Live-Action RC Mario Kart™

6.111 Final Project Proposal

BRADLEY GROSS[1] | JONATHAN MATTHEWS[2] | NATHANIEL RODMAN[3]

OVERVIEW

Mario Kart 64 is a go-kart style racing video game created by Nintendo in which four players can each control one of eight Mario characters. The characters race in karts around a track, collecting power-ups and maneuvering to win the race. The goal of this project is to bring the classic Nintendo 64 video game to the physical world using a projector and camera, several miniature RC cars, and an Artix-7 FPGA on a Nexys 4 board. The FPGA will render an image of a two-dimensional track, which will be projected onto a flat platform from above as shown in Figure 1. The camera, mounted next to the projector, will determine the positions of the cars using IR LEDs as they race around on the platform. This camera data will be passed into the game logic on the FPGA to determine the state of the game and to control the performance of the cars accordingly. Using Nintendo 64 controllers, players will be able to collect power-ups, monitor their progress on the track, and compete with their friends.

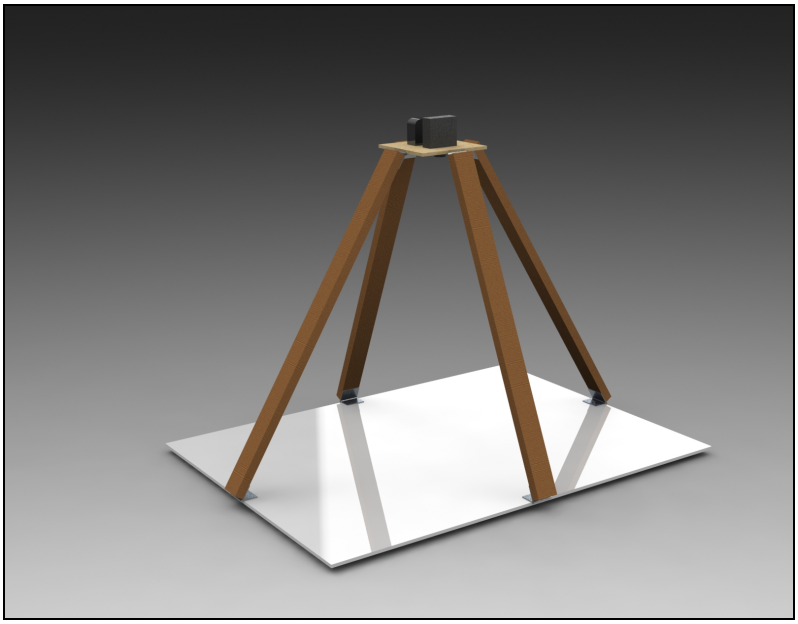

Figure 1: This is a mockup of the physical game components. The camera and projector are mounted directly above the game board with two-by-fours and plywood

[1] blgross@mit.edu
[2] jmatth@mit.edu
[3] naterod@mit.edu

GAME DESIGN

When a player turns on the FPGA a series of screens will be projected onto the game board. As they progress through the screens shown in Figure 2, each player will each be able to select one of eight iconic avatars and as well as a racetrack to drive on. Each of the available avatars has slightly different performance characteristics. For example, Donkey Kong, a large ape, has a slow acceleration but a high top speed. These characteristic will be used as parameters in the game logic to alter how the cars move on the track and interact with power ups. After all of the players have selected their characters and a track, the game advances to the race phase.

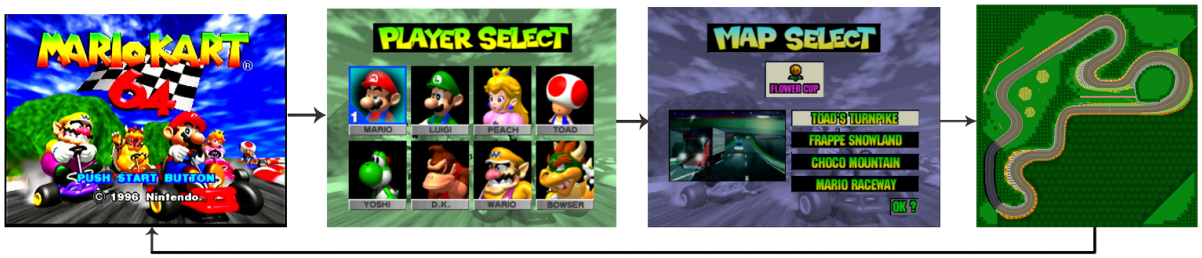

Figure 2: On startup players will navigate through these screens using N64 controllers. The splash screen will transition to the player select screen when a player presses the start button. The player select screen transitions to the map select screen and from there to the actual racetrack as players make selections with the joystick and 'A' button.

