway
    end
end
```

- clock operation
  1) reset button

```
if (!rst_bar) begin

    temp_c_h1 <= h1_in;     //  clock time is set to input time
    temp_c_h0 <= h0_in;
    temp_c_m1 <= m1_in;
    temp_c_m0 <= m0_in;
    temp_c_s1 <= 3'b0;      //  secs set to 00
    temp_c_s0 <= 4'b0;

    temp_a_h1 <= 2'b0;      //  alarm time set to 00:00 (hh:mm)
    temp_a_h0 <= 4'b0;
    temp_a_m1 <= 3'b0;
    temp_a_m0 <= 4'b0;
    temp_a_s1 <= 3'b0;
    temp_a_s0 <= 4'b0;

    alarm <= 1'b0;          //  alarms if ringings sounds off
end
 if (!rst_bar) begin          //  alarms turns off at reset
     alarm <=  1'b0;
     min_1 = 1'b0;
     min_9 = 1'b0;
     loop_3 = 1'b0;
     num_1 = 8'b0;
     num_9 = 10'b0;
     num_3 = 2'b0;
 end
```

  2) load time

```
else if (load_time) begin
    temp_c_h1 <= h1_in;     //  clock time is set to input time
    temp_c_h0 <= h0_in;
    temp_c_m1 <= m1_in;
    temp_c_m0 <= m0_in;
    temp_c_s1 <= 3'b0;      //  secs set to 00
    temp_c_s0 <= 4'b0;
end
```

  3) load alarm

```
else if (load_alarm) begin
    temp_a_h1 <= h1_in;     //  clock time is set to input time
    temp_a_h0 <= h0_in;
    temp_a_m1 <= m1_in;
    temp_a_m0 <= m0_in;
    temp_a_s1 <= 3'b0;      //  secs set to 00
    temp_a_s0 <= 4'b0;
end
```

4) counting with time

```
else begin
    if (temp_c_s0 >= 9) begin                                   //  is s0 = 9, then s0 = 0
        temp_c_s0 <= 0;                                         //  and s1 ges incremented
        if (temp_c_s1 >= 5) begin                               //  ss == 59, it should get to 00
            temp_c_s1 <= 0;                                     //  and m0 gets incremented
            if (temp_c_m0 >= 9) begin                           //  if m0 = 9, then m0 = 0
                temp_c_m0 <= 0;                                 //  and m1 gets incremented
                if (temp_c_m1 >= 5) begin                       //  mm == 59, it should get to 00
                    temp_c_m1 <= 0;                             //  and hh gets incremented
                    if (temp_c_h1 < 2) begin                    //  if 19 gets to 20
                        if (temp_c_h0 >= 9) begin               //  if 09 gets to 10
                            temp_c_h0 <= 0;                     //  but shouldn't reach till 29
                            temp_c_h1 <= temp_c_h1 + 1;
                        end
                        else temp_c_h0 <= temp_c_h0 + 1;
                    end
                    else if (temp_c_h1 >=2) begin               //  so this is seprated
                        if (temp_c_h0 >= 3) begin               //  after hh = 23, it gets to 00
                            temp_c_h0 <= 0;
                            temp_c_h1 <= 0;
                        end
                        else temp_c_h0 <= temp_c_h0 + 1;
                    end
                end
                else temp_c_m1 <= temp_c_m1 + 1;
            end
            else temp_c_m0 <= temp_c_m0 + 1;
        end
        else temp_c_s1 <= temp_c_s1 + 1;
    end
    else temp_c_s0 <= temp_c_s0 + 1;
```

- alarm operation

