



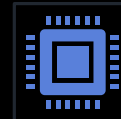
# BRYAN GUNER ENGINEERING PORTFOLIO



# TABLE OF CONTENTS



Contact Information



Software Skills



Project Experience & Code  
Samples



# Contact Information:

Address: 150 Henley Place, Weehawken, NJ, 07086

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

Phone (mobile): 201-406-3342

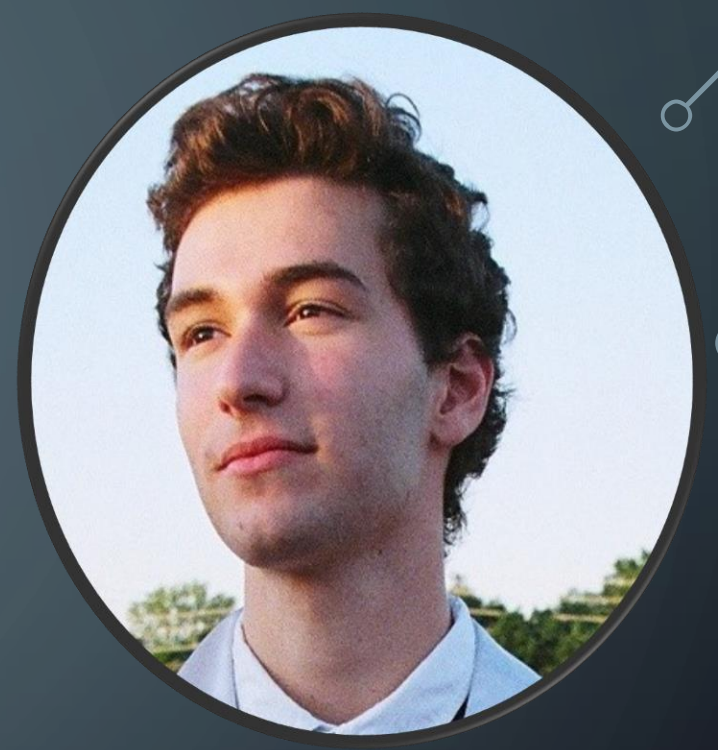
Email(s):

[bryan.guner@gmail.com](mailto:bryan.guner@gmail.com)

[gunerb1@tcnj.edu](mailto:gunerb1@tcnj.edu)

GitHub:

<https://github.com/gunerb1>





# TECHNICAL SKILLS & SOFTWARE

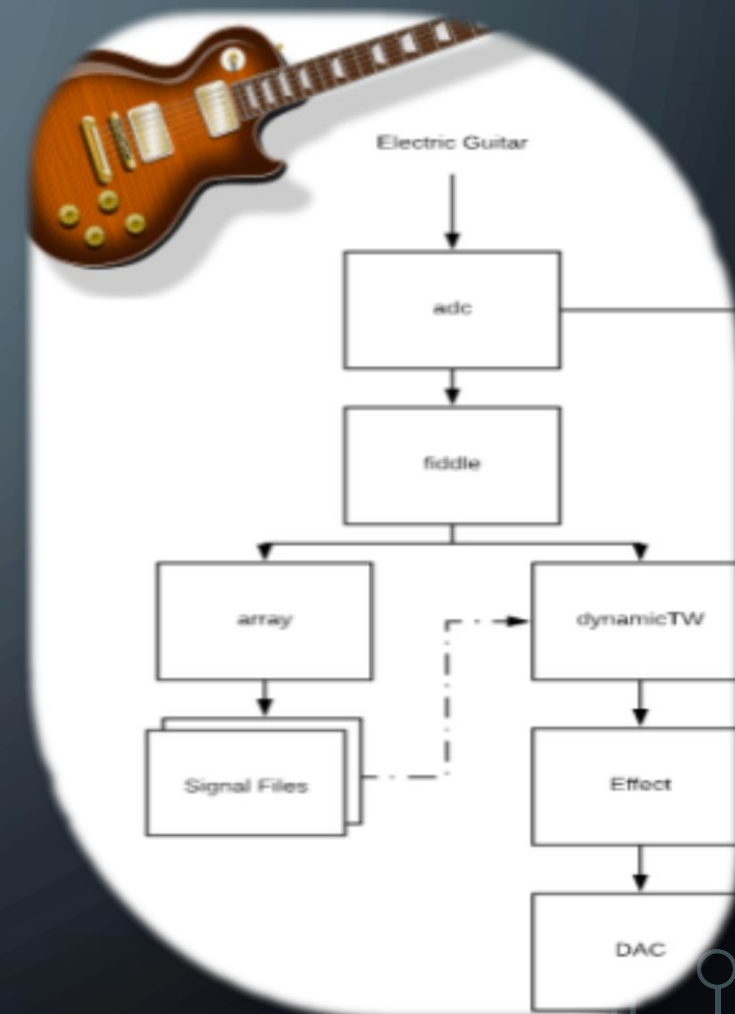
# DYNAMIC TIME WARPING TRIGGERED GUITAR EFFECTS PLATFORM

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.

