ASCII LED Matching Game

List of Authors (Tim Pietrzyk, Dylan Powell, Nick Schwartz, Brandon Troppens)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: twpietrzyk@oakland.edu, dmpowell@oakland.edu, ndschwartz@oakland.edu, bjtroppens@oakland.edu

The project is a ASCII LED matching game. The goal of the game is for the player to press a key on the keyboard that corresponds to the displayed ASCII code. Each correct match rewards the player a point.

I. INTRODUCTION

The project's scope includes the design and implementation of a matching game. The game utilizes ASCII code represented in binary form as seen by the user on LEDS. The Nexys4 DDR Board was used to display both the LEDs, as well as the seven segment displays. The seven segment displays consisted of a timer that counts up to sixty seconds, and the score that increments when the user inputs a correct result. The point of the game is to test the user's ability on their knowledge of ASCII code in binary form. The motivation was for each team member to learn ASCII code in a fun and entertaining way.

The team learned many different aspects from class that they implemented in the design of the final project. For example, the team included many VHDL architecture code in the design, including multiplexers, counters, decoders, and finite state machines. The team also executed their own VHDL codes, including utilizing a comparator and a PS2 keyboard via USB.

The applications of the team's project allow players to further their knowledge of ASCII characters in the numerical range from 0 to 9 seen in the Nexys 4 datasheet [2]. This also corresponds to the player's knowledge of hex symbols.

II. METHODOLOGY

A. Keyboard Data

The keyboard data was one of the main incorporations in the project. To make it easy for the user to select a ASCII key character two of the inputs are clock and data from the keyboard. The team decided to use only the characters 0-9. At the core the keyboard circuit data involved three main components. The first components involved generating a done and 10-bit output signal. The done signal is the signal generated by the keyboard to represent the key down press and key up code. To control the done signal generated a Finite State Machine (FSM) is used to generate the ASCII data when a key is pressed and released. To generate this the FSM had two states. State 1 controlled the done signal and the data generated by the done signal F0 had to match. The reason for this is, "When a key is released, an F0 key-up code is sent, followed by the scan code of the released key" [2]. After the key release code "F0" and done signal are sent the FSM enters state 2. In state 2 the signal Ep sent a logic high to the register when the done signal was complete by the key lifting. This signal latched the FSM to the register.

The register component captured the data output from the FSM attached to the keyboard circuit. By latching the data sent by the keyboard FSM the ASCII output can now be presented to the scoring circuit. To test the circuit the signals were port mapped in a top file and implemented on the Nexys board. The output of the ASCII key data was represented by the switch LEDs on the Nexys board. A diagram for the ASCII data output was provided by the Nexys 4 DDR datasheet [2], and is below in (Diagram 1). The entire schematic of the keyboard data circuit is shown in (Figure 1).

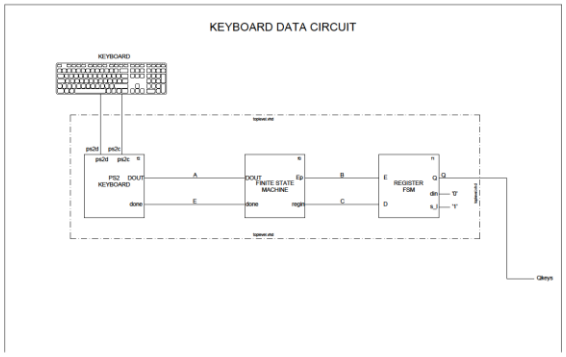

Figure 1: PS2 Keyboard Schematic

B. *Scoring Modual*

The team's first application consisted of the portion of the project that compared the user's keyboard input with the currently lit up LED pattern. This LED pattern was chosen by a multiplexer with preset values that appeared random by the user. If the user's input value matched with the current multiplexer value, then the comparator would send out a signal to two different counters. The first counter would output a signal back to the multiplexer that would act as a select, allowing a new value to be outputted by the multiplexer.

