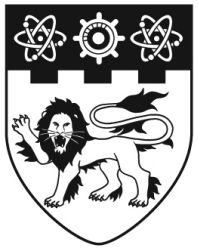


NANYANG TECHNOLOGICAL UNIVERSITY

EE2073 Project Report

Automatic Volume Control for Audio Amplifier System

Student Name: Hong Si Wei Nicholas
Matric No.: U2322578F
Project Group: EJ10

School of Electrical and Electronic Engineering
Academic Year 2024/25
Semester 2

Table of Contents

Objective

1. **Design and Construction:** Construct a multi-stage audio amplifier system with an automatic volume controller. This includes building each sub-system – a voltage-controlled amplifier (VCA), a power amplifier (PA), and a volume unit (VU) meter – and integrating them to work together.
2. **Testing and Evaluation:** To learn and practice the skills of developing, testing, and evaluating each sub-system individually, characterizing their performance and the entire integrated system under various audio signal conditions.
3. **Closed-Loop Control:** To implement a digital feedback control mechanism that automatically adjusts the amplifier gain to maintain a preset output volume.

Introduction

2.1 Automatic Volume Controller

An automatic volume controller is essentially a closed-loop control system for audio loudness. The goal is to maintain the audio output at a roughly constant volume level despite changes in the input signal's amplitude. In our project, this is achieved by continuously sensing the output volume and adjusting the amplifier gain through a control signal. If the input sound gets louder, the controller will "turn down" the amplifier (reduce gain) to avoid a loud output; if the input gets softer, it will boost the gain. This way, the output stays within a comfortable range. The system consists of a control interface (our microcontroller and code) as the "brain," a VCA as the actuator (volume knob controlled electronically), a PA as the power stage to drive the speaker, and a VU meter as the sensor providing feedback of the output volume.

In concept, it's analogous to someone manually adjusting a volume knob while watching a volume level meter – here the "watching" and "adjusting" are done automatically by the control loop. The automatic controller continuously compares the measured volume level to a desired setpoint and tweaks the VCA's control voltage to minimize the difference.

2.2 Voltage Controlled Amplifier (VCA)

The VCA is an audio signal processing sub-system whose gain (or attenuation) is set by an external DC control voltage. In other words, it's an amplifier with an electronically adjustable volume. In our design, the VCA core is the THAT 2180C precision VCA integrated circuit paired with an OP275 operational amplifier in the circuit. This device is designed such that the gain in decibels is approximately linear with the control voltage applied (with a specified sensitivity around 6 mV per dB). A more positive control voltage causes the VCA to attenuate the signal more (lower gain), and a more negative control voltage lets the signal through with higher gain (even amplification). We leveraged this characteristic so that a single control voltage (denoted as VaV_a) effectively acts as our volume knob.

In the overall system, the VCA is the actuator that the controller manipulates to adjust volume. The audio input signal feeds into the VCA, and the VCA's output goes on to the power amp. By changing VaV_a (for example, via a DAC output from our microcontroller), we can smoothly vary the audio signal level going forward. This sub-system is key to automatic volume control: it does the actual job of turning the volume up or down as instructed by the control loop.

2.3 Power Amplifier (PA)

The power amplifier sub-system is responsible for boosting the conditioned audio signal's power to a level sufficient to drive a loudspeaker. Our PA uses the LM380N audio power amplifier IC (14-pin DIP). This IC provides a fixed gain (in theory about 34 dB, roughly a voltage gain of 50) and can drive low-impedance loads like an 8 Ω or 16 Ω speaker. We powered the PA with a single +13.5 V supply (tapped from the controller's variable DC output). The PA input comes from the VCA output, and its output feeds the speaker (or in our tests, sometimes a dummy load and the VU meter circuit).

The PA is essentially the output stage of the amplifier system. It doesn't have adjustable gain; instead, it always amplifies by the same factor. In our design, we expect the PA to provide most of the amplification needed for the audio signal. One important aspect is the PA's bandwidth – ideally it should amplify all audible frequencies (~20 Hz to 20 kHz) evenly. In practice, coupling capacitors and internal limitations will cause some roll-off at the low end (bass) and high end (treble). We measured these characteristics in the lab to see how close the PA comes to the expected 34 dB gain across the audio spectrum. The PA's output is also the point where we take feedback (via the VU meter) to gauge loudness.

2.4 Volume Unit Meter (VU Meter)

The VU meter sub-system serves as the sensor for output volume. It takes a sample of the audio output (in our setup, directly from the PA output line) and produces a DC voltage proportional to the audio signal's amplitude (volume level). The VU meter we built is an analog circuit using an op amp (a CA3140 MOSFET input op-amp) along with diodes, resistors, and capacitors to perform rectification and smoothing. Essentially, it rectifies the AC audio signal and filters it to get something like the average or peak volume. The circuit provides three test nodes: VOUT1V_{\text{OUT1}} (after the first op-amp stage, which inverts and scales the signal), VOUT2V_{\text{OUT2}} (after a second stage that further processes the signal), and VOUT3V_{\text{OUT3}} which is a DC output representing the volume (after a rectifier + filter). In normal operation, VOUT3V_{\text{OUT3}} is the main output of the VU meter, providing a measure of how strong the audio signal is.

In the full system, the VU meter's DC output (which we called VvolumeV_{\text{volume}}) is fed into the microcontroller's analog input as the feedback signal. This is how the controller "hears" how loud the output is. For instance, a higher VvolumeV_{\text{volume}} means a louder sound. One thing to note is that the VU meter has its own dynamic response – it might not be perfectly linear, and it has a lower threshold below which it reads near zero (very quiet sounds may not register well) and an upper limit where it can't increase much further even if the sound gets louder (due to

supply limits and diode drops). These factors would play into how accurately we can control the volume, but within a certain range it gives a good enough indicator of relative loudness.

2.5 Microcontroller and Data Acquisition System (DAQ)

To tie everything together and actually implement control, we used a microcontroller board with a custom interface (the "VScope" board). This board, based on an STM32 microcontroller running MicroPython, acted as our DAQ and control interface. It provides:

- Two arbitrary waveform output channels (denoted W1 and W2) via DACs,
- Two high-speed analog input channels (CH1 and CH2) for oscilloscopic readings,
- Additional analog input channels PC0/PC1 for slower voltage measurements, and
- Programmable power supplies (VDCP and VDCN) for ±13.5 V rails.

In the labs, we connected our circuits to this controller. For example, we powered the breadboard circuits by wiring +13.5 V from VDCP and -13.5 V from VDCN, and we connected grounds accordingly. The microcontroller's W1 output was used to generate test audio input signals (like sine waves of various frequencies) for measurement and characterization of the subsystems. The W2 output was configured as a DC voltage source VaV_a to drive the VCA's control input. Meanwhile, we used CH1 and CH2 to monitor analog waveforms (e.g. viewing input vs output signals on different nodes), and we used the PC0 input to read the VU meter's output voltage VvolumeV_{\text{volume}} as a feedback in the automatic control experiment. All of this was orchestrated via a Jupyter Notebook on the PC, communicating with the microcontroller over serial (using MicroPython commands).

By using this DAQ system, we could automate a lot of the testing – for instance, sweep a frequency and record the PA output, or implement the volume control algorithm in code. It essentially replaced standalone lab instruments: W1/W2 acted as a signal generator, CH1/CH2 as an oscilloscope, and PC0 as a DVM, all controlled in software. This made it convenient to log data and to implement the feedback control loop in Lab 9 as a Python function that periodically adjusts VaV_a.

(In summary, the automatic volume control system we built consists of a VCA (actuator), PA (power output), VU meter (sensor), and a microcontroller running a control algorithm. Initially, we will describe how each part was tested on its own, and then how they all worked together in both manual and automatic volume control modes.)

Experimental Procedure

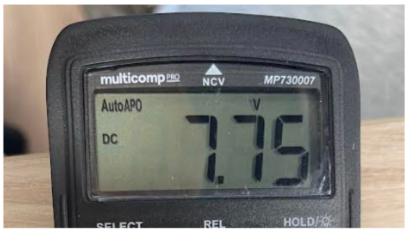
Overview: The project was conducted through a series of lab sessions (Lab 1 to Lab 9) where we incrementally built and tested the system. Early labs focused on familiarization with the microcontroller and basic signal I/O, while later labs focused on building the actual analog circuits and integrating them. The general procedure was:

- **Lab 1 – Microcontroller Setup:** We connected the STM32 microcontroller (with the VScope board attached) to the PC and established a serial connection using PuTTY and Jupyter. We ran simple MicroPython commands to ensure the board was responsive (e.g., printing text, toggling on-board LEDs D2 and D3). Then we tested the programmable DC supply: we placed a 20 kΩ resistor between the +VDCP output and ground on a breadboard and used a multimeter across it. Initially about ~5.5 V was present (default). By sending specific SPI commands in MicroPython (provided in the manual), we set the output to 8.0 V and then 12.0 V. We indeed observed the multimeter reading change to ~8 V and ~12 V accordingly. This confirmed that we could control analog outputs from code. After these tests, we closed the serial connection to free the interface for the next steps.

4.1 Connecting to the controller unit using PuTTY

We placed 20kOhms resistor on a breadboard and connect it to VDCP and GND on the controller unit using jumping wires, then connect a digital multimeter across the resistor to measure the DC voltage.

We changed the DC voltage to 8V and 12 V using the PuTTY commands noted in the Lab manual.



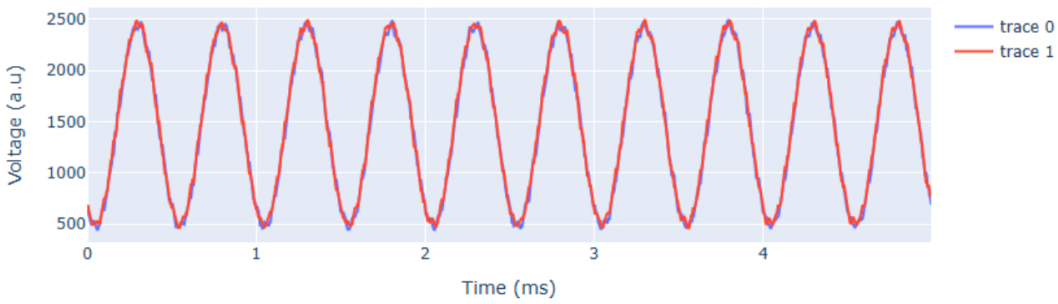
4.2 Setting up Python virtual environment

```
(base) PS C:\Users\leeh0123> conda create --name ee2073
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 23.5.2
  latest version: 25.1.1

Please update conda by running

    $ conda update -n base -c defaults conda

Or to minimize the number of packages updated during conda update use

     conda install conda=25.1.1



## Package Plan ##

  environment location: C:\Users\leeh0123\.conda\envs\ee2073



Proceed ([y]/n)? y

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate ee2073
#
# To deactivate an active environment, use
#
#     $ conda deactivate

(base) PS C:\Users\leeh0123> conda activate ee2073
```

```
(ee2073) PS C:\Users\leeh0123> conda install notebook ipywidgets
```

```
(ee2073) PS C:\Users\leeh0123> conda install pyserial numply plotly
```

- **Lab 2 – Data Acquisition and Visualization:** Using the Jupyter environment, we wrote Python code to generate a test sinusoidal signal on W1 and read it back via CH1. We configured W1 to output a 2 kHz sine wave (0.5 V amplitude) and connected W1 through a resistor to CH1 (to simulate a load and a measurement point). We set the ADC parameters (gain and offset for CH1) so that the ~1 Vpp signal would be centered and scaled properly for capture. We then triggered a capture of 1000 samples at 200 kHz sampling rate. Using NumPy and Plotly, we plotted the recorded waveform. We obtained a nice sine wave trace in the time domain, matching the expected amplitude (~1.0 V peak-to-peak) and frequency (one cycle every 0.5 ms). This exercise verified the DAQ functionality: the system could generate and measure analog signals accurately. It also allowed us to calibrate the channel offsets/gains for proper measurement (for example, we used gain=138 and dco=130 for CH1 based on the manual's guidance to accommodate ~±1 V signals). By overlaying the input and output or by computing FFTs, we gained confidence that subsequent measurements (like spectra and waveforms in later labs) would be reliable.

