LED ASCII Matching Game

Team Members: Tim Pietrzyk, Dylan Powell, Nick Schwartz, Brandon Troppens

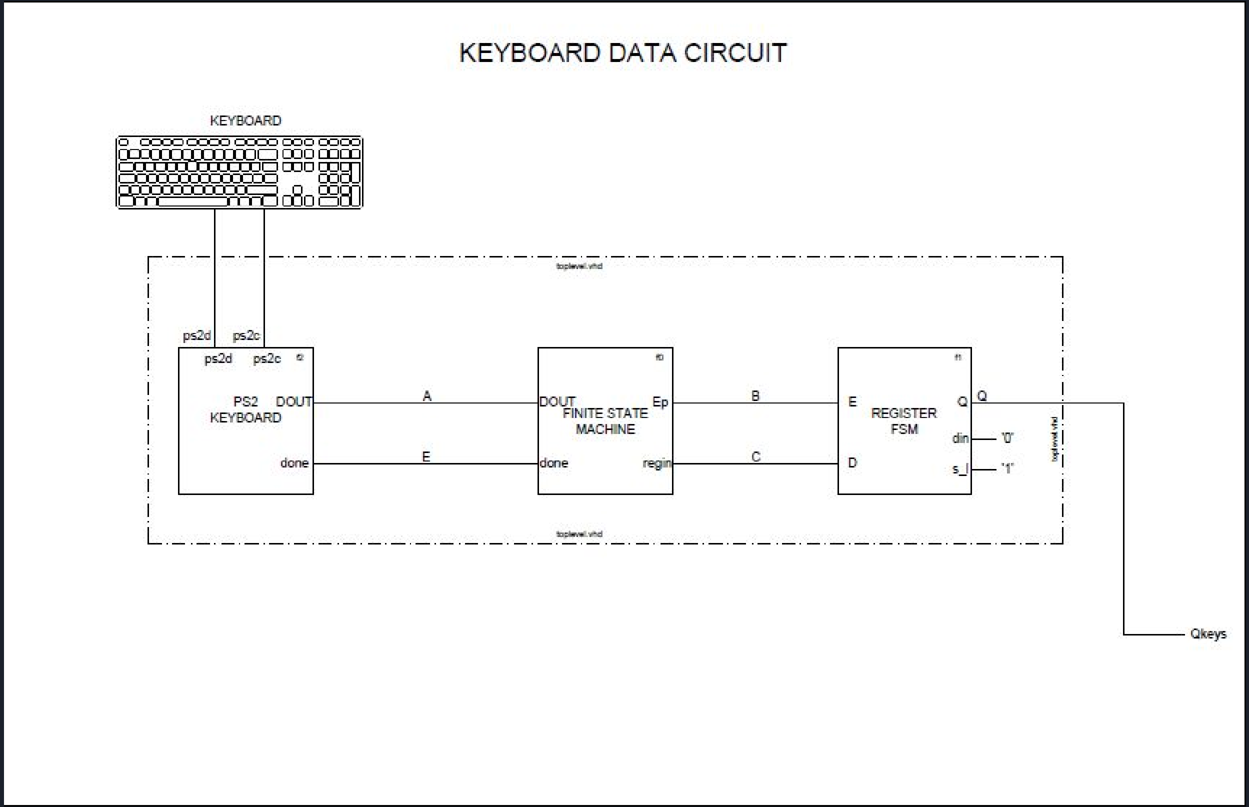
LED Matching Game How to Play

The LED Matching Game consist of an 8 bit random input that lights LEDs on the Nexys4 board. The user has to match the correct input by pressing the right key before the timer runs down.

To limit the options chosen by the 8 bit random number generator the keys that will be used are 0-9 on the keyboard.

If the user presses the correct key the output will be added to a score and displayed on the seven segment display. Then the value will be changed for the user to press another key.

