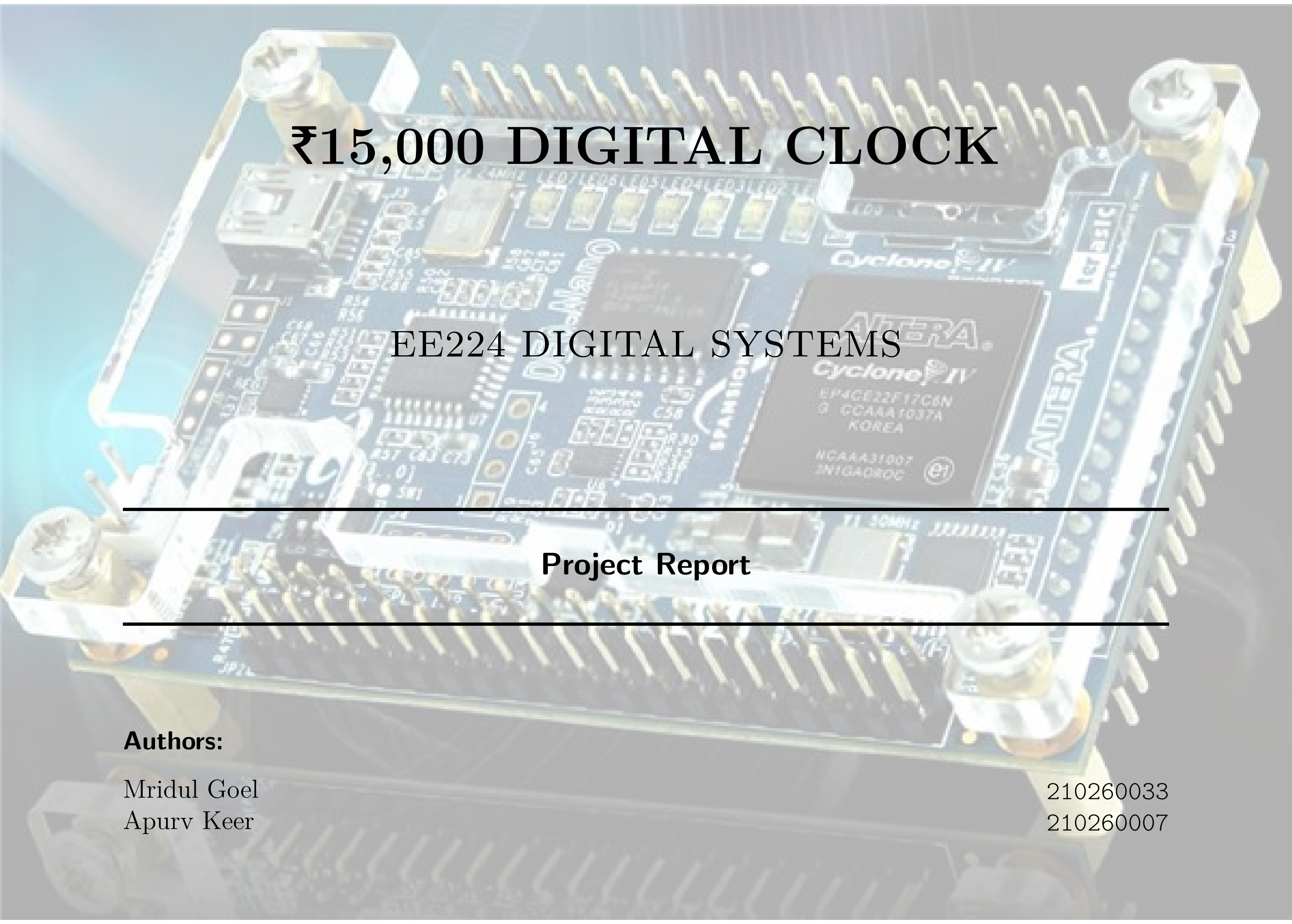


₹15,000 DIGITAL CLOCK

EE224 DIGITAL SYSTEMS

Project Report

Authors:

Mridul Goel	210260033
Apurv Keer	210260007

09-04-2023
2022/2023 – 2nd Semester

Contents

1 Summary

We are developing a digital clock. The format is 24-hour. In this mode, the user can enter a suitable time, and the clock will set its value to that time and begin ticking at that moment.