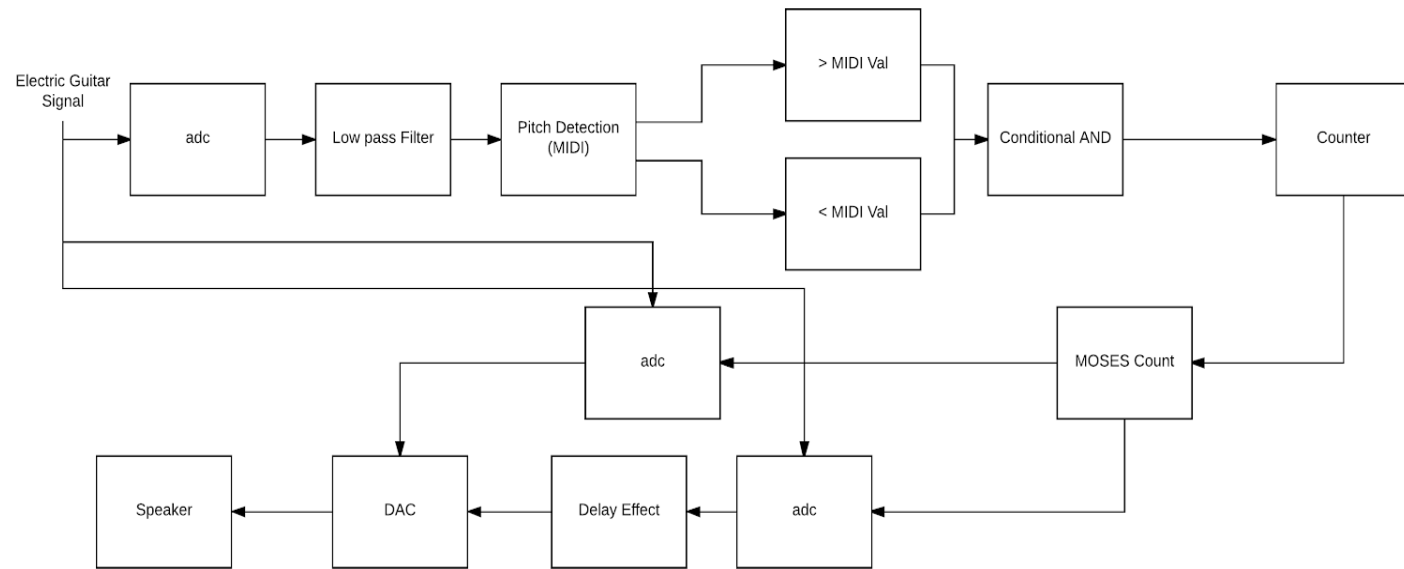
## • System Architecture:

- 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.



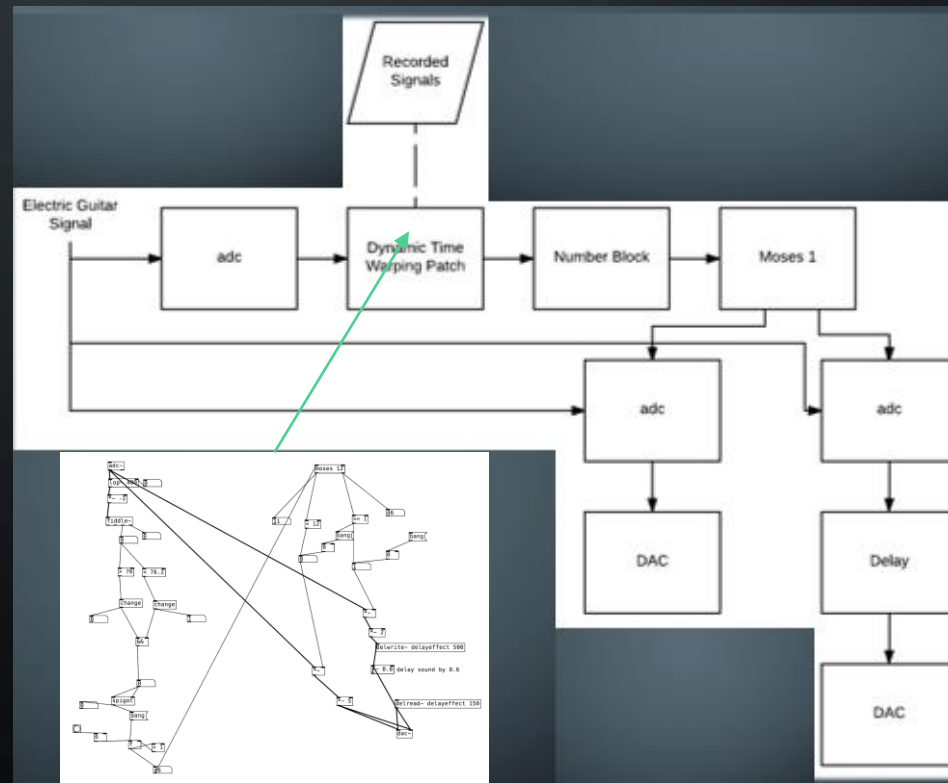
# HOW IT WORKS:

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!



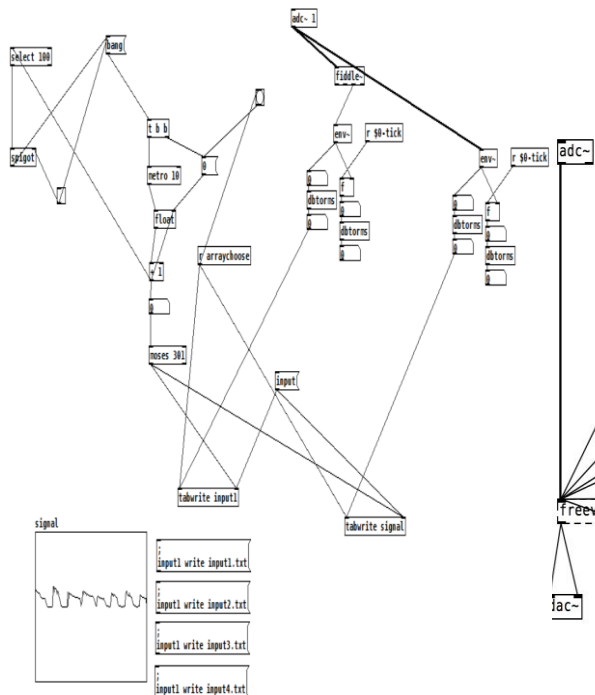
## Dynamic Time Warping External:

- 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

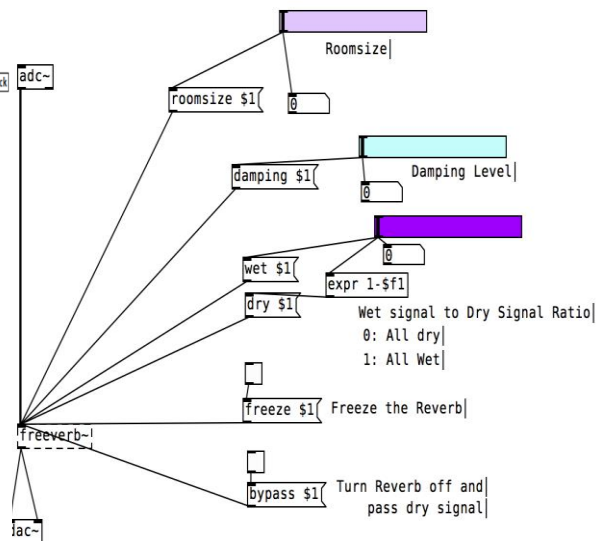




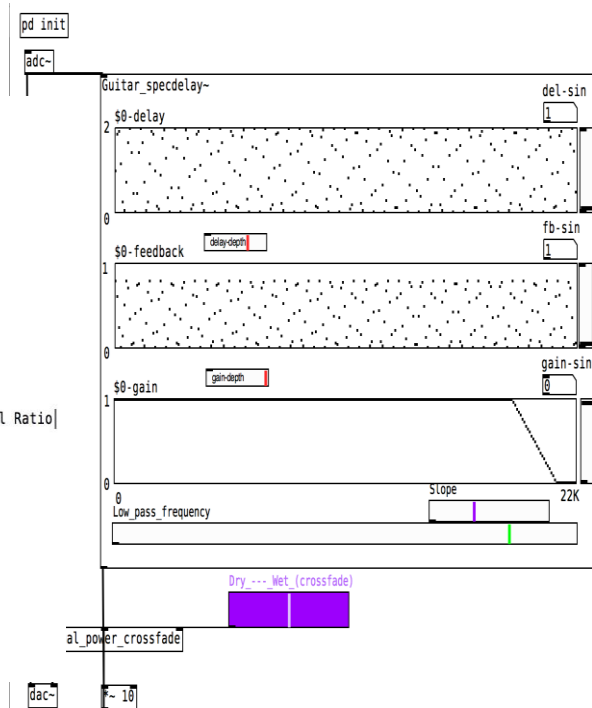
Recording (.wav) Patch:



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 iPhone's mic.