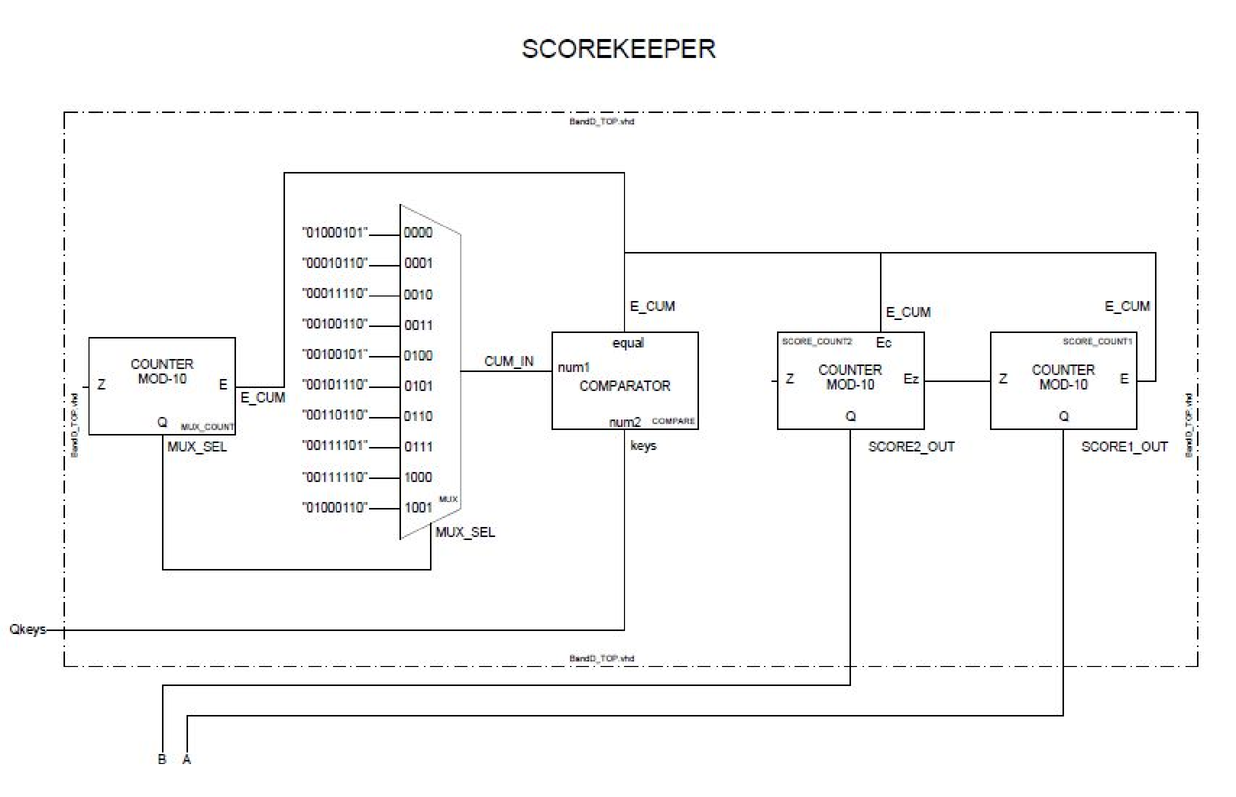
Schematic / Component Layout - Key Data



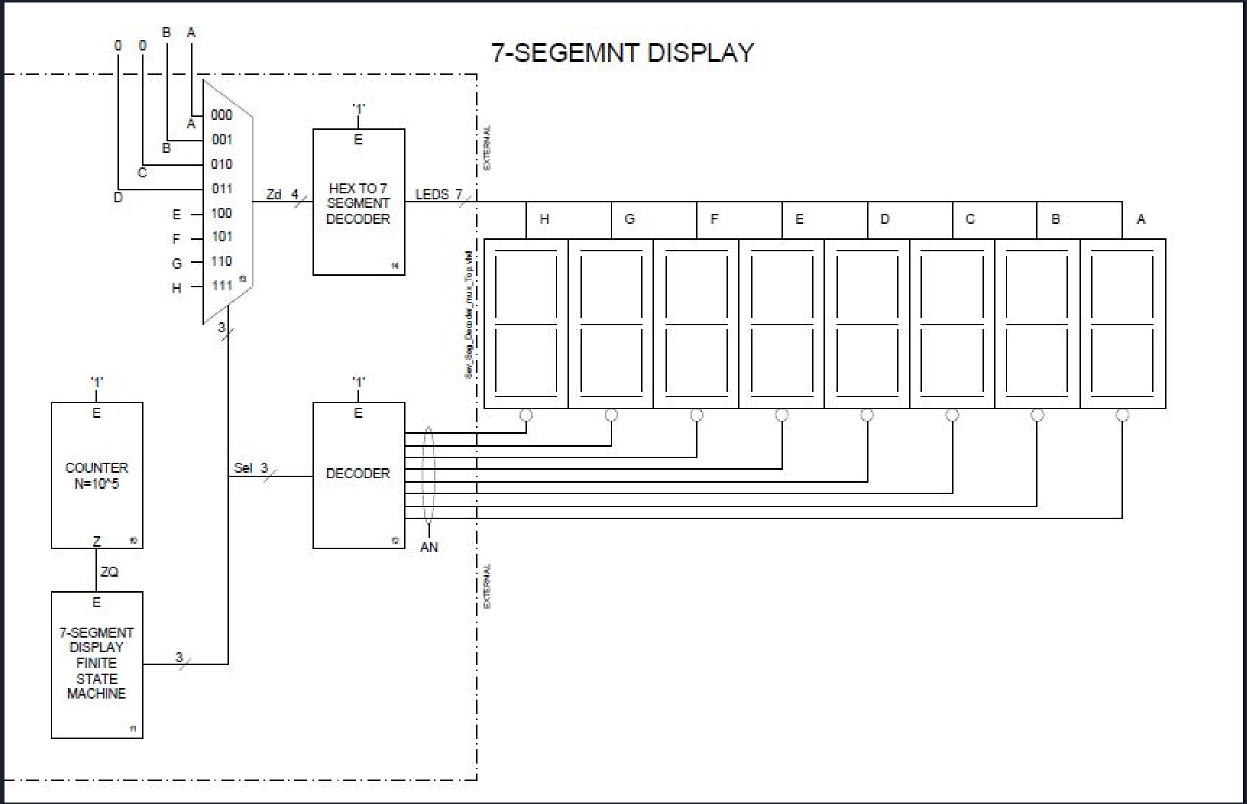
Key Data FSM Code

```
type state is (S1, S2);
signal y: state;
signal FCD : std_logic_vector (7 downto 0);

begin
regin <= FCD;
FCD <= DOUT(7 downto 0);
transitions: process (resetn, clock, DOUT, done)
begin

if resetn = '0' then y <= s1;
        elsif (clock'event and clock = '1') then
            case y is
                when S1 =>
                    if done = '1' and FCD = x"F0" then y <= S2; --F0
                        else y <= S1;
                    end if;
                when S2 =>
                        if done = '1'  then y <= S1;
                            else y <= S2;
                    end if;
                end case;
        end if;
    end process;

 Outputs: process (y, done)
  begin
  Ep <= '0';
    case y is
    when S1 =>
    when S2 =>  if done = '1' then Ep <= '1'; end if;
    end case;
 end process;
end Behavioral;
```

