Rahul Yesantharao
Jacob Pritzker

6.111 Final Project Proposal

I. Project Description

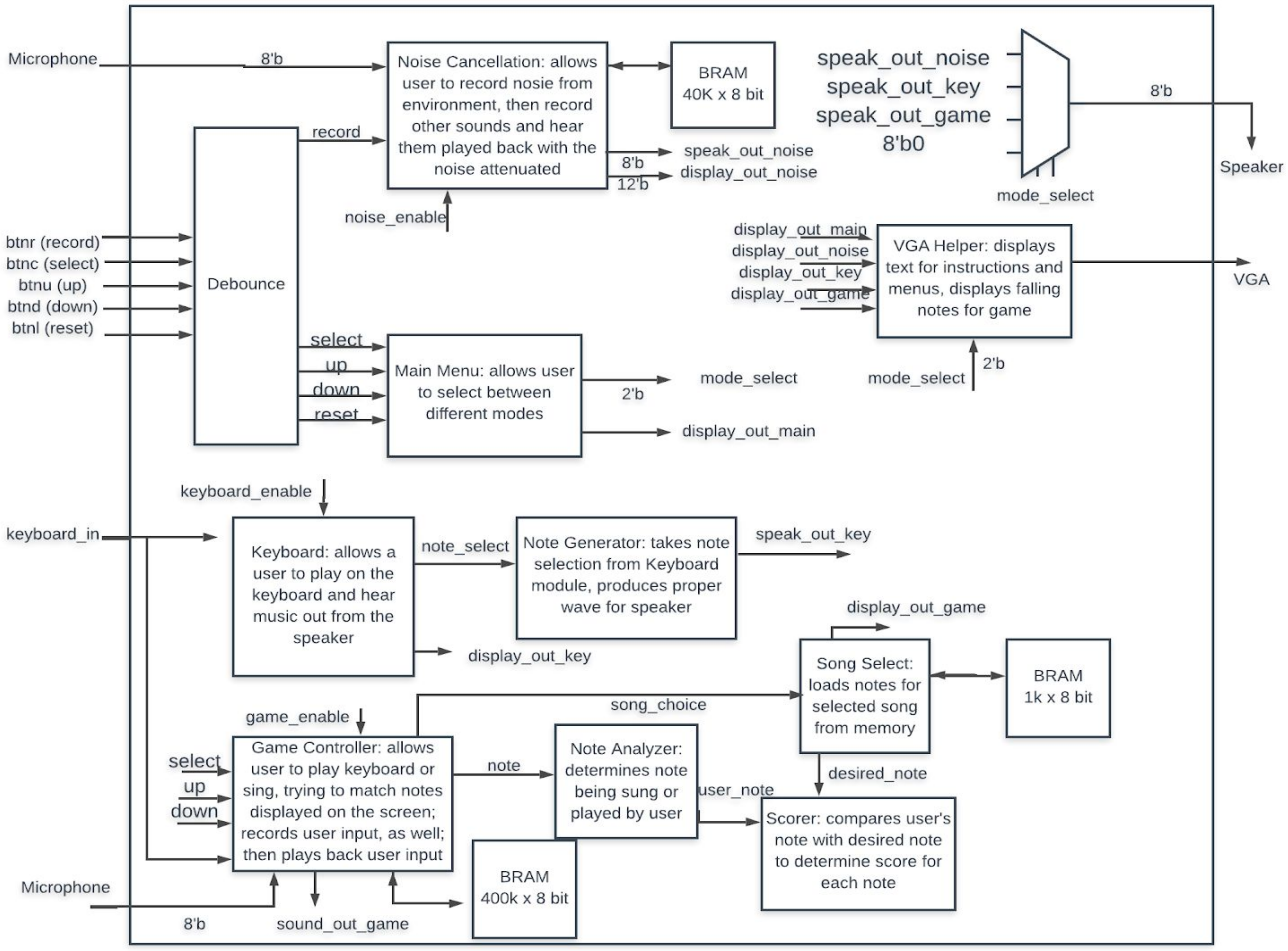
For our project Freeform Production of Gorgeous Audio, we want to make a music system with three primary modes. There will be a noise cancellation mode, a keyboard mode, and a game mode. The project will have a microphone and a keyboard for user input, and a speaker and a display screen for output.

In the noise cancellation mode, a user will record a few seconds of sound from their surrounding environment (simply record without saying anything), and then our system will analyze the recording in order to identify the spectral content of the noise in that environment. Then, a user will be able to make a new recording of them talking or singing, and our system will attenuate the noise from the recording and play back a cleaned version of the recording.

In the keyboard mode, a user will be able to play a basic keyboard. They will be able to press keys and hear the output from those keys through the speaker.

