CSCE 2301 Spring 2024
Project Report
Dr. Mohamed Shaalan
Digital Alarm Clock
Members:

Ahmed Yasser Ahmed Fawzy Abdelkader 900222336
Ahmed Mohamed Abdeen 900225815
Tony Yohana Kamal Asaad Gerges 900222981

Table of Contents:

Abstract:

During the course, we got introduced to the FPGA and learned its applications, such as counters, clocks and others. However, adding an alarm with a buzzer into the clock system and an adjustment option for the clock helped us develop a more advanced system that is close to clock systems we use in our daily life in our mobile phones. In order to build this system, we used the Vivado integrated design environment utilizing its built in features, including a framework for testing and verifying designs in addition to FPGA implementation. Throughout this report, we

delve deep into the development process from system design to hardware selection and software implementation, we highlight the challenges we encountered during the process and how we find solutions to them. The system was comprehensively tested, and the results demonstrated a successful implementation, providing a robust basis for future improvements.

Project Objective:

The aim of the project is to create a digital alarm clock. There are two modes of operation, the clock/alarm mode and the adjust mode. During the clock operation, when the current time matches current alarm time, LD0 blinks, and hitting any button stops the blinking. Based on pressing buttons the user can adjust time hour, time minute, alarm hour, and alarm minute. Also, different buttons increment and decrement the selected values.

Tools and technologies used:

Logism was used to model the circuit. Visio was used to model the FSM, ASM, control unit(CU) and Datapath(DP). Verilog was used for implementation and for writing onto the FPGA.

In clock mode:

6 counters that count when in clock mode
When adjusting the clock in adjust mode we load them
In adjust mode:
4 counters that are used for loading and not counting
6 states
We count using behavioral design
We use a lot of control signals
7Seg Bcd

There are 6 states: clock, alarm, adjust 0, adjust 1, adjust 2, adjust 3. The clock mode consists of 6 counters that count when in clock mode. This consists of 2 modulo 10 counters and 2 modulo 6 counters. When adjusting the clock in adjust mode we load values into the counter. In adjust mode, 4 counters are used for loading and not for counting. Using behavioral design, we implement if conditions

Validation Activities:

Constraint Files:

In order to properly test Verilog code, we opted to use constraint files and test on the FPGA. This was used for each and every part of all the module files.

- LED FOR DEBUGGING:
  We used LED states to know which state we are in. For example, to make sure we are in clock mode, we made all the LEDS as 1s, and a change in the LEDs implied a transitioning of states, To add, we used LEDs to know whether or not we are i alarm mode where if the set alarm time was equal to current time, a LED state was used to blink. This was done as a preliminary step before connecting a buzzer.
- 7 SEGMENT FOR DEBUGGING:

The 7 segment was used to display the clock. It was also utilized for the purpose of displaying the adjust modes where they either increment or decrement

- Push button test module created on its own for testing:

We created a module to test the push buttons where we ensured that they worked correctly.

Logisim Evolution:

We constructed the datapath and the control unit of the system using the ASM. In order to validate their effectiveness and accuracy, we decided to simulate them using Logisim. Therefore, after each one of us constructed their parts in the datapath and control unit, we merged them and tested their outputs. This helped us confirm that our datapath and control unit were a solid foundation which we could use to implement the program.

Contributions:

FSM: All

ASM:
Ahmed AbdelKader: Clock and Alarm States
Ahmed Abdeen: The Adjust_0 and Adjust_1 states
Tony Gerges: The Adjust_2 and Adjust_3 states

Control unit:
Ahmed AbdelKader: Clock and Alarm control unit states circuit
Ahmed Abdeen: The Adjust_0 and Adjust_1 states control unit circuit
Tony Gerges**:** The Adjust_2 and Adjust_3 states control unit circuit

Verilog: All

Datapath:
Ahmed Abdelkader: Overall work on datapath
Ahmed Abdeen: Elements related to adjust states
Tony Gerges: Elements related to adjust states

Logisim:
Ahmed Abdelkader: Overall work on Logisim
Ahmed Abdeen: Elements related to adjust states
Tony Gerges: Elements related to adjust states

Challenges faced:

Incrementing/decrementing the clock time:
During the system design process, we represented the increment and decrement feature in the ASM as increasing the register storing the clock time by one rather than using the up/down input of the counters. In order to stick to our ASM design, we chose to load the counters of the clock with the incremented/decremented value from the clock register and we added logic that accounted for all the cases in the incrementing/decrementing process, such as when reaching 59 minutes, incrementing will return the clock to zero.

Coordinating our work during the design process:
As mentioned above we used visio to design the control unit ,datapath, ASM and the FSM. First we constructed an FSM together on Zoom and then the ASM using visio. After that we used the ASM to construct the datapath and control unit. Each one of us took two states to construct their respective circuits using logisim and then we merged our work using visio. To implement the system using Vivado, we would meet in the lab to work on the implementation and testing together.

Development Process:

The design of our digital alarm clock came after several steps of planning and many meetings. We first met to create the pseudocode for the program which was one of the most important steps we made since it paved the way for the ASM and FSM charts (Figure 1). From there we worked on the ASM Chart while later also creating an FSM chart for clarity and for a deeper understanding of our design. Then came the datapath and control unit simultaneously. Finally, we tested out our implementation in Logisim Evolution (Figure 2).

The following final design implemented on verilog follows heavily from our preparations. The main module runs six states: clock, alarm, adjust 0 to adjust 3. We follow a mealy machine implementation. To move through states the user must use the push buttons as required. We store the states of the push buttons in buttonState (which is 5 bits), and we have found this to be an effective method of handling the push buttons. Notably, we use a Pushbutton_Detector module for debouncing and synchronizing buttons pressed [2]. For the entire program we have three clocks: 100MHz (for the board), 200Hz, 1Hz. We use the 200 Hz clock for the multiplexed 7 segment display and push button recognition, and we use the 1Hz for the clock time, decimal point, LED blinking, and buzzer.

