A 12 Hour Digital Clock Implemented on FPGA using Verilog HDL

Table of Figures

Table of Contents

CHAPTER 1: INTRODUCTION

1.1 WHAT IS VERILOG HDL?

Verilog Hardware Description Language (HDL) is a hardware description language used to model and design digital circuits at the register-transfer level (RTL). It is widely used in the field of digital electronics for describing the behaviour of digital systems and designing electronic circuits.

1.2 NEED FOR VERILOG HDL

Verilog was originally developed to model and simulate digital circuits. Over time, it evolved into a hardware description language used for both simulation and synthesis. Synthesis tools can take Verilog code and generate a netlist that can be implemented in programmable logic devices (PLDs) or application-specific integrated circuits (ASICs).

1.3 WHAT ARE FPGA's?

FPGA stands for Field-Programmable Gate Array. It is a type of integrated circuit (IC) that can be configured by a user or a designer after manufacturing. Unlike traditional application-specific integrated circuits (ASICs) that are designed for a specific purpose and have their functionality fixed during the manufacturing process, FPGAs offer reconfigurability, allowing users to implement custom digital circuits.

1.4 APPLICATIONS OF FPGA

FPGAs find applications in a wide range of fields, including digital signal processing, telecommunications, networking, image and signal processing, aerospace, automotive, and more. They are particularly useful for prototyping, rapid development, and low to moderate volume production of digital systems

CHAPTER 2: LITERATURE REVIEW

2.1 COUNTERS IN VERILOG

Counters are digital circuits in Verilog that count a sequence of events or clock cycles. In digital design, counters are widely employed for tasks such as frequency division, time measurement, and control signal creation.

Moreover, counters are a type of sequential circuits since it requires to store the previous value of count for incrementing or decrementing the value of count and thus require clock for synchronization purposes.

//Sample code for an UpDownCounter

```
module UpDownCounter (
  input clk,
  input reset,
  input up_down, // Up or down control signal
  output reg [3:0] count
);

always @(posedge clk or posedge reset) begin
  if (reset)
    count <= 4'b0000; // Reset the counter to zero
  else if (up_down)
    count <= count + 1; // Increment the counter if up_down is high
  else
    count <= count - 1; // Decrement the counter if up_down is low
end

endmodule
```

2.2 SLOW CLOCK GENERATION

In digital electronics, a "slow clock" typically refers to a clock signal with a lower frequency compared to other clocks in the system. The clock signal is a periodic waveform that oscillates between high and low voltage levels, and it serves as a timing reference for various operations within a digital system. The speed or frequency of a clock is measured in Hertz (Hz) and represents the number of clock cycles per second.

In this project, we’ve used Nexys 4 DDR FPGA which has an inbuilt clock of 100MHz, so to generate various clock signals for the working of push buttons or to create a counter we used the concept of slow clock generation.

2.3 BUTTON DEBOUNCING

Button debouncing is a technique used in digital design to eliminate or reduce the effects of mechanical noise and bouncing that can occur when a button or switch is pressed or released. Mechanical switches are prone to bouncing, causing multiple transitions from open to closed (or vice versa) within a short time when the button is pressed or released. Debouncing ensures that only a single clean signal is generated for each press or release event.

