MARIO

BOWSER

BO KIZILDAG

:: bk2838

BRANDON KHADAN

:: bk2746

NICHOLAS VIDAL de la CRUZ

:: nvd2109

EMBEDDED SYSTEMS

PROF. EDWARDS

SPRING 2024

Project: MARIO, FINAL LEVEL ON FPGA

DESIGN

Overview

The aim of the project is to make and implement a final level of the original Mario on the FPGA module. This will consist of the level preceding the boss fight and will transition to the boss fight at the successful run of the level. Our intention is to have fun building a Mario game, but also do something novel as others who built Mario on FPGAs have always done the first (iconic) level. The **Bowser** level and the boss fight will be a novel take on a Mario project with the aim of presenting an even more ***iconic Mario experience*** against the series' ***defining villain***.

Outline

- 2D Side-scrolling
  - Mario will stay at the center of the screen during movement

- Hit Box Detection
  - Mario will be blocked by the ground and walls. Enemies and coins will execute their interaction when hitting Mario

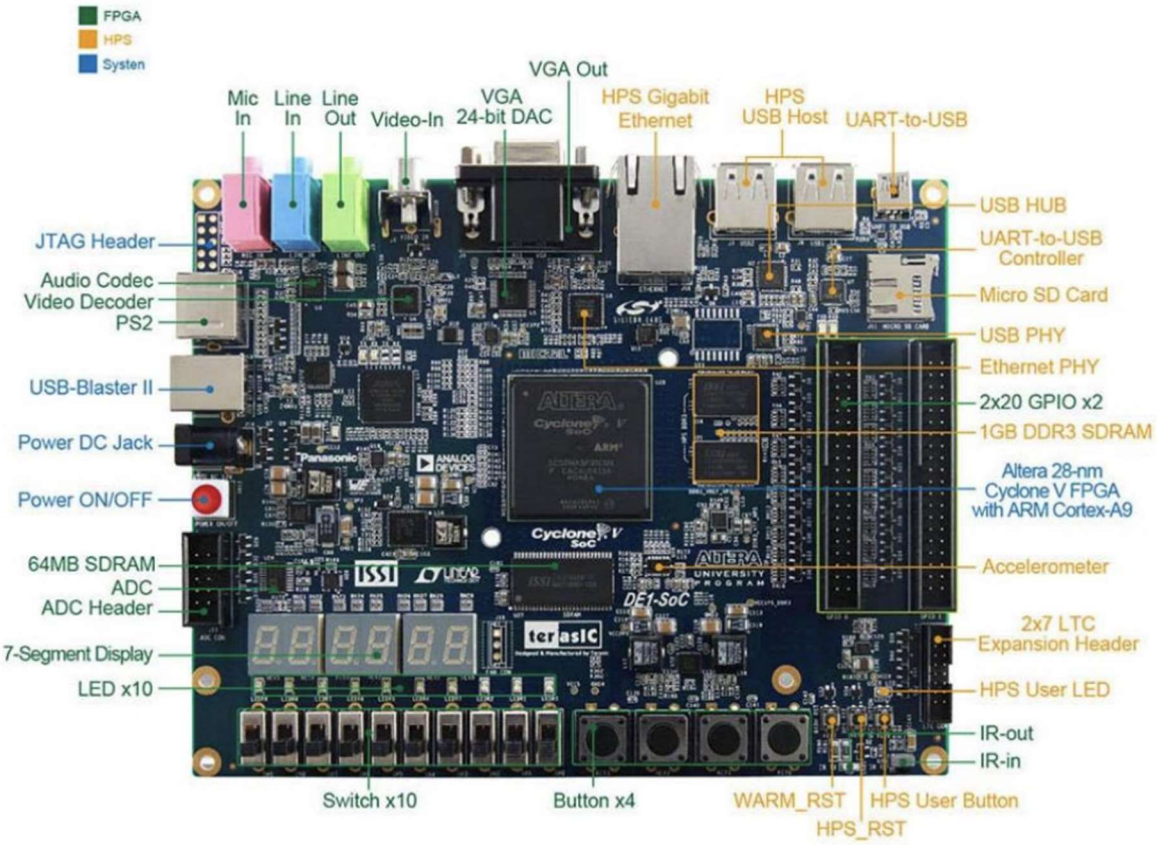
- Audio Output
  - The game will have BGM and audio effects (jump, enemy killed, coins)