In the game mode, the user will play a Guitar Hero-esque game, either singing into the microphone or playing the keyboard, trying to match notes being displayed on the screen. At the end of the song, the user will be scored on how well they did. We will also provide a playback of the song they performed. If they had previously set up noise cancellation, then the recording played back will have attenuated the noise.

II. Block Diagram



III. Module Descriptions

- Debounce Module
    - Clean signals from input buttons
- Main Menu Module
    - This module will allow a user to select between the 3 modes of operation of our project: noise cancellation, keyboard, and game.
    - It will take the select (btnc), up (btnu), down (btnd), and reset (btnl) signals as input. As output, it will have a 2-bit mode_select register (to indicate whether the user is in the main menu, noise cancellation mode, keyboard mode, or game

mode). It will also have VGA output signals that go to the VGA Helper module. The display screen will show the main menu, allowing a user to choose a mode.
    - This module should not involve any complex calculations or have any difficult throughput requirements, and it requires no BRAMs.
    - To test this module, we will use LEDs on the Nexys4 to make sure that when we select a mode, the system registers that we have entered that mode.
    - Jacob will implement this module.
- Noise Cancellation Module
    - This module will have a user record their environment, then it will process those audio samples, and then determine its process for noise cancellation. It will then have a user record new audio samples and play them back with the noise attenuated.
    - This module has the record button (btnr), the 8 bit samples from the microphone, and a noise_enable signal as input. The noise_enable signal is set by the top level module and essentially turns on/off the noise cancellation mode.
    - The module writes to and reads from a 40k x 8-bit BRAM. It initially stores the audio samples of environmental noise there, then processes that data to determine its noise cancellation strategy. Once it has that, it can then take in new audio from the user, overwrite the noise audio in the BRAM, and then play back a noise-cancelled version of this new recording.
        - We plan to read some papers about different noise cancellation techniques, so we are not yet sure of the exact complexity of this module. We imagine it will involve an FFT, but beyond that we need to do some research.
    - The module outputs 8-bit values to the speaker, as well as VGA output signals that go to the VGA Helper module. The display screen will simply display instructions for the user in this mode.
    - This module does not have a throughput constraint, other than needing to sample the audio quickly enough. It is fine if the noise cancellation takes a bit of time.
    - To test this module, we will first record noise from the environment and let the module determine its noise cancellation strategy. Then, we will record noise from the same environment again, and we will know things are working if we hear nothing or very little when the module has attenuated the noise from this new recording. In addition, we will record a few different sounds (singing, humming, snapping, etc.) and make sure that the sounds played back have little noise in them.
    - As of now, we think Jacob will implement this module.
- Keyboard Module
    - This module will allow a user to play the keyboard and hear the corresponding notes from the speaker.

    - As input, it will have the data from the keyboard, as well as a keyboard_enable signal from the top level module that turns on/off the keyboard mode.
    - As output, it will have VGA output signals (the display screen will just instruct the user to play music on the keyboard). It will also have a note_select signal to send to the Note Generator module.
    - This module does not require any difficult computation. Its latency should be low so that there is not a lag between a user pressing a note and hearing that note.
    - To test this module, we will use the LED segment displays on the Nexys4 to see if the correct note(s) are being read in by the Nexys4 when we press keys on the keyboard.
    - Jacob will implement this module.
- Note Generator Module
    - This module will convert notes played by the user on the keyboard into appropriate sine waves to drive the speakers.
    - It will take the note_select signal from the Keyboard Module as input, and it will have an 8-bit output value to drive the speaker.
    - It will use a sine wave generator module (not shown in block diagram for clarity), similar to what we used in Lab 5A.
    - Its latency should be low so that there is not a lag between a user pressing a note and hearing that note.
    - To test this module, we will simply play the keyboard and see if we hear the correct notes out from the speaker.
    - Jacob will implement this module.
- Game Controller Module
    - This module will allow a user to either sing into the microphone or play on the keyboard, playing a Guitar Hero-esque game. It will also record the user input, allowing for playback afterwards.
    - The module will take as input the game_enable signal to determine whether or not it should be actively responding to the data input.
    - It will also take in btnc, btnu, btnd (select, up, down respectively), to allow access to the game menu select controller (move up/down and select).
    - It will also take the microphone input (8 bit depth) so that it can allow for the user to sing the song.
    - This module will output the microphone data to the note analyzer, and receive back the actual note being sung by the user, as well as the sound output that the board should output to the speaker. It will output a song_choice that the user makes to the Song Select Module to allow for the correct song information to display on the screen.

