The Casino Games Board (CPE 233 Project)

Name: Kelechi Igwe

(Note: Go to Page 5 to Read Update on Completion of Project)

Overview

When I was in a class in high school, I had a lot of free time but nothing to do with it. However, once I learned about the programming feature included in TI-83 calculators, my free time was constantly consumed by a program I made where I could play different betting games to increase the virtual money I had on the calculator. For my first Basys3 board project at Cal Poly, I made a synthesizer that allowed a person to play different square waves on the board. However, for this project I wanted to make a game on the Basys3 board. Because we worked on the OTTER the whole quarter and learned how to write assembly code, I decided that I could try to recreate the game I made on my calculator and do that for my CPE 233 project. Therefore, for my final project for CPE 233, I chose to create a casino games program that uses a Basys3 Board to allow a player to use virtual money for multiple games that can be seen at a casino.

The casino games program was built using assembly code and is ran through the OTTER, which is an implementation of the RISC-V instruction set architecture (RV32I). The OTTER, shown in the Figure A below, was given to me and my classmates by our professor, Dr. Joseph Callenes. The architecture includes several modules, including a control unit, PC, register file, two memory modules, and an ALU, which are shown in Figure B below.

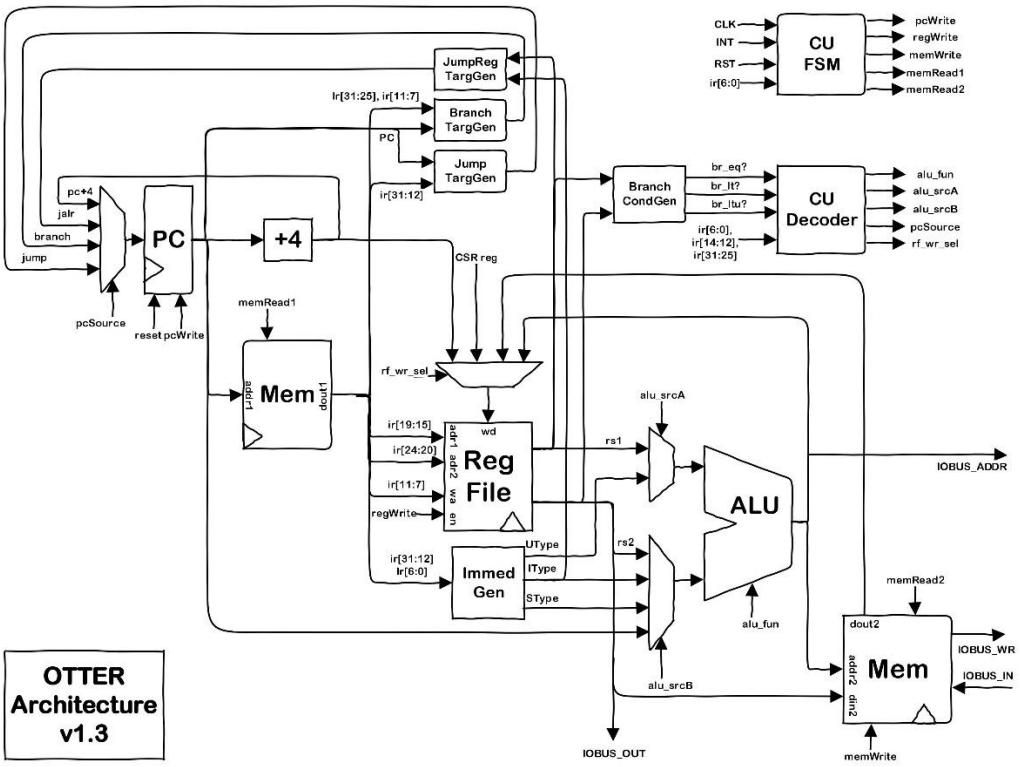

Figure A. Diagram of the OTTER Architecture.

