The American University in Cairo

Digital Design I

CSCE2301

Department OF Computer Science and Engineering

Project 2: Digital Alarm Clock Project Report

Authors:
Ahmed Elkhodary (900213472)
Youssef Ibrahim (900212889)
Khaled Nana (900211952)

Supervised by:
Dr. Mohamed Shalan

SPRING 2024

Contents

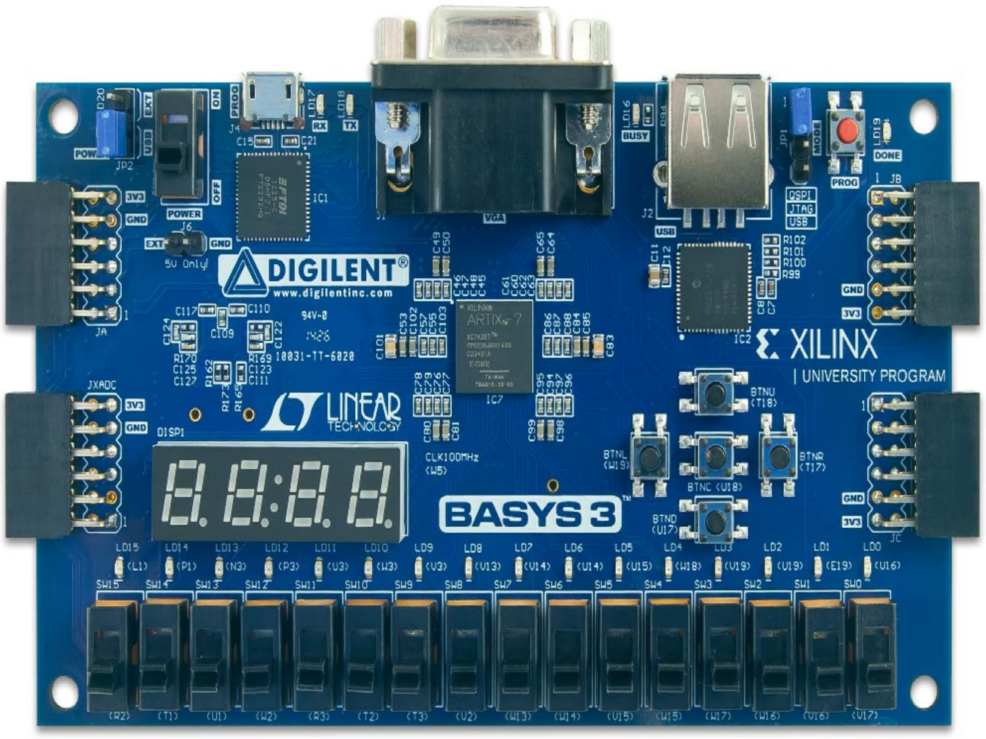


Figure 1: Basys 3 with AMD Artix 7 FPGA Board

1 Introduction

In this project, which is part of the Digital Design 1 course, we designed and implemented a simple digital alarm clock using the BASYS3 FPGA development board. The goal of the project was to create a functional alarm clock that utilizes the BASYS3 board's elements, including LEDs, the 7-segment display, and the push buttons. The digital alarm clock is designed to operate in two primary modes: "clock/alarm" and "adjust". The "clock/alarm" mode serves as the default operational state, displaying the current time and managing the alarm functionality. The "adjust" mode allows the user to set and modify the clock and alarm time settings. This project demonstrates the practical application of FPGA programming and digital circuit design, designing a fully operational digital alarm clock.

2 Individual Contributions

- **Khaled Nana:**
  - ASM Chart - 7.5%
  - Code Implementation - 25%
  - **Total contribution to the project: 32.5%**
- **Ahmed Elkhodary:**
  - Logisim Simulation - 15%
  - Project Report - 20%
  - **Total contribution to the project: 35%**
- **Youssef Ibrahim:**
  - Code Implementation - 25%
  - System Design (Control Unit and Datapath) - 7.5%
  - **Total contribution to the project: 32.5%**

3 System Design

The system design, which includes the Control path and the Data path can be seen in the following pages.