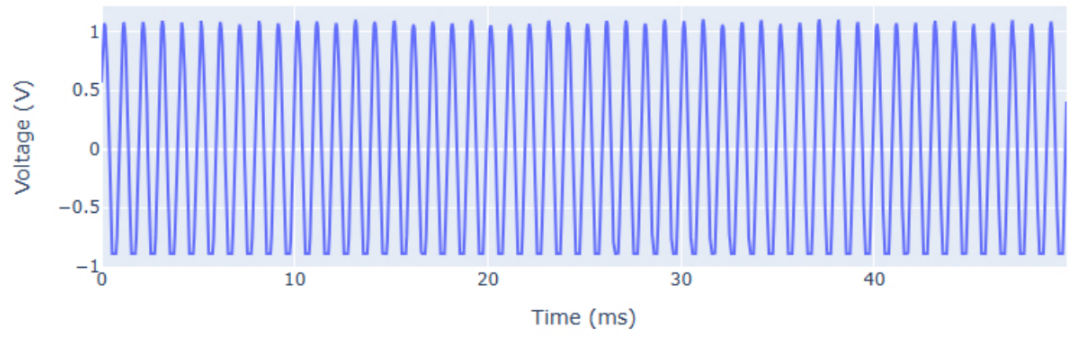


- **Lab 3 – Waveform Modulation and Spectral Analysis:** We used the Python interface to explore more complex signal generation and analysis. First, we generated a 1 kHz sinusoid on W1 and recorded it, then computed its frequency spectrum using FFT. We observed (and noted) that a pure sine has a single spectral peak at 1 kHz with very small harmonics (just noise floor). Next, we changed W1's waveform to a 1 kHz triangular wave and similarly captured its spectrum. The triangle wave's spectrum showed the odd harmonics with amplitudes decreasing at a faster rate than a square or sawtooth (we noticed a small peak at 3 kHz that was a bit higher relative to baseline than for the sine's harmonics – consistent with the idea that a triangle wave contains higher-frequency content than a sine, but still predominantly the fundamental). Then we tried a sawtooth wave at 1 kHz: its spectrum had a rich set of harmonics, significantly larger in amplitude compared to the triangle – in fact, the sawtooth's higher harmonics were quite pronounced, confirming it has the most high-frequency content of the three waveforms. We summarized these observations: *all three waveforms share a 1 kHz fundamental, but their harmonic content differs – the sinusoid has essentially none beyond the fundamental, the triangle has only odd harmonics which are relatively low, and the sawtooth has both odd and even harmonics with relatively larger magnitudes*. This exercise was a warm-up for understanding frequency content and would be relevant when considering audio signal fidelity through our system.

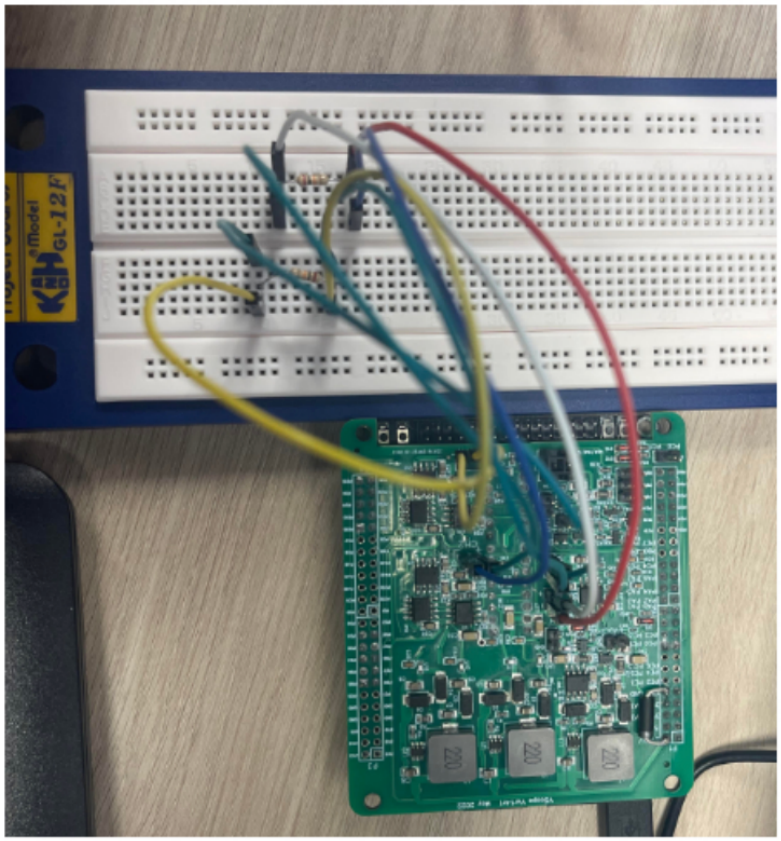
```
import serial
import serial.tools.list_ports
import numpy as np
import plotly.graph_objs as go

# setting up communication channels

VID = 61525
PID = 38912

device = None

ports = serial.tools.list_ports.comports()
for p in ports:
    if p.vid == VID and p.pid == PID:
        try:
            device = serial.Serial(p.device)
        except serial.SerialException:
            print('Reconnect the controller unit.')

if device is None:
    raise Exception('No suitable device detected.')
```

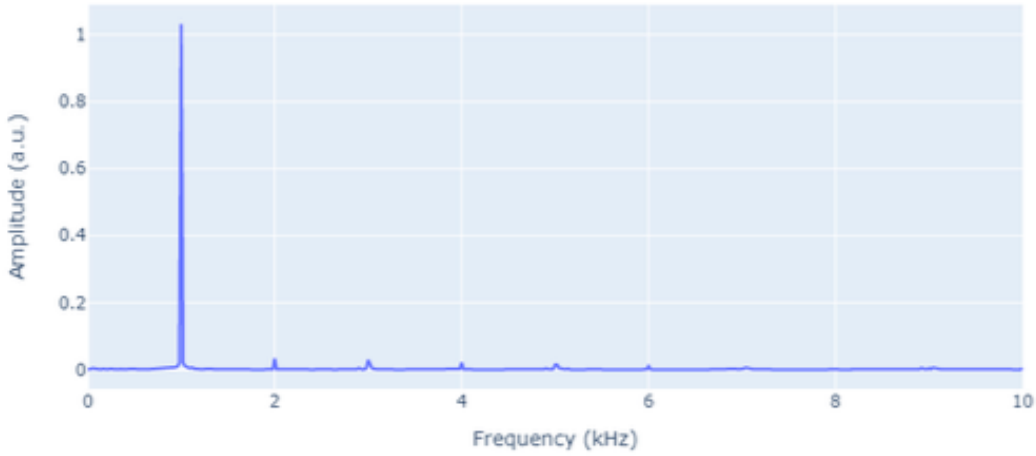
```
# waveform generation

ns = 64
freq = 1000
amp = 1
offset = 0
cmd_gensin = 's100'
cmd_gensin += str(ns).zfill(3) + str(freq).zfill(7) + str(int(amp*100)).zfill(4) + str(int(offset*100)).zfill(4) + '\r'
device.write(bytes(cmd_gensin, 'utf-8'))
```

```
# waveform detection and plotting
fs = 20000
c2 = 0
gain1 = 138
dco1 = 130
c1 = 0
gain2 = 138
dco2 = 130
cmd_readosc = 'm1' + str(fs).zfill(6) + str(c1) + str(gain1).zfill(3) + \
 str(dco1).zfill(3) + str(c2) + str(gain2).zfill(3) + \
 str(dco2).zfill(3) + '\r'
bytedata = bytearray(4000)
device.reset_input_buffer()
device.write(bytes(cmd_readosc, 'utf-8'))
device.readline()
device.readinto(bytedata)
data = np.frombuffer(bytedata, dtype='uint16').reshape((2, 1000))
t = np.arange(1000)/fs
sig = 2*(data[0, :] - np.mean(data[0, :]))/np.ptp(data[0, :]) # adjust
fig = go.Figure()
fig.add_trace(go.Scatter(x=t*1e3, y=sig))
fig.update_layout(xaxis_title='Time (ms)', yaxis_title='Voltage (V)')
```



Connecting 20k resistors with CH1/CH2 and W1/W2 respectively

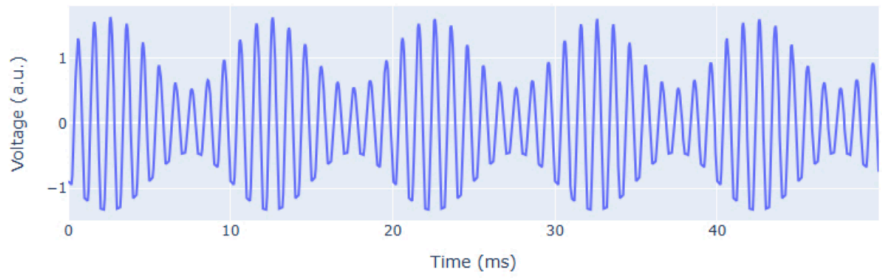




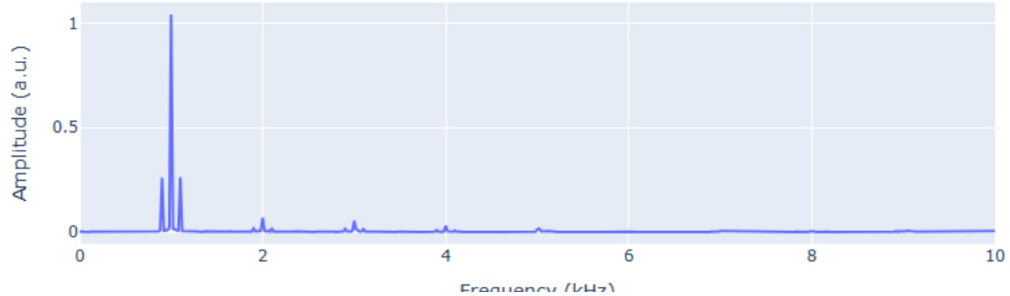
We then demonstrated amplitude modulation (AM). We kept the 1 kHz sine as a carrier and modulated it with a 100 Hz sinusoidal envelope (modulation depth 0.5). Using the microcontroller, we didn't directly generate an AM signal from hardware; instead, we generated the 1 kHz carrier via W1 and then in software multiplied the captured waveform by 1+0.5sin⁡(2π100t)1 + 0.5\sin(2\pi 100 t) to simulate the modulation, then observed the spectrum. As expected, the time-domain signal showed a "wavy" amplitude envelope. More importantly, the spectrum now had sidebands at 1 kHz ± 0.1 kHz, i.e. at 0.9 kHz and

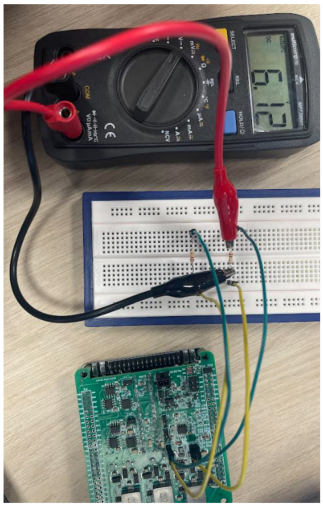
1.1 kHz. We zoomed in and confirmed the sidebands' positions were exactly 100 Hz away from the main carrier peak, and their amplitude related to the modulation depth (0.5). This was a clear verification of the AM concept – a distinct characteristic being those sideband frequencies which were not present without modulation. We documented that increasing the modulation depth made the sidebands stronger, and changing the modulation frequency moved them farther apart (e.g., a 200 Hz modulation would produce sidebands at 0.8 kHz and 1.2 kHz, etc.). We also modulated non-sinusoidal carriers (triangle, sawtooth) and noticed that while the underlying carrier's own harmonics made the spectrum busier, the sidebands still formed around each spectral component similarly. By the end of Lab 3, we had built a solid understanding of signal generation/analysis which set the stage for testing our hardware circuits (which would also involve analyzing waveforms and spectra, e.g., frequency response of the PA, etc.).