The second counter would act as the scoring mechanism for the game. When the player's input and the multiplexer's output match, the counter would increment by one. The team also included a second counter in series with the first one that would account for values in the tens place This second counter would be activated when the first counter is at nine and the player inputs a correct value. The first counter's output is anded with the output of the comparator. The comparator checked to see if the values were equal. Code examples of a comparator were found online in VHDL [3]. The second counter would increment by one when both values are high. The schematic for the scoring and comparator system can be seen below as (Figure 2)

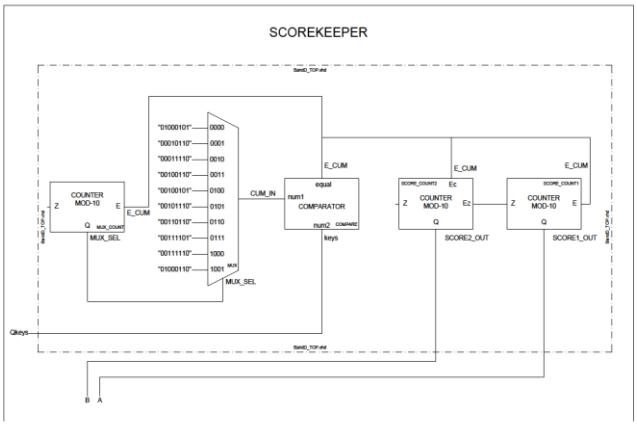

Figure 2: Scoring and Comparator Schematic

C. *Timer Display*

The seven segment display portion of the circuit was designed in the same way as shown in Dr. Llamocca's class notes. Simply put, the signals A-H correspond to the displays A-H in Figure 3. Whatever four-bit value is placed on the mux pins A-H will be displayed on the displays A-H. This is accomplished by rapidly switching which displays are enabled, and rapidly switching the value displayed. The input multiplexor is switched synchronously with the anodes of the displays.

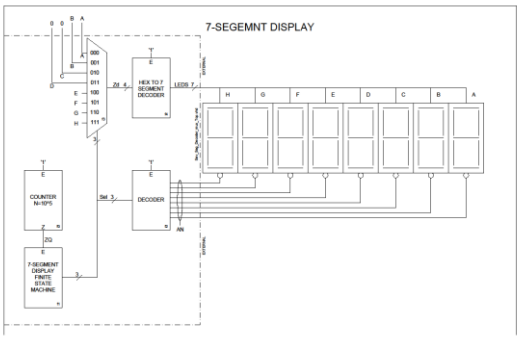

Figure 3: Timer Display Schematic

D. *Game Clock Controller*

The heart of the game clock controller are the 3 mod-10 counters and the mod-6 counter. These counters are controlled by the mod-1000000 counter, which slows down the rightmost counter to 1ns counts. In order to enable the next counter, it's necessary to reach the final count of the current counter. The exception is when the mod-6 counter reaches its' final value. At that point, "game end finite state machine" (Figure 5) will pull "EQ" low, and disable the counters through the and gates. Simultaneously, it switches control of the timer display over to itself through the two channel muxes. The display will then read "60:00", which is hard-coded in the FSM. This indicates that time has expired. The game must be reset in order for the display to switch back to counting again. The FSM has five states which are the initial state, and then four states which check for the "Zq" of each counter.

