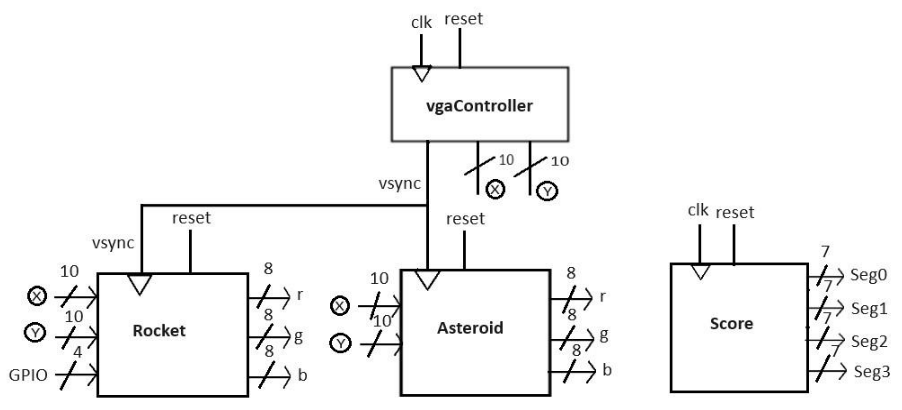
UNIVERSITY OF NEVADA LAS VEGAS. DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING LABORATORIES.

Class:	CPE200L 1001	Semester:	Spring 2024
Points		Document author:	Alireza Bolourian
		Author's email:	bolouria@unlv.nevada.edu
		Document topic:	Final Project Report
Instructor's comments:			

Goal: The project's objective is to apply the knowledge acquired in the CPE 200 laboratory to develop an interactive game. This will be achieved by programming in SystemVerilog, facilitating inter-module communication, and leveraging an FPGA board paired with a VGA for visual output.

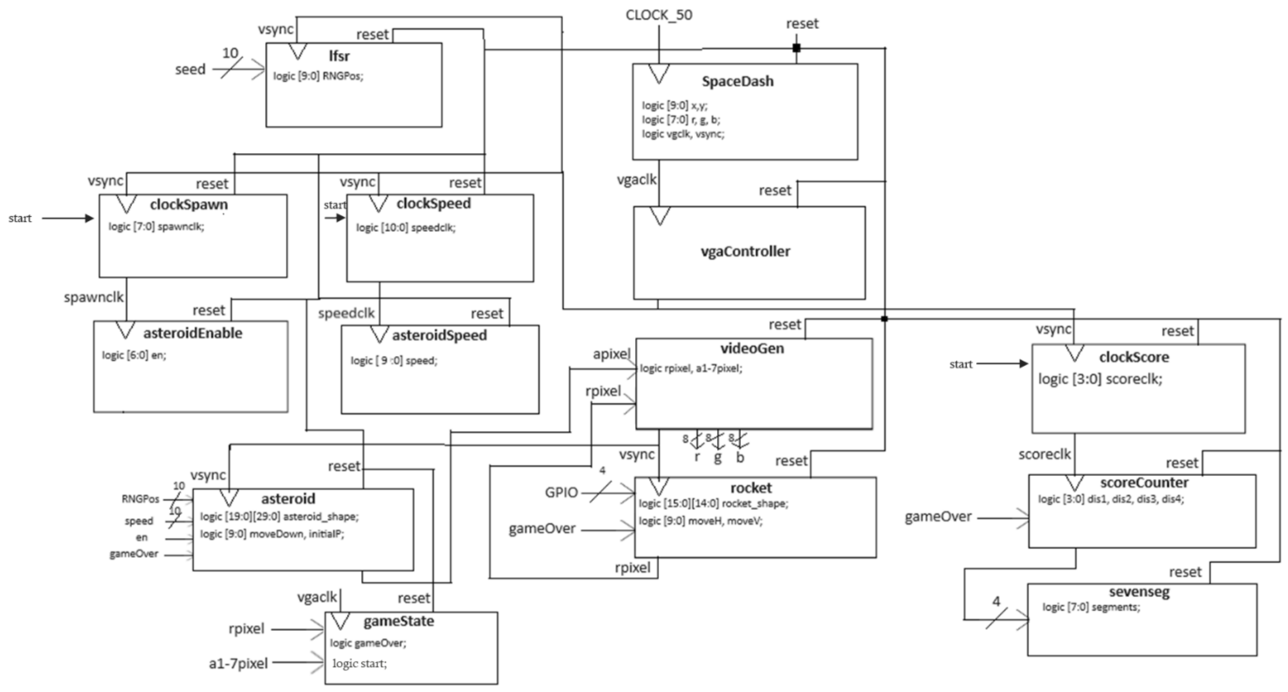
Abstract: This project presents the implementation of a classic arcade-style game on a Field-Programmable Gate Array (FPGA). The game simulates a rocket navigating through a field of asteroids, requiring the player to skillfully maneuver the spaceship to dodge incoming threats. Utilizing SystemVerilog, the project focuses on core game mechanics such as player-controlled movement, vertically descending asteroids with randomized spawn positions, and real-time collision detection. It employs linear feedback shift register (LFSR) to generate randomized asteroid trajectories and leverages clocks to dynamically control asteroid speed for difficulty scaling. A finite state machine (FSM) manages the overall game state. The score is displayed on the LCD display of the FPGA. The project aims to demonstrate fundamental game development concepts using FPGAs while delivering a retro-inspired, engaging gameplay experience.