adc~

\*~ 40

Amplify sound from guitar

clip~ -0.5 0.5

Clip to produce audible distortion

dac~



# PURE DATA PATCH DEMOS:

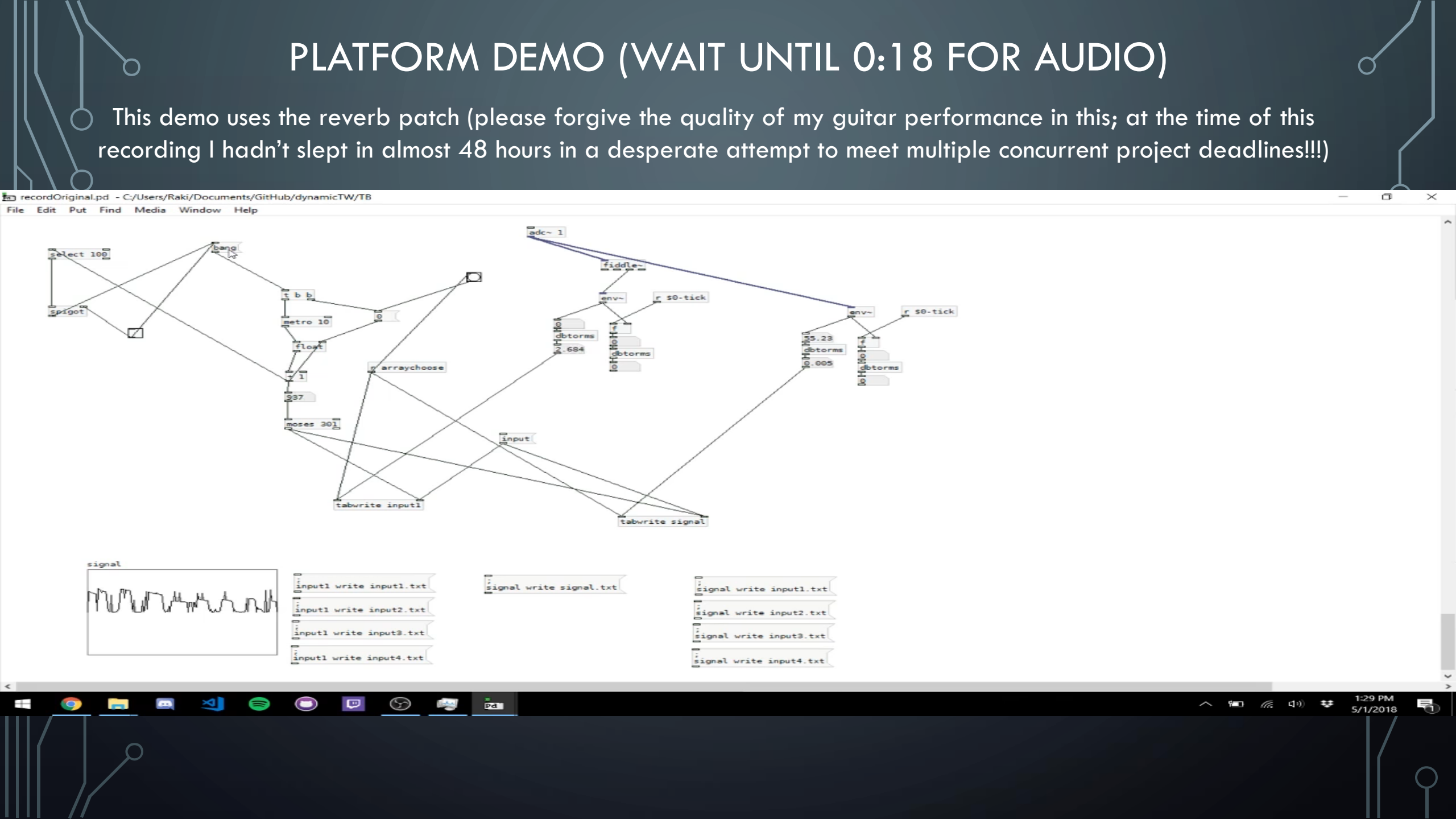


# PLATFORM DEMO (WAIT UNTIL 0:18 FOR AUDIO)

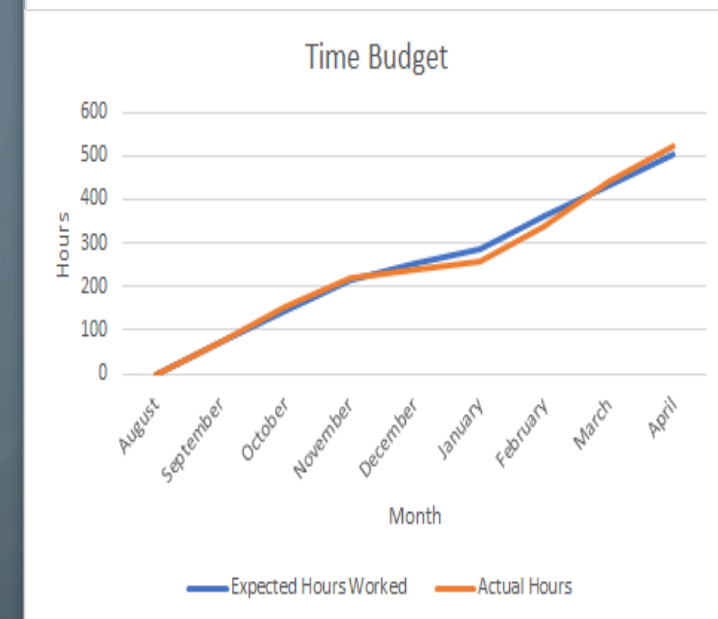
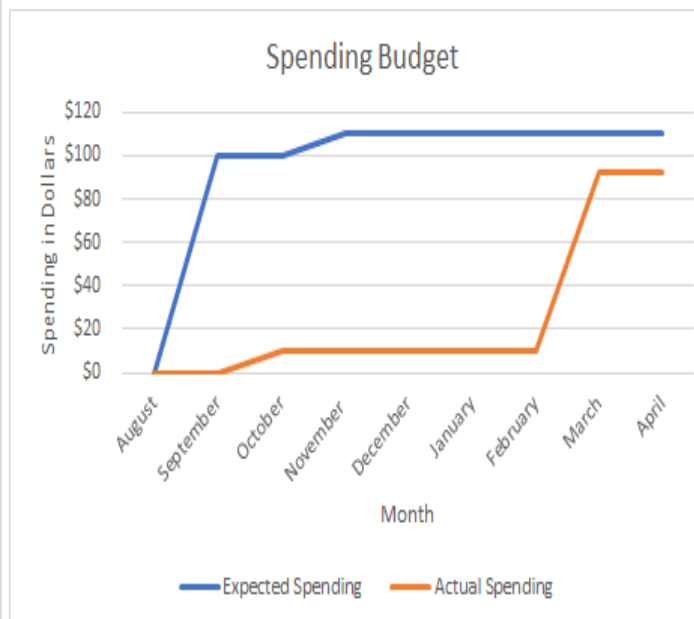
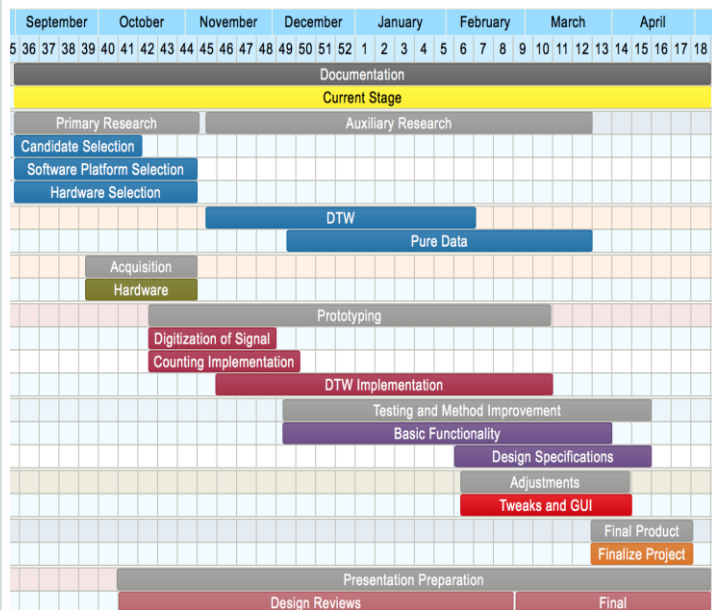
This demo uses the reverb patch (please forgive the quality of my guitar performance in this; at the time of this recording I hadn't slept in almost 48 hours in a desperate attempt to meet multiple concurrent project deadlines!!!)

# 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!!!)







Senior Design Thesis  
Grade Awarded: A-

# DOCUMENTATION



CODE @ EITHER:

[HTTPS://GITHUB.COM/GUNERB1/TRIGGERED-GUITAR-EFFECTS-PLATFORM](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/DYNAMICTW) &

[HTTPS://GITHUB.COM/RAKIROAD/TRIGGERED-GUITAR-EFFECTS-PLATFORM](https://github.com/RAKIROAD/TRIGGERED-GUITAR-EFFECTS-PLATFORM)





