HO CHI MINH UNIVESITY OF TECHNOLOGY
COMPUTER SCIENCE - ENGINEERING

MINI PROJECT REPORT
DIGITAL DESIGN WITH THE VERILOG HDL

ALARM CLOCK

Designed by:
Tran Long Vi
1814804

HO CHI MINH CITY, DECEMBER 2019

CONTENT

I. INTRODUCTION

1. Alarm clock

Alarm clock is a popular device in the world. This device helps notify people about an important time.

The design of this project is a digital alarm clock that displays the hours, minutes and seconds. This design embodies all the basic features which would expect on a standard alarm clock.

2. FPGA

Field Programmable Gate Arrays are a programmable digital logic chip, you can use them to program most of functions of any digital design. There are many documents about FPGA on website but here I want you to pay attention on its name. I saw on many websites people translate FIELD as a field. But that is wrong in this case. FIELD means where the chip is used. Field Programmable means that some chips can be programmed at a user's site instead of being programmed at the manufacturing site. The FPGA is built from an array (matrix or array) of programmable elements that should be called Programmable Gate Array.

The basic architecture of FPGA consists of three main components: reconfigurable logic blocks, Configurable Logic Blocks (CLBs) performing logic functions; Internal connections, Porgrammable Interconnect can be programmed to connect club inputs and outputs and internal I / O blocks, I / O blocks provide communication between peripherals and internal signals.

Applications of FPGA include: DSP digital signal processing, aerospace, defense systems, ASIC prototyping, visual control systems, image recognition analysis, speech recognition, cryptography, computer hardware model, etc. Due to the high flexibility in the design process, FPGA can solve a complex class of problems that were previously only done by computer software. The high logic gate density of FPGAs is applied to problems requiring high computing volumes and used in real-time working systems.

3. Altera DE2i Board

DE2i-150 Board is a circuit board for research and development in the fields of arithmetic logic (digital logic), computer organization and FPGA.

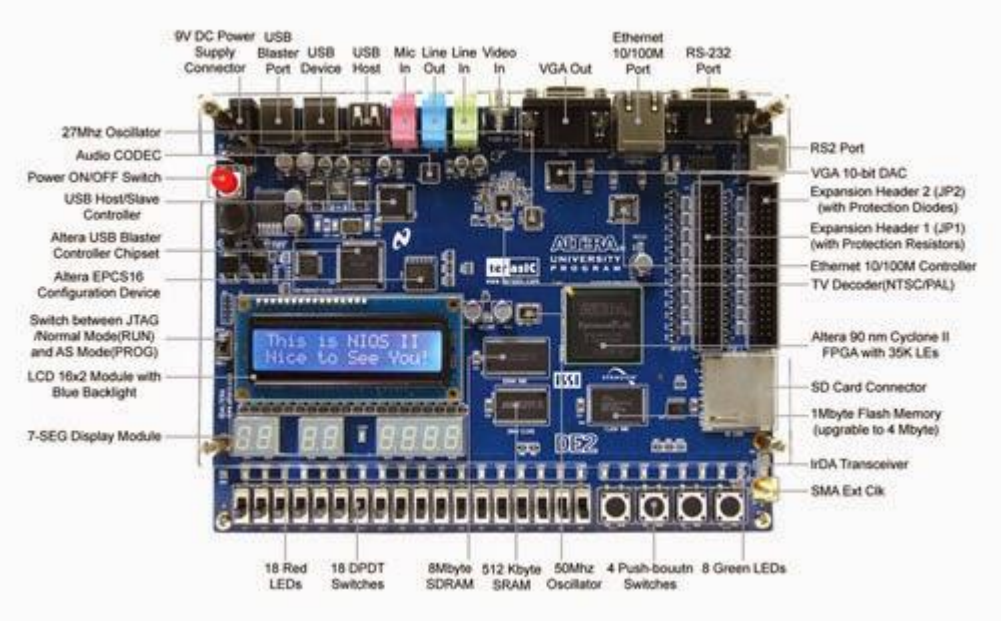

Figure 1: Construction of Altera DE2i Board[1]

Components of the Altera DE2i board include:

- FPGA:
  - FPGA Altera Cyclone IV EP4CGX150DF31 IC.
  - Altera Serial Configuration – EPCS16 IC.
- Import and export devices:
  - USB Blaster for programming and controlling API of users; Support both JTAG and AS programming modes.
  - 10/100/1000 Mbps Ethernet Port Controller.
  - HSMC (High Speed Mezzanine Card) Port.
  - VGA-out Port.
  - TV Decoder and TV-in Connector.
  - USB Host/Slave Controller with USB type A and type B Port.
  - PS / 2 mouse / keyboard Connector.

- Quality optical disc 24-bit decoder / encoder with line-in, line-out, and microphone jacks.
- 2 40-pin extended header with diode protection. - RS-232 communication port and 9-pin Ponnector.
- Infrared communication Port.

- Memory:
  - 128 MB SDRAM.
  - 4 MB SSRAM.
  - 64 MB Flash.
- Switch, LED, LCD, Clock Pulse:
  - 4 Button, 18 Switch.
  - 18 LEDR, 9 LEDG, 8 7-segments LED.
  - LCD 16x2.
  - Clock 50-MHz.

4. Quartus II Software:

Quartus II is a software development tool from Altera, providing a comprehensive design environment for SOPC designs (system on a programmable chip).

This is a full integrated packaging software for logic design with Altera's PLD programmable logic components, including APEX, Cyclone, FLEX, MAX, Stratix ... Quartus provides design capabilities. the following logic:

- Design environment includes drawings, block diagrams, drafting tools for languages: AHDL, VHDL, and Verilog HDL.
- LogicLock design.
- A powerful tool for logical synthesis.
- Ability to simulate functions and time.
- Time analysis.
- Analysis of embedded logic with the analysis tool SignalTap @ II.
- Allow export, create and join source files to create program files.
- Automatic positioning error.
- The ability to program and identify components.
- Quartus II software uses the NativeLink @ integration kit with design tools that provide seamless communication between Quartus and other EDA hardware design tools.
  - Quartus II can also read standard EDIF, VHDL and Verilog HDL circuit files as well as create these netlist files.

- Quartus II has a graphic design environment that helps designers easily write code, compile, proofread, simulate ...

This report will build an Alarm clock System using Verilog HDL code and implement it with Quartus II software and Altera DE2i-150 Board.

II. DESIGN DESCRIPTION

1. Ideas for operating the alarm clock system

- The clock displays the hour, minute, second of real time in a 24-hour type.
- Clock can change the value of real time.
- Up to 5 alarm values can be saved, each with an active notification.
- There is a button to display the alarm time, the alarms will have different display priority (The second alarm is only displayed if alarm 1 is off).
- The alarm will turn off automatically after 60 seconds or can be manually turned off.