2 Theoretical design

2.1 Block Diagram

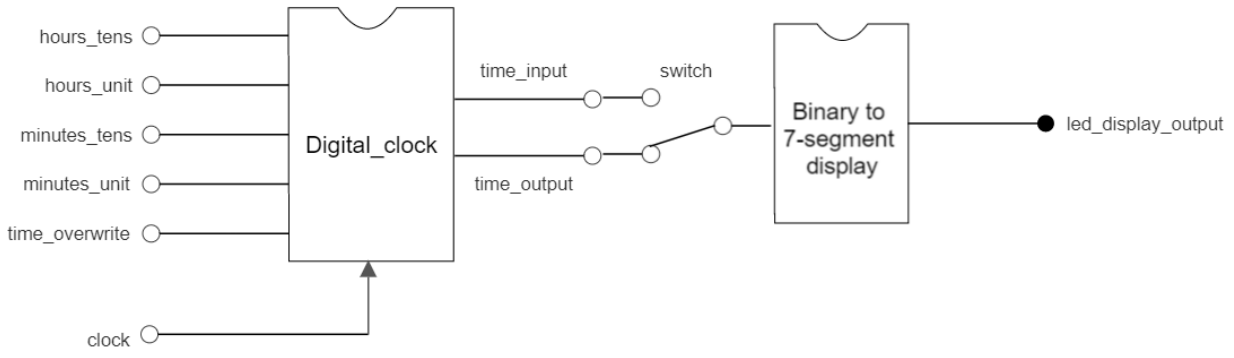


Figure 1: Digital Clock

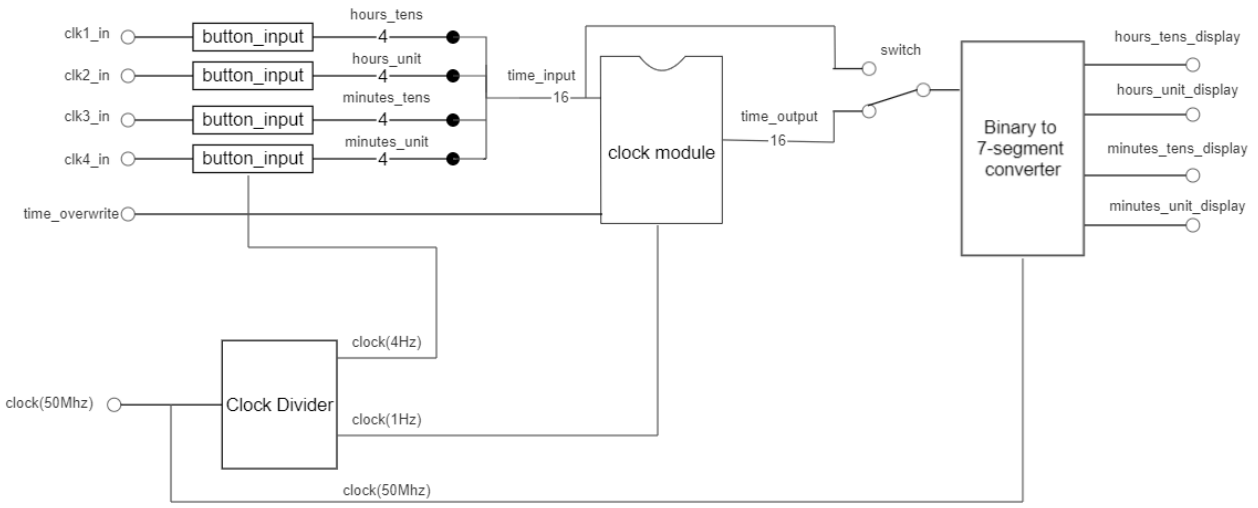


Figure 2: Internal design of the Digital Clock module

2.2 Ports

Input ports

- hours tens (`clk1_in`) - single pin the input for the ten's digit of the hour. It is a single push button, where the digit increments by the number of times of button is pressed.
- hours unit (`clk2_in`) - single pin input for the unit's digit of the hour.
- minutes tens (`clk3_in`) - single pin input for the ten's digit of the minute
- minutes unit (`clk4_in`) - single pin input for the unit's digit of the minute.
- time overwrite (`time_ow`) - a single pin push button to overwrite the current time by the time input
- switch (`switch`) - a single pin switch to change the display from the current time to the time input and vice versa.
- clock (`clk`) - internal clock of the fpga

Output ports

- hours tens [7:0] (`hrs_t`) - output for the ten's digit of the hour. It consists of 8 pins, all of them collectively driving a single 7-segment LED display.
- hours units [7:0] (`hrs_u`) - 8 pins output to drive the unit's digit of the hour. It similarly drives a 7-segment LED display.
- minutes tens [7:0] (`min_t`) - 8 pins output to the tens digit of the minute.
- minute unite [7:0] (`min_u`) - 8 pins output to the units digit of the minute.

