CSM152A – Project – Doodle Fall

Names: Jahan Kuruvilla Cherian; Kevin Xu

Group: Lab 6 – Group 9

UID: 104436427; 704303603

Date Performed: 6/07

TA: Babak Moatamed

1.0 INTRODUCTION

The final project as part of this course was meant to unleash our creativity and newly solidified knowledge of Verilog and FPGA design to build a product that we could be proud of. For this project, we decided to undertake the building of a game, that was heavily influenced by the popular iPhone game – Doodle Jump.

The initial idea was to replicate doodle jump, but instead to do so through the FPGA. Upon undertaking the project, we realized that due to the time crunch a more feasible project would be *Doodle Fall* where instead of bouncing up blocks and shooting monsters, the doodle now fell downwards upon the blocks and tried to stay alive for as long as it could.

Due to this change in idea, the initial project proposal was amended slightly, but for the most part the rubric remained constant.

To build out this project, we wished to incorporate the VGA for an immersive playing experience through visuals on a monitor, a PMOD Joystick to allow for smooth and comfortable playing, the seven segment display to display the scoring of the game and finally a set of three buttons to reset the game, switch display mode between current score and high score, and a button to reset the high score.

The entire project was written entirely in Verilog, and drew some inspiration from legacy code for the implementation of the VGA and PMOD Joystick.

2.0 GAME DESIGN

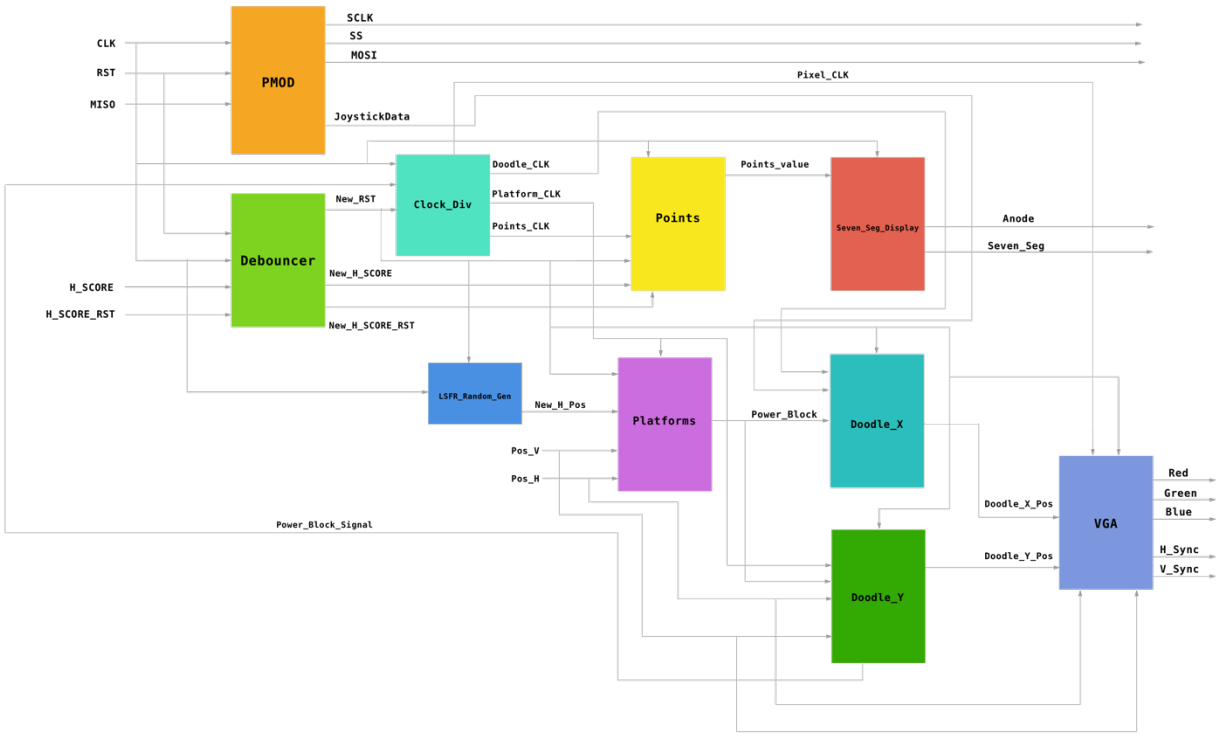

Figure 1: High Level Block diagram of the game design and logic. The key modules implemented in the top module of the game are highlighted with their general inputs and outputs labelled as well.

