

Lebanese American University



Digital Systems Lab

Section 33

Laboratory manual 4

Nour Kachmar: 201902597

Youssef Kourani: 202004265

26/10/2023

Table of Contents

List of Figures	2
List of Tables	3
Introduction	4
Experiment 1	5
Module Verilog Code	5
Testbench Verilog Code	7
Simulation Results.....	8
Experiment 2	9
Module Verilog Code	9
Testbench Verilog Code	10
Simulation Results.....	11
Experiment 3	12
Module Verilog Code	12
Testbench Verilog Code	13
Simulation Results.....	14
Experiment 4	15
Module Verilog Code	15
Testbench Verilog Code	17
Simulation Results.....	18
Conclusion.....	19

List of Figures

Figure 1- Snippet of the fireplace controller code	5
Figure 2- Snippet of the fireplace controller testbench	7
Figure 3- Simulation results of the fireplace controller	8
Figure 4- Snippet of BCD code	9
Figure 5- Snippet of BCD testbench	10
Figure 6- Simulation results for BCD	11
Figure 7- Snippet of the two three bits adder code	12
Figure 8- Snippet of the two three bits adder testbench.....	13
Figure 9- Simulation results of two three bits adder	14
Figure 10- Snippet of the digital lock code.....	16
Figure 11- Snippet of digital lock testbench.....	17
Figure 12- Simulation results for Digital Lock.....	18

List of Tables

No table of figures entries found.

Introduction

In this lab, we will create a BCD to 7 segment display decoder, a two three bits adder display decoder, a fireplace controller and a digital lock using only Verilog and then displaying our designs on hardware.

Experiment 1

In this experiment, we are going to design a fireplace controller. This fireplace controller takes three input: S which is a sensor from 0 to 30, T which is a user set target from 0 to 30 and a power switch that is set to ON or OFF. The value of the sensor will be displayed on 2 BCD's as well for the target. Moreover, the fan and the fireplace will be represented by LED lights which will blink when ON and not blink when OFF.

