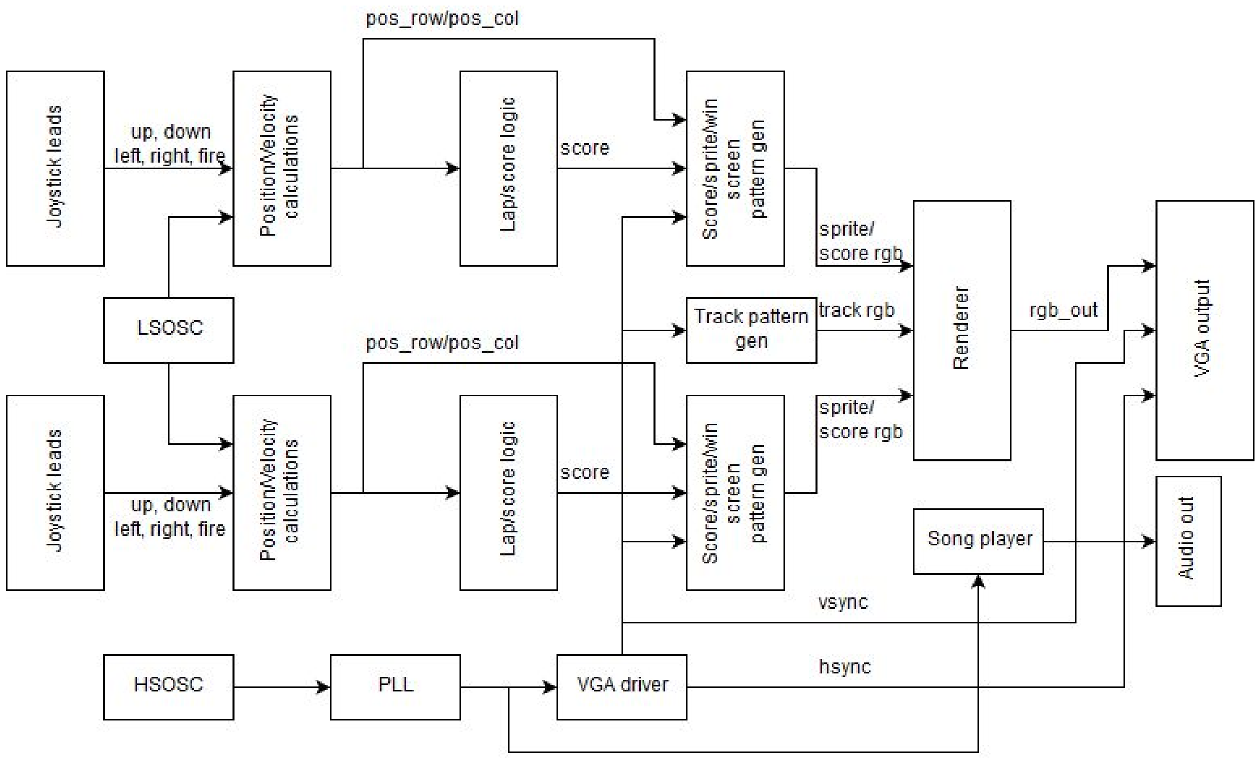
Alejandro Colina-Valeri
Brandon Gray
Joe Reese
Paige Shephard

ES 4 Final Project: Racing Game

PART I: Overview

Our project is a simple racing game where two players control individual sprites as they race each other around a track to see who can be first to five laps. The way it is played is by using a joystick with one button for each of the two players. Holding down the button enables the sprite to move around the screen and the joystick determines which direction the sprite will move. By holding down the button longer, the sprite accelerates while on the black track. When the button is released, the sprite slows down to a complete stop. The grass slows the car down but not to a stop. Every lap, the count at the top corresponding to the player goes up by 1.

The architecture of the game is divided up into the display, the data processing, and the sound. This is most easily seen in the block diagram below.