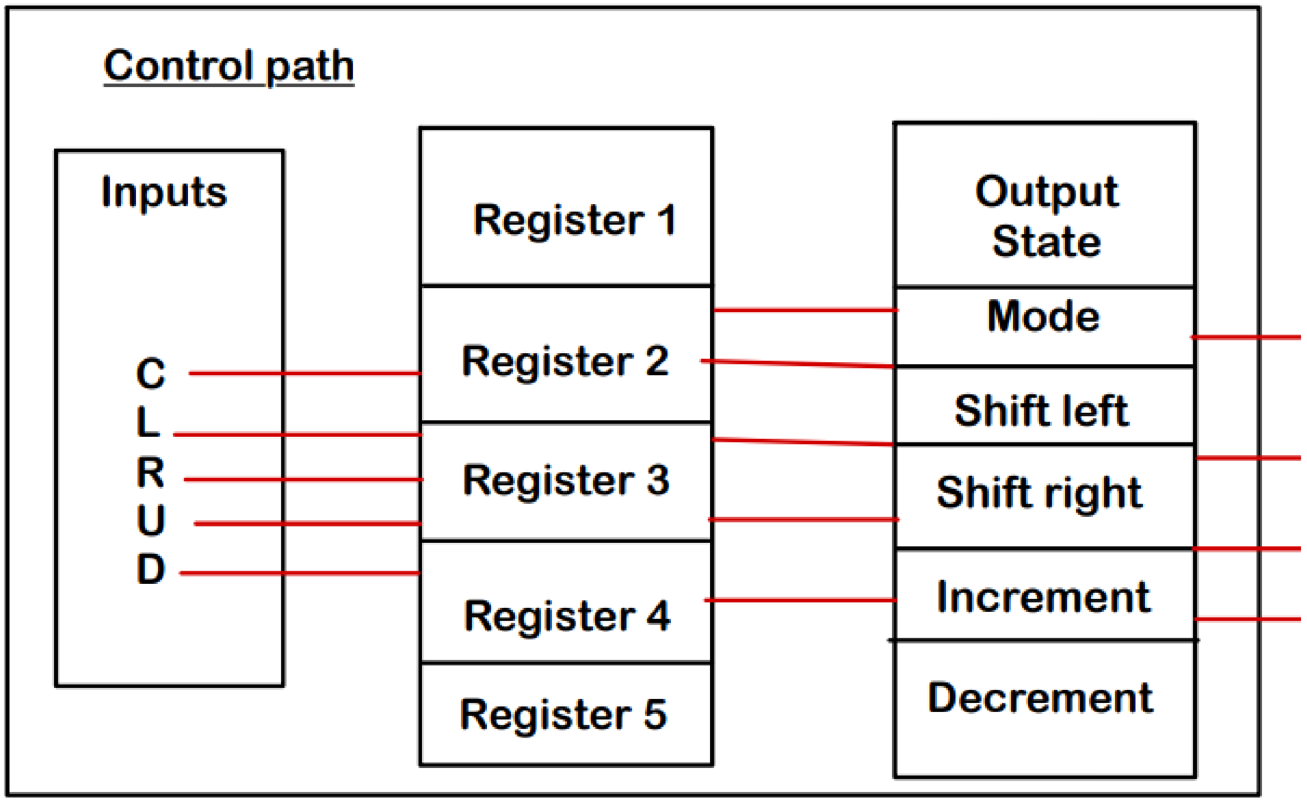


Figure 2: Control Path Diagram

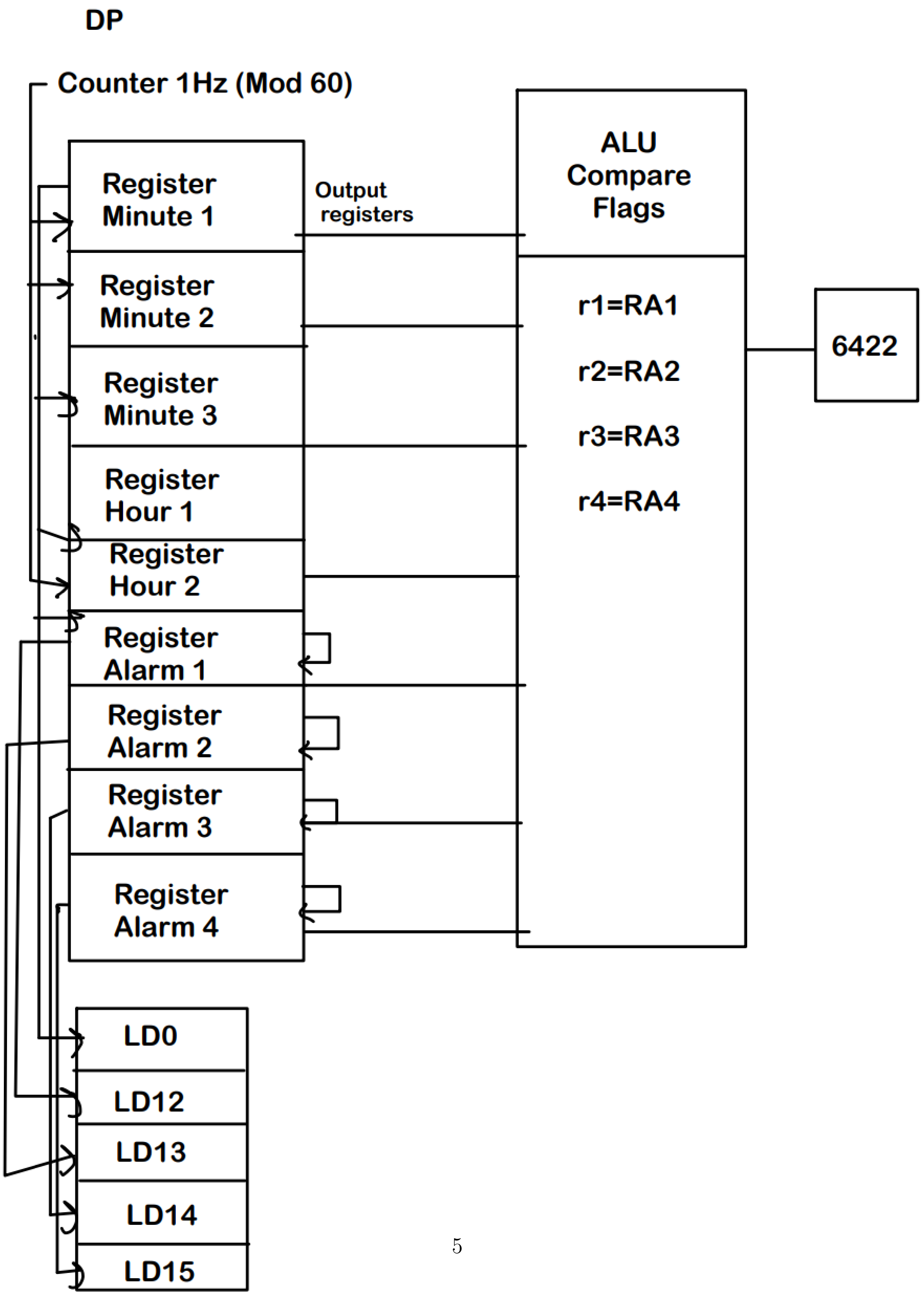


Figure 3: Enter Caption

3.1 Algorithmic State Machine (ASM) Chart

3.1.1 ASM Chart Part 1