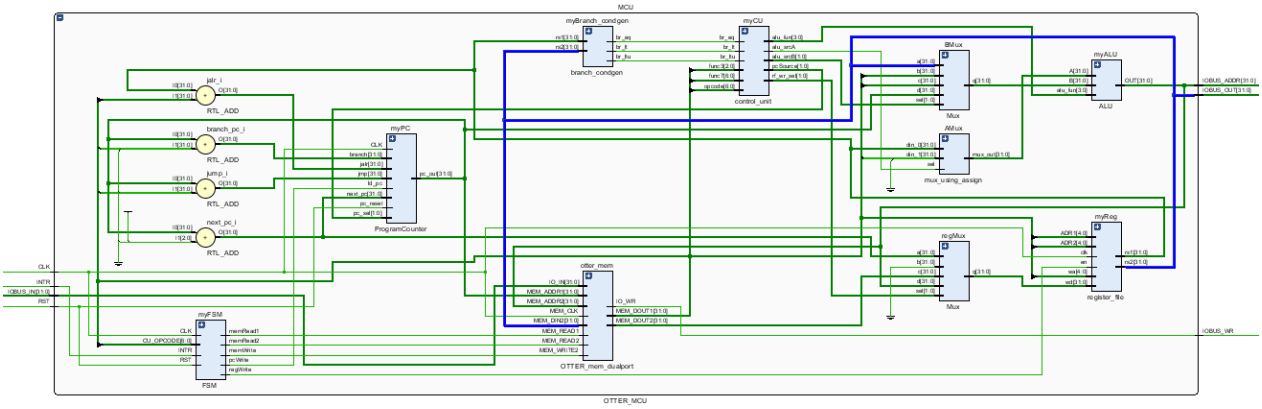

Figure B. Block Diagram of the OTTER MCU

Over the course of the quarter, our class built separate modules of the OTTER in Vivado so that by the time we started our final projects, we had a good enough understanding to make necessary modifications to the ISA. However, most of my project was implemented through software. One of the parts of my casino games board that needed hardware changes from the OTTER we built during class was the Memory Mapped Inputs and Outputs (MMIOs). Originally, the OTTER only had MMIOs setup for the switches, LEDs, and the seven-segment display on the Basys3 board. So, I added MMIOs for the software of my program to read random numbers from a module given by Cal Poly instructor Paul Hummel and all the buttons on the board. These additions to the OTTER are shown below in Figure C. With the implementation of those modules, and the assembly code in the S file included with this document, I was able to make a final project that is fun to play with and simple to make.

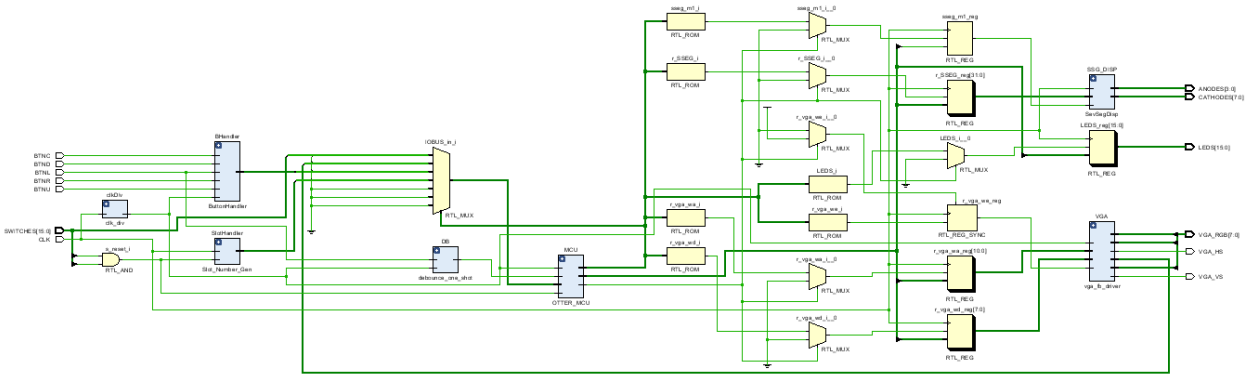

Figure C. Block Diagram of the OTTER Wrapper (hiding the OTTER MCU shown above)