# PSOC PROGRAMMABLE MICROCONTROLLER GUITAR FIR DELAY PEDAL

BRYAN GUNER

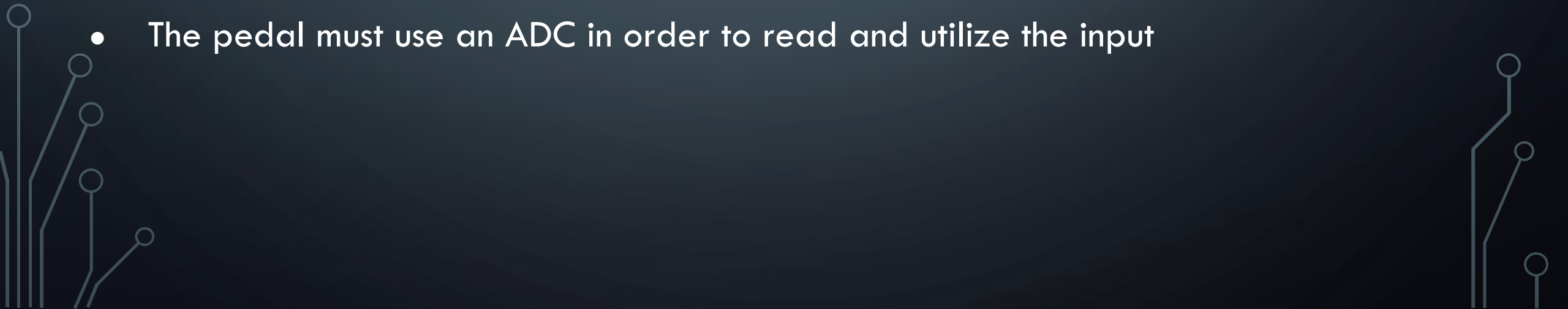


# DESIGN ELEMENTS

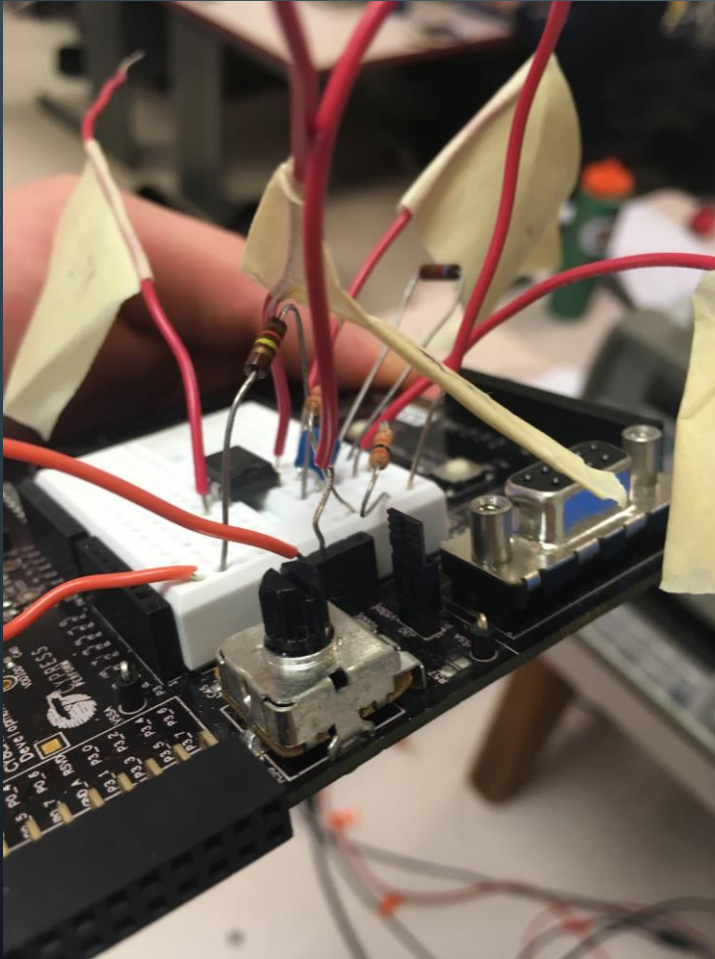
- Analog to Digital Converter (taking analog guitar input, converting to digital to read and utilize)
- Rectifier & Pre-Amplifier (to accommodate the input constraints of the micro controller)
- Digital to Analog Converter (taking digital signal, converting to analog for output)
- Finite Impulse Response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