Module Verilog Code

```
fireController_Code - Notepad
File Edit Format View Help
module fireController_Code(
input [4:0] S,          // Temperature sensor input (0-30°C)
input power,           // Main power switch input (0 - OFF, 1 - ON)
input [4:0] T,          // User-set target temperature (0-30°C)
output reg fireplace,   // Fireplace control (0 - OFF, 1 - ON)
output reg fan,         // Fan control (0 - OFF, 1 - ON)
output reg [3:0] Digit1,
output reg [3:0] Digit2,
output reg [3:0] Digit11,
output reg [3:0] Digit22
);

always @(*) begin
// Check if the power is ON and the room temperature is below the target
if (power == 0 || S >= T) begin
fireplace = 0; // Fireplace is OFF
end else begin
fireplace = 1; // Fireplace is ON
end

// Check if the fireplace is ON and the room temperature is over 15°C
if (fireplace == 1 && S > 15) begin
fan = 1; // Fan is ON
end else begin
fan = 0; // Fan is OFF
end
end

always @(*) begin
// Extract the first and second digits of the temperature sensor value
Digit1 = S % 10;
Digit2 = S / 10;

// Display the second sensor digit on the 7-segment display
if (Digit2 == 3) begin
Digit1 = 0;
end

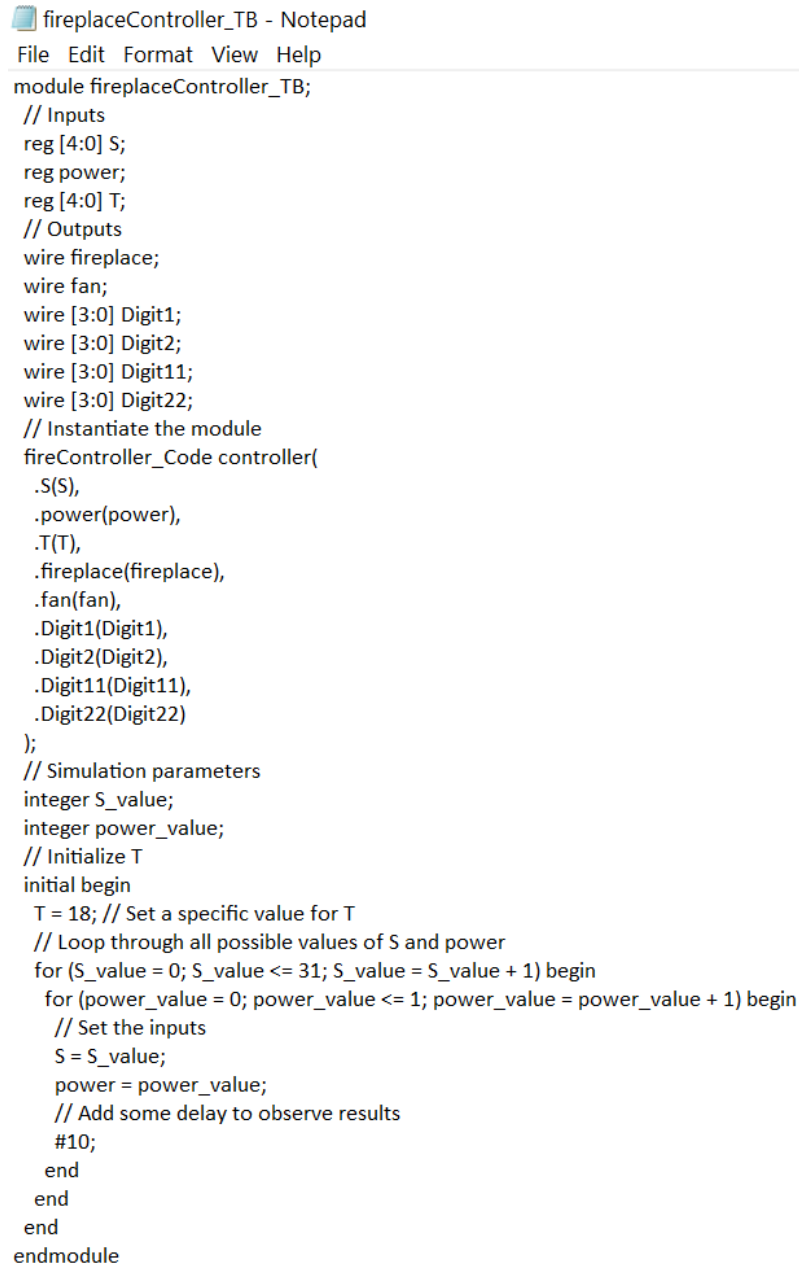
// Extract the first and second digits of the target temperature value
Digit11 = T % 10;
Digit22 = T / 10;

// Display the second target temperature digit on the 7-segment display
if (Digit22 == 3) begin
Digit11 = 0;
end
end
endmodule
```

Figure 1- Snippet of the fireplace controller code

Figure 1 shows the implementation of the fireplace controller design named “fireController_Code”. We have an initial if statement that check if the power is off or the sensor’s value is equal to or has passed the target value. If so, the fireplace turns OFF. Else, the fireplace turns ON. Another if statement is used to check if the fireplace is ON and if the sensor’s value exceeds 15. If so, the fan is turned ON. Otherwise, the fan is turned OFF. We used modulo and division operations to extract each Digit of the Sensor and Target so that we can display each on a BCD; thus having 4 BCD’s which will display the value of when each 2 are combined of S and T. We also force the code to only display values on BCD from 0 to 30 by making sure that if the most significant digit is 3, the second digit will be 0; thus not displaying 31.

Testbench Verilog Code



```
fireplaceController_TB - Notepad
File Edit Format View Help
module fireplaceController_TB;
// Inputs
reg [4:0] S;
reg power;
reg [4:0] T;
// Outputs
wire fireplace;
wire fan;
wire [3:0] Digit1;
wire [3:0] Digit2;
wire [3:0] Digit11;
wire [3:0] Digit22;
// Instantiate the module
fireController_Code controller(
.S(S),
.power(power),
.T(T),
.fireplace(fireplace),
.fan(fan),
.Digit1(Digit1),
.Digit2(Digit2),
.Digit11(Digit11),
.Digit22(Digit22)
);
// Simulation parameters
integer S_value;
integer power_value;
// Initialize T
initial begin
T = 18; // Set a specific value for T
// Loop through all possible values of S and power
for (S_value = 0; S_value <= 31; S_value = S_value + 1) begin
for (power_value = 0; power_value <= 1; power_value = power_value + 1) begin
// Set the inputs
S = S_value;
power = power_value;
// Add some delay to observe results
#10;
end
end
end
endmodule
```