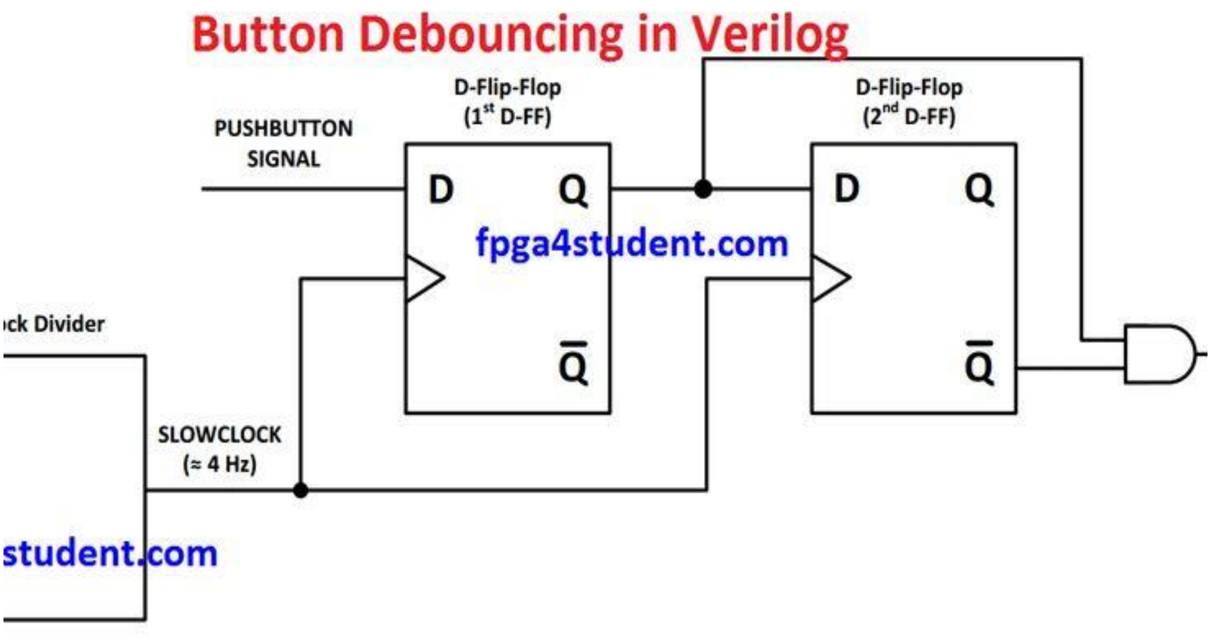

Figure 1: Button Debouncing circuit

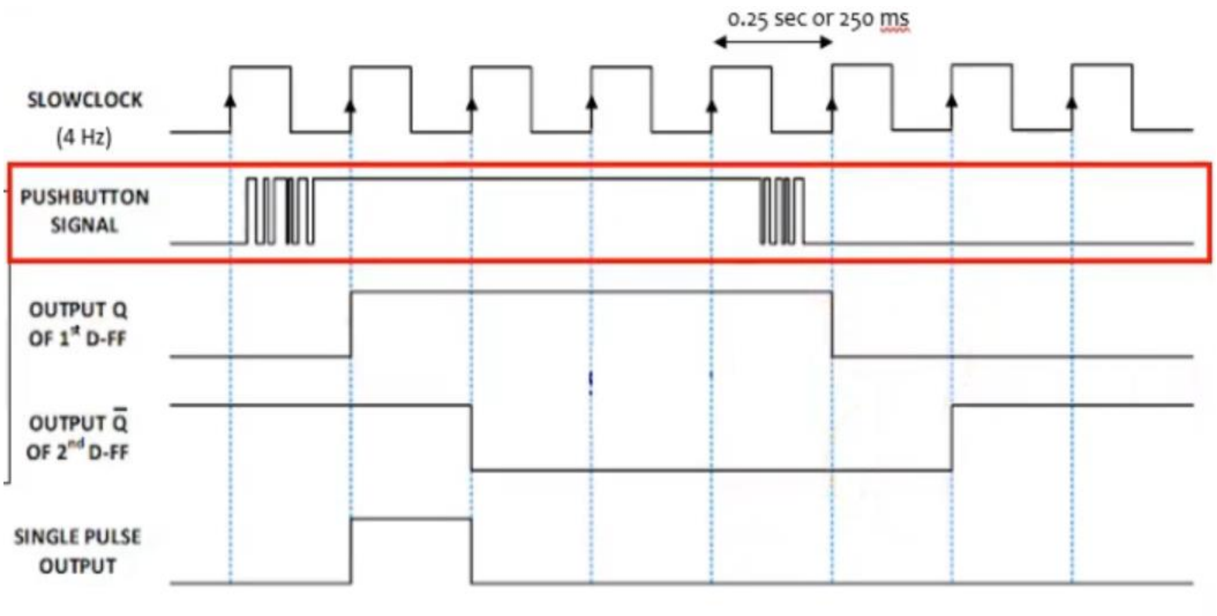

Figure 2: Button Debouncing waveforms

2.4 SEVEN SEGMENT DISPLAY (SSD)

A seven-segment display is a common form of electronic display device used to depict decimal values. It is made up of seven separate LED (Light Emitting Diode) segments grouped in the shape of the number "8." Each section can be separately manipulated to produce the digits 0 through 9. On the seven-segment display, you can display different digits by illuminating particular combinations of segments.

