Chrome Dino Game Using an FGPA and VGA

Joey Volcic, Dylan Wismont, Modathir Bougrine, Andrina Toma
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: volcic@oakland.edu, dwismont@oakland.edu, mbougrine@oakland.edu, Andrinatomatoma@oakland.edu

***Abstract*- the aim of this project is to use our knowledge of VHDL and expand on it to create a version of Google's "No Internet" dinosaur game on an Nexys A-7 FPGA board. There is one input that is used to initiate the game and interact with the dinosaur and that is accomplished using a button. The dinosaur; therefore, will either be in motion with the cactus appearing to approach him or he will be immobile due to a collision with the cactus. This paper covers the methodology and challenges faced during implementation. The results of the project demonstrate the successful creation of a functional game that can run on an FPGA development board.**

I. INTRODUCTION

Google's "No Internet" dinosaur game is a simple game that helps you pass the time when you lose internet connection while browsing the web. The player has one task, jump over the incoming obstacles. As the player avoids more obstacles their score increases, however the speed of the player also increases making the game progressively harder. Once the player inevitably fails the score is displayed and they have the option to restart the game.

The goal of our project is to recreate the simple "No Internet" game using an FPGA and a VGA display. FPGAs are a particularly useful tool for prototyping and testing since they let you create complex circuits without the need to wire up individual gates. To program the FPGA, we will be using Vivado's VHDL development environment. To implement this game on our FPGA we will be utilizing the Nexy's A-7 development boards onboard buttons to control the game state and the actions of the dinosaur, we will be using the VGA port to display the game to the user.

II. METHODOLOGY

To design this circuit, we broke the problem down into blocks making the function of the circuit easier to understand. These blocks were then connected to create the desired output.

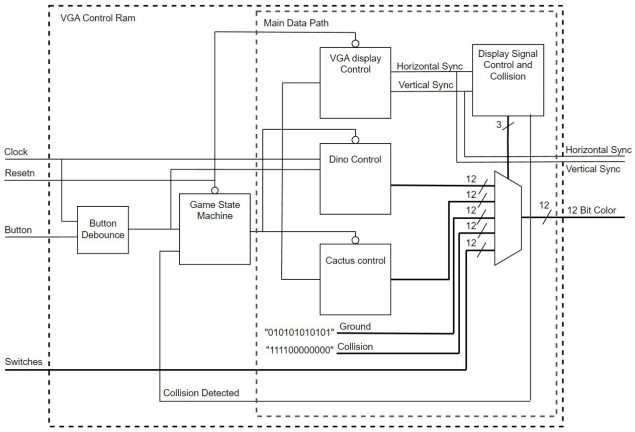

Figure 1. Block Diagram of Main Components

A. *Game State Machine*

The game state machine is used to keep track of the current screen that is being displayed. There are three total screens that we need to replicate to make this game. The first is the starting screen, which is displayed when the player first opens the game. In this state the game should be paused with the cactus on the other side of the screen. The second state is the active game state, this occurs after a button input from the user. When in this state the obstacles start to move and the state ends when the player collides with an obstacle. Which switches the game into the game over state where the dinosaur and cactus are frozen in their current position. This is done by setting the dino and cactus resetn to 0 which stops them from moving. The user can then choose to play again by clicking the button, this brings the game back to the active game state. This cycle will continue as long as the player keeps pressing the button.

