User vs AI Postulation of Binary Vector System

Trevor Hayes

Sam Bray

Jonah Snyder

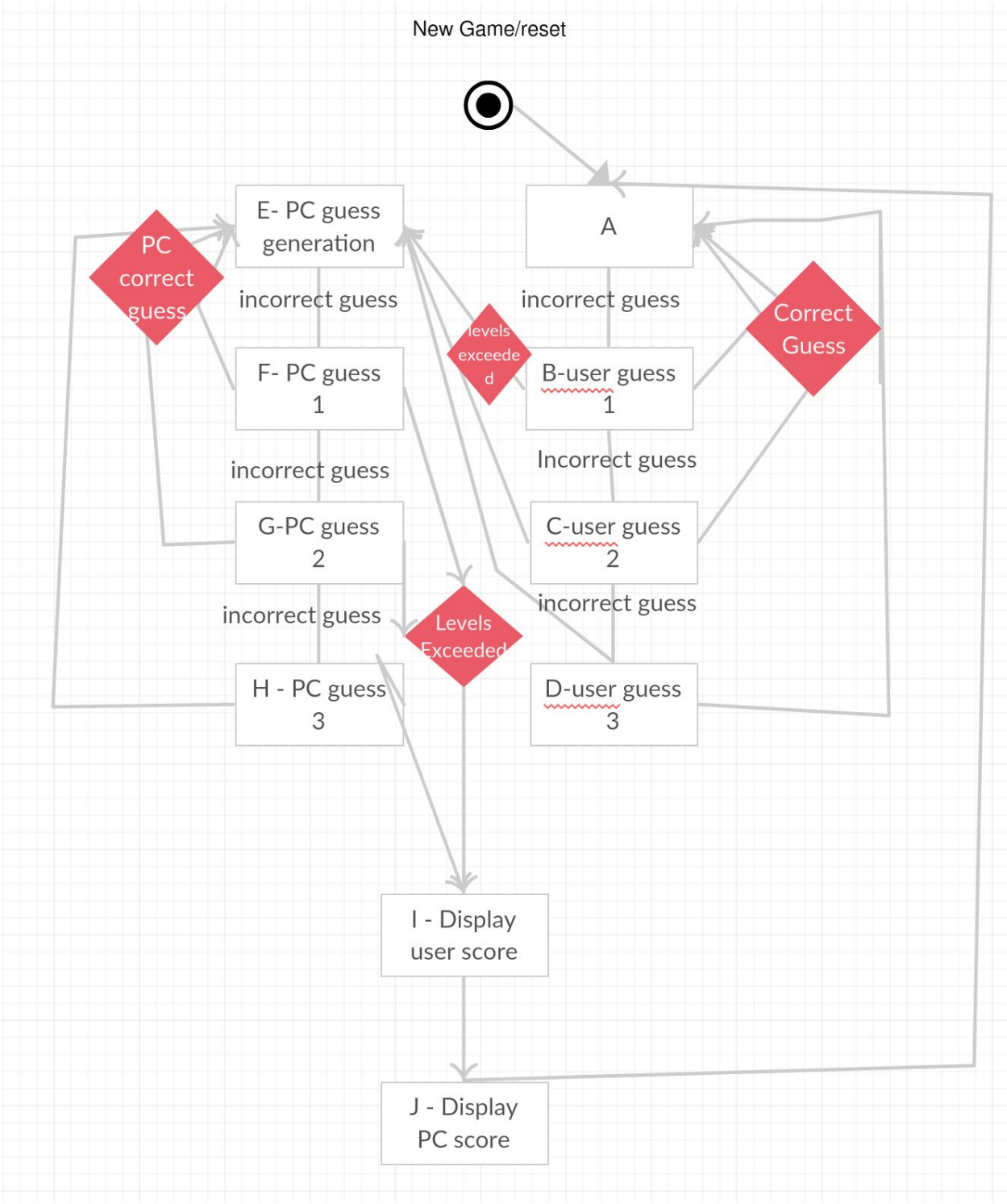
Introduction

Our FPGA project is to build a number guessing game, in varying range. So first the board would generate an int and then blink once it is ready then the user would have three guesses to get it right, guessing by flipping switches to denote the binary version of the guess, and then pressing a button to lock in the guess. Indications of higher or lower will be displayed on the crystal display. Then the user would change their guess again and the board would respond accordingly. If the user gets the number correctly, the board lights up and the random number is changed to one with an extra added bit. Initially the board will start with 2 bits, and will go up the maximum that can be displayed. After the user finally loses their streak, it becomes the FPGAs turn to guess numbers that the user sets, with the same rules for increasing bits. After each guess the user will denote higher or lower to the FPGA with either a switch or one of two buttons, depending on what seems more user friendly in implementation. This goes on until the computer loses and their scores are compared based on the highest level achieved and the number of guesses needed total. So the user essentially competes with the FPGA.

Design

The algorithm for guessing should be similar to the algorithm used in successive approximation, however slightly more complicated. In successive approximation, it takes n times to find a number of n bits each time, this is easily exploitable if a user is to set the number to the highest option each time, narrowly beating out the computer response time, this is avoidable by adding a random element to each guess in the form of making each calculated guess +- 0-50% of the initial interval, thus making each time slightly different and up to luck, meaning if the same number is selected it could take more or less than the standard n bit guess time, allowing room for competition and stretching the limits of play. Research will have to be done to find the best way to generate a psuedo-random number in VHDL, but realistically that should be rather doable. The algorithm for score keeping should just be the level x (some constant C - number of tries at level), this way it scales more points per each more difficult level and is still quite readable and comparable, and encourages faster guessing.

State Diagram



Contributions

Trevor Hayes: I helped with the develop overall structure of the Finite State Machine. Also helped with the debugging of the coding and implementation to the DE2-115 board.

Sam Bray: I was the primary coder in the partner programming strategy we implemented. I also tackled the debugging process and boosted morale of the troops in the wee hours of the morning.

