Introduction and Requirements

The ultimate goal of our project was to implement a variation of the game Frogger, with our own twist. To implement this game, we wanted to allow the user to move their avatar (the "frog") with the buttons on the board. If the user was able to successfully move the frog across the screen, while avoiding all of the obstacles that we implemented, then they won. The user would then be able to see how much time it took for them to complete the game, as well as their high score (the fastest time). The frog, obstacles, and stopwatch were all to be displayed on a VGA monitor. In terms of specific design requirements, we had five, as follows:

- Frog movement functionality - The user had to be able to use the four outer buttons on the board to move the frog up, down, left and right.
- Obstacle movement functionality - The obstacles had to move left and right across the screen at differing speeds.
- Collision functionality - Whenever the frog was hit by an obstacle, the frog would be sent back to the beginning.
- Game display functionality - The VGA monitor had to display the game setting, frog, obstacles, and stopwatches properly. The display had to accurately display the user's inputs.
- High score functionality - Whenever the user gets across the screen is a new fastest time, the high score stopwatch changes to display the high score.
- Reset functionality - Upon clicking the reset button (the center button), the frog is sent back to the starting position stopwatch tracking the current time is set back to 0.

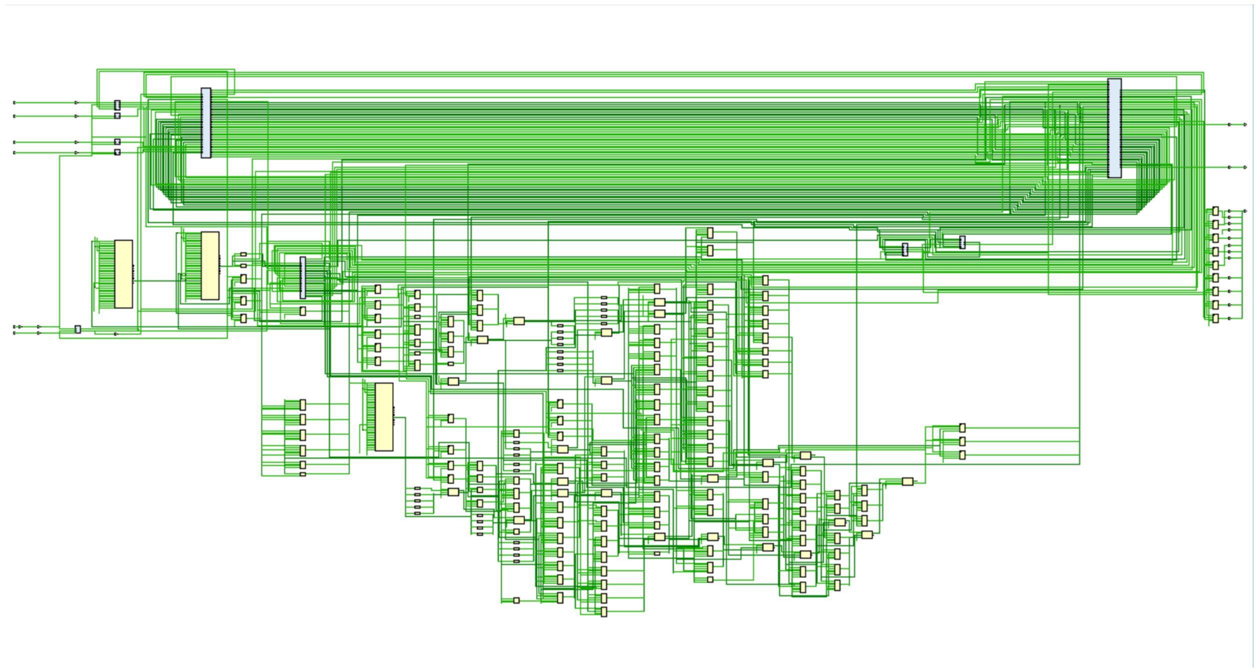
Design Description

To assist in our implementation of the project, we imported some modules which created a stopwatch on a VGA monitor. These modules created a stopwatch which counted the hours, minutes, and seconds since beginning. However, in order to make the necessary changes to the modules to fit our design requirements, we had to familiarize ourselves with how the modules worked and interacted with each other. A description of what each module relating to the stopwatch did and how it interacted with other modules is as follows:

- Clock_digit_rom - This module created the necessary encodings for each digit 1-9, so that the digits could be displayed on the VGA monitor. This module was called by the pixel_clk_gen and pixel_clk_gen_best modules, and the correct encoding was returned depending on the input.
- Pixel_clk_gen - This module determines where on the screen each of the individual digits should be placed, in order to create a string of 6 digits which represent the stopwatch. This module also sets the RGB values for the digits, so that they are actually displayed.
- Pixel_clk_gen_best - This module was custom made by us (it was not imported), but it was very similar to the pixel_clk_gen module. Rather than using a counter to set each of the digits to the appropriate value, it received inputs which represented the best time the

user had achieved. The module then used clock_digit_rom to get the correct encodings of these digits, and then displayed them as done in pixel_clk_gen.