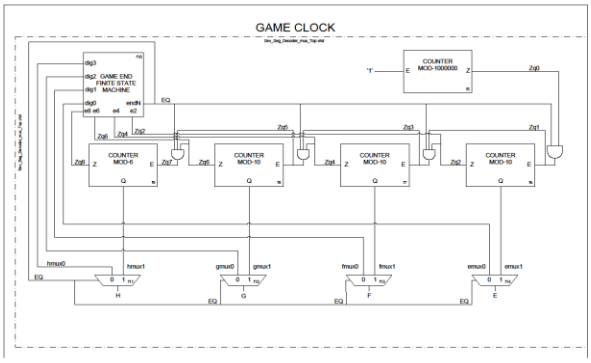

Figure 4: Game Clock Controller

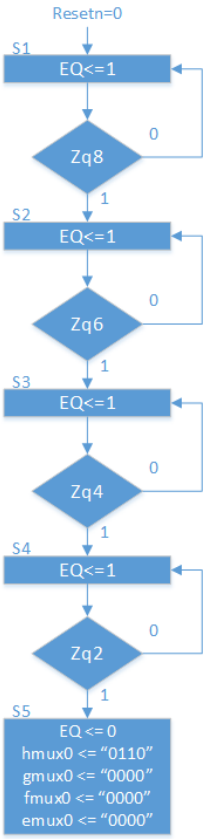

Figure 5: Game End FSM

III. Experimental Setup

The team used only the Nexys4 DDR and a PS2 Keyboard as the hardware of the project. On the software side, the team used Vivado VHDL which consisted of 33 files. A test bench file was created to simulate and debug. The simulation verified the team's anticipated values for what the game should output. Once the simulation was verified, the team worked to configure a bit stream to implement the game in real time.

At first, the team had trouble with the simulation. Some of these problems were simple syntax errors, VHDL hierarchy errors, and code bugs that hindered the team from completing the project. One such problem was the creation of a random number generator. After much research, the team deemed that implementing a random number generator would be too hard to code in VHDL within the time frame. The team decided to omit any random number generator code, and decided to use a simple multiplexer with a number of preset values ranging from 0 to 9. These values were set up in a random order so that each increment would go to the next unexpected value. To the player, this would appear random, making the game challenging to a new user.

Once the team fixed all of the errors, the team was able to program the FPGA and further debug the program for any more errors. The team expected the game to work perfectly, but it did not. One such problem in particular was that one of the tens place counter was not incrementing when it should have. After vigorous simulation, the team figured that the problem arose from an error within the if statements inside the architecture of the counter's code.

IV. Results

The design successfully counts from 0 to 60 seconds and the user's input values are registered form the keyboard. The design notifies the user when the correct ASCII key has been pressed, the user will see a 1 get generated on the scoring seven segment display. Then the LEDs will change to show the next ASCII 8-bit output. The cycle starts all over again when the user selects another key. All inputs and outputs function as desired, and during the presentation display. A board layout picture is shown below to indicate what displays will turn on (Figure 6).

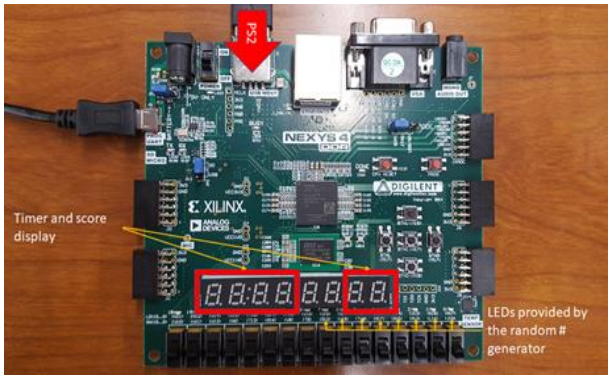

Figure 6: Board Layout Showing Input and Output Displays

V. Conclusions

Integrating the keyboard data, scoring module, timer display, and game clock controller. The ASCII game was created. This project reinforced the understanding of FSMs and generic counters. This was no simple feat, as figuring out the key data simulated a 'F0' signal for the key code down. Improvements to the design that could be making different levels include the notification LED to the user. Thereby drawing more game states to the user when the time has expired. Making a larger number generator MUX with more ASCII values can

also be implemented to provide the user with more questions.

REFERENCES

[1] Llmocca D VHDL Coding for FPGAs. In: VHDL Coding for FPGAs. http://www.secs.oakland.edu/~llamocca/vhdlforfpgas.html. Accessed 2 Nov 2016

[2] *Nexys4 DDR FPGA Board Reference Manual*. , C ed., Pullman, Digilent, 2014. Accessed 29 Nov. 2017.

[3] "VHDL code for 4-Bit magnitude comparator." All About FPGA, 9 June 2015, allaboutfpga.com/vhdl-code-4-bit-binary-comparator/.