# Digital Alarm Clock

Adham Gohar - 900225576
Toqa Hassanien - 900202324
Khadeejah Iraky - 900222731

Department of Computer Science and Engineering, The American University in Cairo

Dr. Mohamed Shalan

May 18, 2024

# Digital Alarm Clock

## Table of Contents



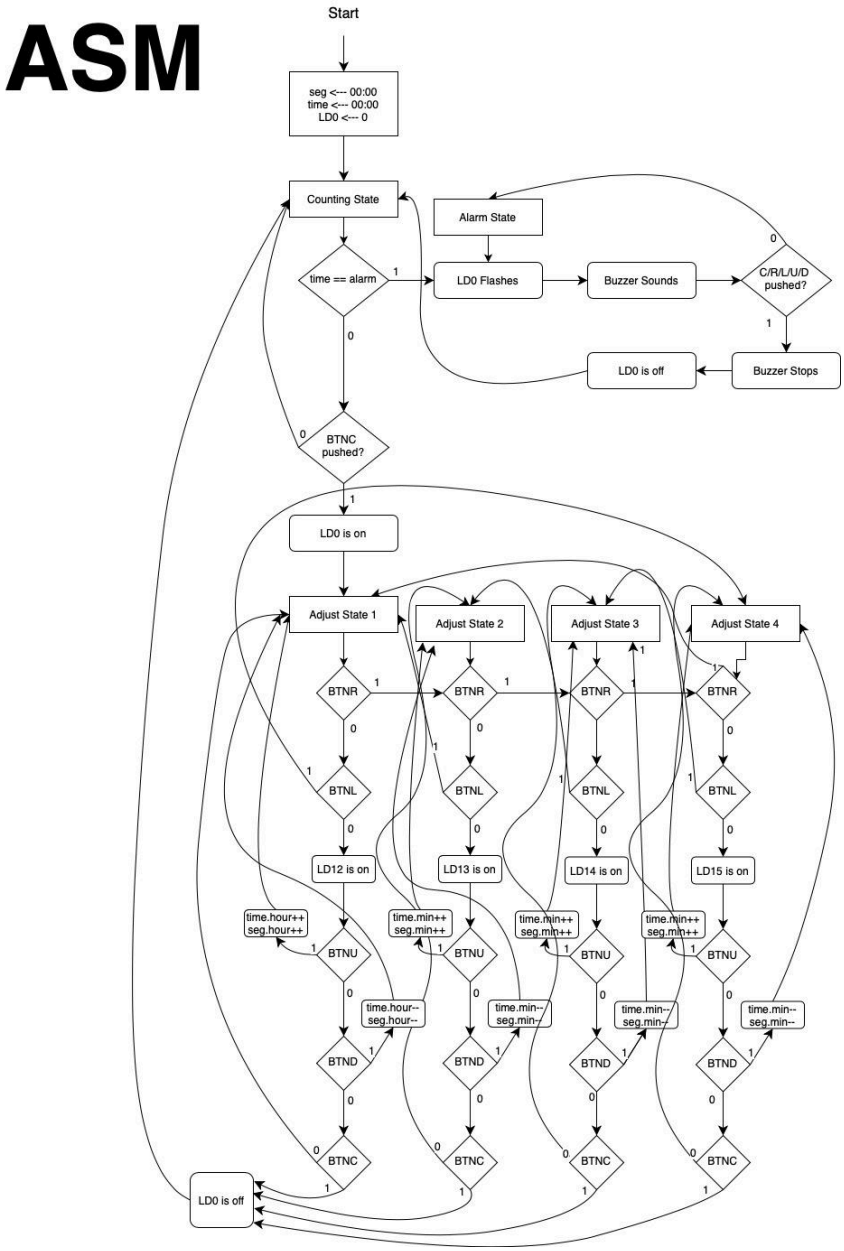
---

Project Github Repository: https://github.com/Adham-Gohar/Digital-Alarm-Clock

# Diagrams

## ASM Design

In the ASM we detailed how the logic would flow properly, this helped us visualize and bring down the project and showed us how we can implement it in detail. This is the third version of the ASM, as the older versions require modifications. We have the alarm state logic on the right and the adjust state logic at the bottom, with each adjust mode next to each other.



## FSM Design

The FSM we designed followed a Moore design with a hierarchical methodology for the Adjust Mode. We have four main modes; Start/ Reset mode, Counting mode, Alarm mode, and Adjust mode (has four modes within it; Adjust Time Hour mode, Adjust Time Minutes mode, Adjust Alarm Hour mode, Adjust Alarm Minutes mode), which makes a total of 7 modes. Additionally, we have a mini FSM for the hierarchical design to detail the increment and decrement parts of all four adjust modes and how they work.