- Animation
  - Score animation, death animation, kill animation, movement animation, and impact animation (each will be done by its respective image)

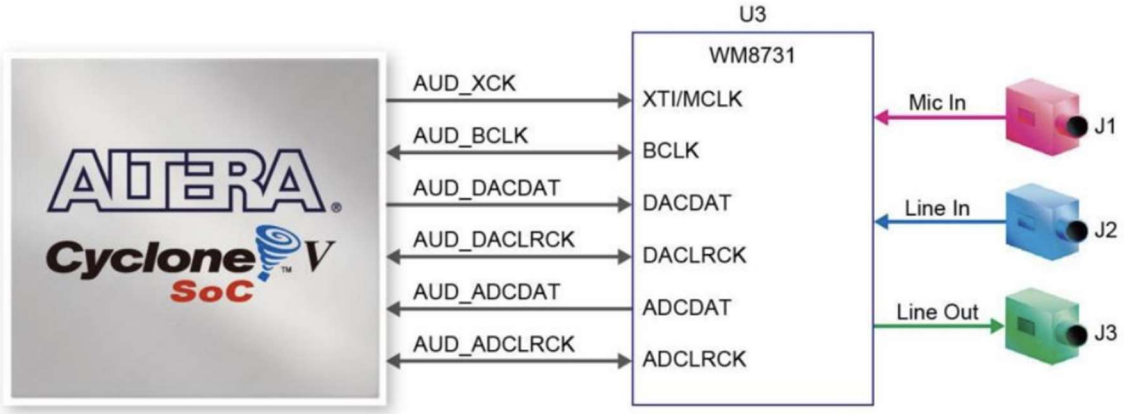
I / O Device

- Video Output: VGA
- Audio Output: Audio jack
- Controller Input: Keyboard

Platform

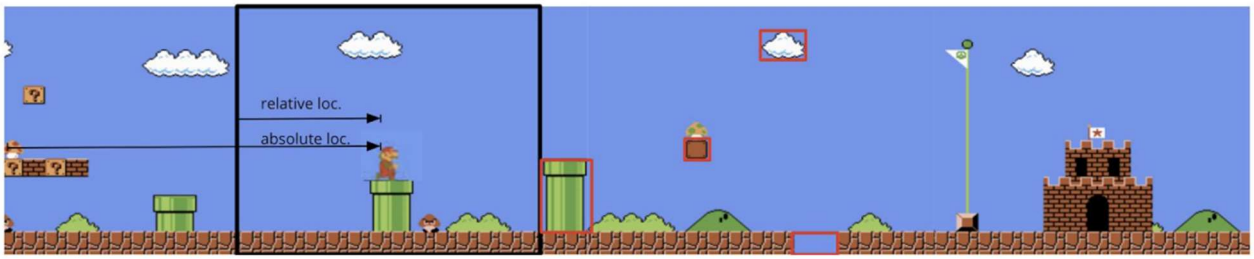


FPGA (from Intel)