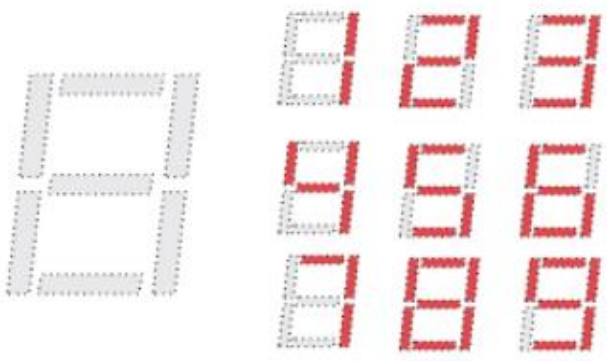

Figure 3: Digits on SSD

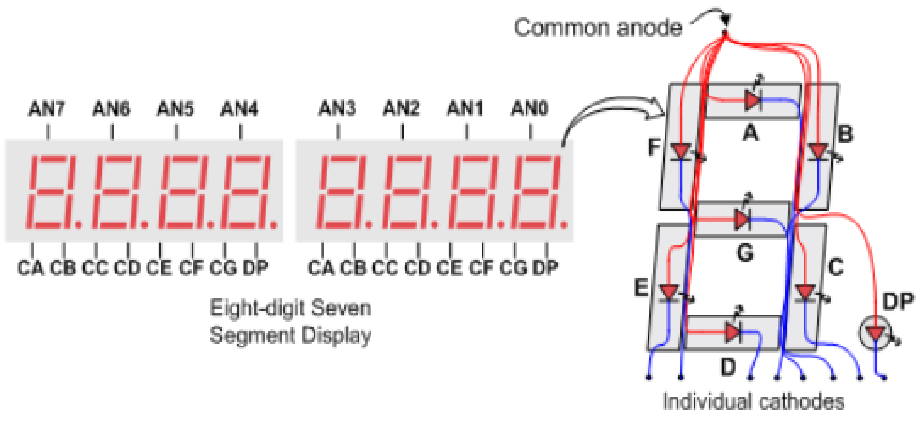
Common anode
AN7 AN6 AN5 AN4 AN3 AN2 AN1 AN0
CA CB CC CD CE CF CG DP CA CB CC CD CE CF CG DP
Eight-digit Seven Segment Display
A
B
C
D
E
F
G
DP
Individual cathodes

Figure 4: SSD configuration on Nexys 4 DDR FPGA board

Nexys 4 DDR FPGA board consists of 8 Seven Segment display units namely (AN0, AN1…….., AN7). Each of the Seven Segment Display is of Common Anode type which means that anode of each of the seven segments of a particular unit of SSD is connected together. The cathodes of similar segments on all four displays are connected into seven circuit nodes labelled CA through CG. For example, the eight "D" cathodes from the eight digits are grouped together into a single circuit node called "CD."

2.5 THE 12 HOUR CLOCK CONCEPT

The 12 Hour clock made in this project uses the concept of counters for its functioning. The seconds value increments once the counter counts up to 60, similar case for the minutes and hours values.

An AM/PM indicator LED has been used to depict all the 24 hours in a day using just 12 hours concept. Once the clock goes past 11:59 AM and the AM/PM indicator LED turns ON depicting noon timings and when the clock goes past 11:59 PM the AM/PM indicator LED turn OFF depicting the morning time.

CHAPTER 3: METHODOLOGY

3.1 OVERVIEW OF PROJECT

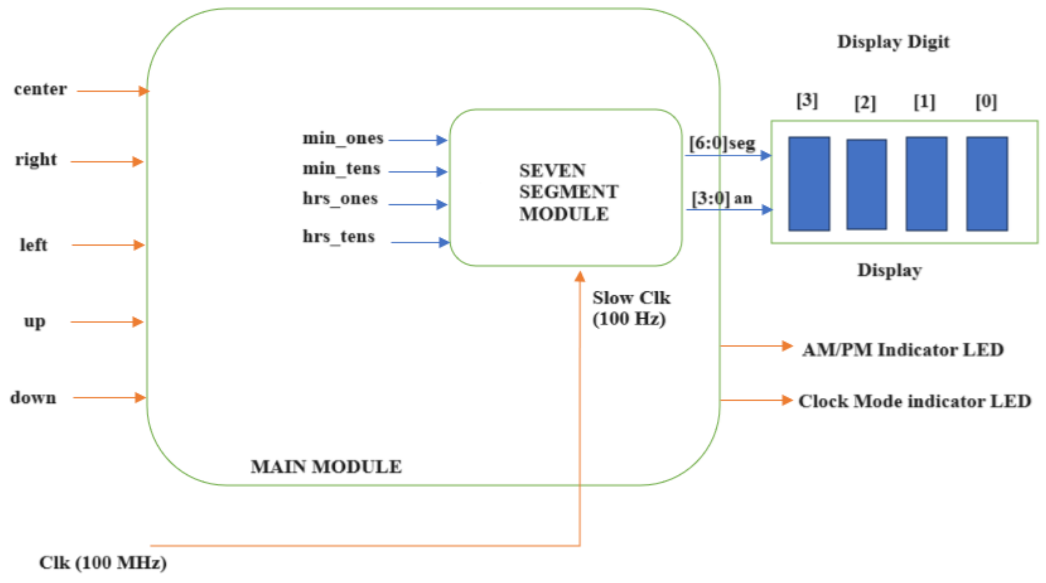

Figure 5: Block Diagram of the clock

The Project is divided into two modules, Main Module and Seven Segment Module. The inputs to the Main Module include the five push buttons (center, right, left, up, down) and the system inbuilt 100 MHz clock. The outputs of the Main Module include 7-bit seg, 4-bit a variable used for controlling the Seven Segment Display, AM/PM indicator LED, Clock Mode Indicator LED.

The inputs to the Seven Segment Module include the minutes and the hours variable (min_ones , min_tens, hrs_ones, hrs_tens) that control the minutes and hours display on the SSD. The outputs include [6:0]seg and [3:0]an variable.

A slow clock of 100 Hz is provided to the Seven Segment Module. The [6:0]seg variable is responsible for the 7 segments of a unit of Seven Segment Display whereas the [3:0]an variable is responsible for selecting a particular Seven Segment Display unit.

