6.111 Final Project Proposal

Abstract

The goal of this project is to simulate the membrane of a drum and synthesize a passable drum sound. We will do so by taking analog input, produced by a piezo grid, which encodes the input force and strike zone (or “epicenter”) of the strike. Using this data we will infer the location of the strike in order to simulate the drum membrane and produce a sound resembling what would happen if playing an actual drum. We hope to be able to do this with a small latency using the Nexy4 DDR board enabled with an Artix-7 FGPA chip.

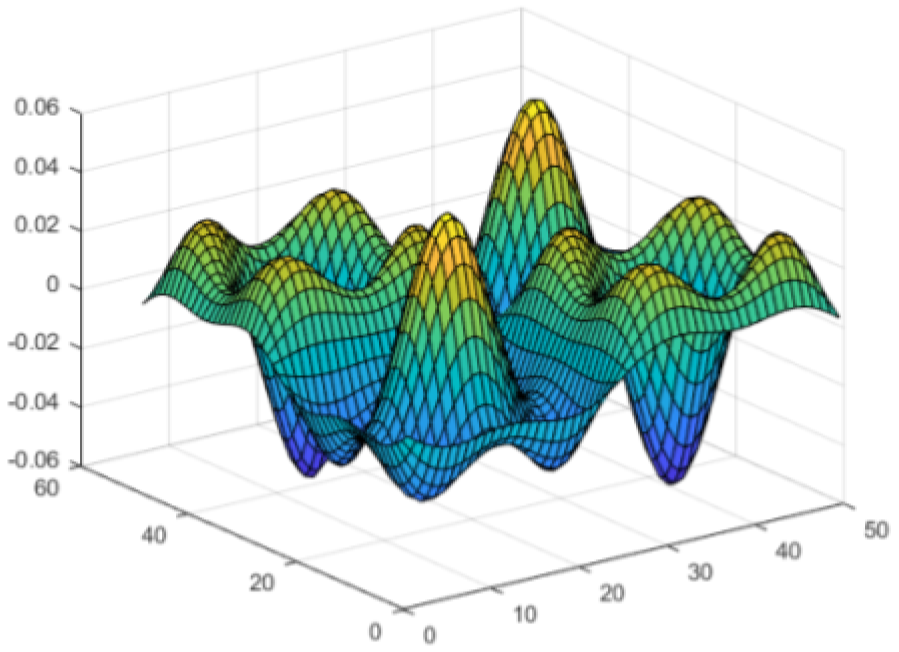
Background

Electronic drum pads use analog signals from piezoelectric components to produce percussive noises using specialized hardware.
Early (“low quality”) electric drums synthesize the sounds by means of simulating the response of the drum as opposed to newer (“higher quality”) percussion instruments, which use mesh heads that closely mimic the response of typical drum membranes. The difference between these models is that the higher quality drums use a single piezo (or very few), in conjunction with tensioned applied mesh heads, and acts as a glorified amplifier. The “lower” quality drums have a peculiar problem in that they need to synthesize the sound somehow.
Our goal is to mimic the perturbation of this membrane using and synthesizing the resulting sound. We believe this to be an interesting problem since simulating multiparticle systems can be computationally expensive; but with some good hardware we can produce a decent sounding circular pecussive membrane within a reasonably small latency.