[41]:
```
# amplitude modulation
mod_freq = 100
mod_depth = 0.5
mod = sig*(1 + mod_depth*np.sin(2*np.pi*mod_freq*t))
fig = go.Figure()
fig.add_trace(go.Scatter(x=t*1e3, y=mod))
fig.update_layout(xaxis_title='Time (ms)', yaxis_title='Voltage (a.u.)')
```



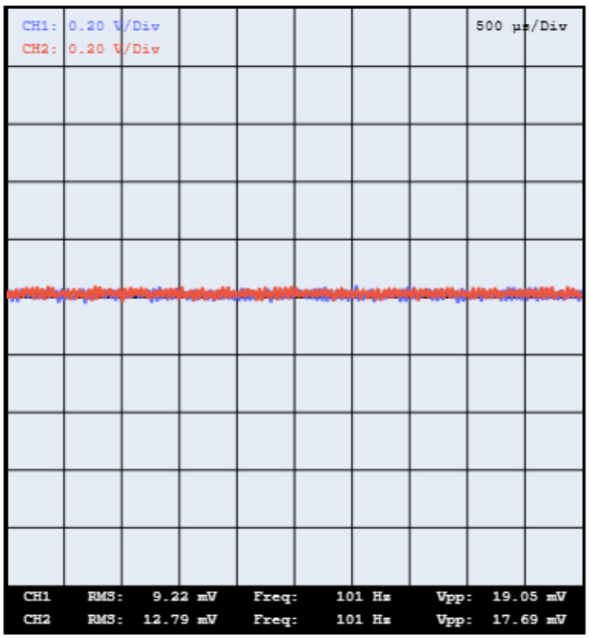

```
# spectral analysis
spec_mod = abs(np.fft.rfft(mod))/len(f)
fig = go.Figure()
fig.add_trace(go.Scatter(x=f*1e-3, y=spec_mod))
fig.update_layout(xaxis_title='Frequency (kHz)', yaxis_title='Amplitude (a.u.)')
```



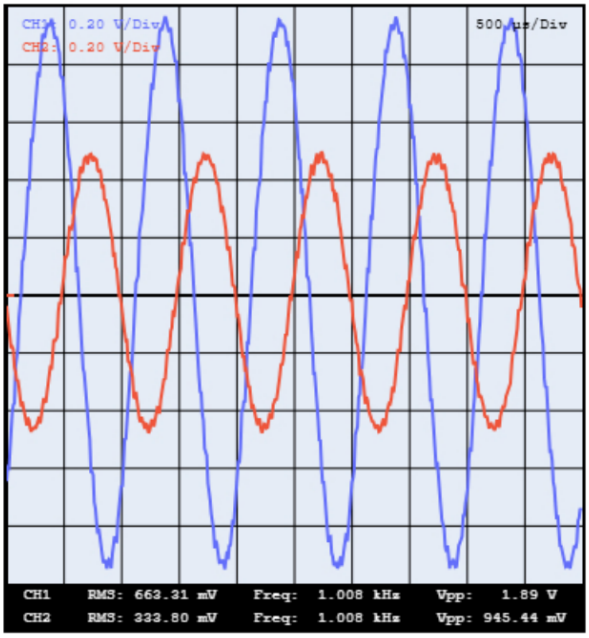


- **Lab 4 – Instrumentation and Automation:** In this session we familiarized ourselves with a provided Python class (the VScopeBoard interface) that bundles the low-level read/write commands into convenient functions (e.g., `vscope.set_vdc(13.5)` to set supply, `vscope.generate_wave(1, 'sine', freq, amp, offset)` for outputs, etc.). We ran and slightly modified the example notebook (lab4_YourName.ipynb) to ensure we could simultaneously control the power supply, wave generator, and oscilloscope channels through this interface. Essentially, Lab 4 was about using pre-built "virtual instruments" rather than manually writing bytes to the serial. After this lab, we were able to more quickly configure the hardware: for instance, launching an oscilloscope view to capture CH1/CH2 traces or adjusting the waveform output via GUI sliders. While Lab 4 didn't produce distinct results of its own (it was more of a tool-building exercise), it greatly sped up subsequent experiments. We could trust that when we set a certain amplitude or frequency via these high-level commands, the microcontroller would output it, and we could capture responses with a single function call. This automation was especially helpful when integrating everything, because we could write a loop to, say, step through control voltages or frequencies and log the system's response without doing everything by hand.

Amplitude = 0 for both W1 and W2



- Amplitude = 1 for W1 and Amplitude = 0.5 for W2 for sinusoidal waveform



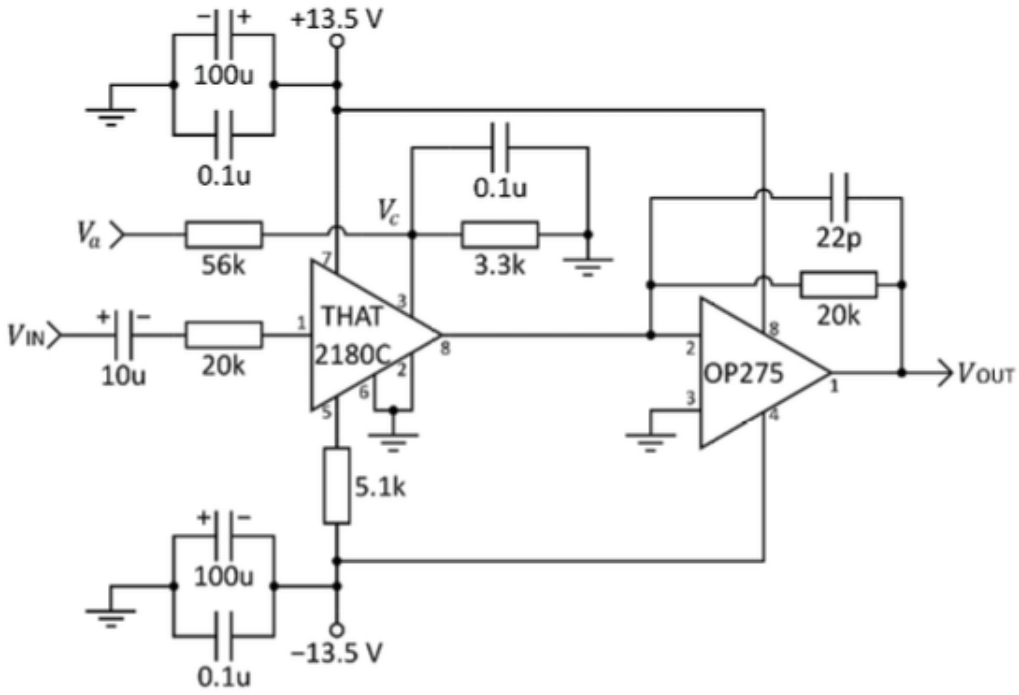
- **Lab 5 – VCA Sub-system Characterization:**

  We constructed the VCA circuit on a breadboard using components - THAT2180C IC, OP275 op-amp, and various resistors/capacitors for biasing and stability.

We took care to use only the left half of the breadboard, leaving room for other subsystems to be added later on the right half. We wired the ±13.5 V supplies from the controller to the VCA's power rails and ground.

For input/output testing, we connected the microcontroller's W1 to the VCA audio input VINV and CH1 to the same node (to monitor the input).

The VCA output VOUT was fed to CH2 to measure the output signal. The control voltage node Va of the VCA was wired to the microcontroller's W2 output (so we could change it programmatically).



We then followed a test procedure to map out the VCA's Gain vs control voltage.

We set a small sinusoidal test signal on V-IN (for example, 100 Hz sine, amplitude ~0.1 V) and stepped Va through a range of values:

(+5.4 V, +4.5 V, +3.6 V, +2.7 V, +1.8 V, +0.9 V, 0 V, -0.9 V, -1.8 V, -2.7 V)

A	B	C	D
Amplitude (V)	Va (V)	VIN-pp (V)	$VOUT$-pp (V)
3	5.4	2.94	0.016
2.8	4.5	2.79	0.027
2.8	3.6	2.75	0.058
2.8	2.7	2.59	0.147
1.8	1.8	1.78	0.25
1.8	0.9	1.795	0.664
1.8	0	1.78	1.69
0.98	-0.9	2.34	1.004
0.88	-1.8	0.917	5.89
0.88	-2.7	0.993	12.525

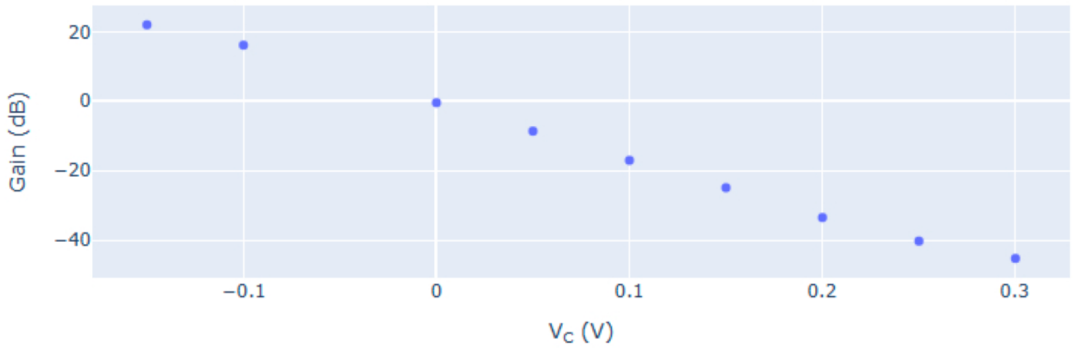
For each Va, we allowed the output to settle, and recorded the peak-to-peak of both VIN and VOUT using the scope.

We observed a clear trend: at the highest control voltage (+5.4 V), the VCA heavily attenuated the signal – the output was almost negligible (0.016V).

As Va decreased toward 0 V and into negative, the output signal grew.

At Va=0V, the gain was around unity (1.69V) (~0 dB, output roughly equates to the input).

At the most negative control we tested (-2.7 V), the output amplitude was significantly larger than the input (gain > 1, an amplification). In fact, by the last couple of points (e.g. -1.8 and -2.7 V), the output sine wave started to clip on the top – the op amp was saturating because the gain was so high that the 0.1 V input tried to drive the output beyond the supply limits.



This result is as expected; those points were beyond the linear range of the VCA/PA combination (since the PA was not yet in circuit, the clipping was likely at the VCA op-

amp output limited by ±13.5 V rails). We noted which points were in saturation and later excluded them when analyzing the linear gain behavior.

Next, we converted our measurements to gain in decibels:

$$G_{dB} = 20 \log_{10}\left(\frac{V_{OUT}}{V_{IN}}\right)$$

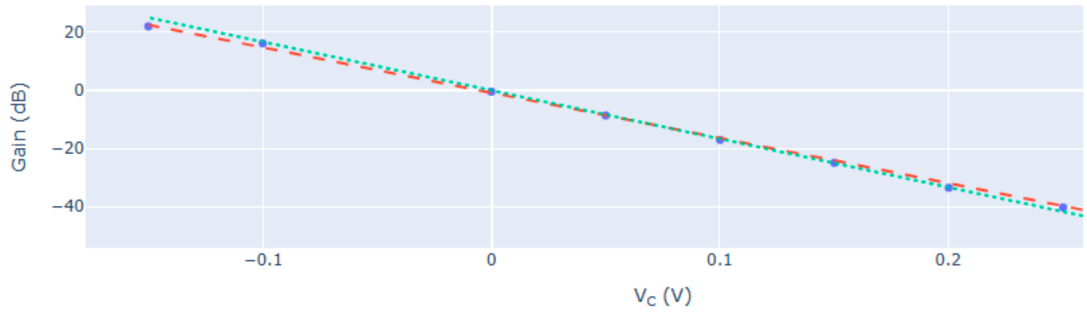
for each setting. Plotting Gdb versus the control voltage VC (which is a scaled version of Va through a resistor network, we found an approximately linear relationship.

We performed a linear fit to the data (excluding the clearly saturated points). The slope of this line corresponded to the gain sensitivity. The specified sensitivity of the chip is -6.0 mV/dB, and our measured result was very close

The discrepancy could be due to temperature, the chip heating up, or minor measurement error. The VCA was behaving as expected. The line we plotted (our measured gain vs VC) also matched well with the theoretical line from the formula

$$G_{dB} = -\frac{V_C}{0.006(1 + 0.0033\Delta T)}$$

They were nearly parallel, confirming the exponential control-law of the VCA's gain core (i.e., linear in dB).



We also observed the extremes: at very high gain (negative Va), besides clipping, there was a lot of audible hisses added to the output. Subjectively, when we later listened with a speaker, if Va was set extremely low (max gain), the amplifier would amplify even the background noise.

