KeySynth: a keyboard synthesizer with audio wave visualization and parameter changing tool.

Abrar Fahim (af4175)[1], Md. Jaber Hossain (mh6426)[1]

1. Department of Electrical and Computer Engineering, New York University, New York 11201, USA;

Abstract: The aim of this project is to make a software synthesizer using various Python-based audio processing techniques. Three main aspects of this project are as follows:
1. Real-time visualization of the frequency spectrum of notes played on the synthesizer.
2. A Graphical User Interface (GUI) to control the gain of the piano notes.
3. Generating signal spectrum of saved composition.
For the project, the following python library was utilized, Wave, Scipy, Math, PyAudio, NumPy, Tkinter, Matplotlib, and OS.

1. Introduction: In modern music, a synthesizer is synonymous with all. A synthesizer is a modern electrical musical instrument that uses the keyboard to generate a wide range of sounds and notes by combining various frequencies and techniques. Synthesizers use various methods such as subtractive, additive, and frequency modulation synthesis to generate audio waveforms **[1]**. The audio waveforms can further be modified using various filters, which can further boost or cut the wave frequency and envelopes. Synthesizers first emerged in the mid-20th century which were based on hundreds of vacuum tubes and punch cards. The Moog Synthesizers developed by Robert Moog was the first synthesizer based on the concepts of voltage-controlled oscillators, envelopes, filters, sequencers, and noise generators. And now, with the modern Digital Signal Processing techniques and high-performance processors, we have synthesizers like Roland Juno-DS61, Yamaha MX 49, etc.

For this project named "KeySynth," a 12 keys keyboard synthesizer was made based on 12 fundamental notes of a piano. The synthesizer uses three different wave forms to generate three different kinds of sounds. One of the main focuses of the is project is the real-time visualization of the notes that are played on the keyboard both in the time and frequency domain. The synthesizing software can also store the music composition and illustrate its Spectrogram.

The project is based on Python and uses libraries like Wave, Scipy, Math, PyAudio, NumPy, Tkinter, Matplotlib, and OS. For the purpose of generating the audio signal, GUI and audio signal recording and visualization.

2. Project Overview: Project "KeySynth" is an amalgamation of three parts, namely:

- Music Production Unit.
- Graphing Unit.
- Parameter Changing Unit.

2.a. Music Production Unit: The music production unit is the heart of the project where the generation of audio signals happens. To develop this unit, the PyAudio, Wave, Scipy, NumPy, and Math libraries played a vital role. The music production unit is based on the 12 basic notes or an Octave of a piano. The basic notes of a piano, along with the corresponding keys of the computer keyboard, are:

Notes	***Corresponding Frequencies (In Hz)***	***Corresponding Keys on the Keyboard***
A	440	A
B flat	466	S
B	494	D
C	523	F
C sharp	554	G
D	587	H
D sharp	622	Z
E	659	X
F	698	C
F sharp	740	V
G	784	B
A flat	831	N

Here each of the notes is geometrically distributed by a factor of $2^{(1.0/12.0)}$ from its previous note. As note A, which is 440 Hz, the next note, which is note B flat, has a frequency of 440 * $2^{(1.0/12.0)}$ = 466 Hz **[2]**. When the corresponding keys on the keyboard are played, an audio signal with the following notes is generated.

The music production unit can have three different input waveforms to generate the notes. The input waveforms are sine, sawtooth, and triangle. For different inputs, the synthesizer generates different types of audio signals. Because to generate the individual outputs for each individual note, the input goes through a linear filter called *lfilter*, which is a filter function defined in SciPy **[3]** library, which requires filter coefficient along with the input. So, at the very beginning of the program, when the first GUI pops up for input waveform selection, the user can select a waveform for which corresponding filter coefficients get generated. For example, the filter coefficients for three different input waveforms corresponding to note A (440 Hz) is illustrated below:

$Frequency, Decay\ Time\ \&\ Sampling\ rate$:

$f1 = 440 * 2^{\left(\frac{0.0}{12.0}\right)}, \qquad Ta = 0.5, \qquad Fs = 8000$