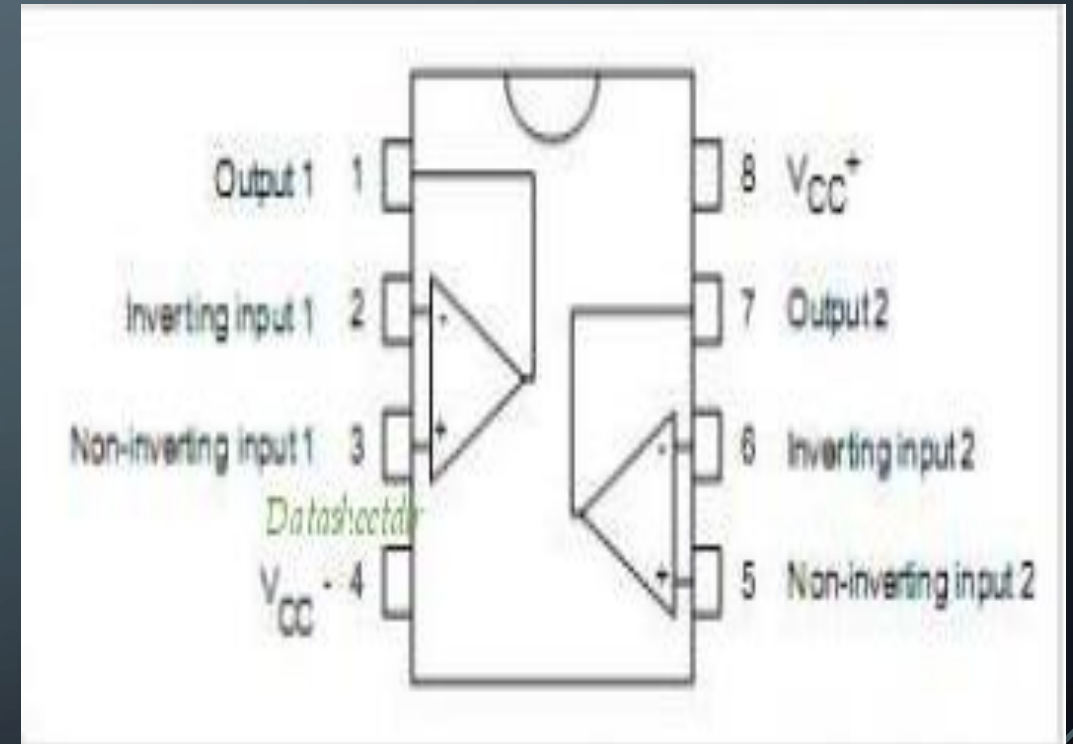
## OBJECTIVE

- 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

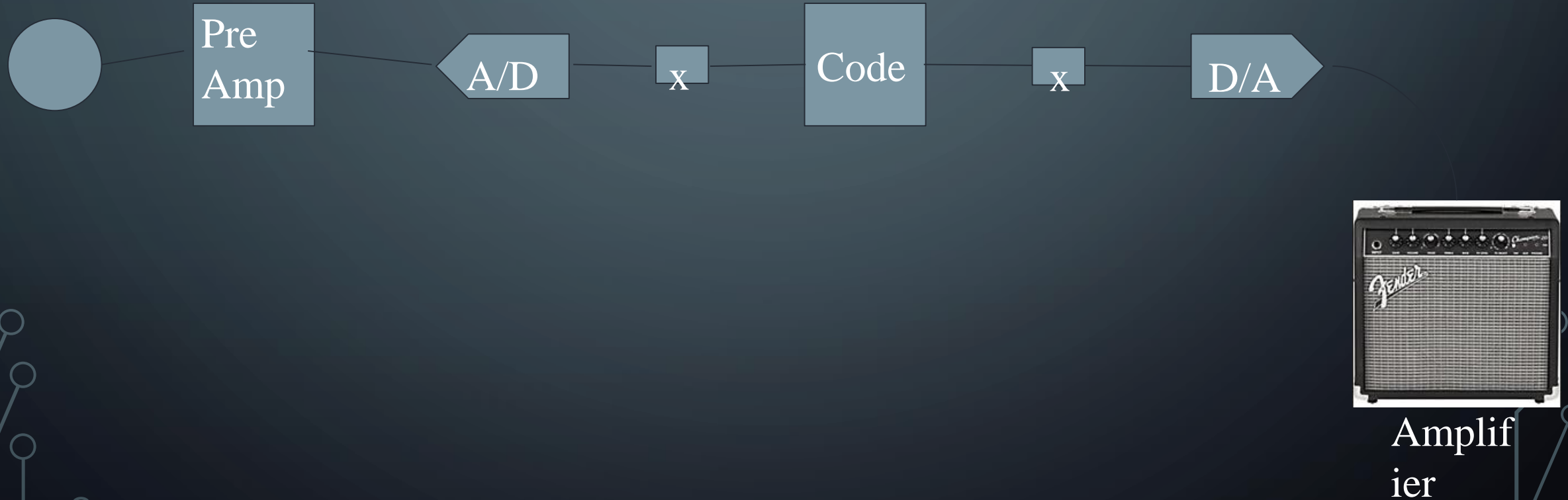


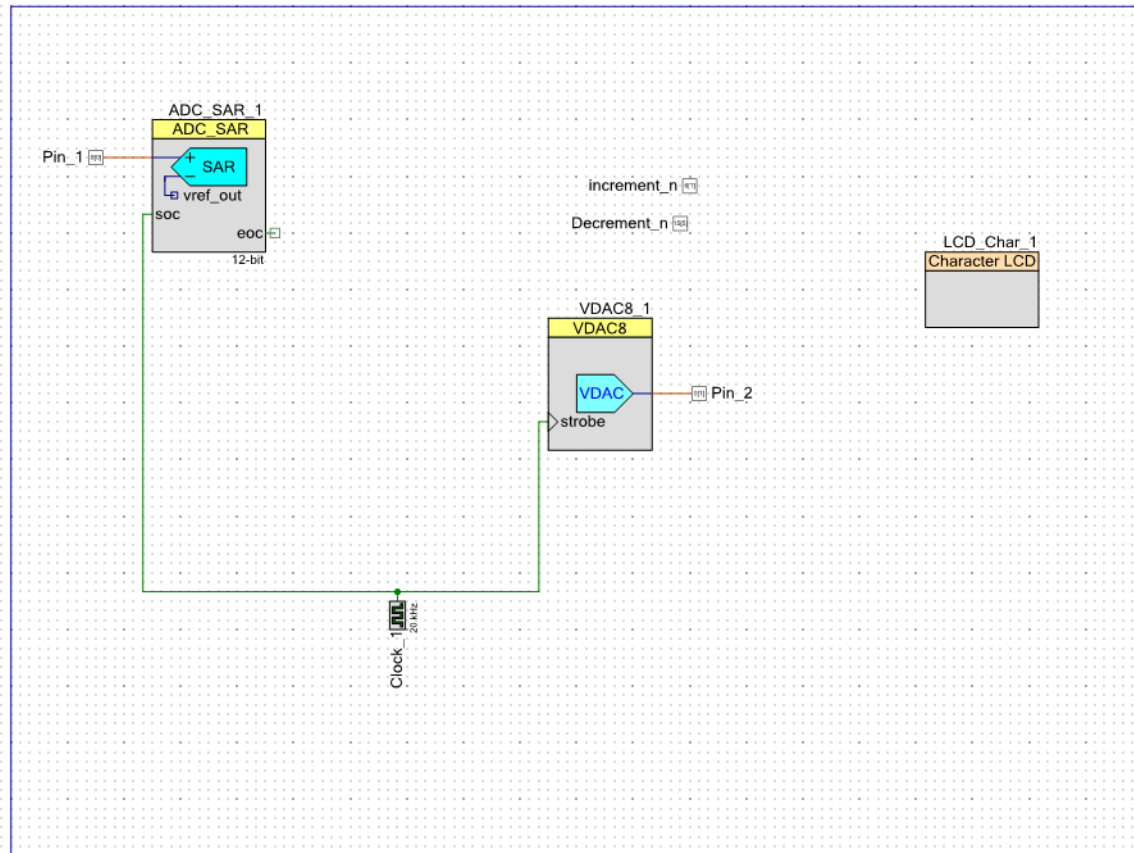
Dual Op-Amp (MC1458)



# BLOCK DIAGRAM

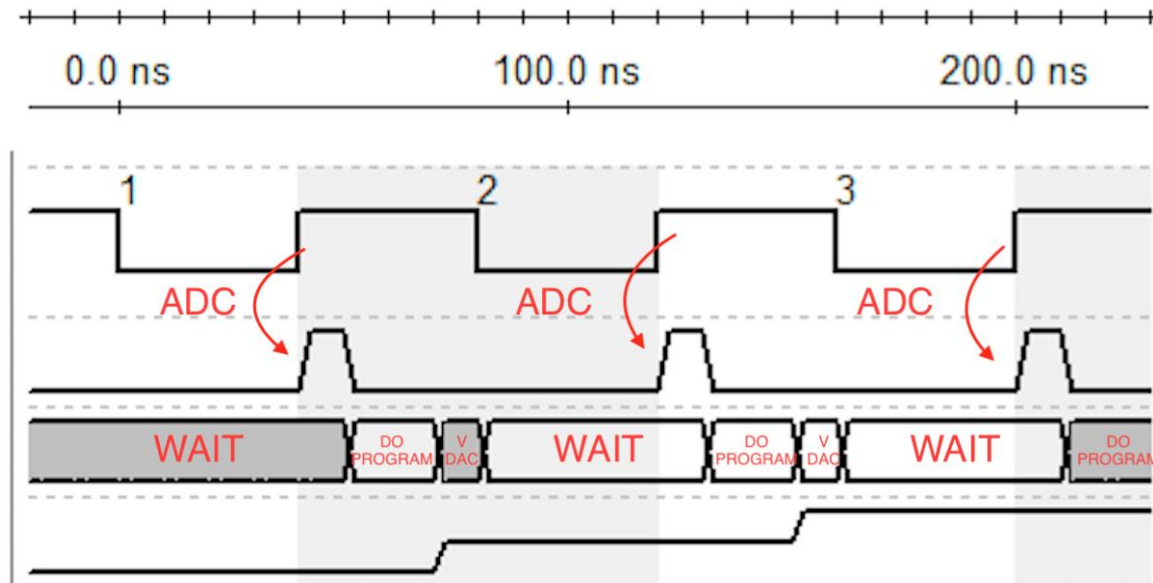
Guitar





# PSOC SCHEMATIC

- Clk
- EOC
- Foreground
- VDAC



# TIMING DIAGRAM



# CURRENT MODEL CODE

```
2|
3| int main()
4| {
5|     int maxs=20000;
6|     int sum=0;
7|     int windex=0;
8|     volatile uint16_t n=10;
9|     int delay=2000;
10|    int k;
11|    uint16_t x;
12|    uint16_t gsa [20000]={0};           //sample array initialized to 0
13|
14|
15|    CyGlobalIntEnable;                  /* Enable global interrupts. */
16|
17|    Clock_1_Start();
18|    ADC_SAR_1_Start();
19|    VDACS_1_Start();
20|    LCD_Char_1_Start();
21|    LCD_Char_1_Position(0u, 0u);
22|
23|    for(;;)
24|    {
25|        LCD_Char_1_ClearDisplay();
26|        LCD_Char_1_PrintNumber(n);      //outputs n to display
27|        CyDelay(500);
28|
29|        if( Decrement_n_Read() == 0 )   // if pressed
30|        {
31|            CyDelay(500);                //to make sure increment/decrement event registers only once per press
32|            n=n-1;                       //decrement
33|        }
34|        else if( increment_n_Read() == 0)
35|        {
36|            CyDelay(500);
37|            n=n+1;                       //Increment
38|        }
39|
40|
41|        ADC_SAR_1_IsEndConversion(ADC_SAR_1_WAIT_FOR_RESULT); //polling
42|        x=ADC_SAR_1_GetResult16();
43|        gsa[windex]=x;                  //indexing sample array
44|        sum=0;
45|        for(k=0;k<n;++k)
46|        {
47|            sum+=gsa[(windex+maxs-k*delay)%maxs]; //circular buffer
48|        }
49|
50|        sum=sum>>4;
51|        sum=sum/n;                      //Prevents increasing amplitude with every echo
52|        VDACS_1_SetValue(sum);
53|        windex=(windex+1)%maxs;
54|    }
55| }
56|
```

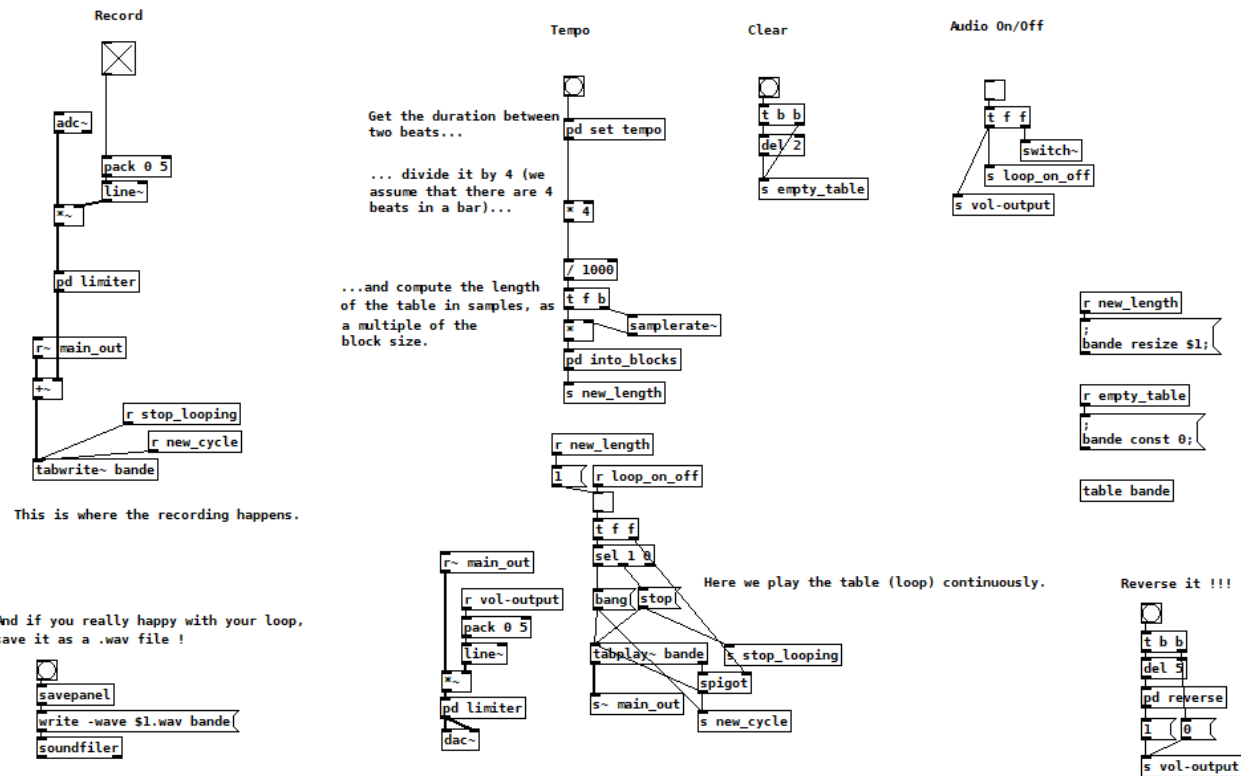
# MATLAB SIMULATION

- 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 +
1 - FIFO ('init');
2 - fs = 20000;
3 - f0=440;
4 - delay = 6000;
5 - n_echos = 6;
6 - x=(1:10*fs);
7 - x=exp(-1.*x./2000).*cos(2*pi()*fs/f0*(1+x/100000).*x);
8 - plot(x(1:1000))
9 - %x(1000:end) = 0;
10 - %sound(x)
11
12 - for n=1:20000 %assuming 10 seconds * 20,000
13 -     y(n)=Echo(x(n),delay,n_echos);
14 - end
15
16 -     sound(y);
17 -     plot(y);