- New_binary_clk - This module is where multiple counters were implemented to simulate multiple clocks. Originally, it was used to calculate hours, minutes, and seconds. However, we altered the code to calculate and output minutes, seconds, and milliseconds, as this was a more appropriate timescale for our game.
- Vga_controller - This module set up the VGA monitor to be displayed, based on physical aspects of the monitor such as resolution, front porch, back porch, and retrace. The result was a display on the monitor, with the contents of the display being determined by our other modules.
- Top.v - This module is where each of our other modules were put together and instantiated. While this was originally imported from the stopwatch repository, we altered this file to show both our frogger game (this module is described below) and our 2 modified stopwatches.
- Btn_debounce - While this module did not directly impact our game's implementation, it was necessary to include since our game required the user clicking buttons. This module simply debounced the button clicks.

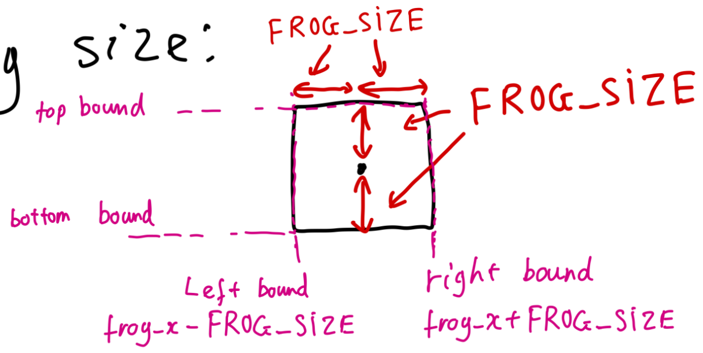


We also created a frogger module, which is where the gameplay itself was implemented. Its description is as follows:

- Frogger - This module is designed to generate pixel color for the frog and obstacles. It takes inputs for player movement and outputs RGB color for display.
    - Frog Control:
        - Movement of the frog is controlled based on button inputs (up, down, left, right) within the game boundaries.

        - Collision detection with obstacles is performed to reset the frog's position if necessary.
        - When the frog reaches the top of the screen, the game indicates a win (is_win).
    - Obstacle Movement:
        - Obstacles move horizontally across the screen within the defined boundaries. Their speed and direction is determined by OBSTACLE_VELOCITIES and obstacle_direction
        - Obstacle_direction is updated upon reaching the board's edges.
    - Pixel color:
        - If the current pixel coordinates match the frog's position within its defined radius (FROG_SIZE), set the RGB color to FROG_RGB to display the frog.

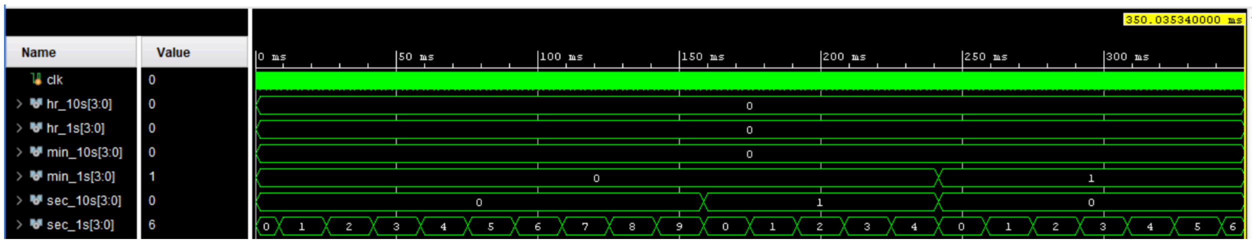
Frog size:



We check if rendering pixel (x,y) is between these bounds to set the RGB accordingly.

        - When the pixel coordinates match an obstacle's position within its radius (OBSTACLE_SIZE), assign the RGB color to OBSTACLE_RGB to display the obstacle.
    - Winning check:
        - When the frog successfully reaches the top of the screen (i.e., when frog_y - FROG_SIZE <= board_Y_MIN), the game considers it a win.
        - Upon detecting this condition, the module sets the is_win signal to 1, indicating that the player has won.
        - Additionally, the frog's position is reset to its starting position (FROG_X_START, FROG_Y_START).
    - Losing check:
        - If the frog collides with any obstacle, detected by overlapping their bounding boxes, the module resets the frog's position to its starting point (FROG_X_START, FROG_Y_START).

Simulation Documentation



Conclusion

In summary,

One difficulty that we encountered was ‘’

- Verilog handled 2D array shifting oddly. When we tried to shift bits in the array, it would just set each entry to 0 rather than actually shifting.
- Trying to create our high score clock display. Originally tried to instantiate 2nd instance of Pixel_clk_gen in top.v but this didnt work. Solution was to create a Pixel_clk_gen_best module.

Citations

Original Frogger Game
Stopwatch Repository
Bouncing Square Repository