3.2 VERILOG CODE OF THE PROJECT

(Main Module – digital_clock)

```
module digital_clock(clk_100MHz,center_pb,right_pb,left_pb,up_pb,down_pb,seg,an,AM-
PM_indicator_led,clock_mode_indicator_led);


//declaring input and output variables for digital clock main module

input clk_100MHz; //system inbuilt clk (100MHz)

input center_pb,right_pb,left_pb,up_pb,down_pb;

output [6:0] seg;

output [7:0] an;

output AM-PM_indicator_led, clock_mode_indicator_led;


//setting up variables for generating slow clock

reg [31:0] counter = 0;

parameter max_count = 10_00_00_000; // 100MHz/1Hz = 100M/2 (for low and high
pulses) = 50M;


//setting up variables for hours and minutes on clock's display

reg [5:0] hrs, min, sec = 0; //hrs: 1-12 , min: 0-59 , sec: 0-59

reg [3:0] min_ones, min_tens, hrs_ones, hrs_tens = 0; //initally all variables are
set to 0

reg toggle = 0; // toggle between min and hrs , 0 -> min , 1 -> hrs


//assigning initial value to indicator leds

reg pm = 0;

assign AM-PM_indicator_led = pm; // initially led is set to 0 , hence clock
displays time in AM

reg clock_mode = 0;
```

```
assign clock_mode_indicator_led = clock_mode; //initially led is set to 0 , hence
clock not in clock mode


// instantiating the seven_seg_disp_module

seven_seg_disp_module ssd(.clk_100MHz(clk_100MHz), .min_ones(min_ones),
.min_tens(min_tens),.hrs_ones(hrs_ones),.hrs_tens(hrs_tens), .seg(seg), .an(an));


// setting up constants for clock functionality

parameter display_time = 1'b0;

parameter set_time = 1'b1;

reg current_mode = set_time;


//setting up 1 second increment for clock and adjusting the new set time

    always @(posedge clk_100MHz)

        begin

            case(current_mode)

                display_time: begin


                            if(center_pb)

                                begin

                                    clock_mode <= 0;

                                    current_mode <= set_time;

                                    //Reset variables to prepare for set time mode

                                    counter <= 0;

                                    toggle <= 0;

                                    sec <= 0;

                                end


                            if(counter < max_count)

                                counter <= counter + 1;

                            else

                                begin

                                    counter <= 0;

                                    sec <= sec + 1;

                                end

                end
                            //setting up hours and minutes to set a new time using up-down
buttons

                set_time : begin

                                if(center_pb)
                                    begin
                                        clock_mode <= 1;
                                        current_mode <= display_time;
                                    end

                                //button debouncing code
                                if(counter < (2_50_00_000))
                                    counter <= counter + 1;
                                else
                                    begin

                                        counter <= 0;
                                        case(toggle)
                                            1'b0: begin

                                                    if(up_pb) begin
                                                        min <= min + 1;
                                                    end
                                                    if(down_pb) begin

                                                        if(min>0)begin
                                                            min <= min - 1;
                                                        end
                                                        else if (hrs>1) begin
                                                            hrs <= hrs - 1;
                                                            min <= 59;
                                                        end
                                                        else if(hrs == 1) begin
                                                            hrs <= 12;
                                                            min <= 59;
                                                        end
                                                        end

                                                       // toggle between hrs and min to
set a new time (Right and Left buttons)

                                                       if(left_pb || right_pb) begin
                                                            toggle <= 1;
                                                       end
                                                end // end 1'b0


                                                1'b1: begin

                                                        if(up_pb) begin
                                                            hrs <= hrs + 1;
                                                        end
                                                        if (down_pb) begin
                                                            if(hrs>1) begin
                                                                hrs <= hrs - 1;
                                                            end
                                                            else if (hrs == 1) begin
                                                                hrs <= 12;
                                                            end
                                                        end
                                                        if (right_pb || left_pb ) begin
                                                            toggle <= 0;
                                                        end

                                                end //end 1'b1
                                            endcase // endcase toggle
                                        end

                          end // end get clock
                       endcase // endcase current mode

                   // digital  clock logic
                       if (sec >= 60)begin
```

```
              sec <= 0;
              min <= min + 1;
        end
        if (min >= 60)begin
           min <= 0;
           hrs <= hrs + 1;
        end
        if (hrs >= 24) begin
           hrs <= 0;
        end

// AM/PM time logic

       else begin
           min_ones <= min % 10;
           min_tens <= min / 10;

           if(hrs < 12) begin
               if(hrs == 0) begin
                   hrs_ones <= 2;
                   hrs_tens <= 1;
               end
           else begin
               hrs_ones <= hrs % 10;
               hrs_tens <= hrs / 10;
           end
           pm <= 0;
           end //end hrs < 12

           else begin
               if (hrs == 12) begin
                   hrs_ones <= 2;
                   hrs_tens <= 1;
               end
               else begin
                   hrs_ones <= (hrs - 12) % 10;
                   hrs_tens <= (hrs - 12) / 10;
```

```
                    end
                    pm <= 1;
                end // end hrs >= 12

            end // end clock display


        end
endmodule
```