2. Design ideas

- Using the 7 segment LED to display the time:
  - HEX 7 – 6 display hour.
  - HEX 5 – 4 display minute.
  - HEX 3 – 2 display second.
- Using 5 LEDR [4: 0] flashes for alarm, LEDR [0] - LEDR [4] for alarm 1 - 5, LEDR will automatically turn off after 60 seconds or when the corresponding alarm is turned off.
- Using the switch and button to control the alarm clock:
  - SW [6: 0] to enter the time value (hour, minute or second).
  - SW [11: 7] to turn on and off the alarms, the priority level will be from SW [7] to SW [11].
  - SW [16] is used to change the value of the real time, when activated it will change the value according to SW [6: 0] and KEY [2: 0].
  - SW [17] is a highly active reset button, when activated it will set everything to 0.
  - KEY [2: 0] is used to save the value of the time entered by SW [6: 0], corresponding to the hour - minute - second value.
  - KEY [3] is used to display the alarm time, the priority displayed is the time of the alarm 1 - 5.

III. DESIGN BY VERILOG HDL

1. Block diagram

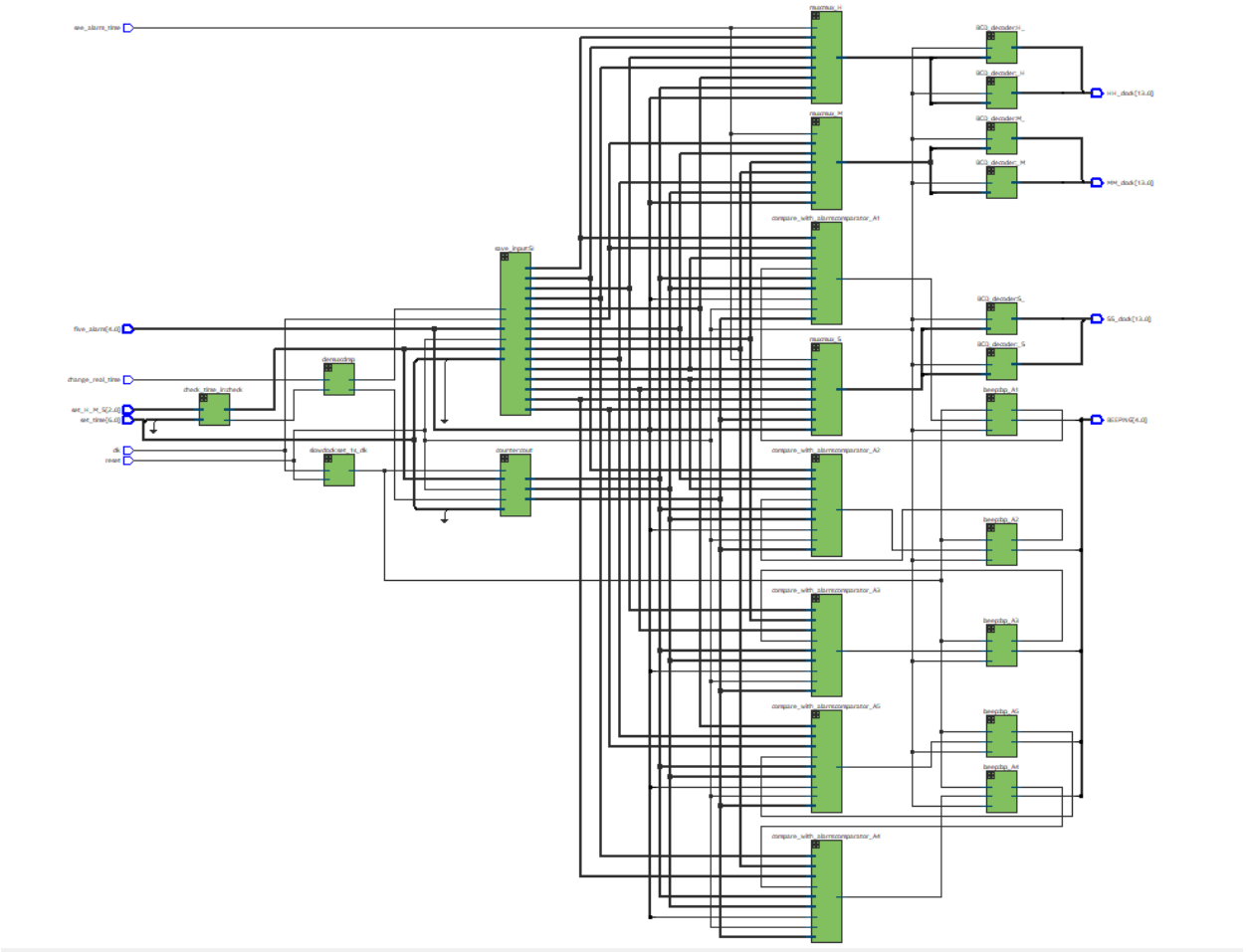

Figure 2: Netlist Viewer images in Quartus

The structure of the design will include the following blocks:

- Frequency division block
- Block for checking input data
- Block stores the value of the alarm
- Block real time counting
- Block comparing alarm values with real time
- Notification block
- BCD decoding block

2. Describing the blocks in the design

2.1. Frequency division block

The frequency division block - *slowclock* has an input of 50 MHz clock frequency and the reset signal, the output is a clock frequency of 1 Hz.

```
1   //sua tan so cua clk ve 1Hz
2   module slowclock(clk, reset, clk_1);
3        input clk;
4        input reset;
5        output clk_1;
6        reg clk_1s;
7        reg [31:0] counter;
8
9        always @(posedge clk, posedge reset) begin
10           if(reset) begin
11                counter <= 0;
12                clk_1s <= 1'b0;
13           end
14           else begin
15             if(counter == 24999999) begin
16                clk_1s <= ~clk_1s;
17                counter <= 0;
18             end
19             else
20                counter <= counter + 1;
21           end
22       end
23       assign clk_1 = clk_1s;
24       //assign clk_1 = clk; //dung de mo phong modelsim
25  endmodule
26
```

Figure 3: *Slowclock* block Verilog code

2.2. Block for checking input data

The block for checking input data - *check_time_in* has input as the time entered and the button selects to set hours - minutes - seconds. This block has the function of checking the input time, if the input is hour, it will check to see whether the number is less than 24, and if it is minutes or seconds, it will check whether the number is less than 60.

When the input is valid, the block will transmit the output to enable the block to store the alarm value or to change the time in the real-time timer block.

```
1   //check input
2   module check_time_in(output check_out,
3                        output [2:0]H_M_S,
4                        input [7:0]time_in,
5                        input [2:0]is_hour);
6       reg [7:0]r_check;
7       reg [2:0]r_H_M_S;
8
9       always @(is_hour, time_in) begin
10              if(is_hour == 3'b011) begin
11                  if(time_in  <= 8'h23) begin
12                      r_check = 1;
13                      r_H_M_S = 3'b011;
14                  end
15                  else
16                      r_check = 0;
17              end
18              else if(is_hour != 3'b111) begin
19                  if(time_in <= 8'h59) begin
20                      r_check = 1;
21                      if(is_hour == 3'b110)
22                         r_H_M_S = 3'b110;
23                      else
24                         r_H_M_S = 3'b101;
25                  end
26                  else
27                      r_check  = 0;
28              end
29              else
30                  r_check = 0;
31      end
32      assign H_M_S = r_H_M_S;
33      assign check_out = r_check;
34  endmodule
35
```

