Colin Cammarano and Stephen Sher

May 1, 2014

EE 201L

Final Project: Report

# A Binary Game—VGA Ready

**Introduction:**

Everyday, more than 344,124,450 people use a computer to accomplish their daily tasks. Understanding how this technology works in our rapidly advancing society is vital for becoming successful in later years. The primary goal for our EE 201L final project was to design a fun, easy to learn, binary to decimal conversion game, using the FPGA board's SSDs, memory, and VGA output capabilities. The game is intended to teach the player basic eight-bit binary to decimal conversions. Its primary mode is a little game where the Nexys-3 SSDs display a number in decimal, and the player uses the switches and push buttons to input a number in binary. This method can effectively teach the player the basis of binary to decimal conversion, allowing them to easily work with larger binary numbers. Furthermore, such a game can serve as an excellent review for more experienced players. By playing this little game, and optionally using the practice mode, the user can gain a better understanding of binary, which forms the basis of all digital logic.

**Overview:**

The binary game, which forms the basis for our project, can be broken up into an array of smaller systems. The backbone of this game is the Mealy state machine, which drives its core logic. The machine iterates through thirteen different states, ranging from initial states, menu states, gameplay and practice states, and a score display state. These states transitions depend on both user input and the values of registers within the machine. For instance, the main "play" state transitions to a "number generation" state or "done" state depending on the previously generated random number. The second main component in our project is our random number generator, which generates an eight-bit number between zero and two hundred and fifty-five. Our project's top end implements a variety of smaller components. The first of these components are the five debouncers that handle single-clock-enable pulses (SCENs) when an FPGA push button is pressed. Another important aspect of the top design is the VGA control system, which sends video signals to a monitor depending on the current state of the game. User input is handled through the use of push buttons and switches, which allow the user to navigate the game states and input a number, respectively. We utilize the Nexys-3 SSDs and LEDs to display the randomly generated numbers, scores, and binary numbers. Lastly, we use registers to hold important gameplay data, including the randomly generated numbers, user input values, scores, and state information. Collectively, these tightly coupled systems drive the logic behind our game.

**Solution:**

Our project had a variety of goals that each demanded its own working implementation. Our first major milestone was setting up a proper version control system for the project on GitHub. This allowed us to collaborate on our code, track our changes, and keep our project in one safe, central location. Since Modelsim and Xilinx produces a lot of extra files, we only pushed the verilog and UCF files. Those were sufficient enough to sync the two instantiations of the project.

In order to maximize efficiency and prevent working on the exact same element of the project and producing conflicts, the project was sorted as follows: One member would develop and code the state logic, state transition, and top file; the other member would code the testbench and run tests to make sure the design works without any bugs. This "mini-production line" maximized our efficiency and allowed us to work without any conflicts.

The mechanics of the product is as follows. Button-up is the reset button; button-down is the back signal; button-left is to transition left in the menu; button-right is to transition right in the menu; and button-center is the select / confirm signal. Each eight of the switches acted as the binary interface, where switch-up is a one and switch-down is a zero. Each LED corresponding a switch will light up when the switch is on as a confirmation of which swtiches are on and which are off. In the menu states, pressing left and right allows the user to transition between the different menus, and pressing the center button will allow the user to enter the selected menu option. Play will generate a random number between zero and two-hundred fifty-five and display it on the SSD, and the user will need to input the corresponding binary equivalent with the switches. If the user gets the binary number correct, the game will loop and generate a new random number. If the user gets the binary number wrong, the game will exit and show the highest score. The reason why we decided to only show the highest score is to motivate the user to try to beat their own scores, and therefore always improving their binary interpretation skills. Practice state will go into a state where the SSD will display the decimal equivalent of the binary input from the switches. The user can use this to get familiar and experiment with the binary system. The scores will simply display the highest score in the current instantiation of the game (will reset upon resetting the chip). Menu quit will bring the game to a done state, and remain there until an acknowledgement is made.

**Discussion:**

There were a variety of challenges that came from our binary game design. The first major hurdle we had to overcome was implementing and testing the generation of large, random numbers. Such a system was difficult to test, since the number generated each time was different. Another major challenge was setting up the top file to correctly send VGA signals to the monitor. Learning how to enable the VGA symbols in the UCF file and correctly positioning the coordinates on screen required a significant amount of trial and error. Perhaps the largest challenge we faced was converting the binary signals produced by the Verilog code to BCD, so decimal signals could be output correctly on the SSDs. We ended up using two-hundred fifty-six if-statements to generate the BCD. While one of us tried to get the BCD conversion working, the other member coded a C++ program to automatically generate if-statements in Verilog to convert binary into BCD. The C++ generated code ended up finishing faster than the conventional BCD conversion, and therefore we simply copy and pasted the C++ generated code; the code worked in one try, had fast build time, and therefore we decided to keep this approach.

**Conclusion:**

The project has been an excellent learning experience, demonstrating a variety interesting and uncommon problems. As previously stated, we had to implement interesting workarounds for testing our random number generator. This taught us the power of the Verilog test fixture in debugging designs, especially since we could use a testbench to see the result of the random number generator without having to synthesize the incomplete design. The second major hurdle that we learned to overcome was learning how to use the Nexys-3 to send VGA signals to a monitor. Learning the method in which monitors synchronize their displays was incredibly interesting. The last major learning experience came from developing the code to transition from binary to BCD. Verilog has no apparent way to do this, and our algorithms were to complex for the synthesis tool to process in a reasonable timeframe. Developing algorithms to convert binary to BCD was by far our most interesting problem to resolve.