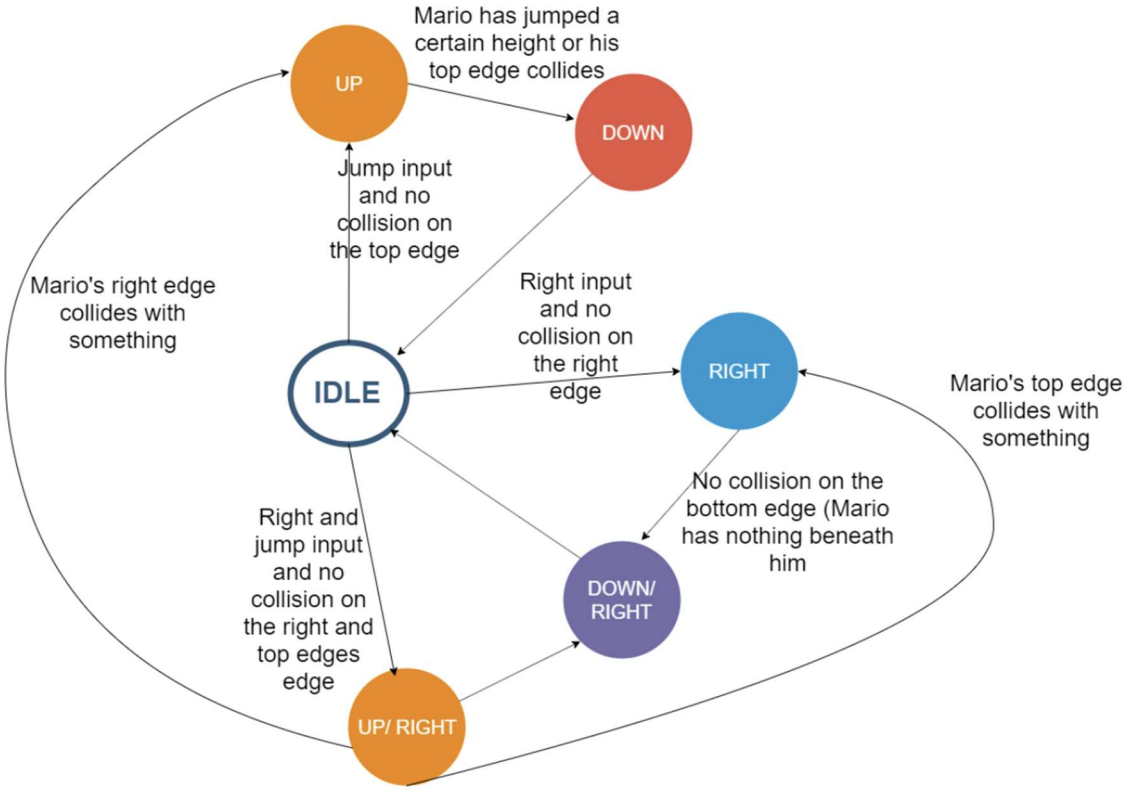


Audio setup (from Intel)

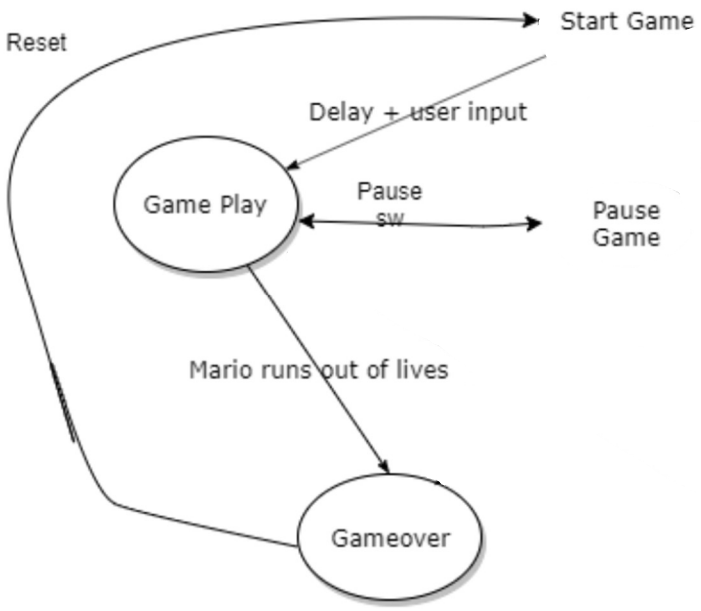
LOGIC



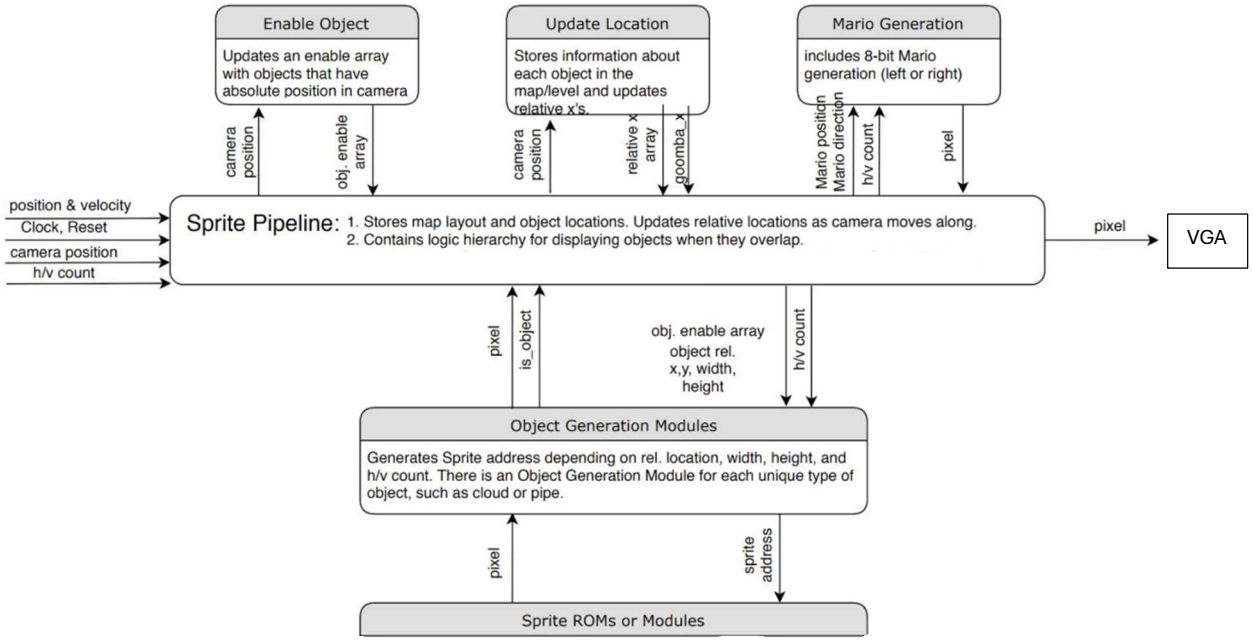
Early diagram of an example level layout (fig. 1)



