BRYAN GUNER ENGINEERING PORTFOLIO

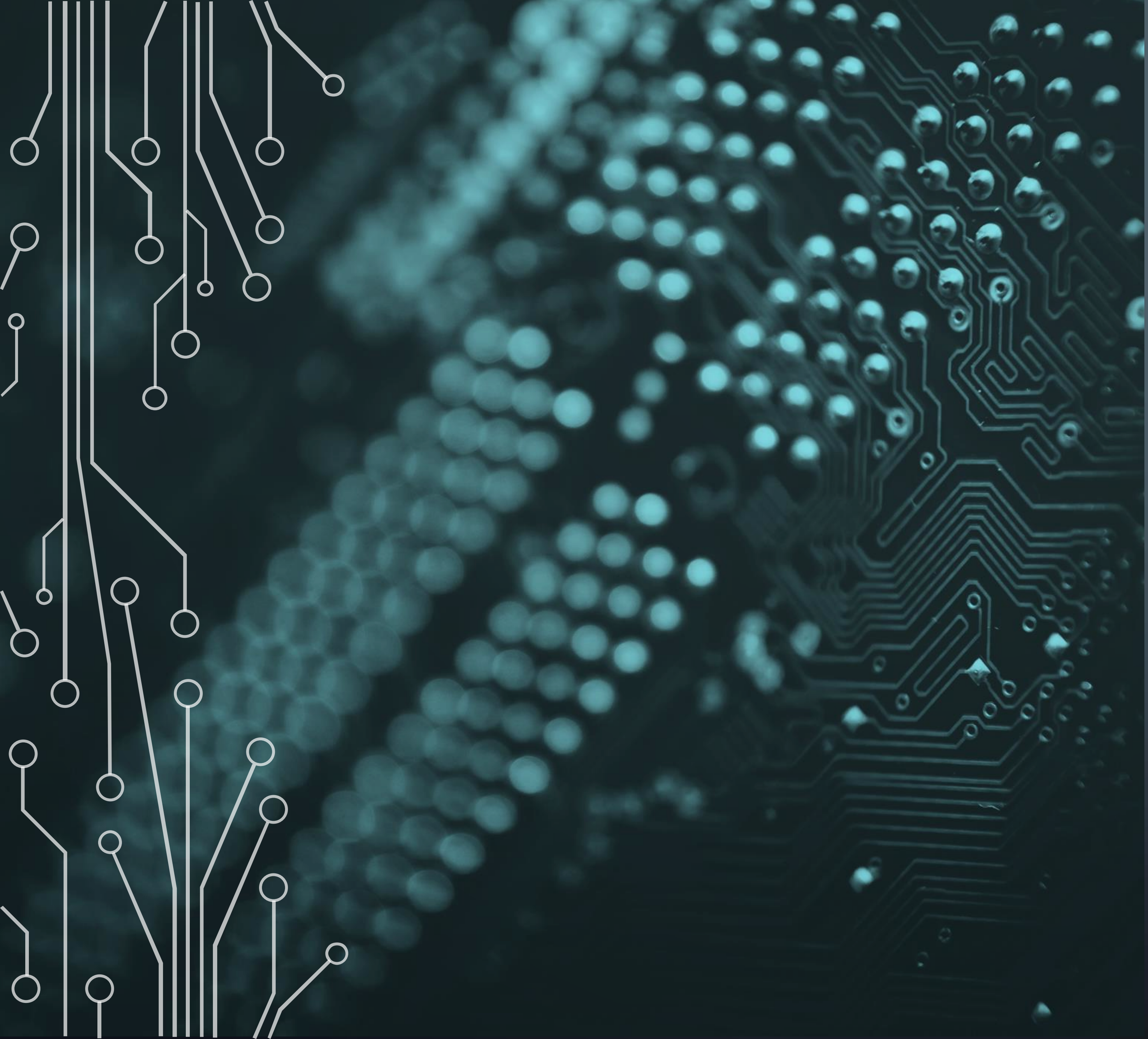


TABLE OF CONTENTS

Contact Information:

Address: 150 Henley Place, Weehawken, NJ, 07086

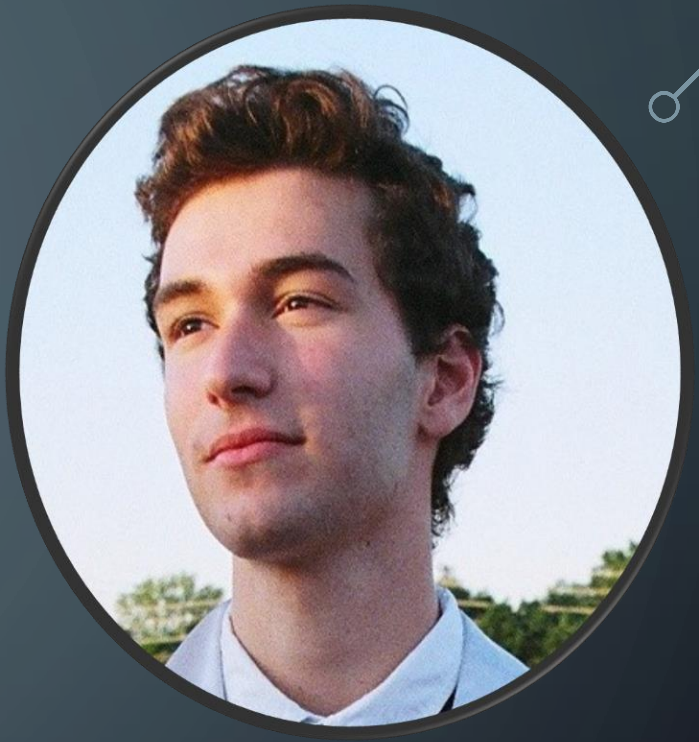
LinkedIn: https://www.linkedin.com/in/bryan-guner-046199128/

Phone (mobile): 201-406-3342

Email(s):

bryan.guner@gmail.com

gunerb1@tcnj.edu



GitHub:

