Lab 1: FPGA-based Mental Binary Math Game

Mubashar Khan

ECE5440

# Introduction

In this lab, we design and build a 2-player binary mental math game using the DE2-115 FPGA board. The game functions in a turn-based system with Player 1 choosing a 4-bit binary number and inputting it using a set of toggle switches. That number is then displayed on a 7-segment display which can display the terms 0-F in hexadecimal. Player 2 can look at the number being displayed on the 7-segment display and mentally should convert it to binary. Then Player 2 should enter a number using their 4-bit toggle switches such that the sum of Player 1’s and Player 2’s 4-bit binary numbers add to 1111 in binary. Player 2’s 4-bit binary input will also be displayed using a second 7-segment display. The FPGA will compute the sum of both players numbers and display the sum on a third 7-segment display. If the third 7-segment display shows ‘F’, signifying 1111 in hexadecimal, Player 2 wins the round and gains 1 point. Players should switch roles each round and record their scores. After reaching a predetermined number of rounds, the player with the most points is the winner. For additional challenge, players can impose a time limit on Player 2’s turn, forcing Player 2 to have to think quickly.

# System Architecture Design

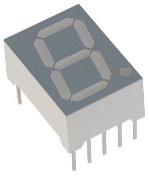
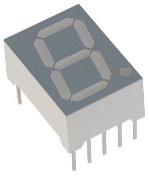
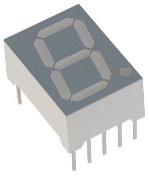
The diagram on the figure 1 shows the top-level system architecture for the binary mental math game. Each player will have their own set of 4 toggle switches to input their 4-bit binary numbers. The game uses a 7-segment display which displays a number based on a 4-bit binary input. This is accomplished using a 4-16 decoder. The decoder takes the 4-bit input and converts it to a hexadecimal value. That value is then displayed on a 7-segment display. When both players have entered a value into the FPGA, the sum is found using a binary adder. The adder takes two 4-bit inputs and outputs a 4-bit value. We do not care about the overflow bit and any sums found that are over binary 1111 will wrap around and start counting from binary 0000. The output from the adder is passed onto the 4-16 decoder which converts the sum to a hexadecimal value that can be displayed on a 7-segment display.

FPGA: DE2-115

7-Segment Display 3

7-Segment Display 2

7-Segment Display 1



4-16 Decoder

4-16 Decoder

4-16 Decoder

4-bit Adder

Toggle Switches

Player 1

Player 2

**Figure 1.** Top-level system architecture for the binary mental math game. The DE2-115 FGPA is used to take two 4-bit inputs,

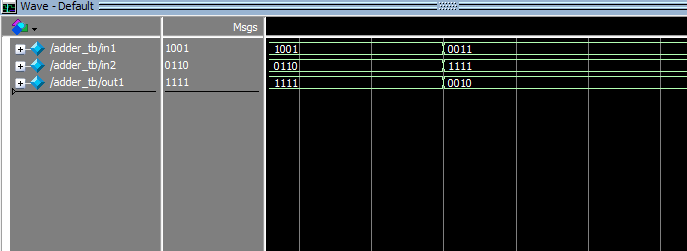
convert them to hexadecimal using a 4-16 decoder for displaying on a 7-segment display and sum them using a 4-bit adder.

The sum is then converted to hexadecimal for displaying.

# Simulation Results

4-bit Adder Test:

Figure 2 shows the simulation done to test the adder Verilog file. Results show that the adder is working as intended.



**Figure 2.** 4-Bit Adder Simulation. Using two sets of inputs, this simulation shows the respective outputs. By doing binary addition, we can see that the output shows the correct sum for both tests.

4-16 decoder: 7-segment display:

Figure 3 shows the simulation done to test the 7-segment display. The LEDs on the 7-segment display are active-low meaning an output of 0 for a bit enables that bit.