The output of the digital clock block, which accepts a clock signal as input, is a 16-bit binary time value. The specifics of it will be revealed later. A 16-bit internal D-register, time input, is also present. The value provided by the user is stored here.
The switch controls the two operating modes, storing and output. The value of the time output is provided to the display driver when the switch is in the output mode. On the other hand, the value of the time input is communicated to the display driver when the switch is in store mode.
To enter a time value, the user must first switch to store mode, and then use the four available buttons to select the desired numbers, digit by digit. Once the time value is entered, the user must switch back to output mode and then click the overwrite button to update the clock's time value.

2.3 Modules

2.3.1 Clock Signal module

This module generates a 1 Hz and a 4 Hz clock signal using the FPGA's original internal clock, which operates at 50 MHz. In order to achieve this, the internal clock's rising edges are accurately counted, and the signal's value is flipped from zero to one and back to zero.

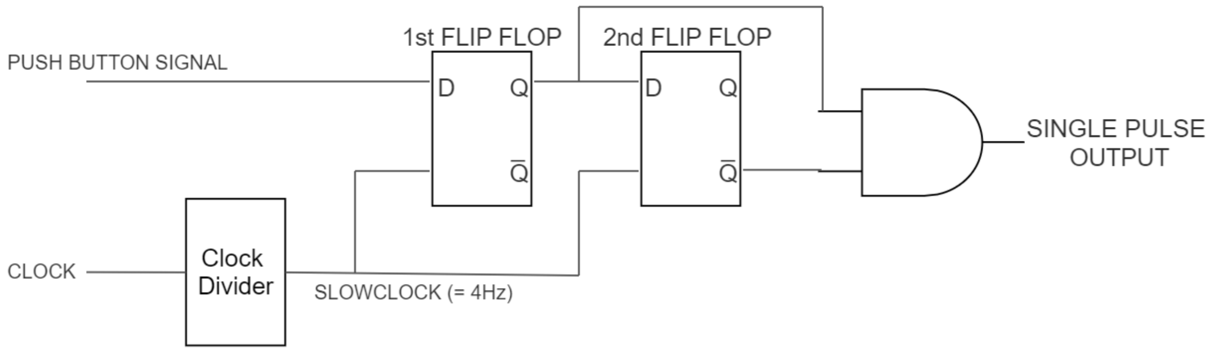


Figure 3: Debouncer Circuit

2.3.2 Input module

This module accepts four inputs: Clk1 in, Clk2 in, Clk3 in, and Clk4 in, each of which corresponds to a physical button. As the buttons are prone to generating extra rising and falling edges due to bouncing, a debouncer circuit is employed to handle these inputs. The circuit schematic for the debouncer is displayed in **??**.
The 4 Hz clock signal is taken from the clocksignal module. Now the debounced signal is passed as a clock signal into a counter. The counter detects a negetive edge of the clock and adds one to its count. This count is given as an output. So now if we press the push button four times, 4 will be stored in the counter which will be given as the output. If the `switch` input is set to *low*, the value being input is displayed on the output LEDs. This way the user can select the required time, selecting each of the two digits individually for the hours and minutes.

