Avinash Bissoondial and Noah Solomon | ECE 2029 Final Project | December 15, 2023

Intro:

The goal of our project was to create a binary converter test game. The game would be used to improve the user's ability to convert binary numbers to decimal numbers. The functionality of the game was to first generate a random 8-bit number, and display the random number on the 7-segment display. The user then inputs a number using 8 of the switches on the board. If the answer is correct, the board would say they are right and their score (the number of correct answers in a row since booting) then generate a new number and display it on the board. If the answer is incorrect, the screen says so and shows the correct answer on the LED lights above the switches.

Design and Implementation:

Overall Design:
Our design consisted of six different instantiated modules that were all called by a top file. Our top file controlled sequential state changes that decided what stage of the game our FPGA board would display. The states we used were as follows:

1. RESET
    a. This state disables all LEDs, generates a random number, converts the number and displays it on the 7-segment display.
    b. We move to the next state after the enter button is pressed.
2. TEST
    a. This state will check if the switches flipped are equal to the random number that was generated.
    b. When the entry button is pressed, if the switches are correct, increase the score and transition to CORRECT state. If they are not correct, set the score to zero and transition to WRONG.
3. CORRECT
    a. Display in order with 1 second between each: "Good"/"Scor" [i.e. score in four letters]/the user's score
    b. If the enter button is pressed at any time after a short delay (to avoid accidentally zooming through this from the first time the button was pressed), go to RESET to play again.
4. WRONG
    a. Display in order with 1 second between each: "Fail"/"You"/"Said"/the user's incorrect entry in decimal.
    b. The lights above the switches that should have been pressed light up.
    c. After a similar delay to in CORRECT, let the user send us back to RESET by pressing the entry button

Below all six of our instantiated modules and what they do.

Num_converter:
This is a module that takes an input of an 8-bit number and uses a binary to binary coded decimal converter. It places each digit into the correct place (hundreds, tens, ones). It assigns these digits into a 28 bit output variable.

Slowclock:
This is a module that simply slows down the clock, and is used to display a number in the right place on the 7 segment display. It is the same slow clock that was used in labs 4 and 5. (For some reason displaying was working very wrong when using the normal clock — see picture)

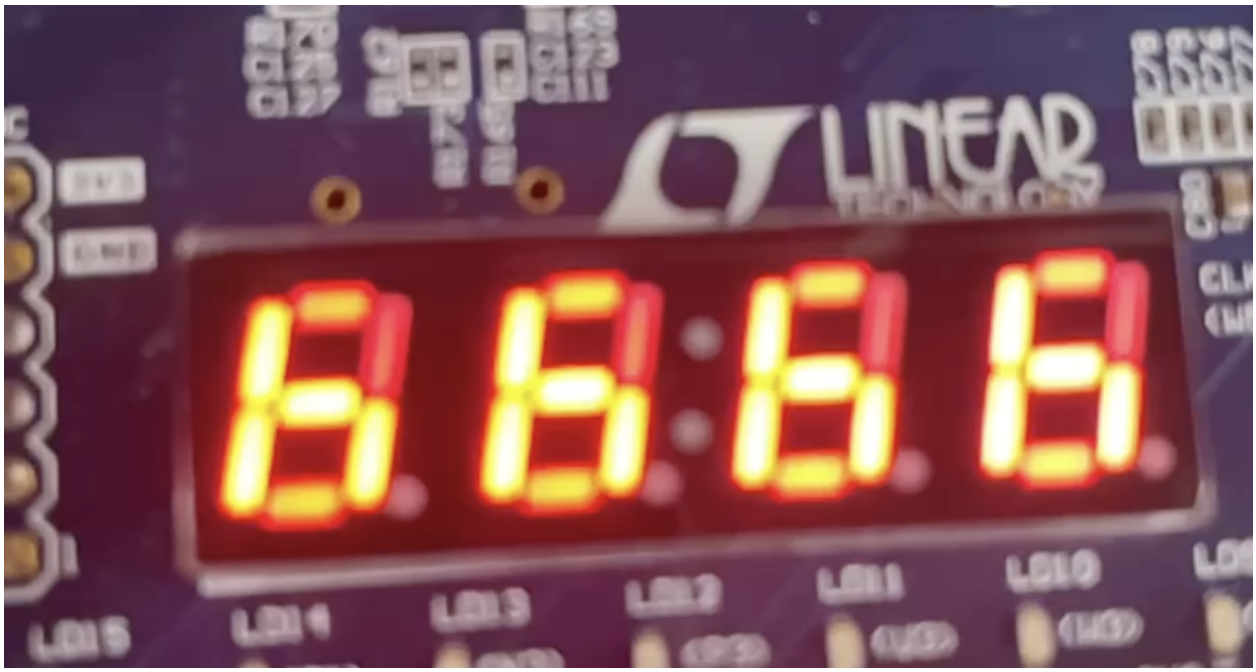

Figure 1: With the fast clock for some reason it would switch between all letters without turning the anodes off at the wrong time. We couldn't figure out why (and the TAs were stumped too) but it was fixed by switching the counter for seg/an to use the slow clock.