Figure 4: *check_time_in* block Verilog code

The output from the *check_time_in* block will go through the demultiplexer port to choose whether the data will be put into the alarm store or the real-time block.

```
152
153 module demux(output out_R,
154                 output out_A,
155                 input enable,
156                 input in);
157     reg r_out_R;
158     reg r_out_A;
159     always @(in, enable) begin
160         if(enable == 1) begin
161             r_out_R <= in;
162             r_out_A <= 0;
163         end
164         else begin
165             r_out_A <= in;
166             r_out_R <= 0;
167         end
168     end
169     assign out_A = r_out_A;
170     assign out_R = r_out_R;
171 endmodule
172
```

Figure 5: *demux* block Verilog code

2.3. Block storing the value of the alarm

The block storing the value of the alarm - *save_input* has the input clock pulse from the slowclock block, the alarm control button, input data, reset and output from the *check_time_in* block.

```
module save_input(output [7:0]A_hour0, output [7:0]A_hour1, output [7:0]A_hour2,
                  output [7:0]A_min0, output [7:0]A_min1, output [7:0]A_min2, output
                  output [7:0]A_sec0, output [7:0]A_sec1, output [7:0]A_sec2, output
                  input [2:0]set_H_M_S,
                  input [7:0]set_time,
                  input [4:0]five_alarm,
                  input check_in,
                  input reset,
                  input clock);
    reg [7:0]r_A_hour[4:0];
    reg [7:0]r_A_min[4:0];
    reg [7:0]r_A_sec[4:0];

    always @(posedge clock, posedge reset) begin
      if(reset) begin
          r_A_sec[0] <= 8'h0;
          r_A_sec[1] <= 8'h0;
          r_A_sec[2] <= 8'h0;
          r_A_sec[3] <= 8'h0;
          r_A_sec[4] <= 8'h0;
          r_A_min[0] <= 8'h0;
          r_A_min[1] <= 8'h0;
          r_A_min[2] <= 8'h0;
          r_A_min[3] <= 8'h0;
          r_A_min[4] <= 8'h0;
          r_A_hour[0] <= 8'h0;
          r_A_hour[1] <= 8'h0;
          r_A_hour[2] <= 8'h0;
          r_A_hour[3] <= 8'h0;
          r_A_hour[4] <= 8'h0;
      end
      else if(check_in) begin
          if(set_H_M_S == 3'b110) begin
            casex(five_alarm)
                5'b10000: r_A_sec[4] = set_time;
                5'bx1000: r_A_sec[3] = set_time;
                5'bxx100: r_A_sec[2] = set_time;
                5'bxxx10: r_A_sec[1] = set_time;
                5'bxxxx1: r_A_sec[0] = set_time;
            endcase
          end
          else if(set_H_M_S == 3'b101) begin
             casex(five_alarm)
                5'b1xxxx: r_A_min[4] = set_time;
                5'bx1xxx: r_A_min[3] = set_time;
                5'bxx1xx: r_A_min[2] = set_time;
                5'bxxx1x: r_A_min[1] = set_time;
                5'bxxxx1: r_A_min[0] = set_time;
            endcase
          end
          else if(set_H_M_S == 3'b011) begin
            casex(five_alarm)
                5'b1xxxx: r_A_hour[4] = set_time;
                5'bx1xxx: r_A_hour[3] = set_time;
                5'bxx1xx: r_A_hour[2] = set_time;
                5'bxxx1x: r_A_hour[1] = set_time;
                5'bxxxx1: r_A_hour[0] = set_time;
            endcase
          end
        end
      end
      assign A_hour0 = r_A_hour[0];
      assign A_hour1 = r_A_hour[1];
      assign A_hour2 = r_A_hour[2];
      assign A_hour3 = r_A_hour[3];
      assign A_hour4 = r_A_hour[4];
      assign A_min0 = r_A_min[0];
      assign A_min1 = r_A_min[1];
      assign A_min2 = r_A_min[2];
      assign A_min3 = r_A_min[3];
      assign A_min4 = r_A_min[4];
      assign A_sec0 = r_A_sec[0];
      assign A_sec1 = r_A_sec[1];
      assign A_sec2 = r_A_sec[2];
      assign A_sec3 = r_A_sec[3];
      assign A_sec4 = r_A_sec[4];

  endmodule
```

Figure 6: *save_input* block Verilog code

This block is used to store alarm values from input data. The data is saved when the following conditions are met:

- Any alarm is enabled
- Pulse clock click edge up
- Reset = 0
- The input satisfies the conditions in the *check_time_in* block
- Input is not used to change real time

2.4. Real time counting block

The real time counting block - *counter* has following inputs: clock pulse, reset signal, output from *check_time_in* block, input data and trigger button to change the real time value.

This unit is used to automatically change the value of seconds according to a clock frequency of 1Hz, the value of minutes will change when the second counts to 59, the value of the hour will change when the minute counts to 59. If the values hour - minute - second counts to the limit value (23 for hours, 59 for minutes and seconds), the value will be changed to 0.

```
1   //counter real time
2   module counter(output [7:0] hour,
3                  output [7:0] min,
4                  output [7:0]sec,
5                  input clk,
6                  input reset,
7                  input [7:0]set_time,
8                  input [2:0]enable,
9                  input save);
10
11
12      reg [7:0]r_hour;
13      reg [7:0]r_min;
14      reg [7:0]r_sec;
15
16      always @(posedge clk, posedge reset) begin
17          if(reset) begin
18             r_hour <= 7'h0;
19             r_min <= 7'h0;
20             r_sec <= 7'h0;
21          end
```

Figure 7: *counter* block Verilog code

```
        else begin
            if(save) begin
                case(enable)
                    3'b011: r_hour <= set_time;
                    3'b101: r_min <= set_time;
                    3'b110: r_sec <= set_time;
                endcase
            end
            else begin
                if(r_sec == 7'h59) begin
                    if(r_min == 7'h59) begin

                        if(r_hour == 7'h23)
                            r_hour <= 7'h0;
                        else begin
                            if(r_hour[3:0] == 4'd9) begin
                                r_hour[3:0] <= 4'd0;
                                r_hour[7:4] <= r_hour[7:4] + 4'd1;
                            end
                            else
                                r_hour[3:0] <= r_hour[3:0] + 4'd1;
                        end
                        r_min <= 7'h0;
                    end

                    else begin
                        if(r_min[3:0] == 4'd9) begin
                            r_min[7:4] <= r_min[7:4] + 4'd1;
                            r_min[3:0] <= 4'd0;
                        end
                        else
                            r_min[3:0] <= r_min[3:0] + 4'd1;
                    end
                    r_sec <= 7'h0;
                end

                else begin
                    if(r_sec[3:0] == 4'd9) begin
                        r_sec[7:4] <= r_sec[7:4] + 4'd1;
                        r_sec[3:0] <= 4'd0;
                    end
                    else
                        r_sec[3:0] <= r_sec[3:0] + 4'd1;
                end
            end
        end
    end
    assign hour = r_hour;
    assign min = r_min;
    assign sec = r_sec;
endmodule
```

Figure 8: *counter* block Verilog code

2.5. Comparing alarm values with real time block

Comparing alarm values with real time block - *compare_with_alarm* has following inputs: *save_input* block output, reset signal, notification block return message and alarm control button.

