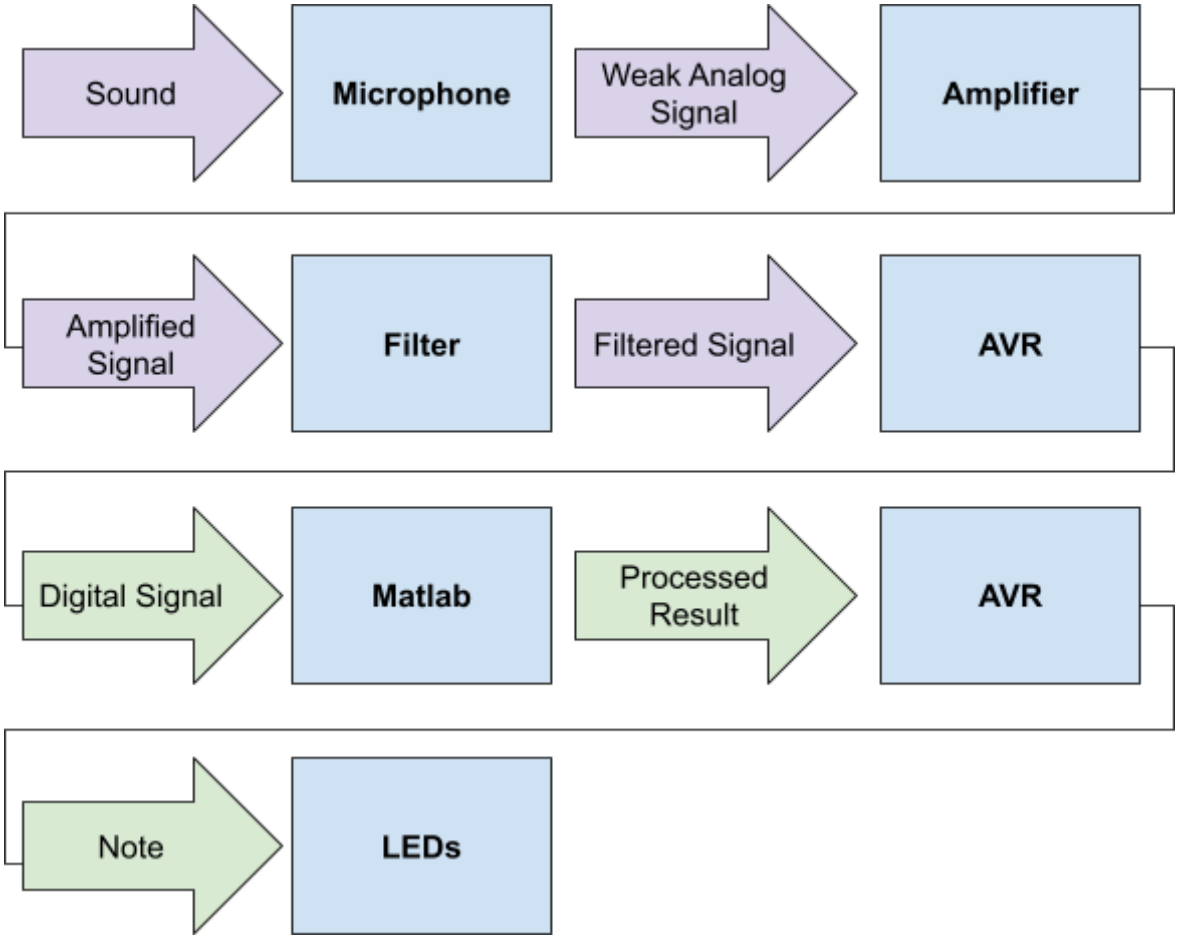
Top Level Design



Interface Definitions

Name	Types	Details
Microphone -> Amp	AC Voltage	Frequency: 0Hz to inf Hz Offset: 0V Amplitude: 2.5V
Amp -> Filter	AC Voltage	Range: 260Hz to 1575Hz Offset: 0.2V Amplitude: 1.25V
Filter -> ADC	AC Voltage	Range: 260Hz to 1575Hz Offset: 2.5V Amplitude: 2.5V
Arduino -> Matlab		Phy: 115200 baud RS-232 Protocol: Sample Buffer Message (see below)
Matlab -> Arduino		Phy: 115200 baud RS-232 Protocol: Led Command Message (see below)
Arduino -> LEDs		x8 Parallel digital bus, TTL voltage

Sample Buffer Message

The message is encoded as ASCII text. It consists of the following:

1. 170 comma-separated decimals integers containing the raw values sampled from the AVR's ADC. Numbers are not padded with zeros on the left side, and there is no comma before the first number or after the last number.

A line-feed character to signify the end of the message.

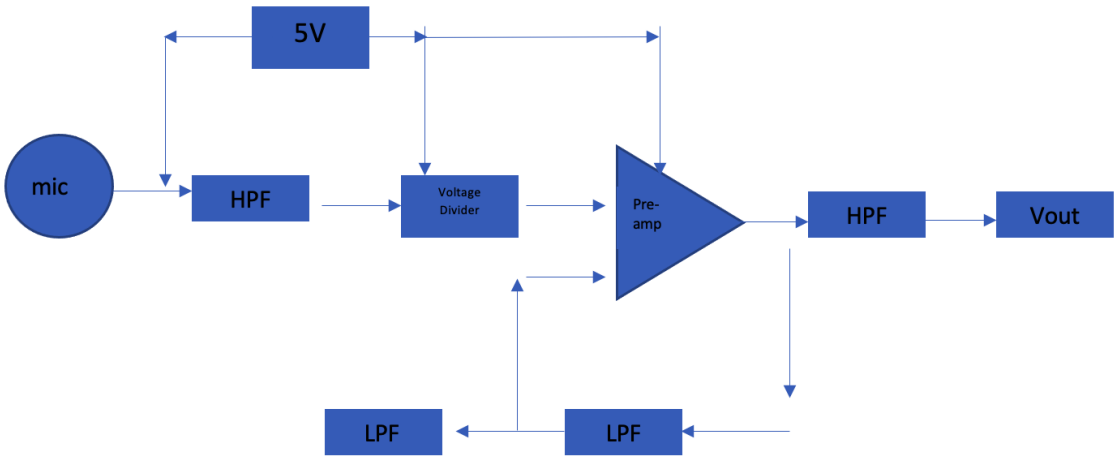
LED Command Message

The LED command message is always sent from Matlab to the Arduino, and serves two purposes:

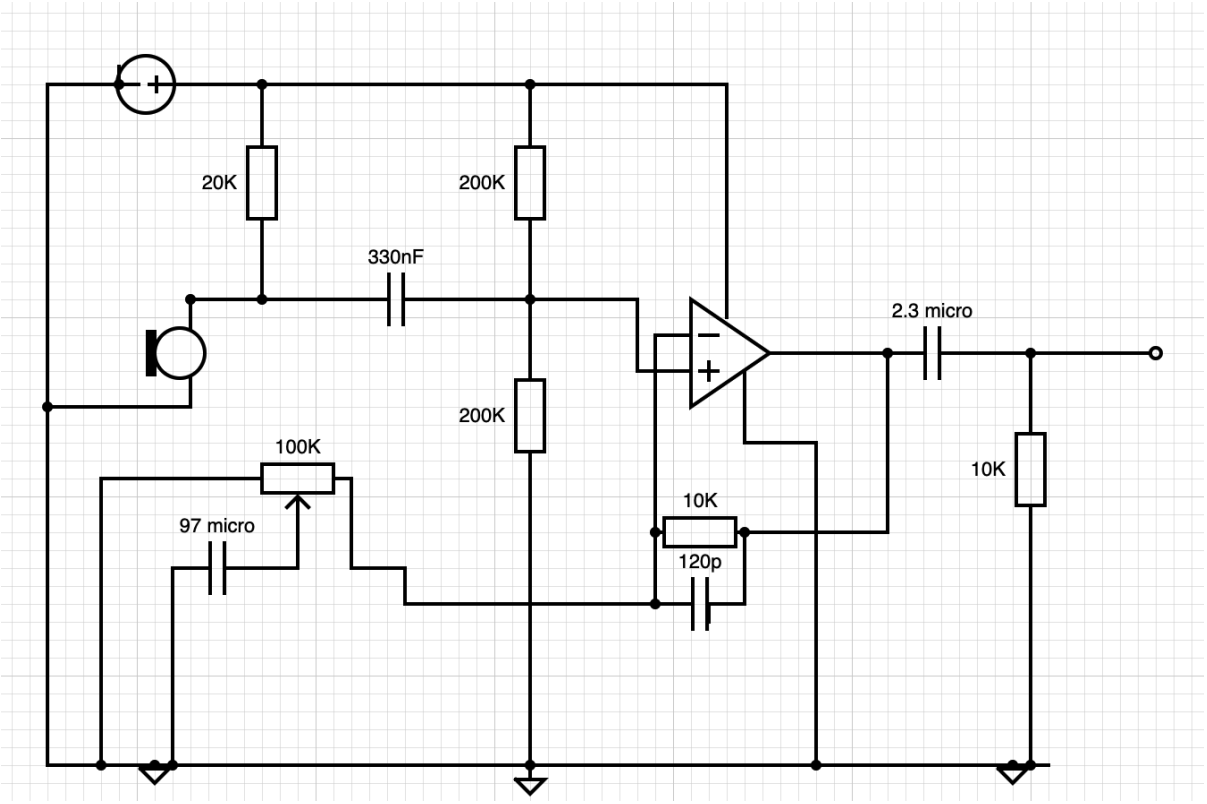
- Tell the Arduino which LED to turn on
- Let the Arduino know that Matlab is ready to accept more data and is waiting for a Sample Buffer Message

Microphone-Amplifier

Amplifier Diagram



Amp- Schematic



Parts needed:

- 330 nF Capacitor

- 2 - 47 uF Capacitors
- 120pF Capacitor
- 3.3 uF Capacitor
- 200k resistors to set a voltage divider.
- 100k potentiometer to set gain of circuit.
- 20K resistor to bias the mic
- 10k resistor as feedback for circuit.
- LMC 6032 Op Amp
- Mic for signal input

