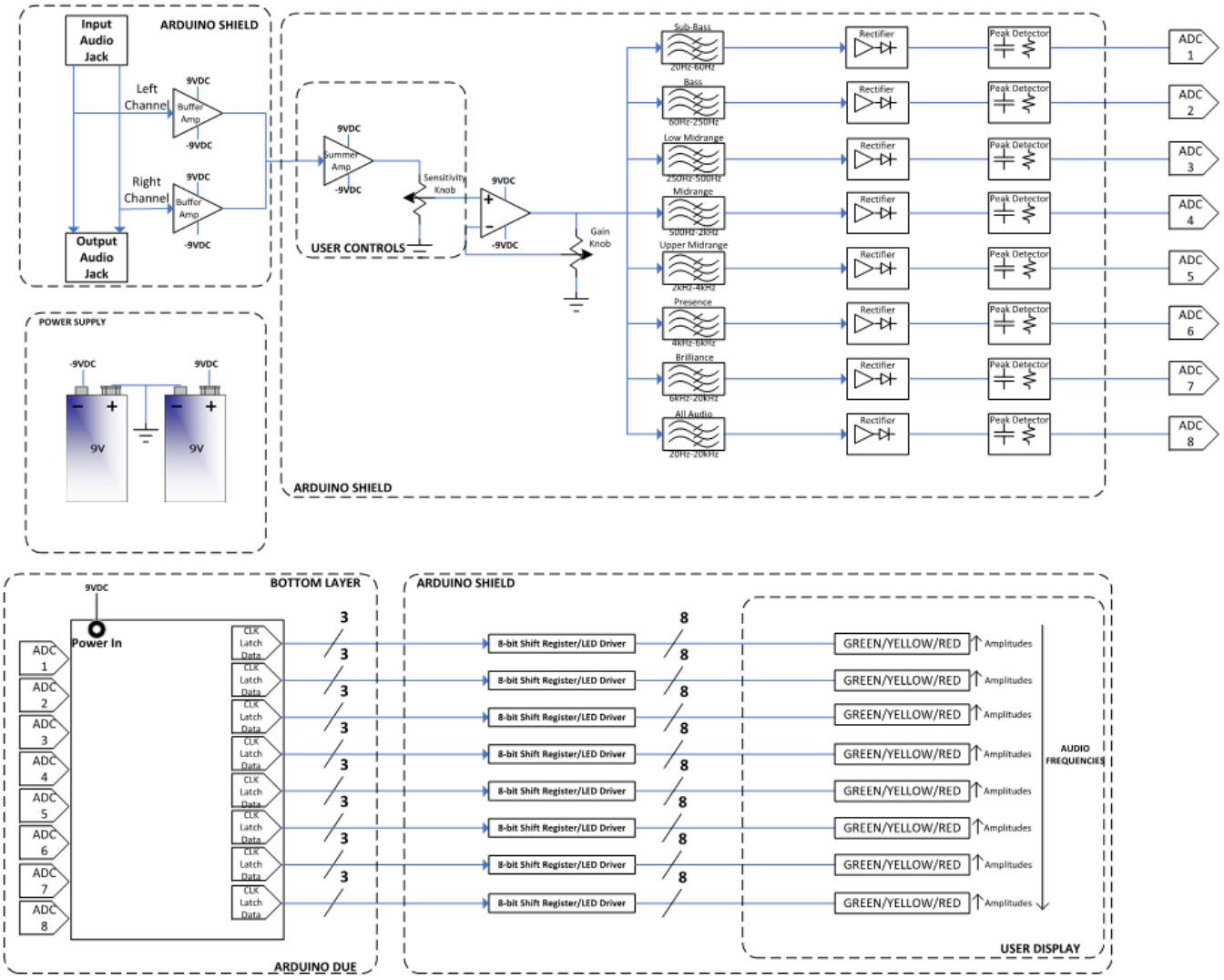
Team 10 - Volume Unit Meter

Ethan Nagelvoort (821234668)
Robert Esposito (822379994)
Carlos Callejas Dominguez (821226920)
Ardiana Krasniqi (820354659)