https://github.com/gunerb1

TECHNICAL SKILLS & SOFTWARE

DYNAMIC TIME WARPING TRIGGERED GUITAR EFFECTS PLATFORM

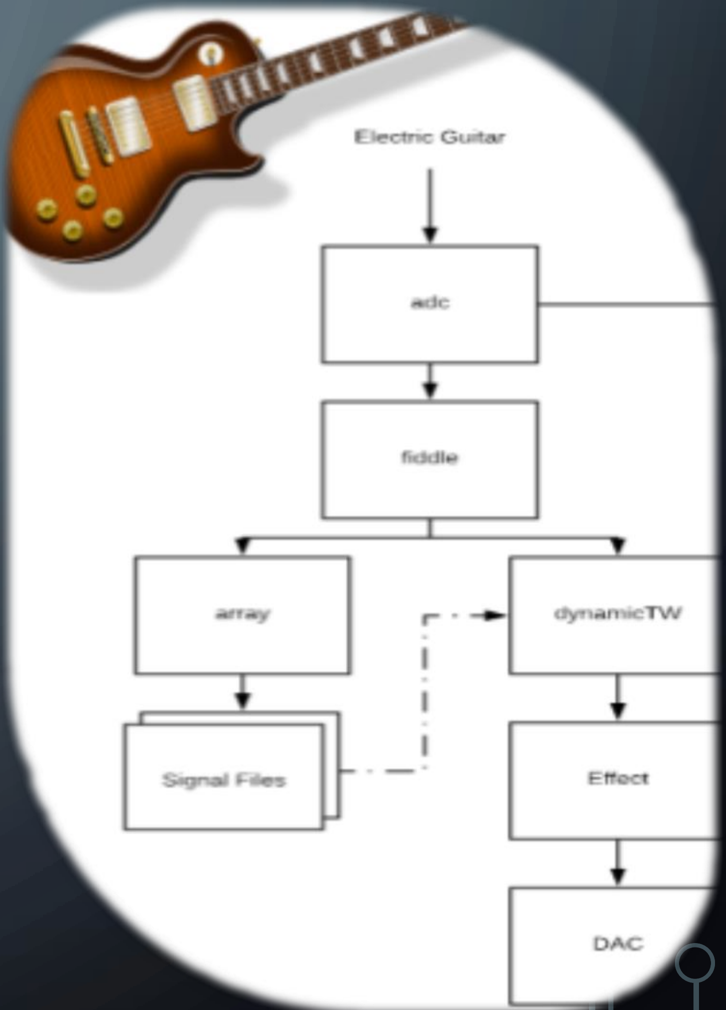
Senior Design Project (TCNJ)

In live performance, guitar effect pedals are a versatile yet limiting asset, requiring presence of mind on the part of the performer. This platform offers an automatic solution to the restrictions that guitar effect pedals present.

- **System Architecture:**

➢ read in a guitar signal during the 'learning' phase and isolate subsections of a performance needed to generate the DTW learned-threshold.

➢ then implement an analysis based on a modified dynamic time warping (DTW) algorithm, to compare the DTW cost function of incoming live audio against the cost-thresholds determined in the pre recording phase.

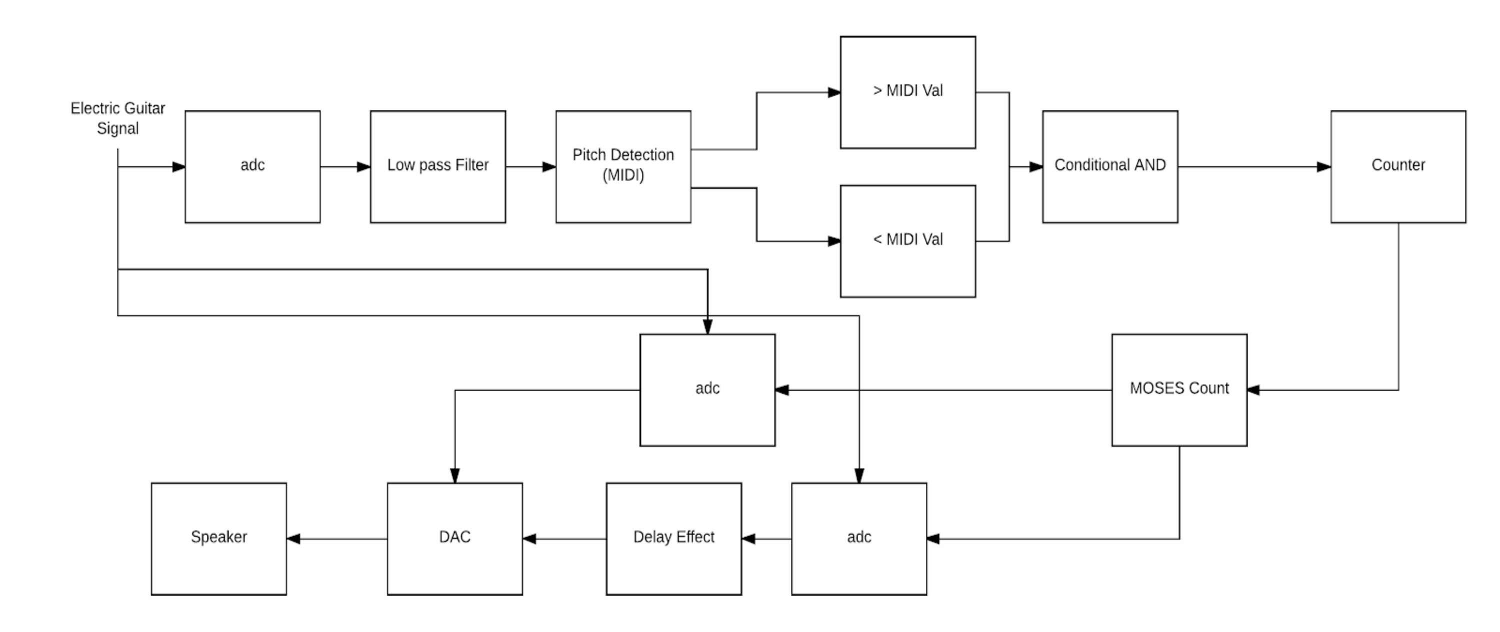


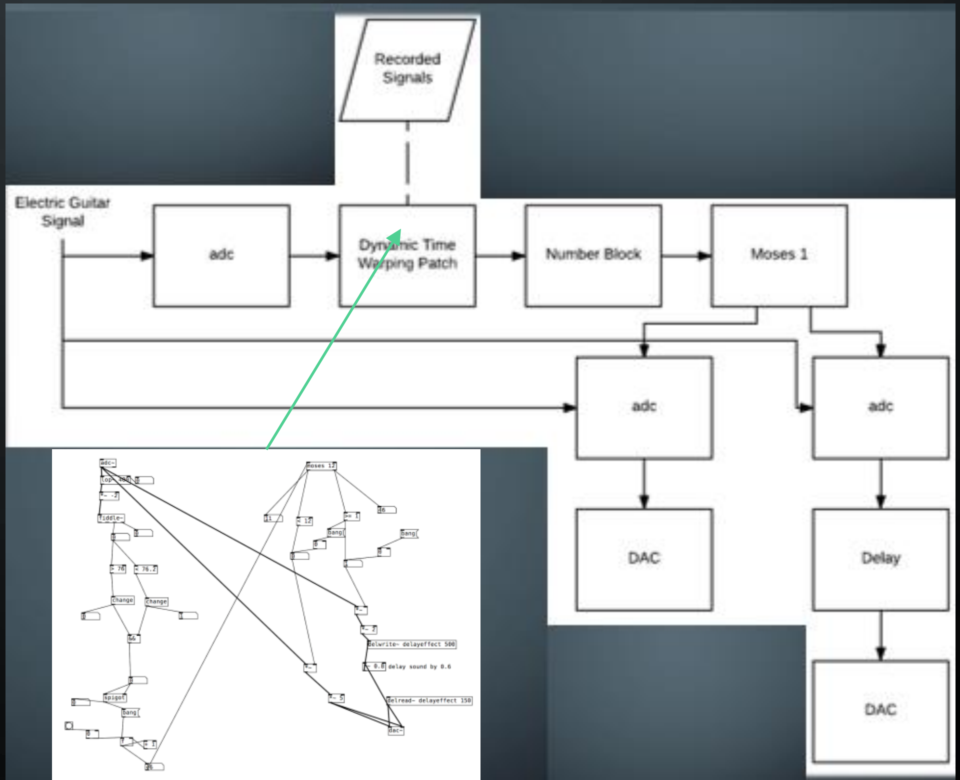
HOW IT WORKS:

This automation was achieved through the use of Pure Data, a GUI for audio manipulation applications, with embedded Python externals. When the algorithm detects a match, the platform runs the ‘pure’ digitalized audio signal through custom made PD effects patches!