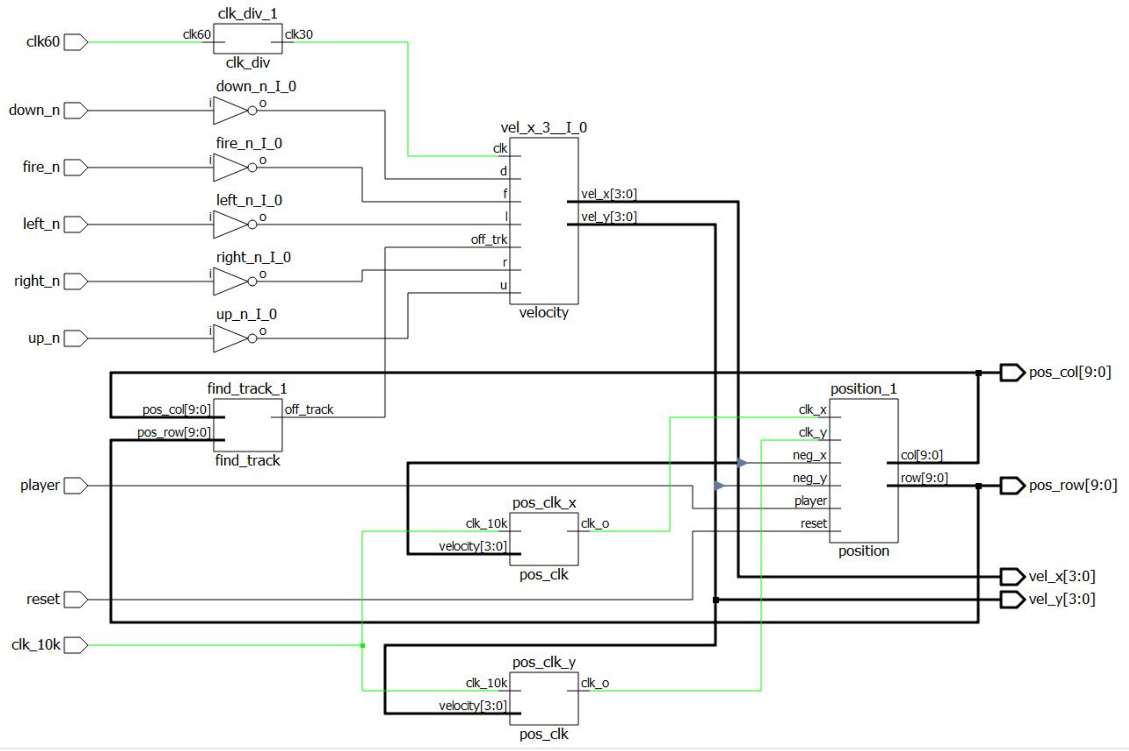
Alejandro Colina-Valeri
Brandon Gray
Joe Reese
Paige Shephard

PART II: Technical Descriptions

Clock signals:

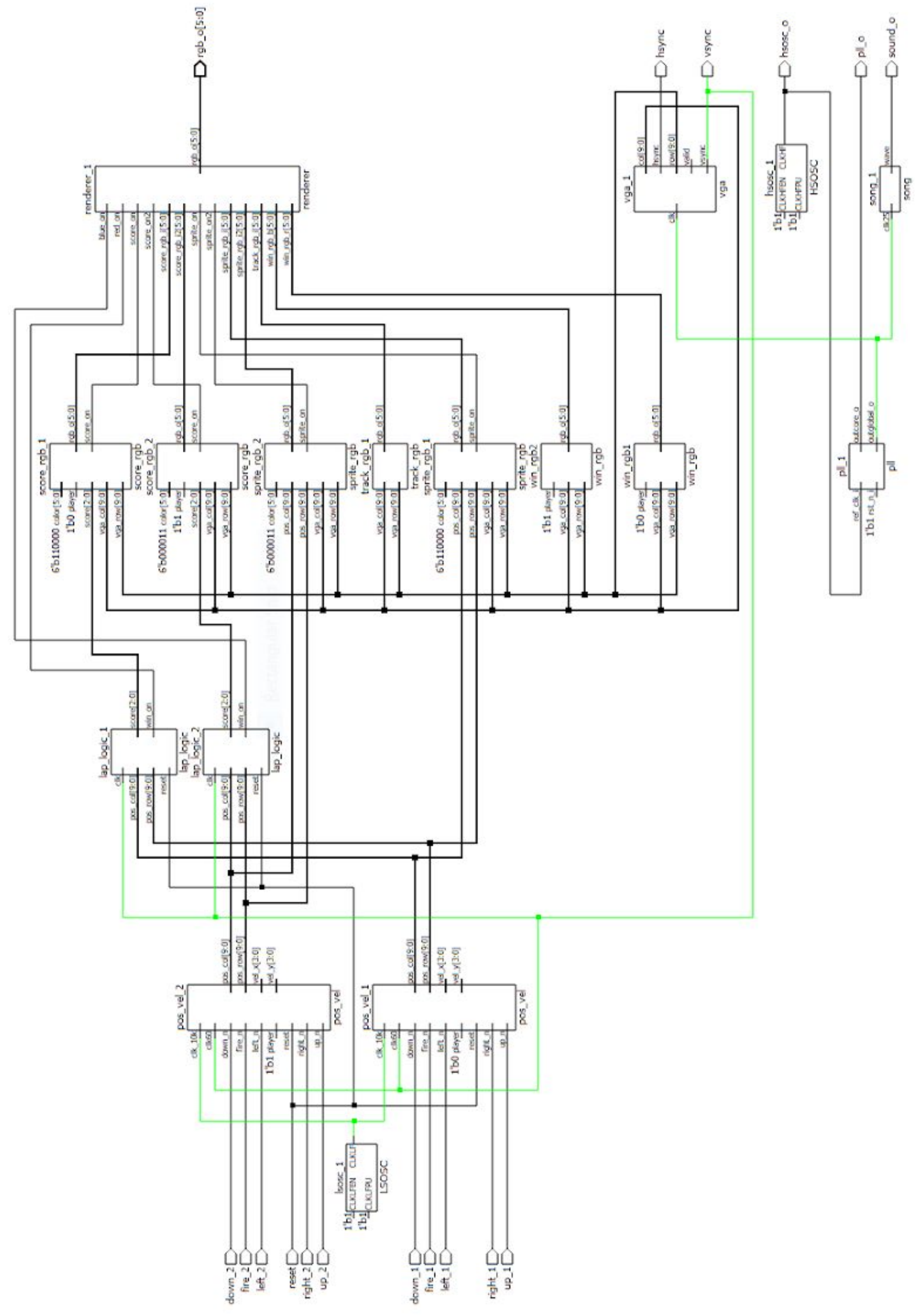
The FPGA's 48 MHz high-speed oscillator (HSOSC) is used to drive the onboard PLL unit, which outputs at 25.125 MHz. The PLL governs the sound modules and the VGA driver (`vga`). The `vga` module generates HSYNC and VSYNC signals required by the display protocol.

The 60-Hz VSYNC pulse drives a counter that outputs a 30 Hz clock to the `velocity` module. (Player velocity is updated once every other frame.) The 10 kHz low-speed oscillator (LSOSC) is routed to the module that determines the frequency at which the player's position is updated (`pos_clk`). LSOSC is used here to reduce the number of bits needed for counters in `pos_clk`.



pos_vel module netlist

Alejandro Colina-Valeri
Brandon Gray
Joe Reese
Paige Shephard



Alejandro Colina-Valeri
Brandon Gray
Joe Reese
Paige Shephard

Above: Netlist block diagrams for top module

Graphics output:

Graphics for the game are handled entirely without memory blocks. All the patterns are generated logically. The VGA driver continuously outputs a row and column value corresponding to the pixel currently being drawn (or the equivalent of when outside of the visible area) based on timing. By comparing this scan position to a set of constraints, images can be built row-by-row in blocks. For example, in VHDL:

```
rgb_o <= "000000" when
      ((vga_row >= 10d"100" and vga_row < 10d"104") and
       (vga_col >= 10d"384" and vga_col < 10d"544")) or
      ((vga_row >= 10d"104" and vga_row < 10d"108") and
       (vga_col >= 10d"372" and vga_col < 10d"584")) or
...
    else "001000" when vga_row < 10d"480" and vga_col < 10d"640"
    else "000000";
```

The above is part of the code used to draw the background. (The same set of constraints are compared against the player's position row and column to determine whether they're on the track.) These constraints would be impractical to code by hand, so a C++ script was written to automate the task. A bitmap image is made using red (hex FF0000) to mark out regions to be rendered by the FPGA. The script parses the image data and scans through the rows, printing out constraints for the start and end columns of each contiguous region of red. The script is set up to be able to "downsample" by skipping some rows and columns, in order to reduce the number of constraints. This is important because the amount of logic required to encode 480 rows of a complex image could consume the majority of logic blocks on the FPGA.