Figure 2- Snippet of the fireplace controller testbench

Figure 2 shows the testbench named “fireplaceController_TB” that was used to simulate the fireplace controller code. It goes over all possible values of S. Note that we chose the target to be 18 for scalability purposes since we cannot go over all the possible targets and show their waveforms. It also alternates the power between 0 and 1 to display the corresponding effects on the outputs.

Simulation Results

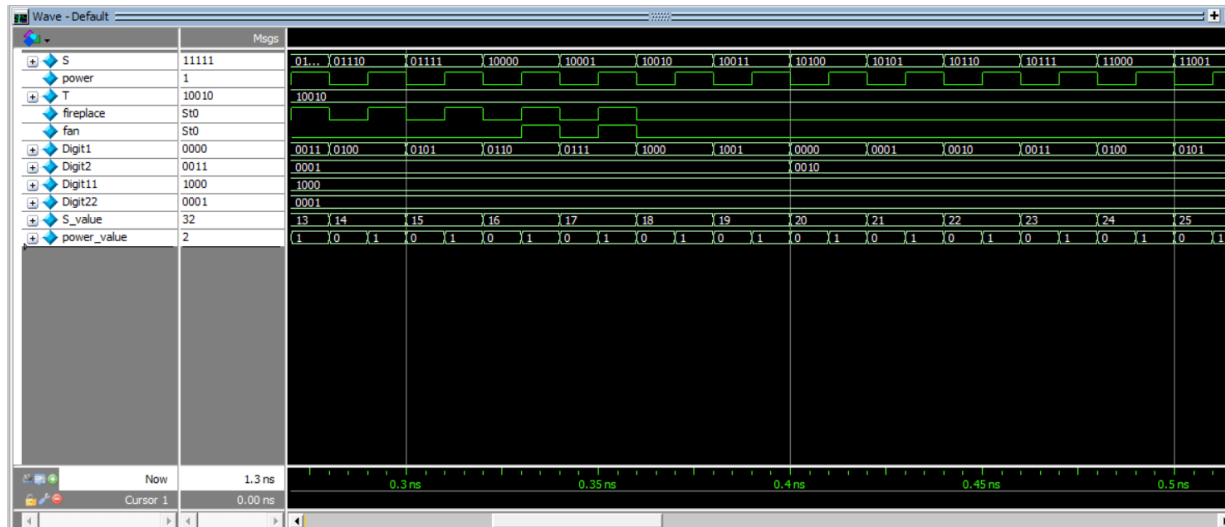


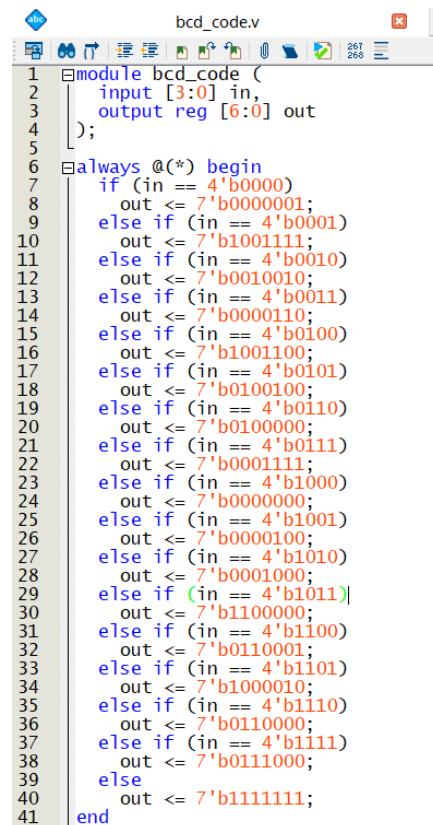
Figure 3- Simulation results of the fireplace controller

Figure 3 shows the waveform of the fireplace controller. In this waveform, we can see that the design we implemented works successfully. For example, when S is greater than the target T which is 18, the fireplace automatically turns OFF. When S is less than the target AND the power is ON, the fireplace turns ON. If the power is OFF, the fireplace is OFF too. Now for the fan, if S is more than 15 and the fireplace is ON, the fan turns ON. However, since our fireplace turns OFF when it hits the target temperature, then the fan and the fireplace will turn OFF. Thus, the fan will only turn ON two times in our case: at S = 16 with power ON and at S = 17 with power ON as shown by our waveform.