Seven Segment Module (seven_seg_disp_module)

```
module
seven_seg_disp_module(clk_100MHz,min_ones,min_tens,hrs_ones,hrs_tens,seg,an);

//declaring inputs and outputs for seven segment module
input clk_100MHz;
input [3:0] min_ones, min_tens, hrs_ones, hrs_tens;
output reg [6:0] seg;
output reg [7:0] an;

//declaring wires and registers
reg [1:0] digit_display_ssd = 0;
reg [6:0] display_ssd [3:0];

//variables for generating the slow clock
reg [18:0] counter = 0;
parameter max_count = 5_00_000;  // 100MHz/100Hz = 1M/2 (for low and high pulses) =
5,00,000;
wire [3:0] four_bit_ssd [3:0];

//assigning values that needs to be displayed on the SSD
assign four_bit_ssd[0] = min_ones;
assign four_bit_ssd[1] = min_tens;
assign four_bit_ssd[2] = hrs_ones;
assign four_bit_ssd[3] = hrs_tens;

//generating 100Hz slow clock from the inbuilt 100MHz clock
//to enable each segment of SSD at refresh rate of 10ms

    always @(posedge clk_100MHz)
        begin
            if(counter < max_count)
            counter <= counter+1;
            else begin
            digit_display_ssd <= digit_display_ssd + 1;
            counter <= 0;end

            //BCD to seven segment display
            case(four_bit_ssd[digit_display_ssd])
                4'b0000 : display_ssd[digit_display_ssd] <= 7'b1000000; //0
                4'b0001 : display_ssd[digit_display_ssd] <= 7'b1111001; //1
                4'b0010 : display_ssd[digit_display_ssd] <= 7'b0100100; //2
                4'b0011 : display_ssd[digit_display_ssd] <= 7'b0110000; //3
                4'b0100 : display_ssd[digit_display_ssd] <= 7'b0011001; //4
                4'b0101 : display_ssd[digit_display_ssd] <= 7'b0010010; //5
                4'b0110 : display_ssd[digit_display_ssd] <= 7'b0000010; //6
                4'b0111 : display_ssd[digit_display_ssd] <= 7'b1111000; //7
                4'b1000 : display_ssd[digit_display_ssd] <= 7'b0000000; //8
                4'b1001 : display_ssd[digit_display_ssd] <= 7'b0010000; //9
                4'b1010 : display_ssd[digit_display_ssd] <= 7'b0001000; //10 A
                4'b1011 : display_ssd[digit_display_ssd] <= 7'b0000011; //11 B
                4'b1100 : display_ssd[digit_display_ssd] <= 7'b1000110; //12 C
                4'b1101 : display_ssd[digit_display_ssd] <= 7'b0100001; //13 D
                4'b1110 : display_ssd[digit_display_ssd] <= 7'b0000110; //14 E
                default : display_ssd[digit_display_ssd] <= 7'b0001110; //else, F
            endcase
```

```
//enabling each segment and displaying the digit

            case(digit_display_ssd)
                0: begin
                    an <= 8'b11111110;
                    seg <= display_ssd[0];
                   end
                1: begin
                    an <= 8'b11111101;
                    seg <= display_ssd[1];
                   end
                2: begin
                    an <= 8'b11111011;
                    seg <= display_ssd[2];
                   end
                3: begin
                    an <= 8'b11110111;
                    seg <= display_ssd[3];
                   end
            endcase

        end

endmodule
```

3.3 SETTING UP THE CONSTRAINTS FILE

```
## Clock signal

set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports {
clk_100MHz }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz

#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports
{clk_100MHz}];




## Indicator LED's

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { AM-
PM_indicator_led }]; #IO_L18P_T2_A24_15 Sch=led[0]

set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports {
clock_mode_indicator_led }]; #IO_L24P_T3_RS1_15 Sch=led[1]




##Push buttons

set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { center_pb
}]; #IO_L9P_T1_DQS_14 Sch=btnc

set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { up_pb }];
#IO_L4N_T0_D05_14 Sch=btnu

set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { left_pb
}]; #IO_L12P_T1_MRCC_14 Sch=btnl

set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { right_pb
}]; #IO_L10N_T1_D15_14 Sch=btnr

set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { down_pb
}]; #IO_L9N_T1_DQS_D13_14 Sch=btnd




## Seven Segment Display


#for seven segments

set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports { seg[0]
}]; #IO_L24N_T3_A00_D16_14 Sch=ca

set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports { seg[1]
}]; #IO_25_14 Sch=cb

set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports { seg[2]
}]; #IO_25_15 Sch=cc

set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports { seg[3]
}]; #IO_L17P_T2_A26_15 Sch=cd

set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports { seg[4]
}]; #IO_L13P_T2_MRCC_14 Sch=ce
```

```
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports { seg[5]
}]; #IO_L19P_T3_A10_D26_14 Sch=cf

set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports { seg[6]
}]; #IO_L4P_T0_D04_14 Sch=cg



#for 4 digits of SSD

set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { an[0] }];
#IO_L23P_T3_FOE_B_15 Sch=an[0]

set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { an[1] }];
#IO_L23N_T3_FWE_B_15 Sch=an[1]

set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { an[2] }];
#IO_L24P_T3_A01_D17_14 Sch=an[2]

set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports { an[3] }];
#IO_L19P_T3_A22_15 Sch=an[3]

set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports { an[4] }];
#IO_L8N_T1_D12_14 Sch=an[4]

set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports { an[5] }];
#IO_L14P_T2_SRCC_14 Sch=an[5]

set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { an[6] }];
#IO_L23P_T3_35 Sch=an[6]

set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { an[7] }];
#IO_L23N_T3_A02_D18_14 Sch=an[7]
```