General Flow of the Program

For the main menu (and for most games), the left button on the Basys3 board (BTNL) is used to go back to a previous menu, the top and bottom buttons (BTNU and BTND) are used to navigate through menus, and the middle button (BTNC) is used to select an option. To reset the program at any point of the program (even during a game), push up the most left switch (R2) and the rightest switch (V17) and press on BTNL and BTNR. The board will light up all LEDs, and when you release the buttons, the program will be reset. Resetting the game will reset the

memory to when the program first started (when you programmed the bitstream onto the device), but it will fix any errors or bugs if you ever happen to run into them (Note: This also resets your wallet and your bank amount). The Seven-segment display (SSEG) is the main display of the program. It displays everything from errors due to invalid inputs to how you won a certain game. The LEDs serve as a support display for the program by displaying what switches are turned on and what part of the main menu you are at. The three left LEDs serve as the indicators for what part of the main menu you are at. A user can push the 4th switch from the left (W2) up and either press the up button (BTNU) or the down button (BTND) to view their current bank account amount or current wallet amount, respectively. Each time a user plays a game, they can only bet the money they have in their wallet.

Slots

The slots minigame starts off by asking you to input a bet. Once the bet is confirmed (and valid), the program asks you to push the middle button. This triggers the slot machine to start and the numbers start to generate. The slot machine uses a random number generator module to create a random four-digit number where each digit represents a separate slot (if you were to imagine it as an actual slot machine). After the number is reached, the program checks the output and compares each digit, seeing if the player has reached any of the output requirements. Those requirements include all even numbers, all odd numbers, a straight (forward or backward), four of a kind, three of a kind, and two of a kind. If any of those requirements are met, the player is told that they won, and the winnings are put in the player's wallet.

Blackjack

The blackjack minigame starts off by asking you to input your bet, like slots. Once the bet is locked in, you are given your first number (first two cards in real blackjack). Then, if you haven't already gotten over 21, you get the choice to either hit or stay and play the game, with the general rules of blackjack applying where if you go over 21 you lose, if you get 21 the dealer has a chance to match it, and if the dealer gets more than you without going over 21 they win. If you win, you get double the amount of what you bet, and if you lose you do not get any of your money back.

ATM

If a user wants to protect some of the virtual money they make from the casino games (or take money from their bank account to continue playing the casino games), the Casino Games Board comes with an ATM. The ATM allows the user to either deposit to or withdraw from their virtual bank account. This allows the user to change how much betting money they have and how much reserve money they have.

How to Build the Program

1) Get the Hardware and Software Needed
    a. For this program, I used the Diligent Basys3 Board, which a link to the board can be found at the end of this document.
    b. For the software, I used Vivado, but any program where you can create a bitstream of Verilog modules to program onto a development board should work fine.

2) Begin Making the Extra Modules
    a. Using the random number generator component, make a module to handle generating the slot number.
    b. Edit the given seven segment display modules by changing the modules inside of it. Instead of using the given cathode driver, create a new module that converts 32-bit numbers to a letter in the alphabet. This, shown in the code for Hex_To_Letter below, is how you will be able to refer to letters in your assembly code. You store the 20-bit immediate to a register and store the register at the SSEG address (For example, STRA on the SSEG will be the immediate 0x94E20).

3) Combine All the Modules Together and Write Assembly Code
    a. After making all the modules, connect the modules to the OTTER Wrapper so that the OTTER MCU and the new modules you created can work together.
    b. After finishing with the Vivado part of the program, you can start with the assembly code. For my program, I had about 10 out of the 32 registers (x4-x8, x12-16) hardwired to a specific value, but you can easily make the program without any hardwired registers or with all hardwired registers.

4) Figure Out the Odds and Winnings of Each Game
    a. For my program, I based my winnings directly from the odds, which can be seen below in Table 1 showing the winnings of slots. However, you can decide on how much a certain outcome reward.