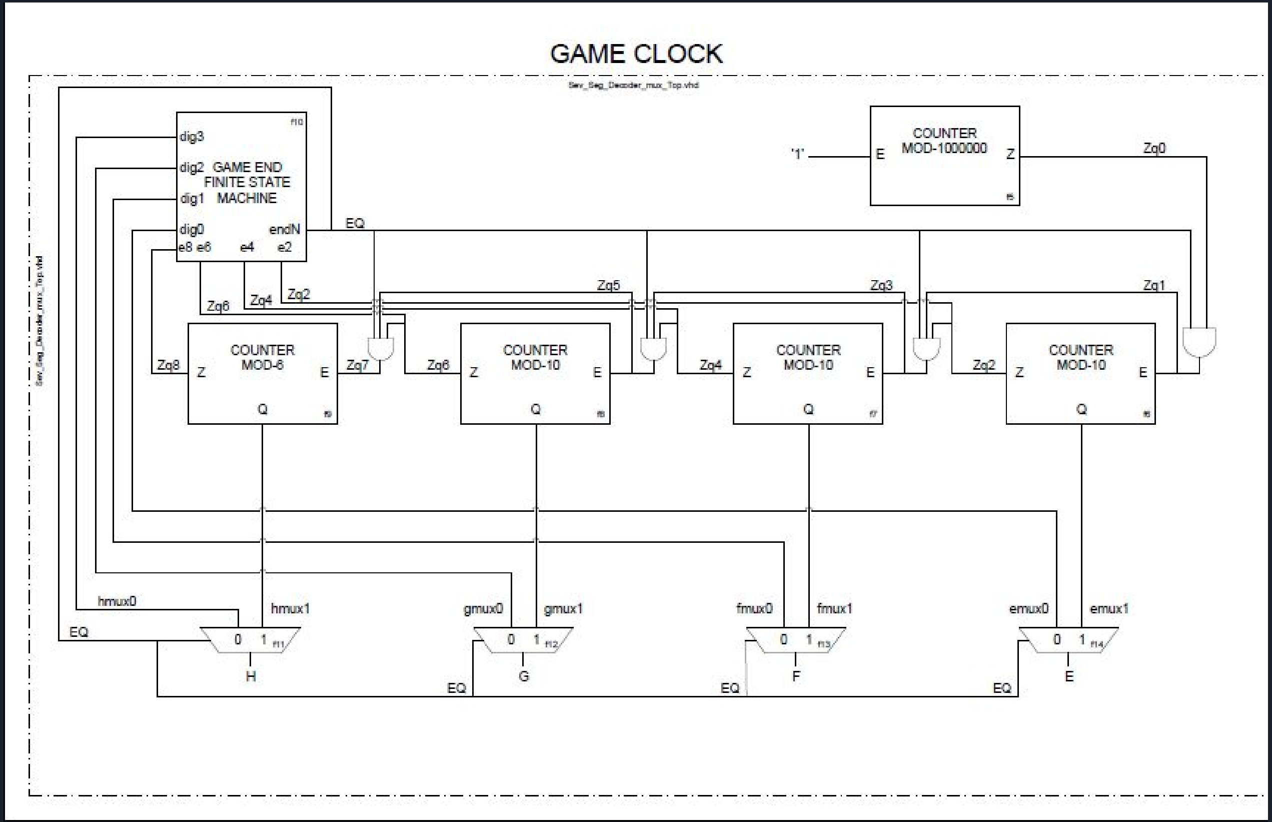
Schematic / Component Layout - Scoring



Schematic / Component Layout - Displays



Schematic / Component Layout - Clock



Code

The code consisted of many digital logic operations that we have learned throughout the semester, including a multiplexer, seven segment display, comparator, and a full 4 bit adder for example. A