**Outputs**

LD0
LD12
LD13
LD14
LD15
Buzzer
Seg
dot

**Inputs**

Clock
BTNC
BTNR
BTNL
BTNU
BTND

**Key**

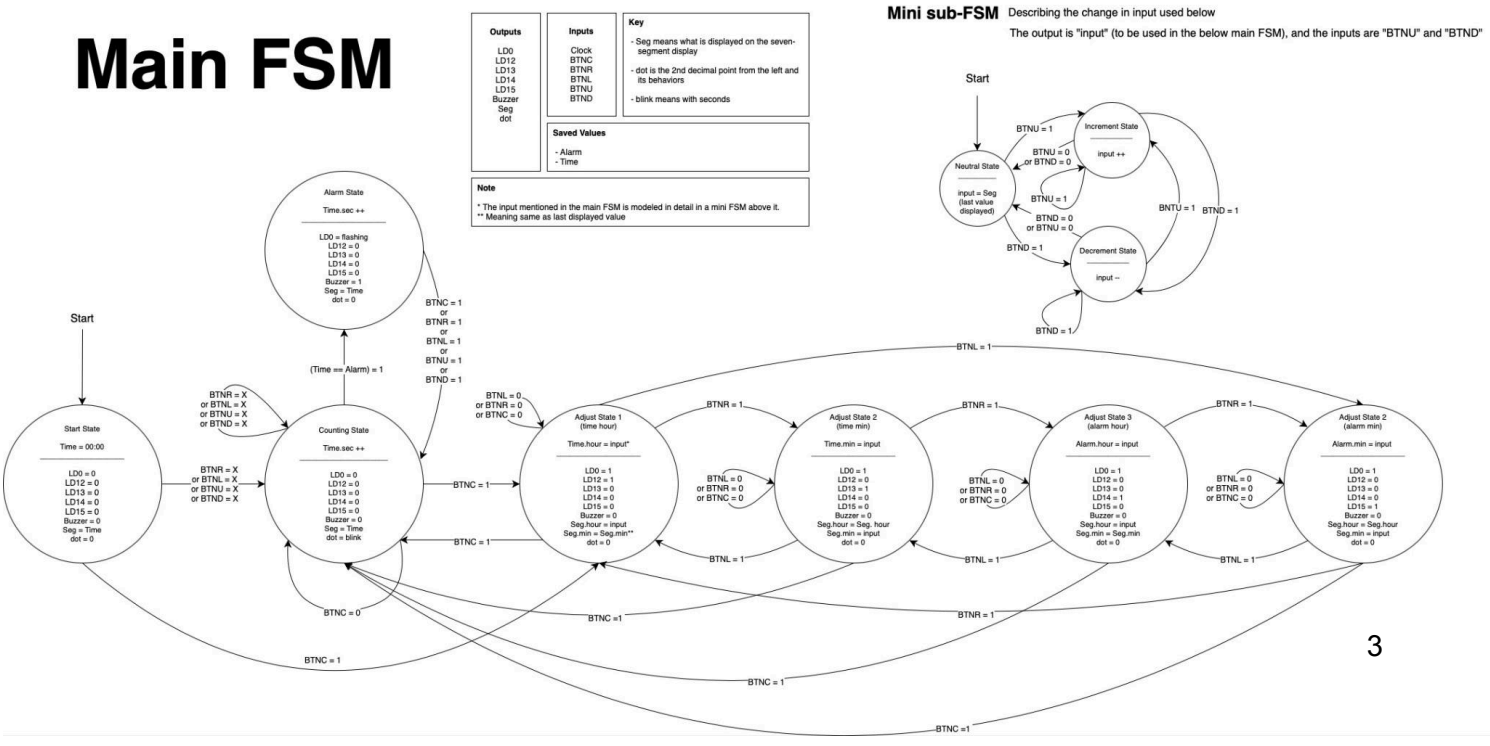
- Seg means what is displayed on the seven-segment display
- dot is the 2nd decimal point from the left and its behaviors
- blink means with seconds

**Saved Values**

- Alarm
- Time

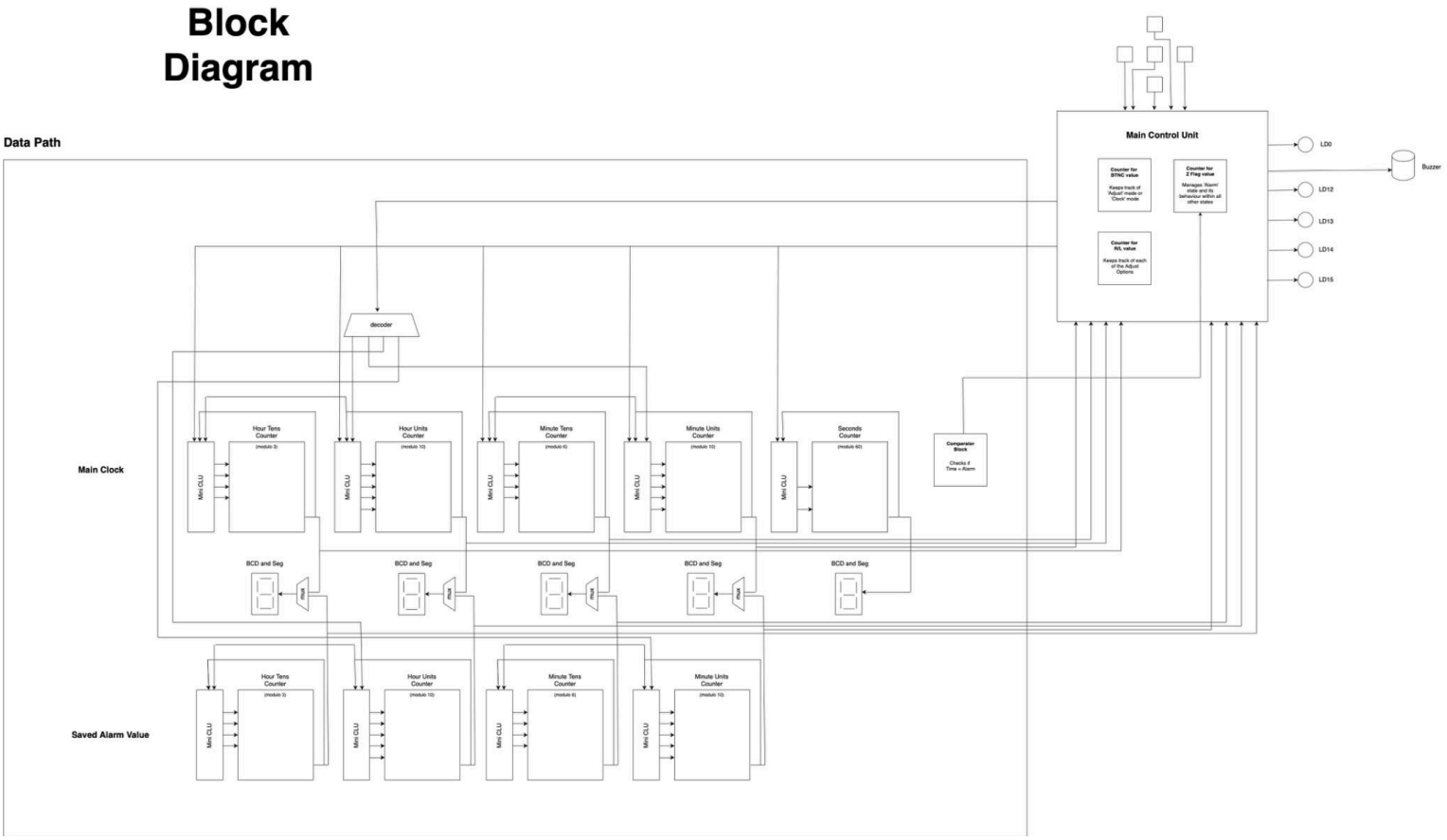
**Note**

* The input mentioned in the main FSM is modeled in detail in a mini FSM above it.
** Meaning same as last displayed value