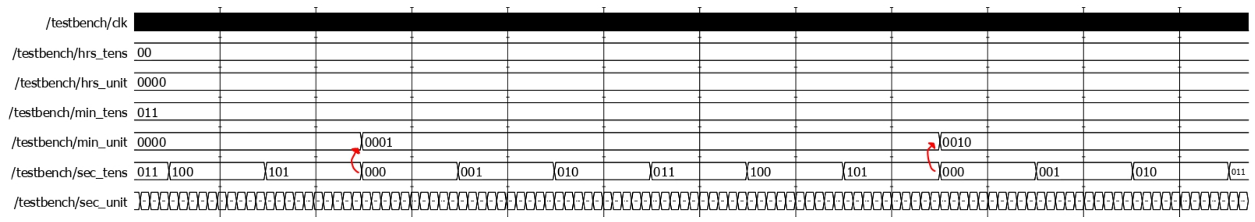
2.3.3 Clock Work module

This module is the main circuit of the digital clock. Each digit of the hours and minutes part is operated separately. This module is mostly made up of various if else statements like when the unit part of minute goes to 9 and its tens part goes to 5, the hours unit place will be incremented by 1 on the next rising edge. Various such conditional statements are present in this module. There is a 16-bit D-Register present in this module corresponding to 4 bits for one digit. The value of time input, the user gives, is the input of these D-Registers. So when the time overwrite button is pressed, a signal is produced which acts as a clock signal for these D-Registers to transfer this data to the main clock module so it starts from that time.
The digital clock module is made just to bring all these different modules together. Now the time output data is ready to be shown on the display.

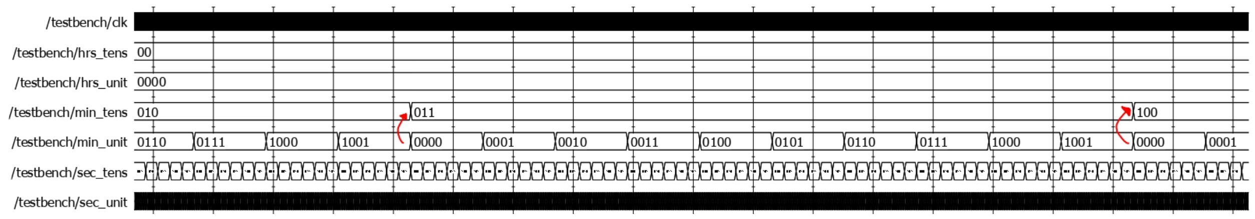
2.3.4 Led Display module

This module helps to show the output on an array of four 7-segment displays. This module divides the 16-bit data into four 4-bit data, one for each digit. Then it converts the 4-bit number into the 7-segment display code and gives an output of 8 pins that drive each LED of the 7-segment display separately.

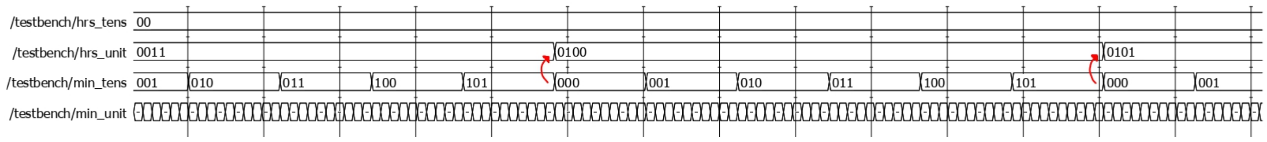
2.4 ModelSim simulation


/testbench/clk
/testbench/hrs_tens 00
/testbench/hrs_unit 0000
/testbench/min_tens 011
/testbench/min_unit 0000 0001 0010
/testbench/sec_tens 011 100 101 000 001 010 011 100 101 000 001 010 011
/testbench/sec_unit

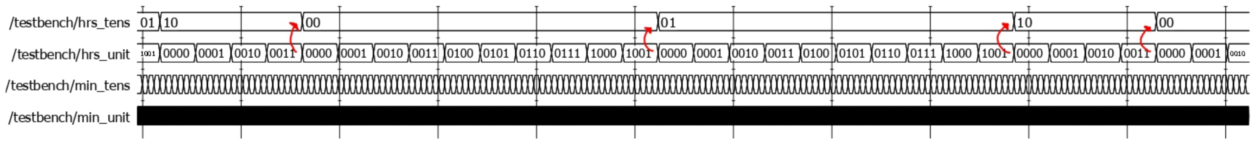
Here you can see as the tens place of the second increments to 6 it falls back to 0 and increments the minute's unit place by 1.


/testbench/clk
/testbench/hrs_tens 00
/testbench/hrs_unit 0000
/testbench/min_tens 010 011 100
/testbench/min_unit 0110 0111 1000 1001 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 0000 0001
/testbench/sec_tens
/testbench/sec_unit

Here you can see as the unit place of the minute increments to 9 it falls back to 0 and increments the minute's tens place by 1.