While the scores, track, and win screen text are all drawn in this way, (including separate constraint sets for digits 0 through 5 for each player's score,) the "cars" are just simple squares. The main reason for this is discussed in the Reflection, but in short, it is a corner-cutting measure to keep the code simple. The constraints for any complex graphic would have to be drawn in reference to a moving central row and column, and ideally would rotate depending on velocity. Complexity here was deemed unnecessary and impractical for the core goals of the project.

Alejandro Colina-Valeri
Brandon Gray
Joe Reese
Paige Shephard

The RGB data from `track_rgb`, `score_rgb`, `sprite_rgb`, and `win_rgb` modules are all fed into the `renderer` module, which chooses which of the patterns is "on top" at any given row and column. For example, if all foreground RGB signals are black, the background color for that point is drawn. If both the sprite and score colors are non-black at that point, then only the score color gets shown. This logic prevents the output from being driven with multiple conflicting color signals at once.

Velocity and position:

The velocity module underwent two major changes. Initially, pushing up, down, left, or right would increase the number of pixels per frame the player's position would change by in direction. The main drawback of this method is that if the player, say, presses left while moving up, their new velocity will not left, but up-left, which is counter-intuitive.

The second iteration overhauled this logic. Inputs now correspond to a "peak velocity" in each direction. For instance, pushing left-down sets the peak to (-7,7). Each clock cycle, the player's current velocity is compared to the peak and incremented or decremented in each axis until they match. If the current velocity is (0,7) and the peak is (7,-7), then the intermediate velocities would be (1,6), (2,5), all the way to (7,-5), (7,-6), and finally (7,-7). This provides a very smooth transition between directions without feeling "slippery."

The final problem was that the speeds were too fast overall. Controlling the "cars" on screen was difficult, and the minimum speed of 1 pixel per frame was not slow enough to really penalize going off-track. The solution was to modify the *rate at which position updates* using `pos_clk`, rather than the number of pixels per update.

The player's signed x and y velocities are first converted to an unsigned magnitude. This magnitude selects different reference values that a counter must reach in order to update the player's position in each axis (`clk_x` and `clk_y`). Max speed means small reference value and rapid updating; min speed means big reference value and slow updating. Zero speed is a special case where the position is never updated. The most significant bit of the signed velocity is used to determine if the position should be increased or decreased along the axis. Controlling a clock frequency rather than a number of pixels per frame enables much more granular speed programming, i.e. "how many pixels per *second*" rather than "how many pixels per 0.016 of a second".

Alejandro Colina-Valeri
Brandon Gray
Joe Reese
Paige Shephard

Lap Logic:

To make sure each player had to complete the entire lap to get the full point, the lap logic module was created. This module consists of a state machine that checks if the player as gone over certain parts of the track to allowed it to get a score for completing the lap. The state machine consists of 2 states. The player starts on the left side of the track. At this point the state machine is inside the fist state where it only jumps to the second state once the player reaches the right side of the track, otherwise the player remains inside the first state.

Once the player reaches the right side of the track, then it the player jumps into the second state. In the second state a lot of things happens. If the player goes all the way back to the left side of the track, then it gets a point and jumps back to the first state. If the player has 4 points, then reaches the left side of the track and acquires a 5th point, then the game switches the screen from the track to a winning screen depending on which player won the game. If the player does not reach the lefts side of the track, then it remains in the second state. Even though the module was mainly created to forced the player to go all the way around the track, it also updates the score of each player it activates the winning screen depending on which player won the game.