Experiment 2

In this experiment, we will create a 7-segment BCD display decoder which takes four inputs. We will use a Verilog code to do so, a test bench to test our values, and finally implement our design on the board.

Module Verilog Code



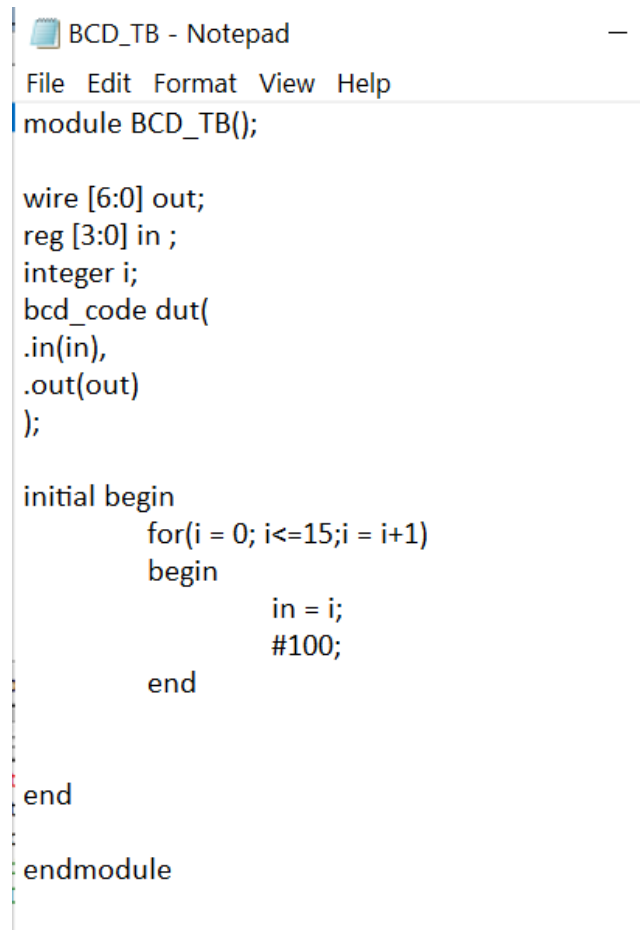
```
1 module bcd_code (
2     input [3:0] in,
3     output reg [6:0] out
4 );
5
6 always @(*) begin
7     if (in == 4'b0000)
8         out <= 7'b0000001;
9     else if (in == 4'b0001)
10        out <= 7'b1001111;
11    else if (in == 4'b0010)
12        out <= 7'b0010010;
13    else if (in == 4'b0011)
14        out <= 7'b0000110;
15    else if (in == 4'b0100)
16        out <= 7'b1001100;
17    else if (in == 4'b0101)
18        out <= 7'b0100100;
19    else if (in == 4'b0110)
20        out <= 7'b0100000;
21    else if (in == 4'b0111)
22        out <= 7'b0001111;
23    else if (in == 4'b1000)
24        out <= 7'b0000000;
25    else if (in == 4'b1001)
26        out <= 7'b0000100;
27    else if (in == 4'b1010)
28        out <= 7'b0001000;
29    else if (in == 4'b1011)
30        out <= 7'b1100000;
31    else if (in == 4'b1100)
32        out <= 7'b0110001;
33    else if (in == 4'b1101)
34        out <= 7'b1000010;
35    else if (in == 4'b1110)
36        out <= 7'b0110000;
37    else if (in == 4'b1111)
38        out <= 7'b0111000;
39    else
40        out <= 7'b1111111;
41 end
```

Figure 4- Snippet of BCD code

Please note that this file was name “bcd_code.v”.