Now onto the clock counter design. We use 6 counters split into units and tens for seconds, minutes, and hours. The counter module used also has a load option to load a certain value into the counter, which we use when adjusting. We use behavioral assignment to control the enables of the counters which was similar to our Logisim Evolution design. This allowed the counter to count up until 23:59 since that was our condition. Then we needed to adjust the clock.

We found that there were two main approaches we could go for either we use an Up/Down counter or edit the count then load it to the registers that store the count. We decided to do the latter because of the straightforwardness of the idea and connection to our design. Similarly, for the Alarm feature we have 4 counters which are loaded with the desired value for the Alarm time by the user.

References:

https://github.com/Ahmed-Fawzy14/Project-2-Digital-Alarm-Clock.git

[2] DD1 Lab

Datapath: DataPath.vsdx

FSM:
https://docs.google.com/document/d/1a3fiMjiTFtAaUMcsg8oXEUl208x1s-p4uzSQ2rqkDUw/edit?usp=sharing

ASM:
https://aucegypt0-my.sharepoint.com/:u:/r/personal/ahmed-fawzy_aucegypt_edu/_layouts/15/Doc.aspx?sourcedoc=%7B84513CBE-E59C-46A4-BC14-82A8EA93D6B9%7D&file=ASM%20Chart.vsdx&action=edit&mobileredirect=true&DefaultItemOpen=1&login_hint=ahmed-fawzy%40aucegypt.edu&ct=1716086350937&wdOrigin=OFFICECOM-WEB.START.REC&cid=2638c524-c22d-4527-ac0d-a6fc43cf8b17&wdPreviousSessionSrc=HarmonyWeb&wdPreviousSession=6eaa3033-097d-4d03-b8e1-bf10161e831c&or=PrevEdit

CU:
https://aucegypt0-my.sharepoint.com/:u:/r/personal/ahmed-fawzy_aucegypt_edu/_layouts/15/Doc.aspx?sourcedoc=%7BFB167D42-C500-4F34-AB99-B582AB3E0B69%7D&file=ControlUnit.vsdx&action=edit&mobileredirect=true&DefaultItemOpen=1&login_hint=ahmed-fawzy%40aucegypt.edu&ct=1716086354140&wdOrigin=OFFICECOM-WEB.START.EDGEWORTH&cid=99f6dca3-cad2-4623-8a1b-b28e8022c97d&wdPreviousSessionSrc=HarmonyWeb&wdPreviousSession=6eaa3033-097d-4d03-b8e1-bf10161e831c&or=PrevEdit

Appendices:

Verilog:-

```
module synchronizer(input  clk,rst,signal,    [2]
  output reg  sync_signal
);
reg Meta;
always@(posedge clk,posedge rst)begin
if(rst==1'b1)begin
  Meta<=0;
  sync_signal<=0;
  end
  else begin
  Meta<=signal;
  sync_signal<=Meta;
  end
  end
endmodule
module Rising_edge_detector(input clk, rst, w, output z);
```

```
reg [1:0] state, nextState;
parameter [1:0] A=2'b00, B=2'b01, C=2'b10;

always @ (w or state)
case (state)
A: if (w==0) nextState = A;
 else nextState = B;
B: if (w==1) nextState = C;
 else nextState = A;
C: if (w==0) nextState = A;
 else nextState = C;
default: nextState = A;

endcase

always @ (posedge clk or posedge rst) begin
if(rst) state <= A;
else state <= nextState;
end
assign z = (state == B);
endmodule
module pushButtonTest(
  input buttonState, //0: BTNC, 1: BTNR, 2: BTNL, 3: BTNU, 4: BTND
  input clk, reset,
  output led
   );

   wire btn;

      clockDivider #(250000) c1  (.clk(clk), .rst(reset), .clk_out(new_clk2));
      Pushbutton_Detector BTNC( .X(buttonState), .clk(new_clk2) , .rst(reset), .Z(led));




endmodule

module SevenSegCase( input [1:0]en,   [2]
                input clk_1Hz,
```

```
            input en_clock,
            input [3:0] num,
            output reg [6:0] segments,
            output reg [3:0] anode_active,
            output reg dp
            );



always @(*) begin



case(en)
0:begin
    anode_active =4'b1110;
    dp = 1;
end
1:begin
    anode_active=4'b1101;
    dp = 1;
end
2:begin
    anode_active=4'b1011;
    dp = en_clock ? clk_1Hz : 1;
 end
3:begin
    anode_active=4'b0111;
    dp = 1;
end
endcase

 case(num)
      0: segments = 7'b0000001;
      1: segments = 7'b1001111;
      2: segments = 7'b0010010;
      3: segments = 7'b0000110;
      4: segments = 7'b1001100;
```

```
            5: segments = 7'b0100100;
            6: segments = 7'b0100000;
            7: segments = 7'b0001111;
            8: segments = 7'b0000000;
            9: segments = 7'b0000100;


          default: segments = 7'b1111111;

        endcase
        end







endmodule

module alarm(
input clk,clk_out_200, rst, load, en,
input [1:0] en_Time, //1: Hr 0: Min
input [13:0]in_count, //current clolck for time NOT ALARM TIME
output [13:0]count, //current clolck for time NOT ALARM TIME
output [6:0] segments,
output [3:0] anodes
    );


  wire [1:0] twoOut;
  reg [3:0] muxOut;



    reg en_minsOnes;
    reg en_minsTens;
    reg en_hrsOnes;
    reg en_hrsTens;
```

```
   //n = Fin/2*Fout
//    assign count = in_count;

//wire reset;
//assign reset = ( count[21:20] == 2 &  count[19:16] == 3 &  count[15:12] == 5 & count[11:8]
== 9);


counterModN  #(4,10)minsOnesCount (clk_out_200,      (rst),en_minsOnes, count[3:0], load,
in_count[3:0]); //Min module alarm
counterModN  #(3,6)minsTensCount (clk_out_200,     (rst), en_minsTens, count[7:4], load,
in_count[7:4]); //Min Tens
counterModN  #(4,10)hrsOnesCount (clk_out_200,      (rst),en_hrsOnes, count[11:8], load,
in_count[11:8]); //Hr module ClockAlarm
counterModN  #(2,3)hrsTensCount (clk_out_200,      (rst),en_hrsTens, count[13:12], load,
in_count[13:12]); //Hr Tens


counterModN #(2,4)twoCount (.clk(clk_out_200), .reset(reset), .en(1'b1), .count(twoOut));

SevenSegCase seg(.en(twoOut), .num(muxOut),
.segments(segments),.anode_active(anodes),.dp(dp), .en_clock(en), .clk_1Hz(clk));



  always @(*) begin

  if(twoOut == 2'b00)
    muxOut = count[3:0];
  else if(twoOut == 2'b01)
    muxOut = count[7:4];
  else if(twoOut == 2'b10)
    muxOut = count[11:8];
  else if(twoOut == 2'b11)
    muxOut =  count[13:12];



end
```

```
    always@(count) begin
    if(en) begin
        if(en_Time == 2'b01) begin
            en_minsTens = 1'b1;
            en_hrsOnes = 1'b0;
            en_hrsTens = 1'b0;
            en_minsOnes  = 1'b1;

          end

       else if(en_Time == 2'b10) begin
          en_hrsOnes  = 1'b1;
          en_hrsTens = 1'b1;
          en_minsOnes  = 1'b0;
          en_minsTens = 1'b0;
//          en = 1'b0;
       end
       else begin
          en_hrsOnes  = 1'b0;
          en_hrsTens = 1'b0;
          en_minsOnes  = 1'b0;
          en_minsTens = 1'b0;
       end

      end


    end




endmodule
module clockDivider #(parameter n = 50000000)
(input clk, rst, output reg clk_out);
parameter WIDTH = $clog2(n);
reg [WIDTH-1:0] count;
```

```
// Big enough to hold the maximum possible value
// Increment count
always @ (posedge clk, posedge rst) begin
if (rst == 1'b1) // Asynchronous Reset
 count <= 32'b0;
else if (count == n-1)
 count <= 32'b0;
else
 count <= count + 1;
end
// Handle the output clock
always @ (posedge clk, posedge rst) begin
if (rst) // Asynchronous Reset
 clk_out <= 0;
else if (count == n-1)
 clk_out <= ~ clk_out;
end


endmodule
 module tb();


   reg clk, reset, en; //seconds ones enable
  wire [6:0] segments;
  wire [3:0] anodes;
    wire alarm;
  wire [21:0] count;

  prog uut
  (

    .clk(clk),
    .reset(reset),
    .en(en), //seconds ones enable
    .segments(segments),
    .anodes(anodes),
    .alarm(alarm),
    .count(count)
```

```
  );




  initial begin

    en = 1'b1;
    clk = 1;
    forever #25 clk=~clk;


    $display("At time: %t, count is: %b", $time, count);


  end


endmodule
module counterModN
  #(parameter x=8,n=4)(input clk, reset, en, output reg [x-1:0]count,input load, input
[x-1:0]load_clock);

always @(posedge clk, posedge reset) begin
 if (reset == 1)
    count <= 0; // non-blocking assignment
 // initialize flip flop here
 else begin
 if(en) begin
     if(load == 1)
        count<= load_clock;
     else if(count==n-1)
        count <=0;
     else
        count <= count + 1; // non-blocking assignment
     end
 end
 // normal operation
end
```

```
endmodule
module debouncer(input clk, rst, in, output out);   [2]
 reg q1,q2,q3;
always@(posedge clk, posedge rst) begin
 if(rst == 1'b1) begin
 q1 <= 0;
 q2 <= 0;
 q3 <= 0;
 end
else begin
 q1 <= in;
 q2 <= q1;
 q3 <= q2;
 end
end
assign out = (rst) ? 0 : q1&q2&q3;
endmodule
module Pushbutton_Detector(input X,input clk , input rst, output Z);
wire debounce;
wire signal1;
debouncer b( .clk(clk), .rst(rst), .in(X),  .out(debounce));