This block is used to compare the value of an alarm with real time. When the two values are equal, the block will transmit a signal that will trigger the notification block. The signal will be cut off if the corresponding alarm is turned off, when the reset button is activated or when the return value from the notification block is received.

```
//so sanh voi alarm-time-set
module compare_with_alarm(output on,
                          input [7:0]a_hour,
                          input [7:0]hour,
                          input [7:0]a_min,
                          input [7:0]min,
                          input [7:0]a_sec,
                          input [7:0]sec,
                          input on_or_off,
                          input rst,
                          input auto_rst);
    reg r_on;
    reg [7:0]r_sec;
    reg [7:0]r_min;
    reg [7:0]r_hour;
    always @(rst, on_or_off, auto_rst, a_hour, hour, a_min, min, a_sec, sec) begin
        if((rst) || (~on_or_off) ||(auto_rst)) begin
            r_on = 0;
        end
        else begin
           if(sec[3:0] == 4'b1001) begin   // giay _9
              r_sec[3:0] <= 0;
              if(sec[7:4] == 4'b0101) begin   //giay 59
                 r_sec[7:4] <= 0;

                 if(min[3:0] == 4'b1001) begin  // phut _9
                    r_min[3:0] <= 0;
                    if(min[7:4] == 4'b0101) begin   // phut 59
                       r_min[7:4] <= 0;

                       if(hour == 8'b00100011)      // gio 23
                          r_hour <= 0;
                       else if(hour[3:0] == 4'b1001) begin    // gio _9
                          r_hour[3:0] <= 0;
                          r_hour[7:4] <= hour[7:4] + 1;
                       end
                       else
                          r_hour <= hour + 1;

                    end
                    else begin
                       r_hour <= hour;
                       r_min[7:4] <= min[7:4] + 1;
                    end
                 end
                 else begin
                    r_hour <= hour;
                    r_min <= min + 1;
                 end
              end
              else begin
                 r_sec[7:4] <= sec[7:4] + 1;
                 r_min <= min;
                 r_hour <= hour;
              end
           end
           else begin
              r_sec[3:0] <= sec[3:0] + 1;
              r_min <= min;
              r_hour <= hour;
           end

           if((a_hour == r_hour) && (a_min == r_min) && (a_sec == (r_sec)))
              r_on <= 1;
        end
    end

    assign on = r_on;
endmodule

```

Figure 9: *compare_with_alarm* block Verilog code

In this design, 5 *compare_with_alarm* blocks is used corresponding to 5 alarm values, each of which will operate independently of each other and transmit signals to 5 different notification blocks.

2.6. Notification block

The notification block - *beep* has following input datas: 1Hz clock pulse, reset signal and *compare_with_alarm* block output.

This block will be activated when receiving signals from *compare_with_alarm* block. When it is enabled, the unit will blink the LED signal to notify according to the click cycle (0 1) of the clock, and also counting from 0 to 59. The block will stop notification and return the counter to 0 when the signal from *compare_with_alarm* block is turned off or when the reset signal is triggered. When the counter reaches 59, the block will automatically transmit a returning signal to the *compare_with_alarm* block to interrupt the signal to the beep block, from which the notification will be turned off.

In this design, 5 *beep* blocks is used to notify 5 alarm values, they will operate independently of each other and output notifications to 5 different LEDs corresponding to 5 alarms.

```
2   //nhay ted trong 60s
3   module beep(output beeping,
4               output auto_reset,
5               input clk,
6               input on,
7               input reset);
8       wire off;
9       assign off = reset | ~on;
10      counter_beep cnt(.beeping(beeping), .auto_reset(auto_reset), .clk(clk), .reset(off)); // bo dem len toi 60
11  endmodule
12
13  module counter_beep(output beeping,
14                      output auto_reset,
15                      input reset,
16                      input clk);
```

Figure 10: *beep* block Verilog code

2.7. BCD decoding block

BCD decoding block - *BCD_decoder* has input data which is selected value between real time and alarm time via *multiplexer* port and reset signal.

multiplexer port Have input signal will be real time, alarm time and select signal. The selected signal is a button to view the alarm time. If the signal is activated, the selected data will be the alarm value. Otherwise, the selected data will be the real-time value.

```
module mux(output [7:0]out,

           input [7:0]A_time0,
           input [7:0]A_time1,
           input [7:0]A_time2,
           input [7:0]A_time3,
           input [7:0]A_time4,
           input [7:0]R_time,
           input [4:0]choose_A,
           input A_or_R);
    reg [7:0]r_out;
    always @(A_or_R, choose_A, R_time, A_time0, A_time1, A_time2, A_time3, A_time4) begin
        if(~A_or_R) begin
            casex(choose_A)
                5'bxxxx1: r_out = A_time0;
                5'bxxx10: r_out = A_time1;
                5'bxx100: r_out = A_time2;
                5'bx1000: r_out = A_time3;
                5'b10000: r_out = A_time4;
                default:  r_out = R_time;
            endcase
        end
        else
            r_out = R_time;
    end
    assign out = r_out;
endmodule
```

Figure 11: *multiplexer* port Verilog code

When *BCD_decoder* block receive value from multiplexer port, it will be decoded and displayed 7-segment LED.

```
//decode BCD to 7-segment LEDs
module BCD_decoder(output [6:0]out_led,
                   input [3:0]time_in,
                   input reset);
    reg [6:0]r_out_led;
    always @(reset, time_in) begin
        if(reset == 1)
            r_out_led = 7'b1000000;
        else begin
            case(time_in)
                4'd1: r_out_led = 7'b1111001;
                4'd2: r_out_led = 7'b0100100;
                4'd3: r_out_led = 7'b0110000;
                4'd4: r_out_led = 7'b0011001;
                4'd5: r_out_led = 7'b0010010;
                4'd6: r_out_led = 7'b0000010;
                4'd7: r_out_led = 7'b1111000;
                4'd8: r_out_led = 7'b0000000;
                4'd9: r_out_led = 7'b0010000;
                default: r_out_led = 7'b1000000;
            endcase
        end
    end
    assign out_led = r_out_led;
endmodule

```

Figure 12: *BCD_decoder* block Verilog code

In this design, 6 *BCD_decoder* blocks is used to support the display of 6 7-segment LED, each hour - minute - second value will use 2 blocks to represent the whole and decimal part of that value.