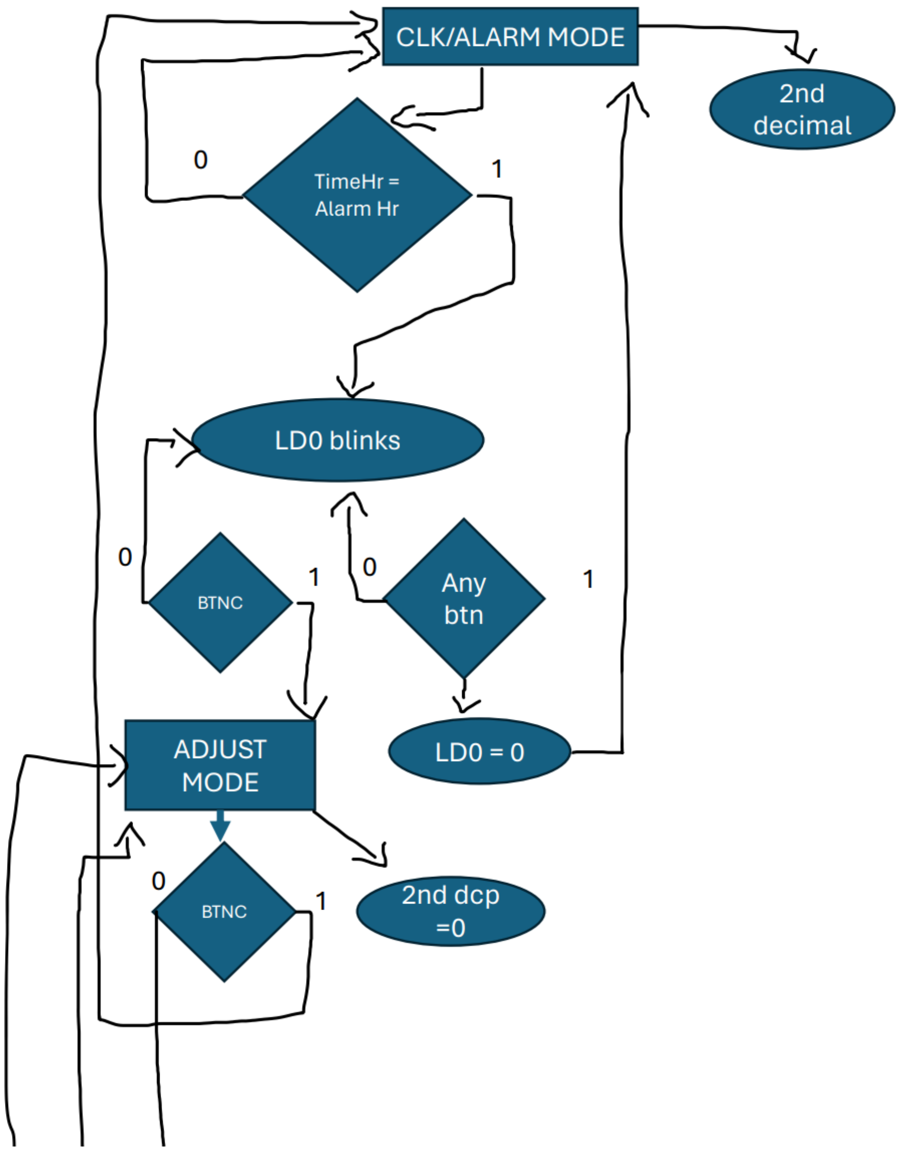


Figure 4: First ASM Chart

3.1.2 ASM Chart Part 2

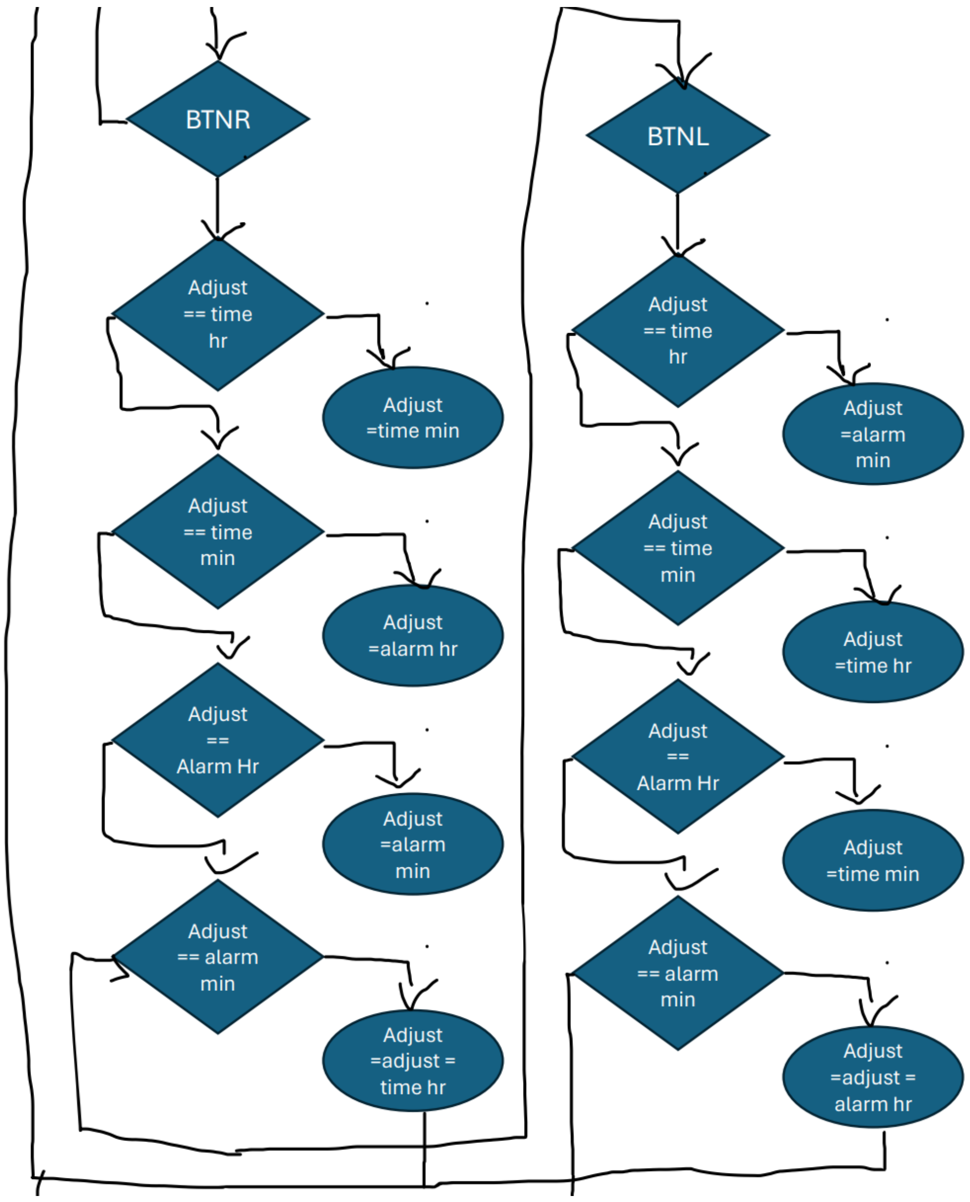


Figure 5: Second ASM Chart

3.1.3 ASM Chart Part 3

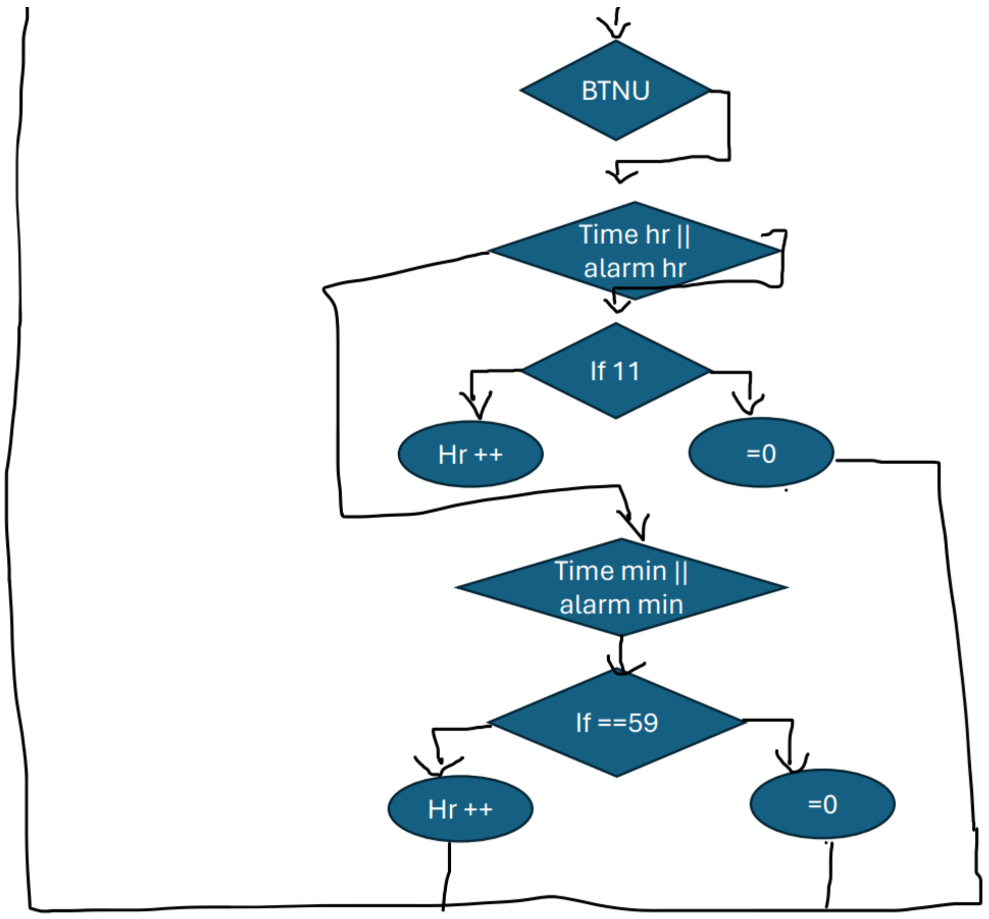


Figure 6: Third ASM Chart

4 Implementation in Logisim Evolution

We used a layered clock in Logisim that contains the 7-segment Display Driver, LED Decoder, and the 4-bit Binary Counter.

