Digital Signal Processing Laboratory
Final Project
EE 384-03

Sean Mitchell

Presented November 16th, 2017

Dates Performed:	November 7th, 2017
	November 9th, 2017
	November 14th, 2017
Instructor:	Fathi Aldukali

Summary

1 Introduction

The main objective of this project was to use MATLAB to create a virtual synthesizer. This program allowed a user to select what note to play and then generated it. The user controlled different aspects of the sound wave generated by the program, changing how it sounded. The user applied various effects to the sound wave to change how it sounded. These effects were achieved using various modulation and filter schemes.

This objective was achieved by breaking the synthesizer into four subsystems. The first subsystem was a method to compute the ADSR Envelope. The second subsystem was to generate random noise on the previously computed ADSR Envelope. The third subsystem was an effects loop, allowing the computed ADSR Envelope to be more precisely manipulated. The final subsystem was a public interface, allowing a user to input what notes, settings, and displayed a plot of the sound wave generated.

2 Theory

Synthesizer (Music) In the context of music, a synthesizer is an electronic musical instrument that generates electric signals that are converted to sound through speakers. Synthesizers can be used to mimic a traditional instrument, such as a piano, or can be used to create their own unique sounds.

Attack Decay Sustain Release (ADSR) Envelope The packet of sound created by a digital synthesizer can be thought of in four distinct phases. The following table describes these four phases.

Term	Definition
$\text{Time}_{\text{atack}}$	The time taken from 0 to the peak value, starting when the key is pressed
$\text{Time}_{\text{decay}}$	The time taken for the subsequent run down from the attack level to the designated sustain level
$\text{Time}_{\text{sustain}}$	The level during the main sequence of the sound's duration, until the key is released
$\text{Time}_{\text{release}}$	The time taken for the level to decay from the sustain level to zero after the key is released

Combining all four phases, it is possible to create a mathematical model to window the soundwave. By multiplying the wave by this window, the wave is manipulated into the desired shape. It is possible to use more than the above four phases to create a model that can control the soundwave more precisely, but this project only used a four phase ADSR envelope.

Audio Effects An audio effect is a mathematical operation performed on a soundwave to change the way it sounds. This can be achieved with various methods, including modulation and filtering. In general, effects can be categorized into six categories based on how the effects are computed and processed (Udo Zolzer).

Table 1: Categories of Audio Effects

Categories	Example
Time-Invariant Filters	LP/HP/BP filter, Equalizer
Time-Varying Filters	Phaser
Delays	Vibrato, Flanger, Chorus, Echo
Modulators	Tremolo, Vibrato
Non-Linear	Compression, Limiters, Distortion
Spacial	Panning, Reverb

A440 Pitch Standard The following tables contain the frequencies in Hz of musical pitches. Each row represents an octave and each column represents an 8-octave range for a given note. Each octave is a 2^n increase in value. The frequencies for notes used in this program came from this table.

Table 2: Standard Frequencies for Musical Notes

	C	**C#**	**D**	**Eb**	**E**	**F**	**F#**	**G**	**G#**	**A**	**Bb**	**B**
0	16.35	17.32	18.35	19.45	20.60	21.83	23.12	24.50	25.96	27.50	29.14	30.87
1	32.70	34.65	36.71	38.89	41.20	43.65	46.25	49.00	51.91	55.00	58.27	61.74
2	65.41	69.30	73.42	77.78	82.41	87.31	92.50	98.00	103.8	110.0	116.5	123.5
3	130.8	138.6	146.8	155.6	164.8	174.6	185.0	185.0	207.7	220.0	233.1	246.9
4	261.6	277.2	293.7	311.1	329.6	349.2	370.0	392.0	415.3	440.0	466.2	493.9
5	493.9	554.4	587.3	622.3	659.3	698.5	740.0	784.0	830.6	880.0	932.3	987.8
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6645	7040	7459	7902

3 Operation and Functionality

3.1 ADSR Envelope

The digital synthesizer can be thought of as the combination of four distinct subsystems. The first subsystem is the function to compute the ADSR envelope. This function took in the frequency of the note, the time of attack, time of decay, time of sustain, and time of release as parameters. Using these parameters the window function was computed. Equation 1 is the piecewise function that MATLAB computes to create the window function, $f_{window}(t)$.

$$f_{window}(t) = \begin{cases} \exp(t - t_a) & \text{if } t \leq t_a \\ \exp(t_a - t) & \text{if } t > t_a \text{ and } t \leq t_a + t_d \\ \frac{1}{\exp(t_d)} & \text{if } t > t_a + t_d \text{ and } t \leq (t_a + t_d + t_s) \\ \frac{\exp((t_d + t_a + t_s) - t)}{\exp(t_d)} & \text{if } t > (t_a + t_d + t_s) \end{cases} \tag{1}$$

When computing Equation 1, MATLAB computed all three functions for the entire length of t, and zeroed the functions where they did not exist, according to the piecewise function. The transition regions between two equations were normalized to ensure a smooth, continuous function. Each of these four functions were computed as separate functions and then summed up into a single function.

Next, the function computed the following sine wave, based on the frequency of the musical note passed as a parameter. Equation 2 was the formula used to compute the sound wave.

$$f_{soundwave}(t) = sin(2\pi f_{note} t) \tag{2}$$

After equation 2 had been computed, any selected effects could be mixed. To achieve this, the sound wave would be passed to the selected effect functions. These functions computed the effect, mixed it with the sound wave, and returned the mixed sound wave.

After equations 1 and 2 had been computed, and the selected effects were applied, the function began creating the final sound wave. This process involved multiplying each of the four equations computed from equation 1 by the mixed sound wave. Next, all four of these functions were added into a single function. Finally, the function was plotted and the function returned a single vector with the complete sound wave.

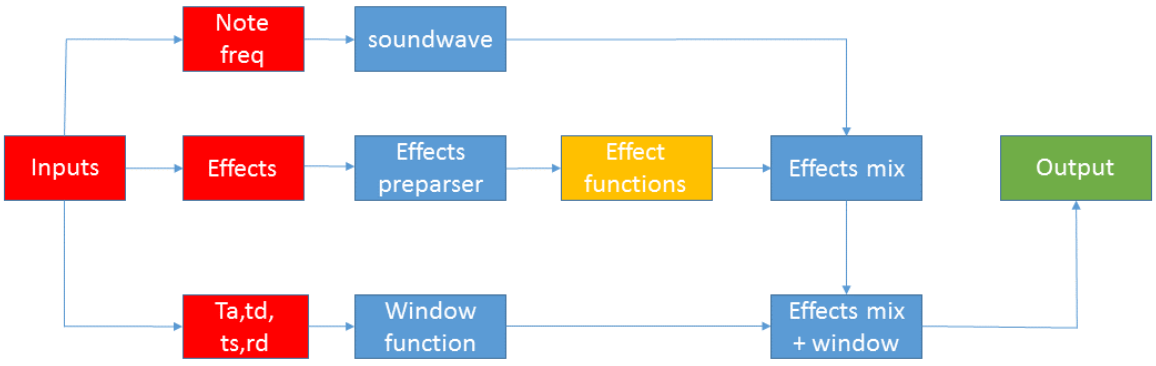


Figure 1: Block diagram of the ADSR envelope generator

3.2 Effect Mixer

The next subsystem was the effects loop, which was briefly mentioned above. This subsystem took input from the GUI, in the form of an id referring to each effect and the integer parameters of that effect. The effect parser then read the effect id, and passed the integer parameters, along with the generated sine wave, to the correct effect function. Each of these functions returned a sine wave mixed with the effect.