In figure 4, we are simply assigning to each 4 bit input its respective segments that will light on. Note that they are active low due to the configuration of the board.

Testbench Verilog Code



```
BCD_TB - Notepad
File Edit Format View Help
module BCD_TB();

wire [6:0] out;
reg [3:0] in ;
integer i;
bcd_code dut(
.in(in),
.out(out)
);

initial begin
    for(i = 0; i<=15;i = i+1)
    begin
        in = i;
        #100;
    end
end

endmodule
```

Figure 5- Snippet of BCD testbench

Please note that this file was name “BCD_TB.v” in ModelSim.

Figure 5 shows that we created a for loop that assigns input values from 0 to 15 each 100 picoseconds covering all the possible values of our BCD.

Simulation Results

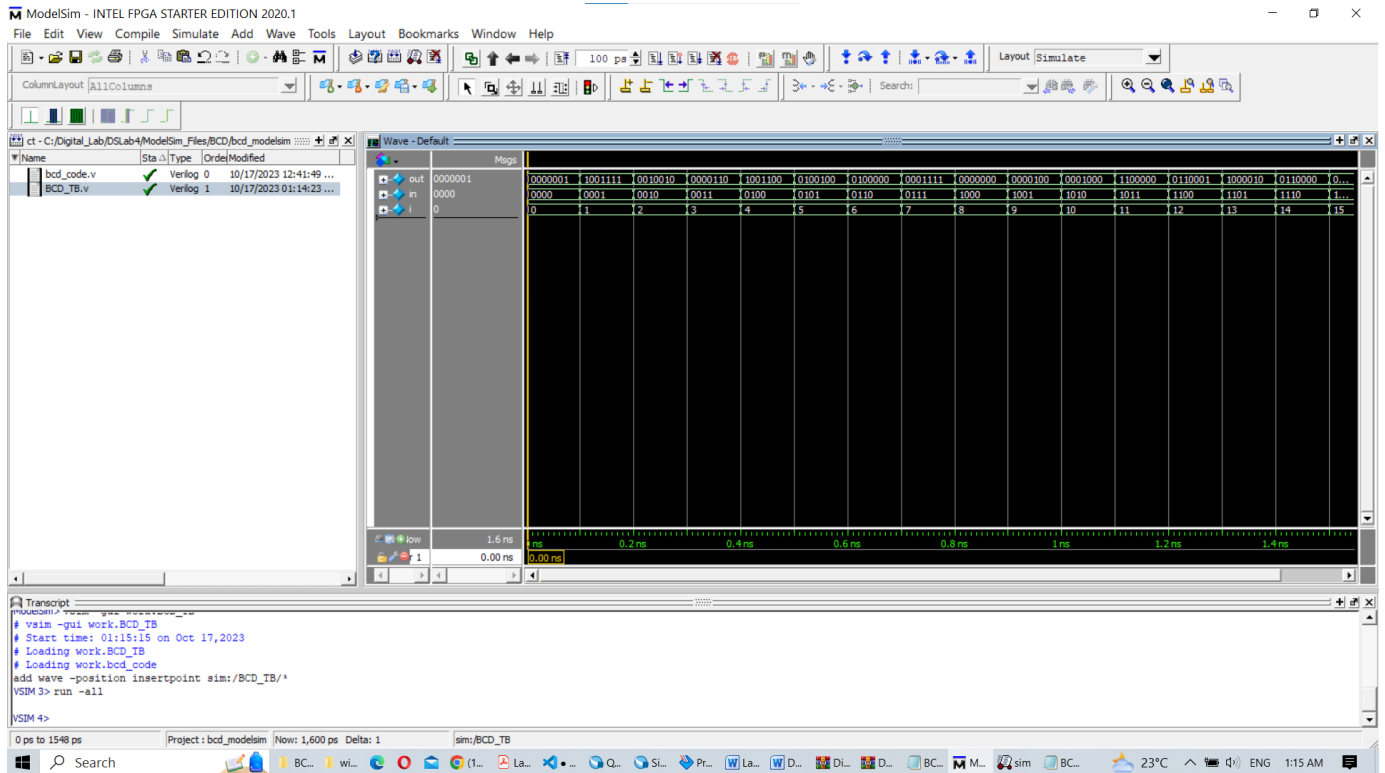


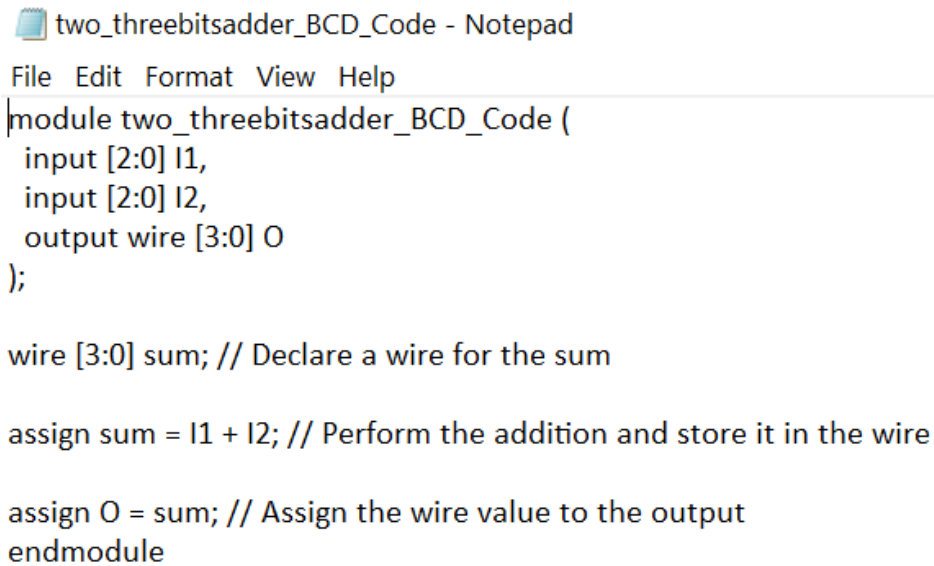
Figure 6- Simulation results for BCD

Figure 6 shows that each input is giving a correct output. For example, when the binary input is 1010, we have all the segments lighting up except segment “d” 0001000 which gives us an A, being the representation of 1010 in hexadecimal.

Experiment 3

Module Verilog Code

In this experiment, we are going to design a two three bits adder while displaying the output of this adder and both inputs each on a BCD 7-segment display decoder respectively.



```
two_threebitsadder_BCD_Code - Notepad
File Edit Format View Help
module two_threebitsadder_BCD_Code (
    input [2:0] I1,
    input [2:0] I2,
    output wire [3:0] O