/testbench/hrs_tens 00
/testbench/hrs_unit 0011 0100 0101
/testbench/min_tens 001 010 011 100 101 000 001 010 011 100 101 000 001
/testbench/min_unit

Here you can see as the tens place of the minute increments to 6 it falls back to 0 and increments the hour's unit place by 1.


/testbench/hrs_tens 01 10 00 01 10 00
/testbench/hrs_unit 1001 0000 0001 0010 0011 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 0000 0001 0010 0011 0000 0001 0010
/testbench/min_tens
/testbench/min_unit

Here you can see as the unit place of the hour increments to 9 it falls back to 0 and increments the minute's tens place by 1. This goes on for two shifts and then the hour's tens place falls to 0 in just four increments of the hour's unit place.

For some reason the annotations on the pdf were not visible in the LaTeX file. Sorry!

3 Experimental Implementation

When we turn the power on, the clock starts from some time which is not significant. We can see the clock working properly. The hours and minutes part of the time is visible on the 7-segment displays.

There are five push buttons and a switch in our clock. The buttons are properly debounced so as to not cause any issues due to fluctuations. So now if the user wants to change the time to set it to the current time, he or she has to do the following:

1. Toggle the switch to enter the input mode. Now you can see that the output displays are set to some constant value, default 11:11. This is the value of the data stored in the time input D-Registers.

2. There are four switches, one for each digit. When you press the button, the digit increments by 1. This way you can set the value for each digit separately. When the unit places of hours and minutes reach 9, incrementing it returns to 0. Similarly, for tens place of minutes, the limit is 6, and for tens place of hours, it's 2.

3. Once you have successfully entered the desired value, toggle the switch back to the output mode to see the implementation of what you did. Now, finally, press the time overwrite button provided. You will be able to see that the time is changed and it starts from the user inputted value.

4 Connection with Physics

To begin this section, we need to first define and understand what exactly a clock is. Essentially, a clock is a representation of the first dimension of the Minkowski space in the other three dimensions. Further, as the speed of the clock can be varied, we can also demonstrate changes in the metric being used. This can be used to demonstrate the time dilation effect of The Special Theory of Relativity. By varying the tick rate of the clock, we can demonstrate the time dilation effect predicted by both the Special and General Theories of Relativty. This effect refers to the fact that time appears to slow down for an observer who is moving relative to a stationary observer. This is a consequence of the fact that the speed of light is constant in all inertial reference frames.

For example, imagine that two identical clocks are set up, one on a spacecraft traveling at a high speed and the other on Earth. According to Special Relativity, the clock on the spacecraft will appear to run more slowly than the clock on Earth, from the perspective of an observer on Earth. This effect has been confirmed experimentally using atomic clocks, which are capable of measuring time with incredible precision.

Taking a more practical approach, we can create a very high-frequency timing mechanism using this clock, which can be useful to measure the time delta in certain very fast experiments. We can utilize the very high speed of the FPGA to measure very small time differences, which can be used to synchronise and measure very fast processes and events, and also determine if there is any small change from expected measurements.

In some applications, FPGAs are used to create precise timing signals that are synchronized with an external clock source, such as a GPS clock or an atomic clock. These timing signals can then be used to measure the time difference between two events with very high accuracy.

Similarly, FPGAs are used in telecommunications applications to measure and synchronize high-speed data transmissions. By using a high-frequency clock and precise timing measurements, it is possible to minimize errors and ensure that data is transmitted and received accurately.

Appendix A - Project Evolution

The `clockwork.v` file's code was the easiest part of the project and we completed it in the first week. However, the input and output modules required more effort as we came up with several ideas and methods to implement them, and we had to test various versions of each method. To test the code, we initially used Display Units, which were simple to use but we were later asked to use more advanced output methods such as LCD or seven-segment displays, which added more complexity to the project.

Initially, we planned to use the JHD 162A 16×2 LCD display for our project. However, we soon encountered difficulties with this choice. To function properly, the LCD display needed to be initialized when the circuit powers up for the first time. Unfortunately, the exact initialization sequence for our particular display was not clearly provided in its datasheet and was not available online. We attempted to modify the initialization sequence for other LCD displays to make ours work, but we were unsuccessful. Even after seeking assistance from the TAs in the lab, we still couldn't find a solution. After devoting approximately 1.5 weeks to the LCD display, we decided to abandon this method for output and switched to using seven-segment displays instead.