Numerical Simulation of Drum Membrane

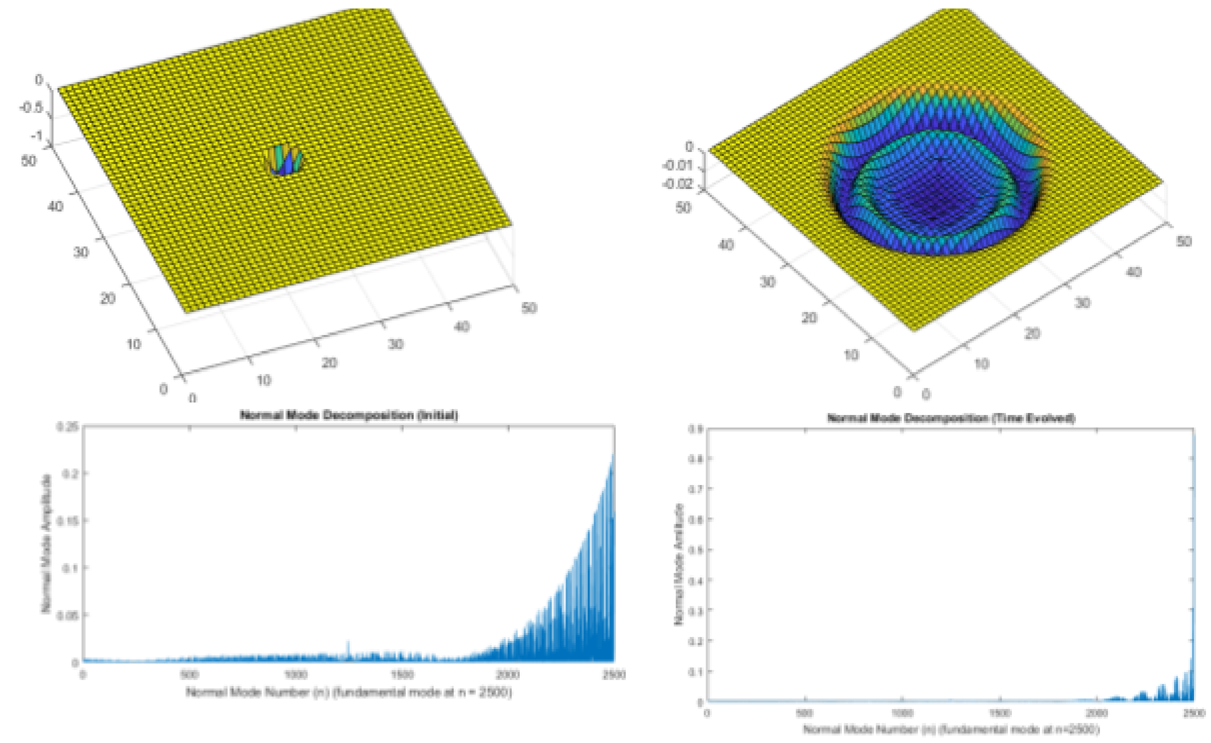
- In order to make this a tractable problem we will be making some reasonable assumptions:
    - The membrane can be represented as discretized regions of 2D space, which we will call “particles”.
    - The membrane is only one “particle”in depth or is composed of a single layer of particles.
    - The air column beneath the membrane does not add to the sound but does damp the movement of the membrane somewhat
    - The tension is strong enough to avoid having to use a damping constant. Although we may explore this later in later iterations.
    - For now we will assume that only the boundary conditions are 0, but may scale by some gaussian function time permitting.

- To characterize the motion of a 2D, finite, membrane we assume the discretized particles interact only with their nearest neighbors.
    - We assume that the force applied by each neighbor is proportional to the positional difference between that neighbor and center particle.
    - Additionally we assume a circular (radius N/2) boundary condition, inscribed an NxN grid of particles.
- Using Euler's method, with small time step corresponding to sampling interval, we calculate the new position vectors for each particle.
- Another method we will try to explore is using the normal modes of the system to characterize the time evolution given an input. We have decomposed the eigenvectors for a 50x50 grid of particle system and found the normal modes using Matlab.
    - Using the normal modes we can characterize the time evolution of the system given some input (strike on the drum pad).
- We have already implemented the simulation described above using Matlab to calculate the normal modes of the system. We plan to store the normal modes found by matlab in the ROM of the Nexys board so they won't need to be calculated in real time.



Above: The 25th Normal mode of a 50 x 50 particle system as calculated by Matlab. These normal modes could be stored in ROM on the Nexys board to find the frequencies present in the membrane simulation

- Matlab has also been used to verify our methods for the time-evolution of a 50 x 50 particle system and our methods for finding the time-domain frequencies present in the system, and our demonstrate our method for calculating the time-domain frequencies in the system by calculating the amplitudes of the normal modes present in the system, which can be used to find the fourier transform



Above: A small gaussian perturbation is introduced to a 50 x 50 particle simulation on the left. The system was time evolved for 250 small time steps and shown again on the right. Notice the perturbation propagating outwards from the center and oscillating as a function of time. The normal mode decomposition was performed for both cases, and the amplitude of each normal mode is shown below the corresponding spatial representation. Note how the lower frequency normal modes quickly become dominant (for some reason Matlab put the low frequency normal modes with a higher mode number).

Sound Generation

- In our first iteration we will just use Euler's method to determine the state of the system one small time step into the future. From this we will take the particles' amplitudes and transform them in some meaningful way to produce a digitized sound.
- In later iterations we will use the fourier transform of the membrane provided by decomposing the simulation data into its normal modes, then using the iFFT gives the sound amplitude in the time domain.
- DAC output (or the audio output of the Nexys board) will be used to convert from our digitized sound to analog output.

Modules

We need to run a time step of the simulation, recalculate the fourier transform and the IFFT every sample (or at 48KHz). If we use a 100MHz clock, then we have 2083 clock cycles to perform these tasks.