);

    wire [3:0] sum; // Declare a wire for the sum


    assign sum = I1 + I2; // Perform the addition and store it in the wire

    assign O = sum; // Assign the wire value to the output
endmodule
```

Figure 7- Snippet of the two three bits adder code

Figure 7 shows snippets of the code “two_threebitsadder_BCD_Code” that was used to simulate the two three bits adder. It simply adds to inputs and stores it in an output O.

Testbench Verilog Code

 two_threebitsadder_BCD_TB - Notepad
File Edit Format View Help

```
module two_threebitsadder_BCD_TB;

    // Inputs
    reg [2:0] I1;
    reg [3:0] I2;

    // Outputs
    wire [3:0] O;

    // Instantiate the module under test
    two_threebitsadder_BCD_Code uut (
        .I1(I1),
        .I2(I2),
        .O(O)
    );

    // Testbench stimulus
    initial begin : stimulus
        // Loop through all possible input combinations
        integer i, j;
        for (i = 0; i <= 7; i = i + 1) begin
            for (j = 0; j <= 7; j = j + 1) begin
                I1 = i;
                I2 = j;

                // Wait a few time units
                #10;
            end
        end
    end

endmodule
```

Figure 8- Snippet of the two three bits adder testbench

Figure 8 shows the snippet for the testbench “two_threebitsadder_BCD_TB” for the two three bits adder. It covers all the possible input variations of I1 and I2 the digits that should be displayed on the two BCD pairs.