2.8. Main block

alarm_clock block is the main block including all of the above blocks. This unit is used to wire and link all the above blocks together to create a complete alarm clock.

```
module alarm_clock (output [4:0]BEEPING,          //5 leds used to beep 5 alarms
                    output [13:0]HH_clock,     //2 7-segment LEDs
                    output [13:0]MM_clock,     //2 7_segment LEDs
                    output [13:0]SS_clock,     //2 7-segment LEDs

                    input reset,               //1 switch
                    input clk,
                    input [6:0]set_time,       // 7 swith use to set alarm's time
                    input [2:0]set_H_M_S,      // 3 button use to save alarm's time
                    input [4:0]five_alarm,     // use 5 switchs
                    input change_real_time,    // 1 swith to set real time
                    input see_alarm_time);     // see alarm time

    wire clk_1s; // 1s clock
    slowclock set_1s_clk(clk, reset, clk_1s);

    wire [7:0] A_hour[4:0];
    wire [7:0] A_min[4:0];
    wire [7:0] A_sec[4:0];
    wire [7:0] five_a;

    wire [7:0] hour;
    wire [7:0] min;
    wire [7:0] sec;

    wire w_check;
    wire [2:0]w_H_M_S;
    check_time_in check(.check_out(w_check), .H_M_S(w_H_M_S),
                        .time_in({1'b0, set_time}), .is_hour(set_H_M_S));

    wire w_alarm_t;
    wire w_real_t;
    demux dmp(.out_A(w_alarm_t), .out_R(w_real_t),
              .enable(change_real_time), .in(w_check));
    save_input SI(.A_hour0(A_hour[0]), .A_hour1(A_hour[1]), .A_hour2(A_hour[2]), .A_hour3(A_hour[3]), .A_hour4(A_hour[4]),
                  .A_min0(A_min[0]), .A_min1(A_min[1]), .A_min2(A_min[2]), .A_min3(A_min[3]), .A_min4(A_min[4]),
                  .A_sec0(A_sec[0]), .A_sec1(A_sec[1]), .A_sec2(A_sec[2]), .A_sec3(A_sec[3]), .A_sec4(A_sec[4]),
                  .set_H_M_S(w_H_M_S), .set_time({1'b0, set_time}), .five_alarm(five_alarm), .check_in(w_alarm_t),
                  .reset(reset), .clock(clk));

    counter cout(.hour(hour), .min(min), .sec(sec), .clk(clk_1s), .reset(reset),
                 .set_time({1'b0, set_time}), .enable(w_H_M_S), .save(w_real_t));

    wire [4:0]auto_reset;    //stop beeping after 60s
    wire [4:0]turn_on;       //beeping
    //alarm 1
```

Figure 13: *alarm_clock* block Verilog code

```
    compare_with_alarm comparator_A1(.on(turn_on[0]),
                                     .a_hour(A_hour[0]), .hour(hour), .a_min(A_min[0]), .min(min), .a_sec(A_sec[0]), .sec(sec),
                                     .on_or_off(five_alarm[0]), .rst(reset), .auto_rst(auto_reset[0]));
    beep bp_A1(.beeping(BEEPING[0]), .auto_reset(auto_reset[0]),
                .clk(clk_1s), .on(turn_on[0]), .reset(reset));
    //alarm 2
    compare_with_alarm comparator_A2(.on(turn_on[1]),
                                     .a_hour(A_hour[1]), .hour(hour), .a_min(A_min[1]), .min(min), .a_sec(A_sec[1]), .sec(sec),
                                     .on_or_off(five_alarm[1]), .rst(reset), .auto_rst(auto_reset[1]));
    beep bp_A2(.beeping(BEEPING[1]), .auto_reset(auto_reset[1]),
                .clk(clk_1s), .on(turn_on[1]), .reset(reset));
    //alarm 3
    compare_with_alarm comparator_A3(.on(turn_on[2]),
                                     .a_hour(A_hour[2]), .hour(hour), .a_min(A_min[2]), .min(min), .a_sec(A_sec[2]), .sec(sec),
                                     .on_or_off(five_alarm[2]), .rst(reset), .auto_rst(auto_reset[2]));
    beep bp_A3(.beeping(BEEPING[2]), .auto_reset(auto_reset[2]),
                .clk(clk_1s), .on(turn_on[2]), .reset(reset));
    // alarm 4
    compare_with_alarm comparator_A4(.on(turn_on[3]),
                                     .a_hour(A_hour[3]), .hour(hour), .a_min(A_min[3]), .min(min), .a_sec(A_sec[3]), .sec(sec),
                                     .on_or_off(five_alarm[3]), .rst(reset), .auto_rst(auto_reset[3]));
    beep bp_A4(.beeping(BEEPING[3]), .auto_reset(auto_reset[3]),
                .clk(clk_1s), .on(turn_on[3]), .reset(reset));
    // alarm 5
    compare_with_alarm comparator_A5(.on(turn_on[4]),
                                     .a_hour(A_hour[4]), .hour(hour), .a_min(A_min[4]), .min(min), .a_sec(A_sec[4]), .sec(sec),
                                     .on_or_off(five_alarm[4]), .rst(reset), .auto_rst(auto_reset[4]));
    beep bp_A5(.beeping(BEEPING[4]), .auto_reset(auto_reset[4]),
                .clk(clk_1s), .on(turn_on[4]), .reset(reset));

    //display hour
    wire [7:0]w_hour;

    mux mux_H(.out(w_hour),
                .A_time0(A_hour[0]), .A_time1(A_hour[1]), .A_time2(A_hour[2]), .A_time3(A_hour[3]), .A_time4(A_hour[4]),
                .R_time(hour), .choose_A(five_alarm), .A_or_R(see_alarm_time));
    BCD_decoder _H(.out_led(HH_clock[6:0]), .time_in(w_hour[3:0]), .reset(reset));
    BCD_decoder H_(.out_led(HH_clock[13:7]), .time_in(w_hour[7:4]), .reset(reset));

    //display minute
    wire [7:0]w_min;

    mux mux_M(.out(w_min),
                .A_time0(A_min[0]), .A_time1(A_min[1]), .A_time2(A_min[2]), .A_time3(A_min[3]), .A_time4(A_min[4]),
                .R_time(min), .choose_A(five_alarm), .A_or_R(see_alarm_time));
    BCD_decoder _M(.out_led(MM_clock[6:0]), .time_in(w_min[3:0]), .reset(reset));
    BCD_decoder M_(.out_led(MM_clock[13:7]), .time_in(w_min[7:4]), .reset(reset));

    // display second
    wire [7:0]w_sec;

    mux mux_S(.out(w_sec),
                .A_time0(A_sec[0]), .A_time1(A_sec[1]), .A_time2(A_sec[2]), .A_time3(A_sec[3]), .A_time4(A_sec[4]),
                .R_time(sec), .choose_A(five_alarm), .A_or_R(see_alarm_time));
    BCD_decoder _S(.out_led(SS_clock[6:0]), .time_in(w_sec[3:0]), .reset(reset));
    BCD_decoder S_(.out_led(SS_clock[13:7]), .time_in(w_sec[7:4]), .reset(reset));


endmodule
```

Figure 14: *alarm_clock* block Verilog code

IV. SIMULATION

1. Modelsim Simulation

In Modelsim simulation, the slowclock block is ignored to make it easier to see.