```
always @(negedge clk_1 or negedge rst_bar) begin

    if (!rst_bar) begin         //  alarms turns off at reset
        alarm <=  1'b0;
        min_1 <=  1'b0;
        min_9 <=  1'b0;
        loop_3 <= 1'b0;
    end

    else if (set_alarm) begin   //  when set_alarm is high and clock time matches alarm time
        if ({temp_c_h1,temp_c_h0,temp_c_m1,temp_c_m0,temp_c_s1,temp_c_s0} ==
            {temp_a_h1,temp_a_h0,temp_a_m1,temp_a_m0,temp_a_s1,temp_a_s0} ) begin
        alarm <=  1'b1;         //  alarm rings
        min_1 <=  1'b1;
        min_9 <=  1'b0;
        loop_3 <= 1'b1;
        end
    end
end
```

Testbench

- Input and Output ports

```
`timescale 1ms/1ms
`include "alarm_clock.v"

module tb_alarm_clock;

    reg clk, rst_bar, load_time, load_alarm, set_alarm, stop_alarm;     //  inputs are declared reg datatype

    reg [1:0] h1_in;  // MSB of hrs,  (0-2)
    reg [3:0] h0_in;  // LSB of hrs,  (0-9)
    reg [2:0] m1_in;  // MSB of mins, (0-5)
    reg [3:0] m0_in;  // LSB of mins, (0-9)

    wire alarm;         // outputs are declared wire datatype

    wire [1:0] h1_out;  // MSB of hrs,  (0-2)
    wire [3:0] h0_out;  // LSB of hrs,  (0-9)
    wire [2:0] m1_out;  // MSB of mins, (0-5)
    wire [3:0] m0_out;  // LSB of mins, (0-9)
    wire [2:0] s1_out;  // MSB of secs, (0-5)
    wire [3:0] s0_out;  // LSB of secs, (0-9)
```

Setting the timescale with units and precision both 1ms, and including the file containing design under test. Input ports are declared reg type and output ports as wire type.

- Module instantiation & clock

```
alarm_clock DUT
(
rst_bar, clk, h1_in, h0_in, m1_in, m0_in, load_time, load_alarm, set_alarm, stop_alarm,
alarm, h1_out, h0_out, m1_out, m0_out, s1_out, s0_out
);

localparam CLK_PERIOD = 10;             //  clock with timeperiod 10ms (100 Hz)
always #(CLK_PERIOD/2) clk=~clk;
```

Module is instantiated with name ‘DUT’, input and output parameters are fed. We are using clock of 50Hz, hence declared using localparam

- Initialising the input values and dumping

```
initial begin
    $dumpfile("tb_alarm_clock.vcd");                                    //  file to be dumped inn, for gtkwave
    $dumpvars(1, alarm, h1_out, h0_out, m1_out, m0_out, s1_out, s0_out);        //  variables to be dumped
    $monitor($time," rst_bar = %b set_alarm = %b stop_alarm = %b input time = %d%d : %d%d output time = %d%d : %d%d alarm =
end

initial begin                           //  initialising at t = 0
    clk = 1'b0;
    rst_bar = 1'b0;

    h1_in = 2'd1;   h0_in = 4'd1;       //  initialised input time 11:43 (hh:mm)
    m1_in = 3'd4;   m0_in = 4'd3;

    set_alarm = 1'b0;   stop_alarm = 1'b0;
end
```

- Active low reset button

```
initial begin   //  handles reset button (active low)

    #1e3    rst_bar = 1'b1;     //  t = 1sec
    #32e6   rst_bar = 1'b0;     //  t = 32001 sec
    #1e3    rst_bar = 1'b1;     //  t = 32002 sec
    #274e4  rst_bar = 1'b0;     //  t = 34742 sec
    #1e3    rst_bar = 1'b1;     //  t = 34743 sec

end
```

- Input time

```
initial begin   //  handles input time

    #1e4    h1_in = 2'd0;   h0_in = 4'd7;       //  input time 07:10 (hh:mm) at t = 10 sec
            m1_in = 3'd1;   m0_in = 4'd0;

    #4e4    h1_in = 2'd0;   h0_in = 4'd9;       //  input time 09:05 (hh:mm) at t = 50 sec
            m1_in = 3'd0;   m0_in = 4'd5;