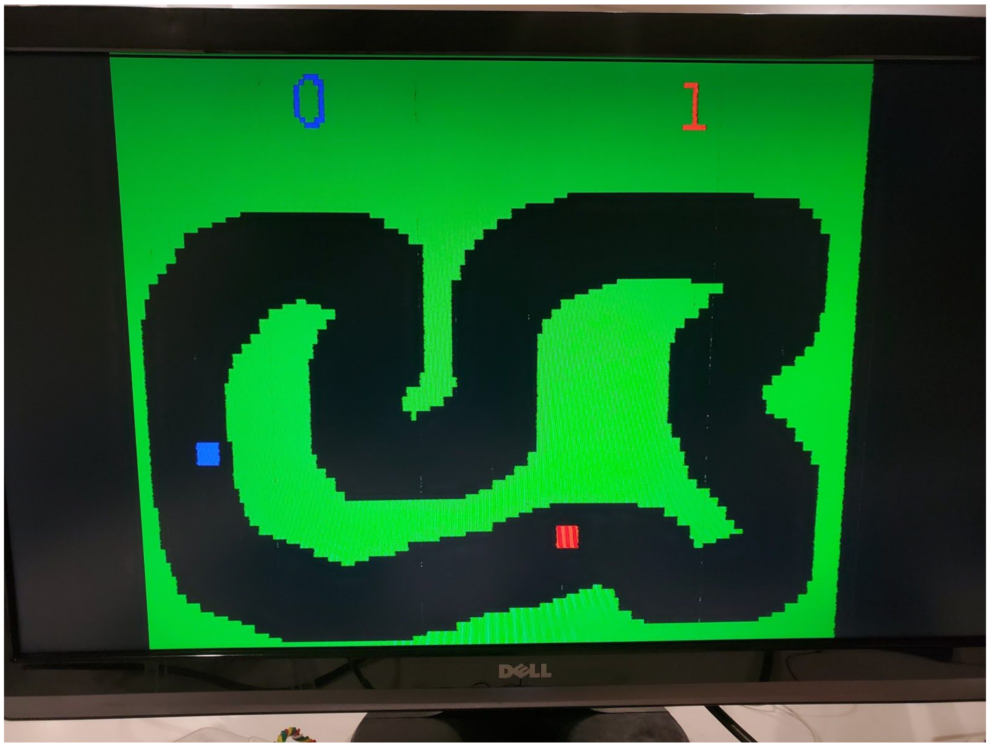
Sound:

The sound was done relatively simply by using a clock divider. The way a speaker works is by pushing and contracting air. These movements change the pressure in the air and create what we know as sound. The pitch of the sound is determined by the frequency at which these pulses occur. This is where the clock divider comes in. By using the high speed clock and stepping it down to the desired frequency, we can create a square wave to be output at different frequencies on a single pin. This meant in theory, we have a monophonic synthesizer (one note at a time is played). This could easily be assigned to the buttons of the controller and different pitches be played with each switch, but we wanted to use the sound in a loop to play a melody similar to many older video games.

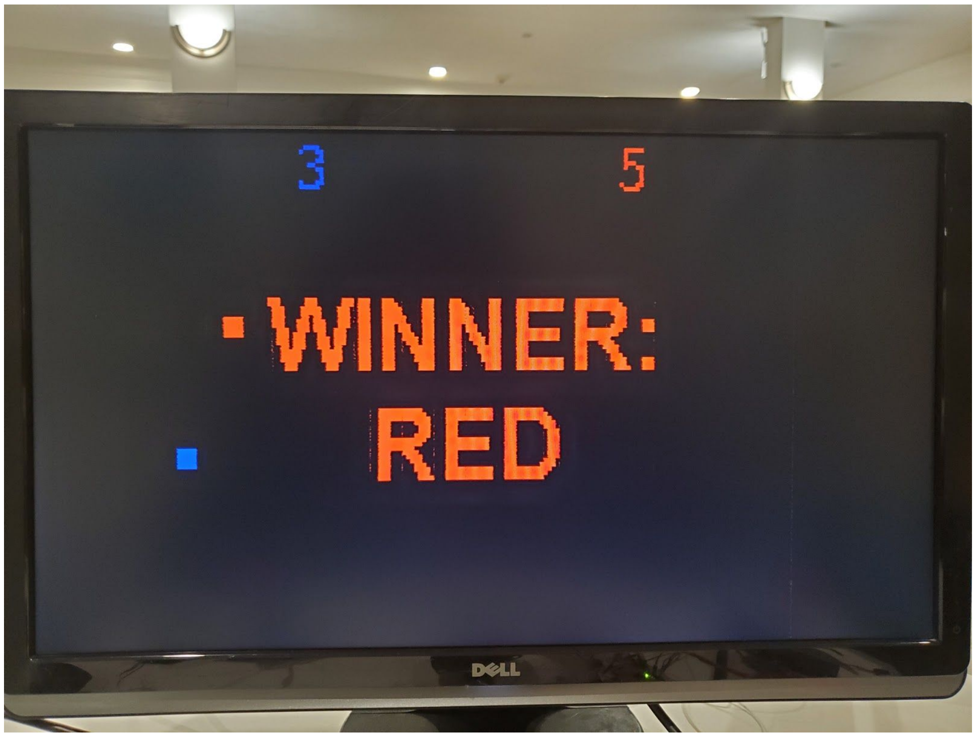
This was done by using a constant logic vector. If every note had a vector where 1 signified a note on and 0 was note off, we can create a melody. We used a counter to divide the clock to a frequency that was about 120 bpm and then had another counter that was the length of the vector of notes. This vector stepped through every element of the vector and when a 1 appeared, that note was output to the speaker. Once the note counter was equal to the length of the vector, it restart to 0 and played again. This is how the loop was created. The square wave was sent to the speaker directly and didn’t require any amplification or a current limiting resistor because the output voltage and current was enough to drive the speaker and the 8Ω of the speaker acted as a resistor. We did include a potentiometer so that the gain of the speaker could be changed if it was too loud or too soft.