CHAPTER 4: RESULTS AND INFERENCES

4.1 OUTPUT

//NOTE: Some digits might appear not to be lit up while capturing of the photos due to frequency variations. All digits on the right SSD module light up in real time.

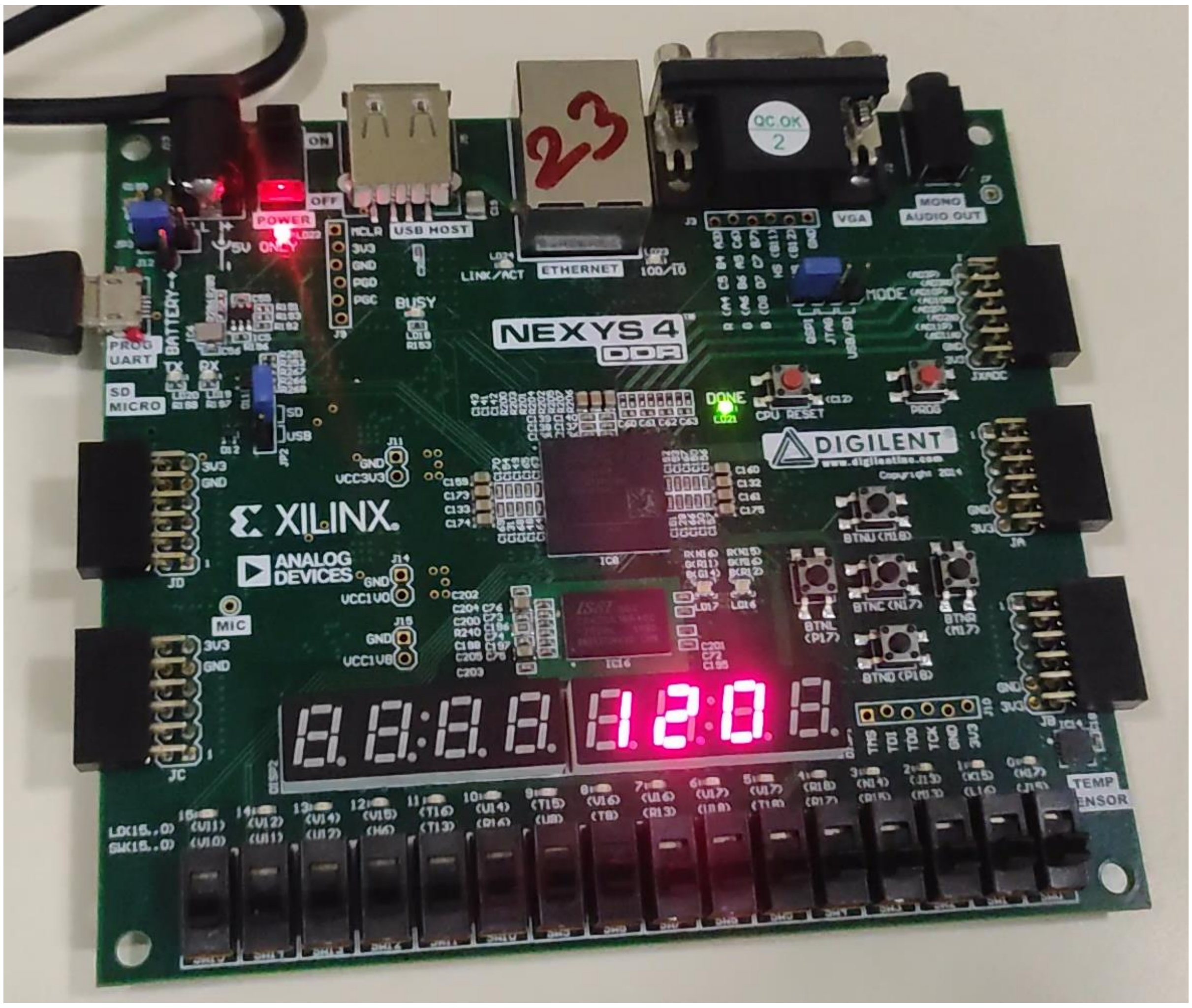

Figure 6: FPGA Output 1

- The clock starts at 12:00 AM initially. In order to set time, the center push button needs to be pressed to enter the set time mode.