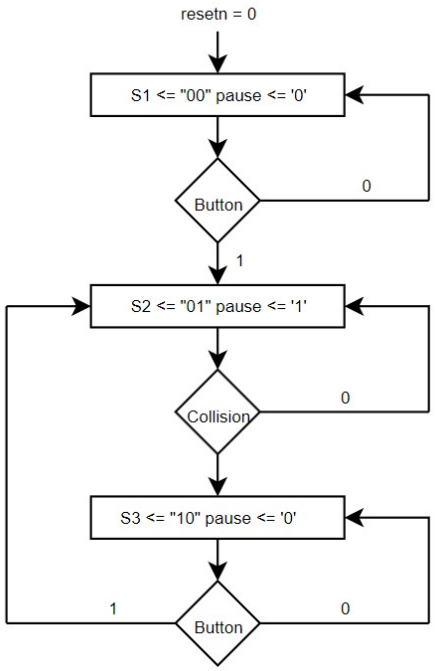

Figure 2. Game Algorithmic State Machine Diagram

B. *VGA Control*

The main output from the circuit is a VGA signal. A VGA signal requires 14 sub-signals to create a picture on a screen. Twelve of these are used to determine the current pixel's color, four bits for red, four for green, and four for blue. The other two signals are horizontal sync (HS) and vertical sync (VS); these are responsible for the horizontal and vertical position of the next pixel displayed on the screen. These sync counters also allow us to retrieve the correct pixel color from the ram [1]. Accurately determining these values is done using a series of counters running off a 25MHz clock. This clock was chosen because we want our refresh rate to be 60 frames per second, with a 640 by 480-pixel display area. Due to the front and back porch restrictions of VGA, which gives tube displays the time for the electron beam to speed up and slow down, to account for this, we need a resolution of 800 by 525. Hence, 25MHz / (800 * 525) is roughly 60 frames per second. When we are not in the 640 by 480-pixel display range we cannot write any pixel colors to the screen.

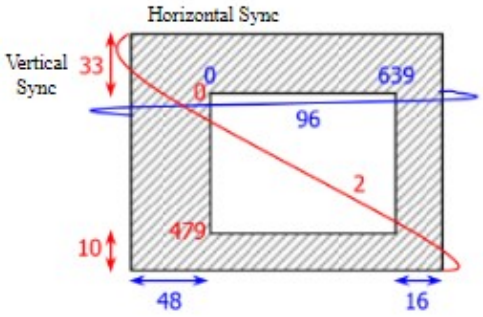

Figure 3. Display Area with porches

C. *Dino Control*

To recreate the game's feel, we needed to make the dinosaur look and behave like the dinosaur from the game. We created the dinosaur's look using ms paint to draw a picture of it, commonly known as a game sprite and then converted it into a text file using Matlab [1]. These text files were loaded onto the ram, letting us retrieve the images later.

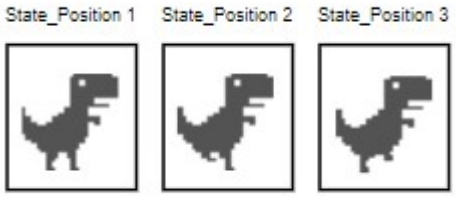

Figure 4. Dinosaur Sprites

Then we created the behaviors of the dinosaur: standing, running, and jumping. Standing was the simplest of the cases, as we just needed to display a sprite of the dinosaur with both feet on the ground and stay in that position. To animate running, we cycled between two sprites of the dinosaur, one with the right leg up and one with the left leg up. Finally, we created the jumping animation. Instead of changing the sprite, a height value was incremented to animate the dinosaur. These animations were controlled using a finite-state machine constrained by two asynchronous inputs that determined what the dinosaur should do next. Controlling these inputs asynchronously reduces the feeling of input lag for the user. If the button is pressed, the jump animation is triggered, and animation changes are stopped if a collision is detected