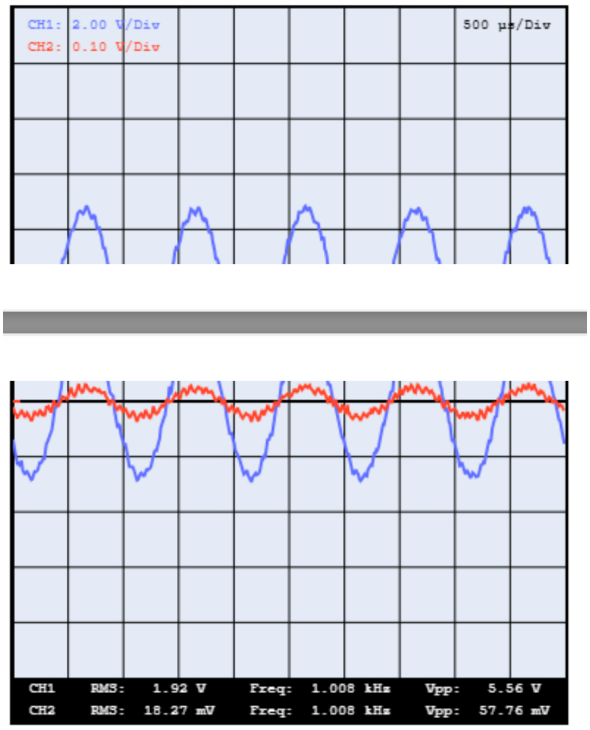
This was as expected – when the VCA is at high gain, any small input (including noise)

gets amplified. Conversely, at very high Va (min gain), the output was near silent; the noise floor was so low we could barely hear anything.

This demonstrated the wide dynamic range of the VCA: it can almost mute the signal or greatly amplify it depending on control.

In a realistic scenario, we wouldn't operate at the extreme ends because of noise or clipping – and indeed in the automatic control we would aim to keep it in a range where it can smoothly adjust volume without hitting those limits.

Waveform example,



In our lab 5 reflection: *What is the function of the VCA in the overall system?*

The VCA is like the volume knob of the entire amplifier system, where its audio volume can be adjusted electronically. In the automated system, instead of a person turning a knob, the microcontroller will adjust Va to keep volume steady. The VCA's role is crucial because it provides a point where the system can control the gain of the audio path in real time based on feedback. Without it, we'd have no way to regulate volume except to cut or boost the input signal externally. So, the VCA sub-system gives us the "handle" by which our controller can effect changes in output loudness. We noted this analogy in our report: the VCA is essentially an electrically controlled volume control that makes automatic

volume regulation possible.

Results & Discussion

VCA Sub-system Performance

In summary of the VCA tests, we confirmed a nearly exponential gain control behavior.

The measured gain ranged roughly from -50 dB (very attenuated) up to about +20~+25 dB (amplification) over the control voltage range we tried.

This matches the expected span for the THAT2180C device. The slight non-linearity at the extreme ends (and the onset of clipping for Va≤−1.8V_a \le -1.8 V in our case) pointed out practical limits: the op amp can only swing so far on ±13.5 V, and the device's internal transistors can only linearize gain up to a point.

We addressed this by avoiding those extremes in normal operation. When the VCA was integrated into the full system, we anticipated that if the control loop asks for an extremely low Va(max gain) while a large input signal is present, the PA stage would likely clip the output.

We also noted that the VCA introduces some noise (particularly at high gain).

If the environment is quiet (input is low) and the controller cranks up the gain, a listener might hear a hiss. This is the trade-off in any automatic gain control system. Our VCA chip is designed for low noise, but no component is perfect. During tests, when we set a small 10 mV sine input and a very low Va(high gain), the output was audible and the sine came through but superimposed on a noticeable white noise background. Conversely, at high attenuation, the output noise was negligible.

The signal-to-noise ratio (SNR) is best at moderate or low gains and worst at maximum gain. In practice, the automatic controller will only drive the gain high when the input signal itself is very soft, so the absolute noise may not be too disturbing in context (one might just hear a slight hiss in very quiet passages).

The VCA provided a controllable gain element. We confirmed that a simple linear model in dB holds over most of its range. This allowed us to use it in the closed-loop system. We ended Lab 5 with a functional VCA sub-system ready to be combined with the PA.

Power Amplifier Sub-system Performance

We built the PA sub-circuit in Lab 6 on the same breadboard, placing the LM380N IC and its supporting components (coupling capacitors at input/output, a 22 µF bootstrap cap between pins 1 and 8 as recommended, and a 0.1 µF decoupling cap close to the supply pins). Pin 14 of the LM380N was connected to +13.5 V, and pins 3,4,5 and 10,11,12 were tied to ground (the LM380 has multiple ground pins for heat dissipation – we made sure pin 7 (signal ground reference for output) was properly isolated.

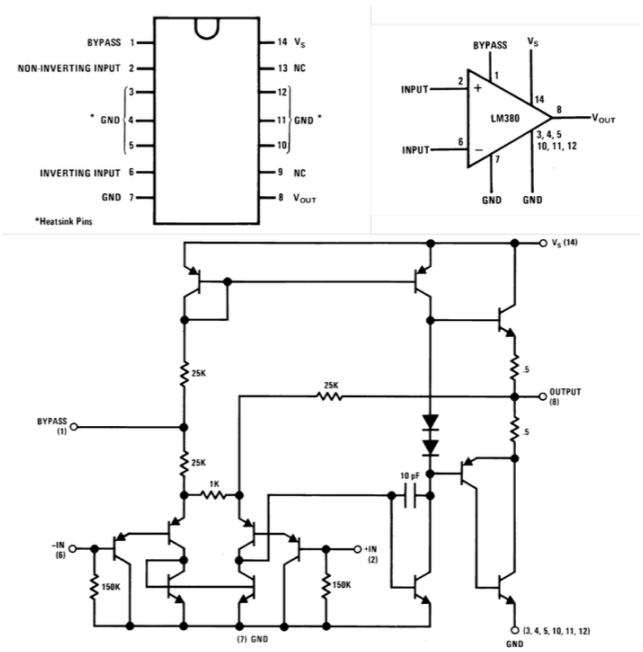
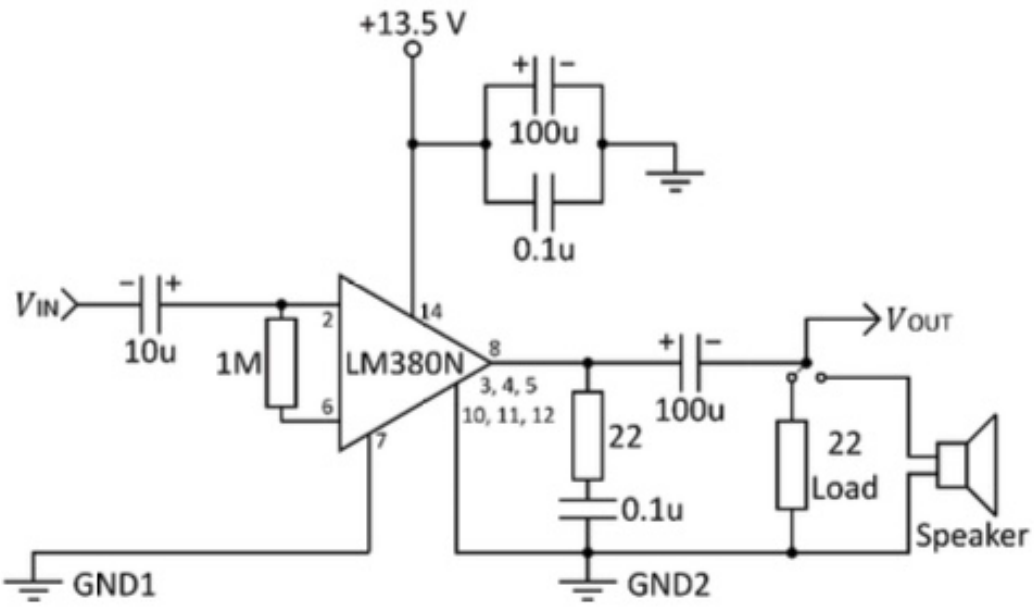


Fig. 6.1. LM380N power amplifier IC.



To measure the frequency response of the PA, we conducted a sweep. We used W1 to inject a small sinusoidal signal (around 0.05–0.1 V amplitude) into the PA input and measured the output amplitude via CH2, across a range of frequencies from 100 Hz up to 50 kHz. We chose two input amplitude levels for comparison: 0.1 V and 0.2 V.

Amplitude = 0.1		
Frequency (Hz)	Vin-pp (V)	Vout-pp (V)
100	0.1764	3.14
200	0.2519	8.82
500	0.258	9.03
1000	0.2322	8.75
2000	0.2368	8.15
5000	0.2219	7.53
10,000	0.2254	7.31
20000	0.2282	7.69
50000	0.2335	6.19

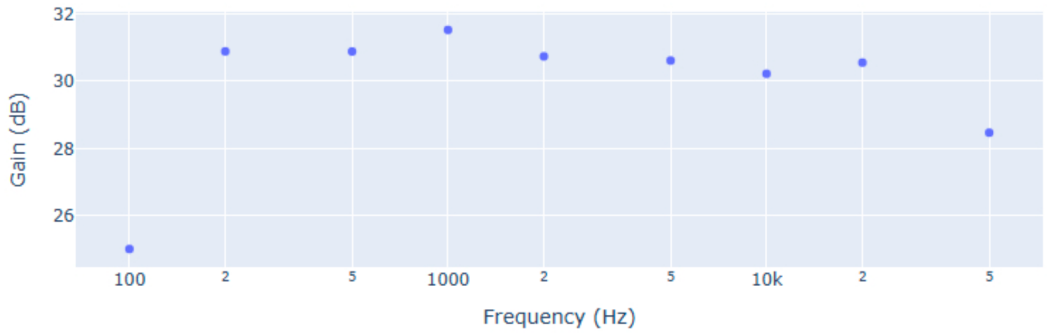
At 0.1 V amplitude, the PA output remained clean (no clipping) at all frequencies tested, and the gain was roughly constant through much of the range: about 30–32 dB from 200 Hz up to 20 kHz.

At 100 Hz, we observed the output was lower – the gain was below 26dB, indicating a low-frequency roll-off. This makes sense because there is an input coupling capacitor forming a high-pass filter with the input resistor; at 100 Hz some attenuation occurs.

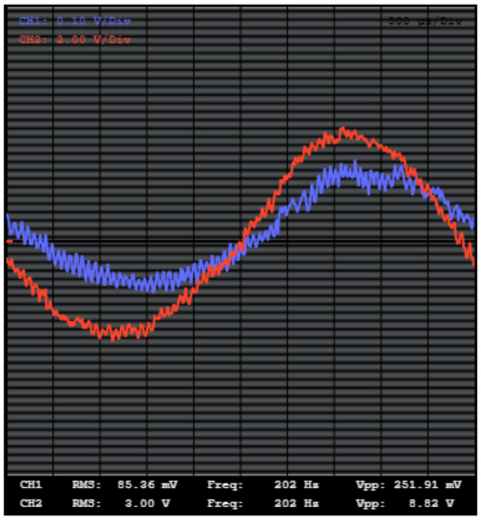
On the high-frequency end, at 20 kHz the gain was around 30.5 dB, and by 50 kHz it dropped to ~28.5 dB. So the PA does have a limited bandwidth: by 50 kHz we saw a noticeable drop.

Throughout the audio band (20 Hz–20 kHz), the gain stayed reasonably close to about 31 dB(ideal 34 dB).

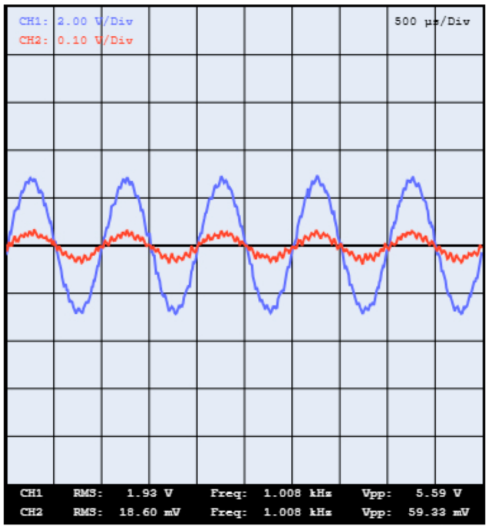
This could be due to the fact that we powered with 13.5 V instead of 15 V (LM380 is specified at 18 V max, but 12 V min; at 13.5 V maybe it doesn't achieve full gain) and also our measurement references (maybe the input coupling cap or measurement method could cause a slight underestimation). In any case, the PA provided a large fixed gain as expected.



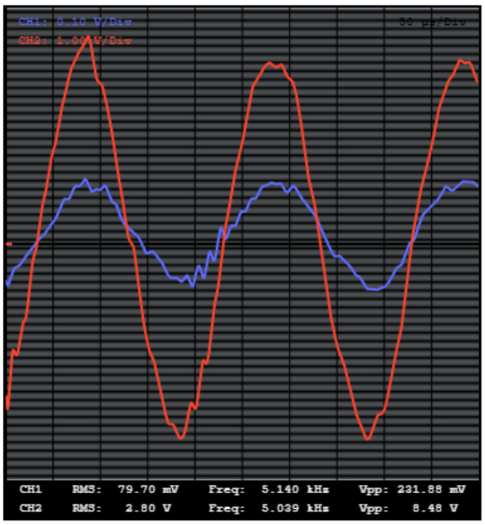
A = 0.1, f = 200 Hz



A = 0.1, f = 1000 Hz



A = 0.1, f = 10000 Hz



At the 0.2 V input amplitude test,

Amplitude = 0.2		
Frequency (Hz)	Vin-pp (V)	Vout-pp (V)
100	0.4239	11.62
200	0.4458	11.72
500	0.4455	10.09
1000	0.4292	9.83
2000	0.4256	9.75
5000	0.3941	10.12
10,000	0.6167	10.22
20000	0.422	10.61
50000	0.4153	11.11

we looked for any nonlinear behavior. The midband output with 0.2 V in should ideally be double

that of 0.1 V in (since gain is fixed), which would be about 2x voltage => same gain in dB.

At 200 Hz, for example, with 0.1 V in we got ~8.8 Vpp out; with 0.2 V in, we expected ~17.6 Vpp. What we actually measured at 200 Hz was ~11.7 Vpp out – clearly less than expected.

The PA had saturated or hit a limit. Indeed, 11.7 Vpp is about the maximum unclipped swing we could get given a 13.5 V supply (the LM380N cannot swing fully to the rails; ~12 Vpp was the max output clean sine we observed).

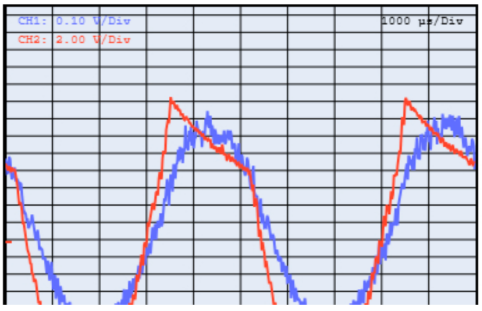
So at low frequencies, the 0.2 V input drove the PA into a mild clipping – the top and bottom of the sine started flattening. This means the gain effectively plateaued: increasing input beyond a certain point didn't increase output proportionally.

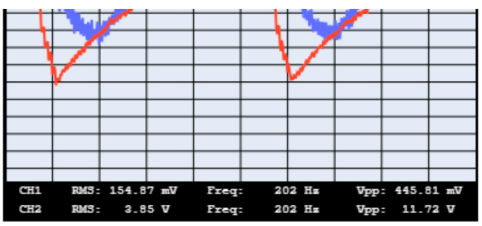
At 100 Hz with 0.2 V in, we measured ~11.6 Vpp out (similar story). As frequency increased, the required output amplitude for a given gain drops due to the roll-off; by 1 kHz and above, the 0.2 V input was no longer saturating the PA because the gain is slightly lower and the output stays within ~9–10 Vpp.

For instance, at 1 kHz we got about 9.8 Vpp out for 0.2 V in (which is ~32.3 dB gain, not far from the small-signal gain). So above a certain frequency (~500 Hz in this case), the 0.2 V input case and 0.1 V input case produced proportional outputs (no clipping). It was only in the low-frequency region where the PA tried to produce very large swings that it hit its output voltage ceiling.

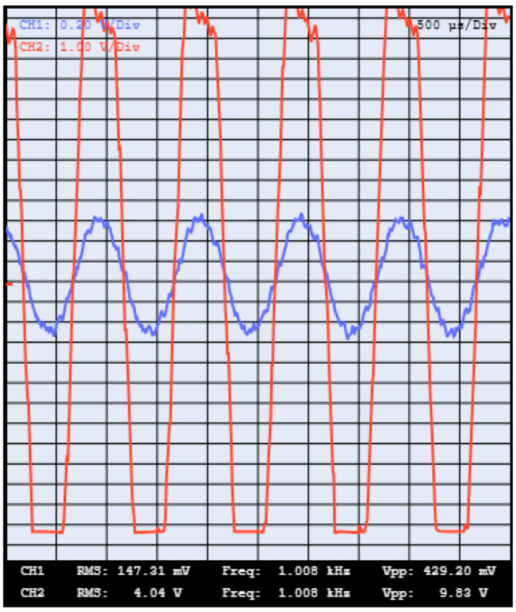
We discussed that in context: in a real audio scenario, very low bass at high volume might distort first, which is a common characteristic of amplifiers with coupling capacitors (low frequencies may also cause the output stage to saturate if the supply isn't large enough for the required swing).

A = 0.2, f = 200 Hz

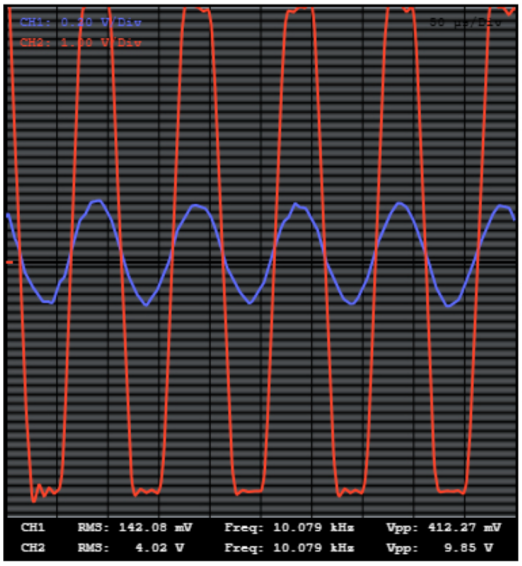




A = 0.2, f = 1000 Hz



A = 0.2, f = 10,000 Hz

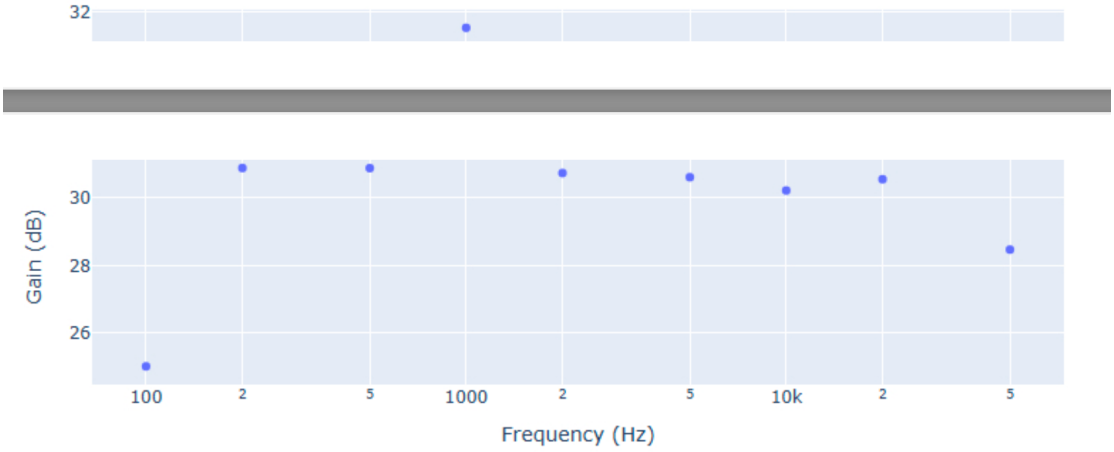


In addition to amplitude saturation, we also considered the operating range and current:

The open-ended question in Lab 6 asked how much current flows through the output stage ground at “loudest volume.”

If the PA is driving a low-impedance load (say an 8 Ω speaker) at maximum undistorted output (~11 Vpp which is ~3.9 V RMS), the current would be I = VRMS / R ≈ 3.9/8 = 0.487 A (about 0.5 A). In our tests, we used a 22 Ω load (as a proxy for a small speaker or just a power resistor). For 11.7 Vpp (~4.14 V RMS) across 22 Ω, the current is 0.188 A.

The chip did get warm to the touch after driving high output for a while. We felt the LM380N package and noted it was slightly hot when pushing near its limits. We ensured it had adequate airflow.



In conclusion for the PA: its gain vs frequency profile indicates it will amplify the mid-range of audio quite consistently.

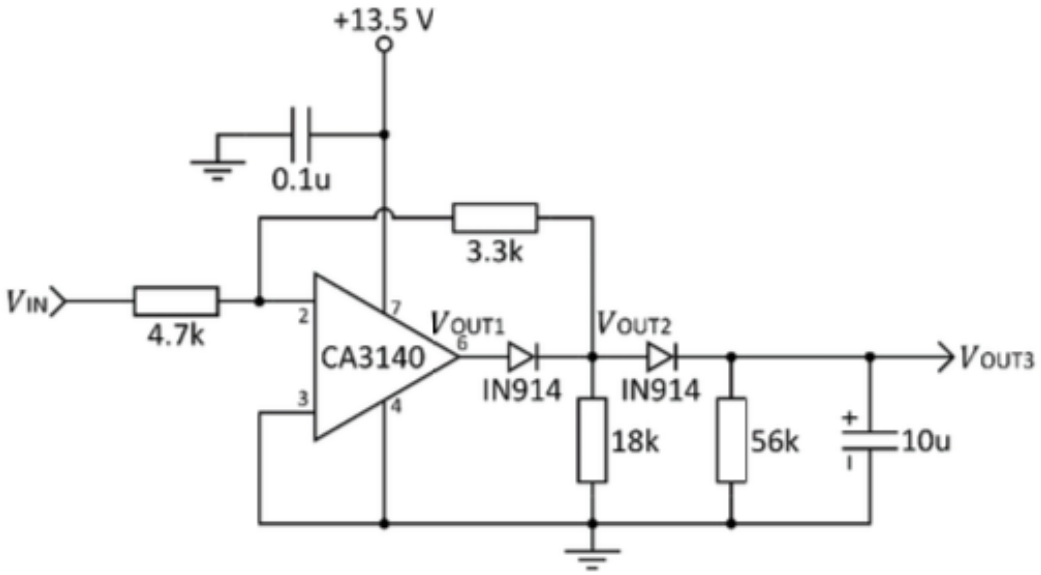
Very low bass (~100 Hz and below) will be attenuated).

The treble up to 20 kHz was fine.

We also recognized that pushing the amplifier to its limits introduces distortion; thus, part of our volume control design would implicitly avoid those limits – if volume gets too loud (approaching clipping), the controller should reduce gain.

This is in fact beneficial, because the automatic volume controller will prevent the amplifier from saturating for extended periods by design (it will turn the gain down as clipping starts to make the VU meter shoot up).

VU Meter Sub-system Performance



In Lab 7, we wired up the VU meter circuit on the breadboard (sharing +13.5 V supply and ground with the PA, to which it connects). The VU meter input V-IN was taken from the PA output node (through a 47k resistor as in the design, to avoid loading the PA significantly). We monitored the three outputs VOUT1, VOUT2, and VOUT3 with the oscilloscope by switching CH2 among those test points as needed (CH1 remained on the input as a reference point). We then drove the system with W1 at 1 kHz sine waves of various amplitudes and recorded the peak-to-peak voltages at each test point, as well as the VOUT3-RMS.