4.1 7-Segment Display Driver

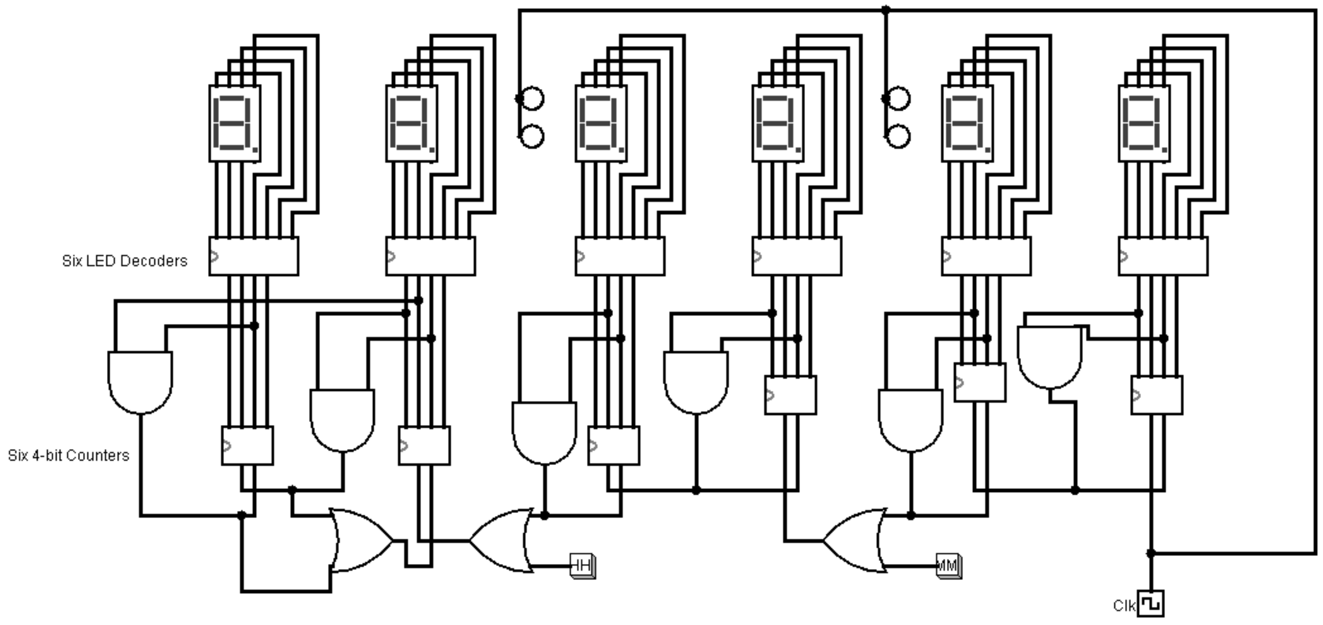
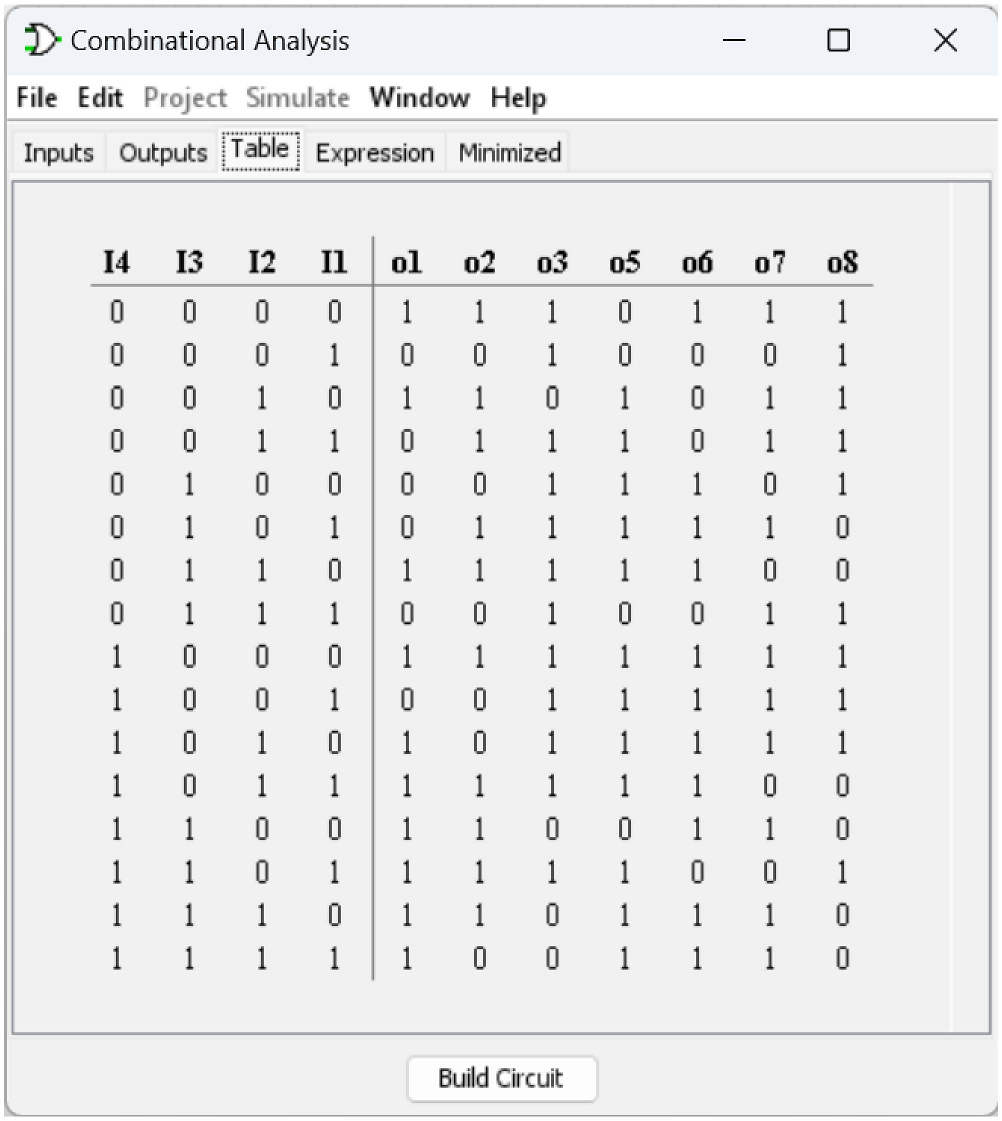


Figure 7: 7-segment Display Driver

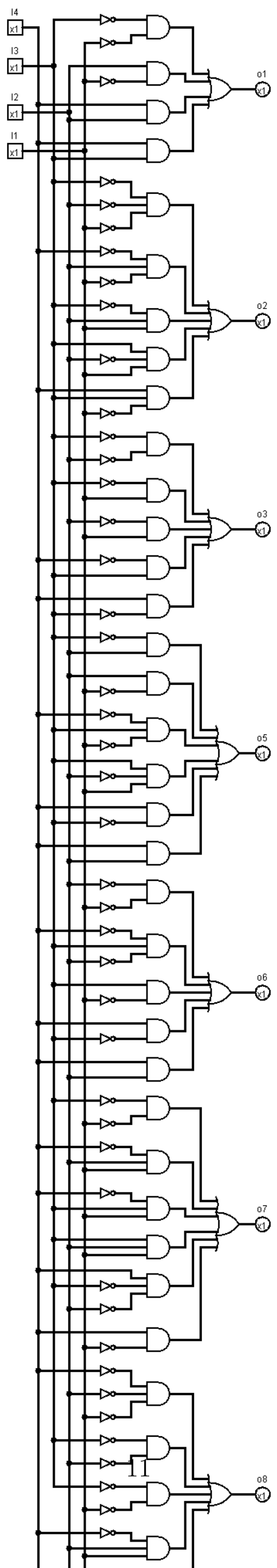

Combinational Analysis
File Edit Project Simulate Window Help
Inputs Outputs Table Expression Minimized
Build Circuit