    - This module will have an FSM controlling the game state and menu, and it will require a large amount of BRAM (400K x 8-bit) in order to store the user's singing/playing for playback. It will be fairly complex to maintain all of the correct states, and will use a corresponding amount of hardware (state registers, muxes/LUTs for transitions).
    - To test this module, we will have a test bench that asserts various combinations of the control/select inputs and check whether the internal state is correct. We will then try to actually play the game and make sure different sequences of inputs works.
    - Rahul will implement this module.
- Note Analyzer Module
    - This module will take in the audio from the microphone or the keyboard (routed through the game controller), and will perform Fourier analysis to determine which note is being sung by the user (or combinational logic in the case of keyboard input).
    - It will require the usage of the FFT hardware to perform this analysis quickly and accurately. It is not necessary for it to have a high throughput, as it only analyzes one note at a time, but it needs to have very low latency so that this analysis can be done in real time as the user sings.
    - To test this module, we will have a test bench that inputs various waveforms of pure notes (as well as notes modulated by some noise) and ensure that it accurately detects them. We will then test it "live" by singing notes into the module and ensuring that it accurately detects them.
    - Rahul will implement this module.
- Song Select Module
    - The Song Select module takes as input the index of the song selected by the user, and it uses this to begin outputting the notes of that song, in sequence. It reads these notes from BRAM. We currently plan on using a small amount of BRAM (1K x 8-bit) to store some basic songs to verify the functionality of our game. The Song Select module also outputs the VGA information representing the game (falling notes, similar to Guitar Hero) so that the VGA helper module can correctly display the game.
    - This module will not need to have a very large throughput, but it will have to have very low latency, because it is streaming song data in real time to both the Scorer and the VGA Helper Module.
    - To test this module, we will have a test bench that asserts various song select options, and checks whether the resulting waveform is accurate. This module will also be tested when we ultimately play our game and see if the correct song is played.

    - Rahul will implement this module.
- Scorer Module
    - The scorer module takes input from the Song Select Module and the Note Analyzer module, and scores the user on whether or not they are singing/playing the correct note. Because the Note Analyzer Module normalizes keyboard/singing input to a simple integer representing the input note, the module simply has to compare this against the expected note at that time interval, input by the Song Select Module. Thus, it has simple combinational logic and uses a small amount of register space to save the user's score.
    - To test this module, we will have a test bench that inputs various combinations of user notes and song notes, and check whether the internal score state is correctly updated for each choice.
    - Rahul will implement this module.
- VGA Helper Module
    - The VGA Helper module takes as input the display output information from all of the submodules that perform display functions (described in each individual module description). It then puts all of this data in the correct positions on the screen, and outputs the complete pixel information to the VGA driver.
    - This module needs to have a good throughput, as it is driving a VGA screen of size 1024x768, and needs to support a 60Hz refresh rate. In order for this to be the case, it needs to efficiently perform all of the incorporation of the display signals it receives.
    - To test this module, we will first hook up each of the individual modules displays directly to the VGA driver, to ensure that they correctly produce VGA output. Then, having ensured this, we will input all of their output to this module and check whether it accurately combines them for the final visual output.
    - Jacob will implement this module.
- Top Level Module
    - The top level module will instantiate all of the other modules, and will receive the physical input from the board. It will appropriately route these input signals and intermediate signals between the various submodules, effectively "wiring" everything together. This module will not have much complex hardware beyond the submodules, and will mainly encompass appropriate signal routing. Thus, it will not have much hardware or throughput requirements; it is mainly important that it routes the signals with very low latency, as everything is operating in real time.
    - To test this module, we will use a simple test bench to ensure that any top level state is properly maintained. Then, we will test the system as a whole with

everything hooked up in top level and check for any bugs, which must then be from top level (if everything else is tested).
    - Rahul will implement this module.

IV. External Hardware

In order to accomplish all of our goals (including stretch), we will require the following.

- Keyboard (for piano input): ~$40
    - The keyboard will most likely be interfaced through USB input to the FPGA. This is dependent on which keyboard we end up purchasing, and whether there is already one available for us to use in the lab. Upon choosing the device, we will be able to determine exactly how complex the input interface module will be, but it should be moderately complex in expectation.
- Speakers (for audio output): ~$10
    - The speaker will be interfaced through the mono audio output jack on the FPGA. This is driven by a PWM signal, as used in Lab 5A, and should not be too complex to set up.
- Microphone (for audio input): ~$20
    - The microphone is interfaced to the FPGA through the JXADC PMOD port as in Lab 5A. The setup should be similar to that used in Lab 5A.
- External Monitor (to interface with the user): (in lab)
    - The external monitor will be 1024x768 (as used in Lab 3), and will be interfaced through the VGA port on the FPGA. It will be fairly complex to drive all the necessary display components (game, menus, etc.) on the display.

Although we have listed the prices above, we believe that most of these components are already available in the lab; after conferring with staff we will have better estimates.