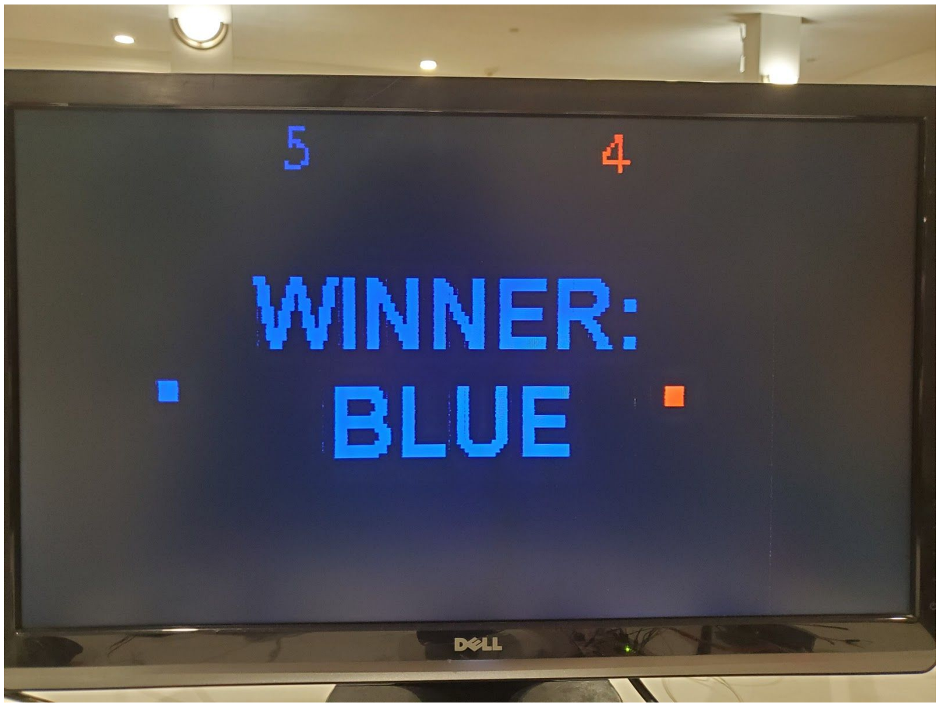
Part III: Results

After using all of these modules, the final product turned out to be a fun and playable game for two people. The sound is monophonic so only one pitch is played at a time. Both sprites are able to move independent of each other and the lap counter works for each one separately. All in all, our project set out what we were going to do and went a bit further.