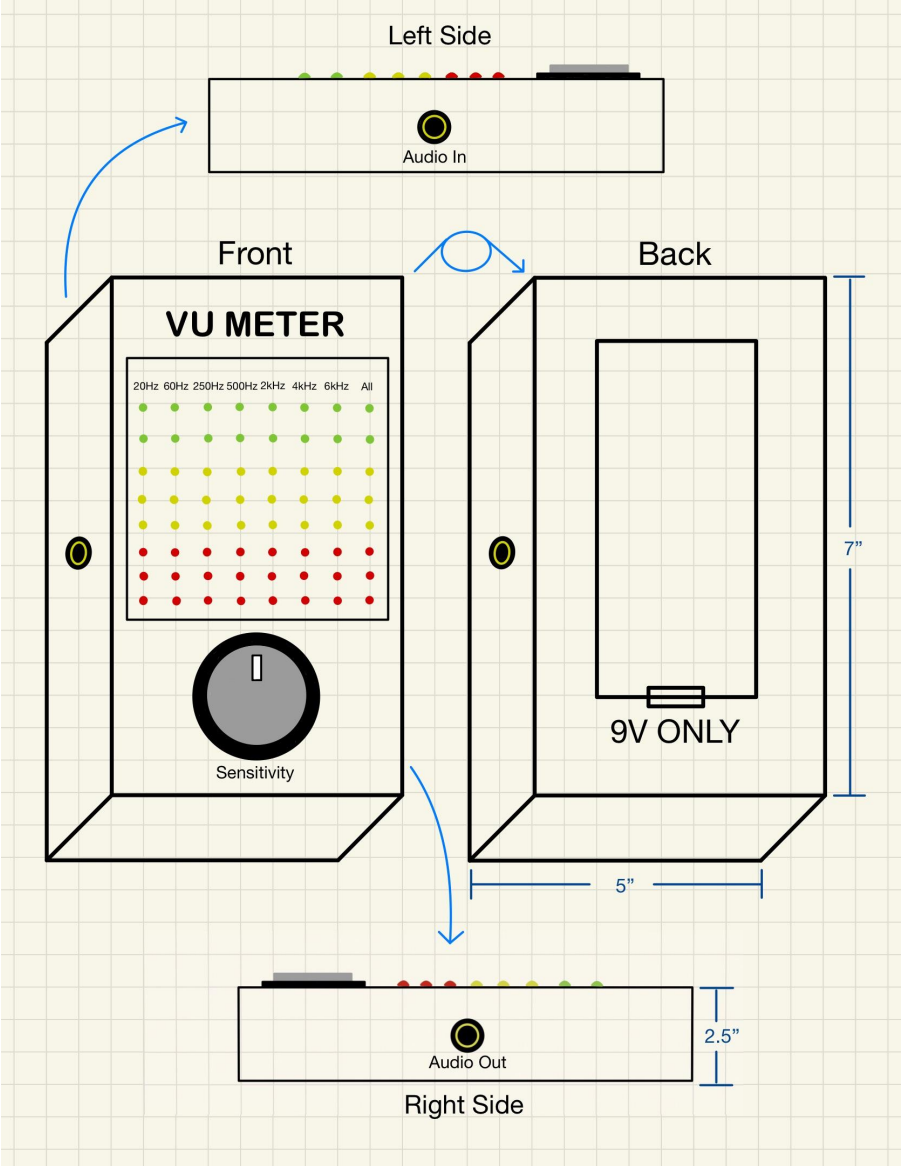
System Description

Determining how loud your music is can be difficult, but having a fun gizmo to measure the relative loudness in different frequency bins of the music can make it more amusing. For our Senior Design project, we will be creating a Volume Unit Meter that will allow a user to plug in an audio source and show the relative loudness in different frequency bins within the signal. The system will operate using eight columns of LEDs, with eight LEDs per column. In total there are going to be sixty-four LEDs. Each column of LEDs will be split into three colors. The first three LEDs in the column will be red, the next three will be yellow, and the last two will be green. The red LEDs represent low volume, the yellow LEDs represent mid volume, and the green LEDs represent high volume. The bottom LED in each column will represent -21dB, which is the lowest decibel level detectable by the device. The top LED in each column will represent 0dB, which is the highest decibel level. Between 0 dB and -21 dB, each LED reflects a 3dB increase or decrease. The device will have a sensitivity knob, implemented using a log potentiometer, which allows the user to adjust the relative loudness accordingly. Each LED column will represent different frequency bins that can be found in an audio signal between 20 Hz and 20 kHz. The first column of LEDs will represent the sub bass range between 20 Hz and 60 Hz, the second column will represent the bass range between 60 Hz and 250 Hz, the third column will represent the lower midrange between 250 Hz and 500 Hz, the fourth column will represent the midrange between 500 Hz and 2 kHz, the fifth column will represent the upper midrange between 2 kHz and 4 kHz, the sixth column will represent the presence range between 4 kHz and 6 kHz, the seventh column will represent the brilliance range between 6 kHz and 20 kHz, and finally the eighth column will represent the entire frequency range between 20 Hz to 20 kHz. These are the common frequency bins used by audio engineers on home audio equalizers. This device takes in the audio signal through a stereo 3.5mm input audio jack that can be connected to an audio source such as an iPhone, a laptop, or a microphone. The stereo input audio jack will provide audio signals to the left and right channels of the Volume Unit Meter. Two high input impedance amplifiers are used as buffers and will ensure that the input impedance