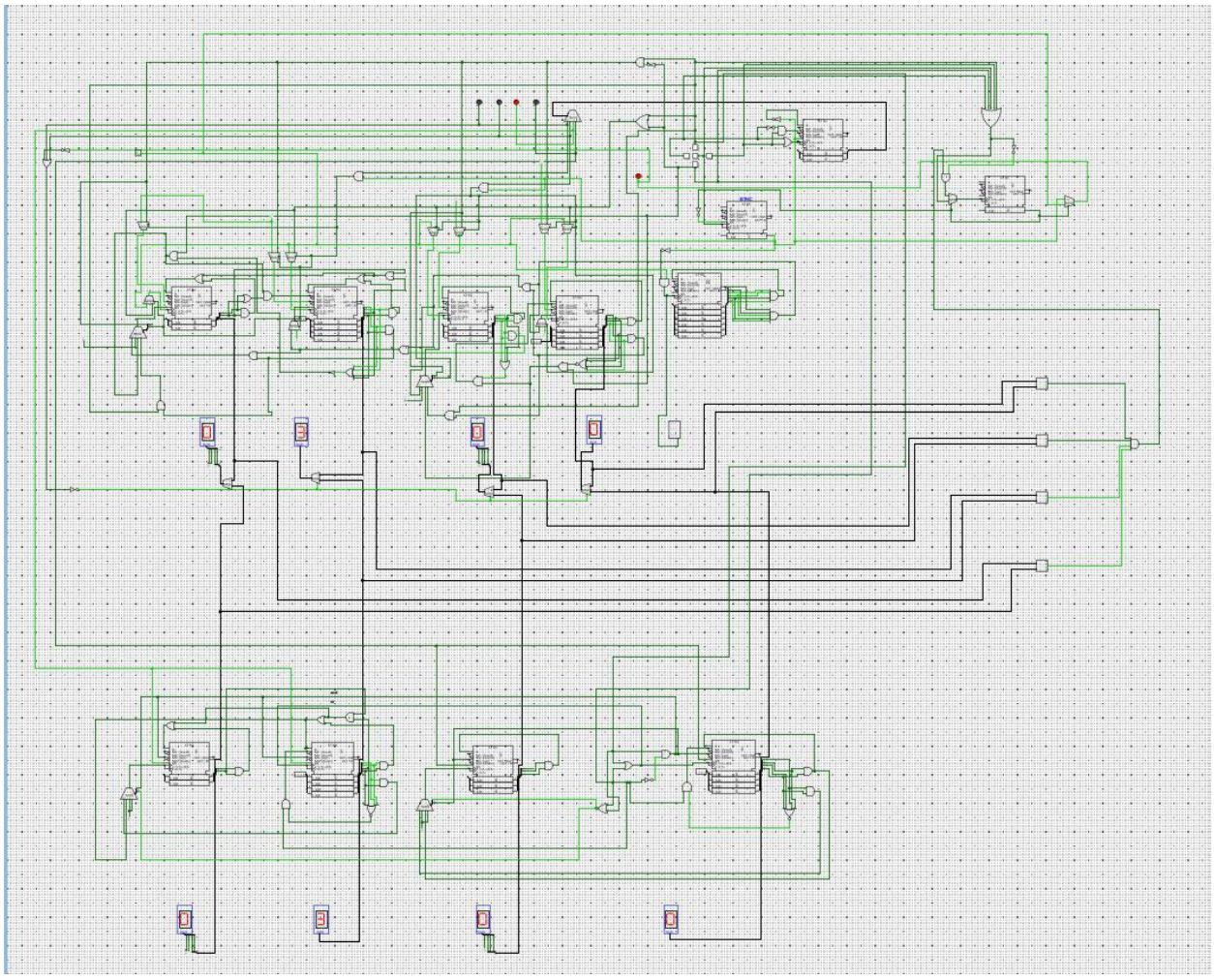
## Block Diagram

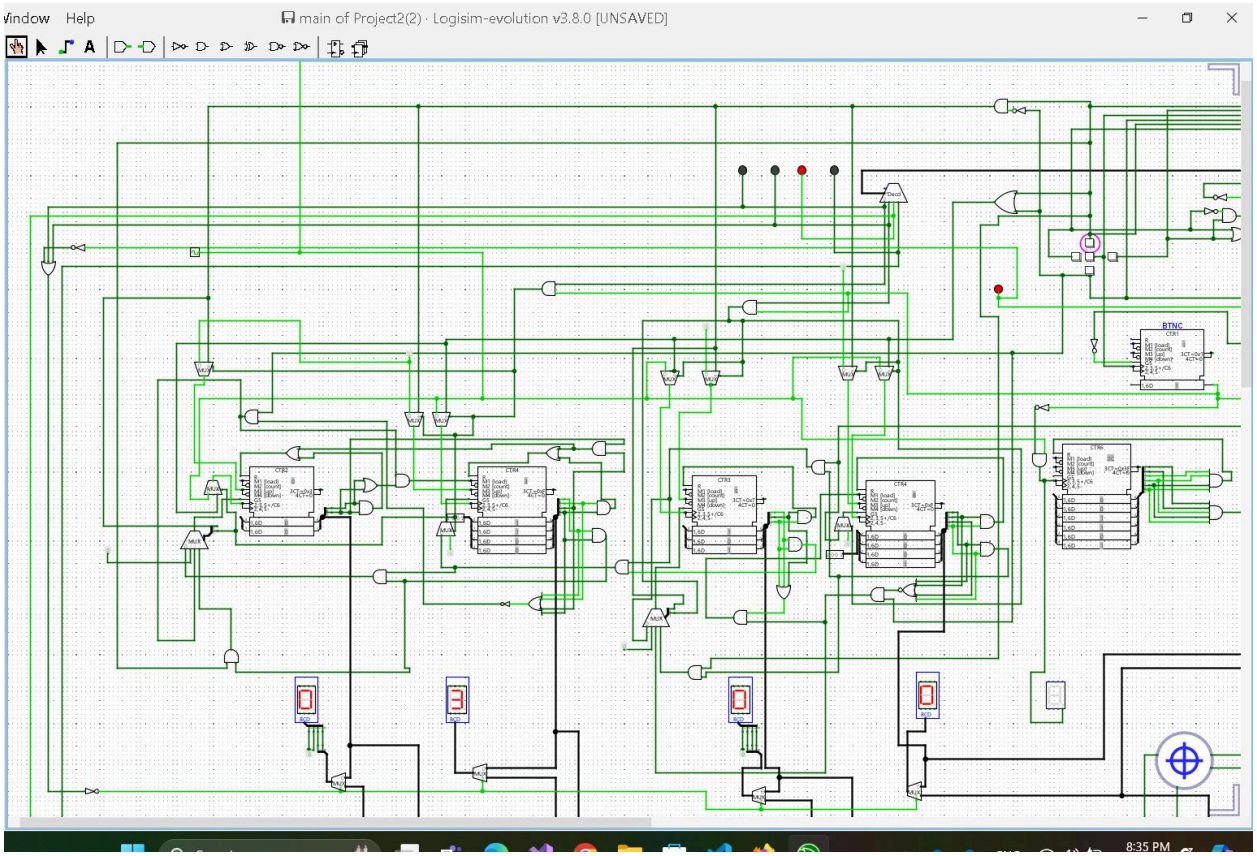
The block diagram is a simplified version of Logism, showing clearly the data path and the control unit. In our design, we have a main control unit with the buttons as input and the buzzer and LEDs (LD0, LD12, LD13, LD14, LD15) as outputs. Each counter is also controlled by a mini control unit that organizes the main signals and gives them to each respective counter, making the datapath. Other than the counters and display, the datapath has a decoder for the different adjust modes and a comparator for the alarm state.