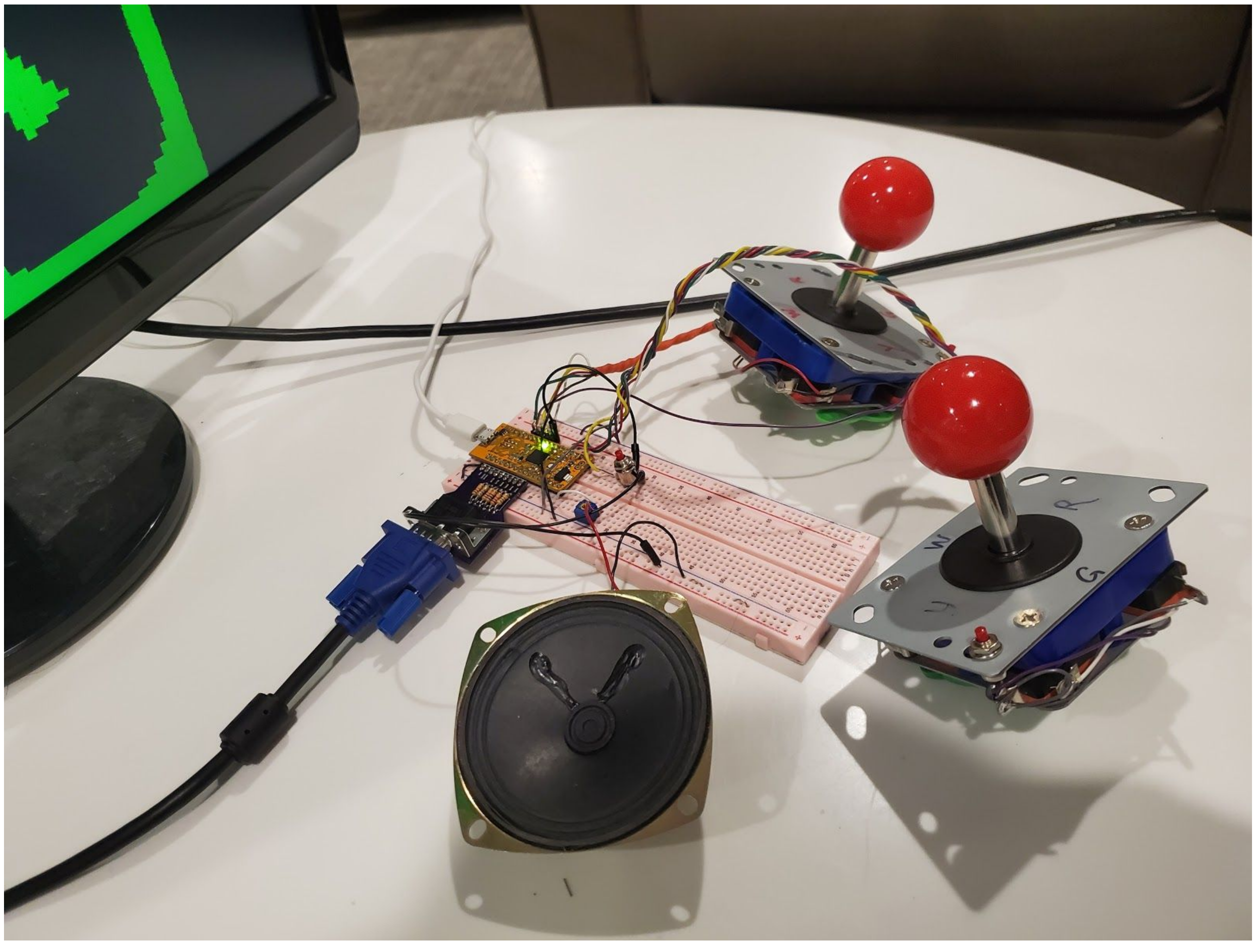


Alejandro Colina-Valeri
Brandon Gray
Joe Reese
Paige Shephard

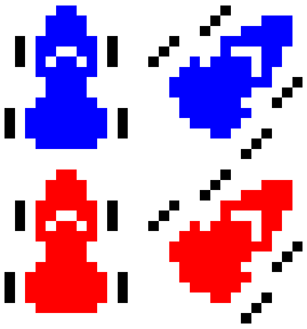




Alejandro Colina-Valeri
Brandon Gray
Joe Reese
Paige Shephard



The only part of the game that wasn't able to be added in this time frame were actual sprite shapes for the two players. Our final product saw them as just squares of color when originally, we want to have actual character sprites for them.



Alejandro Colina-Valeri
Brandon Gray
Joe Reese
Paige Shephard

Part IV: Reflections

The project went well in general. Our team reached the goal of designing a playable, winnable racing game in the style of an Atari classic. Reflections will be given in regards to both technical aspects and team-dynamics.