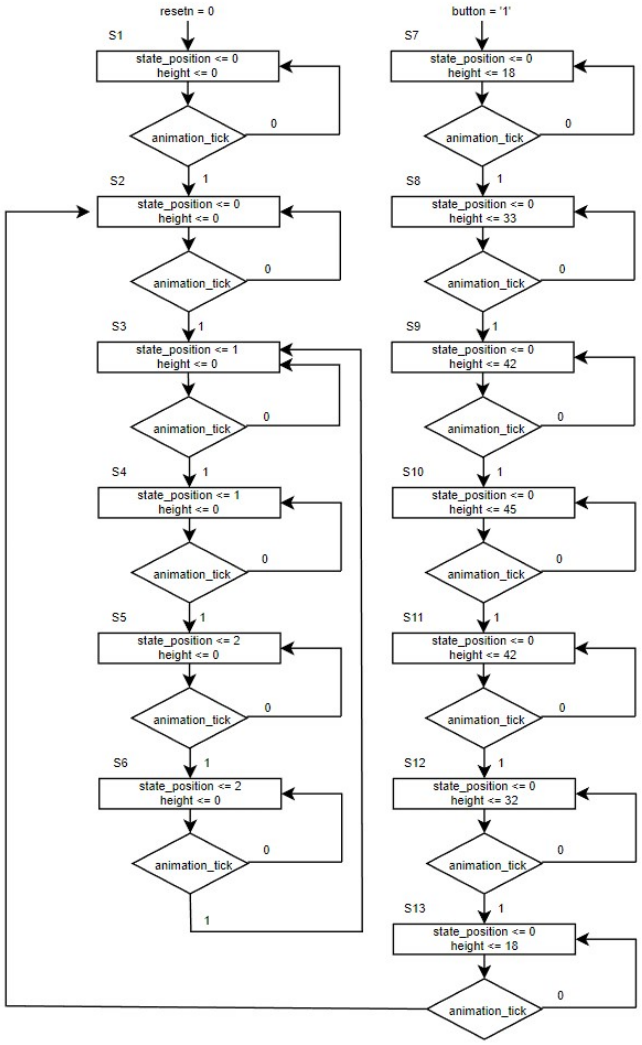

Figure 5. Dinosaur Control Algorithmic State Machine

Based on the outputs from this state machine, we can select the correct sprite to be displayed on the screen using a mux tied to the output of all the ram blocks; since the ram blocks all have the same ram address line, they all output the same pixel position making this technique possible. Finally, the height of the dinosaur is sent out to the display signal control block for further position calculations.

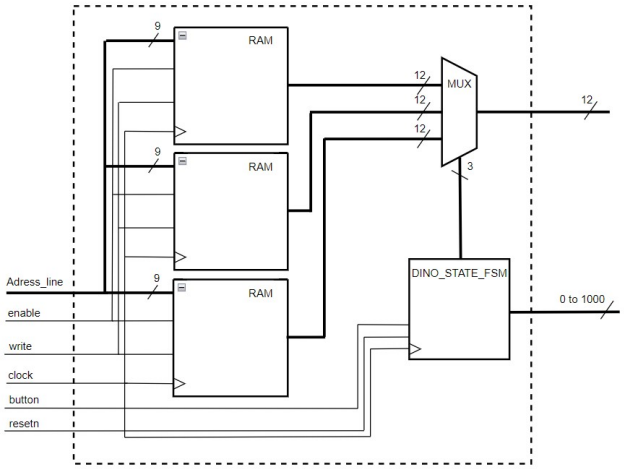

Figure 6. Block Diagram of the Dino Control Component

D. Cactus Control

In this version of the dinosaur game, the cactus control is composed of the cactus state machine and the cactus Ram memory. The cactus state machine is a component of the VHDL code that defines the behavior of the cactus obstacles. The state machine controls the movement of the cactus, transitioning between different states based on certain conditions. For example, at first the cactus is generated in an “offscreen” position or inactive state. Then, once the position tick is initiated, the cactus reaches the edge of the screen, then it transitions into the active state. Lastly, the cactus starts moving towards the dinosaur. Now, if the cactus collides with the dino, the collision detection is initiated and the game ends. Alternatively, if the player is successful in jumping over the obstacle; in our case the cactus, the state machine transitions back to the “offscreen” state and generates a new obstacle. This state machine is designed to challenge the user and requires quick reflexes and precise timing to avoid the obstacles.

Next, the Ram memory is a storage established in the FPGA through VHDL code to store our image of the cactus. First, the image of the cactus is saved in the VHDL code as a series of binary values that are representative of the pixels of the cactus. These values are created using a conversion code on MATLAB which converts the image pixels into a binary format that can be stored easily in a RAM established in our code. By saving the cactus image this way, we are able to easily generate and display the cactus on the screen when needed making the game engaging and enjoyable.

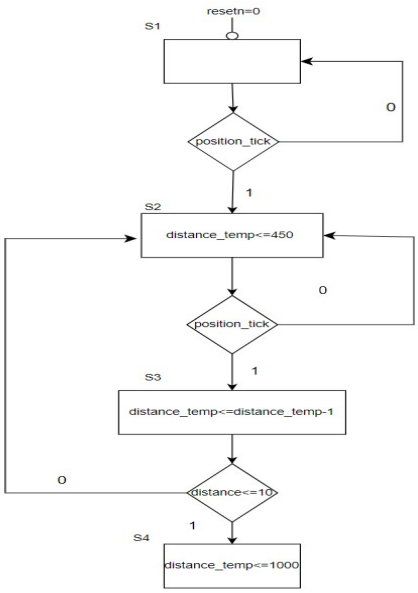

Figure 7. Cactus Algorithmic State Machine

E. *Collision Detection*

The collision detection between the dinosaur and the cactus is a critical component in the implementation of the dinosaur game in VHDL. The collision detection process involves checking whether the dinosaur sprite and the cactus sprite overlap in any way; this later is done by comparing the positions of the two sprites while the game is running.