Table 1. Winnings and Outcomes of the Slots Minigame

Outcome	Example(s)	Odds	Winnings (where x is bet)
Four of a Kind	8888	10 / 10000	496 * x
Straight	1234/8910/4321	20 / 10000	248 * x
Three of a Kind	5255	370 / 10000	13 * x
Odds	1739	625 / 10000	9 * x
Evens	2684	625 / 10000	8 * x
Two of a Kind (Pair)	9101	4960 / 10000	1 * x (bet returned)

5) Testing (Synthesis, Implementation, Bitstream, Constraints)
    a. To verify the functionality of my program, I ran multiple synthesis, implementation, and bitstream runs to make sure my code was logically correct and physically correct.
    b. I also did a lot of testing with the software after programming my assembly code onto the board to test every possible outcome from each game and every possible input at each menu.

After that, your program should be ready to use on an FPGA. Below is the code for the Hex_to_Letter module.

UPDATE (1/2/2021)

After about two years of sleeping in the land of projects that were started and never completed by me, I decided to come back to this project and finish it before I graduate. As most computer engineering students know, it is common for us to work on a project that has ambitious goals that end up never being reached. However, I wanted to see if I can push myself to look at my old code in languages that I have not used in years to try and complete something I wished I finished before.

As of today, I have completed this project to my satisfaction. I looked over and fixed many bugs and features I left in the system due to not finishing the final project years ago. Although I imagined this program to have more features than it does now, I am happy about how it turned out. I resolved all the issues that I could find whilst testing the program, and I completed all three features of the Casino Games Board (Slot Machine, Blackjack Table, and ATM). The assembly code will be posted in the GitHub repository (which a link will be provided at the end of this document).