Calculation where based on equations used as in this reference document

Non-Inverting Microphone Pre-Amplifier Circuit

https://www.ti.com/lit/an/sboa290/sboa290.pdf?ts=1606250372304

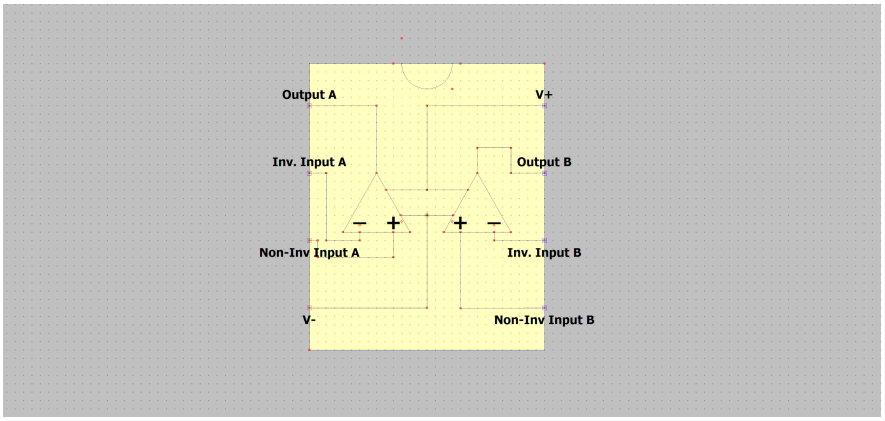
Bill of materials

Value	Datasheet		Manufacturer No.	Qty.
330nF	https://www.mouser.com/datasheet/2/427/kseries-1763315.pdf		K334M20X7RF53H5	1
47uF	https://product.tdk.com/info/en/catalog/datasheets/leadmlcc_halogenfree_fa_en.pdf		FA26X7R1E105KNU06	2
120pF	https://www.mouser.com/datasheet/2/427/kseries-1763315.pdf		K121K10X7RF5UH5	1
3.3uF	https://product.tdk.com/info/en/catalog/datasheets/leadmlcc_halogenfree_fg_en.pdf?ref_disty=mouser		FG28X5R1E335KRT00	1
200kΩ		1		2
100kΩ potentiometer		1		1
20kΩ		1		1
10kΩ		1		2
LMC6032	https://www.ti.com/lit/gpn/lmc6032		LMC6032IMX/NOPB	1
CMC-5042	https://www.cuidevices.com/product/resource/cmc-5042pf-ac.pdf		CMC-5042PF-AC	1

Filter

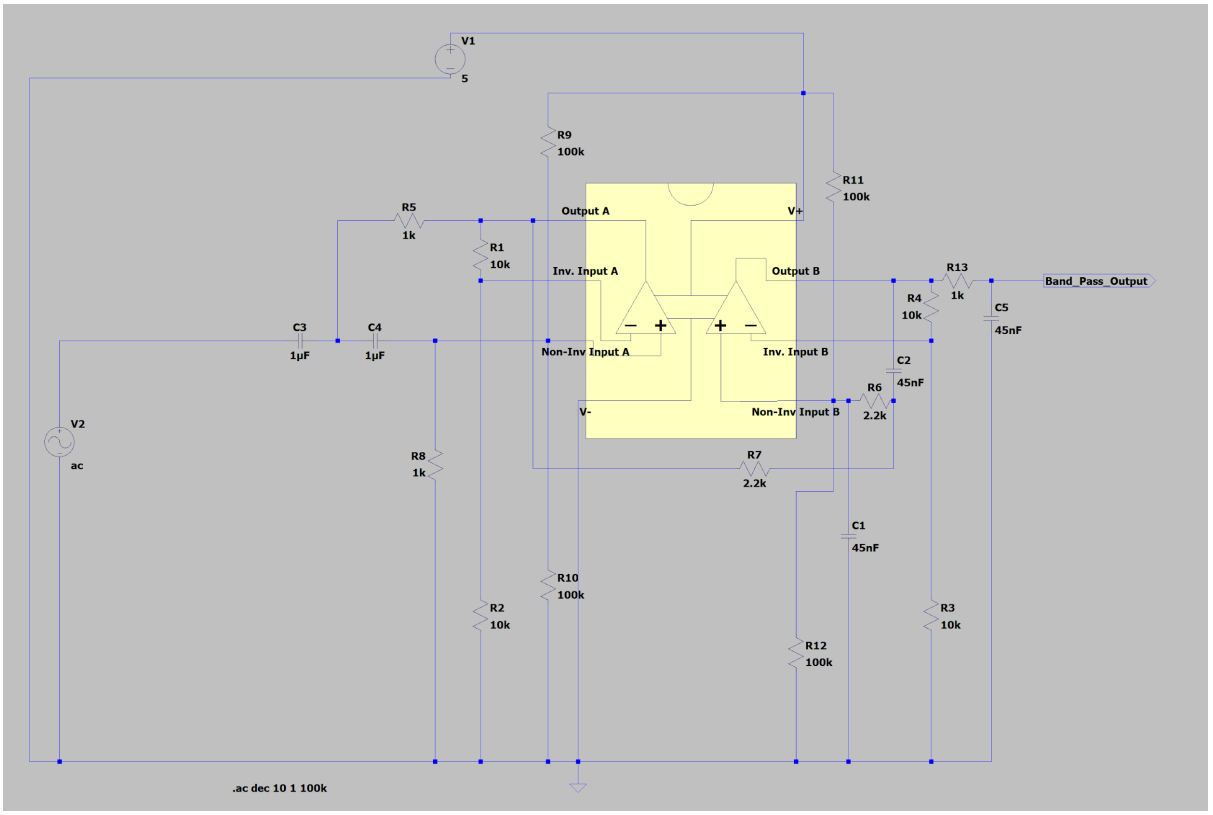
Schematic

An LTSpice schematic was created as a final design. First, the LMC6032 needs to be modeled. A picture of the LMC6032 symbol created in LTSpice can be seen on the following page.



LTSpice LMC6032 symbol

After generating an LTSpice model for the LMC6032, the schematic for both filters was created as shown below:



A final single order low-pass filter was added to the output of the filter schematic, just before "Band_Pass_Output" to quench high frequency noise that was showing up in the AC signal response.