    #9e6    h1_in = 2'd1;   h0_in = 4'd2;       //  input time 12:10 (hh:mm) at t = 9050 sec
            m1_in = 3'd1;   m0_in = 4'd0;

    #1e7    h1_in = 2'd1;   h0_in = 4'd3;       //  input time 13:40 (hh:mm) at t = 19050 sec
            m1_in = 3'd4;   m0_in = 4'd0;

    #6e6    h1_in = 2'd1;   h0_in = 4'd5;       //  input time 15:30 (hh:mm) at t = 25050 sec
            m1_in = 3'd3;   m0_in = 4'd0;

    #6e6    h1_in = 2'd2;   h0_in = 4'd3;       //  input time 23:15 (hh:mm) at t = 31050 sec
            m1_in = 3'd1;   m0_in = 4'd5;

end
```

- load time and load alarm

```
initial begin                   //  handles load_time button (active high)

    #3e4    load_time = 1'b1;   //  t = 30 sec
    #1e3    load_time = 1'b0;   //  t = 31 sec

end

initial begin                   //  handles load_alarm button (active high)

    #7e4    load_alarm = 1'b1;  //  t = 70 sec
    #1e3    load_alarm = 1'b0;  //  t = 71 sec
    #9e6    load_alarm = 1'b1;  //  t = 9071 sec
    #1e3    load_alarm = 1'b0;  //  t = 9072 sec
    #1e7    load_alarm = 1'b1;  //  t = 19072 sec
    #1e3    load_alarm = 1'b0;  //  t = 19073 sec
    #6e6    load_alarm = 1'b1;  //  t = 25073 sec
    #1e3    load_alarm = 1'b0;  //  t = 25074 sec

end
```

- set alarm and stop alarm

```
initial begin                   //  handles set_alarm button (active high)

    #1e7    set_alarm = 1'b1;   // t = 10000 sec

end

initial begin                   //  handles stop_alarm button (active high)

    #24e6   stop_alarm = 1'b1;  //  t = 24000 sec
    #1e3    stop_alarm = 1'b0;  //  t = 24001 sec
    #6059e3 stop_alarm = 1'b1;  //  t = 30060 sec
    #1e3    stop_alarm = 1'b0;  //  t = 30061 sec

end
```

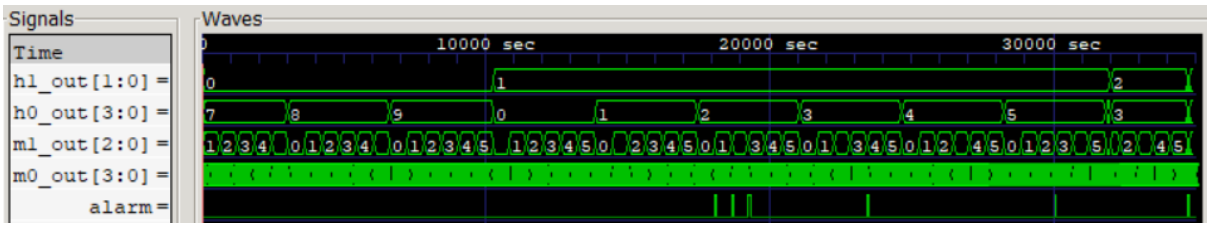
Synthesising and Output

Synthesised using command – iverilog -o <destination_file.vvp> <source_file.v>

After executing this command, a .vvp file is synthesised along with a .vcd file with file name as in $dumpfile()

Then accessed using command – vvp <destination_file.vvp>

To view the output, use gtkwave and open .vcd file in it



All the outputs, exactly matched with the expected results.

Conclusion

The Verilog code for an alarm clock involves defining the various components and their connections within the circuit, specifying their functionalities and behaviours. This includes defining clock signals, input/output signals, and the logic for controlling the display, alarm settings, triggering the alarm sound and snooze feature.