In Modelsim simulation, the real time value is changed to 00h00’20 ", then assign 5 alarms in values from 00h00’30 " to 00h00’34 " respectively.

```
`timescale 1ns/1ns
module testbench_BTL();
   wire [4:0]BEEPING;          //5 leds used to beep 5 alarms
   wire [13:0]HH_clock;      //2 7-segment LEDs
   wire [13:0]MM_clock;      //2 7_segment LEDs
   wire [13:0]SS_clock;      //2 7-segment LEDs


   reg reset;                //1 switch
   reg clk;
   reg [6:0]set_time;        // 7 swith use to set alarm's time
   reg [2:0]set_H_M_S;       // 3 button use to save alarm's time
   reg [4:0]five_alarm;      // use 5 switchs
   reg change_real_time;     // 1 swith to set real time
   reg see_alarm_time;

   alarm_clock uut(.BEEPING(BEEPING), .HH_clock(HH_clock), .MM_clock(MM_clock), .SS_clock(SS_clock),
                   .reset(reset), .clk(clk),
                   .set_time(set_time), .set_H_M_S(set_H_M_S), .five_alarm(five_alarm), .change_real_time(change_real_time), .see_alarm_ti
   always #5 clk = ~clk;

   initial begin
       reset = 1;
       clk = 1;
       set_time = 0;
       set_H_M_S = 3'b111;
       change_real_time = 0;
       see_alarm_time = 1;
       five_alarm = 0; #10;
       reset = 0;
       change_real_time = 1;
       set_H_M_S = 3'b110;
       set_time = 7'h20; #20;
       change_real_time = 0; #10;
       set_time = 7'h40;
       five_alarm = 5'b00001; #20;
       set_time = 7'h41;
       five_alarm = 5'b00010; #20;
       set_time = 7'h42;
       five_alarm = 5'b00100; #20;
       set_time = 7'h43;
       five_alarm = 5'b01000; #20;
       set_time = 7'h44;
       five_alarm = 5'b10000; #20;
       set_H_M_S = 3'b111; #20;
       five_alarm = 5'b11111;  #300;
       five_alarm = 0; #60;
       $stop;
   end

endmodule
```

Figure 15: Testbench Verilog code for the Alarm clock system

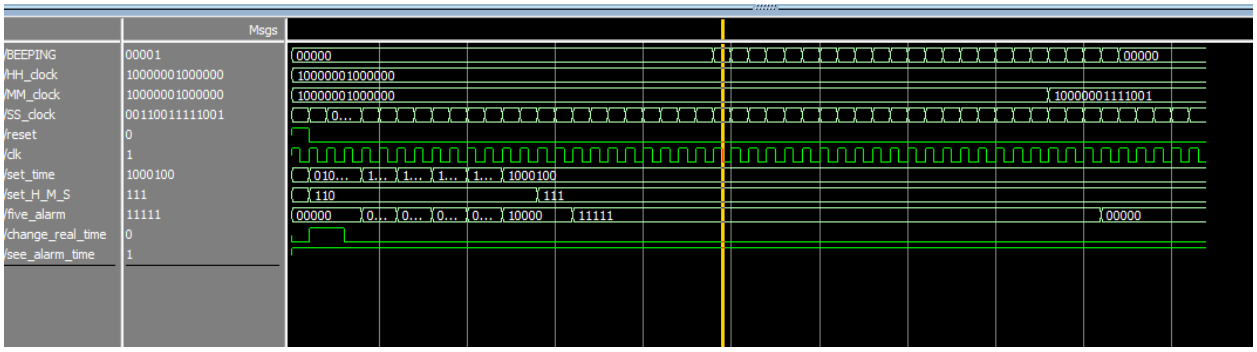

Figure 16: Waveform simulates for Alarm clock

2. Simulation on Altera DE2i Board

2.1. Import Assignments

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate
BEEPING[4]	Output	PIN_T21	5	B5_N1	2.5 V (default)		16mA (default)	2 (default)
BEEPING[3]	Output	PIN_W25	5	B5_N0	2.5 V (default)		16mA (default)	2 (default)
BEEPING[2]	Output	PIN_V27	5	B5_N0	2.5 V (default)		16mA (default)	2 (default)
BEEPING[1]	Output	PIN_T24	5	B5_N0	2.5 V (default)		16mA (default)	2 (default)
BEEPING[0]	Output	PIN_T23	5	B5_N0	2.5 V (default)		16mA (default)	2 (default)
HH_clock[13]	Output	PIN_D12	8	B8_N1	2.5 V (default)		16mA (default)	2 (default)
HH_clock[12]	Output	PIN_C12	8	B8_N1	2.5 V (default)		16mA (default)	2 (default)
HH_clock[11]	Output	PIN_C11	8	B8_N1	2.5 V (default)		16mA (default)	2 (default)
HH_clock[10]	Output	PIN_C10	8	B8_N0	2.5 V (default)		16mA (default)	2 (default)
HH_clock[9]	Output	PIN_F9	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
HH_clock[8]	Output	PIN_F8	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
HH_clock[7]	Output	PIN_E10	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
HH_clock[6]	Output	PIN_E9	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
HH_clock[5]	Output	PIN_D9	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
HH_clock[4]	Output	PIN_D8	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
HH_clock[3]	Output	PIN_C9	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
HH_clock[2]	Output	PIN_C8	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
HH_clock[1]	Output	PIN_B10	8	B8_N0	2.5 V (default)		16mA (default)	2 (default)
HH_clock[0]	Output	PIN_B9	8	B8_N1	2.5 V (default)		16mA (default)	2 (default)
MM_clock[13]	Output	PIN_B6	8	B8_N1	2.5 V (default)		16mA (default)	2 (default)
MM_clock[12]	Output	PIN_A11	8	B8_N0	2.5 V (default)		16mA (default)	2 (default)
MM_clock[11]	Output	PIN_A6	8	B8_N1	2.5 V (default)		16mA (default)	2 (default)
MM_clock[10]	Output	PIN_A7	8	B8_N1	2.5 V (default)		16mA (default)	2 (default)
MM_clock[9]	Output	PIN_A9	8	B8_N1	2.5 V (default)		16mA (default)	2 (default)
MM_clock[8]	Output	PIN_A10	8	B8_N0	2.5 V (default)		16mA (default)	2 (default)

Figure 17: Pins Assignment to output values

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate
MM_clock[7]	Output	PIN_D3	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
MM_clock[6]	Output	PIN_C3	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
MM_clock[5]	Output	PIN_C4	8	B8_N1	2.5 V (default)		16mA (default)	2 (default)
MM_clock[4]	Output	PIN_C5	8	B8_N1	2.5 V (default)		16mA (default)	2 (default)
MM_clock[3]	Output	PIN_C6	8	B8_N1	2.5 V (default)		16mA (default)	2 (default)
MM_clock[2]	Output	PIN_C7	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
MM_clock[1]	Output	PIN_A13	8	B8_N0	2.5 V (default)		16mA (default)	2 (default)
MM_clock[0]	Output	PIN_A14	8	B8_N0	2.5 V (default)		16mA (default)	2 (default)
SS_clock[13]	Output	PIN_D4	8	B8_N1	2.5 V (default)		16mA (default)	2 (default)
SS_clock[12]	Output	PIN_D5	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
SS_clock[11]	Output	PIN_E3	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
SS_clock[10]	Output	PIN_E4	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
SS_clock[9]	Output	PIN_E6	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
SS_clock[8]	Output	PIN_D7	8	B8_N1	2.5 V (default)		16mA (default)	2 (default)
SS_clock[7]	Output	PIN_D10	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
SS_clock[6]	Output	PIN_F10	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
SS_clock[5]	Output	PIN_F4	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
SS_clock[4]	Output	PIN_F6	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
SS_clock[3]	Output	PIN_AG30	5	B5_N2	2.5 V (default)		16mA (default)	2 (default)
SS_clock[2]	Output	PIN_F7	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
SS_clock[1]	Output	PIN_G7	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)
SS_clock[0]	Output	PIN_G8	8	B8_N2	2.5 V (default)		16mA (default)	2 (default)