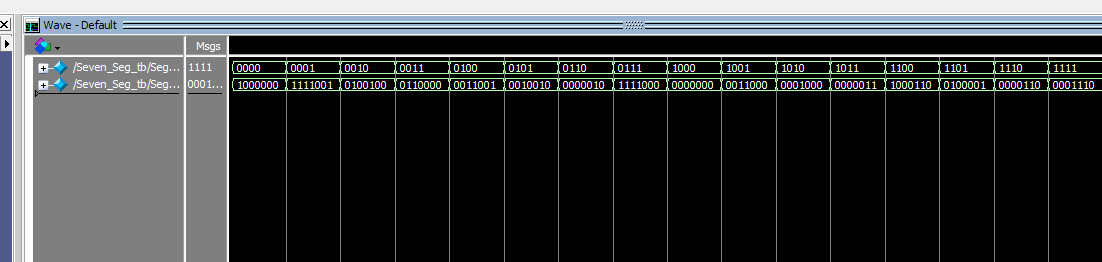


Figure 3. 4-16 Decoder Simulation. This simulation shows all possible outputs using the full range of a 4-bit input.

# FPGA Board Testing Results

The following images show the DE2-115 FPGA board with the Quartus file uploaded to it. The leftmost 7-segment display is connected to switches 14-17 which is where Player 1 provides inputs. The second 7-segment display is connected to switches 0-3 which is where Player 2 provides their inputs.

A circuit board

Description automatically generated

A circuit board

Description automatically generated

A circuit board

Description automatically generated

A circuit board

Description automatically generated

A circuit board

Description automatically generated

We can see through these pictures that the 4-bit binary adder is working and that inputs are properly being converted from their 4-bit form to their hexadecimal equivalent.

# Video Demo

The following link is to a video showing the operation of binary mental math game.

<https://drive.google.com/file/d/1CQv7KU_kPwXUhqRWP6rJahYrf8mgxPv_/view?usp=sharing>

# Conclusion

Lab 1 had us design and build a binary mental math game. The game is a 2-player game that can be used to practice and test a person’s ability to convert numbers between hexadecimal and binary. Additionally, the game tests a person’s ability to add in binary. The goal of the game is for Player 2 to determine which binary number must be added to Player 1’s input in order to reach binary 1111.

# Appendix

The following is all the code created for this lab. It consists of 5 modules, Lab1\_KHAN\_M, Seven\_Seg.v, Seven\_Seg\_tb.v, adder.v, and adder\_tb.v.

// ECE 5440/6370

// Author: Mubashar Khan, 2949

// Lab1\_KHAN\_M

// This module in the top code that calls all necessary functions

// Its function is to take thhe 4 bit input from players and pass those values to the adder and decoder functions.

// The returned value from the adder function is also passed to a decoder and all 3 decoder values are used to enable the proper pins on the 7-segment display.

//

module Lab1\_KHAN\_M (

//inputs

switch\_num1, switch\_num2,

//outputs

seg7\_num1, seg7\_num2, seg7\_sum);

input [3:0] switch\_num1, switch\_num2;

output [6:0] seg7\_num1, seg7\_num2, seg7\_sum;

wire [3:0] sum;

adder adder\_1 (switch\_num1, switch\_num2, sum);

Seven\_Seg seg7\_number1 (switch\_num1, seg7\_num1);

Seven\_Seg seg7\_number2 (switch\_num2, seg7\_num2);

Seven\_Seg seg7\_summation (sum, seg7\_sum);

Endmodule

// Seven\_Seg

// This module is the code for the 4-16 decoder.

// Its function is to take a 4-bit input and convert it to a 7-segment number that can be displayed on a 7-segment display

//

module Seven\_Seg (Seg\_in, Seg\_out);

input [3:0] Seg\_in;

output [6:0] Seg\_out;

reg [6:0] Seg\_out;

always @ (Seg\_in)

begin

case(Seg\_in)

