Project Proposal: Pop'n FPGA Rhythm Game

Team Members:

- Vince-Arvin Magno (vm2787)

1. Introduction & Core Concept

The core of this project is a hardware/software co-designed rhythm game inspired by the arcade game Pop'n Music. The system will parse a beatmap, play synchronized audio, and challenge the user to press buttons on a custom 9-button USB controller, powered by an Arduino Leonardo, as falling notes cross a judgment line on the screen.

The project demonstrates real-time I/O handling, hardware-accelerated sprite rendering, and precise timing synchronization between a Linux software environment and FPGA hardware. The software, running on the ARM HPS, will handle the complex game logic, audio streaming, and USB input, while the hardware accelerator on the FPGA will be responsible for drawing dozens of falling note sprites without the latency or flickering associated with pure software rendering.

Gameplay of Pop’n Music

2. System Specifications

- **Framerate and Resolution:** The display will target a flicker-free 60 frames per second at a standard VGA resolution of 640x480.
- **Graphics Generation:** To conserve the DE1-SoC's limited on-chip memory, the background will be statically generated or tiled by the hardware. The falling notes, or sprites, will be drawn dynamically based on coordinate data supplied by the software.
- **Audio:** Music playback will be handled by the software utilizing the onboard Wolfson audio codec, allowing for standard audio file decoding, such as WAV, using Linux libraries.

- **User Input:** The game will interface with a custom 9-button controller. Because the controller utilizes an Arduino Leonardo to act as a USB HID keyboard, the software will read button presses asynchronously via the Linux /dev/input/ event system to minimize input lag.

3. Hardware-Software Interface

The system will rely heavily on the Lightweight HPS-to-FPGA bridge to pass game state data from the software to the hardware at 60Hz.

Because the software recalculates the position of every active note each frame based on the song's elapsed time, we will likely use a memory-mapped register approach. The software will write an array of active note coordinates, specifically X, Y, and color or lane, to a shared memory block. The hardware sprite generator will scan these registers during the VGA blanking interval to render the notes on the screen. Additionally, the software will send a 9-bit mask representing current button presses to trigger visual "lane hit" feedback effects in hardware.

4. Major Tasks

- **System Architecture Design:** Finalize the memory map, the hardware/software interface style, and the maximum number of simultaneous sprites supported by the hardware.
- **Software Development:** Write C/C++ code to read beatmap files, synchronize a game timer with audio playback, calculate hit/miss accuracy, and continuously poll the USB controller for input events.
- **Hardware Accelerator Design:** Write SystemVerilog for the VGA timing generator, the lane background generator, and the sprite rendering engine.
- **Device Driver and Bridge:** Develop the Linux interface to allow the C program to efficiently write the frame's note coordinates to the FPGA registers.
- **Integration and Testing:** Synchronize the hardware video rendering with the software audio and calibrate the input windows for the USB controller.