- **Drum Pad Input (ADC) [Ben]**
  The first step of our input processing is a module to collect and process the outputs of our piezos from our drum head. This will take the raw outputs from the piezos as data inputs and process these into 32 intelligible bits that will be passed along to other modules in order to simulate a drum.
- **Infer Epicenter [Robbie]**
  This will include finding a way to represent the normally floating number output as a 32-bit integer and will pass along the information about how hard the strike was as well as the x-y position on the drumhead where the strike occurred. These outputs go to an image generation module and a module for simulating what this strike would do to an actual drum head.
- **Calculate State [Ben]**
  After inferring the epicenter and magnitude of the input force we can calculate the resulting state of the multi-particle system. This includes finding the new position and velocity vectors of each particle in the system. The positions and velocities of each particle in the system would need to be stored in RAM. We would ideally simulate a system that consists of thousands of particles, and thus this module would require thousands to tens of thousands of bytes of BRAM/SRAM. The new positions and velocities would be calculated for the time step between each audio sample by solving each particle's equations of motion using Euler's Method. This module would take the output of the infer epicenter module as well as the previous position and velocity state stored in RAM, and its output would be the updated position and velocity data stored in RAM. As this module would require thousands of bytes of RAM and thousands of arithmetic calculations, it's complexity is high.
- **Sound Generator [Robbie]**
  Using the current state of the system we will transform the amplitudes of the particles into some meaningful output (possibly the way the amplitudes interfere at some fixed point above the surface of the drum). This amplitude is scaled appropriately for the DAC output and is fed into the speaker.
- **Time Evolution Solver * [Ben]**
  As one of our reach goals, we would like to explore an alternative sound generation method. This method would entail decomposing the simulation data at each timestep into normal modes/standing waves, which each correspond to a

specific frequency. This would allow us to construct the fourier transform of the overall system in the time domain. Taking the IFFT of this would yield the audio output. This module would require the precalculated normal modes (or some subset of them) to stored in ROM. The position data from the simulation would be input. For every audio sample, the vector projection of the input data onto each normal mode would be calculated, and used to weight the corresponding normal mode frequency in the fourier transform. An IFFT would then be used to generate the audio output for each sample. This module would require a significant amount of ROM (e.g. hundreds of thousands of bytes) and clock cycles, and thus is high in complexity.