Black Box Diagram



The black box diagram shows that the system design comprises of four major components. The VGA Controller cycles through the pixels and outputs the x and y pixel values. These values are used inside the rocket and asteroid modules to determine the pixel color values. Additionally, the VGA Controller creates the vsync signal with frequency of 60Hz that can be used to update the positions of the objects inside the modules. The rocket module controls the spaceship and responds to player inputs. The asteroid module is responsible for asteroids falling down and spawning them back on the top of the screen. The score module acts as a counter that displays the score on the LCD display of the FPGA.

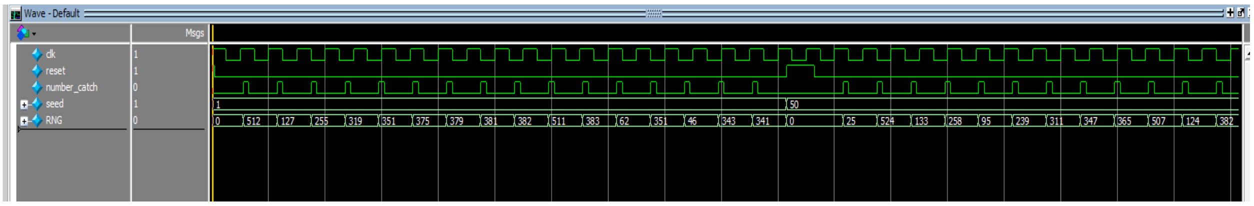
Block Diagram



Modules Overview:

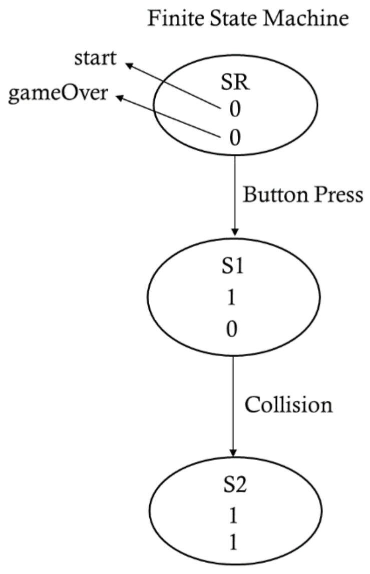
- **SpaceDash Module**
    - Generates the vgaclk signal.
- **VGAController**
    - Cycle through the pixel values in the x and y directions.
    - Generate timing signals for pixel display.
- **VideoGen Module**
    - Instantiates other modules and handles signal routing to other modules.
    - Determine color of pixel using RGB values

- **Rocket Module**
    - Defines the rocket's visual representation.
    - Handles player input into movement of the rocket at 60 frames per second based on inputs from GPIOs connected to pushbuttons corresponding right, left, up and down movements.
- **ClockSpawn and AsteroidEnable Module**
    - Generate a clock for enabling asteroids at the interval of one period.
- **ClockSpeed and AsteroidSpeed**
    - Generate a clock for increasing asteroid speed at the interval of one period.
- **LFSR module**
    - Receive the initial seed and apply the function to generate a random number for the position of the asteroid each time it falls below the screen.