Simulation Results

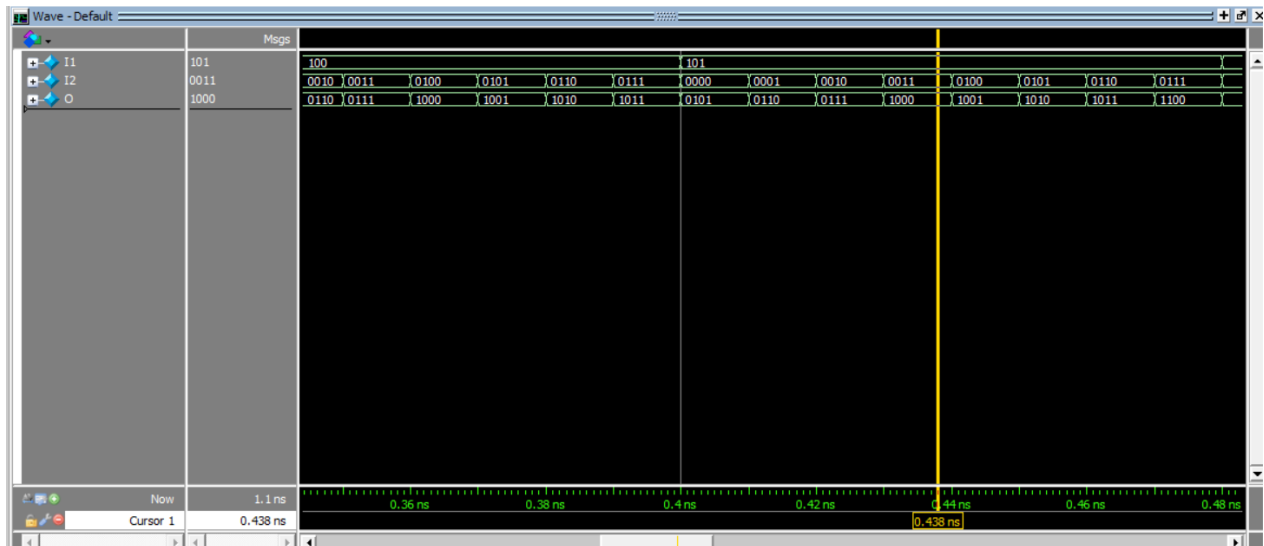


Figure 9- Simulation results of two three bits adder

Figure 9 shows multiple input variations for I1 and I2 with the corresponding output on each BCD. Note that we could not cover all the cases so we provided a random snippet.

Experiment 4

In this lab experiment, we designed a digital lock counter. This counter counts 4 consecutive 0's or 1's while displaying the counters' value on a BCD and setting F to 1 when the 4 consecutive 0's or 1's are counted.

Module Verilog Code

```
DigitalLock_code - Notepad
File Edit Format View Help
module DigitalLock_code(
    input sys_clk,
    input reset,
    input w,
    output reg F,
    output reg [3:0] state,
    output wire[2:0] led_out
);

parameter idle_state = 0;
parameter S1 = 1;
parameter S2 = 2;
parameter S3 = 3;
parameter S4 = 4;
parameter S5 = 5;
parameter S6 = 6;
parameter S7 = 7;
parameter S8 = 8;

wire f_out_ex;

assign f_out_ex = (state == S4) ? 1 : (state == S8) ? 1 : 0;
// reg [3:0] state;
assign led_out = (state>S4)?{state[3],state[1:0]}:state[2:0];
always @(posedge sys_clk)
begin
    if (reset)
    begin
        state <= idle_state;
        F <= 0;
    end else
    begin
        case(state)
            idle_state: begin
                F <= 0;
                if (w)
                    state <= S5;
                else
                    state <= S1;
            end
            S1 : begin
                F <= 0;
                if (w)
                    state <= S5;
                else
                    state <= S2;
            end
        end
    end
end
```



```

DigitalLock_code - Notepad
File Edit Format View Help

        else
            state <= S5;
        else
            state <= S2;
        end
    S2 : begin
        F <= 0;
        if (w)
            state <= S5;
        else
            state <= S3;
        end
    S3 : begin
        F <= 0;
        if (w)
            state <= S5;
        else
            state <= S4;
        end
    S4 : begin
        F <= 1;
        if (w)
            state <= S5;
        else
            state <= S4;
        end
    S5 : begin
        F <= 0;
        if (w)
            state <= S6;
        else
            state <= S1;
        end
    S6 : begin
        F <= 0;
        if (w)
            state <= S7;
        else
            state <= S1;
        end
    S7 : begin
        F <= 0;
        if (w)
            state <= S8;
        else
            state <= S1;
        end
    S8 : begin
        F <= 1;
        if (w)
            state <= S8;
        end
    endcase
end
end

        S8 : begin
            F <= 1;
            if (w)
                state <= S8;
            else
                state <= S1;
            end
        default: begin
            F <= 0;
            state <= idle_state;
        end
    end
endmodule

