Project Report

Nachiket Bhatt, [nachiket.bhatt@mail.utoronto.ca](mailto:nachiket.bhatt@mail.utoronto.ca), 1004703332

Sijia Cai, [s.cai@mail.utoronto.ca](mailto:s.cai@mail.utoronto.ca), 1005281696

Why this project?

We brainstormed a lot of ideas before settling on this project. We though of a graphing calculator for limited amount of functions to angry birds. We settled on this one, called “Cloudy with a chance of meat squares” because it was something unique no one else was doing and it would demonstrate out understanding of the course material since it combined early parts and the newer things taught to us. And it involved some sort of animation, which is always cool.

What did we do (methods)?

We divided the project into three sections, as required. Our first task was to generate pseudo random blocks of two different colors and animate them falling. We created a linear feedback shift register to generate pseudo randomness. It looks random because circuit delays, which are random, contribute to it. We created 30 registers of 58 bits where every 2 bits, tells us if the block is background, red block or green block. We had some trouble with this part since we were animating for the first time and sometimes the refreshing was too quick for the register. But we sorted it out by the end of the lab.

Next, we had to implement user input, which is the mouse input. The “catcher” as we like to call it, needs to catch some the falling blocks and dodge the others. As the user moves the mouse left and right, the catcher moves with it. Here, we made 2 versions of user input: one using the mouse and the other using keyboard. We planned to go ahead with the version which worked better, which was the mouse version. Although, the refresh rate is slow as the FPGA and VGA adapter can only handle so much, it’s smooth and we’re proud of how it turned out.

Last, we needed a way to calculate points and lives (as in when it’s game over). For calculating the points and lives, we created another module, which tests for the position of the mouse and the block above it. If it’s a red block, we give plus one point and minus one life if it’s a green block and do nothing if it’s empty. The real hard part was to update and display the points while the game is going on. We’ve previously learned how to create a hex decoder, but for this we created a decimal decoder. We created our own system to communicate with other modules to display the number we created. And we do have a game over screen, which was hard in its own way, since you have to create another controller just to display those specific characters.

In the end, it did work. In retrospect, if we had more time from other courses, we would’ve made it significantly smoother. And, after finishing the project, we are better Verilog coders than we ever were. So now, creating a similar project would be easier since we are now better. One thing, if anything, we learnt the hard way from this assignment is that Verilog, VHDL System Verilog etc. are hardware description languages and not OOP languages. To be a good Verilog programmer, one must think of how it translates to hardware. If I am using a for-loop in my code, I must know how it will be synthesized by the FPGA to make better and working designs. Another thing, which we take for granted, is the frequency of a clock. My current PC currently has a 3.3 GHz clock inside it which is way more than the 50 MHz we get. More clock cycles means a smoother device, if used correctly.

Something we would’ve know at the start was that you can create multi-dimensional arrays in Verilog. That would’ve helped in the huge amount of registers we have in out project. Another thing differently we would do is use rudimentary Verilog operations more often i.e. using less for-loops because, as we stated earlier, Verilog is not a OOP language.