synchronizer q2(.signal(debounce), .clk(clk), .rst(rst),.sync_signal(signal1));
Rising_edge_detector re(  .clk(clk), .rst(rst),.w(signal1), .z(Z));



endmodule

module clockState(
   input clk,clk_out_200, reset, en,
   input [21:0] in_count,//enable for seconds counter
   input load,
   input [1:0] en_Time, //1: Hr 0: Min
   output [6:0] segments,
   output [3:0] anodes,
   output [21:0] count,
   output dp  //MADE THIS NOT REG
//   output alarm,
   );
```

```
  wire [1:0] twoOut;
  reg [3:0] muxOut;
  wire [21:0] count;


Clock clock(.clk(clk), .rst(reset), .en(en) , .in_count(in_count), .count(count),.load(load),
.en_Time(en_Time));


counterModN #(2,4)twoCount (.clk(clk_out_200), .reset(reset), .en(1'b1), .count(twoOut));

SevenSegCase seg(.en(twoOut), .num(muxOut),
.segments(segments),.anode_active(anodes),.dp(dp), .en_clock(en), .clk_1Hz(clk));



   always @(*) begin  // choosing which part of the time (hours or minutes) to be displayed based
on its anode position

  if(twoOut == 2'b00)
    muxOut = count[11:8];
  else if(twoOut == 2'b01)
    muxOut = count[15:12];
  else if(twoOut == 2'b10)
    muxOut = count[19:16];
  else if(twoOut == 2'b11)
    muxOut =  count[21:20];



end



endmodule

module programRun(
   input clk, reset, user_en, /*turn off clock*/ //en, //enable for seconds counter
   input [4:0] buttonState, //0: BTNC, 1: BTNR, 2: BTNL, 3: BTNU, 4: BTND
   output reg [6:0] segments,
```

```
   output reg [3:0] anodes,
   output dp,
   output reg alarmSound,
   output  reg[4:0] LEDstate

    );
    //THIS SHOULD COME FROM ALARM MODULE
    reg clk_to_clock;
    reg en_to_clock;
    wire new_clk, new_clk2;
    reg load, load_alarm;
    reg BL;

    reg [2:0] state, nextState;
    parameter [2:0]  Clock = 3'b000, Alarm = 3'b001, Adjust_0 = 3'b010, Adjust_1 = 3'b011,
Adjust_2 = 3'b100, Adjust_3 = 3'b101;

//    assign hours[6:0] = count_from_clock[21:16];


     wire [21:0] count_from_clock;
     reg [21:0] count_to_clock;
     reg [5:0] temp;
     wire dp_to_clock;
     wire [4:0] BTN;
     wire[13:0] count_from_alarm;
     reg[13:0] count_to_alarm;
     reg [1:0] en_Time_Alarm, en_Time;




    //assign count_to_clock_wire = count_to_clock; //**********************
    Pushbutton_Detector BTNC( .X(buttonState[0]), .clk(new_clk2) , .rst(rst), .Z(BTN[0]));
    Pushbutton_Detector BTNR( .X(buttonState[1]), .clk(new_clk2) , .rst(rst), .Z(BTN[1]));
    Pushbutton_Detector BTNL( .X(buttonState[2]), .clk(new_clk2) , .rst(rst), .Z(BTN[2]));
    Pushbutton_Detector BTNU( .X(buttonState[3]), .clk(new_clk2) , .rst(rst), .Z(BTN[3]));
    Pushbutton_Detector BTND( .X(buttonState[4]), .clk(new_clk2) , .rst(rst), .Z(BTN[4]));

   //ENABLE FOR CLOCKS
```

```
reg en_clock, en_alarm;

 wire [6:0] segAlarm;
 wire [6:0] segClock;
 wire [3:0] anodeAlarm;
 wire [3:0] anodeClock;

 always@(*) begin
    case(state)
    Clock:
    begin
          segments = segClock;
          anodes = anodeClock;
    end
    Alarm:
     begin
          segments = segAlarm;
          anodes = anodeAlarm;
    end
    Adjust_0:
    begin
          segments = segClock;
          anodes = anodeClock;

    end
    Adjust_1:
    begin
          segments = segClock;
          anodes = anodeClock;

    end
    Adjust_2:
    begin
          segments = segAlarm;
          anodes = anodeAlarm;

    end
    Adjust_3:
    begin
          segments = segAlarm;
```

```
            anodes = anodeAlarm;
        end

     endcase
    end


    clockState clock(.clk(clk_to_clock),.clk_out_200(new_clk2), .reset(reset),
.segments(segClock), .anodes(anodeClock), .dp(dp), .count(count_from_clock), .en(en_clock),
.in_count(count_to_clock), .load(load), .en_Time(en_Time));
    alarm AlarmModule (.clk(new_clk), .clk_out_200(new_clk2), .rst(reset),.segments(segAlarm),
.anodes(anodeAlarm), .count(count_from_alarm),  .in_count(count_to_alarm),
.load(load_alarm), .en_Time(en_Time_Alarm), .en(en_alarm));
    clockDivider #(250000) c1  (.clk(clk), .rst(reset), .clk_out(new_clk2));
    clockDivider #(50000000) c2 (.clk(clk), .rst(reset), .clk_out(new_clk));

       //////////////////////



    ////////////////////////////
    //In programRun:
    //Create ONE Clock object that we will change
    //create a clockDivder that gives 200hz
    //create a clockDivder that gives 1hz
    //when going to adjust: send 200hz, disbale, incremeant and send it as in_count
    //When we are returing to clock from adjust: enable clocks, send 1hz

    integer i;



wire zFlag;
assign zFlag = ((count_from_clock[11:8] == count_from_alarm[3:0]) &
(count_from_clock[15:12] == count_from_alarm[7:4]) & (count_from_clock[19:16] ==
count_from_alarm[11:8]) & (count_from_clock[21:20] == count_from_alarm[13:12])) &
(count_from_clock[3:0] == 0) & (count_from_clock[7:4] == 0);
```

```
    always@(BTN or state) begin
        case(state)

        Clock:
        begin
            if(zFlag & (count_from_clock[11:8] != 0))
                begin//If currentTime and alarmTime are the same AND any button is pressed or not
pressed expect BTNC
                    nextState = Alarm;
                    LEDstate={4'b0000, new_clk};
                   en_Time = 2'b00;
                    en_clock=1;
                       en_alarm = 0;
                     alarmSound = new_clk;
                     load_alarm = 0;
                   en_Time_Alarm = 2'b00;
                    clk_to_clock = new_clk;



                end
             else if(BTN == 5'b00001) begin

                    nextState = Adjust_0;
                    LEDstate= 5'b10001;
                    en_clock  = 0;
                    clk_to_clock = new_clk2;
                   en_Time = 2'b00;
                   en_Time_Alarm = 2'b00;
                   load_alarm = 0;
                   alarmSound = 0;
                    en_alarm = 0;


            end
            else begin
                nextState = Clock;
                LEDstate =5'b00000;
                en_clock = 1;
                clk_to_clock = new_clk;
```

```
        en_Time = 2'b00;
        en_Time_Alarm = 2'b00;
        load_alarm = 0;
        alarmSound = 0;
        en_alarm = 0;
    end
end