4'b0000 : begin Seg\_out = 7'b1000000; end //0 //7'b1000000; end 7'b0000001; end

4'b0001 : begin Seg\_out = 7'b1111001; end //1 //7'b1111001; end 7'b1001111; end

4'b0010 : begin Seg\_out = 7'b0100100; end //2 //7'b0100100; end 7'b0010010; end

4'b0011 : begin Seg\_out = 7'b0110000; end //3 //7'b0110000; end 7'b0000110; end

4'b0100 : begin Seg\_out = 7'b0011001; end //4 //7'b0011001; end 7'b1001100; end

4'b0101 : begin Seg\_out = 7'b0010010; end //5 //7'b0010010; end 7'b0100100; end

4'b0110 : begin Seg\_out = 7'b0000010; end //6 //7'b0000010; end 7'b0100000; end

4'b0111 : begin Seg\_out = 7'b1111000; end //7 //7'b1111000; end 7'b0001111; end

4'b1000 : begin Seg\_out = 7'b0000000; end //8 //7'b0000000; end 7'b0000000; end

4'b1001 : begin Seg\_out = 7'b0011000; end //9 //7'b0011000; end 7'b0001100; end

4'b1010 : begin Seg\_out = 7'b0001000; end //A //7'b0001000; end 7'b0001000; end

4'b1011 : begin Seg\_out = 7'b0000011; end //b //7'b0000011; end 7'b1100000; end

4'b1100 : begin Seg\_out = 7'b1000110; end //C //7'b1000110; end 7'b0110001; end

4'b1101 : begin Seg\_out = 7'b0100001; end //d //7'b0100001; end 7'b1000010; end

4'b1110 : begin Seg\_out = 7'b0000110; end //E //7'b0000110; end 7'b0110000; end

4'b1111 : begin Seg\_out = 7'b0001110; end //F //7'b0001110; end 7'b0111000; end

default : begin Seg\_out = 7'b1000000; end //0 //7'b1000000; end 7'b0000001; end

endcase

end

endmodule

// Seven\_Seg\_tb

// This module is the test bench for the 4-16 decoder

// Its function is to be used to test sample inputs and observe their outputs.

//

`timescale 10ns/100ps

module Seven\_Seg\_tb ();

reg[3:0] Seg\_in;

wire[6:0] Seg\_out;

Seven\_Seg DUTSeven\_Seg(Seg\_in,Seg\_out);

initial

begin

Seg\_in = 4'b0000; // 0

#10;

Seg\_in = 4'b0001; // 1

#10;

Seg\_in = 4'b0010; // 2

#10;

Seg\_in = 4'b0011; // 3

#10;

Seg\_in = 4'b0100; // 4

#10;

Seg\_in = 4'b0101; // 5

#10;

Seg\_in = 4'b0110; // 6

#10;

Seg\_in = 4'b0111; // 7

#10;

Seg\_in = 4'b1000; // 8

#10;

Seg\_in = 4'b1001; // 9

#10;

Seg\_in = 4'b1010; // A

#10;

Seg\_in = 4'b1011; // b

#10;

Seg\_in = 4'b1100; // C

#10;

Seg\_in = 4'b1101; // d

#10;

Seg\_in = 4'b1110; // E

#10;

Seg\_in = 4'b1111; // F

end

endmodule

// adder

// This module adds 2 four-bit inputs. We do not care about overflow and the output is also four-bits.

//

module adder (in1, in2, sum);

input [3:0] in1, in2;

output [3:0] sum;

reg [3:0] sum;

always @ (in1, in2)

begin

sum = in1 + in2;

end

endmodule

// adder\_tb

// This module is the test bench for the adder

// Its function is to be used to test sample inputs and observe their outputs.

//

`timescale 10ns/100ps

module adder\_tb ();

reg [3:0] in1, in2;

wire [3:0] out1;

adder DUT\_adder(in1, in2, out1);

initial

begin

in1 = 4'b1001;

in2 = 4'b0110;

#10;

in1 = 4'b0011;

in2 = 4'b1111;

end

endmodule

The following is all the code created for Lab 1. It consists of 5 modules, Lab1\_KHAN\_M, Seven\_Seg.v, Seven\_Seg\_tb.v, adder.v, and adder\_tb.v.

// ECE 5440/6370

// Author: Mubashar Khan, 2949

// Lab1\_KHAN\_M

// This module in the top code that calls all necessary functions

// Its function is to take thhe 4 bit input from players and pass those values to the adder and decoder functions.

// The returned value from the adder function is also passed to a decoder and all 3 decoder values are used to enable the proper pins on the 7-segment display.

//

module Lab1\_KHAN\_M (

//inputs

switch\_num1, switch\_num2,

//outputs

seg7\_num1, seg7\_num2, seg7\_sum);

input [3:0] switch\_num1, switch\_num2;

output [6:0] seg7\_num1, seg7\_num2, seg7\_sum;

wire [3:0] sum;

adder adder\_1 (switch\_num1, switch\_num2, sum);

Seven\_Seg seg7\_number1 (switch\_num1, seg7\_num1);

Seven\_Seg seg7\_number2 (switch\_num2, seg7\_num2);

Seven\_Seg seg7\_summation (sum, seg7\_sum);

Endmodule

// Seven\_Seg

// This module is the code for the 4-16 decoder.