```

Figure 10- Snippet of the digital lock code

Figure 10 shows the snippet of the code. By using the state diagram, we were able to do a step by step construction of the code using cases for the possible states.

Testbench Verilog Code

```
*DigitalLock_TB - Notepad
File Edit Format View Help
module DigitalLock_TB();

    reg sys_clk;
    reg sys_reset;
    reg w;

    wire f_out;
    wire [3:0] bcd_in;
    wire [2:0] led_out;
    parameter clock_period = 10;

    initial
        sys_clk <= 0;

    always #5 sys_clk <= ~sys_clk;

    initial begin
        w <= 0;
        sys_reset <= 1;
        #(clock_period*5)
        sys_reset <= 0;
        #(clock_period*3)
        w <= 1;
        #(clock_period*6)
        w <= 0;
    end

    DigitalLock_code digital_lock_inst (
        .sys_clk (sys_clk),
        .reset   (sys_reset),
        .w       (w),
        .F       (f_out),
        .state   (bcd_in),
        .led_out (led_out)
    );

endmodule
```

Figure 11- Snippet of digital lock testbench

Figure 11 shows the snippet testbench of the code.

Simulation Results

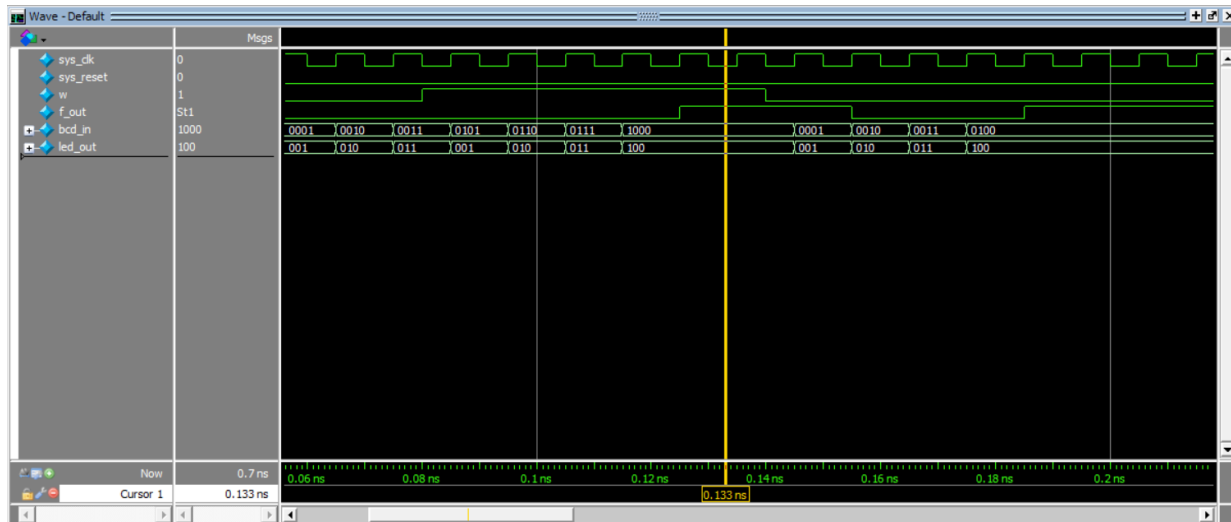


Figure 12- Simulation results for Digital Lock

Figure 12 shows a waveform that shows our digital lock. As we can see, all of our simulation seems to be correct.

Conclusion

In this lab, we created a BCD display decoder, a two three bits adder display decoder, a fireplace controller and a digital lock only using Verilog without going into any circuitry on Quartus.