For the seven-segment displays, we were advised to use the IC HD3991 driver to convert the four-bit integer input signal into eight-bit output signals that could be fed into the display. However, we encountered a problem where the IC did not function as we had expected. Despite spending several hours attempting to debug our circuit, we were unable to find a solution. Ultimately, we decided to implement the four-bit to eight-bit conversion on the FPGA itself.

We developed a separate module in Verilog that enabled the conversion of four-bit numbers into the corresponding eight bits required by the 7-segment display. Each display was connected directly to the FPGA using eight pins. However, this approach resulted in a larger number of output pins than we desired. To address this issue, we planned to implement a sequential output system in which each digit's output signal would be sent sequentially from the FPGA, and a demultiplexer would select which display to send the signal to. Unfortunately, due to time constraints, we were unable to construct a functioning circuit at the time of writing this report. Therefore, we have omitted it from the remainder of the report. We plan to implement this if possible after submitting the report.

For the input module, we considered various options, and the most favourable one appeared to be the usage of the 4×4 number pad. However, as with many other teams loooking to use the number pads, we were unable to extract accurate button press information from the pad, and discarded the idea. We next considered using the analog pins with the inbuilt ADC on the FPGA. We planned to use a potentiometer knod to allow the user to input a specific 2 digit number according to the time he/she/they/them/etc. wanted. However, the potentiometers available in the lab weren't exatcly user-friendly knobs, and so we instead decided to work on a variation of the analog input.

We made a push-button input circuitry, where the value of the digit in focus is incremented by one on every push of the button. Push buttons being what they are, we obtained lots of erronous inputs, and realised that we needed to implement a debouncer circuit. Once again, instead of implementing the circuit on a breadboard, we wrote a code which vaguely acted as a debouncer on the FPGA itself and got the buttons to work satisfactorily.

Just before writing this report, we tried to add date and month output to the system, but it resulted in a very complicated and messy circuit in the absence of a sequential output module, and the circuit proved to be not working and very difficult to debug. We discarded this with time running out. As a last effort, we also tried implementing an alarm system, but didn't really have time to complete it. As of now, after writing this report, we plan to work on the sequential output, as it will make a large difference to our external circuit as well as allow us freedom to implement other features as well.

Appendix B - Verilog code

For your reference, I am also including the github link to our codes here. Our code consists of 5 separate `verilog` files.

```
/
└─ clocksignal
   └─ clockwork
      └─ input1
         └─ digital_clock
            └─ four_bit_to_led
```

clocksignal.v|

Let's begin with the file `clocksignal.v` file. It is the simplest, and just generates the clock signals of the required frequencies

```verilog
module clock_1sec(clk_og,clk_1hz);
    input clk_og;
    output reg clk_1hz=0;

    reg[25:0] count = 0;

    always@(posedge clk_og)
    begin
        if(count<250000)
        begin
            count<=count+1;
        end
        else
        begin
            count<=0;
            clk_1hz<=~clk_1hz;
        end
    end

endmodule

module clock_4Hz(clk_og,clk_4hz);
    input clk_og;
    output reg clk_4hz=0;

    reg[25:0] count = 0;

    always@(posedge clk_og)
    begin
        if(count<6250000)
        begin
            count<=count+1;
        end
        else
        begin
            count<=0;
            clk_4hz<=~clk_4hz;
        end
    end

endmodule
```