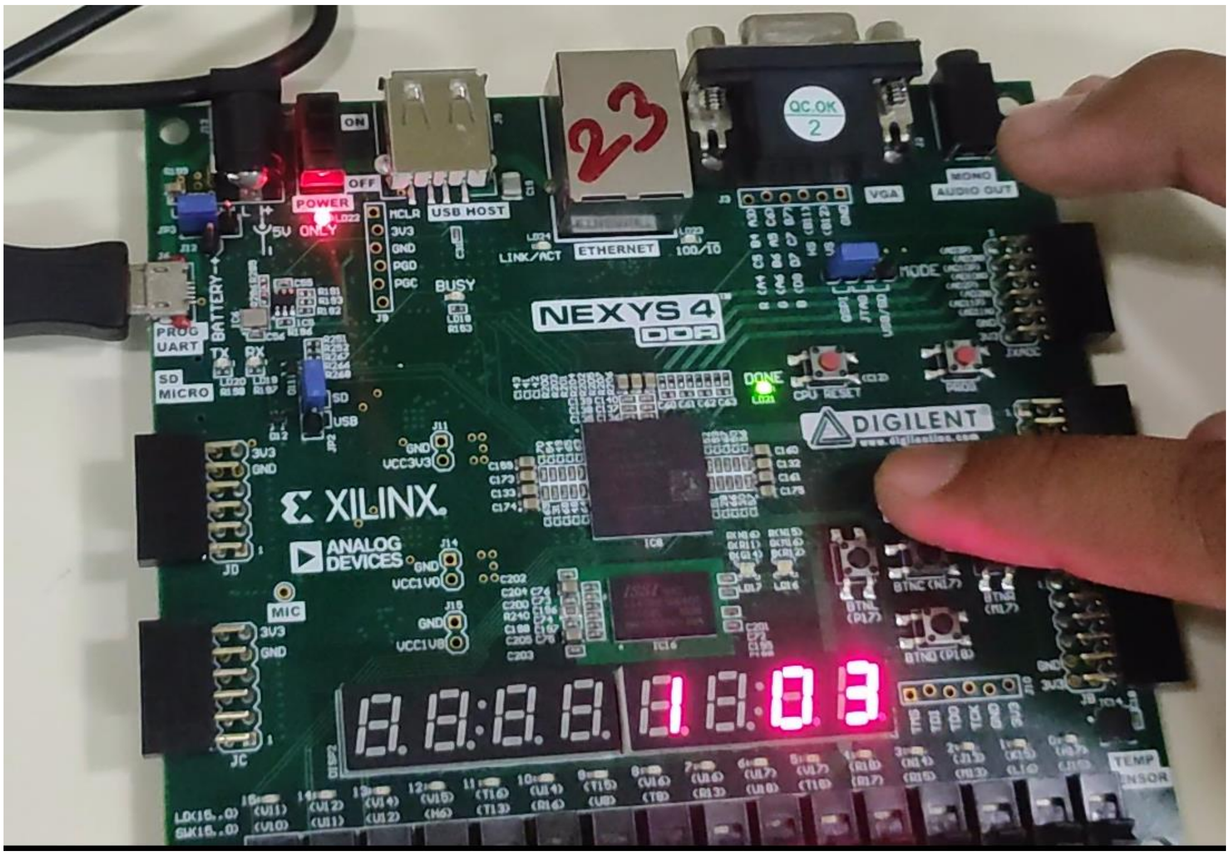

Figure 7: FPGA Output 2

- Up and Down Push button are used to increment and decrement time. By default, the min_ones is incremented or decremented. To set the hour, the user needs to press either the left of the right Push button. To return back to set the minutes value the user needs to again press either left of right button. Left or right button’s functionality is to toggle between the hours and the minutes digits.