Jonah: I helped with debugging, coding, and implementation.

Known Issues (if any)

Issues:

1. All of the inputs and outputs are in binary so the user will have to have knowledge into how to count in binary.
2. Creating a random number in VHDL turned out to be way more difficult than we had previously anticipated.
    a. VHDL doesn't allow the use of non constant reals, or really basically any use of reals in synthesizable code. There is, however, the IEE float package which was released for VHDL 2008 that gets around all these issues by allowing the use of floats in their place, which can be easily defined, used, with decent math functionality. Quartus 14.0 does not support this package yet, and therefore the project struggled to get around this barrier for random number and algorithm generation and the clock had to be used with ints to calculate the numbers to guess. Usable algorithms and code with reals were implemented, but can only be used for test bench of the product.
    b. The DE2115 FPGAs also suffer from some misfiring between the clock pins and notoriously get blocked up and either break or freeze, despite there being moderately sound code. Basically everything has to be debounced or there have to be decent wait periods between any button press or stage transition to ensure the board does not malfunction in conjunction with the synthesized hardware description.

Possible Solutions:

1. Given more complicate input tools like more buttons, we could implement a input system in decimals instead of binary.
2. Finding a suitable third party package to use, or creating our own more useful package (which was worked on a bit before we changed direction) could navigate this problem. Surely there's a way to force feed external ieee packages down Quartus II's logic canals.

Intellectual Property

IP: All the code minus 3 lines of test bench logic are our intellectual property.

Merits:. No outside help within the university was needed in the production or formulation of our project.

Conclusion

Through the development of the number guessing game we were better able to understand VHDL. The most challenging part of this project was not being able to generate random number and VHDL not having the ability to use non constant reals. We were able to work around not being able to use a random number generator by creating a kind of random number generator within the code using an array and pulling values from the array according to the clock. We also found a way around the problems with the non constant reals by changing the guessing.

References

https://help.github.com/articles/about-github-wikis/
https://github.com/braysd/ConvolutedCode.git