clockwork.v|

This file handles the updation of the seconds, minutes and hours.

```
module clockWorkDec(clk_1hz, time_in, time_out, time_ow);
  input clk_1hz, time_ow; //1 Hz clock (clock), Time overwrite (asynchronous reset)
  //Time signal format: hh_hhhh:mmm_mmmm:sss_ssss only decimal values
  input [19:0] time_in; //time input
  output [19:0] time_out; //main output

  //separated time signals to respective meaning
  wire [6:0] sec_in, min_in;
  wire [5:0] hour_in;
  reg [6:0] sec_reg, min_reg;
  reg [5:0] hour_reg;

  //separation and combination of time signals
  assign {hour_in, min_in, sec_in} = time_in;
  assign time_out = {hour_reg, min_reg, sec_reg};

  //handle seconds
  always@(posedge clk_1hz or negedge time_ow)
    begin
      if(~time_ow)
        begin
                 sec_reg <= sec_in;
        end
      else
        begin
          casex(sec_reg)
            7'h59:
              begin
                sec_reg <= 7'd0;
              end
            7'h?9:
              begin
                sec_reg <= {(sec_reg[6:4]+3'd1),4'h0};
              end
            default:
              begin
                sec_reg <= sec_reg + 7'd1;
              end
          endcase
        end
    end

  //handle minutes
  always@(posedge clk_1hz or negedge time_ow)
    begin
      if(~time_ow)
        begin
                min_reg <= min_in;
        end
      else
        begin
          if(sec_reg == 7'h59)
            begin
              casex(min_reg)
                7'h59:
                  begin
                    min_reg <= 7'h0;
                  end
                7'h?9:
                  begin
                    min_reg <= {(min_reg[6:4]+3'd1),4'h0};
                  end
                default:
                  begin
                    min_reg <= min_reg + 7'd1;
                    end
                  endcase
                end
            end
        end

    //handle hours
    always@(posedge clk_1hz or negedge time_ow)
      begin
        if(~time_ow)
          begin
              hour_reg <= hour_in;
          end
        else
          begin
            if((sec_reg == 7'h59)&(min_reg == 7'h59))
              begin
                casex(hour_reg)
                  6'h23:
                    begin
                      hour_reg <= 6'd0;
                    end
                  6'b0?1001: //09 & 19
                    begin
                      hour_reg <= {(hour_reg[5:4]+3'd1),4'd0};
                    end
                  default:
                    begin
                      hour_reg <= hour_reg + 6'd1;
                    end
                endcase
              end
          end
      end
endmodule
```

input1.v|

This file hanles the input module.

```
module button_input (clk, clk1_in, clk2_in, clk3_in, clk4_in, min_unit, min_tens, hrs_unit,
        hrs_tens);
    input clk1_in, clk2_in, clk3_in, clk4_in, clk;
    wire clk1, clk2, clk3, clk4;
    output reg [3:0] min_unit, hrs_unit = 4'd0;
    output reg [2:0] min_tens = 3'd0;
    output reg [1:0] hrs_tens = 2'd0;

    debouncer A (
        .clk(clk),
        .button(clk1_in),
        .signal(clk1)
    );

    debouncer B (
        .clk(clk),
        .button(clk2_in),
        .signal(clk2)
    );

    debouncer C (
        .clk(clk),
        .button(clk3_in),
        .signal(clk3)
    );

    debouncer D (
```

```verilog
        .clk(clk),
        .button(clk4_in),
        .signal(clk4)
    );

    always@ (negedge clk1) begin
        if (min_unit == 9) begin
            min_unit = 4'd0;
        end
        else begin
            min_unit = min_unit + 4'd1;
        end
    end

    always@ (negedge clk2) begin
        if (min_tens == 5) begin
            min_tens = 3'd0;
        end
        else begin
            min_tens = min_tens + 3'd1;
        end
    end

    always@ (negedge clk3) begin
        if (hrs_unit == 9) begin
            hrs_unit = 4'd0;
        end
        else begin
            hrs_unit = hrs_unit + 4'd1;
        end
    end

    always@ (negedge clk4) begin
        if (hrs_tens == 2) begin
            hrs_tens = 4'd0;
        end
        else begin
            hrs_tens = hrs_tens + 4'd1;
        end
    end

endmodule

module debouncer(clk, button, signal);

    input button, clk;
    wire clk_4hz;
    reg d1, d2, d3;
    output signal;

    clock_4Hz (
        .clk_og(clk),
        .clk_4hz(clk_4hz)
    );

    always@ (posedge clk_4hz) begin
        d1 <= button;
        d2 <= ~d1;
        d3 <= (d1 & d2);
    end

    assign signal = d3;

endmodule
```