Alarm:
 begin

    if((BTN != 5'b00000))
    begin
    //add another empty state for the buzzer
        nextState = Clock;
        LEDstate =5'b00000;
        load = 0;
        en_Time = 2'b00;
        en_clock = 1;
        en_Time_Alarm = 2'b00;
        load_alarm = 0;
        alarmSound = 0;
       en_alarm = 0;
       clk_to_clock = new_clk;


    end
    else begin
       nextState  = Alarm;
       LEDstate = {4'b0000, new_clk};
       load = 0;
       en_Time = 2'b00;
       en_clock = 1;
       en_Time_Alarm = 2'b00;
       load_alarm = 0;
       alarmSound = new_clk;
       en_alarm = 0;
       clk_to_clock = new_clk;
```

```
    end

end

Adjust_0:
begin

    if(BTN == 5'b00001)
    begin
        nextState = Clock;
        en_clock = 1;
        LEDstate = 5'b00000;
        load = 0;
        clk_to_clock = new_clk;
        en_Time = 2'b00;
        en_Time_Alarm = 2'b00;
        load_alarm = 0;
        alarmSound = 0;
        en_alarm = 0;
    end

    else if (BTN == 5'b00010)//BTNL
    begin
        nextState = Adjust_1;
        LEDstate =5'b01001;
        load = 0;
        en_clock = 0;
        clk_to_clock = new_clk2;
        en_Time = 2'b00;
        en_Time_Alarm = 2'b00;
        load_alarm = 0;
        alarmSound = 0;
        en_alarm = 0;

    end
    else if (BTN == 5'b00100)//BTNR
    begin
        nextState = Adjust_3;
        en_clock = 0;
        load = 0;
```

```
    LEDstate =5'b00011;
    clk_to_clock = new_clk2;
    en_Time = 2'b00;
    en_Time_Alarm = 2'b00;
    load_alarm = 0;
    alarmSound = 0;
    en_alarm = 0;

end
else if(BTN == 5'b01000) begin //adjust time hour
nextState = Adjust_0;

if (count_from_clock[19:16] == 4'd9)
begin
    nextState = Adjust_0;
    if (count_from_clock [21:20] == 2'd1)
    begin

        count_to_clock[21:20] = 2'd2;
        count_to_clock[19:16] = 4'd0;
    end
    if (count_from_clock[21:20] == 2'd0)
    begin
        count_to_clock [21:20] =2'd1;
        count_to_clock [19:16] =4'd0;
     end
end
else if (count_from_clock [19:16] == 3 & count_from_clock [21:20] == 2)
begin

    count_to_clock [21:20] =0;
    count_to_clock [19:16] =0;

end

else
begin
count_to_clock [19:16] =  count_from_clock [19:16] + 4'd1;
count_to_clock [21:20] = count_from_clock[21:20];
```

```
end


    nextState = Adjust_0;
    load = 1;
    LEDstate = 5'b10001;
    en_Time = 2'b10;
    clk_to_clock = new_clk2;
    en_clock = 0;
    en_Time_Alarm = 2'b00;
    load_alarm = 0;
    alarmSound = 0;
    en_alarm = 0;

end
 else if(BTN == 5'b10000) begin //adjust time hour
   nextState = Adjust_0;

if (count_from_clock[19:16] == 4'd0)
begin
       if (count_from_clock[21:20] == 2'd0)
    begin
        count_to_clock [21:20] =2'd2;
        count_to_clock [19:16] =4'd3;
     end
    else if (count_from_clock [21:20] == 2'd2)
    begin

        count_to_clock[21:20] = 2'd1;
        count_to_clock[19:16] = 4'd9;
    end
    else if (count_from_clock[21:20] == 2'd1)
    begin
        count_to_clock [21:20] =2'd0;
        count_to_clock [19:16] =4'd9;
     end

end
else
begin
```

```
      count_to_clock [19:16] =  count_from_clock [19:16] - 4'd1;
      count_to_clock [21:20] = count_from_clock[21:20];

    end


      nextState = Adjust_0;
      load = 1;
      LEDstate = 5'b10001;
      en_Time = 2'b10;
      clk_to_clock = new_clk2;
      en_clock = 0;
      en_Time_Alarm = 2'b00;
      load_alarm = 0;
      alarmSound = 0;
      en_alarm = 0;
     end
   else
   begin
     nextState = Adjust_0;
     LEDstate =5'b10001;
     en_clock = 0;
     load = 0;
     clk_to_clock = new_clk2;
     en_Time = 2'b00;
     en_Time_Alarm = 2'b00;
     load_alarm = 0;
     alarmSound = 0;
      en_alarm = 0;

   end
end
Adjust_1:
begin

   if(BTN == 5'b00001)
   begin
      nextState = Clock;
      load = 0;
      LEDstate =5'b00000;
```

```
    en_clock = 1;
    en_Time = 2'b00;
    en_Time_Alarm = 2'b00;
    clk_to_clock = new_clk;
    alarmSound = 0;
    en_alarm = 0;
    load_alarm = 0;

end
else if (BTN == 5'b00010)
begin
nextState = Adjust_2;
LEDstate =5'b00101;
en_clock = 0;
clk_to_clock = new_clk2;
en_Time = 2'b00;
en_Time_Alarm = 2'b00;
load = 0;
alarmSound = 0;
en_alarm = 0;
load_alarm = 0;

end
else if (BTN == 5'b00100)
begin
nextState = Adjust_0;
en_clock = 0;
clk_to_clock = new_clk2;
en_Time = 2'b00;
load = 0;
en_Time_Alarm = 2'b00;
LEDstate =5'b10001;
 alarmSound = 0;
en_alarm = 0;
load_alarm = 0;


end
```

```
else if(BTN == 5'b01000) begin //adjust time min
    nextState = Adjust_1;
    if (count_from_clock[11:8] == 4'd9)
begin
       if (count_from_clock [15:12] == 4'd5)
       begin
          count_to_clock[11:8] = 4'd0;
          count_to_clock[15:12] = 4'd0;
       end
       else if (count_from_clock [15:12] == 4'd4)
       begin
          count_to_clock[11:8] = 4'd0;
          count_to_clock[15:12] = 4'd5;
       end
        if (count_from_clock [15:12] == 4'd3)
       begin
          count_to_clock[11:8] = 4'd0;
          count_to_clock[15:12] = 4'd4;
       end
        if (count_from_clock [15:12] == 4'd2)
       begin
          count_to_clock[11:8] = 4'd0;
          count_to_clock[15:12] = 4'd3;
       end
        if (count_from_clock [15:12] == 4'd1)
       begin
          count_to_clock[11:8] = 4'd0;
          count_to_clock[15:12] = 4'd2;
       end
        if (count_from_clock [15:12] == 4'd0)
       begin
          count_to_clock[11:8] = 4'd0;
          count_to_clock[15:12] = 4'd1;
       end

 end
else
begin
count_to_clock [11:8] =  count_from_clock [11:8] + 4'd1;
count_to_clock [15:12] = count_from_clock[15:12];
```

```
    end


        nextState = Adjust_1;
        load = 1;
        LEDstate = 5'b01001;
        en_Time = 2'b01;
        clk_to_clock = new_clk2;
        en_clock = 0;
        en_Time_Alarm = 2'b00;
         alarmSound = 0;
      en_alarm = 0;
      load_alarm = 0;


      end
      else if(BTN == 5'b10000) begin //adjust time min
                  nextState = Adjust_1;

if (count_from_clock[11:8] == 4'd0)
      begin
            if (count_from_clock [15:12] == 4'd5)
            begin
               count_to_clock[11:8] = 4'd9;
               count_to_clock[15:12] = 4'd4;