In our VHDL code, we execute the collision detection using an "AND" logic gate (figure 8), where if parts of the two sprites overlap in any way, then the collision detection is initiated; the game is paused and the area where the collision happened is highlighted on the screen in red as seen in figure 9. The code for the collision detection is incorporated in our VGA control Ram.

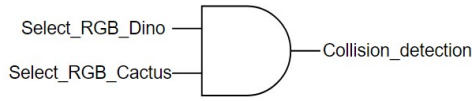

Figure 8. Collision Detection Logic

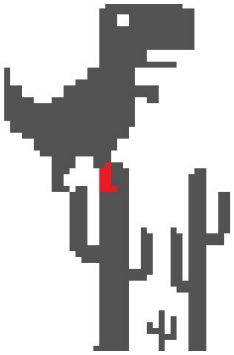

Figure 9. Visible Collision Indicator

F. *Other Components*

To achieve the final goal of this project in a timely manner, parts of our project were sourced from pre-existing code. We used VGA_12_bit [1] and its subcomponents as a starting point to build off.

III. EXPERIMENTAL SETUP

In the process of making this game, it was vital to simulate state machines and their components. However, it was very difficult to devise a way to simulate the entire project using the simulation in VIVADO. So, for that part, we decided to simulate the game using the VGA and solve any issues we encountered.

Simulating the different state machines and controls individually using VIVADO helped us see if our components are able to achieve the necessary functions. While testing these components, we could fix if there were any issues and instantly re-simulate accordingly.

IV. Results

The VHDL implementation of Google's Dino game was successful. We managed to effectively create a functional and functioning game that could be played on an FPGA development board. The game was able to generate cactus obstacles and move the dinosaur character; furthermore, the game was able to detect collision between the cactus and the dinosaur.

Because of the chosen refresh rate of 60 fps, the game was able to run smoothly and consistently; which provided an engaging gaming experience for the users.

However, we faced many challenges in making this game. The first challenge was to optimize the game logic and components to ensure that it could run efficiently on the limited resources of the nexys7 board. Another challenge was controlling the position and the motion of image to be displayed on the screen; therefore, debugging the VHDL code was a time-consuming process and requiring careful testing and verification since we had to synthesize, implement, generate bitstream, and program the board every time we made any change to the VHDL code.

V. Conclusions

In conclusion, the team was able to successfully recreate Google's "No Internet" dinosaur game using VHDL and an FPGA board. The Nexys A-7 development board's onboard buttons were used to control the game state while the VGA port was utilized to display the game to the user. By breaking down the problem into smaller components, the team was able to develop a game state machine to keep track of the current screen, a VGA control system to accurately determine the location of pixels, and multiple finite state machines to control the position and behavior of the obstacles and dinosaur. The final implementation provided users with a challenging gameplay experience that required quick reflexes and precise timing, just like the original game.

Overall, the project showcased the team's ability to work with complex hardware and software systems, as well as their skills in problem-solving and programming. The successful recreation of the game demonstrated their proficiency in VHDL programming and FPGA board development. Additionally, the project highlighted the importance of breaking down complex problems into manageable pieces, which can be solved individually and then integrated into a complete solution. The team was able to apply this methodology effectively in developing the various components of the game. In conclusion, the project was a great success, and the team members gained valuable experience and knowledge through their participation in it.

References

[1] D. Llamocca, "VHDL Coding for FPGAs," 01 04 2023. [Online]. Available: http://www.secs.oaklsnd.edu/~llamocca/VHDLforFPGAs.html..

[2] D. Graham, "T-Rex Runner Game in Scratch," 14 07 2020. [Online]. Available: https://deejaygraham.github.io/2020/07/14/trex-runner-in-scratch/. [Accessed 10 03 2023].