does not affect the summation circuit they are attached to. The summation circuit will then combine the two channels into a single channel. Once the two channels have been summed together, they will go into a sensitivity knob circuit followed by a gain stage which will produce an ideal signal for filtering. The signal is then split between eight parallel bandpass filters, each of which will be designed to capture our different frequency bins. Once the signal has been split into these eight bins, they will each enter an ideal rectification circuit followed by a peak detector circuit which will determine the value of the different peaks in the amplitude of the audio signal. The device

will implement an Arduino Due microcontroller that will be programmed to read the outputs of the eight peak detectors through eight of its analog input pins. The Arduino Due samples the DC values from each peak detectors, determines a decibel value referenced to 3.3V, and sends this value as serial data to shift registers to control the LEDs. We will be using a technique called "Bit Banging" to multiplex serial data to the shift registers which control the sixty-four LEDs. The shift registers will share universal clock and latch signals that will come from two of the microcontroller's digital output pins to ensure everything is synced to the audio signal correctly. Each column of LEDs will be driven by the shift registers which will act as LED drivers. Lastly, a 3.5mm stereo output audio jack will connect in parallel with the 3.5mm stereo input audio jack to provide the user the ability to listen to the audio signal being fed through the system. Currently we have the system being powered by two 9V batteries. However, this will be replaced by a 9V DC wall wart in the future.



Block Diagram



Sketch

System Validation

1) Set a function generator to generate a sinusoid signal at 0.5 Vpeak at 1kHz.
2) Connect speaker to output audio jack and verify the 1kHz tone is heard.
3) Set a function generator to generate a 40 Hz sinusoid signal.
4) Adjust sensitivity knob until all the LEDs of the 20Hz column are turned ON.
5) Make sure the remaining frequency bins LEDs remain OFF.
6) Adjust sensitivity knob down by 3dB and verify that the LEDs are turned OFF one by one until no LEDs are turned ON. Make sure that the remaining frequency bins LEDs are OFF.
7) Repeat the steps above for the following frequencies: 155Hz, 375Hz, 1.25kHz, 3kHz, 5kHz, and 13kHz.