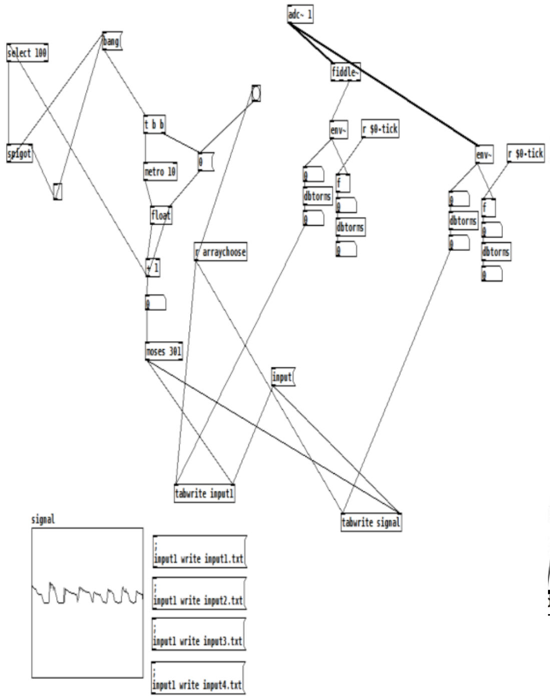
Dynamic Time Warping External:

- Feed in two song performances for learning phase and obtain Least Cost Path (LCP) for each sub signal
- Compare Incoming live signal with sub signal of one of the recorded performance
- When LCP value is less than or equal to the LCP obtained from learning phase, trigger guitar effect

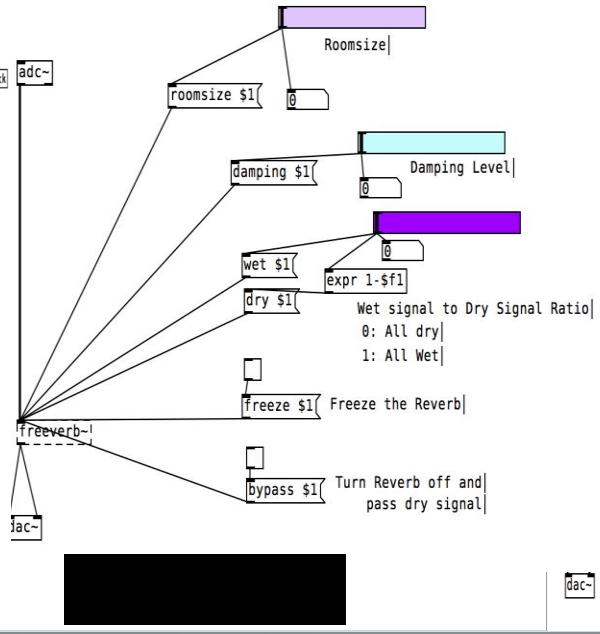




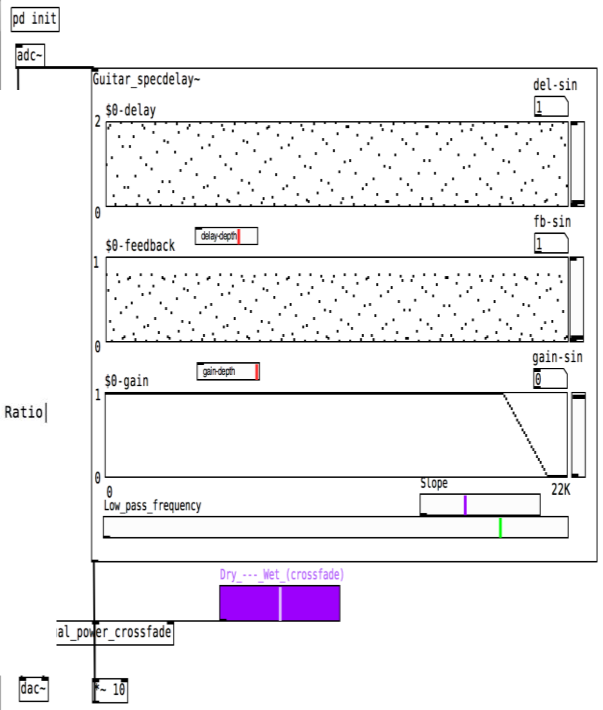
Recording (.wav) Patch:



Reverb Patch

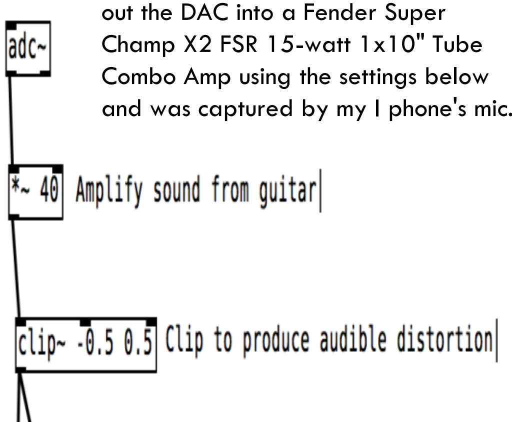


Spectral Delay Patch



(visible part of)Fuzz Patch

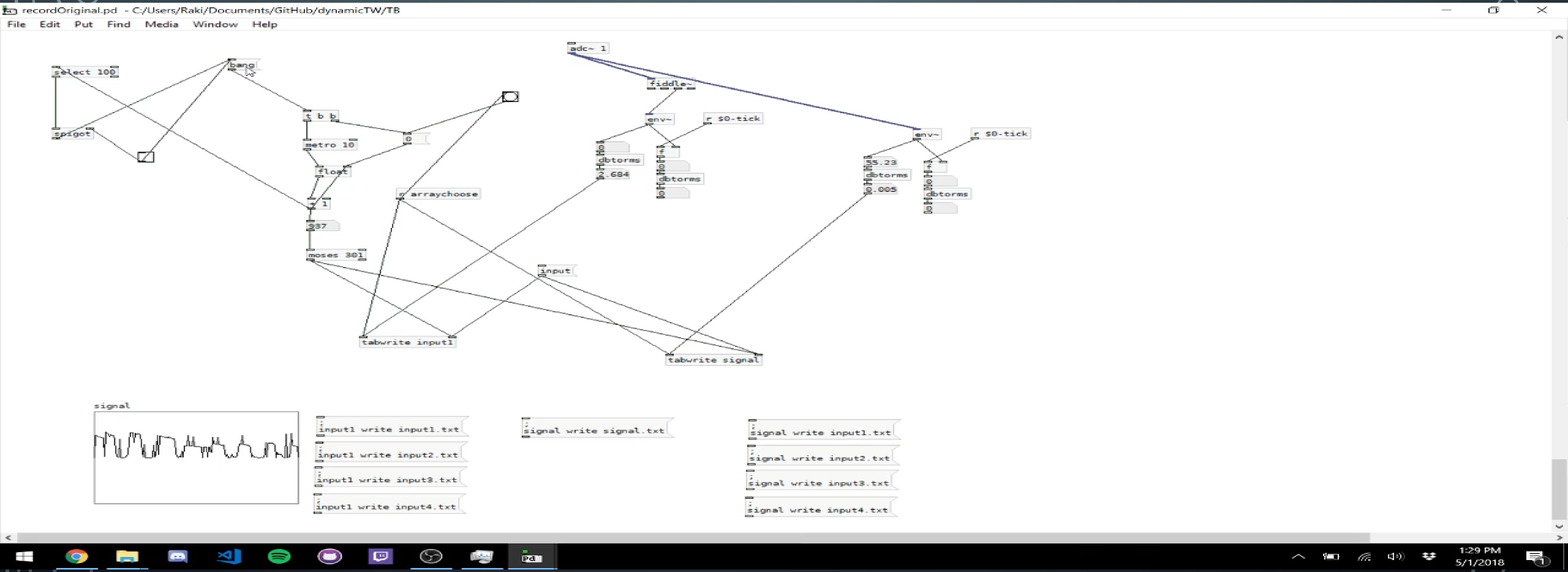
Signal sent DI into Scarlet 2i2 to be ported into pure data and then back out the DAC into a Fender Super Champ X2 FSR 15-watt 1x10" Tube Combo Amp using the settings below and was captured by my I phone's mic.