AVR Firmware

```
/*-------------------------------------------------
//  OSU JD AP sampling firmware
//  By Anton Liakhovitch
//  Copyright Febuary 2021
//
//  This software is licensed under WTFPL v2.0,
//  except where limited by Oregon State University
//  policy for assignment code.
-------------------------------------------------*/


// Globals

// Size of buffer, in samples
#define BUFF_SIZE 170
// Serial baud rate
#define BAUD_RATE 115200

// Sample buffer
uint16_t sample_buff[BUFF_SIZE];
// Sample iterator
uint16_t sample_count = 0;
// Semaphore flag to signal end of sampling
volatile uint8_t sample_flag = 0;
// Pin numbers for LEDs
const int ledpins[8] = {2,3,4,5,6,7,8,9};


// Reads ADC value
int16_t read_adc() {
  // Start measurement
  ADCSRA = (ADCSRA | (1 << ADSC));
  // Wait for measurement to finish
  while (bit_is_set(ADCSRA, ADSC));
  // Return result
  return ADC;
}


// Timer ISR, triggers at 10989Hz and samples ADC
ISR (TIMER0_COMPA_vect){
  // Read ADC to buffer
  sample_buff[sample_count] = read_adc();
  // Increment the sample index
  sample_count += 1;
  // If buffer is full:
  if(sample_count >= BUFF_SIZE) {
    // Set ready flag
    sample_flag = 1;
    // Disable timer interrupt
    TIMSK0 &= ~(1 << OCIE0A);
  }
}


// Collect data:
// Enable the data collection ISR and wait for data collection
//  to finish.
void collect_data(){
  // Reset buffer iterator
  sample_flag = 0;
```

```
  // Reset semaphore
  sample_count = 0;
  // Enable output compare interrupt so ISR can do its thing
  TIMSK0 |= (1 << OCIE0A);
  // Wait for ISR to signal full buffer with semaphore
  while(!sample_flag);
}


// Clear serial input buffer
// We do this a little while after receiving a command, in order
//  to drop any trailing newline characters that were sent.
// Otherwise, buffered text would immediately be interpreted as
//  another command.
void clear_input_buff() {
  while (Serial.available() > 0) Serial.read();
}


// Sends the sample buffer over serial
// Format is comma delimited text -
//   The entire sample buffer is printed on one line;
//   There is no comma at the end of the line;
//   The line is terminated by a single LF character;
//   First sample collected is first sample sent (FIFO).
void send_data(){
  // Iterate over sample buffer
  for(uint16_t i = 0; i < BUFF_SIZE; i++) {
    // Send value as text
    Serial.print(sample_buff[i]);
    // Don't send comma after last value
    if(i != BUFF_SIZE-1) Serial.print(',');
    // Clear sample buffer -- not necessary but good for debugging
    sample_buff[i] = 0;
  }
  // Terminate line
  Serial.print('\n');
  // Wait for transmission to finish
  Serial.flush();
}


// Turn on one of the eight LEDs, turn off the rest.
// Input is an integer from 0 to 7.
void set_led(uint8_t led_num){
  for(uint8_t i = 0; i < 8; i++){
    if (i == led_num){
      digitalWrite(ledpins[i], HIGH);
    }else{
      digitalWrite(ledpins[i], LOW);
    }
  }
}


// Turn off all LEDs
void clear_leds(){
  for(uint8_t i = 0; i < 8; i++){
    digitalWrite(ledpins[i], LOW);
  }
}
```

```
// Receive and execute an LED switch command
// Command is a single ASCII character -
//   A number from 0-7 is an LED switch command;
//   Anything else is an LED shutoff command;
//   Trailing newlines are dropped.
void receive_command(){
  // Wait for command character
  while (Serial.available() == 0);
  // Read in command character
  char ascii = Serial.read();
  // Execute command
  if(ascii >= 48 && ascii <= 55){
    set_led(ascii - 48);
  }else{
    // If we got anything but a valid LED number,
    // clear LEDs
    clear_leds();
  }
}


// One-time setup function
void setup() {
  // Configure status LED
  pinMode(13, OUTPUT);
  // "Real" embedded C developers would cringe,
  // But the Arduino libraries ARE here;
  // And I got 'em, so I WILL flaunt 'em.
  for(uint8_t i = 0; i < 8; i++){
    pinMode(ledpins[i], OUTPUT);
  }

  // Configure Serial
  Serial.begin(BAUD_RATE);

  // -------------
  // Configure ADC
  // -------------

  // ADC MUX
  ADMUX = (\
  // Use VCC as analog reference
  (0 << REFS1)|(1 << REFS0)| \
  // Right-pad result
  (0 << ADLAR)| \
  // Use ADC0 pin
  (0 << MUX3)|(0 << MUX2)|(0 << MUX1)|(0 << MUX0));

  // ADC Control and Status Register
  ADCSRA = (\
  // Enable ADC
  (1 << ADEN)|\
  // Disable auto trigger
  (0 << ADATE)|\
  // Disable ADC interrupt (we will use a busy loop)
  (0 << ADIE)|\
  // Set the prescaler to x64
  (1 << ADPS2)|(1 << ADPS1)|(0 << ADPS0));

  // --------------------
  // Configure TIM0 Timer
  // --------------------
```

```
  // Timer/Counter Control Register A
  TCCR0A = (\
  // Disable output compare pin toggling
  (0 << COM0A1)|(0<<COM0A0)|(0 << COM0B1)|(0<<COM0B0)|\
  // Set timer mode to CTC
  (1 << WGM01)|(0 << WGM00));

  TCCR0B = (\
  // Finish setting timer mode to CTC
  (0 << WGM02)|
  // Set prescaler to 8
  (0 << CS02)|(1 << CS01)|(0 << CS00));

  // We will be counting to 182 for appprox. 11KHz (10989 Hz) frequency
  // Set up output compare
  OCR0A = 182;

}

// Main program loop
void loop() {
  // Set status LED
  digitalWrite(13, HIGH);

  // Collect data into sample buffer
  collect_data();

  // Clear extra characters (newlines, etc)
  // from serial input buffer
  clear_input_buff();

  // Send sample buffer over serial
  send_data();

  // Clear status LED
  digitalWrite(13, LOW);

  // Wait for a command from Matlab
  receive_command();

}
```