Code

Hex to Letter

```
//////////////////////////////////////////////////////////////////////////////////
///
// Company: Cal Poly
// Engineer: Paul Hummel
//           Kelechi Igwe
//
// Create Date: 06/28/2018 11:50:35 PM
// Module Name: CathodeDriver
// Target Devices: Basys3, 4 digit 7 segment display
// Description: Display hex or letter values on a 4 digit 7 segment display
with common
//              anode and common cathode, both configured as negative logic
so
//              0s in the CATHODE light up the segments for any digit
//              corresponding with a 0 in the ANODE
//
//              CATHODES = {dp,a,b,c,d,e,f,g}
//              ANODES = {d4, d3, d2, d1}
//
// Revision:
// Revision 0.01 - File Created
// Revision 1.0 - Changed Name to Hex_to_Letter. Made multiple changes to add
//                compatibility to Casino Board System.
//////////////////////////////////////////////////////////////////////////////////
///


module Hex_to_Letter(
    input CLK,
    input [19:0] HEX,
    input letter_mode,
    input scientific,
    input slots_on,
    output logic [7:0] CATHODES,
```

```
    output logic [3:0] ANODES
    );

    logic s_clk_500 = 1'b0;             // 250Hz refresh clock
    logic [1:0] r_disp_digit = 2'b00;   // current digit being displayed
    logic [19:0] clk_div_counter = 20'h00000;

    // Clock Divider to create 500 Hz refresh from 100 MHz clock
    always_ff @(posedge CLK) begin
        clk_div_counter = clk_div_counter + 1;

        // x186A0 = 1*10^5 = 1 ms toggle (x30D40)
        if ( clk_div_counter == 20'h186A0) begin
            clk_div_counter = 20'h00000;
            s_clk_500 = ~s_clk_500;   // toggle every 1 ms creates 500 Hz
clock
        end
    end

    // Refresh Seven Segment Display every 240 Hz
    always_ff @(posedge s_clk_500) begin
        case (r_disp_digit)
            2'b00: begin
                ANODES= 4'b1110;
                if (letter_mode == 0)
                begin
                    case (HEX[3:0])
                        4'b0000: CATHODES = 8'b10000001; //0
                        4'b0001: CATHODES = 8'b11001111; //1
                        4'b0010: CATHODES = 8'b10010010; //2
                        4'b0011: CATHODES = 8'b10000110; //3
                        4'b0100: CATHODES = 8'b11001100; //4
                        4'b0101: CATHODES = 8'b10100100; //5
                        4'b0110: CATHODES = 8'b10100000; //6
                        4'b0111: CATHODES = 8'b10001111; //7
                        4'b1000: CATHODES = 8'b10000000; //8
                        4'b1001: CATHODES = 8'b10001100; //9
                        4'b1010: CATHODES = 8'b10001000; //a
                        4'b1011: CATHODES = 8'b11100000; //b
                        4'b1100: CATHODES = 8'b10110001; //c
                        4'b1101: CATHODES = 8'b11000010; //d
                        4'b1110: CATHODES = 8'b10110000; //e
                        default: CATHODES = 8'b11111111; // failsafe turn off
                    endcase
                end
                else
                begin
                    case (HEX[4:0])
                    5'b00000: CATHODES = 8'b10001000; //a
                    5'b00001: CATHODES = 8'b11100000; //b
                    5'b00010: CATHODES = 8'b10110001; //c
                    5'b00011: CATHODES = 8'b11000010; //d
                    5'b00100: CATHODES = 8'b10110000; //e
                    5'b00101: CATHODES = 8'b10111000; //f
                    5'b00110: CATHODES = 8'b10000100; //g
                    5'b00111: CATHODES = 8'b11101000; //h
                    5'b01000: CATHODES = 8'b11001111; //i
```

```
                5'b01001: CATHODES = 8'b11000111; //j
                //k isn't available in this module
                5'b01011: CATHODES = 8'b11110001; //l
                5'b01100: CATHODES = 8'b10011001; //first half of m
                5'b11010: CATHODES = 8'b10001101; //second half of m
                5'b01101: CATHODES = 8'b11101010; //n
                5'b01110: CATHODES = 8'b10000001; //o
                5'b01111: CATHODES = 8'b10011000; //p
                5'b10000: CATHODES = 8'b10001100; //q
                5'b10001: CATHODES = 8'b10111001; //r
                5'b10010: CATHODES = 8'b10100100; //s
                5'b10011: CATHODES = 8'b11110000; //t
                5'b10100: CATHODES = 8'b11000001; //u
                5'b10101: CATHODES = 8'b11010101; //v
                5'b10110: CATHODES = 8'b11100001; //first half of w
                5'b11011: CATHODES = 8'b11000011; //second half of w
                //x isn't available in this module
                5'b11000: CATHODES = 8'b11000100; //y
                5'b11001: CATHODES = 8'b10010010; //z