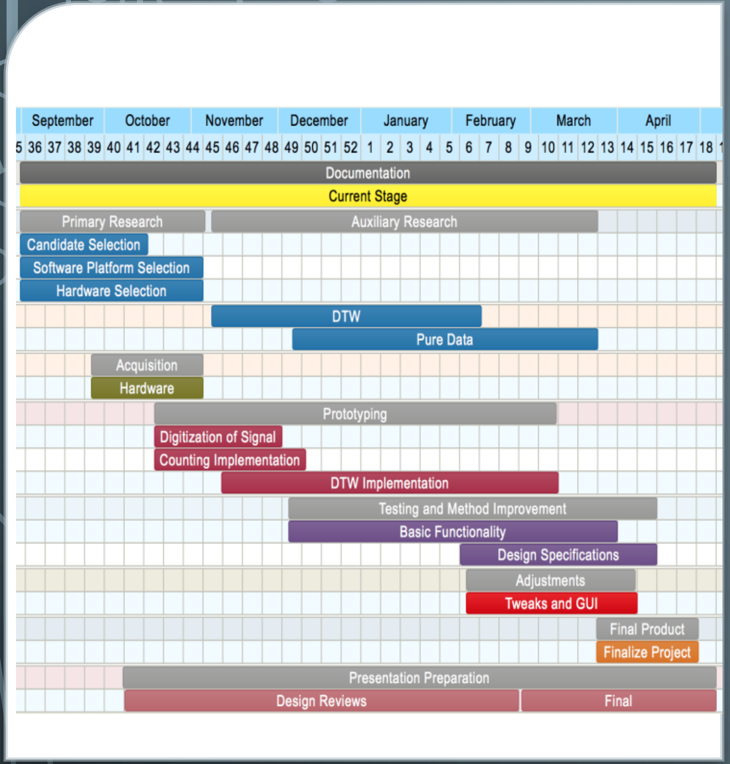


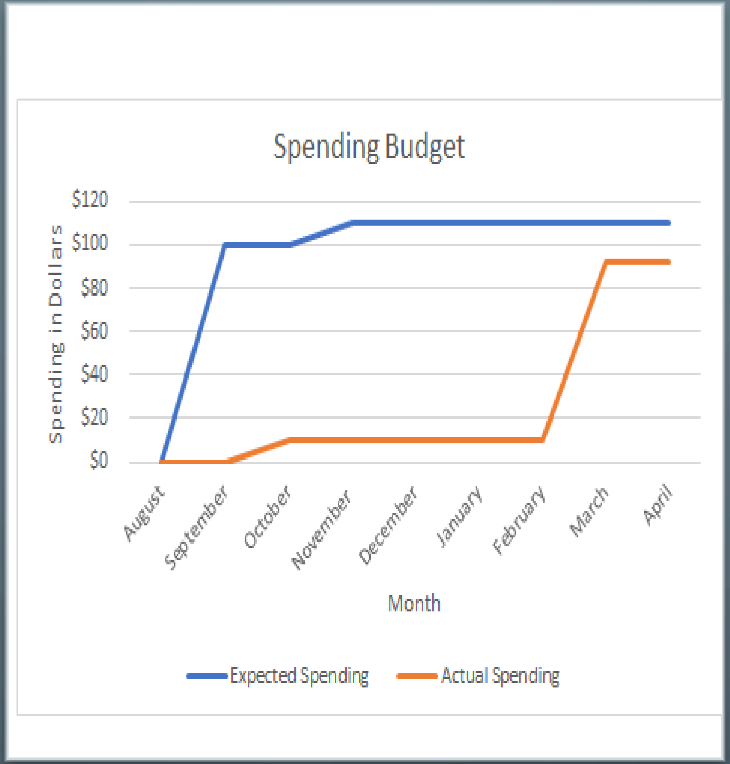
PURE DATA PATCH DEMOS:

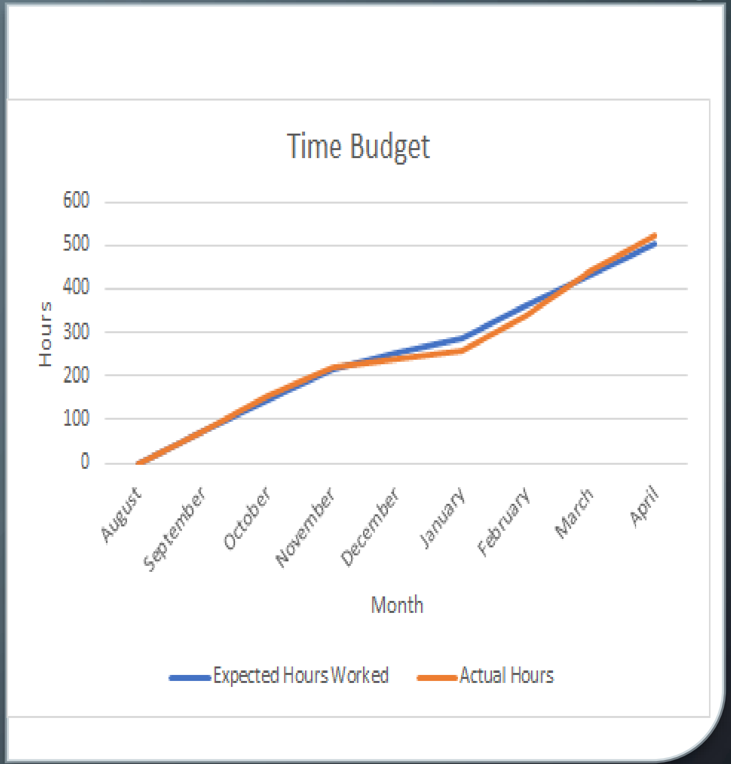
PLATFORM DEMO (WAIT UNTIL 0:18 FOR AUDIO)

This demo uses the reverb patch (please forgive the quality of my guitar performance in this; at the time of this recording I hadn't slept in almost 48 hours in a desperate attempt to meet multiple concurrent project deadlines!!!)