Wave - Default
Msgs
clk 1
reset 1
number_catch 0
seed 1 1 50
RNG 0 0 512 127 255 319 351 375 379 381 382 511 383 62 351 46 343 341 0 25 524 133 258 95 239 311 347 365 507 124 382

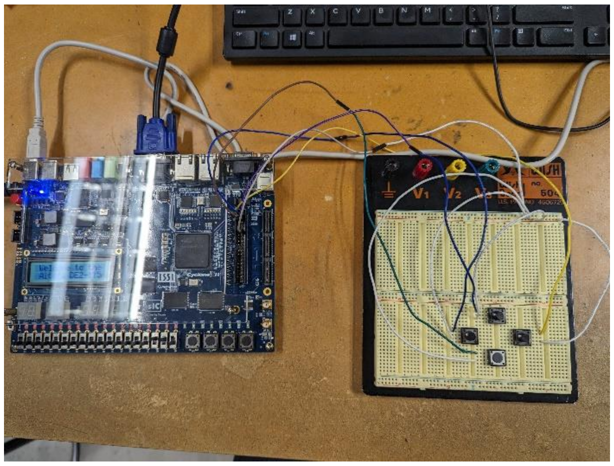
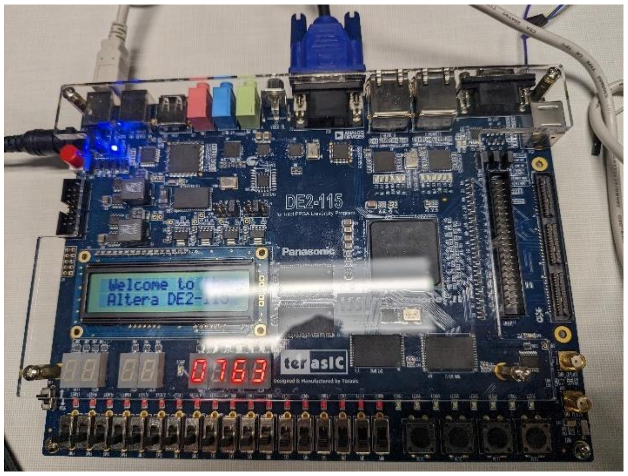
The testbench for lfsr shows that random numbers are generated within the range of 0 and 640.

- **Asteroid Module**
    - Spawning the 7 asteroids at random pixels given by the lfsr module
    - Controls the movement pattern (straight down)
    - Adds speed to the asteroid movements for difficulty scaling.
- **GameState**
    - Finite state machine responsible for keeping track of the state of the game.
    - Initially the game awaits a response from the player in the form of pressing any buttons.
    - Then the game starts, and the clocks are initiated.
    - When collision is detected enters state s2 and gameOver is asserted