Matlab Code

File: “main.m”

```
%  OSU JD AP host program
%  By Anton Liakhovitch
%  Copyright March 2021
%
%  This software is licensed under WTFPL v2.0,
%  except where limited by Oregon State University
%  policy for assignment code.

% Constants
% Size of sample buffer (in samples)
buff_size = 170;
% Sampling frequency
sample_freq = 10989;
% Size of the fourier transform
fft_size = 2048;
% Ignore frequencies below this one
start_freq = 250;
% Ignore frequencies above this one
end_freq = 600;
% Note frequency table. Table index corresponds to LED code
note_freqs = [261.63, 293.66, 329.63, 349.23, 392.00, 440.00, 493.88, 523.25];

% Calculate indicies of start and end frequencies
start_freq_index = ceil(start_freq*fft_size/sample_freq);
end_freq_index = floor(end_freq*fft_size/sample_freq);

% Calculate map of fourier buckets -> frequencies
frequencies = 0:(sample_freq/fft_size):sample_freq/2;
% Calculate map of samples -> time (microseconds)
time_axis = 0:1/sample_freq*1000000:(buff_size/sample_freq*1000000)-1;

% Setup the serial connection
if exist('port', 'var') == 1
    clear port
end
port = setup_serial(input("Enter serial port path:",'s'));

% Button to stop data collection
% Variable: should data collection be running?
% Using global is bad practice but acceptable here
global runstate;
runstate = true;
% Create UI
window = uifigure;
btn = uibutton(window, 'push', 'Text', 'Stop', 'ButtonPushedFcn', @(btn,event)
stop_button_pushed());

% Create led command variable
% Default is no LED lit
led_command = uint8(8);

% Main loop
while runstate == true
    % Retrieve sample buffer from Arduino
    buffer = get_buffer(port,buff_size,led_command);
    % Compute the Fourier transform
    transform = fft(buffer, fft_size);
```

```
    % Discard useless upper half of transform
    transform = transform(1:length(transform)/2+1);
    % Compute the point magnitudes
    magnitudes = abs(transform)/buff_size;
    % Find base frequency
    [M, I] = max(magnitudes(start_freq_index:end_freq_index));
    % Adjust index
    I = I+start_freq_index-1;

    % Find the closest note to our frequency
    [p,led_command] = min(abs(note_freqs-frequencies(I)));
    % Fix Matlab's weird indexing
    led_command = led_command-1;

    % Display frequency for debugging
    fprintf("Frequency: %f, LED number: %i\n", frequencies(I), led_command);

end

% Turn off LEDs
buffer = get_buffer(port,buff_size,8);

% Close serial connection
clear port

% Close stop button window
close(window);

% Print SNR
disp("Signal to noise:");
disp(snr(double(buffer)));

% Raw data graph
subplot(1,2,1)
plot(time_axis, double(buffer)*(5/1024));
xlabel("Time (Microseconds)");
ylabel("Magnitude (V)");
title("Audio data");

% Frequency spectrum graph
subplot(1,2,2)
plot(frequencies, magnitudes);
xlabel("Frequency (Hz)");
ylabel("Magnitude");
title("Frequency Distribution");

% Utility functions

% Callback function for button
function stop_button_pushed()
    global runstate
    runstate = false;
end
```

File: “setup_serial.m”

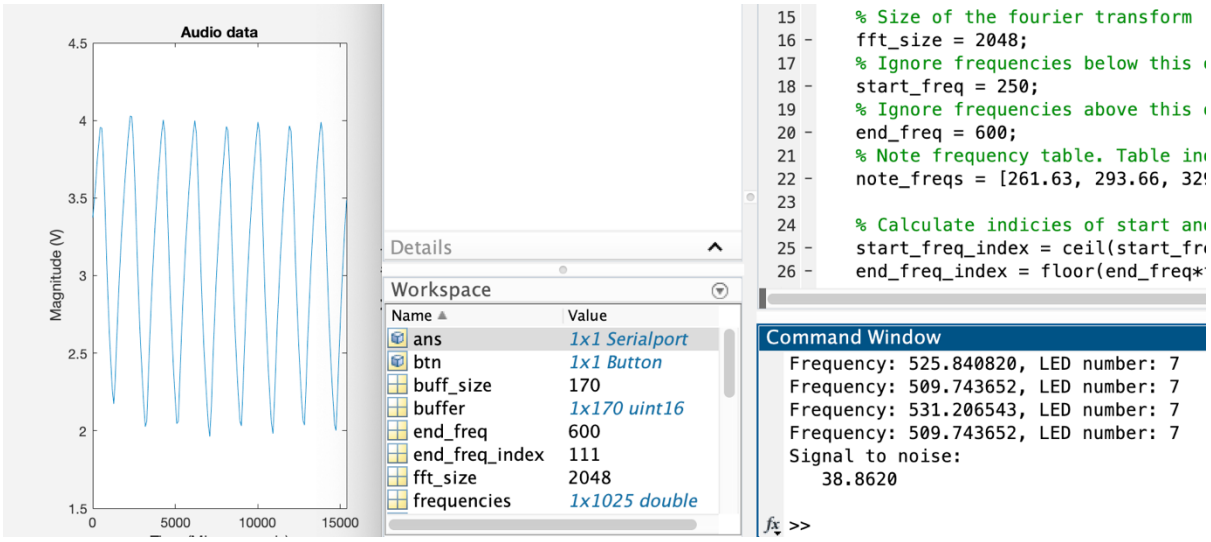
```
%  OSU JD AP serial communication driver
%  By Anton Liakhovitch
%  Copyright March 2021
%
%  This software is licensed under WTFPL v2.0,
%  except where limited by Oregon State University
%  policy for assignment code.

function [port] = setup_serial(portname)
% Input: path to port (eg, "COM0", "/dev/ttyUSB0")
% Output: serial object

% Validate arguments
arguments
    portname string = "/dev/ttyUSB0"
end

% Baud rate constant
baud = 115200;

% Initialize and open port
% Note: this resets the Arduino!!! Do this once per program run.
port = serialport(portname, baud, "Timeout", 10000);
configureTerminator(port, "LF");

% Wait for Arduino to send its first data line, and then ignore it.
% This is needed to ensure Arduino is in a known state.
readline(port);

end
```