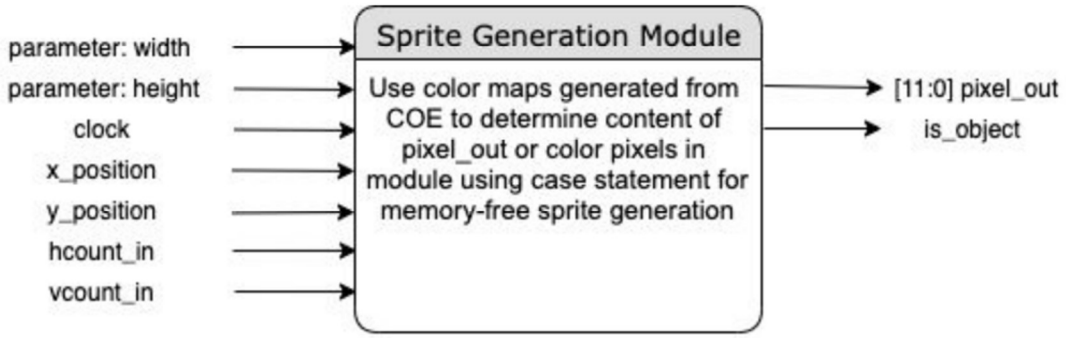
Movement logic diagram (fig. 2)



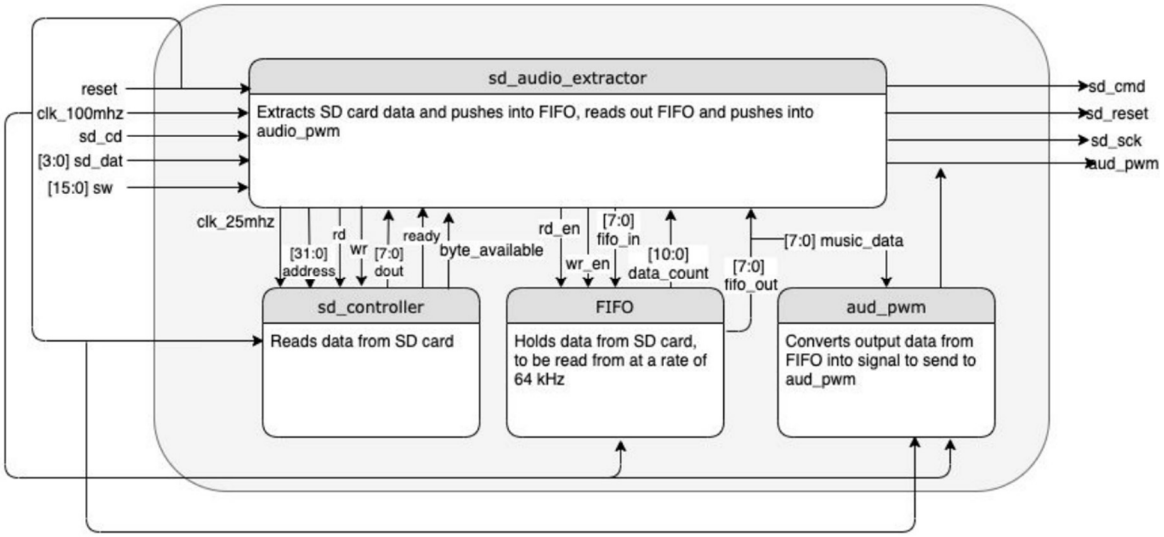
Gameplay logic diagram (fig. 3)