snippet of the code can be seen below for the comparator:

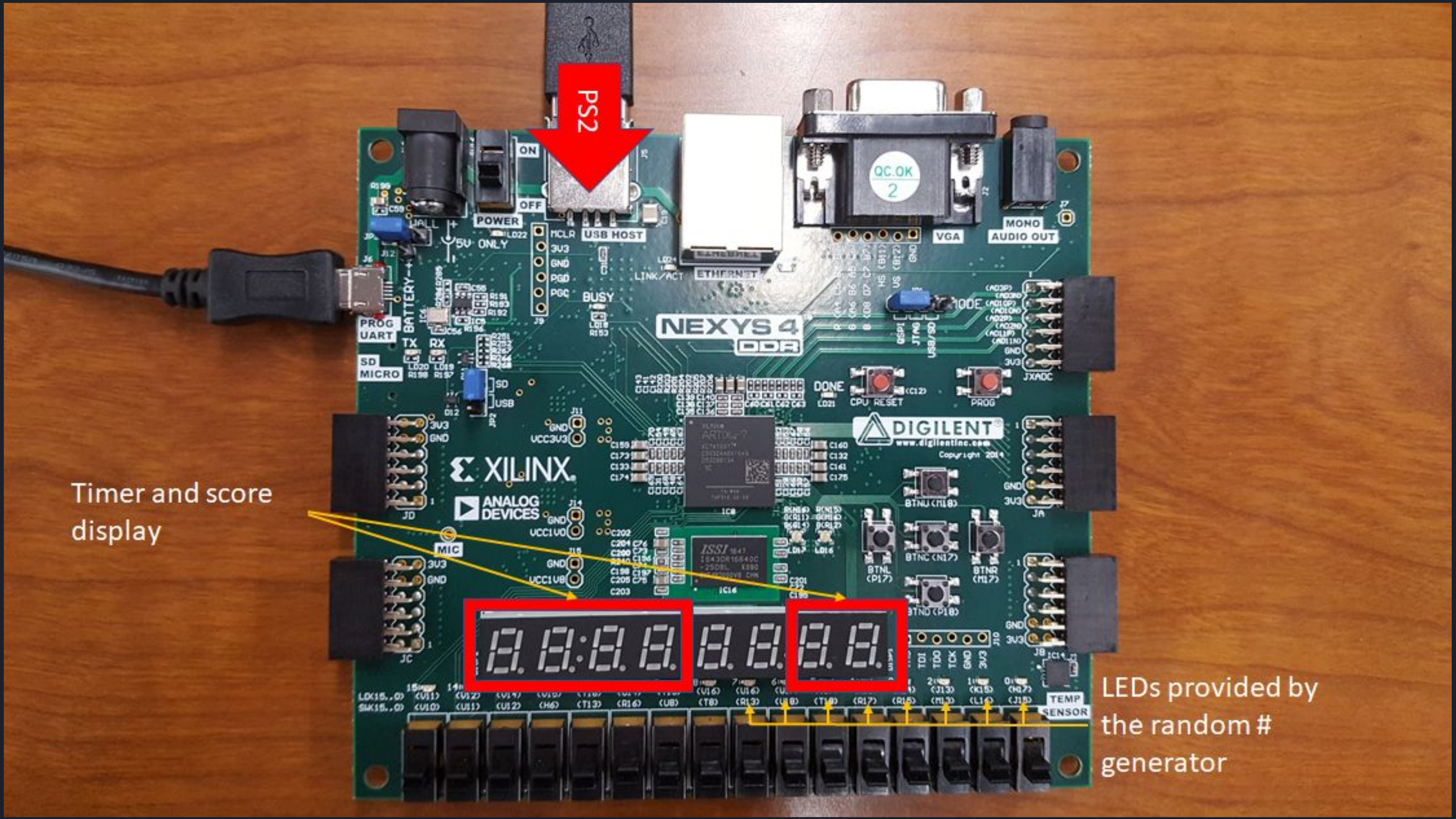
```
architecture Behavioral of COMPARATOR is
    signal clk: std_logic;
    begin
    process(num1,num2)
    begin
            if ( num1 > num2) then
            elsif(num1 < num2) then
            else
            equal <= '1';
            end if;

    end process;
```