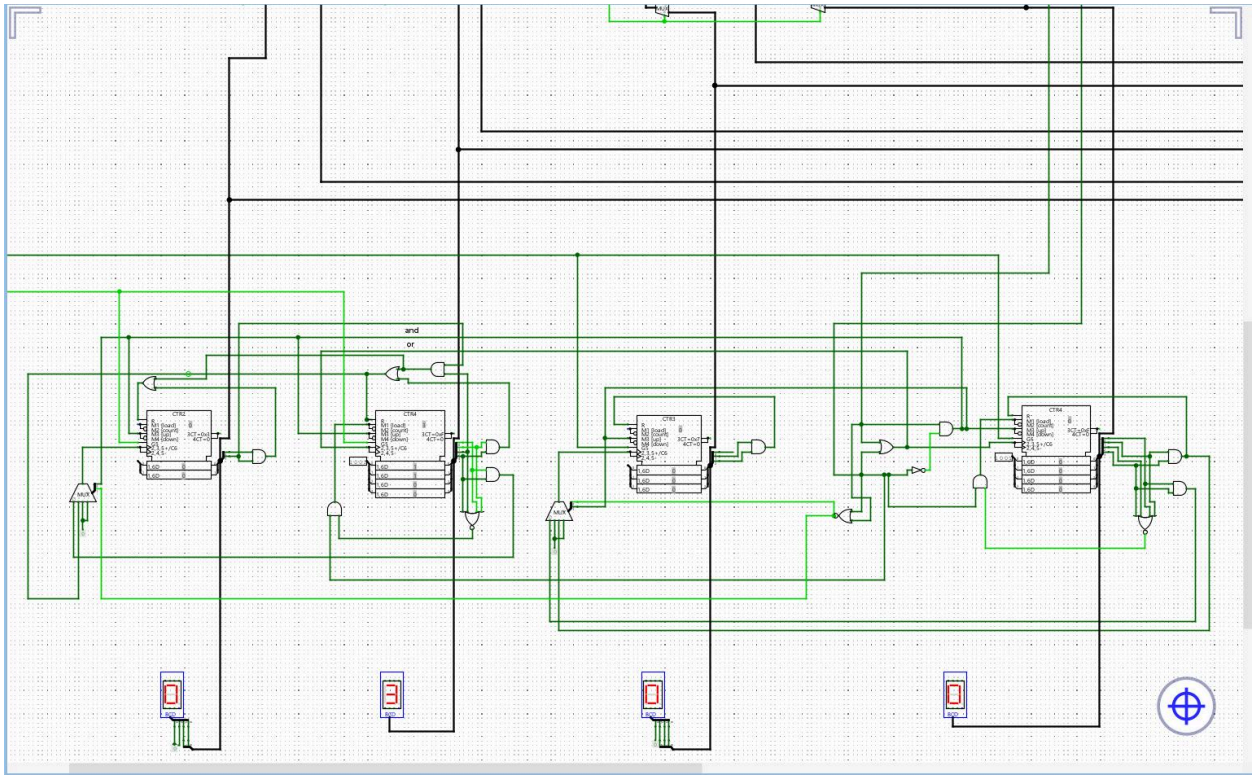


# Logisim

In Logism, we turned the concept into its basic hardware components and how they would be implemented and connected in detail. We spent a great deal of time debugging every component to make sure the display and design worked as intended, which in turn allowed us to learn a lot about how to turn concepts into hardware and apply topics learned in class. Its main components are the counters and the various multiplexers that control how the counter operates to increment and decrement properly.