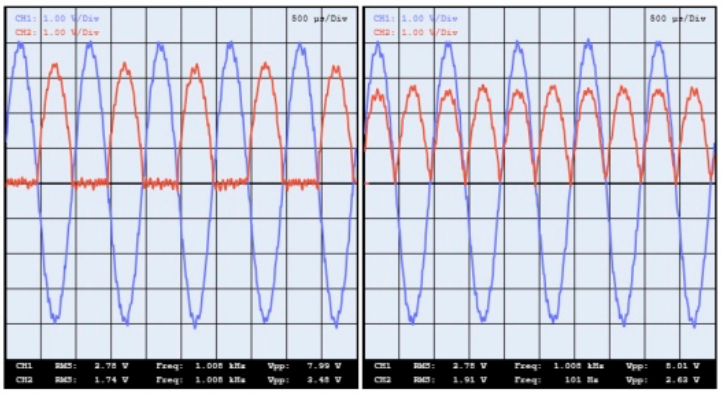
VU Meter Gain Measurement

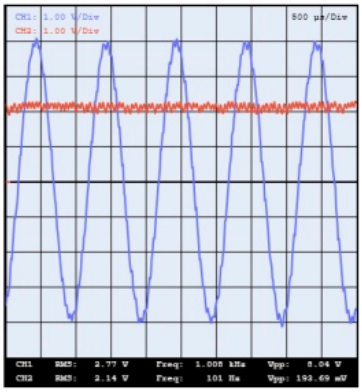
Amplitude (V)	VIN-pp (V)	VOUT1 -pp (V)	VOUT2 -pp (V)	VOUT3 -RMS (V)
4.2	7.99	3.48	2.63	2.14
3.3	6.31	2.89	2.12	1.59
3.2	6.01	2.83	2.07	1.51
2.2	4.2	2.14	1.43	0.909
2.5	4.8	2.34	1.57	1.1
1.7	3.3	1.82	1.17	0.613
1.8	3.48	1.87	1.19	0.673
0.2	0.384	0.602	0.135	0.05
0.3	0.578	0.685	0.191	0.065

For a representative selection: with an input amplitude of about 3.3 V, we measured VOUT1≈2.89, VOUT2≈2.12V and VOUT3≈1.59V.

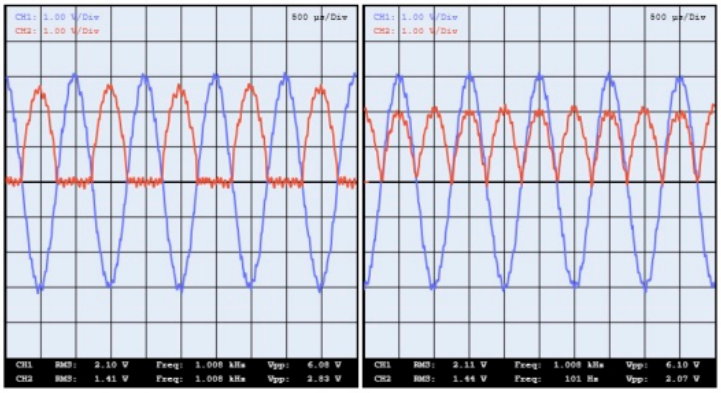
We noticed a general trend. We observed that As V-IN increases, V-OUT1, V-OUT-2 scale roughly linearly with input (since they are basically amplified and filtered versions of the input waveform), while VOUT3 increases in a non-linear fashion – it's small for low input, then larger for high input, but not one-to-one because of the diode threshold and logarithmic charging of the capacitor.

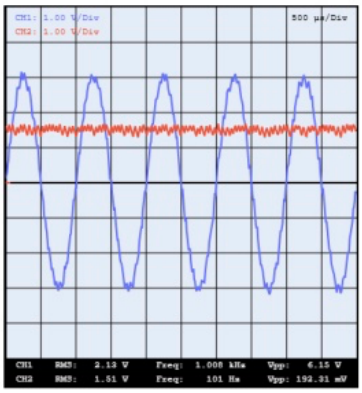
When Amplitude=4.2,





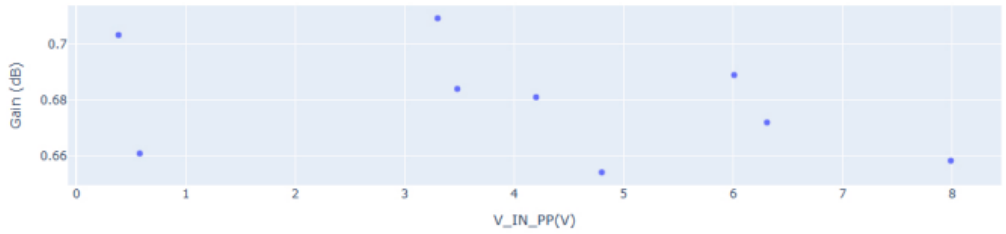
When Amplitude=3.2,





To interpret the numbers: VOUT1 is the output of the first op amp, which was configured as an inverting amplifier with a certain gain (close to -0.5 by design). Indeed, when input was 6 Vpp,

VOUT1~2.83 Vpp – roughly a third of input, consistent with a divider/gain. VOUT2 further attenuates or level-shifts the signal. The key output VOUT3V is after the diode and RC smoothing, so it represents something like the peak or average of the rectified signal. It was highest (≈2.14 V DC) for our largest input (which was about 7.99 Vpp in one trial), and almost zero (for our smallest input (around 0.3 Vpp,).



We then plotted the Gain vs V-IN to see the linearity. The theoretical gain of that stage was -0.70 (i.e., output should be 0.7 times input in opposite phase, in peak-to-peak terms). Our measured points hovered around -0.66 to -0.72, which is very close.

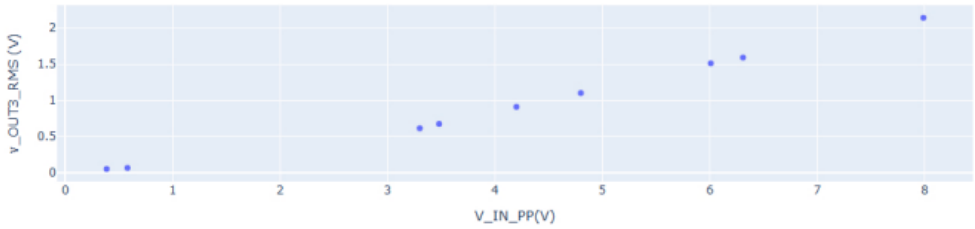
Thus, the VU meter's amplifier was working properly.

The difference could be either due to tolerance in resistor values or the diode forward voltage affecting bias. We consider that an excellent result: the VU meter accurately reflected changes in input amplitude.

For dynamic behavior, we captured example waveforms: at a high input amplitude (4.2 V), VOUT1 was a clipped version of the sine (because the op amp was saturating on peaks – the first stage likely hit its limits on the negative swing due to its biasing). VOUT2was a smaller waveform (and possibly mostly above 0, since after the rectifier the second stage might output only the positive portion). VOUT3 in time domain was essentially a DC level with a slight ripple at 2 kHz which is due to the capacitor charging and discharging each cycle.

At lower inputs (0.2 V), the waveforms at VOUT1 and VOUT2 were tiny, and VOUT3 was near 0. These traces demonstrated the range of the VU meter: it has a certain sensitivity threshold (below ~0.5 Vpp input, the output is very low and might be dominated by noise offsets) and a saturation point (above some level, further increases in input don't raise VOUT3 proportionally because it approaches a max of ~2–3 V). The useful range for VOUT3 is within those bounds where it responds approximately proportional to the log of input amplitude.

In plain terms, the VU meter compresses the actual voltage range into a more convenient scale. We saw that as input voltage increased by a factor of ~10 (from 0.6 Vpp to ~6 Vpp, which is 20 dB increase), the VOUT3 rose by on the order of a volt or so. This compression is good because it means the feedback signal won't swing wildly for big changes in volume – it will vary in a smoother way that our controller can manage.

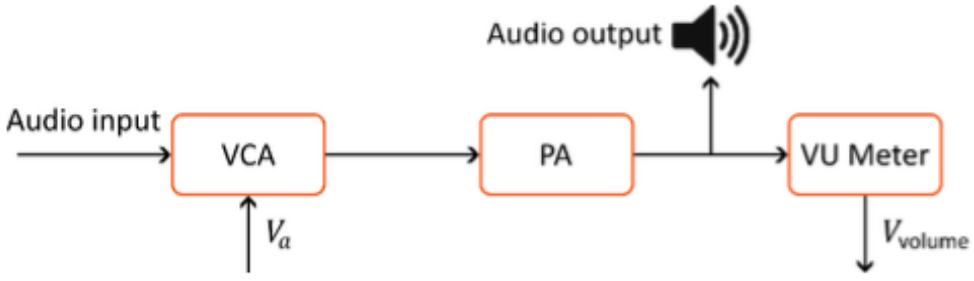


We also contemplated the response speed: The VU meter's output smoothing capacitor introduces some lag – we noticed that VOUT3 would take a brief moment to rise and fall with sudden changes in input amplitude (not instantaneous). This is intentional, to mimic how analog VU meters have a certain rise and fall time (so that the reading is stable and not flickering every millisecond). For our purposes, this adds a bit of damping to the feedback loop, which likely helps avoid overreaction to very short transients. We kept this in mind when designing the control algorithm, as it effectively filters the measured volume.

The VU meter sub-system was operational and calibrated. We obtained a mapping of output voltages for known input levels, which gave us a sense of how to choose a setpoint later. For example, if we wanted the system to maintain a "moderate" volume, we might target Vvolume≈0.2V. If we wanted it louder, maybe 0.3 V or more (closer to the upper range). We also confirmed that the VU meter doesn't significantly load the PA.

Integration of Sub-systems – Manual Volume Control

With all three sub-systems (VCA, PA, VU meter) built and tested in isolation, Lab 8 was about integrating them into one full circuit on the breadboard and performing a manual volume control test. We connected the pieces as follows:



The audio input from W1 goes into the VCA's VIN. The VCA output feeds into the PA input. The PA output goes to a 22 Ω load (simulating a small speaker) and also to the VU meter input. The VCA's control voltage VaV_a is provided by W2. Essentially, it's the same setup as in individual tests, just all in one pipeline now: W1 -> VCA -> PA -> VU -> PC0 (feedback measurement).

We powered the combined circuit with ±13.5 V rails (with that 820 µH inductor splitting the analog +13.5 V supply between the VCA and PA sections to decouple noise – we did include this inductor

as instructed, between the node that feeds VCA and the node that feeds PA, both from VDCP). Importantly, we double-checked the ground connections: earlier, we had an incident where the PA's signal ground (pin 7 of LM380N) was accidentally tied to the wrong ground rail which introduced a weird behavior – the PA output AC coupling was lost, causing the output to flatline when measured at certain points. We corrected that by isolating pin 7 to the proper ground (ensuring all sub-circuits share a common ground reference in the correct way). After this fix, the integrated circuit worked seamlessly: the PA output showed the amplified audio, and the VU meter produced a reading.

Audio amplifier system testing:

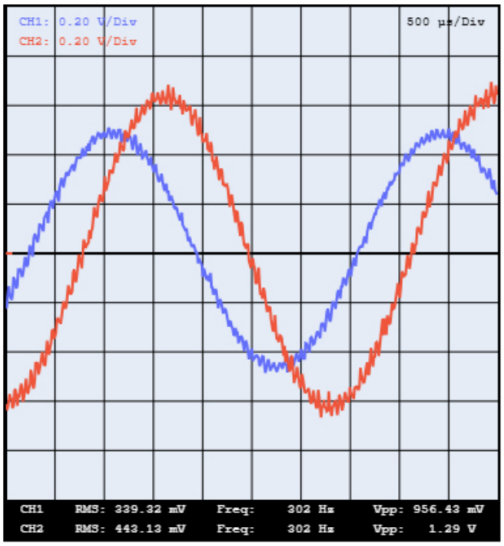
Amplitude (V)	Va (V)	Vsignal - pp (V)	Vaudio - pp (V)	Vvolume - pp (V)
4.3	5.4	7.97	0.36531	0.37787
3.3	4.5	6.17	0.6693	0.3183
2.3	3.6	4.3	1.12	0.34786
1.3	2.7	2.43	1.55	0.16951
0.83	1.8	1.64	2.58	0.07753
0.33	0.9	0.72957	2.66	0.48956
0.09	0	0.18193	1.83	0.21498
0.04	-0.9	0.08886	1.96	0.30181

The RMS value of Vvolume when the peak-to-peak Vaudio is at its maximum is 0.48986V

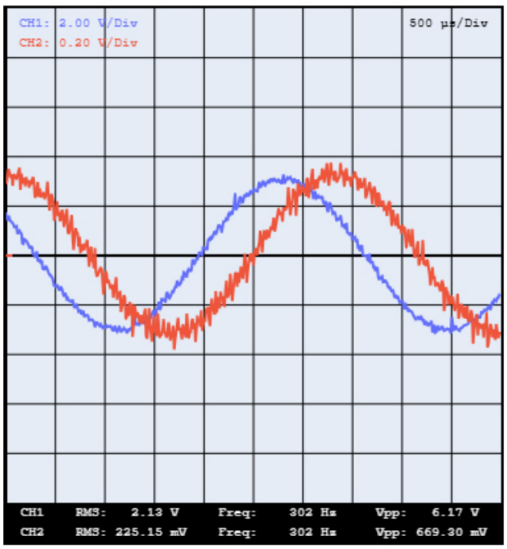
For the manual test, we essentially used our microcontroller as a manual controller: we would set a certain Va value and observe the outcome (the output amplitude and the VU reading). We prepared a table of results by varying both the input amplitude from W1 and the control voltage Va. For example, we set the input to a moderate level (say 0.5 V amplitude, ~1 Vpp) and tried a few Va values: at Va=0V_a = 0 V, the VCA gave ~unity gain, so the PA output was around 1 Vpp (times PA gain ~50, so actually about 50 Vpp, which saturated the PA – indeed that combination would clip; in practice we used smaller input when Va was low to avoid that).