Figure 18: Pins Assignment to output values

change_real_time	Input	PIN_C30	6	B6_N0	2.5 V (default)		16mA (default)
clk	Input	PIN_AJ16	4	B4_N2	2.5 V (default)		16mA (default)
five_alarm[4]	Input	PIN_N26	6	B6_N2	2.5 V (default)		16mA (default)
five_alarm[3]	Input	PIN_R26	6	B6_N2	2.5 V (default)		16mA (default)
five_alarm[2]	Input	PIN_R29	6	B6_N2	2.5 V (default)		16mA (default)
five_alarm[1]	Input	PIN_P30	6	B6_N2	2.5 V (default)		16mA (default)
five_alarm[0]	Input	PIN_R30	6	B6_N2	2.5 V (default)		16mA (default)
reset	Input	PIN_H25	6	B6_N0	2.5 V (default)		16mA (default)
see_alarm_time	Input	PIN_AE26	5	B5_N2	2.5 V (default)		16mA (default)
set_H_M_S[2]	Input	PIN_AF30	5	B5_N2	2.5 V (default)		16mA (default)
set_H_M_S[1]	Input	PIN_AE25	5	B5_N2	2.5 V (default)		16mA (default)
set_H_M_S[0]	Input	PIN_AA26	5	B5_N2	2.5 V (default)		16mA (default)
set_time[6]	Input	PIN_T28	6	B6_N2	2.5 V (default)		16mA (default)
set_time[5]	Input	PIN_U21	5	B5_N1	2.5 V (default)		16mA (default)
set_time[4]	Input	PIN_AB30	5	B5_N1	2.5 V (default)		16mA (default)
set_time[3]	Input	PIN_C2	8	B8_N2	2.5 V (default)		16mA (default)
set_time[2]	Input	PIN_V21	5	B5_N1	2.5 V (default)		16mA (default)
set_time[1]	Input	PIN_U30	5	B5_N0	2.5 V (default)		16mA (default)
set_time[0]	Input	PIN_V28	5	B5_N0	2.5 V (default)		16mA (default)

Figure 19: Pins Assignment to iutput values

2.2. Simulation image on the Altera DE2i Board

The following pictures show the cases when assigning alarms have values from 00h01’00 " to 00h01’04 " respectively.