            end
            else if (count_from_clock [15:12] == 4'd4)
            begin
               count_to_clock[11:8] = 4'd9;
               count_to_clock[15:12] = 4'd3;
            end
             if (count_from_clock [15:12] == 4'd3)
            begin
               count_to_clock[11:8] = 4'd9;
               count_to_clock[15:12] = 4'd2;
            end
             if (count_from_clock [15:12] == 4'd2)
            begin
               count_to_clock[11:8] = 4'd9;
```

```
            count_to_clock[15:12] = 4'd1;
        end
         if (count_from_clock [15:12] == 4'd1)
        begin
            count_to_clock[11:8] = 4'd9;
            count_to_clock[15:12] = 4'd0;
        end
         if (count_from_clock [15:12] == 4'd0)
        begin
            count_to_clock[11:8] = 4'd9;
            count_to_clock[15:12] = 4'd5;
        end

 end
else
begin
count_to_clock [11:8] =  count_from_clock [11:8] - 4'd1;
count_to_clock [15:12] = count_from_clock[15:12];

end


    load = 1;
    LEDstate = 5'b01001;
    en_Time = 2'b01;
    clk_to_clock = new_clk2;
    en_clock = 0;
   en_Time_Alarm = 2'b00;
     alarmSound = 0;
    en_alarm = 0;
     load_alarm = 0;
    nextState = Adjust_1;

end
else
begin

nextState = Adjust_1;
en_clock = 0;
load = 0;
```

```
    LEDstate = 5'b01001;
    clk_to_clock = new_clk2;
    en_Time = 2'b00;
    en_Time_Alarm = 2'b00;
    load_alarm = 0;
    alarmSound = 0;
    en_alarm = 0;

    end
 end
Adjust_2:
begin

    if(BTN == 5'b00001)
    begin
        nextState = Clock;
        en_clock = 1;
        LEDstate = 5'b00000;
        load = 0;
        clk_to_clock = new_clk;
        en_Time = 2'b00;
        en_Time_Alarm = 2'b00;
        load_alarm = 0;
        alarmSound = 0;
        en_alarm = 0;
    end

    else if (BTN == 5'b00100)//BTNL
    begin
        nextState = Adjust_1;
        LEDstate =5'b01001;
        load = 0;
        en_clock = 0;
        clk_to_clock = new_clk2;
        en_Time = 2'b00;
        en_Time_Alarm = 2'b00;
        load_alarm = 0;
        alarmSound = 0;
        en_alarm = 1;
```

```
        end
        else if (BTN == 5'b00010)//BTNR
        begin
           nextState = Adjust_3;
           load = 0;
           LEDstate =5'b00011;
           clk_to_clock = new_clk2;
           en_Time = 2'b00;
           en_clock = 0;
           en_Time_Alarm = 2'b00;
           load_alarm = 0;
           alarmSound = 0;
           en_alarm = 1;

        end
else if(BTN == 5'b01000) begin //adjust alarm hour
         nextState = Adjust_2;
         if (count_from_alarm [11:8] == 3 & count_from_alarm [13:12] == 2)begin
               count_to_alarm [13:12] =0;
               count_to_alarm [11:8] =0;
         end
         else if (count_from_alarm[11:8] == 4'd9)
            begin
               count_to_alarm [11:8] =  0;
               count_to_alarm [13:12] = count_from_alarm[13:12] + 2'd1;

           end
         else begin
           count_to_alarm [11:8] =  count_from_alarm [11:8] + 4'd1;
           count_to_alarm [13:12] = count_from_alarm[13:12];
         end


           nextState = Adjust_2;
           load = 0;
           LEDstate = 5'b00101;
           en_Time = 2'b00;
           clk_to_clock = new_clk2;
           en_clock = 0;
           en_Time_Alarm = 2'b10;
```

```
    load_alarm = 1;
    alarmSound = 0;
    en_alarm = 1;

end
else if(BTN == 5'b10000) begin //adjust alarm hour
 nextState = Adjust_2;
  if (count_from_alarm [11:8] == 0 & count_from_alarm [13:12] ==0)begin
        count_to_alarm [13:12] =2;
        count_to_alarm [11:8] =3;
 end
 else if (count_from_alarm[11:8] == 4'd0)
     begin
        count_to_alarm [11:8] =  4'd9;
        count_to_alarm [13:12] = count_from_alarm[13:12] - 2'd1;

    end
 else begin
    count_to_alarm [11:8] =  count_from_alarm [11:8] - 4'd1;
    count_to_alarm [13:12] = count_from_alarm[13:12];
 end



    nextState = Adjust_2;
    load = 0;
    LEDstate = 5'b00101;
    en_Time = 2'b00;
    clk_to_clock = new_clk2;
    en_clock = 0;
    en_Time_Alarm = 2'b10;
    load_alarm = 1;
    alarmSound = 0;
    en_alarm = 1;

end

else
begin
    nextState = Adjust_2;
    load = 0;
```

```
        LEDstate = 5'b00101;
        en_Time = 2'b00;
        clk_to_clock = new_clk2;
        en_clock = 0;
        en_Time_Alarm = 2'b00;
        load_alarm = 0;
        alarmSound = 0;
        en_alarm = 1;
    end
 end

Adjust_3:
begin

    if(BTN == 5'b00001)
    begin
        nextState = Clock;
        en_clock = 1;
        LEDstate = 5'b00000;
        load = 0;
        clk_to_clock = new_clk;
        en_Time = 2'b00;
        en_Time_Alarm = 2'b00;
        load_alarm = 0;
        alarmSound = 0;
        en_alarm = 0;
    end

    else if (BTN == 5'b00100)//BTNL
    begin
        nextState = Adjust_2;
        LEDstate =5'b00101;
        load = 0;
        en_clock = 0;
        clk_to_clock = new_clk2;
        en_Time = 2'b00;
        en_Time_Alarm = 2'b00;
        load_alarm = 0;
        alarmSound = 0;
        en_alarm = 1;
```

```
        end
        else if (BTN == 5'b00010)//BTNR
        begin
           nextState = Adjust_0;
           load = 0;
           LEDstate =5'b10001;
           clk_to_clock = new_clk2;
           en_Time = 2'b00;
           en_clock = 0;
           en_Time_Alarm = 2'b00;
           load_alarm = 0;
           alarmSound = 0;
           en_alarm = 1;

        end
else if(BTN == 5'b01000) begin //adjust alarm min
         nextState = Adjust_3;
         if (count_from_alarm [3:0] == 9 & count_from_alarm [7:4] == 5)begin
               count_to_alarm [3:0] =0;
               count_to_alarm [7:4] =0;
         end
         else if (count_from_alarm[3:0] == 4'd9)
            begin
               count_to_alarm [3:0] =  0;
               count_to_alarm [7:4] = count_from_alarm[7:4] + 2'd1;

           end
         else begin
           count_to_alarm [3:0] =  count_from_alarm [3:0] + 4'd1;
           count_to_alarm [7:4] = count_from_alarm[7:4];
         end


           nextState = Adjust_3;
           load = 0;
           LEDstate = 5'b00011;
           en_Time = 2'b00;
           clk_to_clock = new_clk2;
           en_clock = 0;
```