The program supported computing between 0 and 3 effects in the signal chain. These effects could be any combination of 3 or less, including the same effect repeated. This was achieved by passing the result of each effect into the next effect. In the case that no effect was active for that part of the signal chain, the effect parser returned the sound wave that was passed to it as a parameter, bypassing any of the computational expensive effect functions.

3.3 GUI

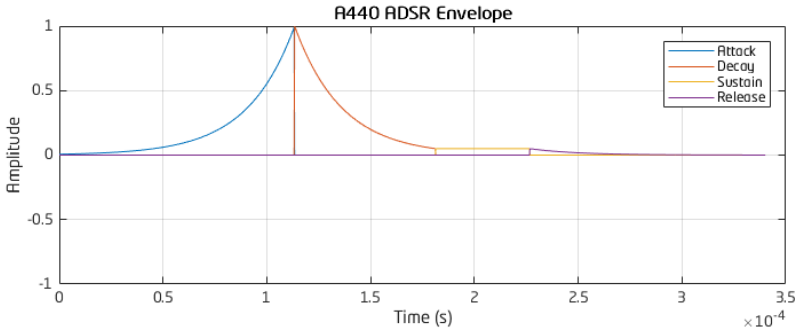
The final subsystem was the graphical user interface. This interface sat on top of the other functions and allowed an end user to effectively use the synthesizer. This interface accomplished four critical goals. The first allowed what note, and octave, the sound wave generated was supposed to be. The second allowed the variables of the ADSR envelope to be modified, shaping the sound wave. The third allowed the end user to select which effects to apply and in what order. The fourth displayed a plot of the sound wave, allowing the end user to see just how they shaped the sound wave.

4 Results

By combining all the subsystems previously discussed, the goal of creating a digital synthesizer was accomplished. The resulting program completed all three of the goals in the design specification. The ADSR envelope completed the design goal of being able to generate a sound wave. The various effect functions allowed for the resulting sound wave to be manipulated. The graphical user interface allowed for an end user to specify what note to generate, what settings to use, and what effects to apply.

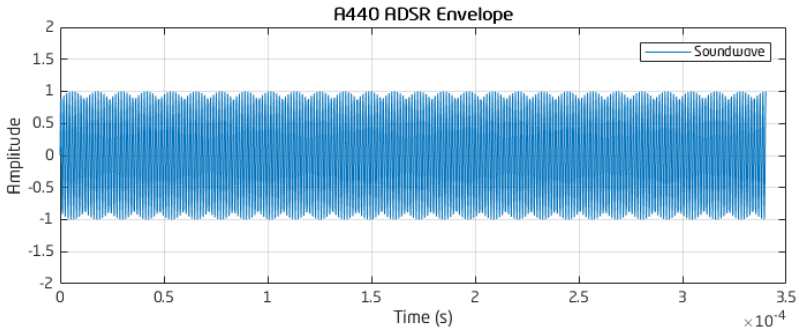
4.1 ADSR Envelope

Using MATLAB, both the window function, equation 1, and the sine wave, equation 2, were computed. After these were computed, they were multiplied together to form the ADSR envelope. The following three plots show the window function, the sound wave, and ADSR envelope. These plots show how the window function modified the sound wave. The tables included with each plot list the values of each of the variables used in that plot.



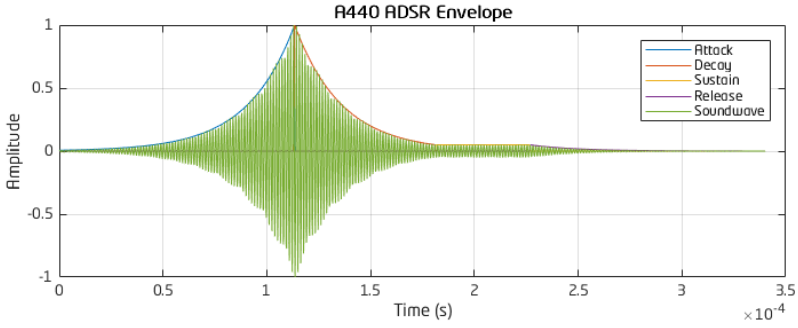
Variable	Value
$Time_{atack}$	5
$Time_{decay}$	3
$Time_{sustain}$	2
$Time_{release}$	5

Figure 2: The window function



Variable	Value
Frequency	16.35 Hz

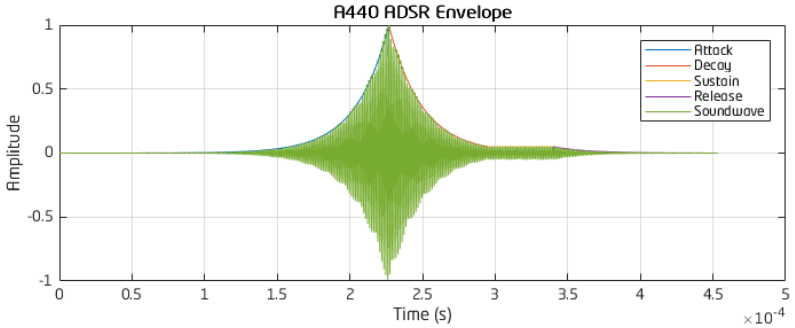
Figure 3: The sinewave of a C note (16.35 Hz)



Variable	Value
$Time_{atack}$	5
$Time_{decay}$	3
$Time_{sustain}$	2
$Time_{release}$	5
Frequency	16.35 Hz

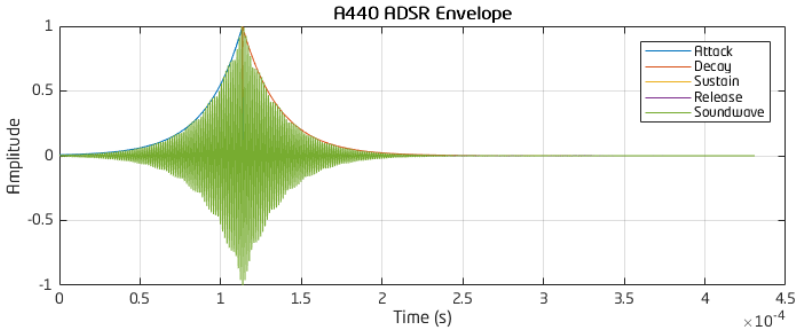
Figure 4: The sinewave of a C note (16.35 Hz) after the window function has been applied

The following six plots show how changing the five variables affects the shape of the ADSR envelope. The tables included with each plot list the values of each of the variables used in that plot.