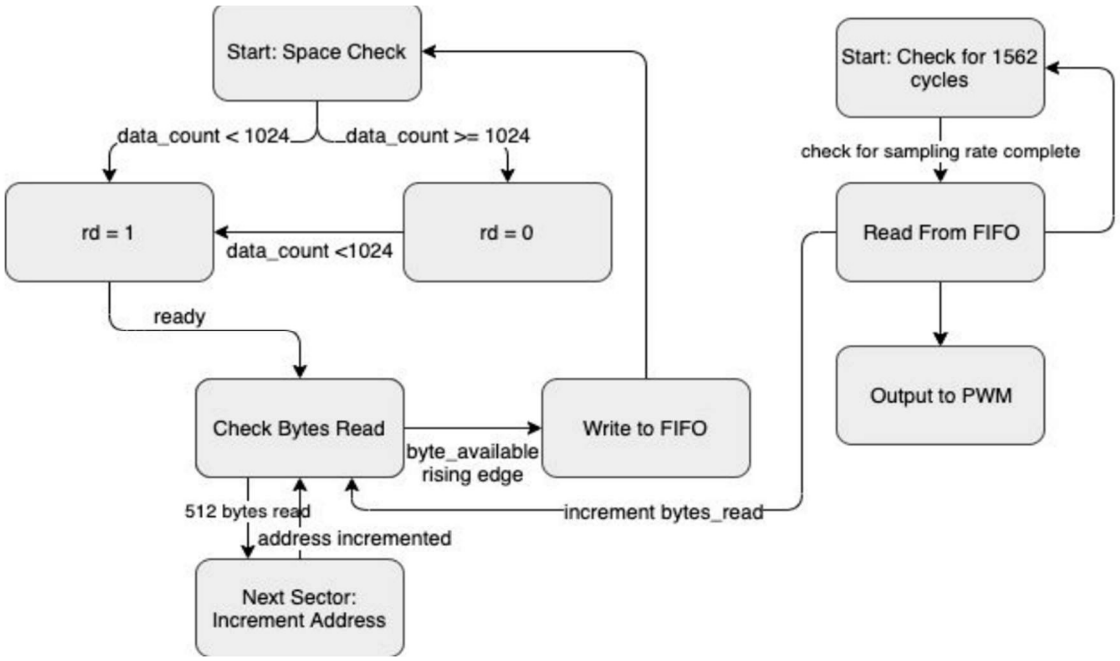
Visual logic diagram (fig. 4)



Sprite generation diagram (fig. 5)