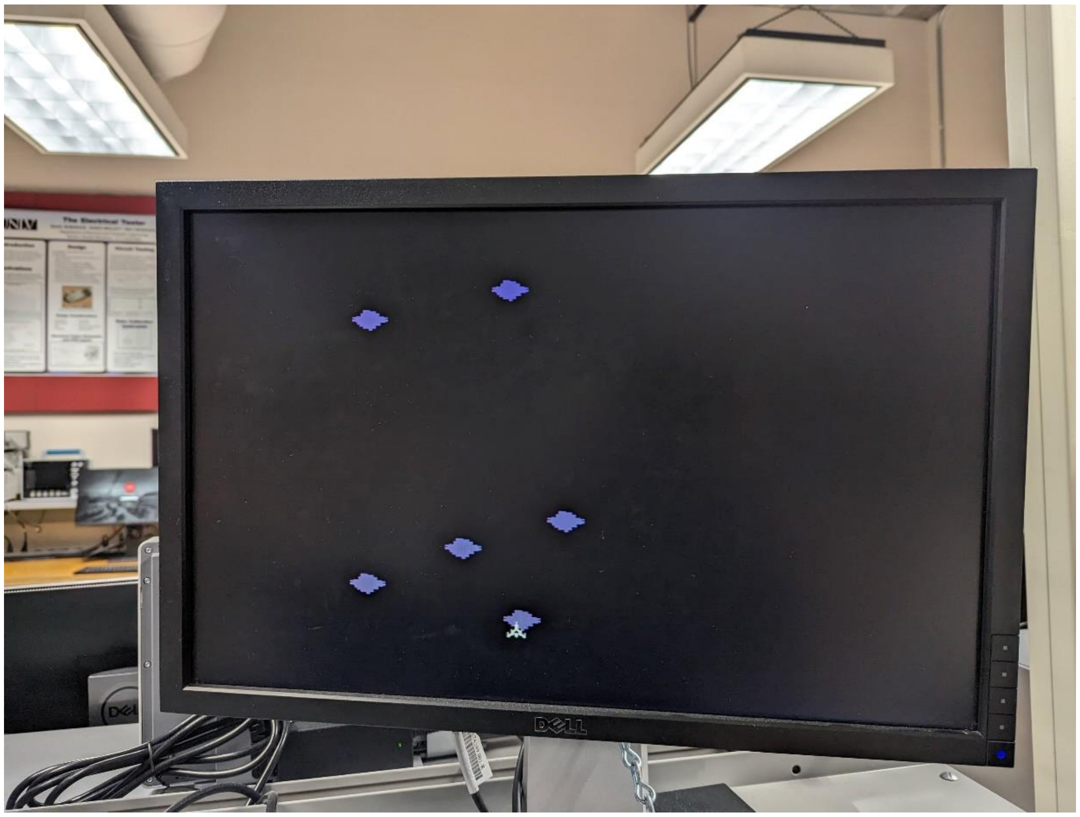


- **ClockScore and Score Module**
    - Generate a slow clock for increasing score.
    - Keeps track of the player's score.
- **SevenSeg**
    - Display the score on the LCD display.

Implementation:

The picture on the left shows how the pushbuttons are connected to the FPGA GPIO pins. These buttons control the movement of the spaceship. The picture on the right shows the the score counter on the LCD display of the FPGA.



This image displays the rocket with the white color and the asteroid with purple color moving down the screen. After collision is detected, gameOver signal is asserted and rocket and asteroid stop moving and score counter is stopped.

Lessons Learned:

I attempted to connect a joystick with buttons to control rocket movements in four directions. However, I soon realized that the buttons weren't directly one-to-one mapped to movement, making the task more complex. Consequently, I opted to create a straightforward circuit on a breadboard, using push buttons connected to GPIO inputs. In my project, clocks played a crucial role in managing spawn intervals, increasing difficulty, and updating the score. I've come to appreciate how essential clocks are in video games and other applications. Properly managing clocks becomes even more critical when different modules operate on their own clocks and need to interact seamlessly. Additionally, I've recognized the significance of random number generation for introducing unpredictability.

Future Steps:

The game presents numerous opportunities for enhancement. Introducing a shooting mechanism to the rocket could offer a more dynamic gameplay experience. This could involve monitoring the shot's trajectory and deactivating any asteroid upon matching its pixel value with that of the projectile. Additionally, incorporating asteroids that descend in varied trajectories, not just vertically, would add complexity and interest. Integrating a joystick for rocket control could also enhance the player's experience, offering a more realistic and engaging interaction.

Conclusions:

In conclusion, my understanding of how modules in SystemVerilog connect to each other and create more complex functionalities has deepened. I've honed my time management skills for complex projects, learning to break down tasks into smaller, more manageable segments. This project has been thoroughly enjoyable, and it has significantly boosted my confidence in programming with SystemVerilog and in my coding abilities overall.