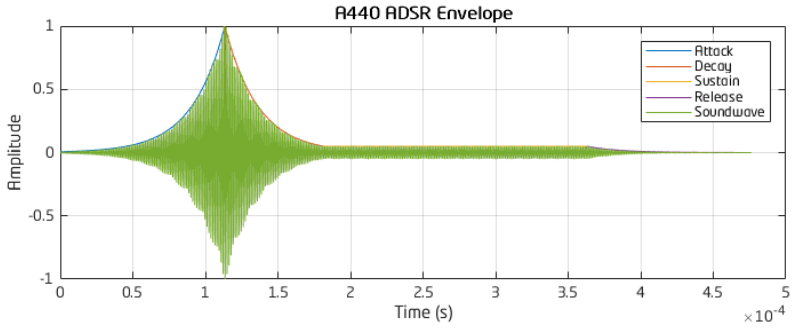
Variable	Value
$\text{Time}_{\text{atack}}$	10
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz

Figure 5: Changing the time of attack



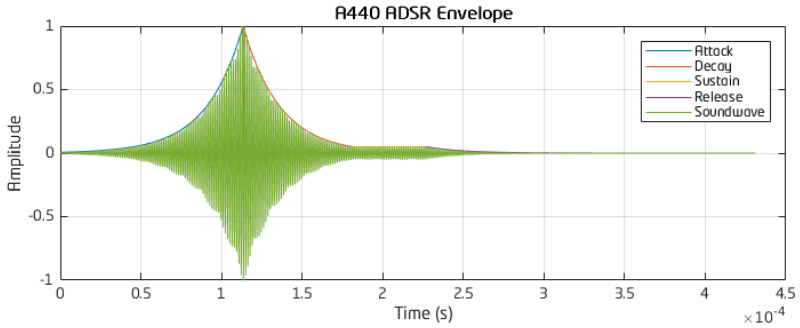
Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	7
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz

Figure 6: Changing the time of decay



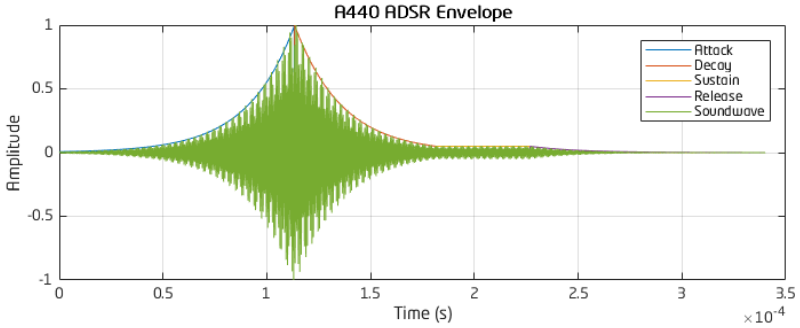
Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	8
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz

Figure 7: Changing the time of sustain



Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	9
Frequency	16.35 Hz

Figure 8: Changing the time of release



Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	30.87 Hz

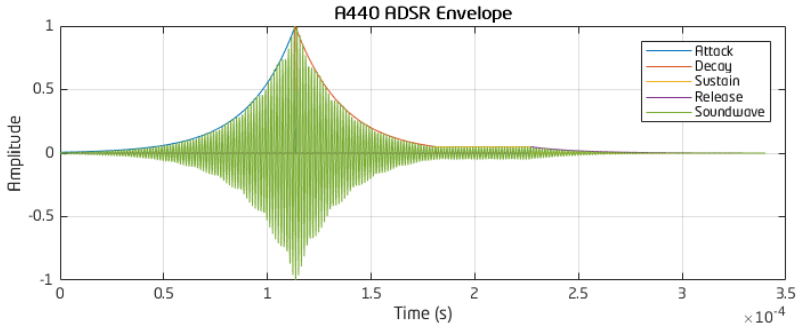
Figure 9: Changing the time of sustain

4.2 Effect: Overdrive

Overdrive is an effect where audio at a low input level is driven by higher input levels in a non-linear curve characteristic. To achieve this, the effect applies symmetrical soft clipping to the sound wave. This means that low input values are multiplied much more than high input values. The following equation is the Schetzen Formula, which is a non-linear soft saturation scheme (Udo Zolzer).

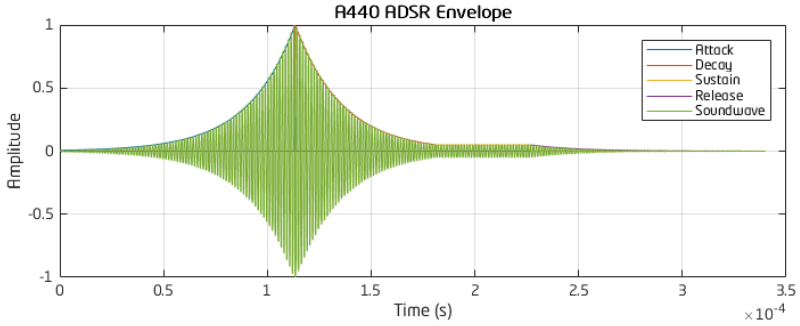
$$f_{overdrive}(t) = \begin{cases} 2x & \text{for } 0 \leq t < 1/3 \\ \frac{3-(2-3x)^2}{3} & \text{for } 1/3 \leq t < 2/3 \\ 1 & \text{for } 2/3 \leq t < 1 \end{cases} \tag{3}$$

The following plots show the original sound wave versus the sound wave with the overdrive effect applied. The values in the following tables are the parameters used to generate each sound wave.



Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
Effect	N/A

Figure 10: The original soundwave



Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
Effect	Overdrive

Figure 11: The soundwave after the Overdrive effect had been applied

4.3 Effect: Fuzz

Distortion is a similar effect to an overdrive effect. The main difference is that distortion has a wider tonal area. A fuzz effect can be thought of as a harsher form of distortion. The following function is a common non-linear function used to model distortion and fuzz (Udo Zolzer).

$$f(t) = \frac{t}{|t|}(1 - \exp{(\frac{1 - \alpha t}{|t|})}) \tag{4}$$

Where α is the gain. The gain controls the level of distortion or fuzz. The result of this equation is often a mix with the original signal, t, for the final output. The following plots show the original sound wave versus the sound wave with the fuzz effect applied. The values in the following tables are the parameters used to generate each sound wave.

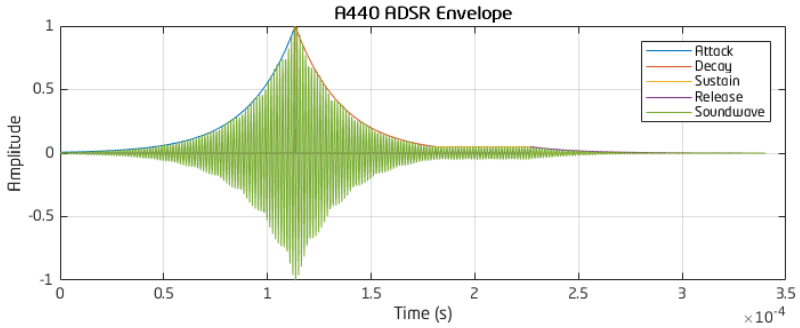


Figure 12: The original soundwave

Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
Effect	N/A
Effect mod	0

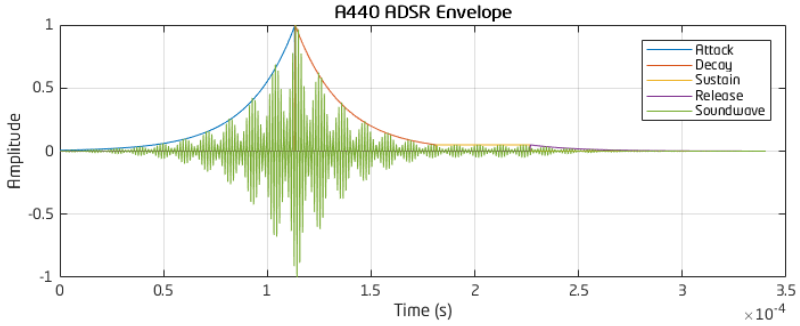


Figure 13: The soundwave after the fuzz effect has been applied

Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
Effect	Fuzz
Effect mod	11

4.4 Effect: Tremolo

A tremolo effect can be generated using amplitude modulation (AM). By modulating the sound wave with another low frequency sine wave, it is possible to create a change of pitch in the sound wave. The follow three plots show the original sound wave and the effects of applying tremolo with modulation frequencies.