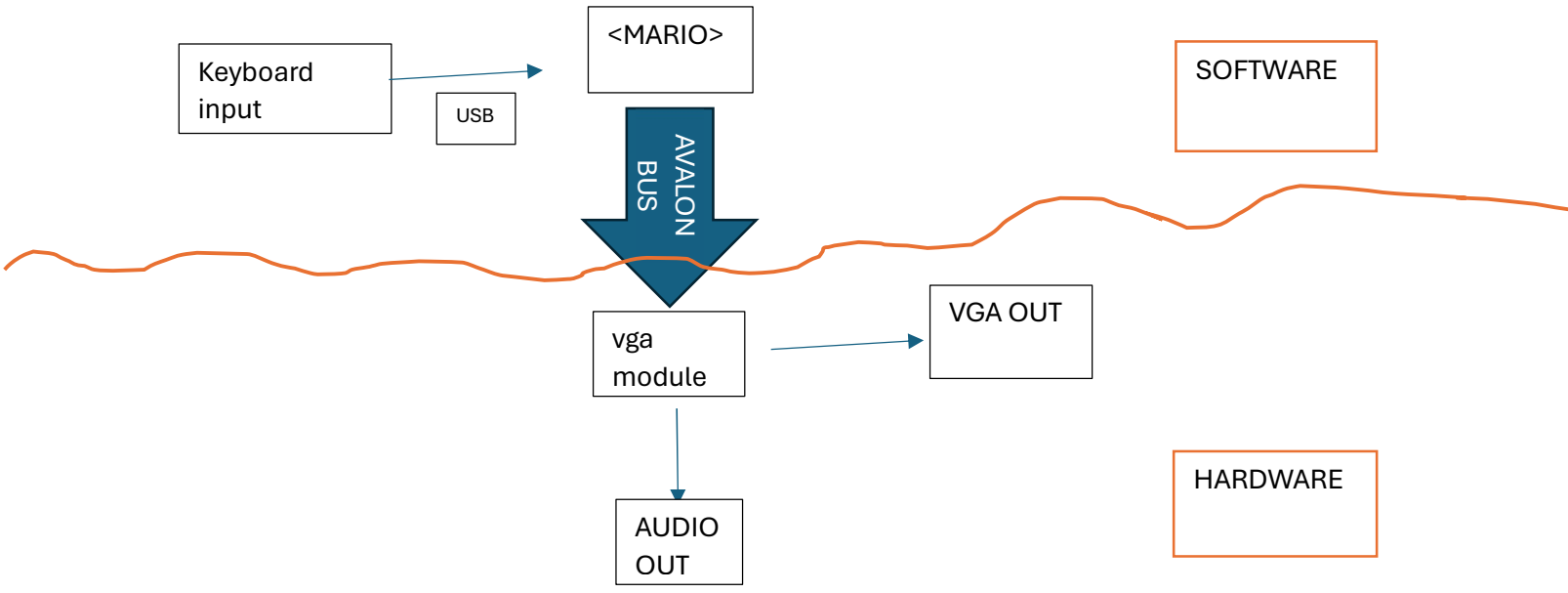
Audio logic diagram (fig. 6)



SD audio module block diagram (fig. 7)

⇨ To read data, set the **rd** wire high when the **sd_controller** signals readiness. SD cards are segmented into 512-byte sectors; hence, to access subsequent sectors, increment the address by 512 after reading 512 bytes. To write new bytes from the **sd_controller** to the FIFO, toggle **wr_en** to 1 on the **byte_available** signal's rising edge and to 0 otherwise. Given the FIFO's limited capacity, prevent overflow—potentially losing audio data—by halting reads (setting the **rd** signal to 0) when its **data_count** exceeds 1024. For audio output, data is read from the FIFO at 64 kHz, necessitating a counter to trigger **rd_en** every 1562 cycles (about 10,000,000/64,000) for efficient data transfer to the **aud_pwm** module.

MAKING THE SOFTWARE AND HARDWARE TALK



BUDGET

Component	Memory (bits)	Memory (KB)
Tile Map	12.000	1.46
Tileset Graphics	262.144	32.00
Sprite Graphics	10.240	1.25
Game logic	262.144	32.00
Audio	524.288	64.00

Total Memory Budget: 1.070.816 bits (approx. 1.02 MB)

MILESTONES

1. Finish keyboard inputs and background visual including side scrolling

2. Complete sprite interactions/effects (sound, animations, etc.)

3. Complete the sprite AI and level design and test/debug for final product

(The exact milestone to-dos for the progression tracking will be discussed with the TA in our upcoming meeting)

Progression

Split off relevant tasks

At each stage assign relevant tasks and responsibilities to team members for implementation or alternatively assign larger stages for individual implementation.

Hardware Configuration

Completely encode keyboard input into distinguishable and identifiable inputs, implement and test and audio/VGA output from the DE1-SOC FPGA.

Sprite Movements/Effects

Attach the simple movements for the main sprite from the UI keyboard. Design and implement animations for the sprites based on keyboard inputs and environmental interactions. Complete hit registration for sprite

interactions with environment/other sprites, implement relevant sound effects and audio as well as animations for distinct types of interaction.

AI Development

Implement AI for non-user controlled elements in movement. First address AI that deals with static movement/actions irrespective of user/sprite movement, then work with AI that responds to current user/sprite location and movement in real time.

Level Design

Implement full level design. Assign sprites to specific locales, add audio cues based on location in level, and add final boss stage that renders rest of level inaccessible and has special functions.

Testing and Debugging

Test individual features and movement, first in expected base cases and then edge cases for input/output, movement, interaction, and AI. Subsequently test full run through of level, attempting to source errors within the design/stages.

NOTE

With the design process ongoing, any part is subject to change in accordance with the main goal of creating Mario final level + boss fight on the FPGA.

Thank you for reading our design document, we are looking forward to amazing you with our game!