The system will ensure that an IR LED corresponding to each car and player is present on the track starting line. If this check fails, an error screen will be presented, asking the players to make sure all of the cars are visible to the overhead camera. The players will not have control over their individual cars until the race begins, as signaled by a countdown sound-effect. Players must then complete three laps of the shown track to finish the race. Lap times will be recorded and displayed after the finish showing the order of the players and their respective lap times.

While racing, the speed of a players car will be determined by the buttons they press, the character they choose, the track segment they are located on, and any power-ups they are using. The camera will determine what type of surface each car is on and limit top speed accordingly. Cars will be able to move fast on pavement, but their top will get progressively lower as the player drives further and further into the grass area surrounding the track.

Players will be able to pick-up powerups by driving over item boxes. When a character drives over the box he is pseudorandomly given an item. When a player uses the *boost* power

up, their car's speed will be increased for a limited amount of time. When a player uses a *banana* power-up, a banana is placed on the track which will slow players and invert their turning controls for a short time if they drive over it. There are additional power ups planned for the game, which appeared in the original version, such as lightning, shells and bombs which are all variations seen in these two core types. Music and sound effects will play throughout the entire game with different music for each screen and different sounds for dropping a banana or boosting a car.

SYSTEM DESIGN AND MODULE IMPLEMENTATION

The system will consist of two overall module types as shown in Figure 3. Peripheral modules interface with analog components and memory. Core logic modules keep track of and update game state, compute car physics and item collisions, and prepare composite sprite images and audio layers.

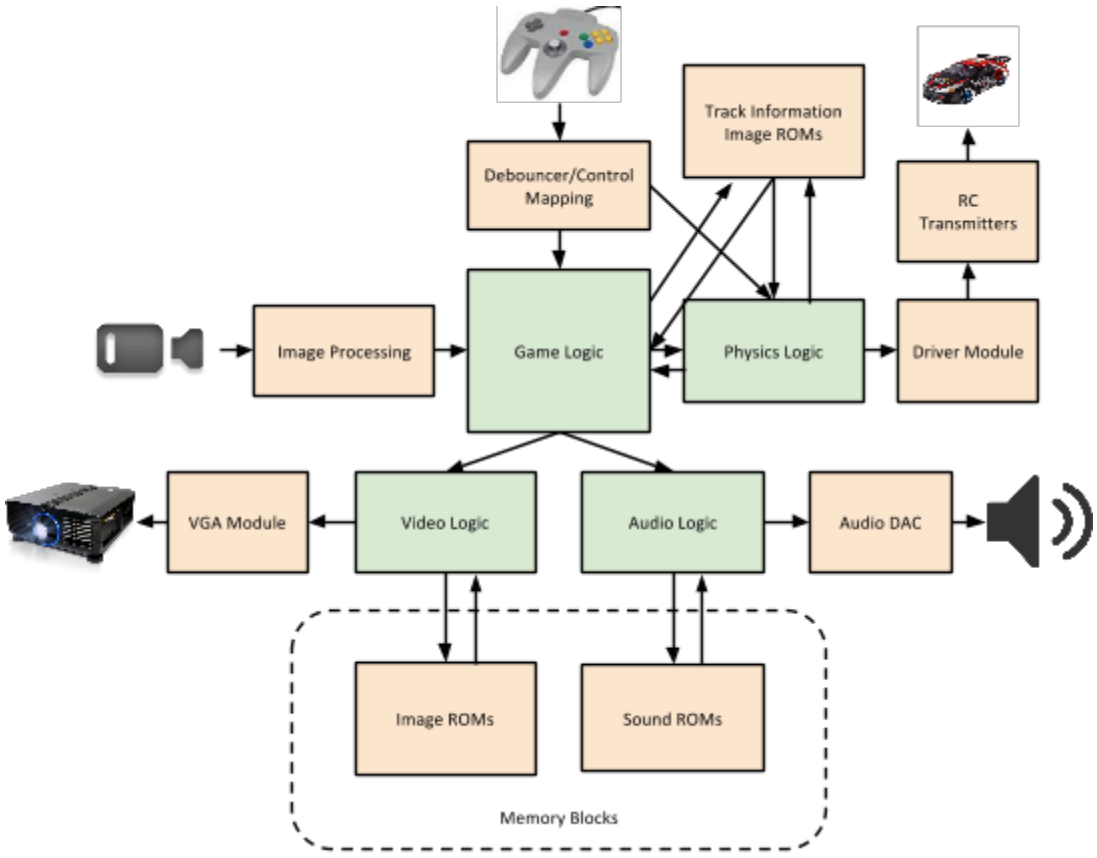

Figure 3: This is a block diagram of the essential system modules. The logic module are displayed in green, and peripheral modules are shown in brown.