DOCUMENTATION

Senior Design Thesis
Grade Awarded: A-

CODE @ EITHER:
HTTPS://GITHUB.COM/GUNERB1/TRIGGERED-GUITAR-EFFECTS-PLATFORM

OR (FOR MORE COMPREHENSIVE DOCUMENTATION REFER TO MY PARTNER RALPH QUNITO’S REPOSITORIES!)

HTTPS://GITHUB.COM/RAKIROAD/DYNAMICTW &

HTTPS://GITHUB.COM/RAKIROAD/TRIGGERED-GUITAR-EFFECTS-PLATFORM

PSOC PROGRAMMABLE MICROCONTROLLER GUITAR FIR DELAY PEDAL

BRYAN GUNER

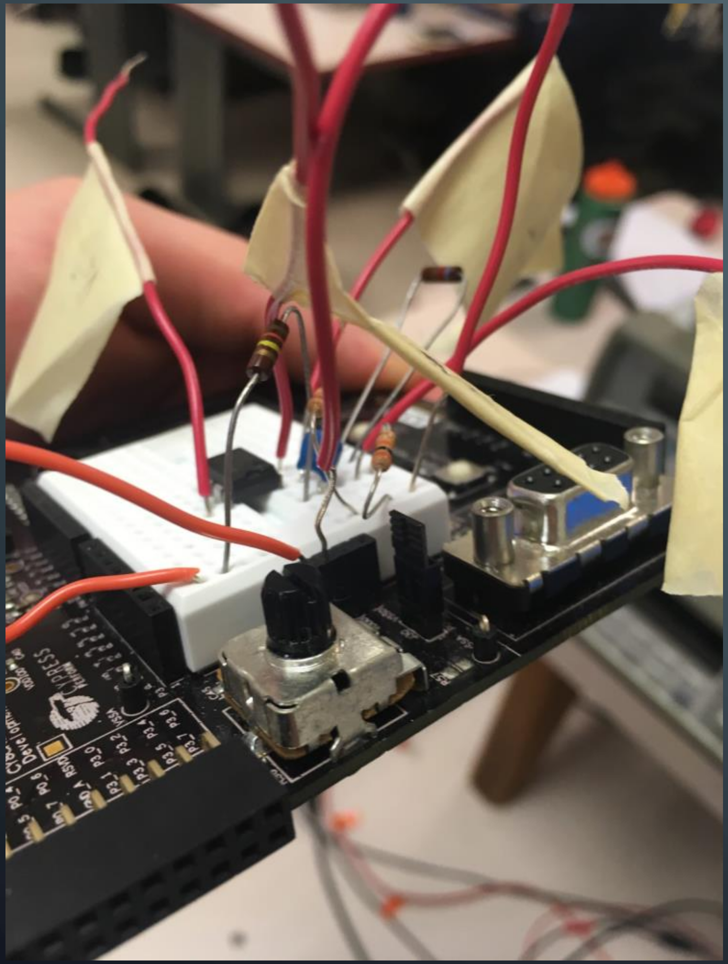
DESIGN ELEMENTS

- Analog to Digital Converter (taking analog guitar input, converting to digital to read and utilize)
- Rectifier & Pre-Amplifier (to accommodate the input constraints of the micro controller)
- Digital to Analog Converter (taking digital signal, converting to analog for output)
- Finite Impulse Response Digital Filter
- 2 Push Buttons

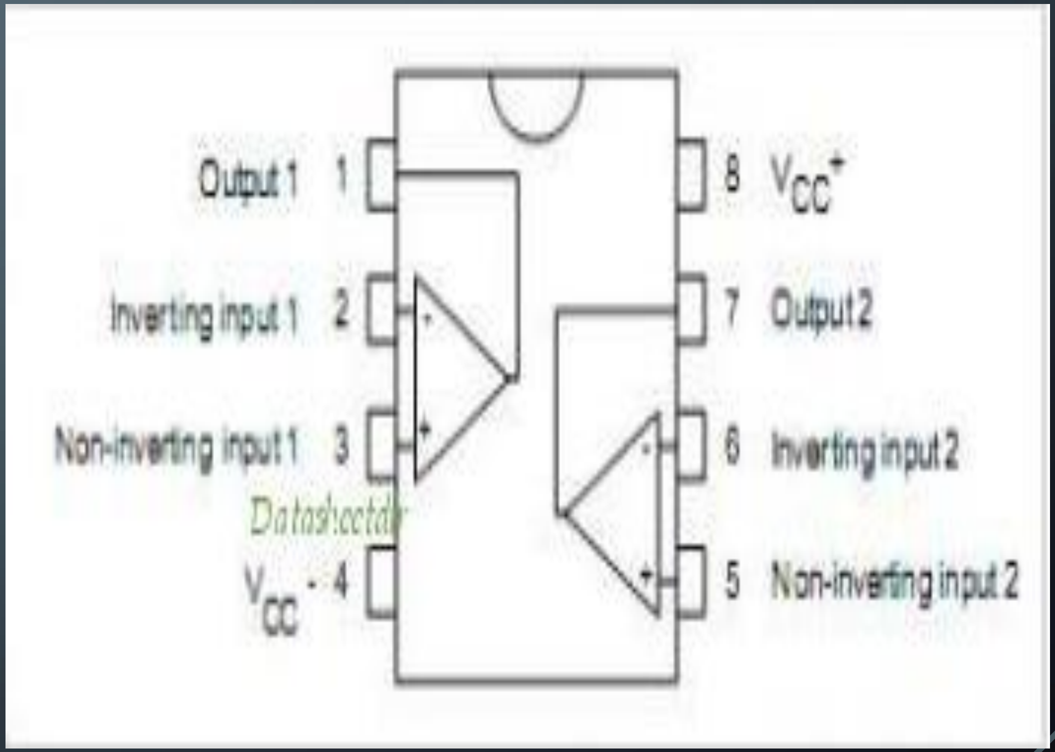
OBJECTIVE

- Design a guitar pedal to take a guitar signal as an input, and output the delayed and echoed sound numerous times
- The pedal must implement a preamplifier in order to manage the negative voltage input
- The pedal must poll for updated input data in order to produce a continuous output
- The pedal must use an ADC in order to read and utilize the input