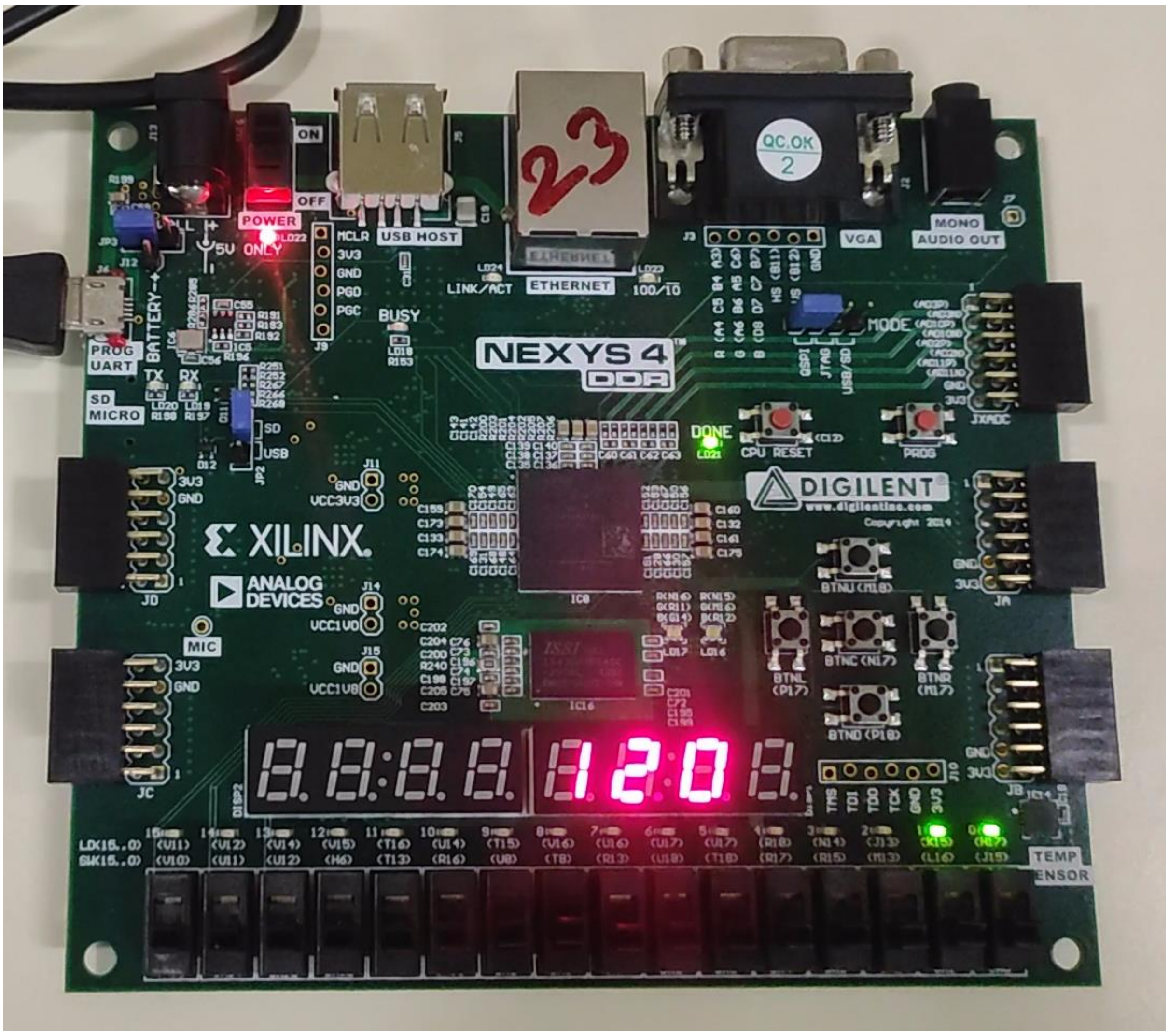

Figure 8: FPGA Output 3

- Once the required time is set by the user, the user again needs to press the center Push button to enter into the clock mode from set time mode which is indicated by clock mode indicator LED (K15). Once the clock reaches at 12:00 PM from 11:59 AM the AM/PM indicator LED (H17) turns ON indicating the noon timings.

CHAPTER 5: CONCLUSIONS

This project is about implementing a 12-hour digital clock on a Nexys 4 DDR FPGA board using Verilog HDL.

By pressing the push buttons on the Nexys 4 DDR FPGA board, you can access various clock functions. Initially, the clock is in clock mode; to set time, press the center push button; to increment / decrement time, hit the up and down push buttons, respectively. To switch between hours and minutes, either the right or left push buttons must be used. To return to the clock mode, the user must press the center push button once again.

Indicator LEDs on pin numbers H17 and K15 indicate the AM/PM time and clock mode, respectively.

Counters, push button debouncing, slow clock, and setting up the seven-segment display, as well as the fundamental concepts of conditional statements, blocking and non-blocking assignments in Verilog HDL, are used to effectively complete this project.

To extend the project, one can convert the clock into an alarm clock and implement it using buzzers on FPGA, which would necessitate more logic in the coding section.

Such digital clocks are particularly useful in modern digital electronics and can be found everywhere, from public locations to everyday use at homes.

CHAPTER 6: REFERENCES

"Counter Design using verilog HDL," GeeksforGeeks. Accessed: Nov. 18, 2023. [Online]. Available: https://www.geeksforgeeks.org/counter-design-using-verilog-hdl/

"digilent-xdc/Nexys-4-DDR-Master.xdc at master · Digilent/digilent-xdc," GitHub. Accessed: Nov. 18, 2023. [Online]. Available: https://github.com/Digilent/digilent-xdc/blob/master/Nexys-4-DDR-Master.xdc

"Nexys 4 DDR Reference Manual - Digilent Reference." Accessed: Nov. 18, 2023. [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-4-ddr/reference-manual

"Seven Segment Displays," GeeksforGeeks. Accessed: Nov. 18, 2023. [Online]. Available: https://www.geeksforgeeks.org/seven-segment-displays/

"Verilog code for debouncing buttons on FPGA." Accessed: Nov. 18, 2023. [Online]. Available: https://www.fpga4student.com/2017/04/simple-debouncing-verilog-code-for.html

"Electronics with Prof. Mughal - YouTube." Accessed: Dec. 09, 2023. [Online]. Available: https://www.youtube.com/@ElectronicswithProfMughal

"Welcome to Real Digital." Accessed: Nov. 18, 2023. [Online]. Available: https://www.realdigital.org/doc/9c21eab4a0f85c50486858a87380d1f6