                default: CATHODES = 8'b11111111; // failsafe turn off
                endcase
            end
        end
        2'b01: begin
            ANODES= 4'b1101;
            if (letter_mode == 0)
            begin
                if (scientific == 1'b1)
                    CATHODES = 8'b11111111;
                else begin
                    case (HEX[7:4])
                        4'b0000: begin
                            if (HEX[15:12] == 4'b0000 &&
                                HEX[11:8] == 4'b0000 &&
                                slots_on == 1'b0)
                                CATHODES = 8'b11111111;
                            else
                                CATHODES = 8'b10000001;
                        end
                        4'b0001: CATHODES = 8'b11001111;
                        4'b0010: CATHODES = 8'b10010010;
                        4'b0011: CATHODES = 8'b10000110;
                        4'b0100: CATHODES = 8'b11001100;
                        4'b0101: CATHODES = 8'b10100100;
                        4'b0110: CATHODES = 8'b10100000;
                        4'b0111: CATHODES = 8'b10001111;
                        4'b1000: CATHODES = 8'b10000000;
                        4'b1001: CATHODES = 8'b10001100;
                        4'b1010: CATHODES = 8'b10001000; //a
                        4'b1011: CATHODES = 8'b11100000;
                        4'b1100: CATHODES = 8'b10110001;
                        4'b1101: CATHODES = 8'b11000010;
                        4'b1110: CATHODES = 8'b10110000;
                        default: CATHODES = 8'b11111111; // all off on
error
                    endcase
                end
```

```
        end
        else
        begin
            case (HEX[9:5])
            5'b00000: CATHODES = 8'b10001000; //a
            5'b00001: CATHODES = 8'b11100000; //b
            5'b00010: CATHODES = 8'b10110001; //c
            5'b00011: CATHODES = 8'b11000010; //d
            5'b00100: CATHODES = 8'b10110000; //e
            5'b00101: CATHODES = 8'b10111000; //f
            5'b00110: CATHODES = 8'b10000100; //g
            5'b00111: CATHODES = 8'b11101000; //h
            5'b01000: CATHODES = 8'b11001111; //i
            5'b01001: CATHODES = 8'b11000111; //j
            //k isn't available in this module
            5'b01011: CATHODES = 8'b11110001; //l
            5'b01100: CATHODES = 8'b10011001; //first half of m
            5'b11010: CATHODES = 8'b10001101; //second half of m
            5'b01101: CATHODES = 8'b11101010; //n
            5'b01110: CATHODES = 8'b10000001; //o
            5'b01111: CATHODES = 8'b10011000; //p
            5'b10000: CATHODES = 8'b10001100; //q
            5'b10001: CATHODES = 8'b10111001; //r
            5'b10010: CATHODES = 8'b10100100; //s
            5'b10011: CATHODES = 8'b11110000; //t
            5'b10100: CATHODES = 8'b11000001; //u
            5'b10101: CATHODES = 8'b11010101; //v
            5'b10110: CATHODES = 8'b11100001; //first half of w
            5'b11011: CATHODES = 8'b11000011; //second half of w
            //x isn't available in this module
            5'b11000: CATHODES = 8'b11000100; //y
            5'b11001: CATHODES = 8'b10010010; //z
            default: CATHODES = 8'b11111111; // failsafe turn off
            endcase
        end
    end
    2'b10: begin
        ANODES= 4'b1011;
        if (letter_mode == 0)
        begin
            case (HEX[11:8])
                4'b0000: begin
                    if (HEX[15:12] == 4'b0000 &&
                        slots_on == 1'b0)
                        CATHODES = 8'b11111111;
                    else
                        CATHODES = 8'b10000001;
                end
                4'b0001: CATHODES = 8'b11001111;
                4'b0010: CATHODES = 8'b10010010;
                4'b0011: CATHODES = 8'b10000110;
                4'b0100: CATHODES = 8'b11001100;
                4'b0101: CATHODES = 8'b10100100;
                4'b0110: CATHODES = 8'b10100000;
                4'b0111: CATHODES = 8'b10001111;
                4'b1000: CATHODES = 8'b10000000;
                4'b1001: CATHODES = 8'b10001100;
```

```
                4'b1010: CATHODES = 8'b10001000; //a
                4'b1011: CATHODES = 8'b11100000;
                4'b1100: CATHODES = 8'b10110001;
                4'b1101: CATHODES = 8'b11000010;
                4'b1110: CATHODES = 8'b10110000;
                default: CATHODES = 8'b11111111; // all off on error
            endcase
        end
        else
        begin
            case (HEX[14:10])
            5'b00000: CATHODES = 8'b10001000; //a
            5'b00001: CATHODES = 8'b11100000; //b
            5'b00010: CATHODES = 8'b10110001; //c
            5'b00011: CATHODES = 8'b11000010; //d