Window Help
main of Project2(2) · Logisim-evolution v3.8.0 [UNSAVED]
BTNC
8:35 PM


and
or

# Modules

In this section, we provide a detailed description of the various modules implemented in our Digital Alarm Clock project, following the Moore design approach for Finite State Machine (FSM) implementation. Each module was meticulously designed to fulfill a specific function within the overall system, contributing to the seamless operation of the digital alarm clock. The descriptions below outline the purpose, functionality, and key implementation details of each module, highlighting the modifications and enhancements made to meet the project's requirements.

### Clock Divider

**File:** ClockDiv.v

The Clock Divider module is responsible for generating slower clock signals from the main clock. This is essential for synchronizing the various components of the digital alarm clock to operate at the correct timing intervals.

```
`timescale 1ns / 1ps

module ClockDiv
#(parameter n = 50000000) //10000
(input clk, reset, 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 reset) begin
if (reset == 1'b1) // Asynchronous Reset  count <= 32'b0;
      count <= 32'b0;

else if (count == n-1)
      count <= 32'b0;
else
      count <= count + 1;
end
// Handle the output clock
always @ (posedge clk, posedge reset) begin
if (reset) // Asynchronous Reset
      clk_out <= 0;
else if (count == n-1)
      clk_out <= ~ clk_out;
```

```
end

endmodule
```

### Debouncer

**File:** Debouncer. v

The Debouncer module handles switch debounce to ensure that only valid presses of the buttons are registered. This module is crucial for reliable user input.

```
`timescale 1ns / 1ps

module Debouncer(input clk, rst, in, output out);

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
```

### Rising Edge Detection

**File:** risingEdgeDet.v

The Rising Edge Detection module detects the rising edge of signals, which is important for detecting button presses and other events.

```
`timescale 1ns / 1ps

module risingEdgeDet(input clk, rst, w, output z);
reg [1:0] state, nextState;
parameter [1:0] A=2'b00, B=2'b01, C=2'b10;
// Next state generation (combinational logic)
always @ (w or state)
case (state)
A: if (w==0) nextState = A;
 else nextState = B;
B: if (w==0) nextState = A;
 else nextState = C;
C: if (w==0) nextState = A;
 else nextState = C;
default: nextState = A;
endcase
// State register
// Update state FF's with the triggering edge of the clock
always @ (posedge clk or posedge rst) begin
if(rst) state <= A;
else state <= nextState;
end
// output generation (combinational logic)
assign z = (state == B);
endmodule
```

### Synchronizer

**File:** Synchronizer. v

The Synchronizer module ensures that asynchronous signals are synchronized with the system clock to avoid timing issues.

```
`timescale 1ns / 1ps

module Synchronizer(
input  clk,
input  SIG,
output reg SIG1
```

```
);

      reg meta;


      always @(posedge clk) begin
      meta <= SIG ;
      SIG1 <= meta;
      end

endmodule
```

### Push Button Detection

**File:** PushButtonDet.v

The Push Button Detection module identifies when a push button is pressed and processes the input for further action.

```
`timescale 1ns / 1ps

module PushButtonDet(input clk, rst, in, output z);

wire out;
wire SIG1;

Debouncer DUT_debouncer(clk, rst, in,  out);
Synchronizer DUT_synch(clk, out, SIG1);
risingEdgeDet DUT_edge_det( clk, rst, SIG1,z);

endmodule
```

### N-Bit Counter

**File:** NBitCounter.v

The N-Bit Counter module is a versatile counter designed for various counting tasks within the digital alarm clock. This module extends the functionality of a standard counter by including an

UpDown flag, which determines whether the counter increments or decrements. This feature is particularly useful for implementing the clock adjustment functionality using the BTNU (Button Up) and BTND (Button Down) inputs.

```
`timescale 1ns / 1ps

module NBitCounter #(parameter n = 4, x = 2)(input clk, reset, en, updown,
output reg [(x-1):0] count);

      always @(posedge clk, posedge reset)begin
      if (reset == 1)
            count<= 0;
      else begin
            if (en == 1)begin
                  if(updown ==1) begin
                  if (count < n - 1)
                        count <= count + 1;
                  else if (count == n - 1)
                        count<= 0;
                  end
                  else begin
                  if (count > 0)
                        count <= count - 1;
                  else if( count == 0)
                        count <= n-1;
                  end
            end
            else
                  count <= count;
      end
   end

endmodule
```

### Seven Segment Display

**File:** SevenSegment.v

The Seven Segment Display module drives the 7-segment display, converting binary values into the corresponding segments to be lit. This module is a modified version of the standard

seven-segment display driver, with additional functionality to handle the dot (second dot from the left) that represents the seconds.

```
`timescale 1ns / 1ps

module SevenSegment(
input [3:0] num,
input [1:0] sel,
input en_clk,
input Dot,
output reg [7:0] display,
output reg [3:0] anode_active);

always @(sel) begin
      case (sel)
            2'b00:begin
                  display[0] = 1'b1; // changed b0
                  anode_active <= 4'b1110;
            end
            2'b01: begin
                  display[0] = 1'b1;
                  anode_active <= 4'b1101;
            end
            2'b10: begin
                  display[0] = en_clk ? Dot : 1'b1;
                  anode_active <= 4'b1011;
            end
            2'b11: begin
                  display[0] = 1'b1;
                  anode_active <= 4'b0111;
            end
      endcase
case (num)

      0: display[7:1] = 7'b0000001;
      1: display[7:1] = 7'b1001111;
      2: display[7:1] = 7'b0010010;
      3: display[7:1] = 7'b0000110;
      4: display[7:1] = 7'b1001100;
      5: display[7:1] = 7'b0100100;
      6: display[7:1] = 7'b0100000;
      7: display[7:1] = 7'b0001111;
      8: display[7:1] = 7'b0000000;
```

```
      9: display[7:1] = 7'b0001100;

endcase
end
endmodule
```