$Pole\ radius\ and\ angle$:

$$r = 0.02 * \left(\frac{1.0}{Ta * Fs}\right)$$

$$\omega_1 = 2 * \pi * \frac{f1}{Fs}$$

For Sine Wave:

$$Coefficients\ for\ second\ order\ filter:$$

$$a_1 = [1, -2 * r * \cos(\omega_1), \mathrm{r}^2]$$

$$b_1 = [r * \sin(\omega_1)]$$

For Sawtooth:

$$Coefficients\ for\ second\ order\ filter:$$

$$a_1 = [1, -2 * r * \text{signal.sawtooth}(\omega_1), \mathrm{r}^2]$$

$$b_1 = [r * \text{signal.sawtooth}(\omega_1)]$$

For Triangle:

$$Coefficients\ for\ second\ order\ filter:$$

$$a_1 = [1, -2 * r * \sin^{-1}(\sin(\omega_1)), \mathrm{r}^2]$$

$$b_1 = [r * \sin^{-1}(\sin(\omega_1))]$$

After generating the filter coefficients, the input note along with the generated coefficients are used to generate the output to the input note. Now the final output audio gets generated, which is the summation of each of the outputs generated from each of the input notes. So, if the y_{total} represents total output and $y_1, \dots, y_{12}$ represents individual outputs for each of the notes played the total output can be written as:

$$y_{total} = y_1 + y_2 + y_3 + y_4 + y_5 + y_6 + y_7 + y_8 + y_9 + y_{10} + y_{11} + y_{12} \quad \dots.(1)$$

This is how the music production unit generates the audio signal.

2.b. Graphing Unit: The graphing unit is responsible for generating time vs. amplitude and frequency vs. amplitude graphs in real-time and also generating Spectrogram from previously recorded audio files (.wav files) **[4]**. Here the y_{total} Output amplitude is shown in terms of time. Before that, the output is first fed into a filter that clips any value greater than the predefined maximum and minimum value. After that, Fourier Transformation is performed over the clipped y_{total} Using the NumPy library's *fft* (Fast Fourier Transform) module's *fft* function **[5]**. Which extracts the frequencies present in the output signal. Then the transformed output signal gets plotted as the frequency vs. amplitude graph. Both the time vs. amplitude and frequency vs. amplitude graphs were plotted using Matplotlib's *pyplot* module **[6]**. On the other hand, if the user selects the "Load Saved Audio Spectrogram" of the first GUI, the program generates a Spectrogram of the previously stored output using the Matplotlib library's *specgram* function **[7]**. Besides, all the plotting was done using the Matplotlib library.

2.c. Parameter Changing Unit: The main work of this unit is to change the input signals gain by utilizing a slider made by using the Tkinter library's *Scale* function **[8]**. The initial value is set as 1000 for the slider, and the maximum value is set to 20000. When a user presses a selected key on the keyboard, an impulse is generated at the particular frequency

or note to which the key is attached to. To bind a keyboard key to the program, the *Bind* function of the Tkinter library was used **[8]**.

The following diagram provides an overview of the whole project.