I4	I3	I2	I1	o1	o2	o3	o5	o6	o7	o8
0	0	0	0	1	1	1	0	1	1	1
0	0	0	1	0	0	1	0	0	0	1
0	0	1	0	1	1	0	1	0	1	1
0	0	1	1	0	1	1	1	0	1	1
0	1	0	0	0	0	1	1	1	0	1
0	1	0	1	0	1	1	1	1	1	0
0	1	1	0	1	1	1	1	1	0	0
0	1	1	1	0	0	1	0	0	1	1
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	0	0	1	1	1	1	1
1	0	1	0	1	0	1	1	1	1	1
1	0	1	1	1	1	1	1	1	0	0
1	1	0	0	1	1	0	0	1	1	0
1	1	0	1	1	1	1	1	0	0	1
1	1	1	0	1	1	0	1	1	1	0
1	1	1	1	1	0	0	1	1	1	0

Figure 8: Truth Table for the LED Decoder

4.2 LED Decoder (BCD to 7-segment)

We simulated a circuit for the LED Decoder based on inputting the truth table (Figure 5), automatically generating the LED Decoder circuit (Figure 6) for us based on the logic that we used in the first part of the project.


I4
x1
I3
x1
I2
x1
I1
x1
o1
x1
o2
x1
o3
x1
o5
x1
o6
x1
o7
x1
o8
x1

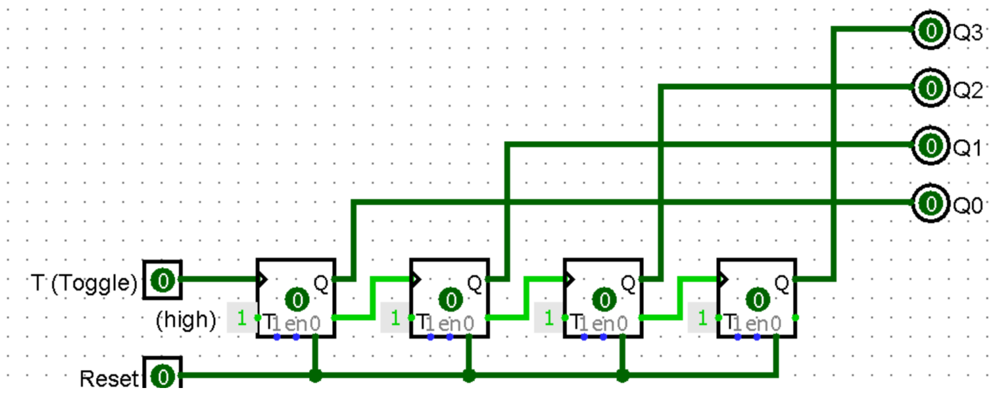


Figure 10: 4-bit Binary Counter

4.3 4-bit Binary Counter

Figure 7 outlined above is a simulated diagram of a 4-bit binary counter circuit, which is built using T flip-flops. Each T flip-flop in this setup represents a single bit of the binary counter. The counter's binary value increases by 1 each time the T input receives a clock pulse. The Q outputs of the flip-flops represent the binary number, with Q_0 being the least significant bit and Q_3 being the most significant bit.

The Q output of each flip-flop is connected to the T input of the next flip-flop in the series. This arrangement ensures that each subsequent flip-flop toggles at half the frequency of the previous one, enabling the counter to count from 0000 to 1111 (0 to 15 in decimal).

There is also a Reset input, which is used to set all Q outputs to 0, thereby resetting the counter to its initial state. This type of counter is commonly used in digital circuits for various counting purposes, such as in timers or clocks.

This implementation may not be the most efficient one due to the fact that it has a limited counting range. The 4-bit binary counter can only count from 0 to 15. This is insufficient for an alarm clock, which needs to count 60 seconds per minute, 60 minutes per hour, and 24 hours per day.

As a result of this insufficiency, we decided to use a different approach in Final Milestone of the project.

5 FPGA Implementation

5.1 Overview

To run the Digital Alarm clock, we used Verilog HDL language in Vivado to program the Basys 3 with AMD Artix 7 FPGA Board. We also used the Mealy's and Moore's Finite State Machine technique in the fsm file, which is the main thing for changing button states, clock alarm, and clock adjustments.

5.2 Design Process

5.2.1 Architecture

Modules: The digital alarm clock consists of the following primary modules:

5.2.2 `bcd_to_7_segment`

The `bcd_to_7_segment` module converts binary-coded decimal (BCD) input into 7-segment display output for displaying minutes and hours.

```verilog
module bcd_to_7_segment (
    input [1:0] en,
    input enable,
    input [3:0] minutes,
    input [2:0] minutes2,
    input [3:0] hours,
    input [1:0] hours2,
    output reg [6:0] segments,
    output reg [3:0] anode);
reg [3:0] num;
always @* begin
    case(en)
        0: begin
            anode = 4'b1110;
            num = minutes;
        end
        1: begin
            anode = 4'b1101;
            num = {1'b0, minutes2};
        end
        2: begin
            anode = 4'b1011;
            num = hours;
        end
        3: begin
            anode = 4'b0111;
            num = {2'b0, hours2};
        end
    endcase
    if(enable == 0) begin
        anode = 4'b1111;
    end
    case(num)
        0: segments = 7'b1000000;
        1: segments = 7'b1111001;
        2: segments = 7'b0100100;
        3: segments = 7'b0110000;
        4: segments = 7'b0011001;
        5: segments = 7'b0010010;
        6: segments = 7'b0000010;
        7: segments = 7'b1111000;
        8: segments = 7'b0000000;
        9: segments = 7'b0010000;
        default: segments = 7'b0001001;
    endcase
end
endmodule
```