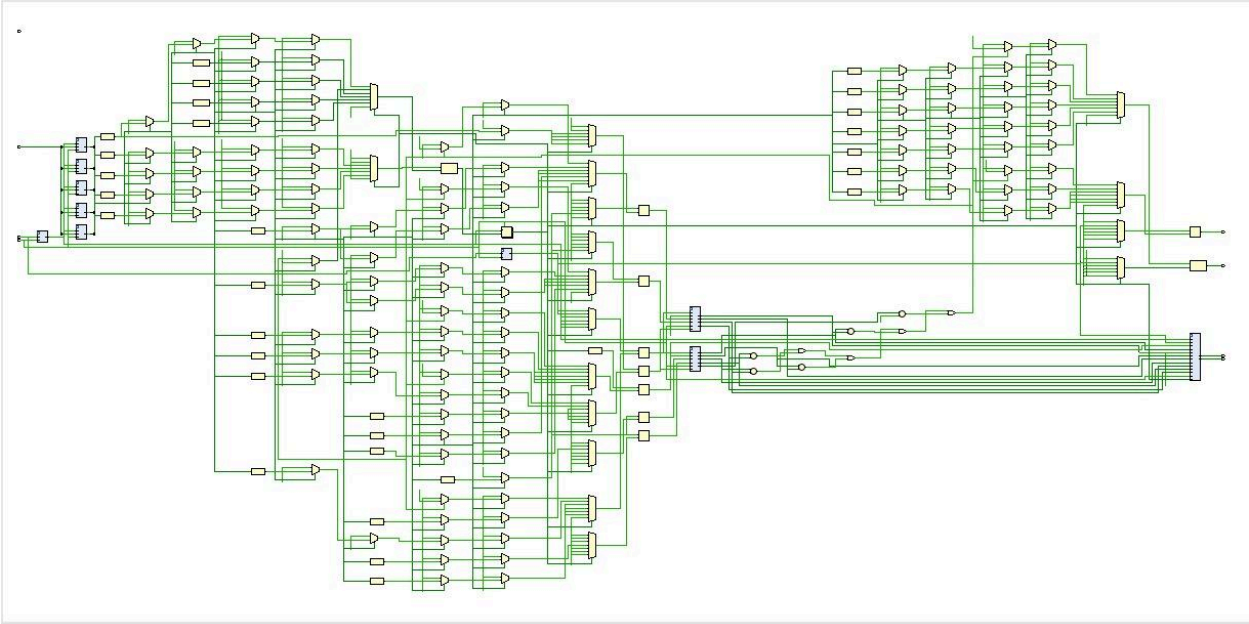
```
            en_Time_Alarm = 2'b01;
            load_alarm = 1;
            alarmSound = 0;
            en_alarm = 1;

          end
         else if(BTN == 5'b10000) begin //adjust alarm hour
          nextState = Adjust_3;
           if (count_from_alarm [3:0] == 0 & count_from_alarm [7:4] ==0)begin
                count_to_alarm [7:4] =5;
                count_to_alarm [3:0] =9;
          end
          else if (count_from_alarm[3:0] == 4'd0)
             begin
                count_to_alarm [3:0] =  4'd9;
                count_to_alarm [7:4] = count_from_alarm[7:4] - 2'd1;

            end
          else begin
            count_to_alarm [3:0] =  count_from_alarm [3:0] - 4'd1;
            count_to_alarm [7:4] = count_from_alarm[7:4];
          end


            nextState = Adjust_3;
            load = 0;
            LEDstate = 5'b00101;
            en_Time = 2'b00;
            clk_to_clock = new_clk2;
            en_clock = 0;
            en_Time_Alarm = 2'b01;
            load_alarm = 1;
            alarmSound = 0;
            en_alarm = 1;

          end
//           else if(BTN == 5'b10000) begin //adjust time hour
//             nextState = Adjust_0;

//           if (count_from_clock[19:16] == 4'd0)
```

```
//          begin
//               if (count_from_clock[21:20] == 2'd0)
//            begin
//               count_to_clock [21:20] =2'd2;
//               count_to_clock [19:16] =4'd3;
//             end
//            else if (count_from_clock [21:20] == 2'd2)
//            begin

//               count_to_clock[21:20] = 2'd1;
//               count_to_clock[19:16] = 4'd9;
//            end
//            else if (count_from_clock[21:20] == 2'd1)
//            begin
//               count_to_clock [21:20] =2'd0;
//               count_to_clock [19:16] =4'd9;
//             end

//          end
//          else
//          begin
//             count_to_clock [19:16] =  count_from_clock [19:16] - 4'd1;
//             count_to_clock [21:20] = count_from_clock[21:20];

//          end


//             nextState = Adjust_0;
//             load = 1;
//             LEDstate = 5'b10001;
//             en_Time = 2'b10;
//             clk_to_clock = new_clk2;
//             en_clock = 0;
//             en_Time_Alarm = 2'b00;
//             load_alarm = 0;
//             alarmSound = 0;
//             en_alarm = 0;
//            end
          else
          begin
```

```
            nextState = Adjust_3;
            load = 0;
            LEDstate = 5'b00011;
            en_Time = 2'b00;
            clk_to_clock = new_clk2;
            en_clock = 0;
            en_Time_Alarm = 2'b00;
            load_alarm = 0;
            alarmSound = 0;
            en_alarm = 1;
          end
        end




      default: nextState = Clock;
endcase



      //else if(BTN == 5'b10000) begin //adjust alarm hour
      //alarmReg[6:0] = alarmReg[6:0] - 1'b1;
      //end

   end
   always@(posedge new_clk2 or posedge reset)
      begin
      if(reset )begin
            state <=   Clock;
       end
          else
       begin
          state <= nextState;
       end
    end
```