### Hour and Minute Counter

**File:** HourMinCounter.v

The Hour and Minute Counter module manages the counting of hours and minutes. This module has been modified to take different enable signals and decide which enable to use based on the current mode. This functionality is crucial for switching between clock/alarm mode and adjust mode, allowing for seamless time adjustments.

```
`timescale 1ns / 1ps

module HourMinCounter
(   input clk,
      input reset,
      input en,
      input enHours,
      input enMins,
      input updown,
      output [5:0] seconds,
      output [3:0] minutes_units,
      output [2:0] minutes_tens,
      output [3:0] hours_units,
      output [1:0] hours_tens
);

wire [5:0] minutes;
wire [4:0] hours;
wire enhours;
wire enmins;

assign enhours = en ? seconds == 59 & minutes == 59: enHours;
assign enmins = en ? seconds == 59 : enMins;
```

```
NBitCounter #(60, 6) DUT0 ( .clk( clk) , .reset(reset),
.en(en),.updown(1'b1), .count(seconds));

NBitCounter #(60, 6) DUT2 ( .clk(clk) , .reset(reset),
.en(enmins),.updown(updown), .count(minutes));

NBitCounter #(24, 5) DUT4 ( .clk(clk) , .reset(reset),
.en(enhours),.updown(updown), .count(hours));

assign minutes_units = minutes % 10;
assign minutes_tens = minutes / 10;

assign hours_units = hours % 10;
assign hours_tens = hours / 10;

endmodule
```

### Multiplexed Output

**File:** MultiplexedOutput.v

The Multiplexed Output module controls the 7-segment display multiplexing, allowing multiple digits to be displayed using a single set of 7-segment LEDs. This module has been modified to receive both the digits of the current time and the digits of the saved alarm, displaying them as needed based on the mode. This ensures that either the current time or the alarm time is shown, depending on the logic.

```
`timescale 1ns / 1ps

module MultiplexedOutput(
input en_clk,
input clk,
input clk_seconds,
input rst,
input [3:0] minutes_units,
input [2:0] minutes_tens,
input [3:0] hours_units,
input [1:0] hours_tens,
output [3:0] anode_active,
```

```
output [7:0] segments,
input [2:0] state,
input [3:0] minutes_units_A,
input [2:0] minutes_tens_A,
input [3:0] hours_units_A,
input [2:0] hours_tens_A
);


 reg [3:0] mux_output;
 wire [1:0] sel;

 NBitCounter #(.n(4)) Counter (.clk(clk), .reset(rst), .en(1'b1),
.updown(1'b1), .count(sel));



      always @(sel)begin

      case(sel)
            2'b00: mux_output = (state==3'b101 | state==3'b100)?
minutes_units_A : minutes_units ;
            2'b01: mux_output = (state==3'b101 | state==3'b100)?
minutes_tens_A : minutes_tens;
            2'b10: mux_output = (state==3'b101 | state==3'b100)?
hours_units_A : hours_units;
            2'b11: mux_output = (state==3'b101 | state==3'b100)?
hours_tens_A : hours_tens;
      endcase
      end


      // Instantiate 7segment
      SevenSegment segment_decoder (
      .sel(sel),
      .num(mux_output),
      .en_clk(en_clk),
      .Dot(clk_seconds),
      .display(segments),
      .anode_active(anode_active)
      );
```

```
endmodule
```