Risk table

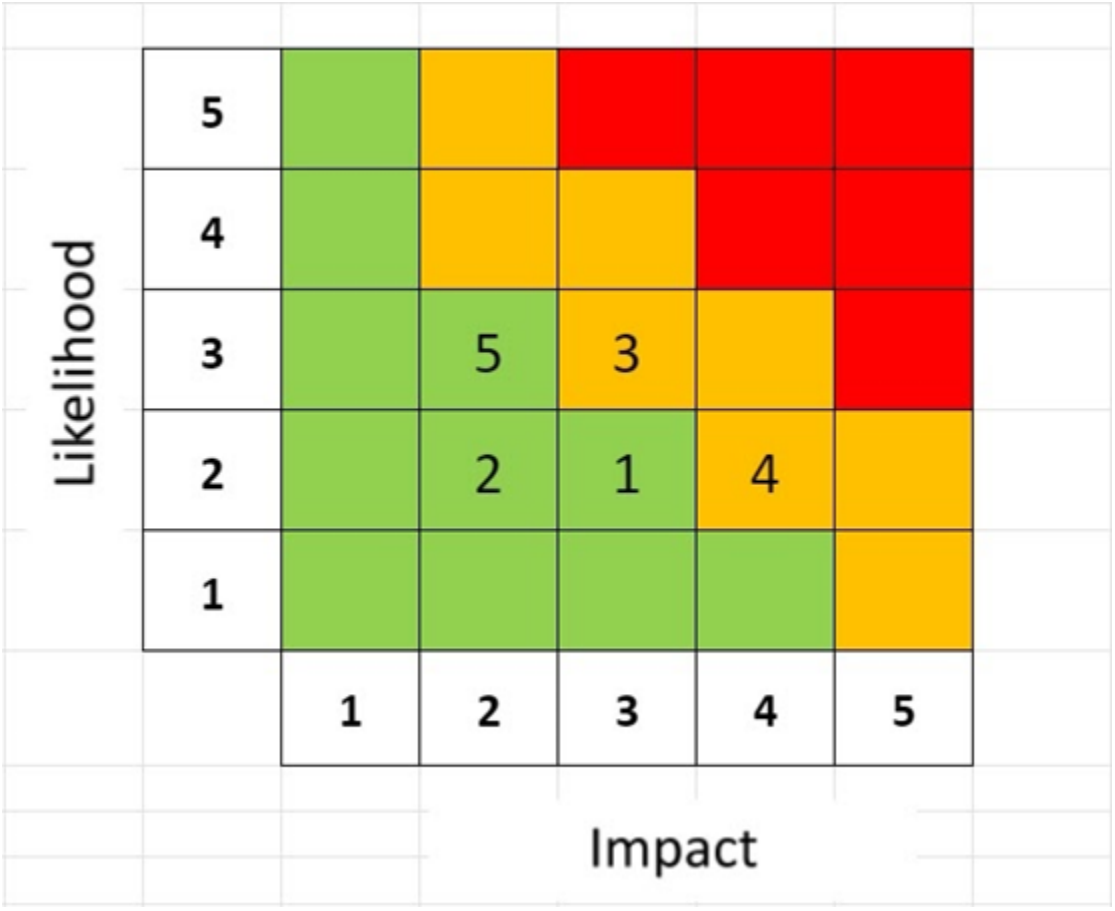
Risk #	Issue *Describe the risk and How it could adversely Affect the project goal*	Measures to minimize probability of occurrence *Describe actions that have been or will be taken to minimize the probability of occurrence of the risk.*	Measures to mitigate impact *Describe actions that will be taken to mitigate the effects of the risk on the project goals if the risk occurs*	Risk likelihood *(1-least likely, 5-most likely)*	Risk impact *(1-least impact, 5-most impact)*
1	**Processing Time**: Our project requires us to sample from eight analog pins via a shared ADC, convert those samples to a dB scale, and output the results to eight shift registers via "Bit Banging". These tasks together are processing intensive and the risk is that our Arduino Due will not be able to complete all of these tasks within one period of our 10kHz sample rate.	We will write pieces of code that emulate each of the tasks we are asking our Arduino to perform. We will use internal timers, along with an oscilloscope where applicable, to time the results of each process. If the sum of these processes times is less than our sample period, we can be confident that our processing will complete on time. For our ADC sampling, we write a piece of code that will cycle through sampling all eight analog channels and use internal timers to time how long each ADC conversion takes. For the dB conversion, we will write a piece of code to convert an ADC sample to our dB scale and use internal timers to time how long the conversion takes.	If our Arduino cannot complete the processing in the given sample period (1/10kHz), we can lower the sample rate down until it is able to complete on time. Since the samples are DC peak values, we do not have to sample at the Nyquist Frequency of our audio input (48kHz). Instead, we can sample at rates as low as 5kHz and still produce an accurate representation of the input signal. We could go even lower than 5kHz if necessary, but aliasing would start to become a significant problem.	2	3