FIFO.m Delay_Model.m Echo.m +
1 - function y = Echo(x, delay, n_echos )
2 -
3 -
4 -
5 -     sum = 0;
6 -     FIFO('write', x);
7 -     for i = 1:n_echos
8 -         sum = sum + FIFO('read', delay * i);
9 -     end
10 -     y = sum;
11 - end
```

```
FIFO.m Delay_Model.m Echo.m +
1 - function result = FIFO (command, arg)
2 -     if strcmp(command, 'read')
3 -         result = read_fifo(arg);
4 -     else
5 -         if strcmp(command, 'write')
6 -             write_fifo(arg);
7 -         else
8 -             if strcmp(command, 'init')
9 -                 init_fifo();
10 -            end
11 -        end
12 -    end
13 - end
14
15 - function init_fifo()
16 -     global fifo_buf write_ptr MAX_DELAY; % globalizing variables
17 -     MAX_DELAY = 20000;%Max delay of 20kHz
18
19 -     fifo_buf = zeros(MAX_DELAY,1);%buffer = array of all zeros
20
21 -     write_ptr = 1;
22 - end
23 - function val = read_fifo(delay)% Val = fifo output
24 -     global fifo_buf write_ptr MAX_DELAY;
25
26 -     % Read delayed values
27 -     % Delay read_ptr by an amount of samples, based on 'delay'
28 -     read_ptr = mod((write_ptr - 2) - delay + MAX_DELAY, MAX_DELAY)+1 ;
29
30 -     val = fifo_buf(read_ptr);
31
32 - end
33
34 - function write_fifo( val )%using output of read_fifo
35 -     global fifo_buf write_ptr MAX_DELAY;
36
37 -     fifo_buf(write_ptr) = val;
38 -     write_ptr = write_ptr + 1;%increment write_ptr by 1
39
40 -     if (write_ptr > MAX_DELAY)%Once write_ptr is greater than max delay, restart
41 -         write_ptr = 1;
42 -     end
43 - end
44
```



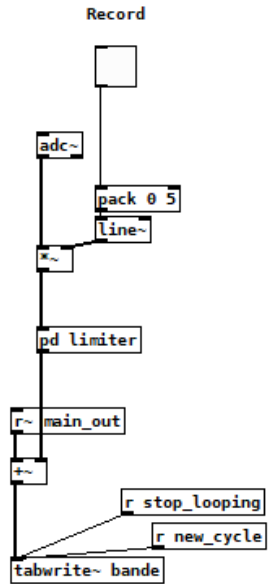
# PURE DATA LOOPER

## Features:

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

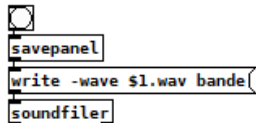


# BASIC OPERATION



This is where the recording happens.

And if you really happy with your loop,  
save it as a .wav file !

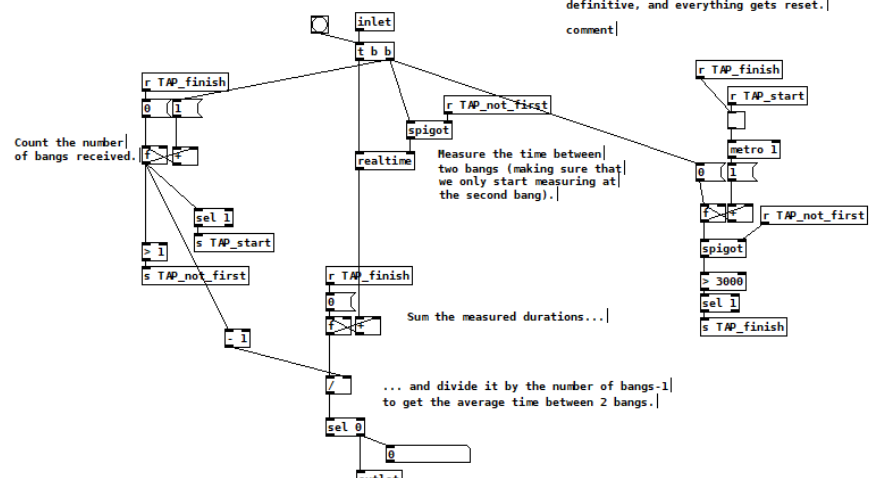


- The loop is stored in a table in Pd, which is played repeatedly.
- While this table is being played, the output and the current input is also written to it.
- If there is no input the loop copied on itself.

# DESIGN CONSIDERATIONS

"Tap tempo" : we don't measure the tempo here, but the duration between 2 beats (bangs). This is used later to set the length of the loop.

A timer gets triggered and reset every time the inlet receives a bang. If nothing happens afterwards for 3000 ms, the "tempo" is considered definitive, and everything gets reset.




- 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

 4.looper.pd - C:/Users/bryan/Documents/Professional job search/GitHub/dy...

File Edit Put Find Media Window Help

looping sample player.

click below to:

read an input file  
run the transformation  
start looping when I change something  
stop looping  
hear the output buffer again  
save the output buffer  
save normalized to max amplitude  
pd guts  
0

output meter  
100 maximum

pd input-sample  
pd output-sample

0 <- transposition up or down, 10ths of a half step  
s transposition  
0 <- loop length, hundredths of a second  
s looplevel  
0 <- start point, hundredths of a second  
s startpoint  
0 <- envelope smoothing, 0-100  
s smoothing  
pd  
10 <- set length in seconds of the output  
pd length buffer... maximum 60  
mute <-- mute button  
pd output  
90 <--set me  
s master-lvl  
LINE OUT LEVEL in dB (100 norm)

## PURE DATA SAMPLER