Oscilloscope Guitar Hero Proposal

Druck Green
Daniel Shaar

Overview

This project aims to recreate the classic video game *Guitar Hero* using only analog circuitry (the only exception being the use of a device, such as an Arduino, whose usage is detailed later). The game will be played on an oscilloscope in XY mode with a few simplifications. A user will have four buttons to press in coordination with the four notes on the screen. The notes will form some rhythmic pattern and generate audio feedback, which will indicate correct and incorrect button presses for the user.

Design

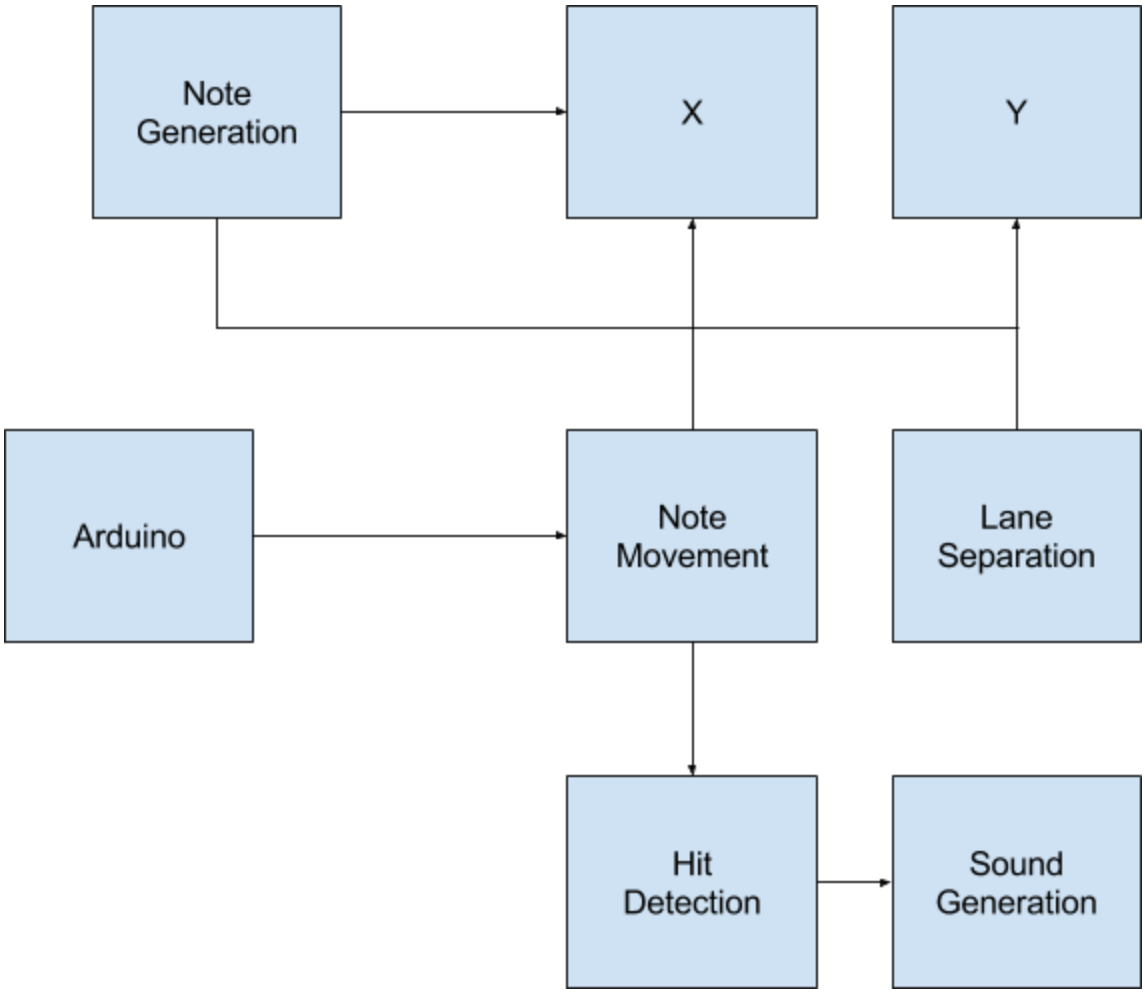

Figure 1: A broad overview of the operation of the circuit. Note that the circuit can be viewed as a black box with the Arduino and user button presses in the Hit Detection module as input, and XY and Sound Generation as output.

Module Descriptions

Before detailing the exact functions and circuitry involved in producing the gameplay, we will give a brief overview of each module's function and interactions with the other modules.

At the core of this game is the ability to produce the notes on the screen for the user to interact with. In order to achieve the visuals necessary for gameplay, many waveforms are generated that are used as input into the X and Y channels of the oscilloscope (note that this is intended for scope tilted sideways, so that the aspect ratio of the screen is longer vertically than horizontally). Rapidly oscillating between the notes, so that they all appear on the screen at a reasonable frames per second screen refresh rate, allows for many waveforms to be visible on the scope at once. To make this as modular as possible, one circuit module is responsible for the creation of a single note on the screen. This module is the **note generation** module.

To produce many notes, we make use of a **lane separation** module. The responsibility of this module is to store the horizontal state of each note, which will be a fixed value. Additionally, this module will rapidly switch between all the note states, so that they can all appear on the screen at the same time. When we add the output of this module to the previous module, we will be able to produce four notes on the screen at different horizontal positions.

However, *Guitar Hero* would not be much of a game if the notes never moved. To accomplish this task, the notes must progress vertically from an initial position to a final position and reset appropriately once they hit the end. This is done in the **note movement** module, which we also combine with the previous modules. This module will have the responsibility of reading note patterns from memory, and producing the appropriate waveform to move the notes.

User interaction capabilities will be handled in the **hit detection** module. This module consists of 4 buttons and circuitry to detect when notes have been hit and missed. This module will utilize the output of the note movement module to determine if the user's timing was accurate.

Finally, once the results of the user's button presses have been processed, we ultimately want to make some noise! This final functionality is handled in the **sound generation** module. Using just the output from the previous module, it will make

different tones depending on whether or not the user successfully timed his or her button presses.

Note Generation

We are using an op-amp Schmitt trigger with RC feedback to produce the initial oscillations that we'll need to generate sinusoids. The square wave output of this module has fourier series coefficients which are proportional in magnitude to $\frac{1}{n}$ at odd Harmonics, and are zero at even harmonics. To take advantage of this to produce a sine wave, we use a Sallen-Key LPF to attenuate all of the harmonics past the first one. The second order Sallen-Key filter provides greater attenuation at higher frequencies than would a first order passive LPF, which means our output will be a closer approximation to a sinusoid. To generate a circle using the X and Y channels requires 2 sinusoids which are 90 degrees out of phase with each other and have the same magnitude. We are using an all pass filter to shift the phase at the desired frequency without changing the magnitude. We then feed the input and output of the all pass filter into channels X and Y to produce a circle.

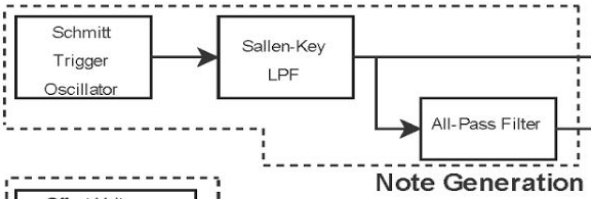

Figure 2: A block diagram containing the key components for the Note Generation module.

Note Movement

We are using an Arduino to store the order of the notes to be played, and to handle the timing for notes appearing on the screen. The arduino will have four outputs, one for each lane of notes. The outputs are normally grounded, and pulse high when it is time for that lane's note to appear at the top of the screen. Each output is connected to a separate ramp circuit, whose output rests at ground. The ramps are triggered when their input goes high and only generate a single ramp per trigger. These ramps will be added to the X component of the notes and provide the DC offset for the notes to fall down the screen. Because each note can have a different height on the screen and only one DC offset can be added to the X channel, we need to switch between ramps in order to generate four notes. In order to avoid any problems with aliasing, we will be switching at a frequency much higher than that of the note generation circuitry. To implement this, we will be using the analog equivalent of a multiplexer with four inputs and four select bits. The select bits are generated by four distinct MOSFET ring oscillator circuits, each with a 25% duty cycle and each 90 degrees out of phase form the next. This guarantees that the output of only one ring oscillator will be high at any moment. In the multiplexer, each oscillator output will correspond to one ramp output, and the output of the

multiplexer will be the buffered output of the ramp circuit whose corresponding oscillator output is high. The analog multiplexer will consist of a non-inverting op-amp adder whose inputs are the outputs of the ramp circuits. Connected between the inputs to the adder and ground are four n-channel MOSFETS whose gate are connected to the outputs of the ring oscillator circuits. This allows us to shunt all but the current note's DC offsets to ground.

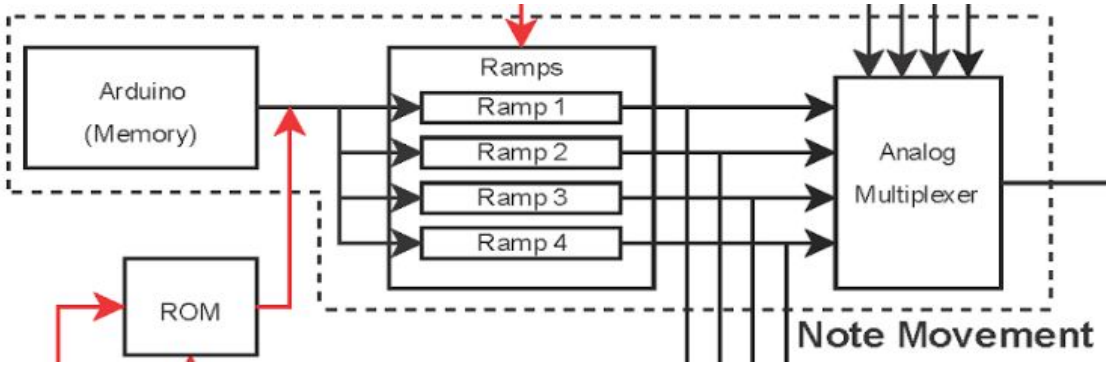

Figure 3: A block diagram containing the key components for the Note Movement module.

Lane Separation

The circuitry here is identical to the note movement circuitry. The only difference is that we have replaced the ramp circuits and Arduino with four DC voltages which do not change with time. The select inputs to the multiplexer are using the same ring oscillator outputs as the note movement circuit to ensure that everything is in sync. These Offsets will provide the separate lanes for notes to fall down.

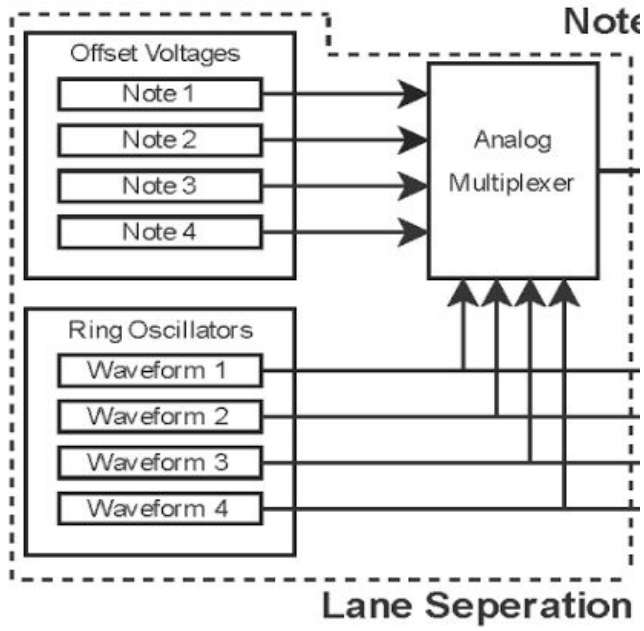

Figure 4: A block diagram containing the key components for the Lane Separation module.

Hit Detection

When the user hits a button, we want to detect whether they timed their hit correctly, which means the note corresponding to that button is near the bottom of the screen. To determine a notes vertical position on the screen, we simply have to look at the output of the ramp circuits from the note movement module. In our specific implementation of the ramps, the peak voltage corresponds to the note reaching the lowest point on the screen (which may fall off screen). Furthermore, when the ramp voltage reaches its peak value, it immediately falls to ground. An easy way of checking whether the note is passing by the bottom edge of the screen then is to simply check it the ramp voltage is above some threshold value. We accomplish this with four op-amp comparators, one for each note. The output of each button (high on press) and its corresponding comparator are then sent through an analog AND gate. Finally, to detect if any notes were hit, the outputs of all four AND gates are added together. A voltage of greater than ground at the module output implies that a note was hit.

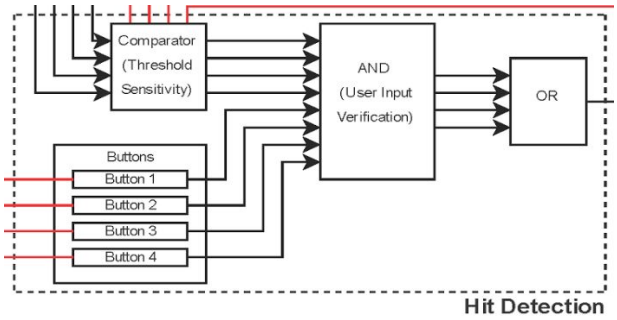

Figure 5: A block diagram containing the key components for the Hit Detection module.

Sound Generation

The sinusoid for the tone is created by a simple ring oscillator with capacitors and resistors for low pass filtering. The oscillator produces an output when power is applied. To control when a tone plays, the power rail to the oscillator will be connected via a MOSFET whose gate is controlled by the output of the hit detection module. To provide enough current and a low output impedance to drive the speaker, the output of the tone generator will be fed through an emitter follower amplifier.

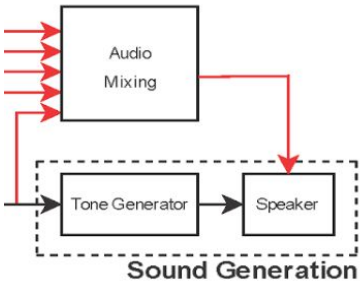

Figure 6: A block diagram containing the key components for the Sound Generation module.

Fully Detailed Block Diagram

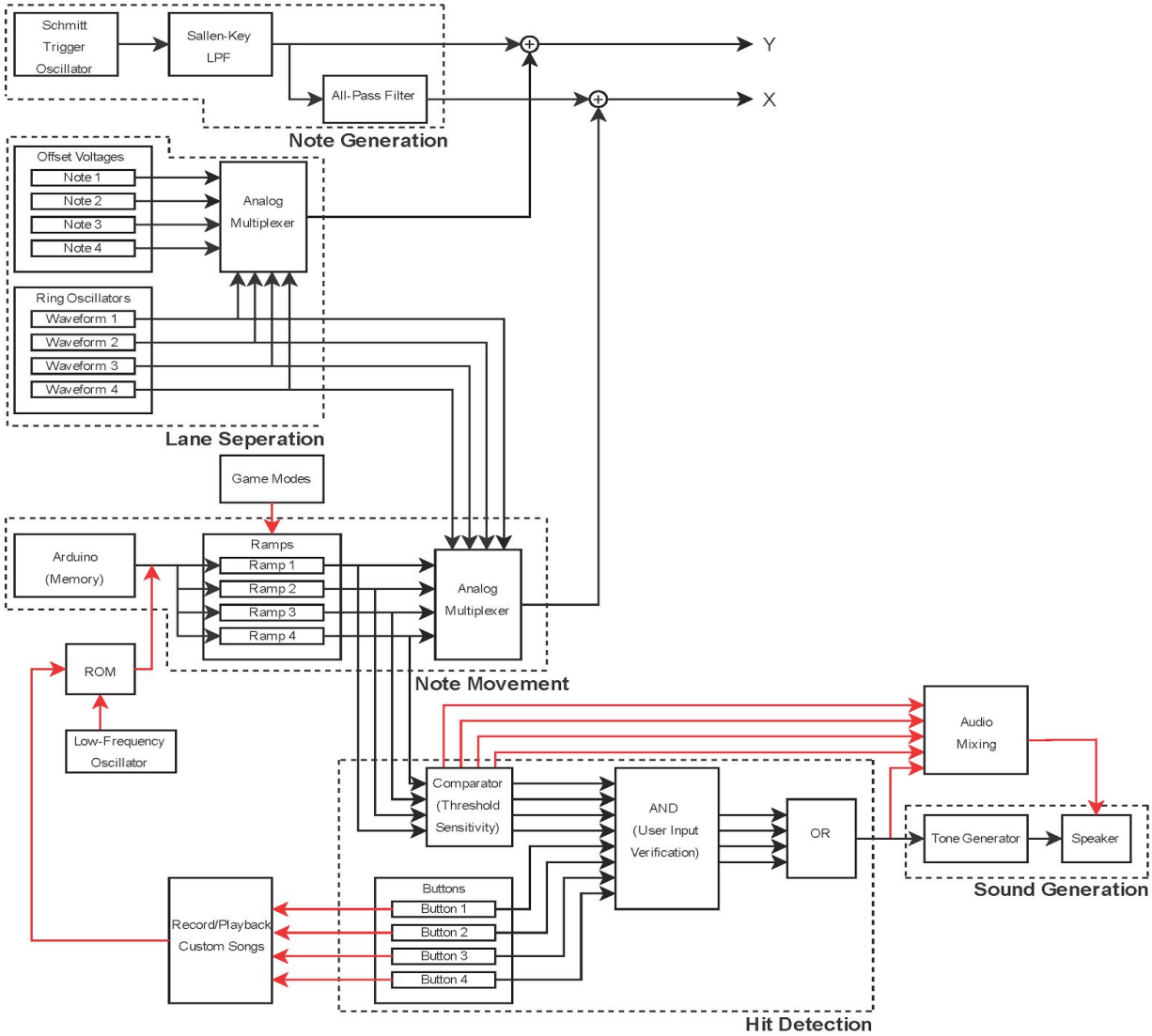

Figure 7: A detailed figure of the operation of the entire circuit.

Stretch Modules

The modules required to meet our stretch goals have their inputs and outputs marked in red. The audio mixing module will allow us to generate 4-bit music, by using a different tone generator for each note, and adding the tones. The game modes module will interact with the note movement circuitry to add different difficulty levels to our game by changing the number of lanes with notes and the speed at which they fall. In order to make our project more fully analog, we would replace the Arduino with a ROM onto which we have loaded the notes to be played in order. The notes would be read using the low frequency oscillator. Finally, the last stretch module would allow users to record a song into ROM and then play the game using that song.

RESPONSIBILITIES

Both members will be responsible for reviewing and quality checking the functionality and design of all of the modules. However, the design and implementation of specific modules will be divided equally between members. Druck's modules include note generation, lane separation, and note movement, while Daniel will be responsible for hit detection and sound generation. Responsibility for stretch modules will be divided based on progress in other modules.

TIMELINE

Week of April 9th - Note Generation Module
Week of April 16th - Note Movement and Lane Separation
Week of April 23rd - Hit Detection and Sound Generation
Week of April 30th - Stretch Goal Modules and Testing/Debugging
Week of May 7th - Finish Testing/Debugging

CONCLUSION

For our team, what makes a circuit interesting is that it is a applicable to a wider audience, practically usable, interactable, and novel. We chose to complete this project because it meets all of these criteria, and is therefore interesting to us. An interesting circuit is also challenging to design. Many parts of our circuit require us to think creatively to implement digital functionalities in an analog setting. This is something that will challenge us by forcing us to find solutions to real-world non-idealities that don't exist in digital electronics.