The game was designed to be an endless free falling game. The VGA Monitor displays a vertical section wherein the doodle's world exists with a set of randomly generated blocks move upwards at an ever increasing pace, while the player has to move the doodle sideways to fall onto blocks at a lower level in order to stay alive. There exists power-blocks that upon contact will slow down the speed of the game to allow the player to live on for longer. The scoring mechanism works as a simple counter, counting the amount of time you remained alive for. Contact with either the upper or lower bounds results in instantaneous death, and requires reset to be pressed to restart the game play. The high level block diagram for the entire game exists in Figure 1 above.

3.0 PROPOSAL RUBRIC

As mentioned before the initial idea of the project shifted slightly as the project moved forward, and so in this section I will list out the initial grading rubric, and transition to the slightly modified new rubric.

The initial rubric stated the following:

1.) Doodle and Platforms (40%) – Implement the doodle character onto the screen along with randomly generated platforms the doodle can land on. The character must be jumping at regular intervals.
2.) Endless Jumping (30%) – Make the game so that passing a certain vertical threshold will move the game screen up by a certain offset, in order to emulate an endless game experience.
3.) Scoring (10%) – Implement the scoring functionality based on regular increments of jumps onto the seven segment display.
4.) Game Restart (10%) – Implement the restart functionality of the game, so as to reset the display and character, with newly generated maps and a fresh start score.
5.) Sound (10%) – Incorporate small clips into the gameplay, connecting it to speakers on the board, for when the character jumps, shoots and kills.

The new rubric upon which the game was built and the report written, is as follows:

1.) Doodle and Platforms (40%) – Implement the doodle character onto the screen along with randomly generated platforms the doodle can land on. The character must be *falling* at regular intervals.
2.) Endless *Falling* (30%) – The game should have an endless playing mode such that the doodle can continue to fall and randomly generated blocks at regular intervals.
3.) Scoring (10%) – Implement the scoring functionality based on regular increments of falling onto the seven segment display.
4.) Game Restart (10%) – Implement the restart functionality of the game, so as to reset the display and character, with newly generated maps and a fresh start score.
5.) Extras (10%) – To replace the necessity of sound, add new features, such as increasing difficulty, high scores and power ups.

3.1 DOODLE AND PLATFORMS

The doodle character was created as a green square of size 20. To maintain the boundaries of the doodle character, we created two wires that represented the x and y positions of the doodle within the overall screen. This point was taken as the lower left corner of the doodle character, and all subsequent collisions were measured with respect to this point.

The platforms themselves were generated in increments of 7 per screen segment, with a height of 20 pixels and a width of 75 pixels.

Before we talk about the implementation of the blocks movement and the doodle movement, it is worth mentioning how the display was actually set up through the VGA. The main aspect of the VGA module that we worked on was the checking for the value of the vertical and horizontal counters with respect to the front and back porches for the top/bottom and left/right points of the screen. These counters essentially maintained a scanner that moved through each row's column pixel to display each individual pixel.

We created the game to emulate the iPhone vertical screen, and so the width of the playable screen was set to 300 pixels while the height remained the original height between the front and back porch in the vertical direction – 480 pixels. This section was drawn to have a black background, blue blocks (and a green power up block) with a green doodle character, while the rest of the surroundings was set to a maroon color. The areas beyond the boundary of this vertical section was set to a solid state such that the doodle upon collision would be blocked from moving further left/right than the wall boundaries.

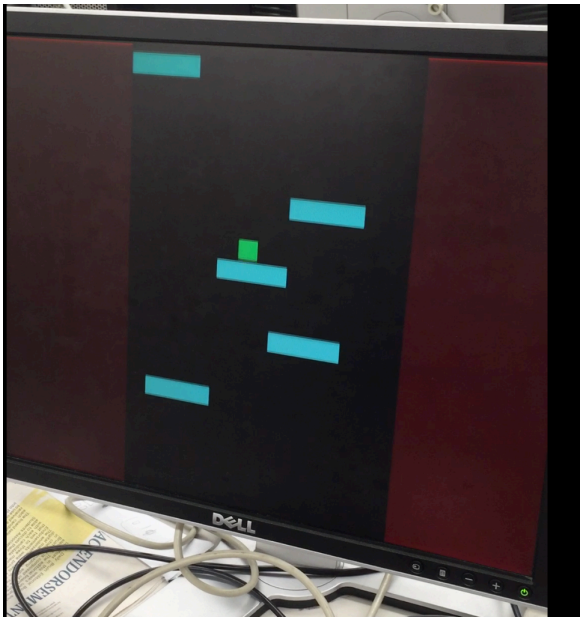

Figure 2: Image to show the vertical screen, randomly generated platforms and the doodle square resting (colliding) with the platforms.

When it comes to the platforms, we focused on the vertical and horizontal positions, by creating a total of 14, 10-bit registers for the 7 vertical and 7 horizontal positions respectively. We had a platforms module that took care of the movement of the platforms upwards, but the implementation of this will be delved with in further in the next section. More importantly in this section of the rubric, we focus on the doodle itself and its collision detection with the blocks.

As mentioned earlier, we maintained an x and y position for the doodle. Thus we created two separate modules, dealing with the horizontal and vertical positions respectively.

The horizontal (x) movement was slightly easier. Because the data that was gathered from the PMOD Joystick module contributed only to horizontal motion, what we did was check the values from the register that held the horizontal positioning of the joystick for respective bounds that determined a left or right movement, and compared that with the horizontal back and front porches to check for collisions with the left and right walls respectively. We also checked the positioning data between a set of values so that if it crossed less than 237 or more than 787, then we would increase the horizontal speed of the doodle. This gave the game the feel of a precision movement and a quick movement. Note that the side collisions were calculated by taking the current x position and offsetting it by the doodle size (for right-wards collisions).

The vertical (y) movement became a little trickier as we had to now check for collisions with each block. Running through the clock, we would increment (fall down) the y position of the doodle if its lower left

corner left the threshold of the platform, and would make it fall at 2 pixels per second (this was our gravity for the game). Because we created 7 platforms per segment, we multiplexed through all the 7 different vertical positions to check for the collision against the bottom of the doodle. This was as straightforward as checking if the current y position of the doodle was within a small (~3 pixel) range from the top of each platform. This way we created a collision system. Note that we also had a special check for the special power up block separately in this module, but we shall go into that in more depth in section 3.5.

After we accurately set the 10-bit x and y wires in the top module through the individual modules mentioned above, we then just created an *if* statement within our VGA module to check for the position of the doodle, and then display the green square at that point.

And so using these techniques, we created a doodle character that was correctly displayed on the screen based on its x and y positions as controlled by the PMOD Joystick horizontal data, while creating vertical collisions for allowing the character to land on a block, and horizontal collisions to prevent the doodle from leaving the game world vertical screen.

3.2 ENDLESS FALLING

In order to generate an endless falling mechanism, we made the platforms move upwards, by decrementing their y position at a platform clock speed. Upon the platform's y position being less than or equal to the vertical front porch (top of screen), we reset its y position to be at the bottom of the screen, but also reset its horizontal position to be a pseudo randomly generated number, to introduce a randomness factor to the game.

To do this we focus on the platforms module and the lfsr random generator module. To begin with we discuss the random generator. This was implemented using a *Linear-feedback shift register* of 16-bits, that shifted its bits and XOR'ed them, after which we OR'ed the first and last bit to get a truly random value. But because this was implemented as a 16-bit counter, the value could be much higher than made sense, and so we used the modulo operator to randomize the number in a range of 225.

Using this randomly generated horizontal position, we enter the platforms module wherein using the 7 different vertical positions (corresponding to each of the seven platforms per segment), we check if the value passes below the front porch and if it does, we simply set its value to the back porch (bottom of the screen), and at the same time, set its corresponding horizontal position to the randomly generated new horizontal position that we got from the random generator discussed above. Within this module we also calculated the probability of creating a power block (simply by modding the randomly generated power block position with 10) and upon doing so, used a 7-bit register where each bit corresponded to the respective platform, to see if it was a power block. We use this information later on when creating the power ups.

The aforementioned methods allowed us to create a seamlessly endless game. We also used the platform clock to make the game go faster to increasingly make it more difficult by constantly decrementing the counter threshold for the platform clock within the clock divider module until it hit a max speed. This will be mentioned in greater depth in section 3.5.

3.3 SCORING

The scoring module was implemented in a similar way to that built in Lab 4, through the use of a seven segment display module, an anode counter and a binary decoder to decode the value to the relevant bits to appear on the screen.

The scoring was calculated at every positive edge of the points clock (10 Hz) wherein our points counter would increment by one. This therefore emulated a scoring function based on how long the player could survive in the game. Thus the game's score depended directly on the falling increments of the player, because if the player didn't fall, or fell too far, he/she would die upon collision detection of the doodle with the vertical front and back porch.

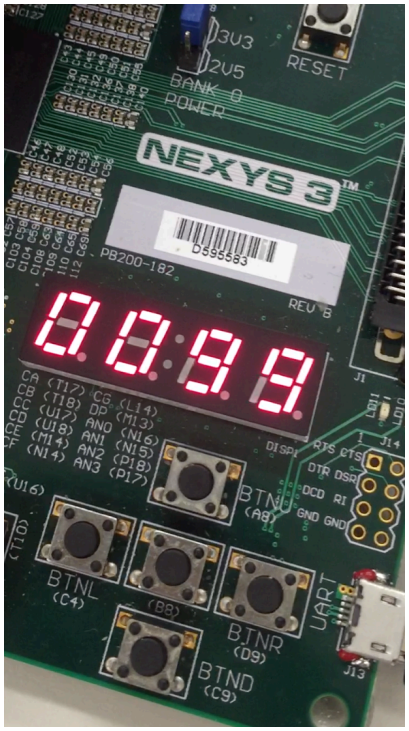

Figure 3: Image showing the points counter acting as a scoring mechanism on the seven segment display on the Nexsys 3 Board. We see the current score set to 99 at this moment.

3.4 GAME RESTART

The Game Restart functionality was implemented using techniques learnt throughout the entire quarter, wherein we implement all our sequential always blocks with the possibility of encountering an asynchronous positive reset upon which we re-initialize the entire scene.

The only key difference is that, upon a reset, we update the lfsr random generator to a random number, so that upon each reset we generated a new and different map. The doodle is also randomly generated on any of the middle platforms.

The reset functionality was debounced to create a responsive button click, using a debouncer that was similar to the one built for Lab 4. This involved creating a counter that counted until a limit before which it set the value of the button pressed to the complement of its current value. This essentially stabilizes the button click.

The reset functionality was especially needed when the player dies upon collision with the top/bottom. Upon such a case, we send a terminated signal that controls the platform module. If the terminated signal is received then we don't move the platforms upwards and we stop incrementing our points counter, thereby stopping the entire game. Upon a reset, the terminated wire is set back to 0 and the game resets at its new initial state.

3.5 EXTRAS

In order to compensate for no sound module being created due to time constraints, we aimed at making up the 10% by creating three extra game features. These include increasing difficulty, a high score function and the power up blocks.

Starting with the increased difficulty, we decided to modify the Platform clock every clock second by decrementing its counter threshold by 1000. The Platform clock starts at the counter value of 500,000 and so decrementing it by 1000 each time over essentially speeds up the clock and so the entire game

that is dependent on the vertical movement of the platforms speeds up, making it more difficult for the player to survive. The max speed hits a critical point when we check for its value at 200,000. This functionality made it more likely for the player to lose the game and allowed for a more competitive environment with other players.