		For “Bit Banging” to the shift registers, we will write a piece of code that will “Bit Bang” sample values to our shift registers and use internal timers to time how long it takes to send all of the data. We will also use an oscilloscope to verify that the clock signal our Arduino will be generating for our shift registers falls within the setup and hold time specifications of the shift registers.			
2	**Timers**: In order to multiplex 8 LEDs using one digital output pin, shift registers are needed to drive the LEDs. Our project uses eight shift registers to multiplex 64 LEDs. These shift registers will share a clock signal and a latch signal. The risk is that if the timing of these clock and latch signals aren’t precise, our LEDs will not light up correctly.	We will set up a timer to toggle one of the Arduino Due’s digital pins on and off when the timer starts and when it resets. We can then use an oscilloscope to verify the timer was set correctly so it can generate the appropriate clock signal or adjust the timing as needed. We can also prototype this multiplexing using the built in Arduino function shiftOut(). This function takes in an output pin, a clock pin, the bit order, and a value as arguments and sends out the value one bit at a time in sync with the clock signal. However, this function uses busy waits to generate the clock signal which will greatly impact our	If we do not get our timer to work properly, we can use the built in Arduino function shiftOut(). We will lower our sampling rate to accommodate the busy waits this function uses while still meeting our processing time requirements.	2	2

		processing time. We can replace pieces of this function with our own code until we have completely replaced it and achieved the same functionality without tying up the CPU with busy waits.			
3	**Sampling Rate**: Selecting an invalid sampling rate will cause the LEDs to no work as intended.	To minimize probability the peak detector provides a constant DC voltage at the inputs of the 8 ADC channels. This will allow for us to sample at a lower rate. To select an appropriate sample time, toggling a digital pin is required to find the time for the following processes: 8 ADC channels, 8 DC to decibel conversations, and 8 multiplexed shift registers. The use of an oscilloscope will be required to probe the output of the digital pin that is being toggled. Once all the processes have been timed, we take the sum of all the times to obtain a sample rate that will work for our system.	To mitigate impact, if the overall time exceeds one sample period of 10kHz, a lower sampling frequency can be selected.	3	3
4	**Active Bandpass Filter Design**: Capacitor and resistance tolerances will shift the 3dB cutoff frequencies of all	To minimize probability, we can select capacitors and resistors with low tolerance values however, this will increase cost.	To mitigate impact a Monte Carlo analysis in MATLAB of about 3000 iterations will be performed to show how the 3dB cutoff	2	4