### Digital Alarm Clock

**Files:** DigitalAlarmClock.v, DigitalAlarmClock_C.v

The main Digital Alarm Clock module integrates all other modules and manages the core functionalities of the alarm clock, including timekeeping, alarm setting, and triggering the alarm. This module follows the Moore design approach in its FSM (Finite State Machine) implementation for switching between different states. The use of the Moore design ensures that the outputs depend only on the current state, providing stable and predictable behavior.

```
`timescale 1ns / 1ps

module DigitalAlarmClock(
      input clk,
      input rst,
      input en,
      input  [4:0] BTN,
      output [7:0] segments,
      output [3:0] anode_active,
      output reg [4:0] LED,
      output reg buzzer
);

reg [2:0] state, nextState;
parameter [2:0] A=3'b000, B=3'b001, C=3'b010, D= 3'b011, E=3'b100,
F=3'b101; // States Encoding
wire [4:0] BTN_Det;
wire [5:0] seconds;
wire [3:0] minutes_units;
wire [2:0] minutes_tens;
wire [3:0] hours_units;
wire [1:0] hours_tens;
wire [3:0] minutes_units_A;
wire [2:0] minutes_tens_A;
wire [3:0] hours_units_A;
wire [1:0] hours_tens_A;
wire clk_200, clk_1;
reg updown;
```

```
reg enAdjTimeMin;
reg enAdjTimeHours;
reg enAdjAlarmMin;
reg enAdjAlarmHours;
reg inputclock;
reg en_sec;
wire BTNC;

genvar i;
generate
      for(i = 0 ; i  < 5 ; i = i + 1)
      PushButtonDet BTN(clk_200, rst, BTN[i],BTN_Det[i]);
endgenerate

//PushButtonDet BTN(clk_200, rst, BTN[0],BTNC);
// assign LED[0] = BTNC;

ClockDiv #(250000) Div1 (clk, rst, clk_200); //for anode

ClockDiv Div2 (clk, rst, clk_1); //for seconds

HourMinCounter clock (.clk(inputclock), .reset(rst),
.en(en_sec),.enHours(enAdjTimeHours),
.enMins(enAdjTimeMin),.updown(updown),
.seconds(seconds),.minutes_units(minutes_units),
.minutes_tens(minutes_tens),
.hours_units(hours_units),.hours_tens(hours_tens));

HourMinCounter alarm (.clk(inputclock), .reset(rst),
.en(1'b0),.enHours(enAdjAlarmHours),
.enMins(enAdjAlarmMin),.updown(updown),
.seconds(),.minutes_units(minutes_units_A), .minutes_tens(minutes_tens_A),
.hours_units(hours_units_A),.hours_tens(hours_tens_A)); // what will the
clk be? ^2

MultiplexedOutput out (.en_clk(en_sec), .clk(clk_200), .clk_seconds(clk_1),
.rst(rst), .minutes_units(minutes_units), .minutes_tens(minutes_tens),
.hours_units(hours_units), .hours_tens(hours_tens),
.anode_active(anode_active), .segments(segments), .state(state),
.minutes_units_A(minutes_units_A), .minutes_tens_A(minutes_tens_A),
.hours_units_A(hours_units_A), .hours_tens_A(hours_tens_A)) ;
```

```
// Next state generation (combinational logic)

always @ (BTN_Det or state)
case (state)

A:begin // clock state
      if (BTN_Det == 5'b00001) nextState = C; // BTNC
      else if (minutes_units == minutes_units_A && minutes_tens ==
minutes_tens_A
      && hours_units == hours_units_A && hours_tens == hours_tens_A &&
seconds == 0 ) nextState = B; // Alarm
      else begin
      nextState = A;
      LED = 5'b00000;
      inputclock = clk_1;
      en_sec = 1'b1;
      buzzer = 0;
      updown = 1'b1;
      enAdjAlarmHours = 1'b0;
      enAdjAlarmMin = 1'b0;
      end
end

B:  begin // Alarm
      if (BTN_Det == 5'b00001)
            begin nextState = A;// BTNC
            end
      else if(BTN_Det == 5'b00010)
            begin nextState = A;// BTNC
            end
      else if(BTN_Det == 5'b00100)
            begin nextState = A;// BTNC
            end
      else if(BTN_Det == 5'b01000)
            begin nextState = A;// BTNC
            end
      else if(BTN_Det == 5'b10000)
            begin nextState = A;// BTNC
            end
      else begin
            LED[0] = clk_1;
            LED[4:1] = 4'b0000;
            inputclock = clk_1;
```

```
            buzzer = clk_1;
            en_sec = 1'b1;
            updown = 1'b1;
            enAdjAlarmHours = 1'b0;
            enAdjAlarmMin = 1'b0;
            nextState = B;
      end
      end

C:  begin // Adjust clock hours

      if(BTN_Det == 5'b00001)
            begin nextState = A;// BTNC
            end
      else if(BTN_Det == 5'b00100) nextState = F; // BTNL
      else if (BTN_Det == 5'b00010) nextState = D; // BTNR

      else if (BTN_Det == 5'b01000) begin // BTNU
            inputclock = clk_200;
            updown = 1'b1;
            enAdjTimeHours = 1'b1;
      end
      else if (BTN_Det == 5'b10000) begin // BTND
            inputclock = clk_200;
            updown = 1'b0;
            enAdjTimeHours = 1'b1;
      end
      else begin// else
            en_sec = 1'b0;
            updown = 1'b1;
            enAdjTimeHours = 1'b0;
            enAdjAlarmHours = 1'b0;
            enAdjAlarmMin = 1'b0;
            inputclock = clk_200;
            LED = 5'b10001;
            buzzer = 0;
            nextState = C;
      end
      end

D:  begin

      if(BTN_Det == 5'b00001)
```

```
            begin nextState = A;// BTNC
            end
      else if(BTN_Det == 5'b00100) nextState = C; // BTNL
      else if (BTN_Det == 5'b00010) nextState = E; // BTNR

      else if (BTN_Det == 5'b01000) begin // BTNU
            inputclock = clk_200;
            updown = 1'b1;
            enAdjTimeMin = 1'b1;
      end
      else if (BTN_Det == 5'b10000) begin // BTND
            inputclock = clk_200;
            updown = 1'b0;
            enAdjTimeMin = 1'b1;
      end
      else begin// else
            en_sec = 1'b0;
            updown = 1'b1;
            enAdjTimeMin = 1'b0;
            enAdjAlarmHours = 1'b0;
            enAdjAlarmMin = 1'b0;
            inputclock = clk_200;
            LED = 5'b01001;
            buzzer = 0;
            nextState = D;
      end
      end

E:  begin

      if(BTN_Det == 5'b00001)
            begin nextState = A;// BTNC
            end
      else if(BTN_Det == 5'b00100) nextState = D; // BTNL
      else if (BTN_Det == 5'b00010) nextState = F; // BTNR

      else if (BTN_Det == 5'b01000) begin // BTNU
            inputclock = clk_200;
            updown = 1'b1;
            enAdjAlarmHours = 1'b1;
      end
      else if (BTN_Det == 5'b10000) begin // BTND
            inputclock = clk_200;
```

```
            updown = 1'b0;
            enAdjAlarmHours = 1'b1;
      end
      else begin// else
            en_sec = 1'b0;
            updown = 1'b1;
            enAdjAlarmHours = 1'b0;
            enAdjAlarmMin = 1'b0;
            inputclock = clk_200;
            LED = 5'b00101;
            buzzer = 0;
            nextState = E;
      end
      end


F:  begin

      if(BTN_Det == 5'b00001)
            begin nextState = A;// BTNC
            end
      else if(BTN_Det == 5'b00100) nextState = E; // BTNL
      else if (BTN_Det == 5'b00010) nextState = C; // BTNR

      else if (BTN_Det == 5'b01000) begin // BTNU
            inputclock = clk_200;
            updown = 1'b1;
            enAdjAlarmMin = 1'b1;
      end
      else if (BTN_Det == 5'b10000) begin // BTND
            inputclock = clk_200;
            updown = 1'b0;
            enAdjAlarmMin = 1'b1;
      end
      else begin// else
            en_sec = 1'b0;
            updown = 1'b1;
            enAdjAlarmMin = 1'b0;
            enAdjAlarmHours = 1'b0;
            inputclock = clk_200;
            LED = 5'b00011;
            buzzer = 0;
            nextState = F;
```

```
      end
      end


   default: nextState = A;

endcase


// State register
// Update state FF's with the triggering edge of the clock

// output generation (combinational logic)
// State Changer on clock edge
always @ (posedge clk_200, posedge rst) begin
      if(rst)begin
      state <= A;
      //buzzer logic
      end
      else
      state <= nextState;
end

endmodule
```