At Va=+2V_a = +2 V, the VCA attenuated the signal significantly and the output was much smaller. Specifically, with input amplitude 0.5 V and Va=+2V_a = +2 V, we observed an output waveform of about 0.36 Vpp and the VU meter reading about 0.38 V (this corresponds to one of the points in Table 8.1). If we increased Va further to +5 V (max attenuation), the output became almost silence (just a few mV of noise). If we decreased Va to negative values,the volume would shoot up – effectively the whole system gain = VCA gain * PA gain, which could be huge.

When A = 0.5V and Va = 2V



When A = 3.3V and Va = 4.5V



We systematically varied the input signal amplitude as well, to simulate different source loudness. For each input amplitude, we could manually tweak Va to control the output. For instance, at one point we set a larger input amplitude (~3.3 V) and a relatively high Va(~4.5 V) – the output came out around 0.87 Vpp (much smaller than the input, since VCA attenuated strongly) and Vvolume≈ 0.32 V.

In another case with smaller input (0.9 V) and low Va(0.7 V), we got a bigger output ~2.66 Vpp and Vvolume≈0.49V. These represent how a user might manually maintain roughly the same output by inversely adjusting Va against input changes. In fact, looking at our collected data, when the input amplitude was higher, we tended to set Va higher to keep output in a similar ballpark of a few volts peak-to-peak. The VU meter reading provided a quantitative way to judge if the outputs were "similar."

We noticed that by choosing appropriate Va values, we could aim for Vvolume be, say, ~0.3 V for all cases, meaning the output loudness is held around a target. This is exactly what the automatic controller will do – but here we did it manually, informed by the VU meter.

One interesting observation was when we inadvertently mis-wired initially: the PA output was present at the op amp (first stage of VU) but not at the actual output post LM380. We realized the mistake in the ground wiring (pin 7 of LM380 was tied incorrectly). It was a moment of confusion: we saw a signal at one node but not at another, which taught us to carefully separate analog grounds as intended. After fixing that, the output signal appeared nicely at the speaker/load and all measurement points.

We also noticed the distortion at high gains more clearly with a speaker attached. For instance, with a fairly loud input and Va=0V(so full VCA gain), the sound from the speaker was very distorted and "fuzzy." On the oscilloscope, this corresponded to a clipped waveform – the PA saturating. It was a good demo of why we need automatic control: no user would want to keep Va=0 in that scenario, you'd immediately hear the distortion and reduce the gain. With our own ears, we played

a test tone and manually adjusted Va until the sound from the speaker cleared up (no audible clipping). This roughly corresponded to where the oscilloscope showed the waveform just below clipping. The VU meter reading at that point was noted, as it essentially indicates the maximum volume we can go without distortion for that input amplitude.

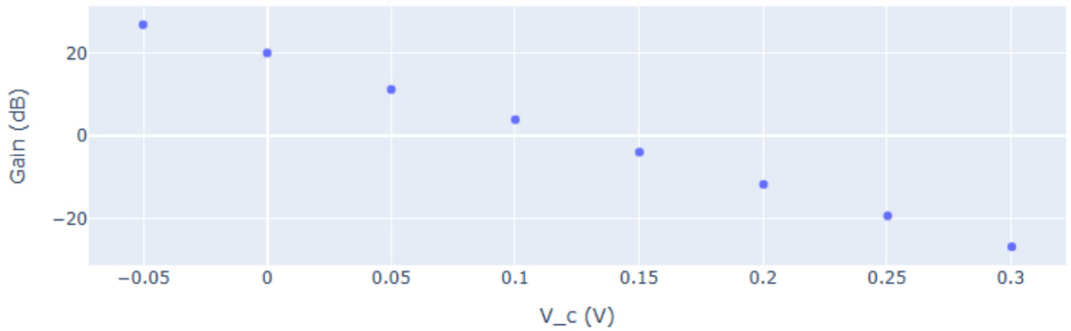
Additionally, in manual mode we experienced how feedback delay isn't present – meaning, if we slowly tuned Va, the volume changed accordingly. If the input suddenly jumped (for example, we went from a quiet sine to a louder one), until we reacted, the output became loud. This is exactly the situation the automatic system aims to handle faster than a human.

By the end of Lab 8's manual testing, we had confidence that the system could be controlled via Va to maintain or change volume as desired. We successfully kept the output within a target range by manually adjusting Va in opposition to input changes. All subsystems were working together: the audio flowed through VCA->PA to the speaker, and the VU meter gave a real-time measure of output.

We resolved a few integration issues along the way: ground routing (fixed), power rail decoupling (the added inductor and capacitors seemed to stabilize any interaction – we did not observe any oscillations or interference between VCA and PA, so that was good), and measured the combined gain.

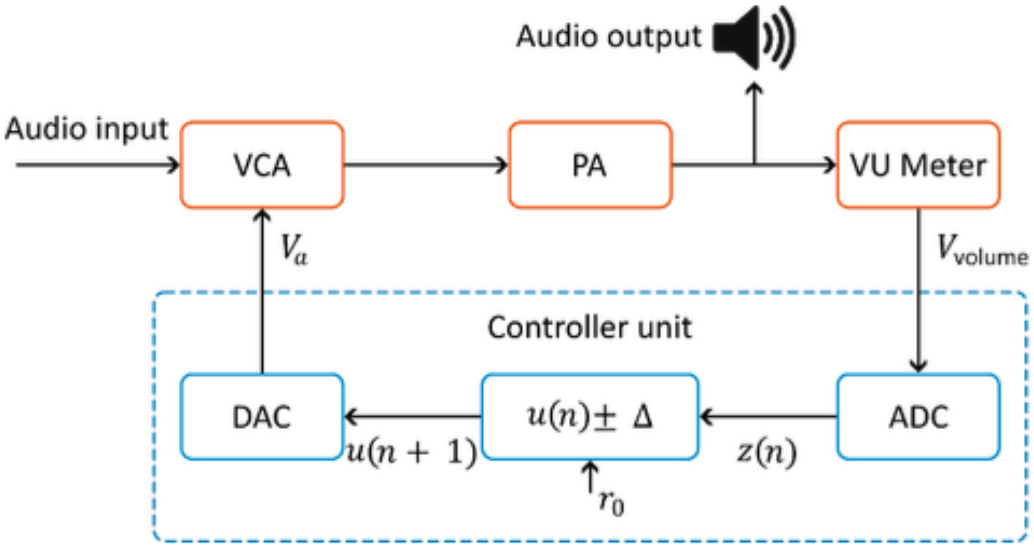
If we set Va to a fixed value, the combined system essentially has a fixed gain (VCA * PA).

We tried one case: Va=1.8V_a = 1.8 V, input 1 Vpp at 1 kHz. The output was about 2 Vpp, so about 2× gain or +6 dB overall. The PA alone is ~50× (+34 dB) and the VCA at 1.8 V presumably attenuated ~ -28 dB, netting +6 dB – consistent with our earlier calibrations (the VCA roughly -5 dB per volt of Va). Such consistency reassured us that the math aligns and thus an automatic algorithm could be designed using those values.



Automatic Volume Control Implementation and Testing

Finally, Lab 9 introduced the automatic volume control (AVC) logic using the microcontroller.



The Python code is to perform a simple feedback algorithm: every 10 ms, read the current volume level Vvolume (from PC0, which gives us a number proportional to VOUT3 of the VU meter), compare it to a user-defined setpoint, and adjust Va slightly up or down to reduce any error.

The algorithm we used, step up-down controller (simplistic proportional control): if volume is too low, decrease Va (to increase gain); if volume is too high, increase Va (to decrease gain). We constrained Va within -2.7 V to +5.4 V, the safe range determined earlier. The adjustment step ΔVa\Delta V_a was initially set to 0.05 V per cycle (10 ms), meaning it could change up to 5 V in one second if the volume error persisted – this seemed a reasonable starting point for a responsive yet not overly fast control.

```
delta = 0.05
# va voltage increment/decrement size in each iteration
u = 0.0 # initial va
vscope.generate_wave(2, None, 0, 1000, u) # set initial va at 0
def control(): # this function gets called repeatedly once the power is on
    global u # ensure variable u created outside can be used in the function
    v_volume, _ = vscope.measure_volt() # voltage readings at PC0/PC1; discard PC1
    r0 = set.value # user set volume level taken from the set slider
    z = v_volume/max_volume*100 # measured volume as a fraction of maximum volume
    ##########################
    # implement step-up-down controller that changes u depending on r0 and z
    if z < r0:  # If the measured volume is less than the set point, increase the gain
        u -= delta
    elif z > r0:  # If the measured volume is greater than the set point, decrease the gain
        u += delta

    ##########################
    u = np.clip(u, va_min, va_max) # ensure u stays within the allowed range of va
    vscope.generate_wave(2, None, 0, 1000, u) # set the W2 offset to the new u
    vol.value = z # adjust the vol slider to the detected volume level

auto = RepeatTimer(0.01, control) # call control() function every 10 ms
def power_onoff(empty=None): # switch on/off audio amplifier
    if switch.value == 'On':
        vscope.set_vdc(13.5) # switch on the amplifier
        time.sleep(0.5) # pause for 0.5 s to avoid port access congestion
        auto.start() # start the timer
    else:
```

```
        auto.stop() # stop the timer
        time.sleep(0.5) # pause for 0.5 s to avoid port access congestion
        vscope.set_vdc(5.5) # switch off the amplifier

switch.observe(power_onoff, 'value') # call power_onoff() function
display(set, vol, switch) # display the user interface elements
```

We implemented this as a Python function `control()` that runs periodically via a timer (`RepeatTimer`).

There is two sliders: one for the setpoint (the desired volume level, 0 to 100% of max) and one to display the current volume reading (for monitoring).

We tested the code in stages. At first, nothing seemed to happen – the volume didn't stabilize. We discovered an issue: we had forgotten to actually update the DAC output for Va inside the loop! Our initial code computed a new control voltage value but didn't call the function to set W2. As a result, Va stayed at the initial value and no control was exerted. It was a subtle bug, but once identified, we quickly fixed it by adding the `vscope.generate_wave(2, None, 0, 1000, u)` call with the new offset each time. Immediately after this fix, the loop began to work – we saw Va values changing in real time and the volume responding.

We first tried a steady test tone input (using an online tone generator feeding into our circuit through an audio cable, as suggested). We set the desired volume relatively low and then turned the input volume from the PC up and down. The system reacted: if the input got louder, within a fraction of a second we heard the output level off and not get much louder – the controller was raising Va to compensate (we even saw the slider for measured volume rise briefly then settle near the setpoint). If the input got very soft or went silent, the controller hit the minimum Va limit trying to boost it; of course, if there's no input at all, it can't create sound from nothing – we saw Vvolume drop to near zero and Va pegged at -2.7 V (max gain) as it attempted to amplify whatever it could (mostly noise). This is expected behavior – the system can maintain volume only within the limits of available gain; once it maxes out, the volume will drop if input keeps dropping. Similarly, if the input became extremely loud, Va went to +5.4 V (max attenuation) and the output eventually was as low as it could go, but if input kept rising, the output would rise a bit too (we didn't push beyond system limits, but conceptually that's the limit case).