Technical Reflections:

Our project was completed successfully, despite several hiccups along the way. Even during the demonstration of the incomplete game, several features stuck out as popular among those who played:

- The game was made for 2 people. Single player games can be engaging, but it's much easier to get people excited when there's competition involved.
- The velocity module code smoothly adds velocity components to the car to make it more realistic. Instead of having the car move robotically, the velocity module simulates acceleration to make the car curve more naturally around corners.
- Punishing players who deviate from the track by slowing down the car adds an element of challenge. Just like in a Mario Kart game, the grass won't stop you from moving, but it will slow you down so much it will give the other player the opportunity to get ahead.

In terms of what could have been better, or that didnt good as well as plan are:

- Using ROM to store images. The track background, screen background, and score were possible thanks to the C++ code that outputs the VHDL code to display those patterns in the screen. However, this required a lot of logic gates, and the output from the C++ code was absurdly long. Allocating these images in ROM would likely have made the graphics much more flexible.
- Adding an actual car sprite for the player's car. The original plan was to put a sprite image that could rotate depending on the velocity in each axis. Using our C++ program for this would have been impractical, since we would have to draw an entirely different car for each rotation. Using memory for image manipulation would have been more efficient.
- Music was added to game as a short background track. However, we did not have time to create sound effects to go along with the car's velocity. The original plan was to add a motor sound to represent how fast each car was going. This

Alejandro Colina-Valeri
Brandon Gray
Joe Reese
Paige Shephard

feature was cut in the end due to unexpected struggles getting the music playing properly.

If we were to do this project again we would:

- Keep the velocity, grass and multiplayer. They help to make the game fun and more realistic.
- Figure how to do memory would have made the track. It would make rendering the score and winning background a lot cleaner. Also, it would have made it possible to implement the rotating sprites.
- If we would have started with music earlier, we could have added more music features to the game.

Team Reflection:

Different factors helped into finishing the project:

- We started working on the project early. By the first week, we were almost halfway done; we had a white pixel with moving in all directions, and were able to display the track on the screen.
- We used GitHub to combine our VHDL code together. At the beginning it was difficult to get used to, but then we noticed how easy was to upload our code, then merge it all together in the master branch. By doing this, we had multiple people working on different computers, able to easily combine their code to work for the final version.
- There was flexibility between tasks. Everyone worked on every part of the project. Even though each one was assigned a certain task, we all reached to one another when we needed help to get started, if we got stuck with the logic behind the game, or if the code was not synthesizing.

Even if our team worked well to deliver a functioning project on a deadline, there were some things that got in the way and kept our project from being fully completed for the project presentation:

- Having completed almost half of the project the first week, we took it for granted that adding the rest would not take as much work. Therefore, not much work was done in the second week, and that forced us to do the rest of the work in the last 3 days before the project display.

- Since we made our own team and did not form up based on availability, our schedules impeded us from all meeting at the same time. Therefore, there was a lot of miscommunication within the team. Even though we had a Slack group and GitHub, communication within the team was lacking and therefore there was a lot of confusion on who was doing what, and how tasks were progressing. It was so hard to schedule when to meet, that we even forgot to meet with our TA, and at the end we had only one member to go and talk to them.
- All of us have our own personal lives and clubs to attend that not only affected our schedule, but also the contribution of each team member on the project. Even though all members contributed in the project, some members had to work harder in the last 3 days to get the project done before the project display.

Looking at the pros and cons that went into the group dynamic, there are certain aspects that should be kept and others that should be improved:

- Github is a must. Sharing files and comparing code is super efficient.
- The dynamic of everyone having their own task, but still being willing to help others in other tasks, made it easy for the group to work together and get along.
- A false sense of confidence on finishing the project before the deadline should not happen again. Even if a lot of progress is made in a week, the team should keep working equally as hard for the rest of the weeks. To achieve this, we should have all team members active and working together, so they could all push each other to work as much as they can.
- Communication and scheduling must be improved. There was not a lot of feedback within the group message, and a scheduling a 4-person meeting was really hard to do. For future projects, it would be better if every member were encouraged to communicate with one another *before* choosing team members, to get an idea of group dynamics and availability.