V. Projected Goals

Baseline goals:

- Make a functional keyboard using the switches on the Nexys4
- Guitar Hero-based game with keyboard input and basic UI (display a note when it should be played)
- Implement noise cancellation

Expected goals:

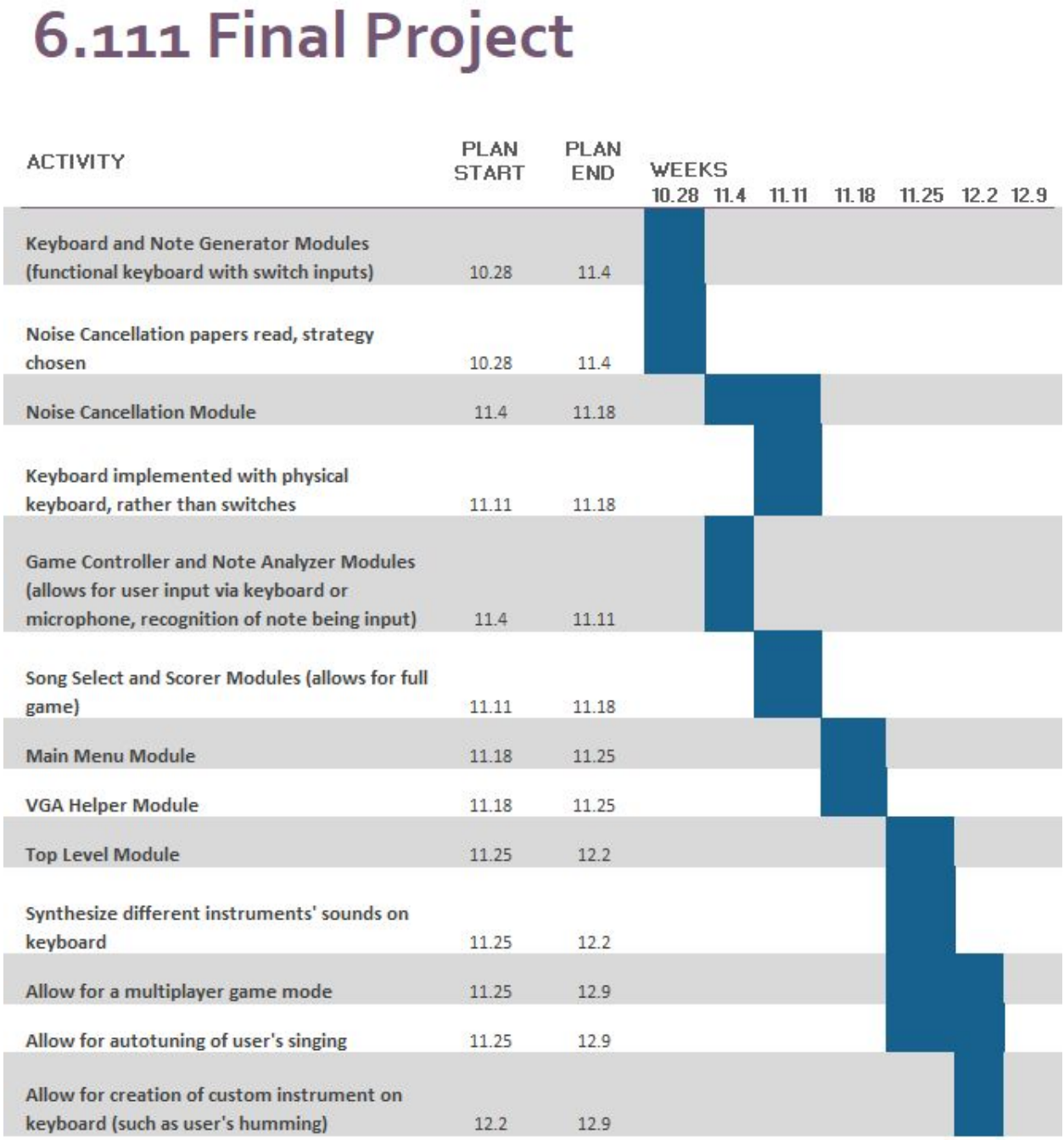
- Use the MIDI keyboard input for piano input (instead of switches)
- Incorporate some synthesized audio types in the keyboard input mode (e.g. violins)

- Incorporate a singing mode in the game.
- Create a menu-based user interface to choose between modes
- Make the game UI more advanced (falling notes, etc.)

Stretch goals:

- Allow a user to synthesize a custom instrument on the keyboard; a user can hum or sing any note, and the system synthesizes the entire musical scale from that single note, then allowing a player to play the keyboard with the output sounding like their humming or singing.
- Make the game multiplayer, with one user playing the keyboard while the other sings.
- Autotune the playback after the game, so that the playback sounds similar to how the song was supposed to be played.

VI. Timeline / Milestone Goals



This Gantt Chart illustrates our planned timeline for the project.

Our milestone goals are essentially the completions of the various modules described above. The testing of these modules (and thus verification that we have reached our milestone goals) is described above.