CURRENT MODEL

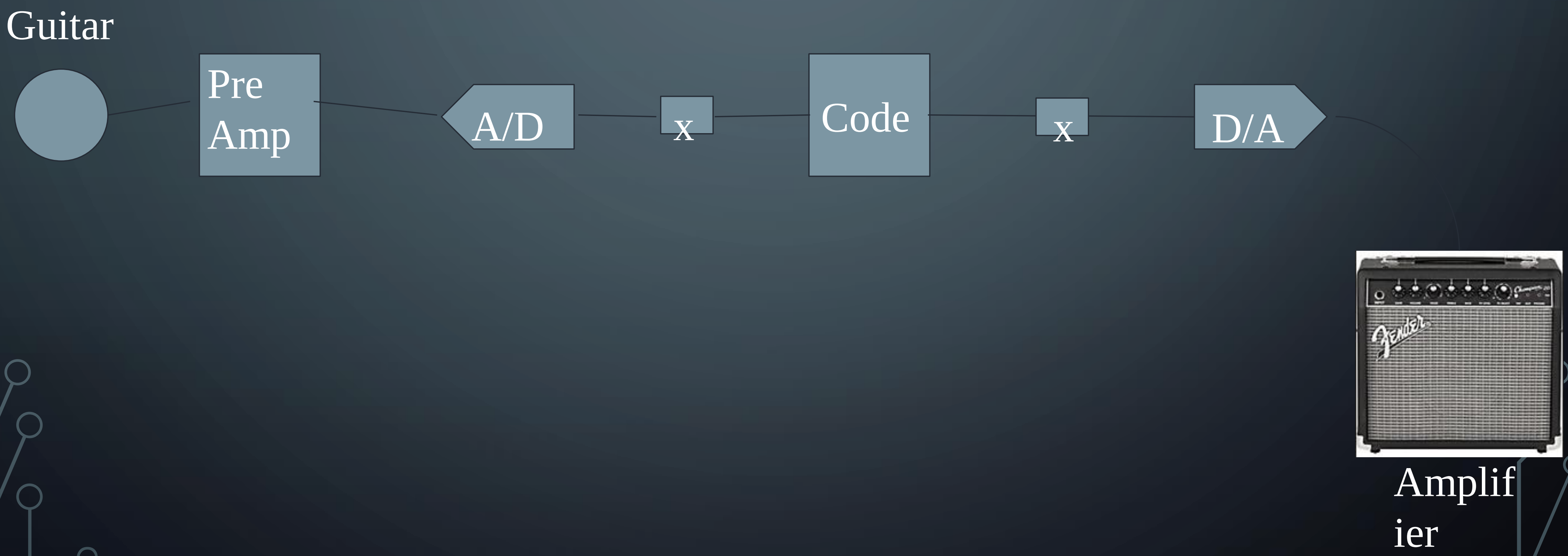


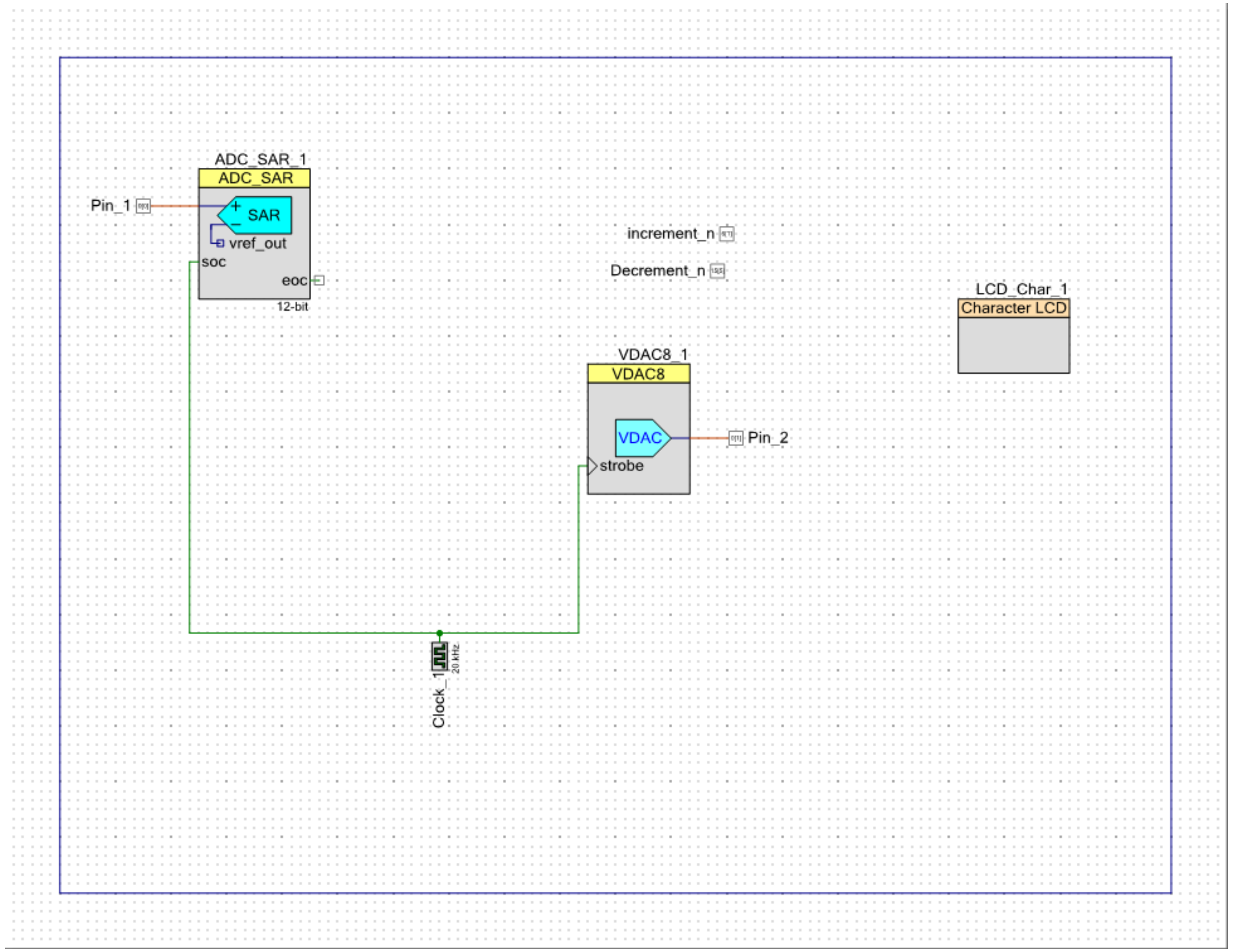
Physical Implementation of Circuit Design



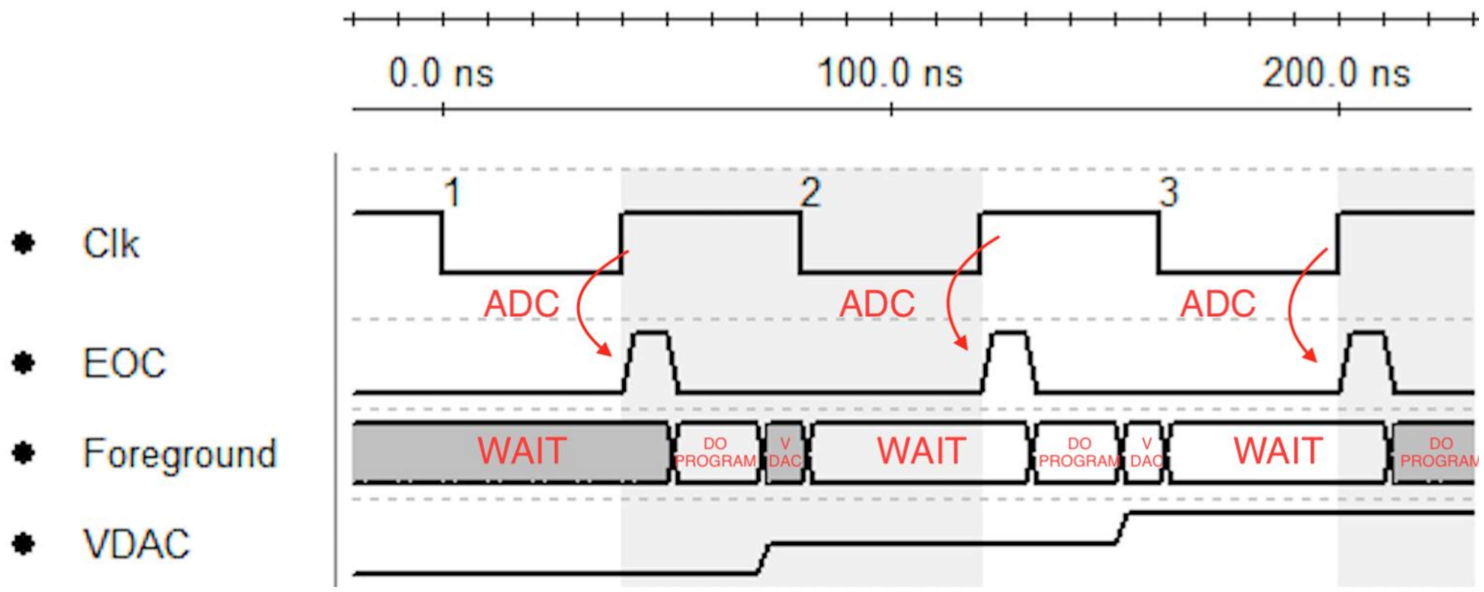
Dual Op-Amp (MC1458)