Listing 1: Verilog Module: `bcd_to_7_segment`

5.2.3 `clk_divider`

The `clk_divider` module divides the input clock frequency by a given factor (`n`) to generate a slower clock output.

```verilog
module clk_divider #(
    parameter n = 5000000
)(
    input clk, rst, enable,
    output reg clk_out
);
wire [31:0] count;
counter #(32, n) counterMod (
    .clk(clk),
    .enable(enable),
    .reset(rst),
    .count(count)
);
always @ (posedge clk, posedge rst) begin
    if (rst)
        clk_out <= 0;
    else if (count == n - 1)
        clk_out <= ~clk_out;
end
endmodule
```

Listing 2: Verilog Module: `clk_divider`

5.2.4 counter

The `counter` module implements a counter that increments on each clock cycle when the `enable` signal is asserted.

```
module counter #(
    parameter x = 3,
    n = 6
)(
    input clk, reset, enable,
    output reg [x-1:0] count);
always @ (posedge clk, posedge reset) begin
    if (reset == 1)
        count <= 0;
    else if (enable == 1)
        if (count == n - 1)
            count <= 0;
        else
            count <= count + 1;
end
endmodule
```

Listing 3: Verilog Module: `counter`

This module takes an input clock (`clk`), a reset signal (`reset`), and an enable signal (`enable`). It outputs a counter value (`count`) that increments on each clock cycle when `enable` is asserted and resets to zero when the `n` count is reached.

This module takes an input clock (`clk`), a reset signal (`rst`), and an enable signal (`enable`). It generates a divided clock output (`clk_out`) by toggling it every `n` cycles of the input clock.

5.2.5 debouncer

The `debouncer` module implements a simple debouncer circuit to remove glitches from an input signal.

```
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
```

Listing 4: Verilog Module: `debouncer`

This module takes an input clock (`clk`), a reset signal (`rst`), and the input signal to be debounced (`in`). It implements a three-stage shift register (q1, q2, q3) to delay the input signal. The output `out` is set to the logical AND of the three registers, providing a debounced output signal.

5.2.6 digital_clock

The `digital_clock` module integrates various submodules to implement a digital clock.

```
module digital_clock (input clk, en, reset, output reg [6:0] segments, output
    [3:0] anode);
wire [1:0] count;
wire [3:0] seconds;
wire [2:0] seconds2;
wire [3:0] minutes;
wire [2:0] minutes2;
wire [3:0] hours;
wire [1:0] hours2;
clk_divider #(250000) divide (
    .clk(clk),
    .reset(reset),
    .enable(en),
    .clk_out(clkout)
);
// Instantiate two-bit counter module
counter #(2, 4) two_bit_counter (
    .clk(clkout),
    .reset(reset),
    .enable(en),
    .count(count)
);
// Instantiate hours, minutes, and seconds counter module
hms_counter hours_and_mins (
    .clk(clk),
    .reset(reset),
    .enable(en),
    .minutes(minutes),
    .minutes2(minutes2),
    .hours(hours),
    .hours2(hours2)
);
// Instantiate BCD to 7-segment display converter module
bcd_to_7_segment segment_1 (
    .en(count),
    .enable(en),
    .minutes(minutes),
    .minutes2(minutes2),
    .hours(hours),
    .hours2(hours2),
    .segments(segments),
    .anode(anode)
);endmodule
```

Listing 5: Verilog Module: `digital_clock`

This module integrates various submodules such as clock divider, counter, and BCD to 7-segment display converter to implement a digital clock.

5.2.7 hms_counter

The **hms_counter** module implements a counter for hours, minutes, and seconds.

```verilog
module hms_counter (
    input clk, rst, en,
    output [3:0] minutes,
    output [2:0] minutes2,
    output [3:0] hours,
    output [1:0] hours2);
wire clkout;
reg rstco;
wire [3:0] seconds;
wire [2:0] seconds2;
wire [5:1] enable;
wire reset;
// Instantiate clock divider module for 1 Hz clock
clk_divider #(50000) clock_1hz (
    .clk(clk),
    .rst(rst),
    .enable(en),
    .clk_out(clkout)
);
// Initialize reset control signal
initial begin
    rstco = 0;
end
// Logic for combining reset signals
assign reset = rst | rstco;
// Instantiate counter modules for seconds, minutes, and hours
counter #(4, 10) sec1_m (.clk(clkout), .reset(reset), .enable(en), .count(
    seconds));
counter #(3, 6) sec2_m (.clk(clkout), .reset(reset), .enable(enable[1]), .count
    (seconds2));
counter #(4, 10) min1_m (.clk(clkout), .reset(reset), .enable(enable[2]), .
    count(minutes));
counter #(3, 6) min2_m (.clk(clkout), .reset(reset), .enable(enable[3]), .count
    (minutes2));
counter #(4, 10) hours1_m (.clk(clkout), .reset(reset), .enable(enable[4]), .
    count(hours));
counter #(2, 3) hours2_m (.clk(clkout), .reset(reset), .enable(enable[5]), .
    count(hours2));
// Control logic for enabling counters based on current values
always @* begin
    // Enable second counter
    enable[1] = (seconds == 9 & en) ? 1 : 0;
    // Enable minute counter when seconds reach 59
    enable[2] = (seconds2 == 5 & seconds == 9 & en) ? 1 : 0;
    // Enable hour counter when minutes reach 59
    enable[3] = (minutes == 9 & seconds2 == 5 & seconds == 9 & en) ? 1 : 0;
    // Enable hour counter when minutes2 reach 59
    enable[4] = (minutes2 == 5 & minutes == 9 & seconds2 == 5 & seconds == 9 &
   en) ? 1 : 0;
    // Enable reset control when hours2 reach 23
    rstco = (hours2 == 2 & hours == 3 & minutes2 == 5 & minutes == 9 & seconds2
    == 5 & seconds == 9 & en) ? 1 : 0;
end

endmodule
```

Listing 6: Verilog Module: `hms_counter`

5.2.8 `push_button_detector`

The `push_button_detector` module detects and debounces a push button signal.

```
module push_button_detector (
    input clk, rst, pushed,
    output reg out_final);
reg out1;
reg out2;
// Instantiate debouncer module
debouncer debounce (
    .clk(clk),
    .rst(rst),
    .in(pushed),
    .out(out1)
);
// Instantiate synchronizer module
synchronizer synchronize (
    .clk(clk),
    .in(out1),
    .out(out2)
);
// Instantiate rising edge detector module
rising_edge_detector detect (
    .clk(clk),
    .rst(rst),
    .in(out2),
    .out_final(out_final)
);
endmodule
```

Listing 7: Verilog Module: `push_button_detector`

This module integrates a debouncer, a synchronizer, and a rising edge detector to reliably detect and debounce a push button signal.