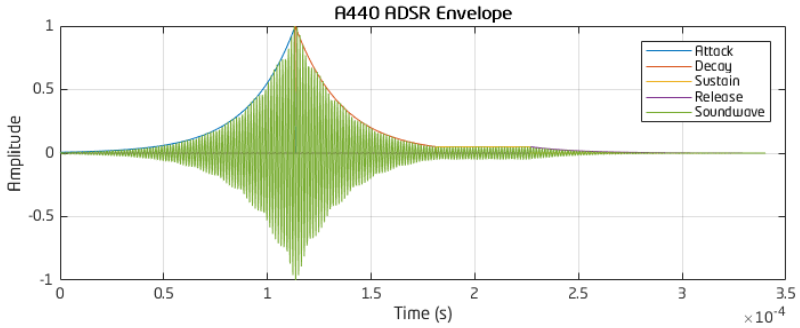


Figure 14: The original soundwave

Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
Effect	N/A
Effect mod	0

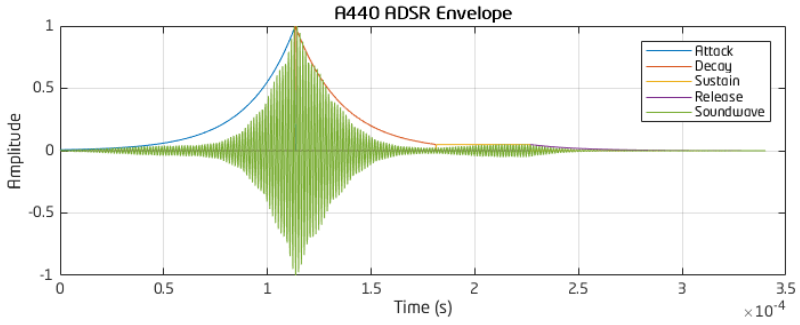


Figure 15: The soundwave after the tremolo effect has been applied

Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
Effect	Tremolo
Effect mod	20 Hz

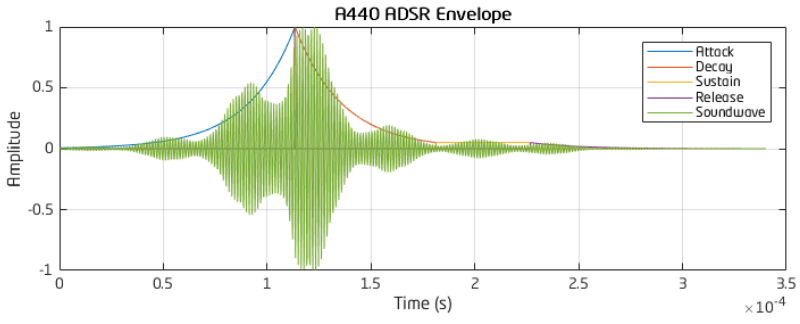


Figure 16: The soundwave after the tremolo effect has been applied

Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
Effect	Tremolo
Effect mod	50 Hz

4.5 Effect: Compressor

A compressor can be used to reduce the dynamics of the input signal. This means that the quiet parts of the input signal are not modified and loud parts are reduced according to a static curve. This results in the difference between the quiet and loud parts of the signal being reduced, which means the overall signal level is boosted. This is achieved by applying a low pass filter to the sound wave. The compressor uses the same code as the expander, but with a negative value of a, instead of a positive value. The variable a is a filter parameter and controls the shape of the low pass filter.

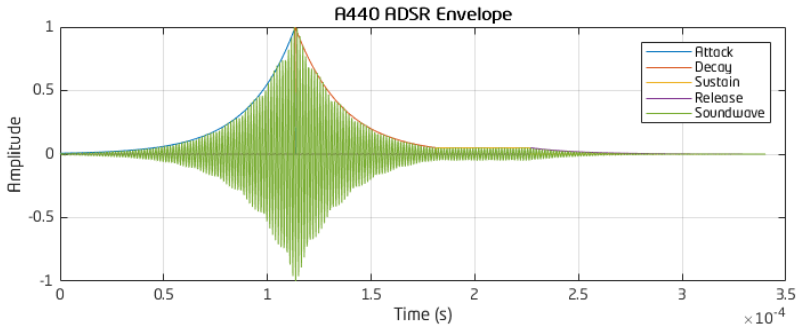


Figure 17: The original soundwave

Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
Effect	N/A
Effect mod	0

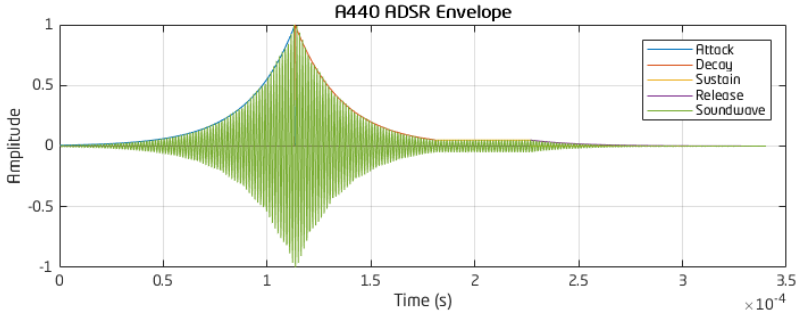


Figure 18: The soundwave after the compressor effect has been applied

Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
Effect	Compressor
Effect mod	-0.5

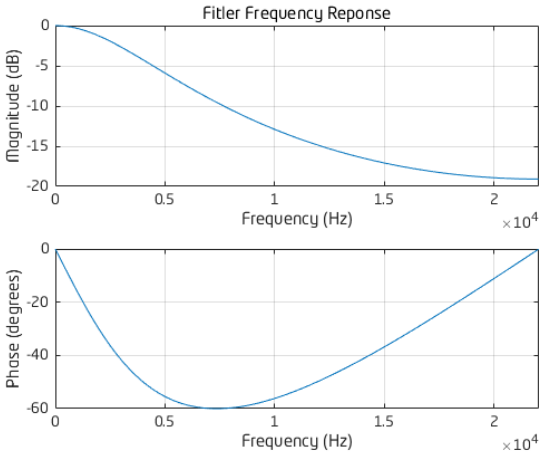


Figure 19: Frequency and phase response of the low pass filter

4.6 Effect: Expander

An expander also can be used to reduce the dynamics of the input signal. Unlike compressors, expanders operate on the quiet parts of the signal. An expander increases the dynamics of these quiet parts. The result is a brighter sound. This is achieved by applying a low pass filter to the sound wave. The expander uses the same code as the compressor, but with a positive value of a, instead of a negative value. The variable a is a filter parameter and controls the shape of the low pass filter.