endmodule

Schematic Design:



Pseudocode used for the system design:

```
//Modules: (go into each module and find these):
//Registers and MUX … which will go into the DP

//Counters (Control Unit)
//Clock divider (Control Unit)
//But the registers they use are DP

-----------------------------------------------------------------------------------------------------------------------------------------

If sub is = 0 and Q is 0 then you were in adjust and want to go to clock
If sub is = 0 and Q is 1 then you were in clock mode and want to go to adjust

CurrentTime; (14 bit register)

3-0 minute one //Mod 10 so we need to have corresponding counters
6-4 minute tens //Mod 6 so we need to have corresponding counters
10-7 hour one //Mod 10 so we need to have corresponding counters
13-11 hour tens  //Mod 6 so we need to have corresponding counters

buttonState; (5 bit register)
0 BTNC
1 BTNL
2 BTNR
3 BTNU
4 BTND

SetAlarmTime; (14 bit register)

3-0 minute one
6-4 minute tens
10-7 hour one
13-11 hour tens



LD0 = 0;  //Led 0 (1 bit variable)
Dp2 = 0;  //Decimal point 2 (1 bit variable) //This is what makes it blink (this is a second)
ClockAdjust = 1; //Current mode (1 bit variable)
AlarmSound = 0;  //Should the alarm make a sound (1 bit variable)

if(buttonState_0 == 1) do
ClockAdjust = 0;
End if;

while(ClockAdjust  == 1) do
ENABLE CLOCKCounter;
if(buttonState_0 == 1) do
ClockAdjust = 0;
End if;

Dp2 = 1; //We should blink the decimal point (MUXs 0 and 1 as inputs and selection from control unit)
//Time registers = signal from the counter
if(CurrentTime == SetAlarmTime) //Zflag
AlarmSound = 1;
LD0 = 1; //This should blink //c2 this is what will make it blink
        while(buttonState == 0) do //No ones //We can send it to the CU so it can loop
                AlarmSound = 1;
                LD0 = 1;
        End while;
        AlarmSound = 0;
        LD0 = 0;
End if;
End if;

End while;
```

Should we stop the clock?

```
LD0 = 1;  //Led 0 (1 bit variable)
Dp2 = 0;  //Decimal point 2 (1 bit variable)
buttonState; (5 bit register)
LEDstate; (5 bit register)
parameter; (7 bit register)
//0 LD0
//1 LD12
//2 LD13
//3 LD14
//4 LD15
ClockAdjust = 0; //Current mode (1 bit variable) (0 means adjust mode)
CurrentTimeMin; (7 bit register)

CurrentTimeHour; (7 bit register)
SetAlarmMin;
SetAlarmHour;
//While in the time options we will display clock on 7SEG, while in the alarm options we will display alarm on
7SEG.
LEDstate_1 = 1;


if(buttonState_0 == 1) do
ClockAdjust = 1;
End if;

//Up/Down mod-4 counter for selection of 4x1 MUX that picks the bits that we want to change
//With buttonState_1 as -1 and buttonState_2 as +1
case(sel) //c1
00: CurrentTimeMin & LD12 = 1 & LD13 = LD14 = LD 15 = 0
01: CurrentTimeHour & LD13 = 1 & LD12 = LD14 = LD 15 = 0
10: SetAlarmMin & LD14 = 1 &  LD13 = LD12 = LD 15 = 0
11: SetAlarmHour & LD15 = 1 & LD12 = LD13 = LD 14 = 0
endcase

//Adjust
if(buttonState_3 == 1) do
TheOneSelected<= TheOneSelected+ 1;
end if;
if(buttonState_4 == 1) do
TheOneSelected<= TheOneSelected- 1;
end if;
```

Frequency in = 100Mhz = 1x10^8

We want one tick to be 60hz since 1hz is one second
and one minute is 60 seconds.

Frequency out = 60hz