// Its function is to take a 4-bit input and convert it to a 7-segment number that can be displayed on a 7-segment display

//

module Seven\_Seg (Seg\_in, Seg\_out);

input [3:0] Seg\_in;

output [6:0] Seg\_out;

reg [6:0] Seg\_out;

always @ (Seg\_in)

begin

case(Seg\_in)

4'b0000 : begin Seg\_out = 7'b1000000; end //0 //7'b1000000; end 7'b0000001; end

4'b0001 : begin Seg\_out = 7'b1111001; end //1 //7'b1111001; end 7'b1001111; end

4'b0010 : begin Seg\_out = 7'b0100100; end //2 //7'b0100100; end 7'b0010010; end

4'b0011 : begin Seg\_out = 7'b0110000; end //3 //7'b0110000; end 7'b0000110; end

4'b0100 : begin Seg\_out = 7'b0011001; end //4 //7'b0011001; end 7'b1001100; end

4'b0101 : begin Seg\_out = 7'b0010010; end //5 //7'b0010010; end 7'b0100100; end

4'b0110 : begin Seg\_out = 7'b0000010; end //6 //7'b0000010; end 7'b0100000; end

4'b0111 : begin Seg\_out = 7'b1111000; end //7 //7'b1111000; end 7'b0001111; end

4'b1000 : begin Seg\_out = 7'b0000000; end //8 //7'b0000000; end 7'b0000000; end

4'b1001 : begin Seg\_out = 7'b0011000; end //9 //7'b0011000; end 7'b0001100; end

4'b1010 : begin Seg\_out = 7'b0001000; end //A //7'b0001000; end 7'b0001000; end

4'b1011 : begin Seg\_out = 7'b0000011; end //b //7'b0000011; end 7'b1100000; end

4'b1100 : begin Seg\_out = 7'b1000110; end //C //7'b1000110; end 7'b0110001; end

4'b1101 : begin Seg\_out = 7'b0100001; end //d //7'b0100001; end 7'b1000010; end

4'b1110 : begin Seg\_out = 7'b0000110; end //E //7'b0000110; end 7'b0110000; end

4'b1111 : begin Seg\_out = 7'b0001110; end //F //7'b0001110; end 7'b0111000; end

default : begin Seg\_out = 7'b1000000; end //0 //7'b1000000; end 7'b0000001; end

endcase

end

endmodule

// Seven\_Seg\_tb

// This module is the test bench for the 4-16 decoder

// Its function is to be used to test sample inputs and observe their outputs.

//

`timescale 10ns/100ps

module Seven\_Seg\_tb ();

reg[3:0] Seg\_in;

wire[6:0] Seg\_out;

Seven\_Seg DUTSeven\_Seg(Seg\_in,Seg\_out);

initial

begin

Seg\_in = 4'b0000; // 0

#10;

Seg\_in = 4'b0001; // 1

#10;

Seg\_in = 4'b0010; // 2

#10;

Seg\_in = 4'b0011; // 3

#10;

Seg\_in = 4'b0100; // 4

#10;

Seg\_in = 4'b0101; // 5

#10;

Seg\_in = 4'b0110; // 6

#10;

Seg\_in = 4'b0111; // 7

#10;

Seg\_in = 4'b1000; // 8

#10;

Seg\_in = 4'b1001; // 9

#10;

Seg\_in = 4'b1010; // A

#10;

Seg\_in = 4'b1011; // b

#10;

Seg\_in = 4'b1100; // C

#10;

Seg\_in = 4'b1101; // d

#10;

Seg\_in = 4'b1110; // E

#10;

Seg\_in = 4'b1111; // F

end

endmodule

// adder

// This module adds 2 four-bit inputs. We do not care about overflow and the output is also four-bits.

//

module adder (in1, in2, sum);

input [3:0] in1, in2;

output [3:0] sum;

reg [3:0] sum;

always @ (in1, in2)

begin

sum = in1 + in2;

end

endmodule

// adder\_tb

// This module is the test bench for the adder

// Its function is to be used to test sample inputs and observe their outputs.

//

`timescale 10ns/100ps

module adder\_tb ();

reg [3:0] in1, in2;

wire [3:0] out1;

adder DUT\_adder(in1, in2, out1);

initial

begin

in1 = 4'b1001;

in2 = 4'b0110;

#10;

in1 = 4'b0011;

in2 = 4'b1111;

end

endmodule