digital_clock.v|

This file forms the backbone of the project, integrating the above four modules with the output module.

```
module digital_clock(clk, time_ow, time_out, clk1_in, clk2_in, clk3_in, clk4_in, time_in);
    input clk, time_ow; //1 Hz clock (clock), Time overwrite (asynchronous reset)
    input clk1_in, clk2_in, clk3_in, clk4_in;
   //Time signal format: hh_hhhh:mmm_mmmm:sss_ssss only decimal values
   output [19:0] time_out; //main output
   wire clk_1hz;
    output [19:0] time_in;
    wire [3:0] min_unit, hrs_unit;
    wire [2:0] min_tens;
    wire [1:0] hrs_tens;
    reg [3:0] sec_unit = 4'b0;
    reg [2:0] sec_tens = 3'b0;

    assign time_in = {hrs_tens, hrs_unit, min_tens, min_unit, sec_tens, sec_unit};

   clock_1sec A (
        .clk_og(clk),
        .clk_1hz(clk_1hz)
    );

   clockWorkDec C (
        .clk_1hz(clk_1hz),
        .time_ow(time_ow),
        .time_in(time_in),
        .time_out(time_out)
   );

    button_input B (
        .clk(clk),
        .clk1_in(clk1_in),
        .clk2_in(clk2_in),
        .clk3_in(clk3_in),
        .clk4_in(clk4_in),
        .min_unit(min_unit),
        .min_tens(min_tens),
        .hrs_tens(hrs_tens),
        .hrs_unit(hrs_unit)
    );


endmodule
```

four_bit_to_led.v|

This is the top level entity of the project. This outputs the desired value onto the displays.

```
module led_output(sec_u, sec_t, min_u, min_t, hrs_u, hrs_t, clk, time_ow, clk1_in, clk2_in,
         clk3_in, clk4_in, switch);
    input clk, clk1_in, clk2_in, clk3_in, clk4_in, time_ow, switch;
    wire [19:0] time_out;
    wire [19:0] time_in;
    reg [19:0] option;
    output reg [7:0] sec_u, sec_t, min_u, min_t, hrs_u, hrs_t = 8'b0;

    digital_clock A (.clk(clk), .time_ow(time_ow), .time_out(time_out), .clk1_in(clk1_in),
                    .clk2_in(clk2_in), .clk3_in(clk3_in), .clk4_in(clk4_in), .time_in(time_in));

    always @(posedge clk) begin

        if (switch) begin
            option <= time_out;
```

```verilog
        end
        else begin
            option <= time_in;
        end

      case (option[19:18])
        2'b00 : hrs_t <= 8'b00000010;
        2'b01 : hrs_t <= 8'b10001111;
        2'b10 : hrs_t <= 8'b00010001;
      endcase

        case (option[17:14])
        4'b0000 : hrs_u <= 8'b00000010;
        4'b0001 : hrs_u <= 8'b10001111;
        4'b0010 : hrs_u <= 8'b00010001;
        4'b0011 : hrs_u <= 8'b00000101;
        4'b0100 : hrs_u <= 8'b10001100;
        4'b0101 : hrs_u <= 8'b01000100;
        4'b0110 : hrs_u <= 8'b01000000;
        4'b0111 : hrs_u <= 8'b00001111;
        4'b1000 : hrs_u <= 8'b00000000;
        4'b1001 : hrs_u <= 8'b00000100;
      endcase

        case (option[13:11])
        3'b000 : min_t <= 8'b00000010;
        3'b001 : min_t <= 8'b10001111;
        3'b010 : min_t <= 8'b00010001;
        3'b011 : min_t <= 8'b00000101;
        3'b100 : min_t <= 8'b10001100;
        3'b101 : min_t <= 8'b01000100;
      endcase

        case (option[10:7])
        4'b0000 : min_u <= 8'b00000010;
        4'b0001 : min_u <= 8'b10001111;
        4'b0010 : min_u <= 8'b00010001;
        4'b0011 : min_u <= 8'b00000101;
        4'b0100 : min_u <= 8'b10001100;
        4'b0101 : min_u <= 8'b01000100;
        4'b0110 : min_u <= 8'b01000000;
        4'b0111 : min_u <= 8'b00001111;
        4'b1000 : min_u <= 8'b00000000;
        4'b1001 : min_u <= 8'b00000100;
      endcase


    end // always @ (posedge clk)


endmodule
```

References

1. Chat GPT, `https://openai.com/blog/chatgpt` has been a constant source of unwavering support in our project, and learning verilog wouldn't have been possible without it.

2. *Suoglu*'s code for a more comprehensive clock in verilog, although not that well written, has been very helpful in providing a starting point for our code.

3. Of course, if Texas Instrument's datasheet for the fpga board doesn't find its place here, no one else would.

4. Although we received nothing in return from it, we spent a lot of time on the 16 LCD display's datasheet.

5. For the `verilog` typesetting in LaTeXfor this report, I used this stackexchange link.