Code

Here is the code for the game end finite state machine. Once you reach state 5, you can't exit unless the game is reset using the CPU reset button.

```
 Transitions : process (resetn, clk, e8, e6, e4, e2, y) -- Definition of State Transitions
 begin
     if resetn = '0' then --asynchronous clear
         y <= S1; --Initial State

     elsif (clk'event and clk ='1') then
         case y is
             when S1 =>
                 if e8 = '1' then y <= S2; else y <= S1; end if; --and e6 ='1' and e4 ='1' and e2 = '1'
             when S2 =>
                 if e6 = '1' then y <= S3; else y <= S2; end if;
             when S3 =>
                 if e4 = '1' then y <= S4; else y <= S3; end if;
             when S4 =>
                 if e2 = '1' then y <= S5; else y <= S4; end if;
             when S5 =>
                 --no escape except resetn
         end case;

      end if;
end process;

 Outputs : process (y) -- Definition of Outputs Based on State
 begin
     endN <= '1';
     case y is
         when S1 => endN <= '1';
         when S2 => endN <= '1';
         when S3 => endN <= '1';
         when S4 => endN <= '1';
         when S5 => endN <= '0'; dig3 <= "0110"; dig2 <= "0000"; dig1 <= "0000"; dig0 <= "0000";
     end case;
 end process;
```

Board layout



Design Difficulties

One design difficulty that was faced was the creation of the random number generator. We first intended to have the numbers randomly generated after each successful button input, but found that it would be quite difficult to implement this into VHDL.

So it was decided to have a random order MUX to the LEDs lighting up, whereas a segment of 10 different numbers would light up in a certain order. This order would then be repeated every time the cycle was completed.

Conclusion

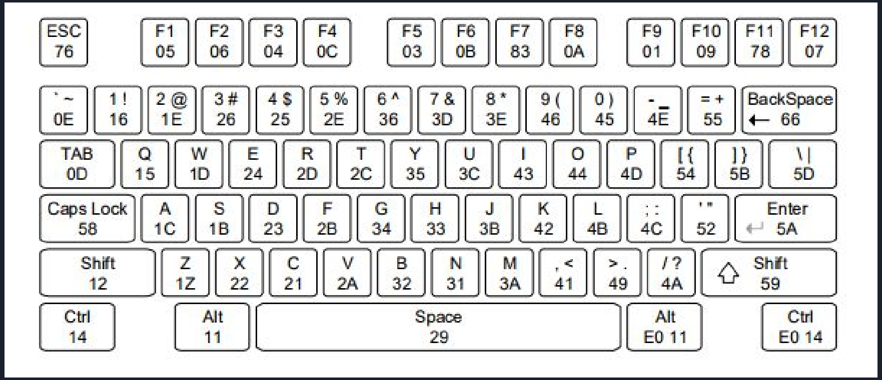
As expected, we made many changes when working throughout the project. Although there were some disagreements, we found a happy medium for the final game. We learned how to communicate with one another and create a project that every member was satisfied with.

Working together as a group allowed us to further our understanding of VHDL and its applications to the real world. The team problem solved through various issues in the project and came out with an entertaining game.

Thank You

Questions and Demonstration

Nexys4 DDR Keyboard:



ASCII to binary conversion table:

ASCII Code	Binary Value
0	01000101
1	00010110
2	00011110
3	00100110
4	00100101
5	00101110
6	00110110
7	00111101
8	00111110
9	01000110