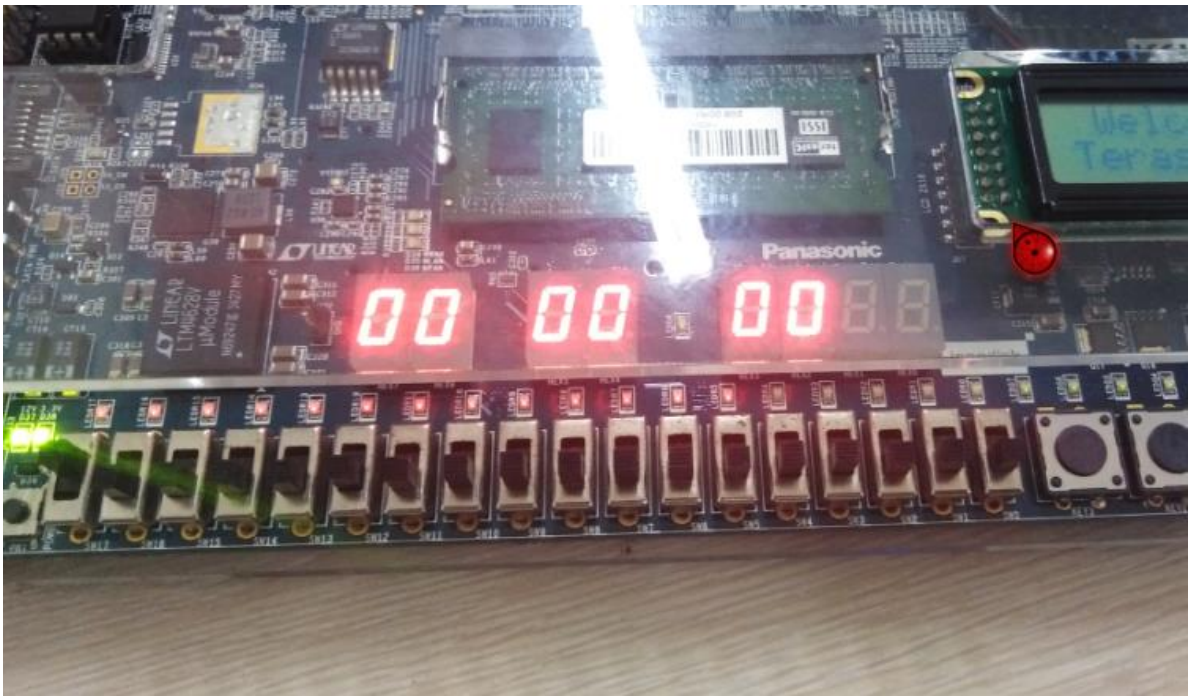

Figure 20: Image of the Board when the reset button is activated

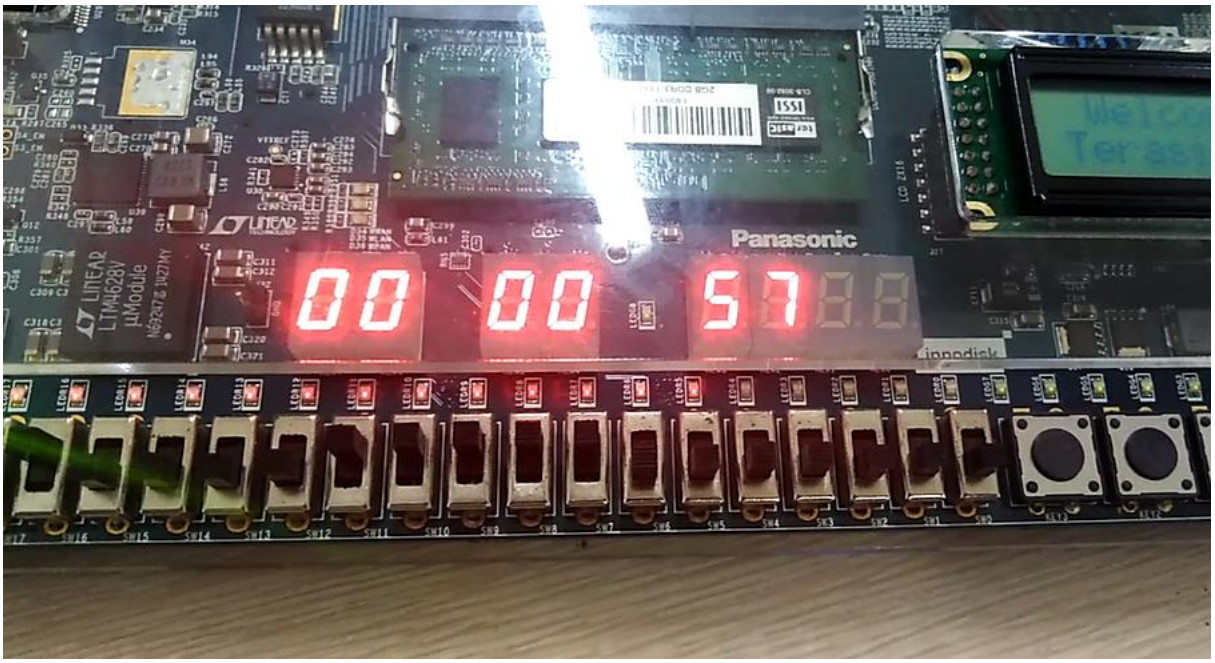

Figure 21: Image of Board when the reset button isn't activated

Figure 22: Image of Board when 1st, 3rd, 5th alarm is notified

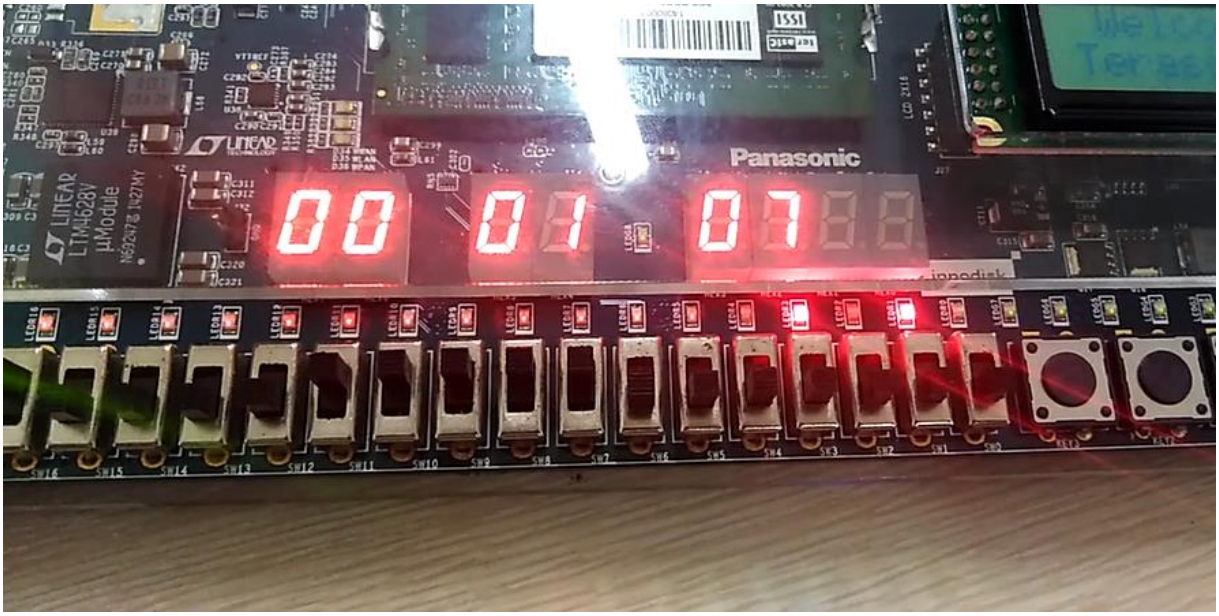

Figure 23: Image of Board when 2nd, 4th alarm is notified

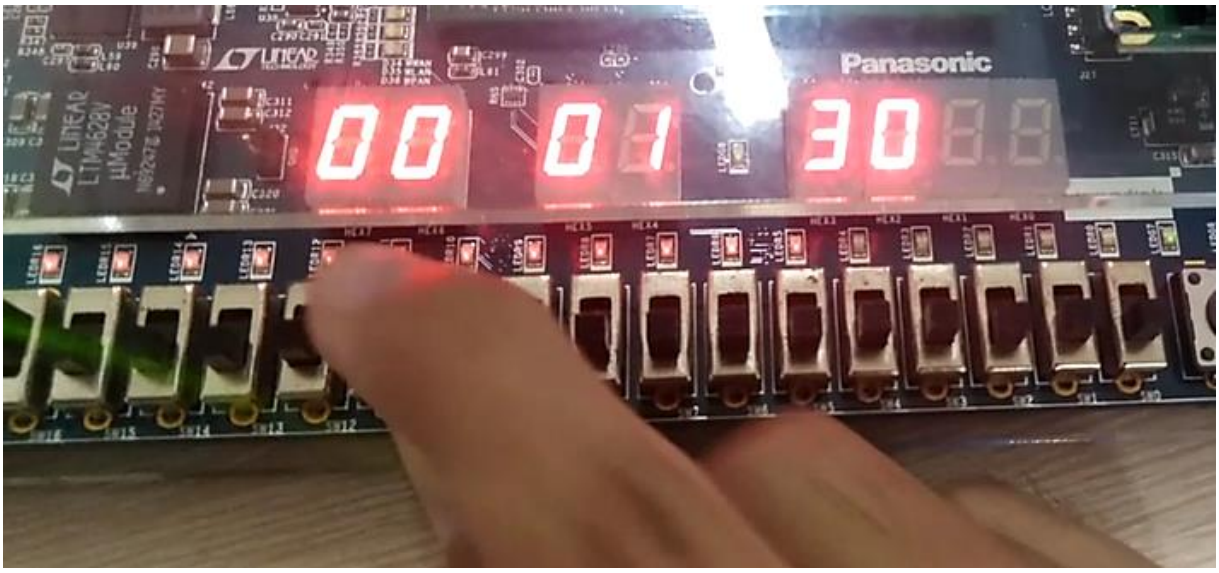

Figure 24: Image of the Board when all alarms are turned off manually

V. CONCLUSIONS AND FUTURE WORD

The alarm clock system designed and implemented by FPGA is more flexible and easier to deploy than control based on PLC microcontroller.

In this design, the team uses 5 LEDs to independently notify 5 alarms, the purpose is to make it easier to observe and test. However, according to what I have shown, we can easily improve to achieve a standard alarm clock system without changing the design too much.

The limitation of this project is not to design the most optimal system of resources to use as well as reduce latency to a minimum. I expect to receive more advises from teachers to improve the system.

REFERENCES

1. http://www.digilentinc.com/
2. http://kitboardmach.blogspot.com/2017/11/altera-de2-board-mach-thi-nghiem-fpga.html
3. https://forums.intel.com/s/question/0D50P00003yyRotSAE/how-to-create-a-simple-alarm-clock-using-verilog-code-and-run-it-on-fpga-board?language=en_US
4. Steve Kilts . (2007) . *Advanced FPGA design.*
   Book source: https://onlinelibrary.wiley.com/doi/book/10.1002/9780470127896
5. Image source:
   [1] :http://kitboardmach.blogspot.com/2014/10/kit-bo-mach-de2i-150-fpga-development.html

All of the links above had their last visit on December 7, 2019