Counter:
This module is also reused from labs 4 and 5. It is a two-bit counter that just counts up by 1, using *slowclock*.

Word_mux:
This module handles all of the final steps of displaying stuff; it takes in a 28-bit code for the seven segment display, and shows it. So it takes the counter in, and a 28-bit variable word; we defined some 28-bit constants for the necessary words, and set the input equal to either that or the output of the number converter depending on context. It outputs to the anodes and the segments.

Blinker:
This module just takes in the random number generated in the random module and outputs that into the LED lights. It is called when the user gets the answer wrong.

Random:
This module generates the random 8-bit number that is displayed as the target value in the game. This module uses the clock to generate the random number — just counts up constantly (looping back to 0 after 255).

To do the delays we had two variables used for counting based on the 100MHz clock — *A* used for RESET and *B* for CORRECT/WRONG — so that there would not be the concern of one not being reset in time for its next usage.

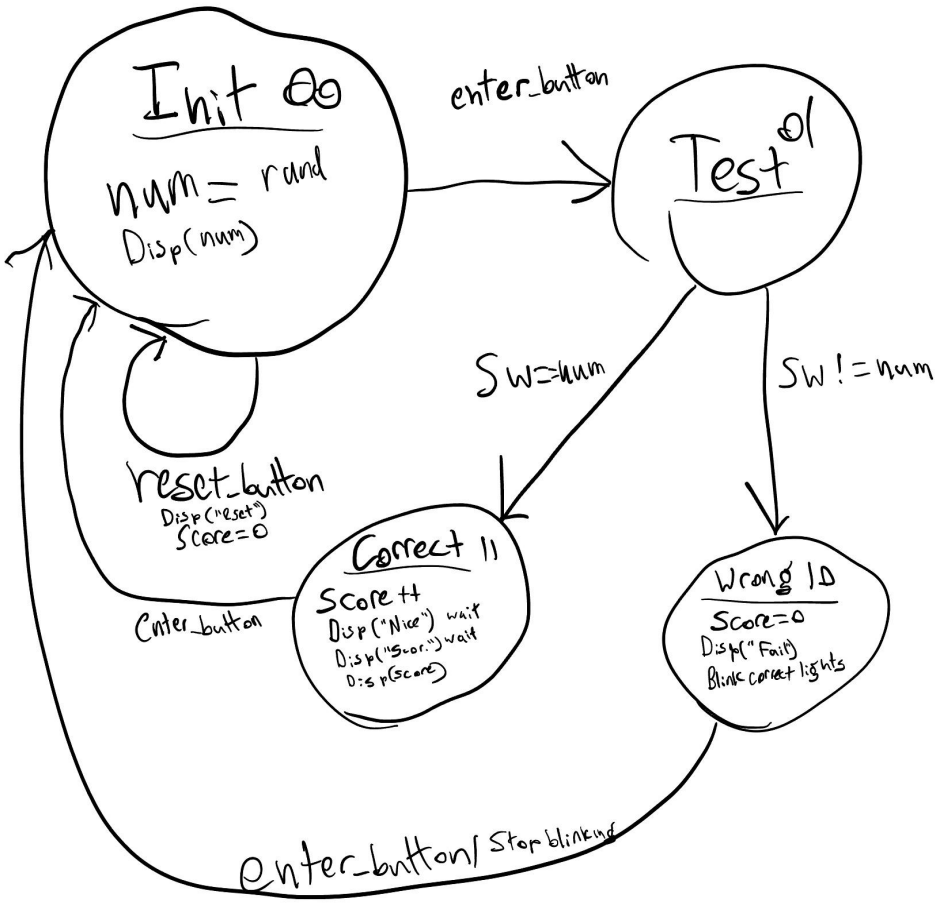

Figure 2: Diagram representing a mid-development plan for the state machine

Outro:
Our project in the end had the same basic concept of functionality that was discussed above, but with some major problems. We were able to read the input and display the number but our waiting logic was broken for reasons we could not at all figure out — for example, we would display Good for a quite long time, and then even though the counter to determine when to

switch was still programmed to go up at the same speed after reaching the next value we were waiting for, it would speed through the next few words then hang on the display of the score. Similar things happened with the incorrect-answer state.

One other bug with our project was that the first random number would always display “000” because the random number was generated off of the clock (after the first the number would vary); there is a function to get random numbers in Verilog but it didn't seem to be working at first, though probably without the other problems out of the way we could have gotten it working fairly easily.

All in all, while it seemed somewhat easy on its face when picking it to do, this was quite an interesting project to work on. Handling the complexity of meshing sequential and combinational logic is tough (and Verilog can be stubborn at times)!