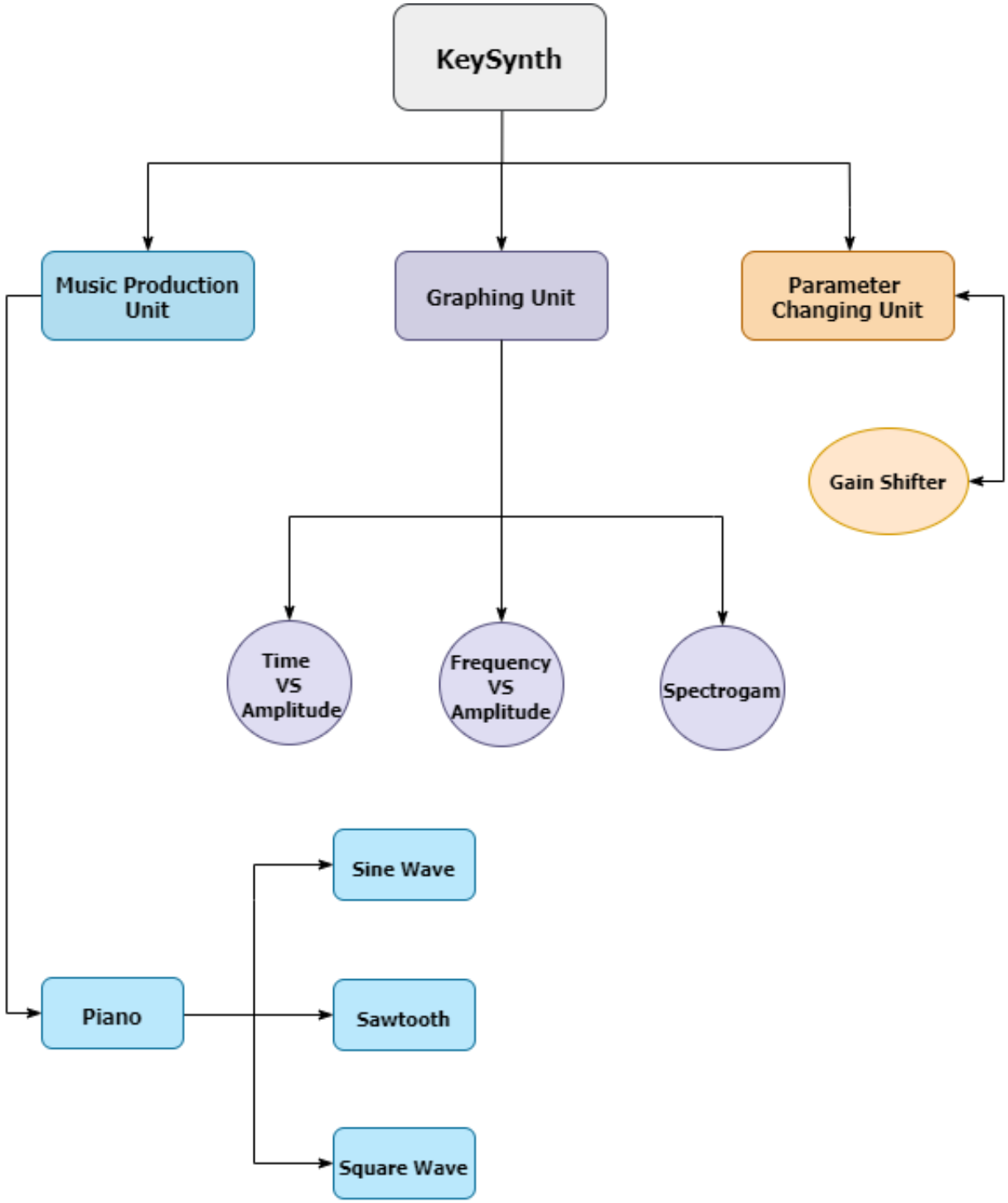


Figure 1. Overview of the project.

3. Working Principle: The working principle of the project is given below**:**

At the start of the program, a GUI opens up, asking for input from the user. Depending on the user's choice of the radio button, it calls a piano_selection_button function that sends a value corresponding to the selected waveform. This value is then used for calling the input_wave() function to generate the necessary coefficients for the linear filter, which resides in a while loop. After the function generates the necessary coefficient values for the linear filter, the program goes into a while loop.

On the other hand, the user is expected to press any of the selected keys on the keyboard to generate impulses of corresponding notes. Depending on whichever button/s are pressed (except for the Q button), a function named "my_function" is called that detects which key is pressed. On the other hand, the while loop which was initiated earlier also receives impulses corresponding to the notes played on the keyboard. Along with the linear filter coefficients and the impulses generated by various notes, the values get pushed into the linear filter called *lfilter* to generate the outputs corresponding to the notes played. Then each of the outputs gets summed together to form the total output called y_{total} (from equation 1). A user can also vary the amplitude of an impulse by sliding the Gain Slider. This in terms of changes the overall output scenario. This can be done at any time during the operation, and its effect can be seen on the graph simultaneously.

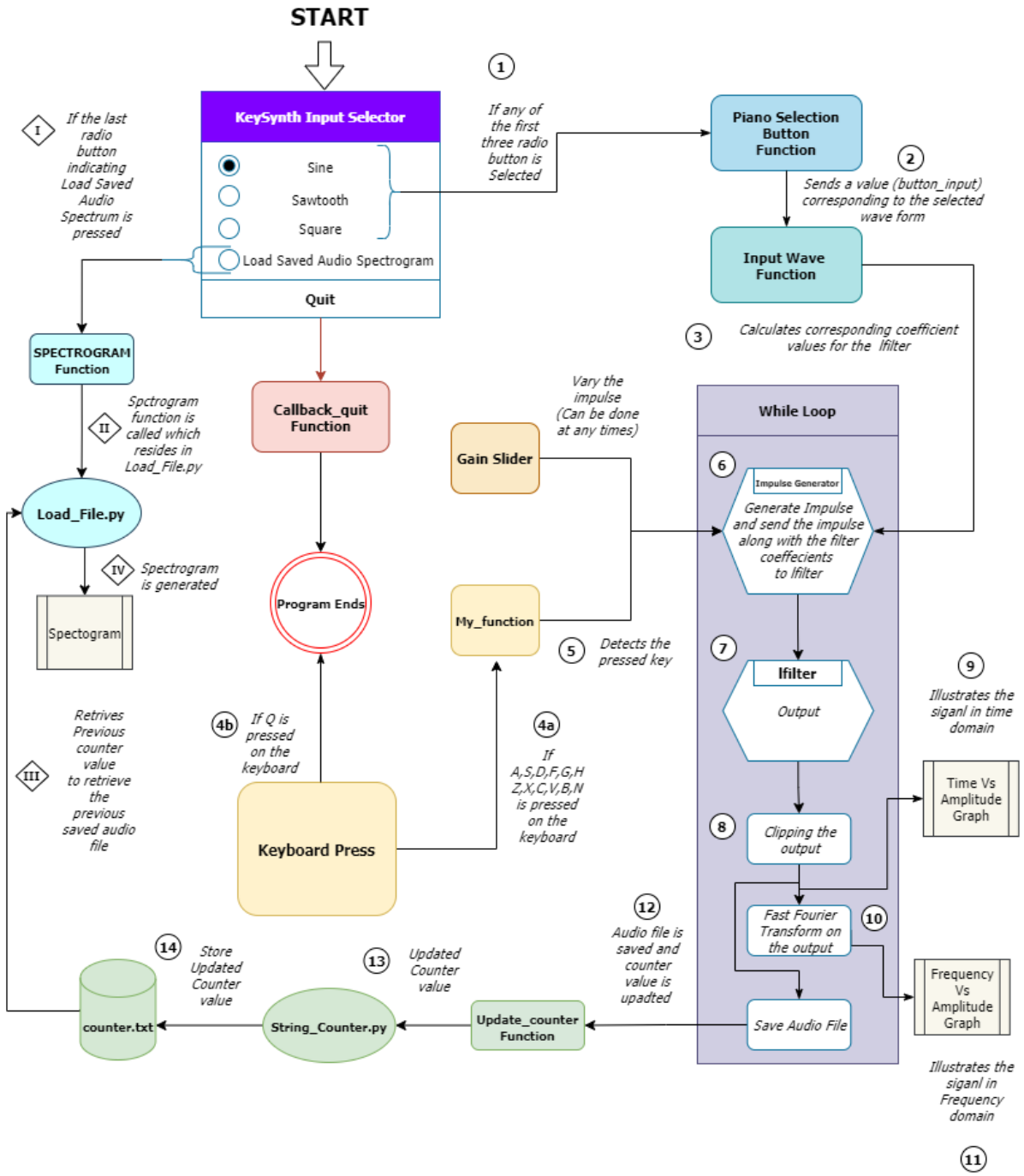
After the program generates the total output (y_{total}) it sends the output value to a clipping function to clip the values out of the selected boundary. And the program then generates the time vs. amplitude graph using the clipped output values. On the clipped output, Fast Fourier Transform is performed to extract the frequencies present in the signal. Then the transformed output signal gets plotted as the frequency vs. amplitude graph.

At the end of the program, the clipped output value is also saved in an audio (.wav) file using the Wave library. After saving the output as an audio file, a counter function is called and updated the counter value, which will be needed to locate the saved file. This is done by calling the string_counter.py program, and the updated counter value is stored in a text file named "counter.txt." All the above-mentioned processes are done if the user chooses any of the first three input radio buttons.

If the radio button labeled as "Load Saved Audio Spectrum" is selected, a "Spectrogram" function will be called, which starts the "Load_File.py" program. It takes the counter value from the "counter.txt" file to retrieve the previously saved audio file, and after that, it generates and displays the Spectrogram of previously saved composition.

The program uses the button "Quit" or the keyboard button "Q" to quit the program. If any of the buttons get pressed, the "callback_quit" function is called, and the program gets terminated.

An overview of how the code project operates is illustrated in the following diagram



KeySynth

Figure 2. Working principle of the project.

4. Results & Future Work: After multiple dedicated trials and vigorous observation, it can be said that the project worked as planned. From the first input wave selection, GUI to keystroke detection, visualization of the graphs, and audio saving file, every aspect of this project worked according to the plan. But as with any other project, there

is always a scope for the betterment, and for this project, this is also true and valid. Therefore, in the future, we would like to add more input waveforms to the synthesizer and also add a piano keyboard GUI to the project, which will certainly enable users to use the synthesizer in a better manner.

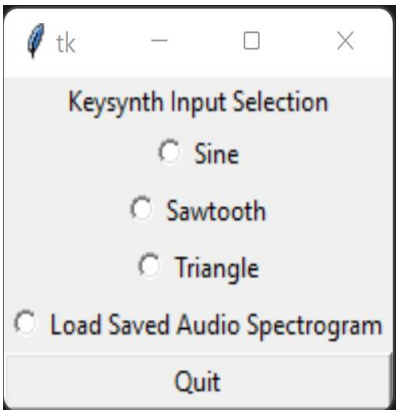


Figure 3. Starting GUI of the Project.

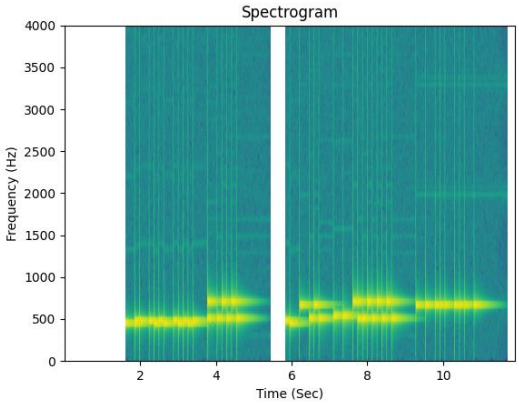


Figure 4. Spectrogram generated by the project.

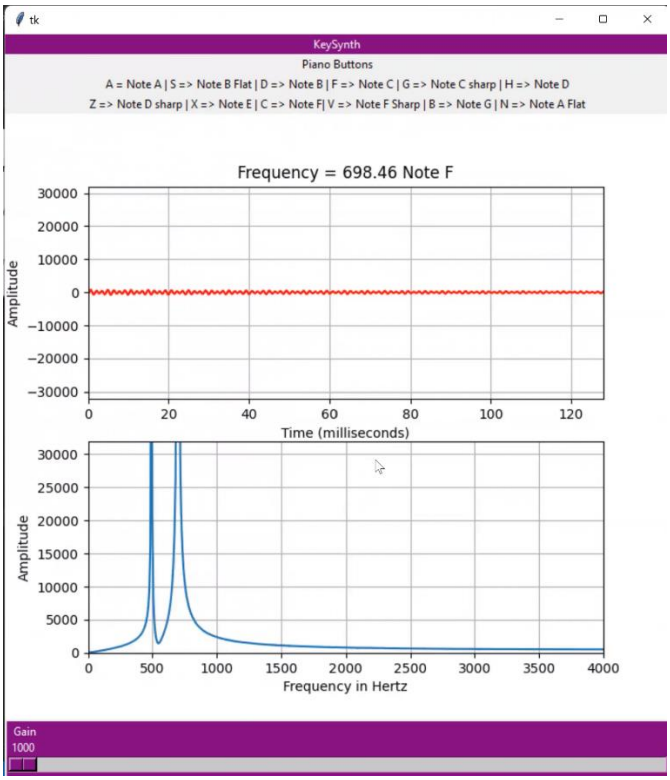


Figure 5. Main GUI of the Project.

5. Conclusion: KeySynth as a synthesizer not only helps the user to develop various musical compositions but also helps to visualize the signals in real-time. Which provides a better understanding for any user.

6. Acknowledgement: This project work would not have been possible without the tremendous efforts provided by the course instructor Professor Ivan Selesnick and his teaching assistants in developing the foundational background knowledge upon which this project stands.

7. References:

1. "Synthesizer." https://en.wikipedia.org/wiki/Synthesizer

2. "Musical Scale." https://ptolemy.berkeley.edu/eecs20/week8/scale.html

3."scipy.signal.lfilter". https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.lfilter.html

4. "wave." https://docs.python.org/3/library/wave.html

5. "Numpy." https://numpy.org/

6. "Matplotlib." https://matplotlib.org/

7. "matplotlib.pyplot.specgram". https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.specgram.html

8. "Tkinter." https://docs.python.org/3/library/tkinter.html