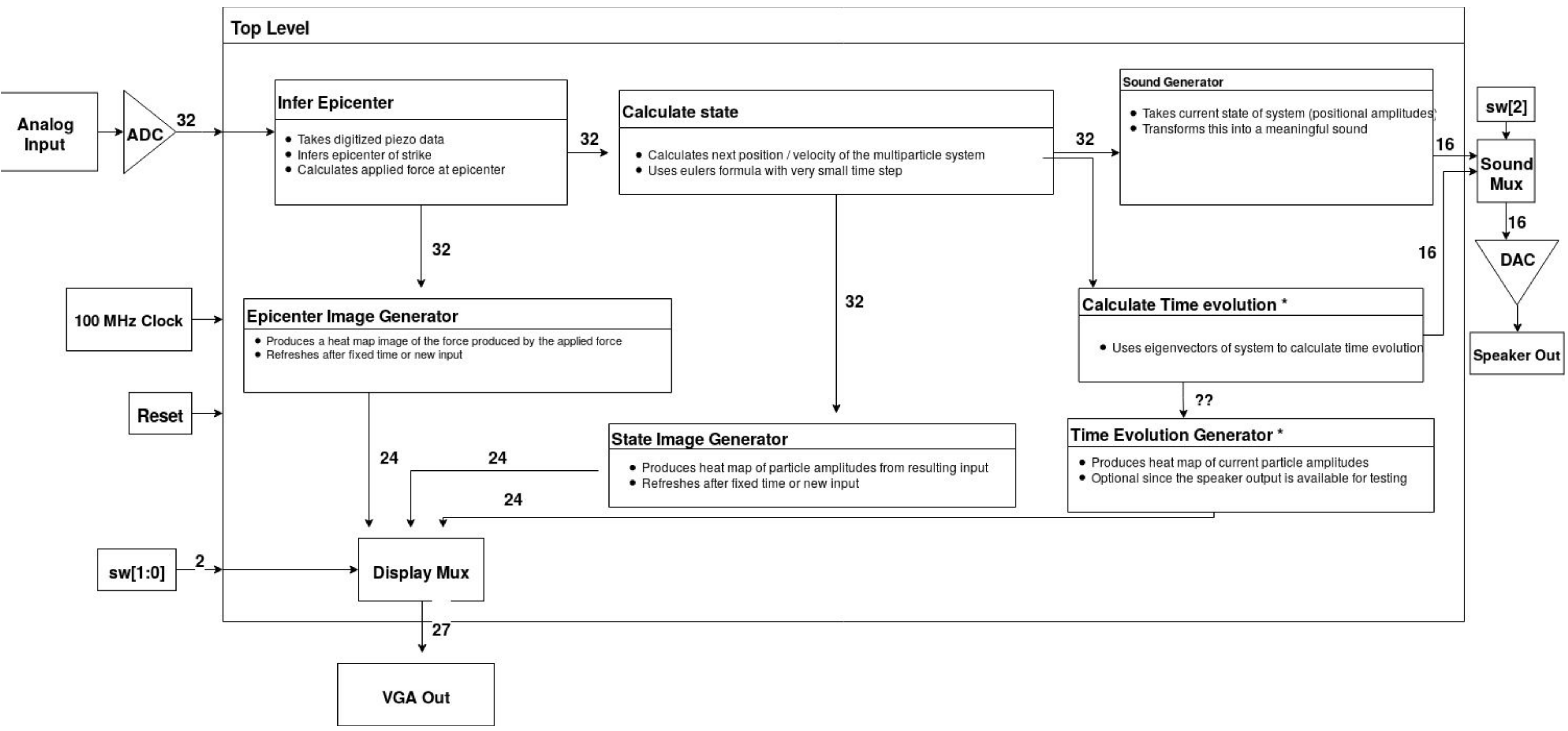
- **XVGA † [Evan / Lab 3]**
  Although not depicted, this module will produce a vclock for image display as well as an hcount, vcount, hsync, vsync, and blank signal that are used for displaying images through a VGA output. These are then passed into each image generation module.
- **Epicenter Image Generator [Evan]**
  Receiving inputs from the module that determines the force and strike position from the piezos and the xvga module, this module displays where a strike has occurred on our pad as well as how hard the strike was.
- **State Image Generator [Evan]**
  This will take in the outputs of the drum membrane simulation module and XVGA module. The module will then output a 2-D image of a drum head and have different colors for discrete points on the simulated drum head that communicate the vertical velocity of each point, using warmer colors for faster velocities. This will help show what parts of our simulated drum head move the fastest and be a debugging step before calculating the output frequencies.
- **Time Evolution Generator * [Evan]**
  This module receives the outputs of the XVGA module and the sound waveform module in order to display a rough approximation of the sound output from our simulated drum. It does so by displaying the past several samples versus time to give us an idea of the output we should expect from the speaker at the end of the process.
- **Sound MUX [Rob, Evan]**
  In later iterations we will be trying to incorporate different methods for generating the sound. This module will allow us to compare the quality and latency of different methods.

- **Display MUX [Rob, Evan]**
  This takes as input the rgb info from each image generation module and determines which to display via the value of sw[1:0], sending the relevant information to the VGA output.

External Components

- We plan to use four piezo transducers to readout the force of the drum hit and infer the position of the hit. We hope to borrow these from the EE stockroom, or else buy them. We anticipate the price would be around $10, as piezos are quite widespread
- To get the voltage range of the piezo's to match that of the Nexys board's ADCs, we may need to use several simple op amp circuits (e.g. LM072's). We hope to use some op amps from the EE stockroom, or else buy them. We don't expect the price of these to be more than $10.
- We plan to use a ceramic plate or a high-density foam slab as the drumpad itself. We hope to buy this from a hardware store or order it, and don't anticipate the price exceeding $20.
- We plan to attach use small foam cones to mechanically couple the piezos to the drum pad. These we would need to purchase or fabricate ourselves from stock material. We don't anticipate the cost of this material exceeding $10.
- The total price of materials (if nothing could be used from the EE stockroom) would be less than or equal to ~$50.

Block Diagram


Top Level
Analog Input
ADC
32
Infer Epicenter
• Takes digitized piezo data
• Infers epicenter of strike
• Calculates applied force at epicenter
32
Calculate state
• Calculates next position / velocity of the multiparticle system
• Uses eulers formula with very small time step
32
Sound Generator
• Takes current state of system (positional amplitudes)
• Transforms this into a meaningful sound
16
sw[2]
Sound Mux
16
DAC
Speaker Out
16
32
100 MHz Clock
Epicenter Image Generator
• Produces a heat map image of the force produced by the applied force
• Refreshes after fixed time or new input
32
Calculate Time evolution *
• Uses eigenvectors of system to calculate time evolution
??
Reset
24
24
State Image Generator
• Produces heat map of particle amplitudes from resulting input
• Refreshes after fixed time or new input
Time Evolution Generator *
• Produces heat map of current particle amplitudes
• Optional since the speaker output is available for testing
24
sw[1:0]
2
Display Mux
27
VGA Out