Core Logic Modules

Game Logic

The game logic fundamentally serves the game's state machine. It switches between menu screens and race tracks and updates the position of cars, the items on the track and in player inventories, and lap times. This localizes all game state to one module and provides a central location for other modules to look to for game state.

Physics Logic

The physics logic module performs bounding-box collision detections on the various elements in the game world, and determines how cars should move based on control commands, active items, and their car location in the game world. It is a largely combinational module, taking input from the current game state and the controllers and outputting the next game state (did a player drive over an item?) and the appropriate drive commands (if a car is on the grass, it should drive slower).

Video Logic

The video logic module interfaces with VGA output, displaying images based on the game state. For example, the game logic contains a list of all item positions — so the video module will draw an item at each position, overlaid on the track image. Because the image assets of the game are cumulatively far greater in size than can fit in on-board BRAM (the Artix-7 FPGA has ~4.8Mbits), the video logic will retrieve the images from an abstracted "Image ROM" module (see below), that handles caching the relevant images in memory.

Audio Logic

The audio logic module is responsible for preparing composite audio samples from several different sound samples (e.g playing music while an item sound effect is played), and interfaces the composite digital signal to the audio DAC module for output. As well, because all of the audio data cannot fit into BRAM, the audio logic module will retrieve samples from an abstracted "Audio ROM" module (see below), that provides the samples from a larger external memory.

Peripheral Modules

Debouncer/Control Mapping

The Debouncer and Control Mapping module will take input from an attached N64 controller and translate the provided pulses into button presses as well as joystick input. The module will use connectors from an old N64 console to physically connect the controller and FPGA.

Image Processing

The Image processing module will use the labkit camera to provide 2D coordinates of the IR led's on top of the cars to the logic modules. The module will make use of a calibration step where the cars will be placed in predetermined quadrants to pair the cars with their associated controller. The module will then use the previous 2D position to help compute the next 2D position of the car by having a small region of interest to search in each iteration.

Memories

To store and access the many different images, sprites and sounds that the Mario Kart system uses, a memory module abstracts the storage locations of theses assets from the logic blocks. All of the data needed will be flashed to an SD card and connected to the board. When an image is expected to be displayed in the current game phase (triggered by the game logic state transition), it will be copied from SD to BRAM. On game load, audio data will be copied to external Nexys 4 CellularRAM memory, which provides greater storage space than BRAM and random access (unlike the SD serial access).

The VGA Module

The VGA module will handle adhering to the timing specifications of a VGA display and outputting the pixels prepared by the video logic module.

Audio DAC

The Audio DAC module will convert a provided signal of audio samples from the audio logic module into a PWM stream that is sent to the Nexys 4 on-board reconstruction filter.

Driver Module

The driver module will take in commands from the physics logic module and determine the appropriate signals to send to the cars transmitters. The transmitters will be modified with transistors in place of the manufactured buttons, allowing the driver module to send a PWM input to the transmitters. This will afford a range of analog values for car turning and speed.

PROJECT TIMELINE

Week	Logic	Physical / Peripheral	Writing
10/27 - 11/2	None	VGA output on the nexys 4 board proof of concept. Memory access proof of concept. Sound output proof of concept.	Proposal draft. SolidWorks design. Block diagram draft.
11/3 - 11/9	Game logic and video logic proof of concept.	Image Processing proof of concept. Car driving proof of concept. Controller Module. Memory Module. VGA Module. Car IR LED assembly. Rig assembly.	Proposal Presentation. Refine Block Diagram.
11/10 - 11/16	Audio logic proof of concept. Game logic and video logic modules. Virtual track and cars.	Image Processing. Driver Module. Sound Module.	Project Proposal Revision.
11/17 - 11/23	Audio logic module. Physics logic proof of concept.	Integration of Image Processing with Logic. Image Processing Calibration. Car driving Calibration.	None
11/24 - 11/30	Physics logic module and logic calibration.	None	Final Report Draft.
12/1 - 12/7	Fixing Bugs	Fixing Bugs	Final Report Revision.

TESTING

To test we are going to create several subprojects to test the individual modules. This will be straightforward with the peripheral ones. To test the Debouncer/Control Mapping and Driver Modules we will have a test project which has an N64 controller move and operate one of the remote controlled cars. For the Image Processing module we will have a project which detects a set number of IR LEDs and displays their locations with rectangles on a monitor. Lastly to test the Memory access and Audio DAC Modules we will have a project which displays one of two images and plays one of two sounds based on the position of switches.

For testing the Game and Physics core modules we will create a series of testbenches to propagate the different states of the logic separate from the other modules. The Video and Audio modules will be tested manually similar to the peripheral modules but as a part of the main project.