### Digital Alarm Clock Constraints

**File:** DigitalAlarmClock_C.xdc

The Digital Alarm Clock Constraints file defines the physical constraints for the design, such as pin assignments and timing constraints. This file is crucial for ensuring that the design operates correctly on the BASYS3 FPGA board. Additionally, it includes the bonus part of the project, which implements a buzzer using the BASYS3 PMOD ports for the first time, as requested by the professor.

```
set_property PACKAGE_PIN W5 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
```

```
set_property PACKAGE_PIN V17 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports rst]

set_property PACKAGE_PIN V16 [get_ports en]
set_property IOSTANDARD LVCMOS33 [get_ports en]

set_property PACKAGE_PIN U18 [get_ports {BTN[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {BTN[0]}]

set_property PACKAGE_PIN T17 [get_ports {BTN[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {BTN[1]}]

set_property PACKAGE_PIN W19 [get_ports {BTN[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {BTN[2]}]

set_property PACKAGE_PIN T18 [get_ports {BTN[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {BTN[3]}]

set_property PACKAGE_PIN U17 [get_ports {BTN[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {BTN[4]}]

set_property PACKAGE_PIN U16 [get_ports {LED[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[0]}]

set_property PACKAGE_PIN P3 [get_ports {LED[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[1]}]

set_property PACKAGE_PIN N3 [get_ports {LED[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[2]}]

set_property PACKAGE_PIN P1 [get_ports {LED[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[3]}]

set_property PACKAGE_PIN L1 [get_ports {LED[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[4]}]

set_property PACKAGE_PIN V7 [get_ports {segments[0]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {segments[0]}]

set_property PACKAGE_PIN W7 [get_ports {segments[7]}]
```

```
    set_property IOSTANDARD LVCMOS33 [get_ports {segments[7]}]

set_property PACKAGE_PIN  W6 [get_ports {segments[6]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {segments[6]}]

set_property PACKAGE_PIN  U8 [get_ports {segments[5]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {segments[5]}]

set_property PACKAGE_PIN  V8 [get_ports {segments[4]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {segments[4]}]

set_property PACKAGE_PIN  U5 [get_ports {segments[3]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {segments[3]}]

set_property PACKAGE_PIN  V5 [get_ports {segments[2]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {segments[2]}]

set_property PACKAGE_PIN  U7 [get_ports {segments[1]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {segments[1]}]

set_property PACKAGE_PIN  U2 [get_ports {anode_active[0]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {anode_active[0]}]

set_property PACKAGE_PIN  U4 [get_ports {anode_active[1]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {anode_active[1]}]

set_property PACKAGE_PIN  V4 [get_ports {anode_active[2]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {anode_active[2]}]

set_property PACKAGE_PIN  W4 [get_ports {anode_active[3]}]

    set_property IOSTANDARD LVCMOS33 [get_ports {anode_active[3]}]

set_property PACKAGE_PIN  M18 [get_ports buzzer]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports buzzer]
set_property PULLUP TRUE [get_ports buzzer]
```

---

# Contributions

## Adham Gohar

- **Debouncer Module:**
  - Engineered to stabilize input signals and manage overall system integration using a Moore finite state machine (FSM).
- **Synchronizer Module:**
  - Aligned asynchronous signals with a system clock, including enhancements like a buzzer for alerts using PMOD ports.
- **MultiplexedOutput Module:**
  - This module controls the routing of output signals to the display, optimizing visual output management.
- **Diagrams:**
  - Worked on the ASM design diagram.
  - Contributed to the FSM design diagram.
  - Contributed to the Block Diagram.
- **Logisim Evolution Design:**
  - Designed the initial clock with adjusting logic.

## Toqa Hassanein

- **N-Bit Counter:**
  - This module created a flexible counter for tracking time increments or decrements based on user button presses.
- **Seven Segment Display:**
  - This module refined the display logic to include a seconds indicator, ensuring clear and accurate time representation.
- **Rising Edge Detection:**
  - Developed to capture the initial moments of signal changes, crucial for timing accuracy.
- **Diagrams:**
  - Worked on the Block Diagram of the project.

    - Contributed to the FSM design diagram.
    - Contributed to the ASM design diagram.
- **Logisim Evolution Design:**
    - Added the alarm functionality to the initial design.

## Khadeejah Iraky

- **Hour and Minute Counter:**
    - Created this module to keep track of time progression, with additional modes for setting and alarm functionality.
- **ClockDiv Module:**
    - Adjusted to support dual-mode display of current time and alarm settings, ensuring smooth transitions between displays.
- **Push Button Detection:**
    - Implemented to accurately detect user interactions while managing signal noise and debounce.
- **Diagrams:**
    - Worked on the FSM design diagram.
    - Contributed to the ASM design diagram.
    - Contributed to the Block Diagram of the project
- **Logisim Evolution Design:**
    - Integrated the multiplexing functionality into the final design.

**DigitalAlarmClock Top Module Integration**

We worked to integrate the modules we developed individually into the **DigitalAlarmClock** top module. This collaboration was crucial in combining the various functionalities into a cohesive system. We then created the constraints specified in the **DigitalAlarmClock_C.xdc** file to ensure the entire Digital Alarm Clock ran effectively on the FPGA, addressing specific hardware limitations and performance requirements.

---

## Challenges Encountered

**Integration Challenges**: We encountered difficulties ensuring that the individually developed modules worked perfectly when combined into the DigitalAlarmClock top module. Aligning the timing and functionality between these modules was hard initially.

**FPGA Implementation**: Running the integrated system on the FPGA introduced constraints related to hardware capabilities. We had to carefully manage resource allocation and optimize the system's performance to fit within the FPGA's limitations.

---

## Resources

- **FPGA Board Documentation**: We heavily relied on the Basys 3 Artix-7 FPGA Board Reference Manual for understanding the specific capabilities and constraints of our FPGA board, which helped in designing and implementing our project. Available at Basys 3 Reference Manual. https://digilent.com/reference/basys3/refmanual

---