            5'b00100: CATHODES = 8'b10110000; //e
            5'b00101: CATHODES = 8'b10111000; //f
            5'b00110: CATHODES = 8'b10000100; //g
            5'b00111: CATHODES = 8'b11101000; //h
            5'b01000: CATHODES = 8'b11001111; //i
            5'b01001: CATHODES = 8'b11000111; //j
            //k isn't available in this module
            5'b01011: CATHODES = 8'b11110001; //l
            5'b01100: CATHODES = 8'b10011001; //first half of m
            5'b11010: CATHODES = 8'b10001101; //second half of m
            5'b01101: CATHODES = 8'b11101010; //n
            5'b01110: CATHODES = 8'b10000001; //o
            5'b01111: CATHODES = 8'b10011000; //p
            5'b10000: CATHODES = 8'b10001100; //q
            5'b10001: CATHODES = 8'b10111001; //r
            5'b10010: CATHODES = 8'b10100100; //s
            5'b10011: CATHODES = 8'b11110000; //t
            5'b10100: CATHODES = 8'b11000001; //u
            5'b10101: CATHODES = 8'b11010101; //v
            5'b10110: CATHODES = 8'b11100001; //first half of w
            5'b11011: CATHODES = 8'b11000011; //second half of w
            //x isn't available in this module
            5'b11000: CATHODES = 8'b11000100; //y
            5'b11001: CATHODES = 8'b10010010; //z
            default: CATHODES = 8'b11111111; // failsafe turn off
            endcase
        end
    end
    2'b11: begin
        ANODES= 4'b0111;
        if (letter_mode == 0)
        begin
            case (HEX[15:12])
                4'b0000: begin
                    if (slots_on == 1'b0)
                        CATHODES = 8'b11111111;
                    else
                        CATHODES = 8'b10000001;
                end
                4'b0001: CATHODES = 8'b11001111;
                4'b0010: CATHODES = 8'b10010010;
                4'b0011: CATHODES = 8'b10000110;
```

```
                    4'b0100: CATHODES = 8'b11001100;
                    4'b0101: CATHODES = 8'b10100100;
                    4'b0110: CATHODES = 8'b10100000;
                    4'b0111: CATHODES = 8'b10001111;
                    4'b1000: CATHODES = 8'b10000000;
                    4'b1001: CATHODES = 8'b10001100;
                    4'b1010: CATHODES = 8'b10001000; //a
                    4'b1011: CATHODES = 8'b11100000;
                    4'b1100: CATHODES = 8'b10110001;
                    4'b1101: CATHODES = 8'b11000010;
                    4'b1110: CATHODES = 8'b10110000;
                    default: CATHODES = 8'b11111111; // all off on error
                endcase

                CATHODES[7] = ~scientific;
            end
            else
            begin
                case (HEX[19:15])
                5'b00000: CATHODES = 8'b10001000; //a
                5'b00001: CATHODES = 8'b11100000; //b
                5'b00010: CATHODES = 8'b10110001; //c
                5'b00011: CATHODES = 8'b11000010; //d
                5'b00100: CATHODES = 8'b10110000; //e
                5'b00101: CATHODES = 8'b10111000; //f
                5'b00110: CATHODES = 8'b10000100; //g
                5'b00111: CATHODES = 8'b11101000; //h
                5'b01000: CATHODES = 8'b11001111; //i
                5'b01001: CATHODES = 8'b11000111; //j
                //k isn't available in this module
                5'b01011: CATHODES = 8'b11110001; //l
                5'b01100: CATHODES = 8'b10011001; //first half of m
                5'b11010: CATHODES = 8'b10001101; //second half of m
                5'b01101: CATHODES = 8'b11101010; //n
                5'b01110: CATHODES = 8'b10000001; //o
                5'b01111: CATHODES = 8'b10011000; //p
                5'b10000: CATHODES = 8'b10001100; //q
                5'b10001: CATHODES = 8'b10111001; //r
                5'b10010: CATHODES = 8'b10100100; //s
                5'b10011: CATHODES = 8'b11110000; //t
                5'b10100: CATHODES = 8'b11000001; //u
                5'b10101: CATHODES = 8'b11010101; //v
                5'b10110: CATHODES = 8'b11100001; //first half of w
                5'b11011: CATHODES = 8'b11000011; //second half of w
                //x isn't available in this module
                5'b11000: CATHODES = 8'b11000100; //y
                5'b11001: CATHODES = 8'b10010010; //z
                default: CATHODES = 8'b11111111; // failsafe turn off
                endcase
            end
        end
        default: begin      // digit error turn everything off
            ANODES = 4'hF;
            CATHODES = 8'hFF;
            r_disp_digit = 2'b00;
        end
    endcase
```

```
        r_disp_digit = r_disp_digit + 1;
    end
endmodule
```

Links

Basys3Board: https://store.digilentinc.com/basys-3-artix-7-fpga-trainer-board-recommended-for-introductory-users/

Github: https://github.com/kigwe/CasinoGamesBoard