	the active bandpass filters. The 3dB cutoff frequency must be within ±5 Hz. This will allow the center frequency of one bin to not interfere on its neighbor frequency bin. In other words, if an audio tone was set to the center frequency of one bin, this tone should only turn ON the LEDs on this bin not its neighbor.		frequency will shift depending on the tolerances of capacitors and resistors. Also, the upper and lower tolerances of each component can be simulated in LTSpice.		
5	**Peak Detector Attack/Release Times**: Inadequate attack and release times will not capture the peaks of the incoming audio signal and will not allow the ADC to sample the DC values correctly.	To minimize probability, selecting a high attack time with a low release time will allow our peak detector to detect the peaks of the incoming signal and hold it long enough for the ADC to sample it. Also, the attack time will depend on the sampling rate of the ADC.	To mitigate impact a sampling time must be determined first to select an appropriate attack time. Once an attack time and a release time are selected, use a function generator to generate a 10kHz sinusoidal signal at 1Vpeak at the input of the rectification circuit. Turn the signal ON and OFF quickly and verify the output of the peak detector with an	2	3

			oscilloscope.		



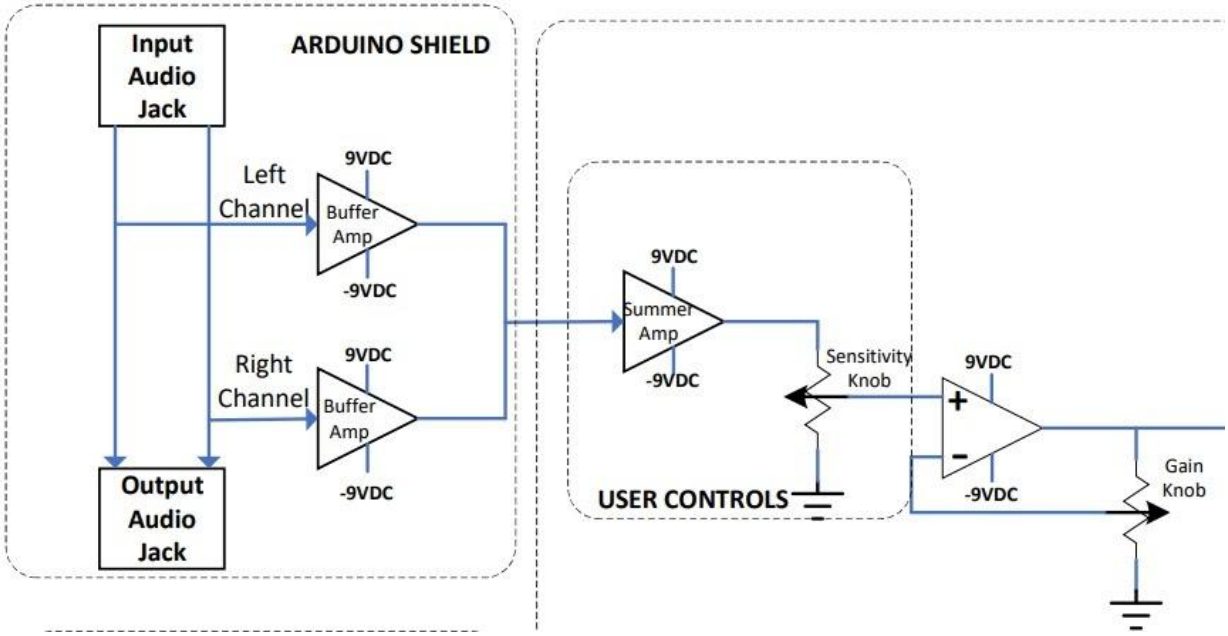
Risk Cube

Key Integration Task

Task: Input Audio to Summation Circuit

Description: The system converts a two-channel audio signal into a signal channel that is optimized for filtering. To do this, it uses high input impedance amplifiers and a summation circuit to combine these two channels into one.

Validation: Use a function generator to produce a 1V sine wave at 1 kHz in both channels of the input. Then use an oscilloscope to see if the output of the summation circuit produces a 2V sine wave at 1 kHz.

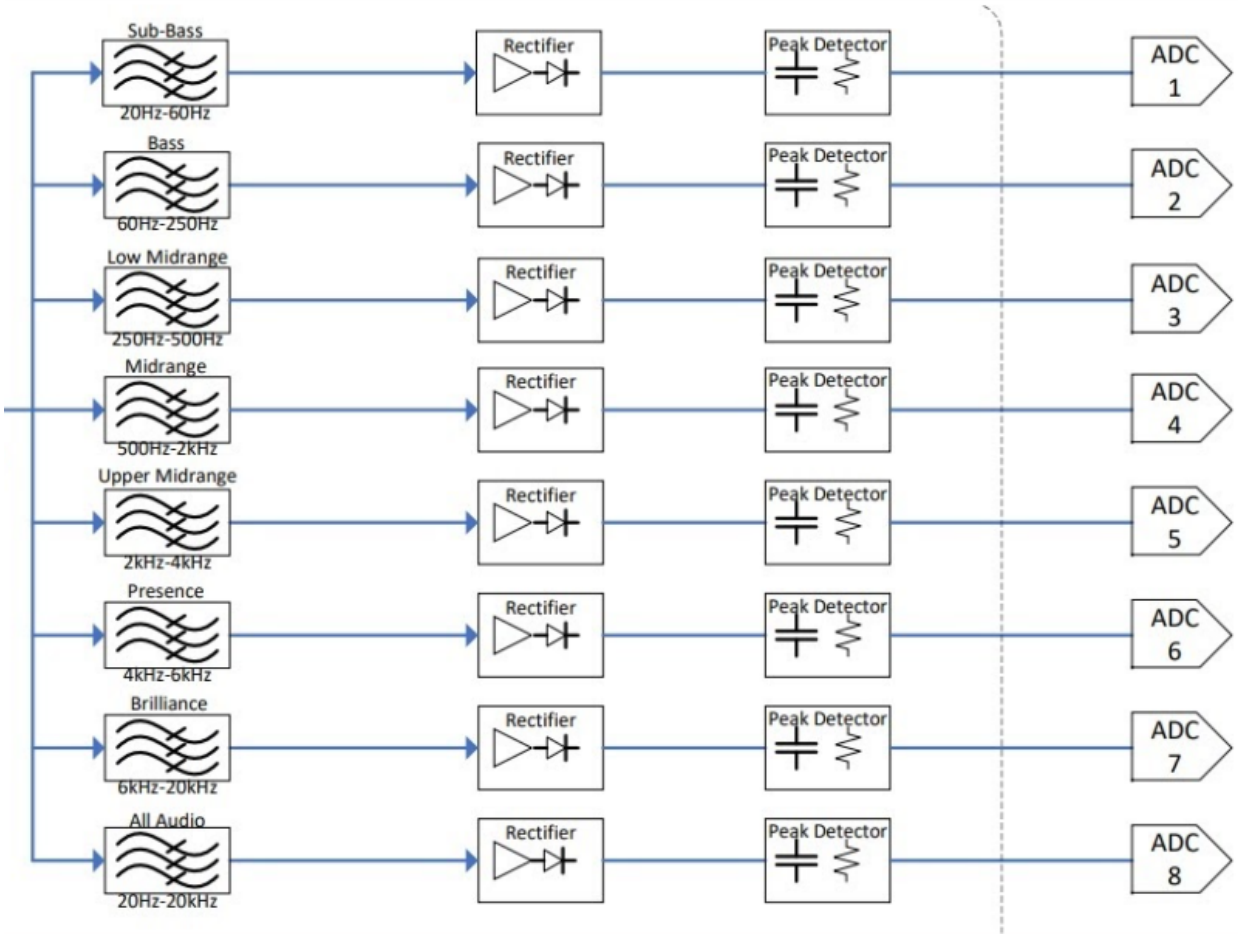


From Input Audio to Summation Circuit

Task: Bandpass Filters to Peak Detector Circuits

Description: The system filters signal into eight frequency bins and then converts those signals into DC values. To do this, it uses eight active bandpass filters which are connected to rectifier and peak detector circuits. This configuration produces a DC value representative of the signal that entered the bandpass filter.

Validation: Use a function generator to generate a 10 kHz sinusoidal signal at 1V peak at the input of the bandpass filter. Verify the output of the peak detector with an oscilloscope is at approximately 1V DC. Perform this process across all frequency bins.

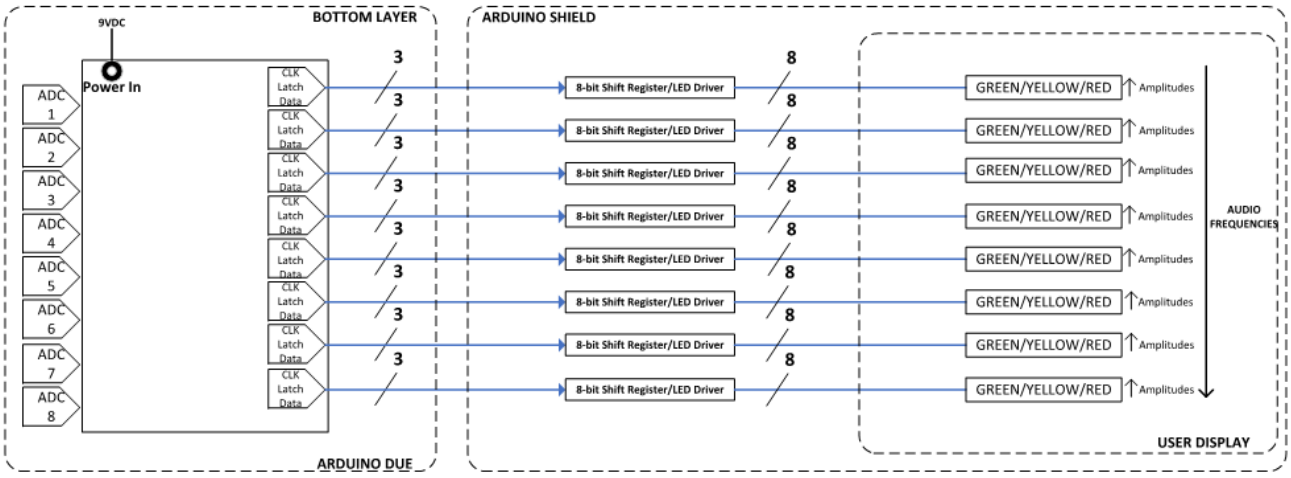


From the Bandpass Filters to the Peak Detector Circuits

Task: ADC to 64 LED

Description: The system multiplexes eight shift registers to drive 64 LEDs. It does this by having each shift register be connected to one digital output pin from the Arduino Due and driving eight LEDs. All of the shift registers share the same clock and latch signals. It is also important to note that the data going into the shift registers is being "Bit Banged" in order to multiplex them.

Validation: Prototype with a small piece of code that multiplexes 8 LEDs with one shift register. Once that has been confirmed to work, continue to add 8 more LEDs and another shift register until all 64 LEDs have been added.



From ADC to 64 LEDs

Cost:

Component Description	Quantity	Cost(per unit)	Cost(total)
Red LED	25	$0.36300	$9.08
Green LED	20	$0.25000	$5.00
Yellow LED	27	$0.73800	$19.93
0.1uF Capacitors	50	$0.66020	$33.01
IC-8bit Shift Register	10	$0.55600	$5.56
10M Resistors	10	$0.10000	$1.00
IC OPAMP	30	$4.62200	$138.66
Diodes	10	$0.27700	$2.77
CAP ALUM 0.1UF 20% 50V Radial	10	$0.20800	$2.08
1/2 W Carbon Film Resistor 10	10	$0.10000	$1.00
Arduino DUE	1	$35.95	$35.95
9V Wall Wart	1	$20.00	$20.00
Box Enclosure	1	$9.86	$9.86
Total			**$318.9**

Cost Chart

Schedule:

Please refer to Teamgantt for a detailed schedule.