BLOCK DIAGRAM





PSOC SCHEMATIC



TIMING DIAGRAM

CURRENT MODEL CODE

```c

int main()
{
    int maxs=20000;
    int sum=0;
    int windex=0;
    volatile uint16_t n=10;
    int delay=2000;
    int k;
    uint16_t x;
    uint16_t gsa [20000]={0};             //sample array initialized to 0


     CyGlobalIntEnable;              /* Enable global interrupts. */

     Clock_1_Start() ;
     ADC_SAR_1_Start();
     VDAC8_1_Start();
     LCD_Char_1_Start();
     LCD_Char_1_Position(0u, 0u);

     for(;;)
     {
     LCD_Char_1_ClearDisplay();
     LCD_Char_1_PrintNumber(n);                             //outputs n to display
     CyDelay(500);

         if( Decrement_n_Read() == 0 )                      // if pressed
         {
             CyDelay(500);                                  //to make sure increment/decrement event registers only once per press
             n=n-1;                                         //decrement
         }
         else if( increment_n_Read() == 0)
         {
            CyDelay(500);
             n=n+1;                                         //Increment
         }


         ADC_SAR_1_IsEndConversion(ADC_SAR_1_WAIT_FOR_RESULT);//polling
         x=ADC_SAR_1_GetResult16();
         gsa[windex]=x;                                     //indexing sample array
         sum=0;
         for(k=0;k<n;++k)
         {
             sum+=gsa[(windex+maxs-k*delay)%maxs];          //circular buffer

         }
         sum=sum>>4;
         sum=sum/n;                                        //Prevents increasing amplitude with every echo
         VDAC8_1_SetValue(sum);
         windex=(windex+1)%(maxs);
         }
}

```

MATLAB SIMULATION

- MATLAB was used to debug and simulate the guitar delay pedal
- Utilized FIFO (first in first out), delay and echo

FIFO.m | Delay_Model.m | Echo.m

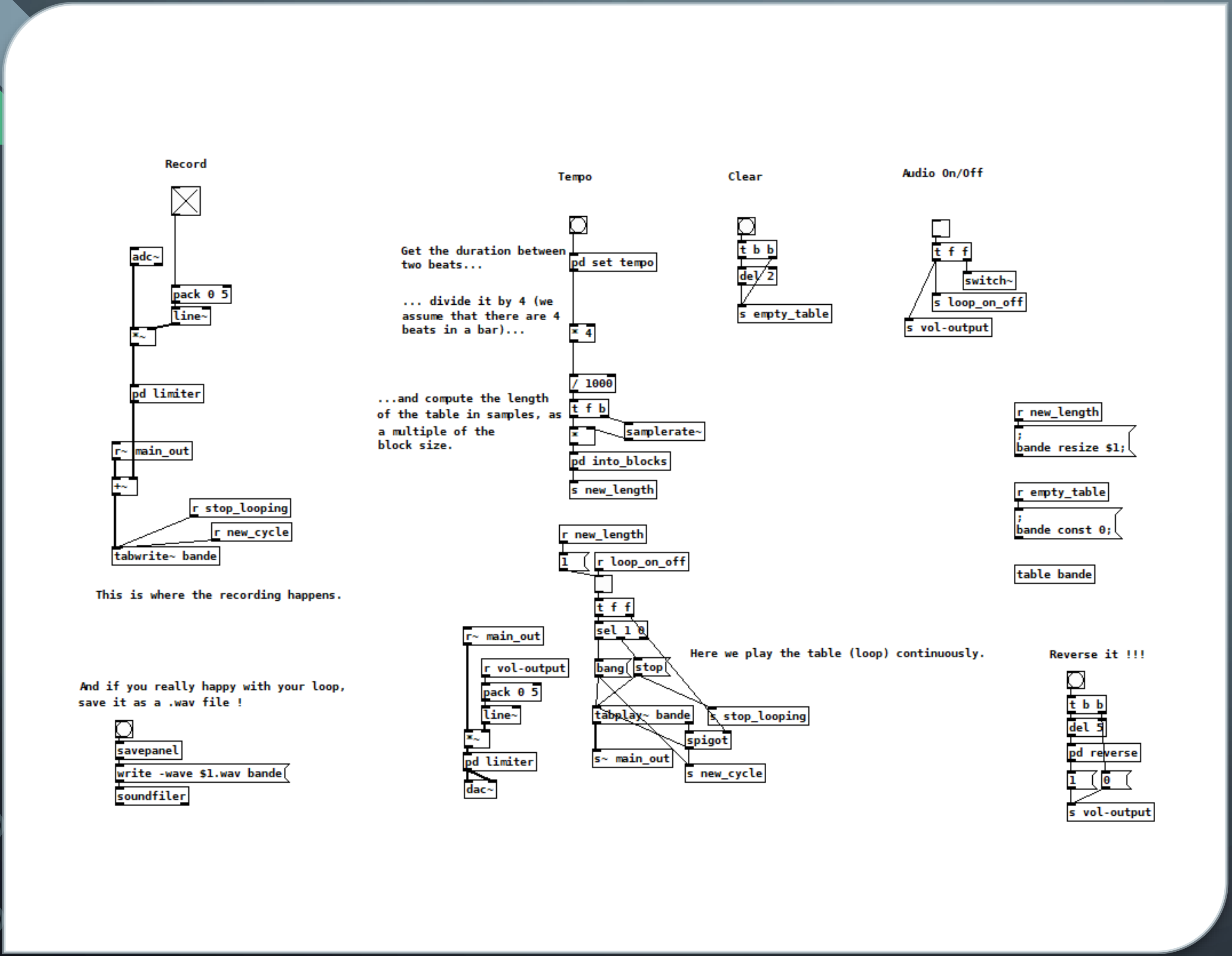
```
FIFO ('init');
fs = 20000;
f0=440;
 delay = 6000;
 n_echos = 6;
x=(1:10*fs);
x=exp(-1.*x./2000).*cos(2*pi()*fs/f0*(1+x/100000).*x);
plot(x(1:1000))
%x(1000:end) = 0;
%sound(x)

for n=1:20000 %assuming 10 seconds * 20,000
    y(n)=Echo(x(n),delay,n_echos);
end

    sound(y);
    plot(y);
```

