David Zhang

2213233

Lab 1: FPGA-based Mental Binary Math Game

ECE6370

# Introduction

In this lab, a mental binary game was implemented onto the Altera DE0-CV FPGA. The mental binary game consists of a player trying to match their opponent’s number such that the resultant sum would come out as 15. Players set their numbers in binary using the 4 slide switches located on the bottom of the FPGA, lighting up their respective 7-segment display with the hex number that they input. The mental aspect of the game comes from the second player having to convert the number that complements their opponent’s number into binary and entering that into their input area. A figure illustrating the play area is depicted in Figure 1.

Diagram

Description automatically generated

Figure : Illustration of Play Area and Controls

# System architecture design

Diagram, schematic

Description automatically generated

Figure : Top-Level System Architecture Model

Figure 2 above depicts the system architecture at a module level. Each player’s switches go through a 4-bit data bus into decoder modules that decode the data stream into 16 total binary combinations corresponding to 0 through F on a 7-segment display. This data output stream is carried through a 7-bit data bus—each bit representing a segment on the 7-segment display to be lit. In addition, each players data 4-bit data stream is fed into a 4-bit Adder module that adds the two binary numbers together while rejecting the carryover digit in case of overflow. The output of this 4-bit adder is also a 4-bit data stream, which is fed into another decoder module to be displayed as in Figure 1. In addition, combinational logic was implemented to light up the left most LED should the output sum be 15, and the rightmost LED should it not.

# simulation results

Each module was tested individually using a custom testbench designed for it in ModelSim. The module and the testbench for the 7-segment decoder can be found in Appendix A.

For efficacy, only certain essential cases were tested. Any cases not tested were identical to one of the cases being tested and were lower priority. For the 7-segment decoder, these essential cases were a display of 0, 5, and F. As long as the combinational logic for these cases were correct, all other cases were derivative. The resultant waveforms for these cases when their corresponding input is given is shown in Figures 3, 4 and 5.

Graphical user interface, application

Description automatically generated

Figure : Output Waveform for Display "0"

Graphical user interface, application

Description automatically generated

Figure : Output Waveform for Display "5"

Graphical user interface

Description automatically generated

Figure : Output Waveform for Display "F"

Since a signal-low corresponds to a lit segment on the 7-segment display, with bit 0 corresponding to the topmost segment, each number to be displayed requires a 7-bit binary number to be displayed correctly. For 0, with only bit-6 being unlit we would expect an output binary number 1000000. We see from Figure 3 that this prediction is correct, and the same is true for the other test cases. 5 would be 0010010, which we see in Figure 4, and F would be 0001110 which we see in Figure 5.

For the 4-bit Adder, module definition for the base module and its testbench can be found in Appendix B. For test cases, the same paradigm as the 7-segment decoder was used wherein essential cases were tested from which the rest could be inferred. The most important test cases were adding two example numbers together that do not equal 15, adding two numbers together that equal 15, and adding two numbers together that would overflow and roll back to a smaller digit. The resultant waveforms for these cases are given in Figures 6 through 8.

Graphical user interface

Description automatically generated

Figure : Output Waveform of 1 + 2

Graphical user interface

Description automatically generated

Figure : Output Waveform of 12+3

Graphical user interface

Description automatically generated

Figure : Output Waveform of 12 + 7

For adding two numbers together that do not equal 15, an arbitrary addition of 1 and 2 was used to test the resultant output. The expected sum should be 3, or 0011 in binary. As can be seen from Figure 6, this prediction is true. Figure 7 is much of the same, with an addition of 12 and 3 resulting to 15, or 1111 in binary which we see. The last case is more complicated, as the way we have implemented the adder module results in no carryover digit should the addition overflow into 5 bits. We ignore the carryover bit, and the overflow should roll back to a smaller digit. Here, an addition of 12 and 7 was tested and the result should roll back 5 instead of logically adding to 19. Since the binary output is 0101, we see that this is correct.

# FPga Board testing results

Depicted in the following figures are the FPGA working with the top-level module downloaded onto it. For visual inspection, as before only 3 essential cases were documented but all were tested to make sure that segments and addition was done correctly.

A picture containing text, indoor, electronics

Description automatically generated

Figure : Baseline Startup

A picture containing text

Description automatically generated

Figure : Example Sum

A picture containing text

Description automatically generated

Figure : Correct Sum with Bonus Feature Implemented

A picture containing text, indoor

Description automatically generated

Figure : Incorrect Sum with Wraparound Sum

As all these cases work correctly, Figure 11 can be used to explain the logic behind all four figures. Player 2’s inputs are 1000, indicated by only slide switch 9 (SW9) being flipped on. Player 1’s inputs are 0111, again indicated by SW2, SW1, and SW0 being flipped. Logically this should result in a sum of 15, or F in hex which we see in the middle display. Additionally, since the bonus feature was implemented this is the only case in which the left-most LED is lit instead of the right. The other figures simply provide more proof that all modules are implemented correctly, including edge cases like wraparound.

# video demo

<https://drive.google.com/file/d/1u2gGTtLYSelw7tTzJdea2Wj8T81DssAQ/view?usp=sharing>

# Conclusion

This lab was intended to provide an introduction to basic combinatorial logic as well as provide a primer on system architecture for modules. All components work as needed, including the bonus feature that was implemented. To avoid creating an unnecessary module for processing a binary decision, a simple if/else case can be used within the module definition to light up the left most LED when the sum is 1111, and the right most LED in every other case.

# appendix A: verilog code for 7-segment decoder

Graphical user interface, text, application

Description automatically generated

Figure : Module Definition for 7-Segment Decoder

Graphical user interface, text, application

Description automatically generated

Figure : Module Definition for Decoder Testbench

# appendix B: Verilog Code for 4-bit adder

Graphical user interface, text, application

Description automatically generated

Figure : Module Definition for 4-Bit Adder

Graphical user interface, text, application

Description automatically generated

Figure : Module Definition for Adder Testbench

# appendix C: Verilog code for top level module

Graphical user interface, text, application

Description automatically generated

Figure : Top-Level Module Definition