We then decided to use this added competitive edge to create a high score feature. Essentially we create another points module that registered the highest score into a 32-bit high score register. If the points ever passed this score, then the high score was set to this value. This value was then used to display onto the seven segment using another seven segment module, where the high score can be toggled by clicking the button above the reset button. In order to allow for new high scores to be added, we also added a high score erase button (below the reset button), which essentially cleared out the high score register. Both of these buttons were debounced to make them responsive, just like the reset button.

Finally, we implemented a power up block, which was colored green to distinguish it from the other blue platforms. If the doodle landed on this block, then the platform clock counter would reset back to its initial value of 500,000 to make the game back to slow mode. There were quite a few small components around the entire module to make this feature. We start with creating a probability of a power block appearing within the platforms module, and if so we set that specific platform to the power block by setting its corresponding bit within a 7-bit power register. Upon detecting a collision with this specific block in the doodle y module, we send a single bit signal to the clock divider module wherein we reset the Platform clock back to its initial value. Finally, we check for each platform within the VGA module whether it is the power block, and if it is, then we color it differently.

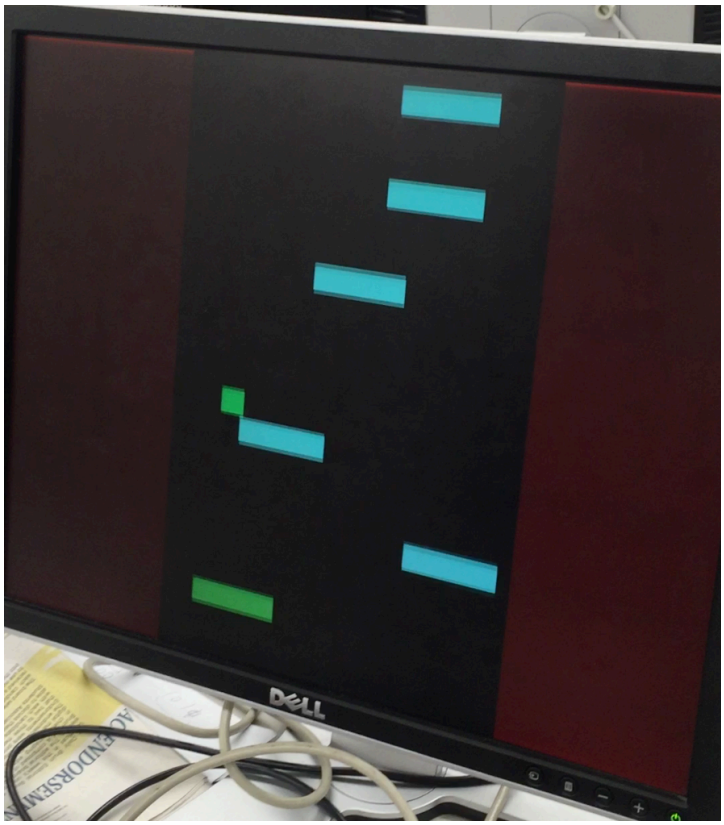

Figure 4: Image showing the power block as highlighted in green. Upon a collision with this block the game slows down, and thus acts as a power up, that occurs every so often.

Through the implementation of the 3 extra features mentioned above we noticed that the game became a lot more enjoyable and a lot more interesting as it now brought the randomness of winning/losing the game to reality.

4.0 CONCLUSION

The entire project was an enjoyable experience and resulted in a rewarding game that is both fun to play and relatively complex. From learning how to read in PMOD Joystick data, controlling the VGA display bits and creating a collision detection system with a random number generator all in Verilog served to be quite challenging tasks.

It is worth noting that though our rubric changed from its initial state, the newer version of the rubric remains just as difficult and in our opinion, adds more complexity and fun to the entire gaming experience.

As far as we were concerned, we believe that we met every requirement from the modified rubric and that the results of this were clearly evident throughout the demonstration of the project.

5.0 SOURCES

As mentioned before, the modules for the PMOD Joystick and the VGA were gathered and modified from external sources. The two links below provide the location of these publicly available sources, before our own modifications.

VGA (Download the NERP_demo.zip):

https://pumpingstationone.org/2013/04/nerp-fpgaok/

PMOD Joystick (Download the Verilog zip file):

https://reference.digilentinc.com/pmod/pmod/jstk/example_code