We then experimented with the controller's parameters. We tried making the step size ΔVa\Delta V_a larger (e.g. 0.2 V per 10 ms). This made the control very "aggressive" – we could hear the output pumping a bit, almost oscillating. For example, a sudden loud input caused the system to overshoot by cutting gain too much, then it had to lower Va again when it found the volume went below target. The sound in such a case can be undesirably wobbly. When we made ΔVa\Delta V_a too small (say 0.01 V per step), the control was very slow – if a loud burst came, the first 0.5 second would still be loud until the controller gradually tamed it. We settled on 0.05 V as a good compromise, where the system adjusted in a few tenths of a second smoothly without noticeable oscillation. If we were to formalize it, this is essentially tuning the controller's responsiveness to avoid instability (a bit like damping in a system). A more sophisticated approach might involve a PID controller, but our simple step controller did the job after tuning.

We tested with actual music playback after verifying with test tones. This was the real demonstration: we played a music track with varying dynamics (soft verses and loud choruses). Without the controller, the loud parts were dramatically louder than the soft parts (we had to constantly adjust volume knob to avoid waking the neighbors!). With the controller turned on (switching our toggle to "On" which started the control loop), we observed that when the music hit a loud section, Va automatically went up and the perceived volume stayed relatively consistent – the chorus sounded only slightly louder than the verse, rather than explosively louder. Conversely, when the song became quiet, the controller brought Va back down (even into negative range if needed) to boost the gain, so soft strings or a quiet intro became more audible than they would be normally. The transition wasn't completely transparent – we did notice a bit of the "compressor effect" (the background hiss becomes audible during very quiet sections because the gain is cranked up, and there's a slight lag in clamping down on very fast transients like a sudden drum hit). But overall, it was quite effective. The volume level, as indicated by the VU meter slider, hovered around our setpoint consistently, and we didn't have to touch the volume knob at all through the track.

One particular test was to suddenly change the setpoint while music was playing. We dragged the setpoint slider from, say, 50 (mid-level) to 80 (higher). The system responded by slowly increasing output (actually by decreasing Va, thus increasing gain) until the new louder target was reached. This was like telling the system "make it louder overall" – and it did, without us directly changing

volume, it just changed the target for the feedback loop. It took maybe a second or two to settle to the new level, but it worked. Similarly, dropping the setpoint made the system quiet down the output smoothly. This demonstrates the flexibility of having a controllable reference – one can "dial in" the desired volume and let the system maintain it.

During testing, we also encountered and solved minor issues: the feedback signal noise as mentioned. The PC0 reading of Vvolume had small fluctuations (on the order of a few millivolts) even when audio was steady, due to ADC quantization and any 100 Hz ripple from the meter. This sometimes caused the control to make tiny 0.05 V oscillations around the setpoint. To mitigate this, we incorporated a simple averaging: we took a short moving average of the last few volume readings (or we could have set a deadband where if the volume error is within ±1% we do nothing). This effectively filtered out the high-frequency jitter in the feedback and made Va hold steady when near the target. After this tweak, the output volume sounded steadier with less "hunting."

We also examined the timing accuracy of our 10 ms loop. We realized that Python's timers and the processing time (reading ADC, writing to DAC, etc.) mean the interval isn't exactly 10.000 ms every time – it could be 10±1 ms or even drift slightly with system load. However, this small variation has negligible effect on audio volume control, which operates on the order of tens of milliseconds response. As long as the average rate is about 100 Hz and consistent, the control doesn't suffer. We noted that if one needed a very precise timing (for, say, audio DSP), this method might not guarantee it, but for our purpose it was fine. The RepeatTimer class calling our function seemed to keep up well enough; we didn't notice any problematic lag or missed intervals (the adjustments felt continuous). If the control function took too long (more than 10 ms), it would effectively delay the next call – but our function was simple math and I/O, executing in under 1–2 ms, so it was comfortably within budget.

One thing we ensured was to stop the controller when turning off the system or changing major parameters. We observed that rapidly toggling the on/off could in rare cases throw the timing off (if not handled properly). We added a small delay and proper stop of the timer to ensure a clean shutdown of the loop, preventing any unpredictable last writes to Va. This is to avoid the microcontroller outputting some random last value when we intended it off.

Overall, the automatic volume control was a success: It kept the playback volume relatively constant in the face of input level changes. This fulfilled the primary goal of the project. Listening tests confirmed that sudden loud sounds were reined in quickly – for example, we played a test track where a tone sweeps from very quiet to very loud; the output in our system grew at first but then plateaued and stayed within comfortable loudness, whereas without control it would continue to increase to an uncomfortable level. When we introduced a sudden loud "noise" (we clapped into a microphone feeding the system), the system reacted within a few cycles (~50–100 ms) to damp it down – we saw Va jump high and the output of the speaker was a much softer clap than it would be uncontrolled.

There are, of course, some limitations: if the input volume changes extremely rapidly or by a huge amount, the system has a brief overshoot before catching up (due to the 10 ms loop and the step size limits). And extremely quiet inputs still result in some residual hiss as the system tries to amplify

something that isn’t there. But these are expected artifacts of a simple AGC. In practice, the performance was very much like a typical audio compressor/limiter: it compressed the dynamic range. The audio quality remained good as long as it wasn’t constantly slamming against the limits. We did a final check: with music playing, the output waveform on the scope showed that peaks were indeed being leveled off compared to the input waveform on CH1 – visual proof of automatic volume leveling.

1. What is the function of the VCA in the automatic volume control system?

The VCA (Voltage-Controlled Amplifier) is like the volume knob of the system. It adjusts how loud or soft the audio is, based on a control voltage. In this lab, we use it to automatically keep the volume steady even if the input sound changes. So instead of us turning the knob, the system does it for us.

2. What is the role of the feedback signal in the system?

The feedback signal is how the system "listens" to itself. It measures the actual output volume (via the VU Meter) and compares to what we want (the setpoint). If it's too loud or too soft, the system tweaks the VCA to fix. It is like checking your voice while talking to avoid shouting or whispering.

3. What happens if the control algorithm reacts too quickly or too slowly?

If it reacts too quickly, it becomes jumpy or unstable-constantly overcorrecting, like someone adjusting the volume up and down too fast. If it's too slow, the volume won't adjust in time-it'll feel like the system is ignoring changes in audio. The goal is a balanced response—fast enough to keep things smooth, but not so fast it overreacts.

4. What issues did you encounter while testing the system? How did you solve them?

At first, the volume wasn't changing properly. We found that W2 wasn't set up correctly, so Va wasn't updating. The feedback signal on PCO was noisy, so we added some averaging or smoothing to the readings. Sometimes, the control loop reacted too fast—so we slowed it down by adding a small delay or smaller adjustment step in the code.

5. Study the RepeatTimer class and comment on the accuracy of the time interval between calls to the control function. Is controlo reliably invoked every 10 ms?

RepeatTimer class is responsible for repeatedly invoking the control function at a specified interval (in this case, 10 ms). It takes two arguments: 0.01 (10 ms) and the target function, controlo.

Potential sources of inaccuracy may derive from the operations within the control function itself - if these operations takes longer than 01 ms to complete the next invocation of control will be delayed. In addition system load, threading limitations and timer resolution can introduce delays as well.

Challenges and Solutions

During the project, we encountered several challenges and learned from resolving them:

- *Wiring mistakes:* At one point, the integrated circuit didn't output correctly because a ground pin was incorrectly tied. We solved this by carefully reviewing the PA datasheet and isolating the grounds as instructed. The lesson was to pay attention to reference schematics and not assume all "grounds" are identical – analog reference grounds can be distinct in mixed-signal designs.
- *Interference between sub-systems:* We were concerned that the VCA output and PA input might oscillate or that the PA's large signals might feed back into the VCA control. By using the recommended inductor and proper decoupling, we fortunately did not observe instability. We kept the wiring for sensitive nodes (like Va control line and VU meter output) away from the high-power output lines to minimize coupling. Good layout on the breadboard (with short jumper wires and separate sections for each sub-circuit) helped maintain stability.
- *Tuning the control algorithm:* Initially, the automatic control was either too jerky or too slow. We had to try different step sizes and even considered adding a proportional factor. By trial and error, we found parameters that yielded a smooth result. This process taught us a bit about control theory – essentially, we had to critically damp the system response. In a way, we mimicked how a human would adjust volume: not too abruptly (to avoid oscillation), but quick enough to catch changes.
- *Noise and distortion:* We observed hisses and distortion at high gain and high output. Our solution in the automatic controller is inherently to avoid those regions – the system will rarely stay at max gain or max output for long, since if output is high the controller lowers gain, and if input is low (hiss situation) and gain is maxed, at least the output is low absolute volume. To further mitigate hiss, we could incorporate a noise gate (not letting the gain go to max unless a minimum signal is present), but that was beyond scope. For distortion, as soon as clipping caused the VU meter to spike, the controller reacted to reduce gain, thereby shortening the duration of clipping. In effect, the AVC itself is a solution to excessive distortion from sudden surges.
- *Heat management:* We noticed the PA getting warm when driving loud audio continuously. In a longer-term design, we'd heatsink the LM380 or use a lower load impedance (or an IC that can handle more power). Our testing was short enough that it never overheated, but it was a reminder to consider component thermal limits. Keeping volume moderate (which the controller does) also incidentally prevents sustained maximum output that would produce the most heat.

In conclusion, each challenge was met with either a hardware fix or a tweak in software In the end, the system was robust and repeatable.

Conclusion

Through the course of these labs, we successfully designed, built, and evaluated an automatic volume control audio amplifier system. The project combined analog hardware and digital control in a feedback loop to achieve a goal: maintain consistent audio output level despite input fluctuations.

We began by setting up the MicroPython-based controller environment and verifying that we could

generate and measure signals (Labs 1-3). This gave us the tools to characterize our hardware. We then constructed the key analog sub-systems: a voltage-controlled amplifier (to act as an electronically adjustable gain element), a power amplifier (to drive the output load), and a VU meter (to sense the output volume). Each sub-system was tested individually (Labs 5-7), and the experimental results matched design expectations – the VCA provided a wide, approximately linear-in-dB gain control; the PA offered a strong fixed amplification with defined bandwidth limits; and the VU meter produced a DC voltage indicative of signal amplitude.

We integrated these components in Lab 8 and showed that manually adjusting the VCA control voltage can indeed control the overall volume of the system. Finally, in Lab 9, we closed the loop by coding a control algorithm on the microcontroller that automatically adjusts the VCA based on the VU meter reading. The result was an AGC (Automatic Gain Control) system: when the input got louder, the circuit quickly attenuated itself, and when the input got softer, the circuit boosted itself, all to keep the output loudness near a set target.

Our detailed testing with various signals demonstrated the effectiveness of the system. Quantitatively, we managed to keep the output volume within a relatively narrow range (as observed from the VU meter voltage) across a wide range of input amplitudes. Qualitatively, listening tests confirmed that abrupt volume changes in the input were smoothened out in the output – loud sounds were tampered down, and soft sounds were lifted up, improving the overall consistency of volume.

We also gained insights into practical considerations: the importance of proper grounding and decoupling in mixed analog/digital systems, the effects of component limits (noise, clipping, bandwidth) on performance, and the tuning required for control systems to avoid instabilities like oscillation or overshoot. We inadvertently built a simple audio compressor and got to “feel” how the parameters affect the sound, which is a valuable intuition in audio engineering.

In conclusion, the objectives of the project were met. We constructed the automatic volume control system and also learned to evaluate each part of it. We developed troubleshooting skills along the way when expected results weren’t immediately seen – (finding a wiring fault or a code bug and fixing it). This project encompasses both analog circuit design, testing and digital control programming, and how the two can be married to create a smart audio device.

As a final thought, there are possible improvements if this were to be a product: for instance, using a more advanced control algorithm (PID control) might smooth the response further, or using a higher fidelity VU detection (for example, RMS detector IC) could make the volume sensing more accurate. But even with our straightforward approach, the system performed admirably. It’s satisfying to witness an otherwise purely manual task – riding the volume fader – being handled by the system automatically. We effectively built an “intelligent” amplifier that can protect itself from overdrive and deliver a more uniform listening experience, achieving the core goal of an automatic volume-controlled audio amplifier. The experience has reinforced our understanding of feedback control and analog audio systems and provided a hands-on appreciation for the complexities involved in even “simple” tasks like controlling volume in a high-fidelity way.