FIFO.m | Delay_Model.m | Echo.m

```
function y = Echo(x, delay, n_echos )



    sum = 0;
   FIFO('write', x);
        for i = 1:n_echos
            sum = sum + FIFO('read', delay * i);
        end
    y = sum;
end
```

FIFO.m | Delay_Model.m | Echo.m

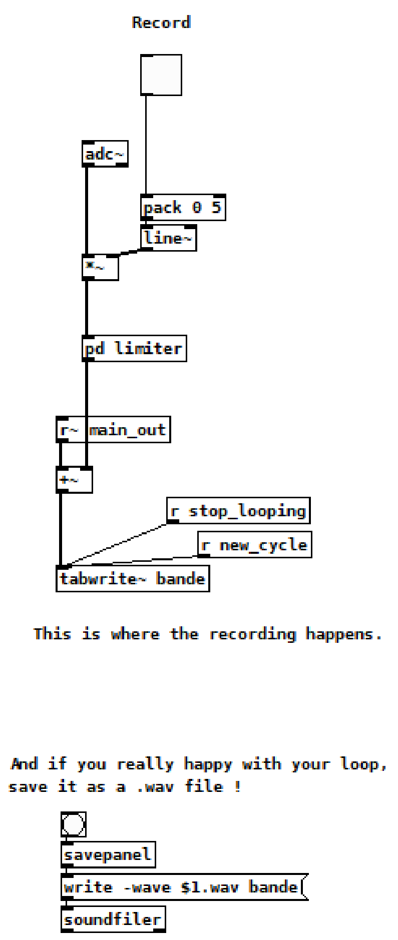
```
function result = FIFO (command, arg)
    if strcmp(command, 'read')
        result = read_fifo(arg);
    else
        if strcmp(command, 'write')
             write_fifo(arg);
           else
             if strcmp(command, 'init')
                 init_fifo();
             end
        end
    end
end

function init_fifo()
global fifo_buf write_ptr MAX_DELAY; % globalizing variables
MAX_DELAY = 20000;%Max delay of 20kHz

fifo_buf = zeros(MAX_DELAY,1);%buffer = array of all zeros

write_ptr = 1;
end
function val = read_fifo(delay)% Val = fifo output
   global fifo_buf write_ptr MAX_DELAY;

    % Read delayed values
    % Delay read_ptr by an amount of samples, based on 'delay'
    read_ptr = mod((write_ptr - 2) - delay + MAX_DELAY, MAX_DELAY)+1 ;

    val = fifo_buf(read_ptr);

end

function write_fifo( val )%using output of read_fifo
global fifo_buf write_ptr MAX_DELAY;

    fifo_buf(write_ptr) = val;
    write_ptr = write_ptr + 1;%increment write_ptr by 1

    if (write_ptr > MAX_DELAY)%Once write_ptr is greater than max delay, restart
        write_ptr = 1;
    end
end
```



PURE DATA LOOPER

Features:

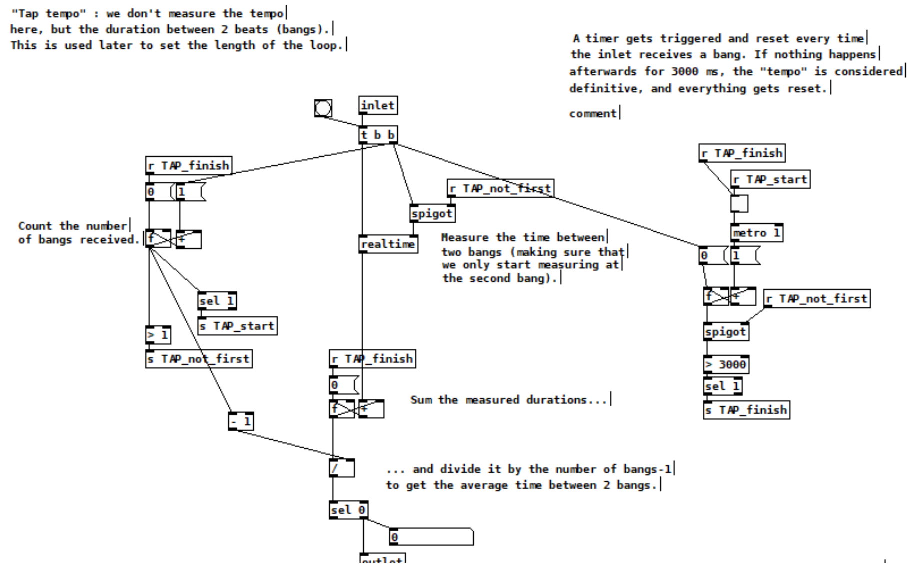
- tap tempo
- Limiter on both the input and the output to prevent intermodulation distortion produced by superimposed layers.
- “Reverse” function for fun!



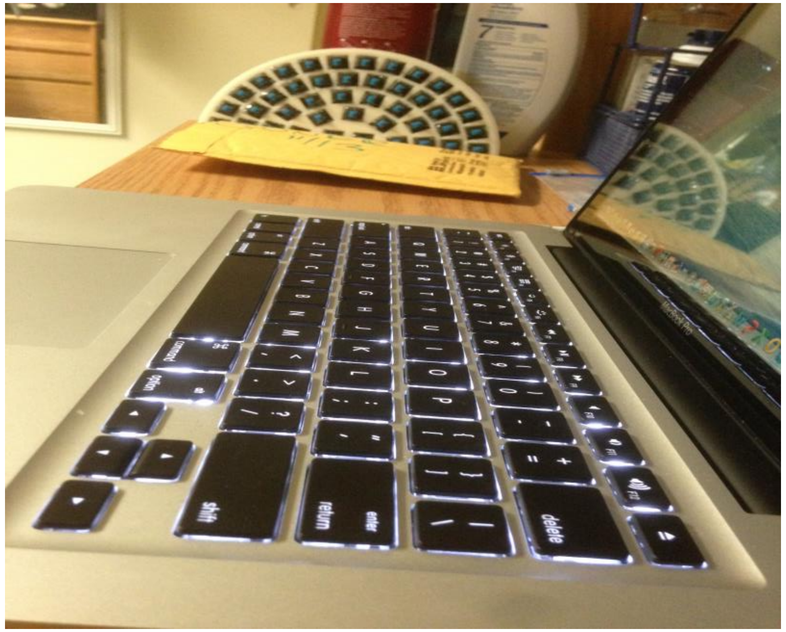
BASIC OPERATION

- The loop is stored in a table in Pd, which is played repeatedly.
- While this table is being played, the output and the current input is also written to it.
- If there is no input the loop copied on itself.

DESIGN CONSIDERATIONS



- Pure Data processes the audio in blocks of samples (64 samples by default).
- Any audio event occurring during a block will only be taken into account at the beginning of the next block.
- The table's length must thus be a multiple of the block size (64 in my patch), so that the end of the last block of the loop corresponds exactly to the beginning of the first block.
- If the table's length is not a multiple of the block size, audible pops and clicks will appear while looping, and gradually turn into a very annoying noise.



Casing designed in SolidWorks and fabricated via TCNJ's 3D printer

Ergonomic placement of characters decided by result of character frequency counter (written in Java) as a substitute for the conventional qwerty layout.

ONE HANDED KEYBOARD

PURE DATA SAMPLER