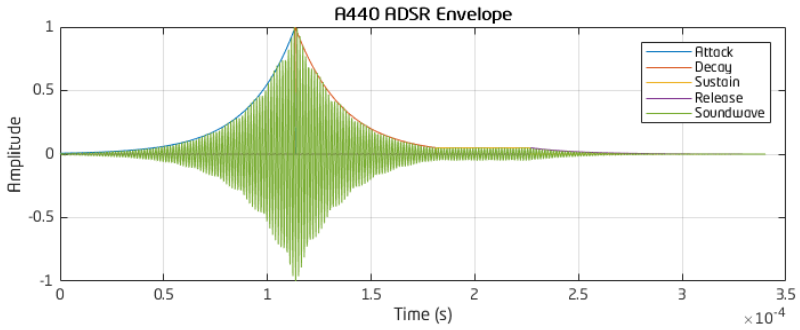


Figure 20: The original soundwave

Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
Effect	N/A
Effect mod	0

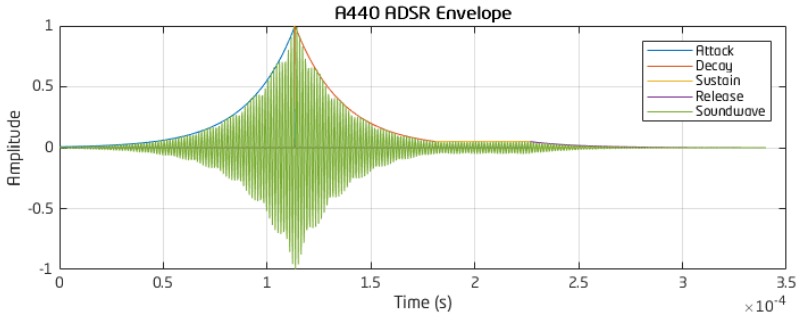


Figure 21: The soundwave after the expander effect has been applied

Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
Effect	Expander
Effect mod	0.5

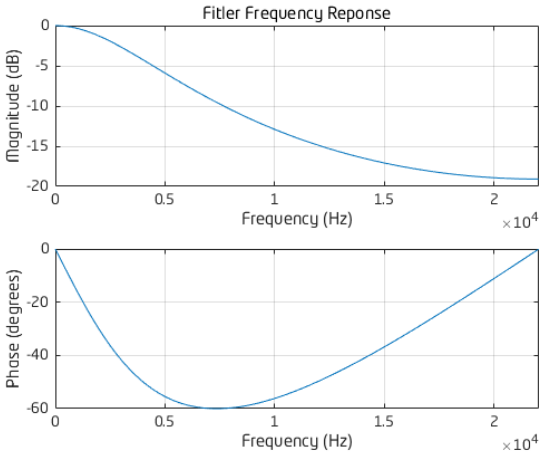


Figure 22: Frequency and phase response of the low pass filter

4.7 Signal Chain

The plots in this subsection are various combinations of effects, comparing how changing the signal chain effects the sound wave.

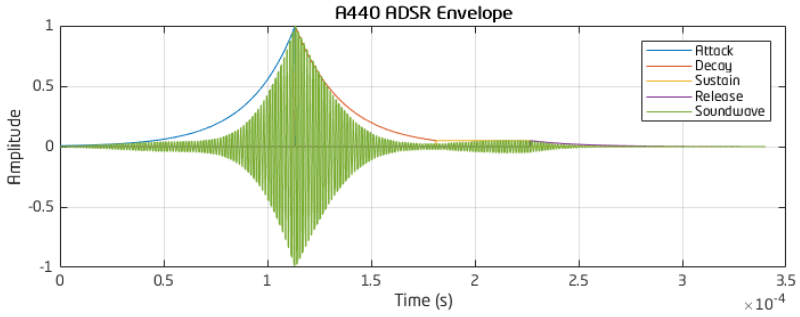


Figure 23: Signal chain: fuzz, overdrive, tremolo

Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
Fuzz Mod	11
OD Mod	N/A
Trem Mod	20

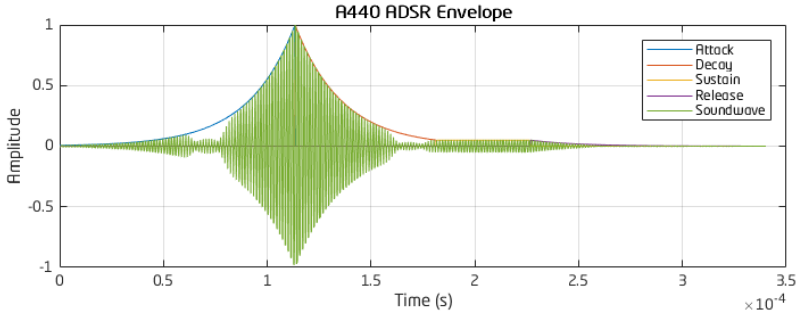


Figure 24: Signal chain: tremolo, overdrive, fuzz

Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
Trem Mod	20
OD Mod	N/A
Fuzz Mod	11

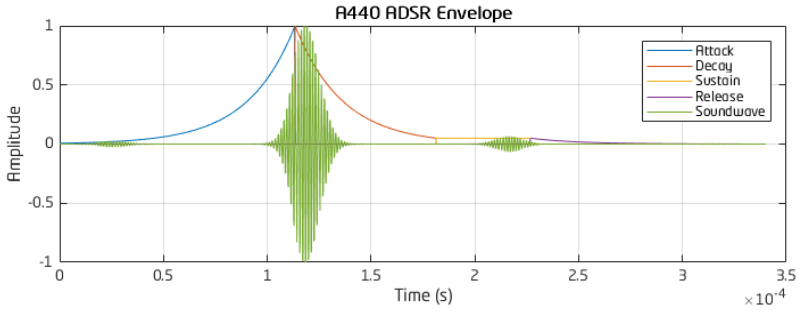


Figure 25: Signal chain: overdrive, tremolo, fuzz

Variable	Value
$\text{Time}_{\text{atack}}$	5
$\text{Time}_{\text{decay}}$	3
$\text{Time}_{\text{sustain}}$	2
$\text{Time}_{\text{release}}$	5
Frequency	16.35 Hz
OD Mod	N/A
Trem Mod	20
Fuzz Mod	11

4.8 Graphic User Interface

The graphical user interface (GUI) of the program served two main purposes. The first was to allow the end user to control the settings of the sound wave that was to be generated. The second was to display a plot of the sound wave that was generated. The following screenshots show the GUI in operation.

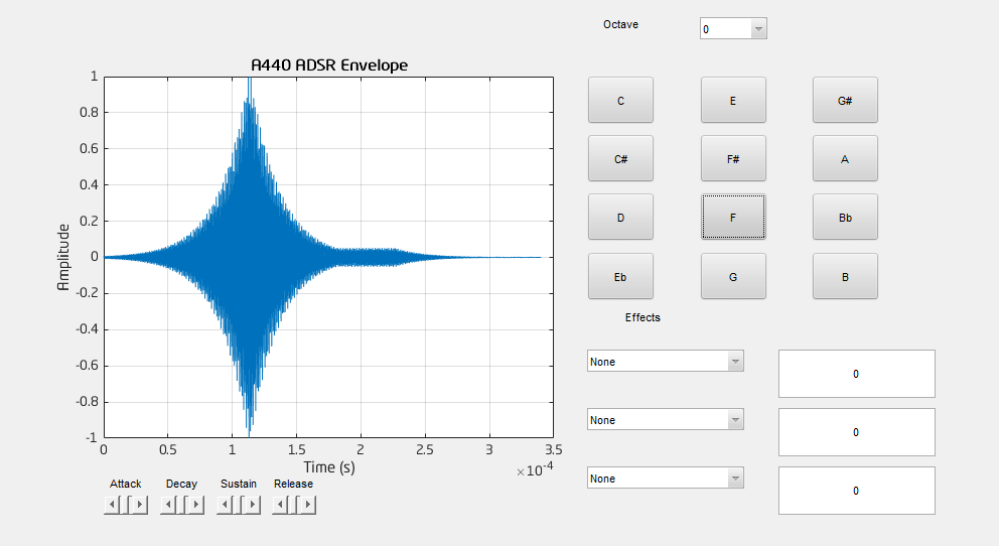


Figure 26: Default GUI

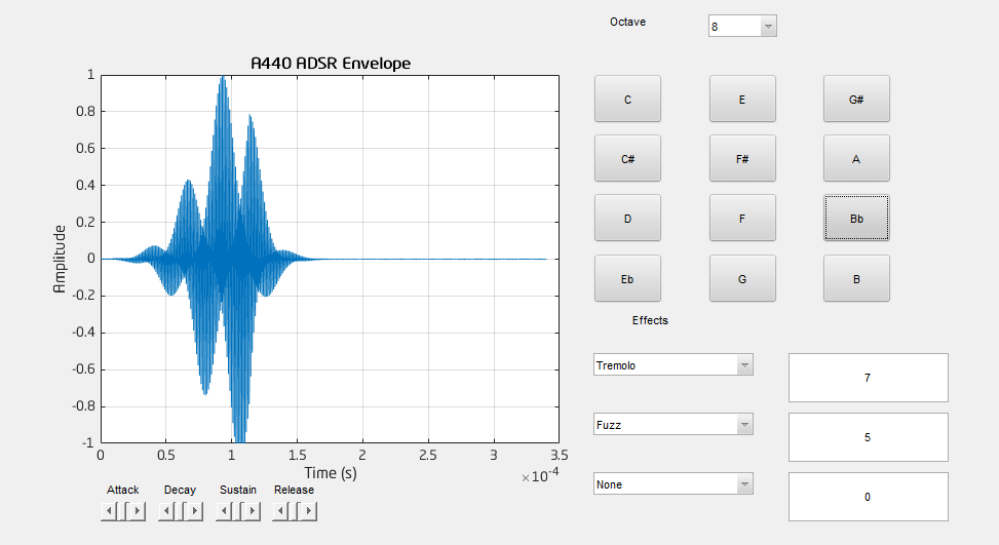


Figure 27: GUI with settings applied

5 Conclusion

In conclusion, this project explored signal generation, audio generation, and signal manipulation in MATLAB. Time was spent creating functions that would allow an end user to shape and control a waveform. Using MATLAB, modulation functions were created allowing a user to apply effects to a sound wave. MATLAB's built-in filter command was also used to create audio effects. Time was also spent using MATLAB's plot commands and user interface functions to create a GUI that would allow an end user to both control the sound wave and visualize the plot of the sound wave.

Combining all these concepts, the result was a digital synthesizer made in MATLAB. This synthesizer could generate a musical note of a given pitch. Through its GUI, a user could control the settings of the sound wave. These settings included the octave and pitch of the note, the shape of the waveform, and applying audio effects to the resulting sound wave. The GUI also displayed a plot of the generated sound wave, allowing the end users to see the impact of their choice of settings.

6 Appendix

6.1 References

[1] Udo Zolzer. *DAFX - Digital Audio Effects*. John Wiley & Sons, 2002. Print

6.2 MATLAB Code

The following code was the main script used in the laboratory.

```
% Sean Mitchell
% EE-384 Final Project
% MATLAB Synthesizer

% gui has to be in a seperate function because of callback subroutines
gui_test();
```

The following code was the function used to compute the ADSR envelope.

```
function [adsr_out] = ADSR_Envelope(note, octave, t_a, t_d, t_s, t_r, effect_1
    , effect_1_mod, effect_2, effect_2_mod, effect_3, effect_3_mod)
t = 0:0.01:(t_a + t_d + t_s + t_r); %samples
time = t/44100; %time

% Base functions
f_t_a = exp(t-t_a);
f_t_d = exp(t_a-t);
f_t_s = (1/exp(t_d)).*ones(size(t));
f_t_r = (1/exp(t_d)).*exp((t_d + t_a + t_s)-t);

f_oct = note * octave;
wave = sin(2*pi*f_oct*t);

% Zero-ing out the windowing functions
f_t_a(t>=t_a) = 0;

f_t_d(t<(t_a)) = 0;
f_t_d(t>=(t_a + t_d)) = 0;

f_t_s(t<t_a+t_d) = 0;
f_t_s(t>=(t_a + t_d + t_s)) = 0;

f_t_r(t<(t_a + t_d + t_s)) = 0;

% Effects mix
wave = Effect_Preparser(effect_1, effect_1_mod, wave);
wave = Effect_Preparser(effect_2, effect_2_mod, wave);
wave = Effect_Preparser(effect_3, effect_3_mod, wave);

% Creating the soundwave functions
f_t1 = wave .* f_t_a;
f_t2 = wave .* f_t_d;
f_t3 = wave .* f_t_s;
f_t4 = wave .* f_t_r;

% Summing the soundwave functions into a single function
f_sum = f_t1 + f_t2 + f_t3 + f_t4;
adsr_out = f_sum/max(f_sum); %normalized output

% Plot of the windowing functions + soundwave
```

```
%figure,
%plot(time,f_t_a,time,f_t_d,time,f_t_s,time,f_t_r,time,adsr_out);
%legend('Attack','Decay','Sustain','Release','Soundwave');
plot(time,adsr_out);
xlabel('Time (s)');
ylabel('Amplitude');
title('A440 ADSR Envelope');
ylim([-1 1]);
grid on
end
```

The following code was the function used to perform the effects parser.

```
function [mix] = Effect_Preparser(id,mod,soundwave)
    if (id == 1)
        mix = soundwave;
    elseif (id == 2)
        mix = tremolo(soundwave,mod);
    elseif (id == 3)
        mix = fuzz(soundwave,mod);
    elseif (id == 4)
        mix = Overdrive(soundwave);
    elseif (id == 5)
        mix = Compressor(soundwave,mod);
    end
end
```

The following code was the function used to create the GUI.

```
function [] = gui_test()
% Default Values
t_a = 5;          % duration of attack phase
t_d = 3;        % duration of delay phase
t_s = 2;          % duration of sustain phase
t_r = 5;          % duration of release phase
effect_1 = 1;
effect_1_mod = 0;

effect_2 = 1;
effect_2_mod = 0;

effect_3 = 1;
effect_3_mod = 0;
octave = 2^0;


% Note Frequencies (Hz)
C = 16.35;
C_Sharp = 17.32;
D = 18.35;
E_Flat = 19.45;

E = 20.60;
F = 21.83;
F_Sharp = 23.12;
G = 24.50;

G_Sharp = 25.96;
A = 27.50;
B_Flat = 29.14;
B = 30.87;

%  Create and then hide the UI as it is being constructed.
f = figure('Visible','off','Position',[360,500,450,285]);

```

```
% Note buttons
button_c = uicontrol('Style','pushbutton','String','C','Position'
    ,[215,220,30,25],'Callback',@C_Button_Callback);
button_c_sharp = uicontrol('Style','pushbutton','String','C#','Position'
    ,[315,190,30,25],'Callback',@C_Sharp_Button_Callback);
button_d = uicontrol('Style','pushbutton','String','D','Position'
    ,[315,160,30,25],'Callback',@D_Button_Callback);
button_e_flat = uicontrol('Style','pushbutton','String','Eb','Position'
    ,[315,130,30,25],'Callback',@E_Flat_Button_Callback);

button_e = uicontrol('Style','pushbutton','String','E','Position'
    ,[265,220,30,25],'Callback',@E_Button_Callback);
button_f_sharp = uicontrol('Style','pushbutton','String','F#','Position'
    ,[365,190,30,25],'Callback',@F_Sharp_Button_Callback);
button_f = uicontrol('Style','pushbutton','String','F','Position'
    ,[365,160,30,25],'Callback',@F_Button_Callback);
button_g = uicontrol('Style','pushbutton','String','G','Position'
    ,[365,130,30,25],'Callback',@G_Button_Callback);

button_g_sharp = uicontrol('Style','pushbutton','String','G#','Position'
    ,[315,220,30,25],'Callback',@G_Sharp_Button_Callback);
button_a = uicontrol('Style','pushbutton','String','A','Position'
    ,[415,190,30,25],'Callback',@A_Button_Callback);
button_b_flat = uicontrol('Style','pushbutton','String','Bb','Position'
    ,[415,160,30,25],'Callback',@B_Flat_Button_Callback);
button_b = uicontrol('Style','pushbutton','String','B','Position'
    ,[415,130,30,25],'Callback',@B_Button_Callback);

% Effects selectors
eff_label = uicontrol('style','text','String','Effects','Position',
    [265,100,50,25]);
effect_select_1 = uicontrol('Style', 'popup','String', {'None','Tremolo','Fuzz
    ','Overdrive','Compressor'},'Position', [265,80,70,25],'Callback',@
    Effect1_Callback);
effect_select_2 = uicontrol('Style', 'popup','String', {'None','Tremolo','Fuzz
    ','Overdrive','Compressor'},'Position', [265,50,70,25],'Callback',@
    Effect2_Callback);
effect_select_3 = uicontrol('Style', 'popup','String', {'None','Tremolo','Fuzz
    ','Overdrive','Compressor'},'Position', [265,20,70,25],'Callback',@
    Effect3_Callback);

% Effect mod selectors
effect1_mod_button = uicontrol('Style','edit','String','0','Position'
    ,[350,80,70,25],'Callback',@Effect1_Mod_Callback);
effect2_mod_button = uicontrol('Style','edit','String','0','Position'
    ,[350,50,70,25],'Callback',@Effect2_Mod_Callback);
effect3_mod_button = uicontrol('Style','edit','String','0','Position'
    ,[350,20,70,25],'Callback',@Effect3_Mod_Callback);

% Octave selector
oct_label = uicontrol('style','text','String','Octave','Position',
    [265,250,50,25]);
octave_select = uicontrol('Style', 'popup','String', {0,1,2,3,4,5,6,7,8},'
    Position', [315,250,30,25],'Callback',@Octave_Callback);
```

```
67
68 % Attack slider
69 slider_attack = uicontrol('Style', 'slider','Min',0,'Max',20,'Value',5,'
       Position', [50 20 20 10],'Callback',@Attack_Slider_Callback);
70 text_attack = uicontrol('Style','text','Position',[50 30 20 10],'String','
       Attack');
71
72 % Decay slider
73 slider_decay = uicontrol('Style', 'slider','Min',0,'Max',20,'Value',5,'
       Position', [75 20 20 10],'Callback', @Decay_Slider_Callback);
74 text_decay = uicontrol('Style','text','Position',[75 30 20 10],'String','Decay
       ');
75
76 % Sustain slider
77 slider_sustain = uicontrol('Style', 'slider','Min',0,'Max',20,'Value',5,'
       Position', [100 20 20 10],'Callback', @Sustain_Slider_Callback);
78 text_sustain = uicontrol('Style','text','Position',[100 30 20 10],'String','
       Sustain');
79
80 % Release slider
81 slider_release = uicontrol('Style', 'slider','Min',0,'Max',20,'Value',5,'
       Position', [125 20 20 10],'Callback', @Release_Slider_Callback);
82 text_release = uicontrol('Style','text','Position',[125 30 20 10],'String','
       Release');
83
84 ha = axes('Units','pixels','Position',[50,60,200,185]);
85 align([button_c,button_c_sharp,button_d,button_e_flat,oct_label],'Center','
       None');
86 align([button_e,button_f_sharp,button_f,button_g],'Center','None');
87 align([button_g_sharp,button_a,button_b_flat,button_b],'Center','None');
88
89 % Initialize the UI.
90 % Change units to normalized so components resize automatically.
91 f.Units = 'normalized';
92 ha.Units = 'normalized';
93 button_c.Units = 'normalized'; button_c_sharp.Units = 'normalized'; button_d.
       Units = 'normalized'; button_e_flat.Units = 'normalized';
94 button_e.Units = 'normalized'; button_f_sharp.Units = 'normalized'; button_f.
       Units = 'normalized'; button_g.Units = 'normalized';
95 button_g_sharp.Units = 'normalized'; button_a.Units = 'normalized';
       button_b_flat.Units = 'normalized'; button_b.Units = 'normalized';
96 slider_attack.Units = 'normalized';slider_decay.Units = 'normalized';
       slider_sustain.Units = 'normalized'; slider_release.Units = 'normalized';
97 text_attack.Units = 'normalized'; text_decay.Units = 'normalized';
       text_sustain.Units = 'normalized'; text_release.Units = 'normalized';
98 effect_select_1.Units = 'normalized'; effect_select_2.Units = 'normalized';
       effect_select_3.Units = 'normalized';eff_label.Units = 'normalized';
99 octave_select.Units = 'normalized';oct_label.Units = 'normalized';
100 effect1_mod_button.Units = 'normalized'; effect2_mod_button.Units = '
       normalized'; effect3_mod_button.Units = 'normalized';
101
102 % Assign the a name to appear in the window title.
103 f.Name = 'MATLAB Digital Synthesizer';
104
```

```
105 % Move the window to the center of the screen.
106 movegui(f,'center')
107
108 % Make the window visible.
109 f.Visible = 'on';
110
111 % Push button callbacks
112 function C_Button_Callback(source,eventdata)
113 % Display surf plot of the currently selected data.
114     f_t = ADSR_Envelope(C, octave, t_a, t_d, t_s, t_r, effect_1, effect_1_mod,
          effect_2, effect_2_mod, effect_3, effect_3_mod);
115     soundsc(f_t);
116 end
117
118 function C_Sharp_Button_Callback(source,eventdata)
119
120     f_t = ADSR_Envelope(C_Sharp, octave, t_a, t_d, t_s, t_r, effect_1,
          effect_1_mod, effect_2, effect_2_mod, effect_3, effect_3_mod);
121     soundsc(f_t);
122 end
123
124 function D_Button_Callback(source,eventdata)
125 % Display mesh plot of the currently selected data.
126    f_t = ADSR_Envelope(D, octave, t_a, t_d, t_s, t_r, effect_1, effect_1_mod,
         effect_2, effect_2_mod, effect_3, effect_3_mod);
127    soundsc(f_t);
128 end
129
130 function F_Button_Callback(source,eventdata)
131 % Display contour plot of the currently selected data.
132    f_t = ADSR_Envelope(F, octave, t_a, t_d, t_s, t_r, effect_1, effect_1_mod,
         effect_2, effect_2_mod, effect_3, effect_3_mod);
133    soundsc(f_t);
134 end
135
136 function F_Sharp_Button_Callback(source,eventdata)
137     f_t = ADSR_Envelope(F_Sharp, octave, t_a, t_d, t_s, t_r, effect_1,
          effect_1_mod, effect_2, effect_2_mod, effect_3, effect_3_mod);
138     soundsc(f_t);
139 end
140
141 function G_Button_Callback(source,eventdata)
142 % Display mesh plot of the currently selected data.
143    f_t = ADSR_Envelope(G, octave, t_a, t_d, t_s, t_r, effect_1, effect_1_mod,
         effect_2, effect_2_mod, effect_3, effect_3_mod);
144    soundsc(f_t);
145 end
146
147 function G_Sharp_Button_Callback(source,eventdata)
148     f_t = ADSR_Envelope(G_Sharp, octave, t_a, t_d, t_s, t_r, effect_1,
          effect_1_mod, effect_2, effect_2_mod, effect_3, effect_3_mod);
149    soundsc(f_t);
150 end
151
```

```
function E_Button_Callback(source,eventdata)
% Display contour plot of the currently selected data.
    f_t = ADSR_Envelope(E, octave, t_a, t_d, t_s, t_r, effect_1, effect_1_mod,
        effect_2, effect_2_mod, effect_3, effect_3_mod);
    soundsc(f_t);
end

function E_Flat_Button_Callback(source,eventdata)
    f_t = ADSR_Envelope(E_Flat, octave, t_a, t_d, t_s, t_r, effect_1,
        effect_1_mod, effect_2, effect_2_mod, effect_3, effect_3_mod);
    soundsc(f_t);
end

function A_Button_Callback(source,eventdata)
% Display contour plot of the currently selected data.
    f_t = ADSR_Envelope(A, octave, t_a, t_d, t_s, t_r, effect_1, effect_1_mod,
        effect_2, effect_2_mod, effect_3, effect_3_mod);
    soundsc(f_t);
end

function B_Button_Callback(source,eventdata)
% Display contour plot of the currently selected data.
    f_t = ADSR_Envelope(B, octave, t_a, t_d, t_s, t_r, effect_1, effect_1_mod,
        effect_2, effect_2_mod, effect_3, effect_3_mod);
    soundsc(f_t);
end

function B_Flat_Button_Callback(source,eventdata)
    f_t = ADSR_Envelope(B_Flat, octave, t_a, t_d, t_s, t_r, effect_1,
        effect_1_mod, effect_2, effect_2_mod, effect_3, effect_3_mod);
    soundsc(f_t);
end

% Slider callbacks
function Attack_Slider_Callback(source,event)
    t_a = source.Value;
end

function Decay_Slider_Callback(source,event)
    t_d = source.Value;
end

function Sustain_Slider_Callback(source,event)
    t_s = source.Value;
end

function Release_Slider_Callback(source,event)
    t_r = source.Value;
end

% Effects dropdown menu callbacks
function Effect1_Callback(source,event)
    effect_1 = source.Value;
end
```

```
201
202 function Effect1_Mod_Callback(source,event)
203     effect_1_mod = str2num(get(effect1_mod_button,'String'));
204 end
205
206 function Effect2_Callback(source,event)
207     effect_2 = source.Value;
208 end
209
210 function Effect2_Mod_Callback(source,event)
211     effect_2_mod = str2num(get(effect2_mod_button,'String'));
212 end
213
214 function Effect3_Callback(source,event)
215     effect_3 = source.Value;
216 end
217
218 function Effect3_Mod_Callback(source,event)
219     effect_3_mod = str2num(get(effect3_mod_button,'String'));
220 end
221
222 function Octave_Callback(source,event)
223     octave = 2^(source.Value - 1);
224 end
225
226 end
```

The following code was the function used to create the tremolo effect.

```
function [y_t] = tremolo(f_t,f_x)
fs = 8500;
index = 1:length(f_t);
alpha = 0.5;
trem = (1 + alpha*sin(2*pi*index*(f_x/fs)));
y_t = trem.*f_t;
end
```

The following code was the function used to create the fuzz effect.

```
function [y_t] = fuzz(f_t,gain)
% Distortion based on an exponential function
% x -input
% gain -amount of distortion,
% mix -mix of original and distorted sound
% 1=only distorted
mix = 0.1;
q = f_t./abs(f_t);
y_t = q.*(1-exp(gain*(q.*f_t)));
y_t = mix*y_t+(1-mix)*f_t;
end
```

The following code was the function used to create the overdrive effect.

```
function [y] = Overdrive(f_t)
N=length(f_t);
y=zeros(1,N); % Preallocate y
th=1/3; % threshold for symmetrical soft clipping by Schetzen Formula
for i=1:1:N,
    if abs(f_t(i))< th, y(i)=2*f_t(i);end;
    if abs(f_t(i))>=th,
        if f_t(i)> 0, y(i)=(3-(2-f_t(i)*3).^2)/3; end;
        if f_t(i)< 0, y(i)=-(3-(2-abs(f_t(i))*3).^2)/3; end;
    end;
    if abs(f_t(i))>2*th,
    if f_t(i)> 0, y(i)=1;end;
    if f_t(i)< 0, y(i)=-1;end;
    end;

end
```

The following code was the function used to create the compressor and expander effects.

```
function [y] = Compressor(x,comp)
% Compressor/expander
% comp - compression: 0>comp>-1, expansion: 0<comp<1
% a    - filter parameter <1
fs = 44100;
a = 0.5;
h=filter([(1-a)^2],[1.0000 -2*a a^2],abs(x));
h=h/max(h);
h=h.^comp;
y=x.*h;
y=y*max(abs(x))/max(abs(y));

L = length(x);
n = pow2(nextpow2(L));
y_dft = fft(x,n);
y_s = fftshift(y_dft);
f = (-n/2:n/2-1) * (fs/n);

figure,
freqz([(1-a)^2],[1.0000 -2*a a^2],n,fs)
grid on
title('Fitler Frequency Reponse','FontWeight','Normal')

end
```