This module integrates various counter submodules for hours, minutes, and seconds and includes logic to control their operation based on current values.

5.2.9 synchronizer

The `synchronizer` module synchronizes an asynchronous signal with the clock domain.

```
module synchronizer (input clk, sig, output reg sig_1);
reg meta;
always @ (posedge clk) begin
    meta <= sig;
    sig_1 <= meta;
end
endmodule
```

Listing 8: Verilog Module: `synchronizer`

This module synchronizes an asynchronous signal (`sig`) with the clock domain (`clk`). It stores the input signal in a register (`meta`) and then assigns it to the output register (`sig_1`) on the next clock edge.

5.2.10 two_by_4_decoder

The `two_by_4_decoder` module implements a 2x4 decoder.

```
module two_by_4_decoder (
    input clk, enable, rst,
    output reg [3:0] codes);
reg clkout;
reg count;
// Instantiate clock divider module
clk_divider #(5000000) divide (
    .clk(clk),
    .rst(rst),
    .clk_out(clkout)
);
// Instantiate two-bit counter module
counter #(2, 4) two_bit_counter (
    .clk(clkout),
    .reset(rst),
    .enable(enable),
    .count(count));
// Decode the count value into 4-bit codes
always @ (posedge clkout) begin
    case (count)
        0: codes = 4'b1110;
        1: codes = 4'b1101;
        2: codes = 4'b1011;
        3: codes = 4'b0111;
    endcase
end
endmodule
```

Listing 9: Verilog Module: `two_by_4_decoder`

This module implements a 2x4 decoder using a clock divider and a two-bit counter. The count value is decoded into 4-bit codes based on the current count value.

5.2.11 Constraints File

The following constraints define the pin assignments and I/O standards for the segments, anodes, clock, enable, and reset signals:

```
set_property PACKAGE_PIN W7 [get_ports {segments[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[0]}]

set_property PACKAGE_PIN W6 [get_ports {segments[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[1]}]

set_property PACKAGE_PIN U8 [get_ports {segments[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[2]}]

set_property PACKAGE_PIN V8 [get_ports {segments[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[3]}]

set_property PACKAGE_PIN U5 [get_ports {segments[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[4]}]

set_property PACKAGE_PIN V5 [get_ports {segments[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[5]}]

set_property PACKAGE_PIN U7 [get_ports {segments[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {segments[6]}]

set_property PACKAGE_PIN U2 [get_ports {anode[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[0]}]

set_property PACKAGE_PIN U4 [get_ports {anode[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[1]}]

set_property PACKAGE_PIN V4 [get_ports {anode[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[2]}]

set_property PACKAGE_PIN W4 [get_ports {anode[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode[3]}]

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

set_property PACKAGE_PIN V17 [get_ports en]
set_property IOSTANDARD LVCMOS33 [get_ports en]

set_property PACKAGE_PIN V16 [get_ports reset]
set_property IOSTANDARD LVCMOS33 [get_ports reset]
```

Listing 10: Pin Constraints

These constraints specify the pin assignments and I/O standards for various signals

6 Implementation Issues Faced and Solutions

6.1 Challenges Faced

We faced a couple of different challenges in the project. The issues were the following in both the First Milestone and the Final Milestone:

6.1.1 For the First Milestone:

- Designing, implementing and testing the component that is responsible for deriving the 7-segment display
- Designing, implementing and testing the HH:MM:SS counter.
- Time management
- Assigning what each persons needs to do
- Time restriction

6.1.2 For the Second Milestone:

- designing the fsm
- incrementing/decrementing the hours and minutes after stopping the counter
- Time restriction
- Clock display issues
- Implementing the alarm sound
- Button control and debouncing
- signal synchronization

7 Validation Activities

7.1 Functional Testing

Functional testing aims to verify that the digital alarm clock performs its intended functions accurately and reliably. This includes:

- Verifying that the clock actually displays time in hours and minutes accurately.
- Testing the clock's ability to update the time based on input from the clock signal.
- Testing the responsiveness of the clock's user interface, including button presses and mode switching.

7.2 Button Testing

Button testing focuses on validating the functionality of the buttons used to interact with the digital alarm clock. such as verifying that each button performs its specified required function, such as setting the time, toggling between adjust and clock/alarm modes, and activating/deactivating the alarm.

7.3 Display Verification

Display verification focuses on ensuring that the 7-segment display accurately represents the time and alarm settings. This includes:

- Verifying that each segment of the display correctly illuminates to represent the digits 0-9,
- Verifying that the hour, minutes, seconds switches from 59:59:59 to 00:00:00 when incrementing the adjust.

7.4 Alarm Functionality Testing

Alarm functionality testing assesses the digital alarm clock's ability to trigger and manage alarms effectively such as verifying that the alarm triggers at the specified time and activates the indicator, such as a sound or flashing LED.

8 Conclusion

In conclusion, this project successfully implemented a digital alarm clock using Verilog and targeted for the BASYS3 FPGA board. The system design incorporated various modules including debouncers, counters, decoders, and clock dividers to achieve the desired functionality. Future enhancements could include additional features such as user interface improvements (like displaying the digital clock to an external VGA display), alarm sound generation, and support for multiple alarm settings.