SUPER MARIO

Bo Kizildag. Brandon Khadan. Nico De la Cruz
idea

Iconic Mario boss fight against Bowser

=>

Everyone else does the first level, for novelty’s sake we just had to go with the final level

=>
ACTUAL GAME SCREENSHOT OF BOWSER
Every particular build choice was made with memory safety as a priority. Each design module, from the sprites to the audio is optimized to consume as little resources as possible.
controls

Keyboard scheme

A - left

D - right

SPACE - jump
peeking under the hood
DE1-SoC board
connections
hardware

- Sprites
  Each sprite has its own memory file and module to handle access

- Pixel processing unit
camera & hitbox
The octave is divided into 12 logarithmically equal steps, each step being a semitone. This division means that the frequency ratio between any two adjacent notes (like C and C#, or E and F) is the twelfth root of two \((\frac{21}{12})\), approximately 1.05946. This system allows for consistent intervals across keys, which is essential for the flexibility in modulation and transposition in modern music composition and performance.

**Calculating frequencies:**

To calculate the frequencies of the other notes from the reference pitch A4 = 440 Hz, gotta use the formula:

\[
\text{Frequency of Note} = 440 \times 2^{\left(\frac{n}{12}\right)}
\]

where \(n\) is the number of semitones away from A4. If \(n\) is positive, the note is higher than A4; if \(n\) is negative, the note is lower.

Example:

- **C4 (Middle C)** is 9 semitones below A4. Hence its frequency is:
  \[
  440 \times 2^{-\left(\frac{9}{12}\right)} \approx 261.63 \text{ Hz}
  \]

- **D4** is 7 semitones below A4, so: 440 \times 2^{-\left(\frac{7}{12}\right)} \approx 293.66 \text{ Hz}

- **E4** is 5 semitones below A4, so: 440 \times 2^{-\left(\frac{5}{12}\right)} \approx 329.63 \text{ Hz}

- **F4** is 4 semitones below A4, so: 440 \times 2^{-\left(\frac{4}{12}\right)} \approx 349.23 \text{ Hz}

- **G4** is 2 semitones below A4, so: 440 \times 2^{-\left(\frac{2}{12}\right)} \approx 392.00 \text{ Hz}
audio

NoteGenerator
Responsible for generating individual musical tones

I2S_Controller
1. Manages the I2S protocol to transmit audio data to the WM8731 codec

AudioGenerator
Orchestrates the overall audio generation, managing the sequence of notes (it can also control when they play)
Tones

Tone generation via a counter to create a square wave at a specific frequency determined by the half_period input.

Utilizes an internal counter that increments on every clock cycle when note_enable is high.

When the counter reaches the half_period value, the output (note_out) toggles, creating a square wave.

The frequency of the square wave is determined by how quickly the counter reaches the half_period value, setting the tone's pitch.
Wolfson DAC

wm8731
software utilization

Platform and in-game struct generation
DAC configuration

(Thank you Dennis Ritchie!)

Everything else was built in SystemVerilog!
gameplay