File: “get_buffer.m”

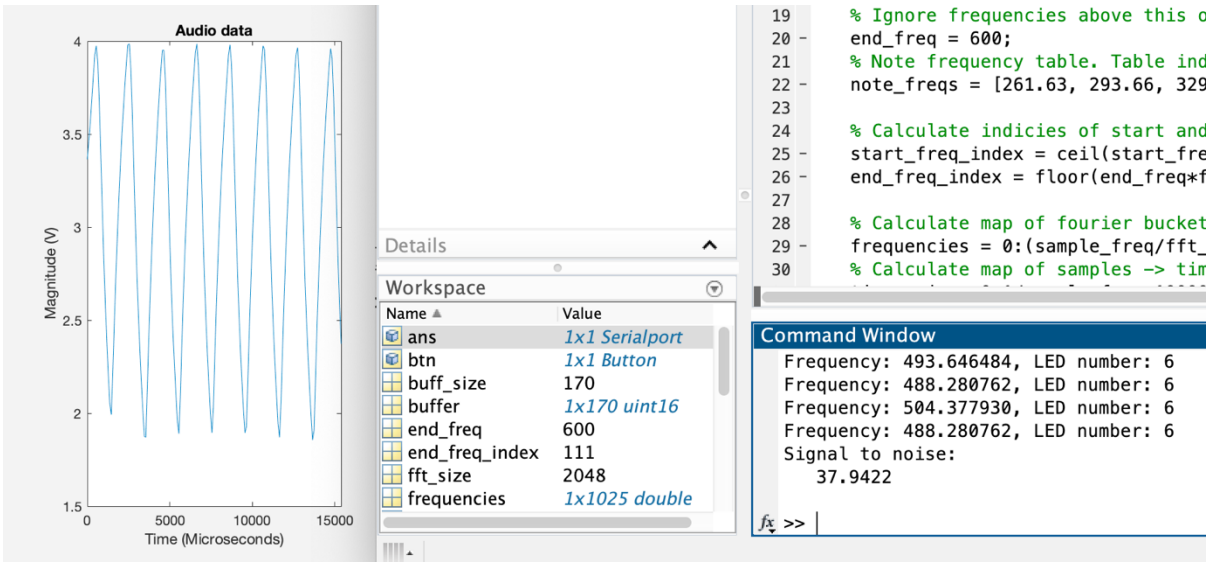
```
%  OSU JD AP serial communication driver
%  By Anton Liakhovitch
%  Copyright March 2021
%
%  This software is licensed under WTFPL v2.0,
%  except where limited by Oregon State University
%  policy for assignment code.

function [buffer] = get_buffer(port, buffsize, ledcommand)
%Inputs: serial object, buffer size, led command
%Output: sample buffer array

% Argument validation
arguments
    port
    buffsize uint16 = 170
    ledcommand uint8 = 0
end

% Flush the port buffer
flush(port);

% Write command to Arduino
writeline(port,sprintf('%i',ledcommand));

% Read a line of ASCII text from arduino
line = readline(port);

% Split string along delimiter into substring array
strings = strsplit(line, ',');

% Assert length of packet
if length(strings) ~= buffsize
    error("Wrong number of samples received!");
end

%buffer = cast(strings, "uint16");
buffer = str2double(strings);
buffer = uint16(buffer);
```

Raw Results

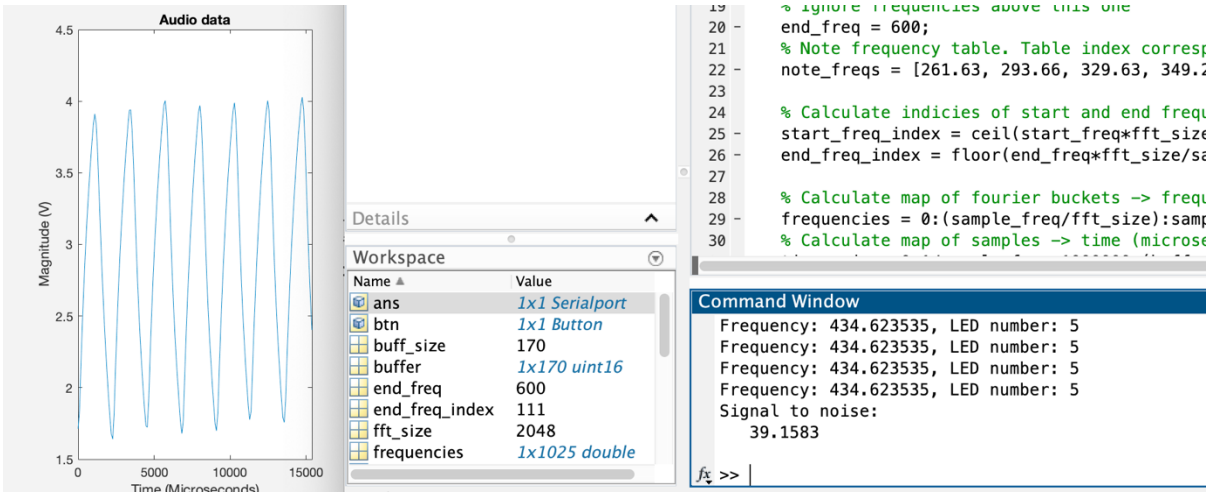
At 520hz signal input:



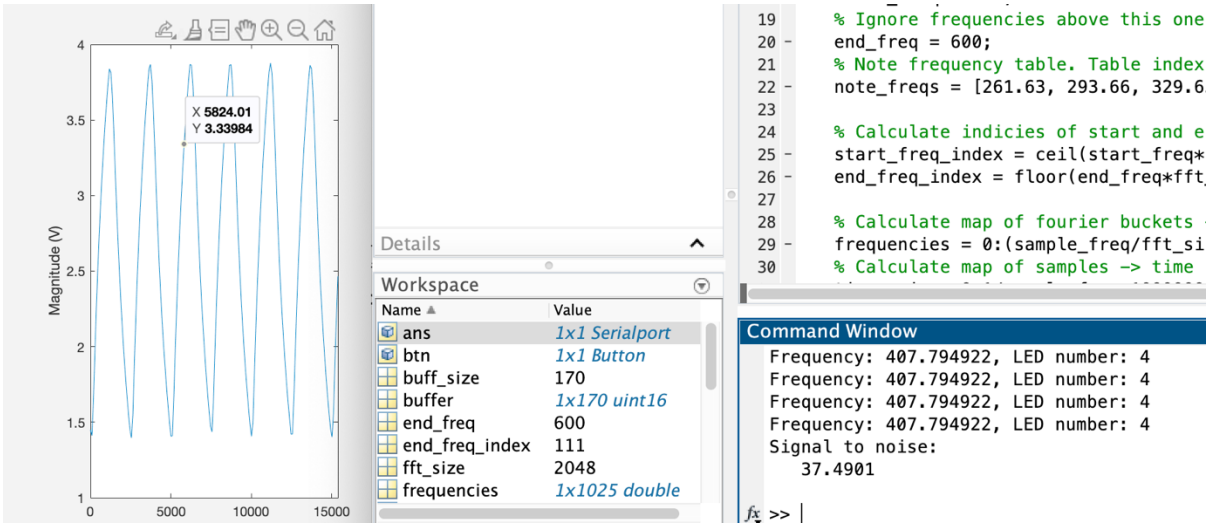
At 500hz signal input:



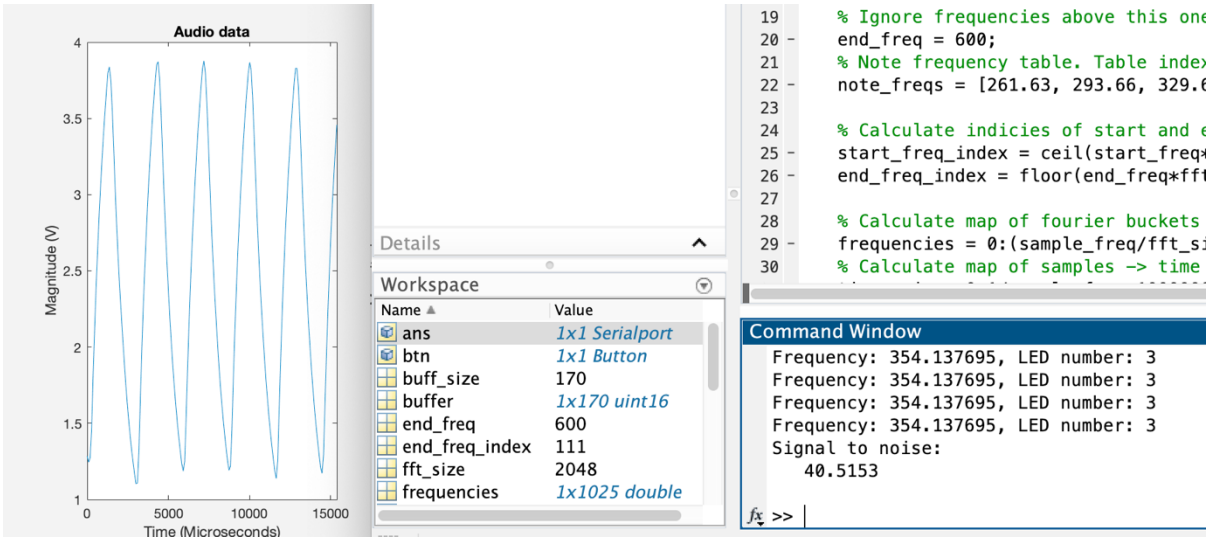
At 430hz signal input:



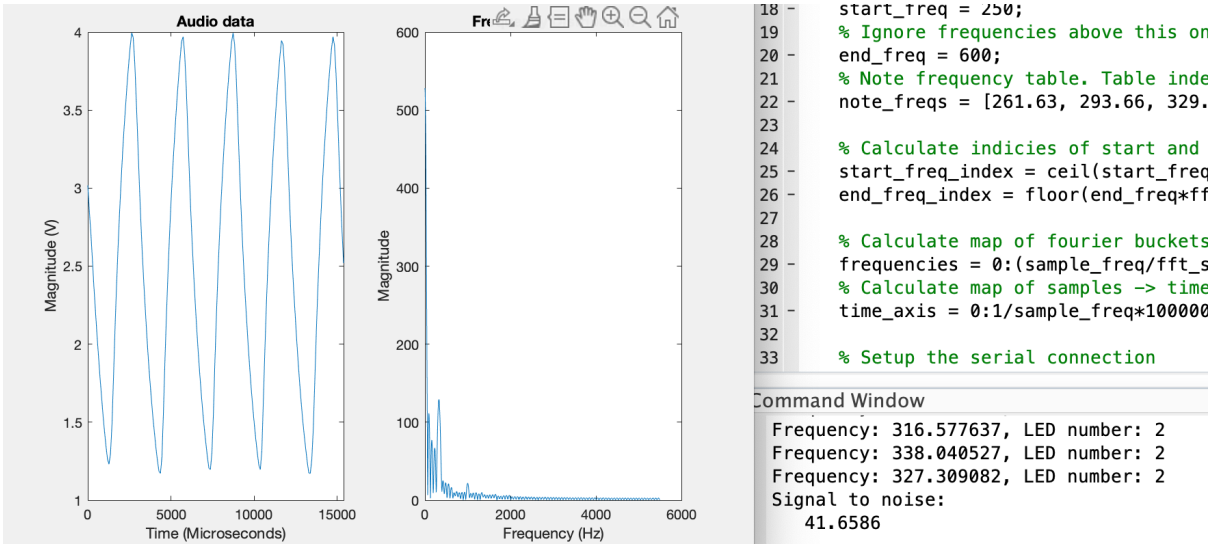
At 400hz signal input:



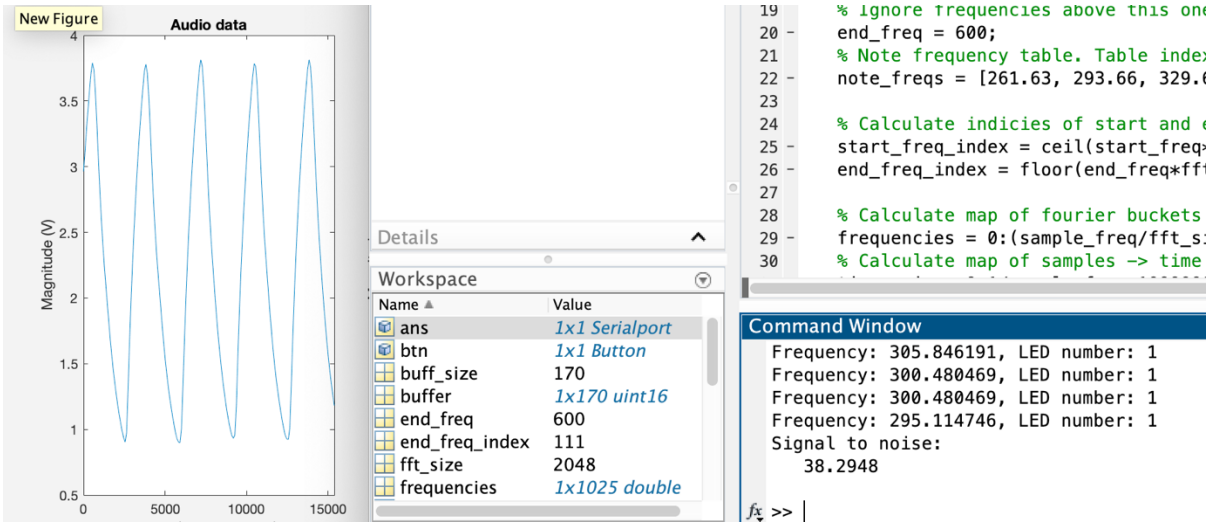
At 350hz signal input:



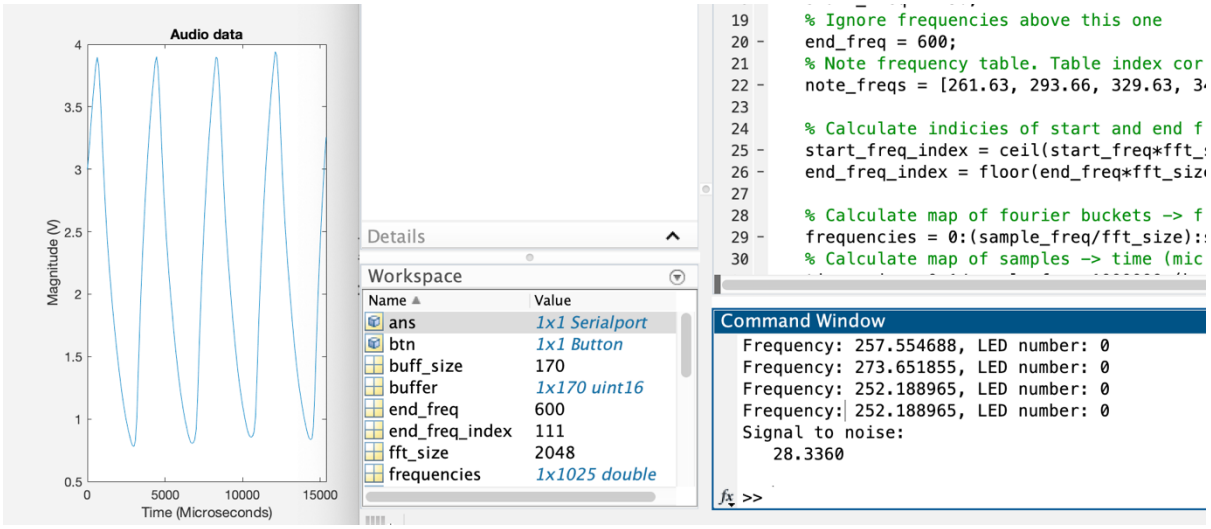
At 330hz signal input:



At 300hz signal input:



At 260hz signal input:



While no input signal: