NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

School of Electrical and Electronic Engineering

EE2073 Introduction to EEE Design and Project

Academic Year 2023/2024 Semester 2

Automatic Volume Control for Audio Amplifier System Laboratory Manual

Dress code in the laboratory:

- Work shirt that covers the upper torso and arms
- Lower body clothing that covers the entire leg
- Closed-toe shoes that cover the top of the foot

For EEE CA guidelines, go to https://ntu.sg.eee-EEEStudentCAGuidelines or scan the QR code below.

Lab 1: Introduction to MicroPython and Microcontroller Board

1. Objective

To introduce the controller unit and setup a Python development environment for writing programs that control the audio amplifier system via the controller unit.

2. Introduction

A microcontroller is a compact integrated circuit that contains a processor core, memory, and input/output peripherals, all on a single chip. It often serves as the brain of an embedded system and extensively used in Internet-of-Things (IoT), robotics, consumer electronics, and industrial automation. Its versatility, low cost, and compact size make it ideal for systems requiring control, computation, and connectivity in a small form factor. We will be using an STM32 microcontroller board developed by STMicroelectronics to control our audio amplifier system. Here are its specifications:

- 32-bit ARM Cortex M4
- 192 kB RAM
- 512 kB flash memory
- ADC, 3 channel 12 bits
- DAC, 2 channel 12 bits

The board is pre-loaded with MicroPython firmware, allowing us to utilize Python programming language to command it.

The microcontroller board itself is not capable of directly driving the audio amplifier system, and therefore it is supplemented by a custom-designed interface board called VScope. The VScope board is physically stacked on top of the microcontroller board. Together, this controller unit is capable of:

- Supplying variable DC voltages of up to ±15 V.
- Generating waveforms.
- Reading voltages.

Figure 1.1 shows the controller unit's top view. The unit can be powered by connecting to a computer using a USB cable through its Type B Mini USB port, which is also used for communicating with the computer. VDCP and VDCN supply variable DC voltages. W1 and W2 are used for outputting voltage waveforms. CH1 and CH2 are for reading time-varying voltage signals, and PC0 and PC1 are for measuring fixed voltages.

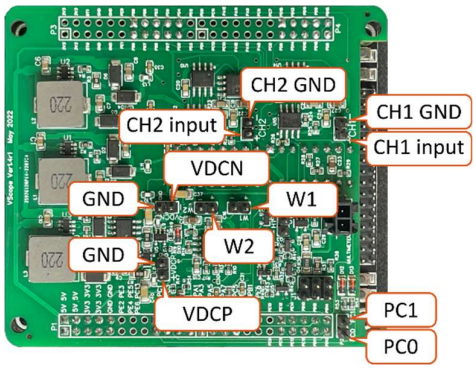


Fig. 1.1. Top view of the controller unit with locations of important pins.

Figure 1.2 depicts the automatic volume-controlled audio amplifier system. It is controlled from a computer through the controller unit. A Python program, which we will develop in Jupyter Notebook, sends commands to the controller unit and triggers MicroPython functions. These functions either set

or read voltage signals on the controller unit pins, providing the necessary power, signal, and control voltage to the audio amplifier circuit, or monitoring the audio volume feedback. The volume feedback is sent back to the computer and processed in the Python program to issue new commands to the controller unit for adjusting the volume.

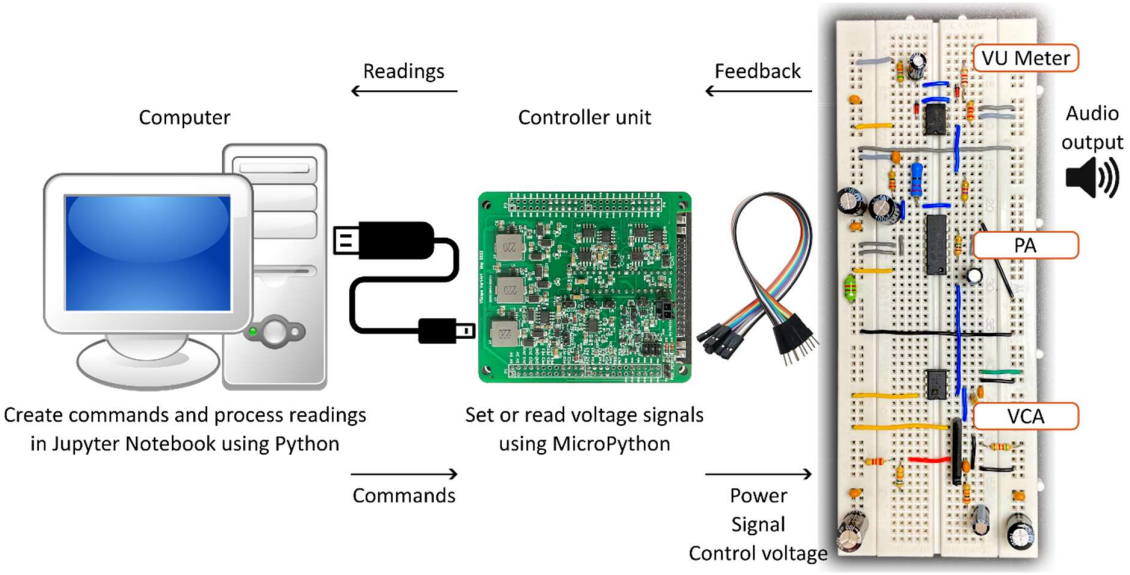


Fig. 1.2. Schematic of the automatic volume-controlled audio amplifier system. VCA: voltage-controlled amplifier; PA: power amplifier; VU Meter: volume unit meter.

3. Equipment and Components

Equipment

- Computer
- Controller unit
- USB cable with Type A and Type B Mini connectors
- Digital multimeter

Components

- Resistors
- Jumper wires
- Breadboard

4. Project Tasks

4.1. Connecting to the controller unit using PuTTY

Connect the controller unit to the computer. The unit will power up and appear as a storage drive named *BLACKF407VE*. It should contain four files as shown in Fig. 1.3.

The *boot.py* file controls the unit's boot-up process. It imports `machine` and `pyb` modules, which contain hardware-specific classes and functions for microcontrollers and peripherals. The *main.py* file is loaded after *boot.py*, and it is where we can write a custom firmware for the unit. The file is currently empty but will be used later in the project to configure the controller unit, manage hardware resources, and handle inputs and outputs.

Determine the serial port number that the controller unit is connected to by opening Device Manager. Refer to Fig. 1.4 as an example, which illustrates its connection to COM32 serial port. Take note of this number.

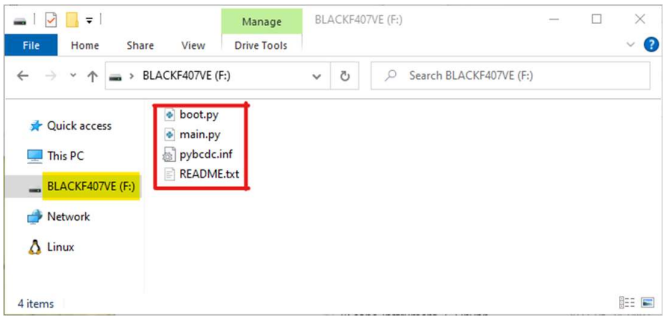


Fig. 1.3. Files in the microcontroller unit shown in File Explorer when connected via USB.

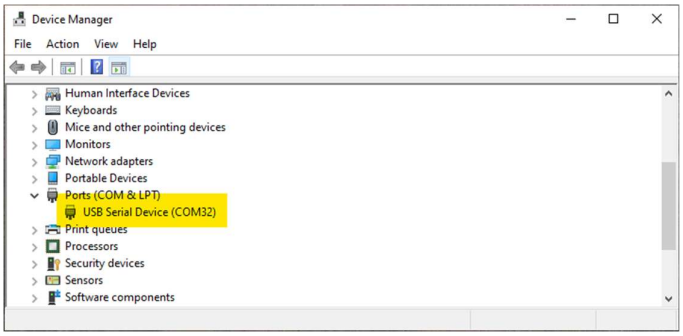
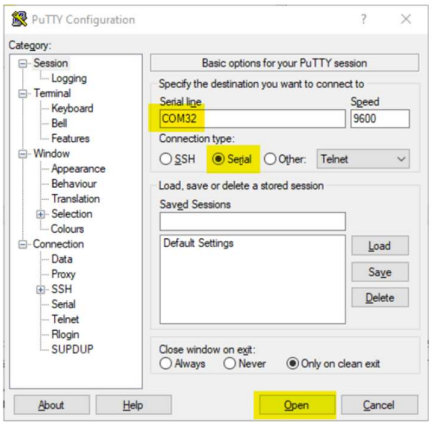


Fig. 1.4. Serial port number of the controller unit shown in Device Manager.

To issue commands directly within the controller unit, launch PuTTY (https://www.putty.org/), a terminal emulator that facilitates serial port connections. Choose Serial as the connection type and enter the previously noted port number (e.g., "COM32") in the serial line field and press Open, as shown in Fig. 1.5(a). Once the connection is established, an interactive MicroPython prompt indicated by `>>>` will show up, where commands can be entered. Refer to Fig. 1.5(b) for an example.

(a)



(b)

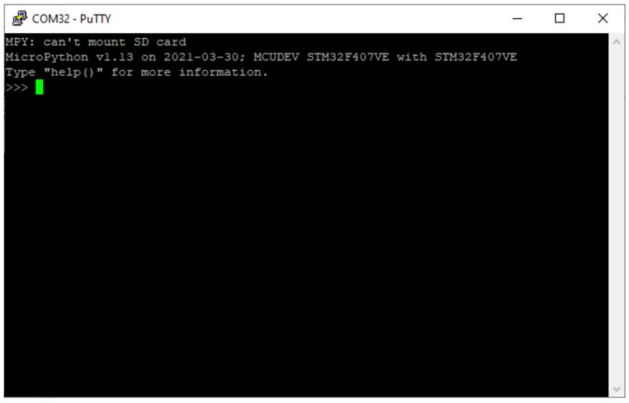


Fig. 1.5. (a) PuTTY settings for establishing a connection with a serial port. (b) MicroPython prompt displayed upon successful connection.

Enter the following command in the prompt, bearing in mind that Python is case-sensitive.

```
>>> print('Hello EE2073!')
Hello EE2073!
```

Test basic MicroPython commands, such as controlling LEDs on the unit using the `pyb.LED` class. We can switch on, off, or toggle the LED status. Omit everything after `#` in each Python line as is is just a comment and not read by the interpreter.

```
>>> pyb.LED(1).on()     # switch on LED D2
>>> pyb.LED(2).on()     # switch on LED D3
```

```
>>> pyb.LED(2).off()    # switch off LED D3
>>> pyb.LED(1).toggle() # toggle the status of LED D2
```

Python has a built-in help function that is useful for listing available classes and functions.

```
>>> help(pyb)     # list classes and functions in the pyb module
>>> help(pyb.LED) # list sub-classes and functions in the pyb.LED class
```

For detailed information on various MicroPython modules, classes, and function, refer to MicroPython documentations available at https://docs.micropython.org/en/latest/.

We want to do more than just simple LED controls on the controller unit. For instance, we want to be able to adjust voltages on the variable DC voltage supply pins. Place a 20 kΩ resistor on a breadboard and connect it to VDCP and GND on the controller unit using jumper wires. Then, connect a digital multimeter across the resistor to measure the DC voltage. The default voltage of around 5.5 V should be measured. Let us change this to 8 V by entering the following lines in the MicroPython prompt. See the circuit diagram of the variable DC voltage supply in Appendix A for more information.

```
>>> volt = 8
>>> from pyb import Pin
>>> from machine import SPI
>>> spi = SPI(sck=Pin('PB13', Pin.OUT), mosi=Pin('PB15', Pin.OUT),miso=Pin('PB14',
          Pin.IN))
>>> dz = Pin('PB12', Pin.OUT)
>>> y = 312 - 1020/volt
>>> dz.value(0)
>>> spi.write(b'\x11')
>>> spi.write(bytes((int(y),)))
>>> dz.value(1)
```

After entering the last line, the DC voltage reading on the digital multimeter will change, which should be around 8 V. To further modify the voltage, this time to 12 V, enter the following lines.

```
>>> volt = 12
>>> y = 312 - 1020/volt
>>> dz.value(0)
>>> spi.write(b'\x11')
>>> spi.write(bytes((int(y),)))
>>> dz.value(1)
```

Verify that the voltage across the resistor is approximately 12 V. We have successfully executed our first set of MicroPython commands. Proceed to close the PuTTY session, so that the serial port is released for other computer applications to control it.

4.2. Setting up Python development environment

Although the interactive prompt allows for simple MicroPython commands to be issued directly within the controller unit, it is not ideal for executing complex tasks. To control the audio amplifier system, it is more convenient to write a Python program that runs on the computer and communicates with the controller unit via the serial port.

Miniconda (https://docs.conda.io/en/latest/miniconda.html) is a lightweight version of Anaconda that includes just the essential components needed for setting up a Python development environment. It is suitable for minimal installations with manual package additions based on user requirements. Install Miniconda and launch Anaconda Prompt (miniconda3) to start a new Command Prompt with the following prompt line.

```
(base) C:\Users\username>
```

The `base` in the prompt line indicates that we are currently in the default base environment.

When beginning a new Python project, it is highly recommended to create a project-specific virtual environment. A virtual environment offers benefits such as isolation, simplified dependency

management, reproducibility, easy clean-up, and portability. To create a virtual environment named `ee2073` for the project, enter the following command.

```
(base) C:\Users\username> conda create --name ee2073
```

Activate `ee2073` virtual environment.

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

Note that the `base` at the beginning of the prompt line has been replaced with `ee2073`, indicating that we are now in `ee2073` virtual environment. To exit the current environment and return to the `base` environment, use the command `conda deactivate`. However, for now, let us stay in the `ee2073` environment to install necessary packages for the project.

Jupyter Notebook (https://jupyter.org/) is a highly recommended interactive prototyping platform, ideal for projects like this. Install Jupyter Notebook in the `ee2073` virtual environment we just created. We also need to install Jupyter Widgets—interactive widgets for Jupyter Notebook.

```
(ee2073) C:\Users\username> conda install notebook=6.5.4 ipywidgets=7.6.5
```

Make sure the `ee2073` virtual environment is activated when issuing the installation command. Installing these packages pulls their dependencies, i.e., other packages required by Jupyter Notebook and Jupyter Widgets. The process may ask for an administrator credential. Click "No" to complete the installation.

There are other packages that are needed for the project in the `ee2073` virtual environment.

```
(ee2073) C:\Users\username> conda install pyserial=3.5 numpy=1.25 plotly=5.9.0
```

PySerial (https://pythonhosted.org/pyserial/) enables Python programs to communicate with devices via serial ports. NumPy and Plotly are essential for data processing, analysis, and visualization. Installing these additional packages also pulls in their dependencies.

4.3. Executing MicroPython commands in Jupyter Notebook

Open Jupyter Notebook (ee2073) and create a new notebook. Save it as *lab1_YourName.ipynb*. Use the notebook file to replicate examples presented in this laboratory manual and to complete exercises. Remember to keep a full record of your attempts throughout each laboratory session.

To establish communication between the controller unit and computer, we need to identify the serial port. Use the following lines to accomplish this.

```
import serial
import serial.tools.list_ports

ports = serial.tools.list_ports.comports() # get list of serial ports with devices
```

The `ports` list contains devices connected to all serial ports on the computer, where one of them should be the controller unit. We can identify the controller unit by its unique hardware information, such as the vendor ID (`vid`) and product ID (`pid`). For our STM32 microcontroller unit, the vendor ID is `61525` and the product ID is `38912`. Once the device is found, we open the communication channel using the `Serial` class as shown below.

```
VID = 61525 # vendor ID of the device
PID = 38912 # product ID of the device

for p in ports:
    if p.vid == VID and p.pid == PID:
        try:
            device=serial.Serial(p.device)
        except serial.SerialException: # raised if the device not available
            print('Reconnect the controller unit.')
```

```
if device is None:
    raise Exception('No suitable device detected.') # if no matching device found
```

Now, we can use the `device` object—an instance of the `Serial` class—to talk to the controller unit. Calling the object retrieves basic information about the connection.

```
device # check the connection information
```

It prints out the connection information, e.g., `Serial<id=0x207f06dadd0, open=True>(port='COM32', baudrate=9600, bytesize=8, parity='N', stopbits=1, timeout=None, xonxoff=False, rtscts=False, dsrdtr=False)`.

Issue simple MicroPython commands to the controller unit using the `write()` function.

```
device.write(bytes('pyb.LED(1).toggle()\r','utf-8')) # toggle the status of D2 LED
```

The `bytes()` function converts the string to bytes using the UTF-8 encoding, and then the `write()` function passes it to the controller unit. The string must end with a carriage return character, `'\r'`, which marks the end of the command string and is necessary to proper registration. In the example above, the command toggles the status of LED D2, like the previous exercise where the same command was issued directly in the controller unit.

👉 Set the DC supply voltage at VDCP to two different values between 5 and 12 V in Jupyter Notebook. Pass the corresponding MicroPython commands using the `write()` function. Use a backslash to escape special characters in Python strings, such as the backslash (`'\\'`) and single quotation mark (`'\''`). Once the commands are issued, use the digital multimeter to check whether the voltage across the resistor has been successfully changed to the desired values.

4.4. Closing connection

To close the communication channel when it is no longer needed, issue the following Python line.

```
device.close() # close connection
```

It releases the port, allowing other program instances to access it.

5. Open-Ended Questions

What is SPI? Refer to the MicroPython documentations and Appendix A to find out. Elaborate on the statement `spi = SPI(sck=Pin('PB13', Pin.OUT), mosi=Pin('PB15', Pin.OUT), miso=Pin('Pin14', Pin.IN))`.

6. Notebook

Remember to save your notebook with all generated outputs and relevant discussions. You are strongly encouraged to use Markdown cells to include explanations, observations, and discussions alongside your code. For examples on how to use Jupyter Notebook, including Markdown cells, go to https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/examples_index.html.

In your notebook for the session, consider including the following:

- Key points observed during examples and exercises
- Discussions on interesting trials and their outcomes

Print your notebook to a PDF file by selecting Print from the browser menu and choosing Save as PDF. Upload the PDF file to NTULearn before leaving the laboratory, regardless of whether you have finished all the tasks or not. It is important to upload everything you have for the day, and you are not permitted to change them once uploaded. Any unfinished work can be included in the notebook for the next laboratory session.

Lab 2: Data Acquisition and Visualization

1. Objective

To install custom firmware on the controller unit, acquire time-sampled readings by sending command strings, and learn to visualize the data.

2. Introduction

To measure signals and transfer the data to a computer, an analog-to-digital converter (ADC) can be employed. The ADC acquires data through sampling, where an analog signal is measured at discrete time intervals. The sampling rate, also known as the sampling frequency, determines how frequently the signal is sampled. The process generates a series of discrete signal values over time as illustrated in Fig. 2.1.

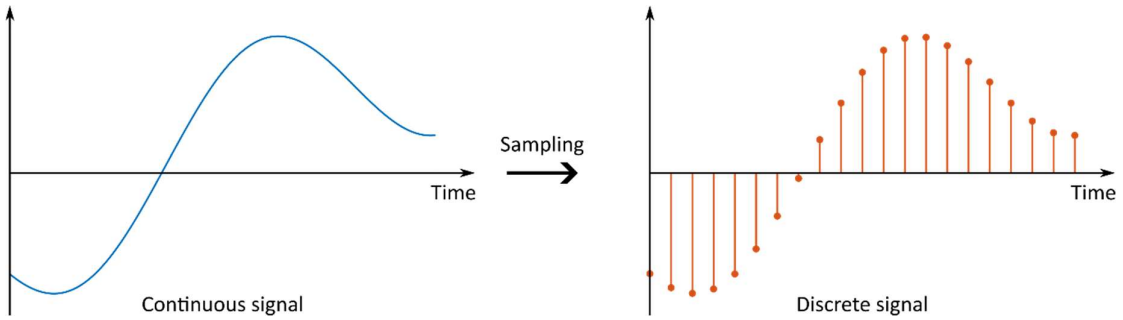


Fig. 2.1. Illustration of digital sampling of an analog signal.

The quality of the conversion process is determined by the sampling rate. A higher sampling rate leads to improved accuracy in representing analog signals. To accurately reconstruct the original signal from the sampled data, the sampling frequency must be at least twice the maximum frequency of the analog signal. This requirement is known as the Nyquist sampling frequency.

After sampling, the next step is to convert the analog signal into a digital format. This conversion process, depicted in Fig. 2.2. is known as analog-to-digital conversion.

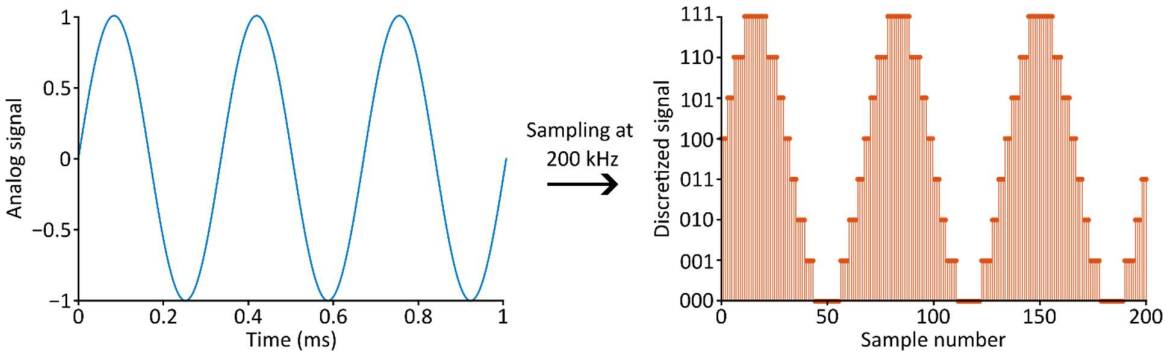


Fig. 2.2. Example analog-to-digital conversion using a 3-bit ADC sampled at 200 kHz.

The precision of the analog-to-digital conversion process relies on the number of bits used by the ADC to represent the sampled value. Higher resolution ADCs allow for more divisions within a given voltage range, resulting in the ability to detect smaller voltage changes. For instance, an 8-bit ADC can express $2^8 = 256$ voltage levels, while a 12-bit ADC can represent $2^{12} = 4096$ levels. The least significant bit (LSB) also varies depending on the operating voltage range of the ADC. If the full scale of the signal is 10 V, the LSB for a 3-bit ADC corresponds to 1.25 V. However, if the signal range is only 1 V, the LSB

for the same ADC becomes 0.125 V. Similarly, for a 12-bit ADC, the LSB for the two ranges mentioned above are 2.44 mV and 0.244 mV, respectively. Therefore, to be able to detect smaller changes, it is necessary to use a higher resolution ADC on a smaller detection range.

To generate analog output in the form of DC or AC voltages from digital values sent from a computer, a digital-to-analog converter (DAC) is employed. The performance of a DAC is determined by the sampling rate and the number of bits used to represent the digital values. Figure 2.3 illustrates how a 3-bit DAC can generate an analog sinusoidal waveform, along with a comparison to a 16-bit DAC generating a sinusoid of the same amplitude and frequency. A DAC with a short settling time and high slew rate—the rate at which an amplifier can respond to change in input—can generate high-frequency signals effectively, as it can quickly and accurate change the output to a new voltage level.

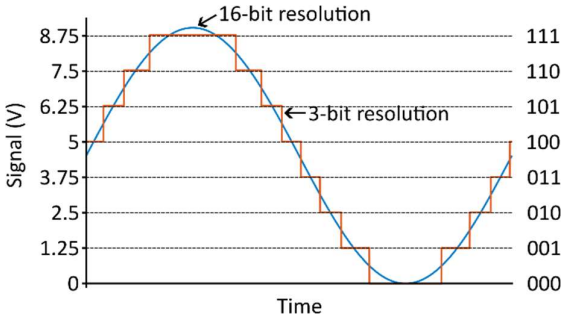


Fig. 2.3. Illustration of an analog signal generation from a 3-bit and a 16-bit DAC.

To process and visualize the digital data acquired from an ADC, NumPy (https://numpy.org/) and Plotly (https://plotly.com/python/) can be utilized. NumPy provides a comprehensive set of mathematical functions for performing various operations on numerical arrays. Plotly is a Python library that enables the creation of interactive graphs, making it suitable for use with Jupyter Notebook. Together, these libraries offer powerful tools for analyzing and visualizing signals.

3. Equipment and Components

Equipment

- Computer with ee2073 conda environment
- Controller unit
- USB cable with Type A and Type B Mini connectors
- Digital multimeter

Components

- Resistors
- Jumper wires
- Breadboard

4. Project Tasks

4.1. Custom firmware Installation

The process of issuing commands to the controller unit can be simplified by creating a custom firmware that resides in the flash memory of the controller unit. Replace the *main.py* file in the controller unit that runs at the device boot-up with a pre-written one available in NTULearn. See Appendix B for listing of this firmware. This file contains necessary initializations, basic operations, and low-level functions for setting the DC voltage, generating waveforms, and measuring signals. Calls to these functions and the matching parameters can be passed from Jupyter Notebook through the serial port. After replacing the *main.py* file, eject the controller unit from the computer and then disconnect/reconnect it to load the new firmware.

4.2. Setting DC supply voltage

We can set the DC supply voltage using the predefined string command in our custom firmware. Connect a 20 kΩ resistor to the VDCP and GND pins and setup a digital multimeter to measure the voltage across the resistor. Open a new Jupyter Notebook and save it as *lab2_YourName.ipynb*. Establish a communication channel with the device.

```
# communication channel setup
import serial
import serial.tools.list_ports

VID = 61525
PID = 38912

ports = serial.tools.list_ports.comports()

device = None
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.')
```

The command string for setting the DC supply voltage takes the form `'dzAAAA\r'`.

- `'dz'` instructs the controller unit to set the DC supply voltage.
- `'AAAA'` represents the intended DC voltage value in 10 mV increments, with zero-padding to fill up four-character spaces.
- `'\r'` is the carriage return character.

For instance, to set the VDCP to 8 V, the command string becomes `'dz0800\r'`. To create this command string in Python, we can use the following expressions:

```
vdc = 8
cmd_setvdc = 'dz' + str(int(vdc*100)).zfill(4) + '\r'
```

In this code snippet, `int(vdc*100)` converts the voltage value to an integer by multiplying it by 100 and then typecasting it. The `str().zfill()` converts the integer value to a string and pads it with leading zero characters (`'0'`) until the string reaches the specified length. The `+` operator concatenates the strings together.

To send the command string to the controller unit, we can use the `write()` function.

```
device.write(bytes(cmd_setvdc, 'utf-8'))
```

After executing this code, the controller unit should receive the command and set the DC supply voltage accordingly. The voltage on the digital multimeter should change to around 8 V. Note that the voltage on the negative DC supply, VDCN, is of around the same magnitude as on VDCP, but with the opposite sign. Make sure to load VDCN with a 20 kΩ resistor before checking this.

👉 Set the DC supply voltage at VDCP to 12 V by sending a command string from Jupyter Notebook.

4.3. Generating timed voltage signals

The controller unit features two channels, W1 and W2, for generating waveforms. Waveforms can be defined by their shape, frequency, amplitude, and offset, and hence the command string should include these parameters.

The command string for generating a voltage signal takes the form `'sBCCDDDEEEEEEEFFFFGGGG\r'`.

- `'sB'` indicates the channel for generating the waveform with `'1'` or `'2'` replacing `'B'` to represent W1 or W2, respectively.
- `'CC'` denotes the waveform shape. Currently, three shapes are supported: sinusoidal (`'00'`), triangular (`'10'`), and sawtooth (`'11'`).
- `'DDD'` specifies the number of samples per oscillation cycle.
- `'EEEEEEE'` represents the oscillation frequency in Hz.
- `'FFFF'` is the signal amplitude in 10 mV.
- `'GGGG'` is the DC offset in 10 mV.

All digits are zero-padded and do not include fractional parts. The Python expressions for creating a command string that produces a sinusoidal waveform with a frequency of 2 kHz, an amplitude of 0.5 V, and a 0 V DC offset on W1 are shown below.

```
ns = 64             # samples per cycle; 64 generally enough
freq = 2000         # frequency
amp = 0.5           # amplitude
offset = 0          # offset
cmd_gensin = 's100' # W1 sinusoidal
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'))
```

Connect W1 to a 20 kΩ resistor. Use a digital multimeter set to AC mode to measure the root-mean-squared (RMS) voltage and frequency of the waveform. If the DC bias is zero, the RMS voltage, V_{RMS}, and amplitude, V_p, of a sinusoidal waveform are related through $V_{\text{RMS}} = 0.707V_p$. However, we should note that despite setting the DC offset to 0 V in our command, the generated signals may still contain residual offset. We will address the removal of this DC bias later in the course.

👉 Generate a triangular waveform of the same amplitude, frequency, and offset. Observe the change in RMS voltage on the digital multimeter. Discuss the observation.

4.4. Reading timed voltage signals

The command string to read timed voltage signals from the two channels, CH1 and CH2, takes the form `'m1HHHHHHIJJJKKKLMMMNNN\r'`. The command activates both channels simultaneously.

- `'m1'`, instructs the controller unit to read timed voltage signals from CH1 and CH2.
- `'HHHHHH'` represents the sampling frequency given in Hz. For our current implementation of the controller unit, the maximum sampling rate that can reliably read both channels is 210 kHz.
- `'IJJJKKK'` and `'LMMMNNN'` are the coupling mode (`'I'` and `'L'`), gain (`'JJJ'` and `'MMM'`), and DC offset (`'KKK'` and `'NNN'`) parameters for CH1 and CH2, respectively. These values are crucial for configuring the ADC to accurately measure the signal within the expected voltage range. Calibration is necessary to determine the values of these parameters for different voltage ranges. In particular, the DC bias values can vary between units and need to be adjusted individually. This adjustment will be addressed in due course.

For now, we will use the gain and DC offset values shown below to read the sinusoidal signal which we generated earlier at W1 with a frequency of 2 kHz, amplitude of 0.5 V, and 0 V DC offset. Connect CH1 to the resistor where the signal from W1 is applied and execute the following lines in Jupyter Notebook.

```
# waveform generation
device.write(bytes(cmd_gensin, 'utf-8')) # use the command string created earlier
```

```
fs = 200000         # sampling frequency
c1 = 0              # CH1 DC coupling mode; 0 for signals within around ±1 V
gain1 = 138         # CH1 gain; set the range of voltage that can be read
dco1 = 130          # CH1 offset; set the zero level
c2 = 0              # CH2 DC coupling mode
```

```
gain2 = 138         # CH2 gain
dco2 = 130          # CH2 offset
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'
```

Once the command string is registered, the controller unit will immediately output the sampled data. It generates two time-series data sets, each consisting of 1,000 entries, from two channels. These sets can be imported into a single byte array of size 4000, i.e., 2 bytes per unsigned integer × 1,000 unsigned integers per channel × 2 channels. Use the `readinto()` function to read the data into the byte array.

```
bytedata = bytearray(4000)     # create a byte array of size 4000
device.reset_input_buffer()  # clear data in input buffer
device.write(bytes(cmd_readosc, 'utf-8'))
device.readline()            # skip line containing the command string
device.readinto(bytedata)
```

Convert the byte array into a 2-dimensional NumPy array to facilitate numerical processing of the data.

```
import numpy as np

data = np.frombuffer(bytedata, dtype='uint16').reshape((2, 1000))
```

Now, the two rows in the NumPy array, `data[0, :]` and `data[1, :]`, contain the readings from CH1 and CH2, respectively.

4.5. Visualizing timed voltage signals

We need a NumPy array representing the sampled time. The length of the time array should match the length of the sampled data, which is 1,000. We can create the time array with a uniform spacing between consecutive sampled time points using the following line.

```
t = np.arange(1000)/fs
```

We visualize the data using the Plotly library. Create a plot of `t` versus `data[0, :]` shown in Fig. 2.4.

```
import plotly

fig = plotly.graph_objs.Figure()
fig.add_trace(plotly.graph_objs.Scatter(x=t*1e3, y=data[0, :]))
fig.update_layout(xaxis_title='Time (ms)', yaxis_title='Voltage (a.u.)')
```

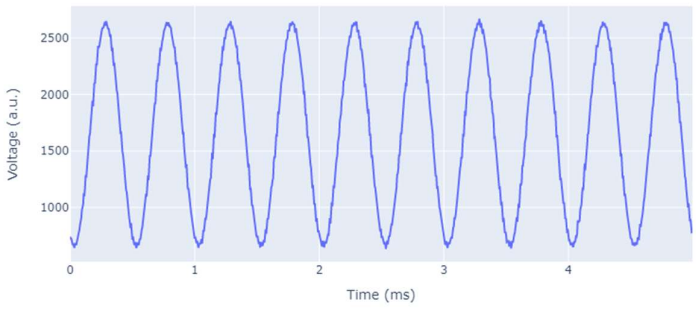


Fig. 2.4. `t` versus `data[0, :]` showing timed voltage readings on CH1.

While the values in `t` represent the real sampled time, the values in `data[0, :]` are arbitrary. They can be expressed in units of volts only after calibration.

👉 Pick up the voltage signal across the same resistor using both CH1 and CH2. Plot the data from both channels together and compare if they are identical. Discuss the observation.

Close the connection by issuing the `close()` command at the end of the laboratory session.

5. Open-Ended Questions

Given that the maximum sampling rate of the ADC that can reliably read both channels is 210 kHZ, what is the highest signal frequency it can capture? What happens if the ADC tries to pick up a signal that is higher than that?

What is the time span of a time series data that has 1,000 data points sampled at 200 kHz sampling frequency?

6. Notebook

Remember to save your notebook with all generated outputs and relevant discussions.

In your notebook for the session, consider including the following:

- Key points observed during examples and exercises
- Discussions on interesting trials and their outcomes

Print your notebook to a PDF file by selecting Print from the browser menu and choosing Save as PDF. Upload the PDF file to NTULearn before leaving the laboratory, regardless of whether you have finished all the tasks or not. It is important to upload everything you have for the day, and you are not permitted to change them once uploaded. Any unfinished work can be included in the notebook for the next laboratory session.

Lab 3: Waveform Modulation and Spectral Analysis

1. Objective

To investigate the spectral properties of waveforms and understand the impact of amplitude modulations.

2. Introduction

A periodic waveform can be expressed as a sum of sinusoids using Fourier series, which takes the form:

$$y(t) = A_0 + \sum_{n=1}^{\infty} A_n \sin(2\pi n f_0 t + \phi_n)\,. \tag{3.1}$$

Equation (3.1) states that a signal $y(t)$ can be decomposed into harmonics of its fundamental frequency, nf_0, with different amplitude, A_n, and phase, ϕ_n, in each component. A_0 accounts for the offset. By performing spectral analysis, we can determine the amplitudes and phases of these frequency components that make up the signal.

The Fourier transform is commonly employed to analyze the frequency content. This mathematical operation converts the waveform from the time domain to the frequency domain, revealing its frequency spectrum. In the spectrum, the horizontal axis represents the range of frequencies, while the vertical axis represents the amplitudes of the frequency components. By examining the spectrum, we can identify the dominant frequencies and their relative strengths within the waveform. Figure 3.1 illustrates this concept with a time domain signal and its corresponding spectrum.

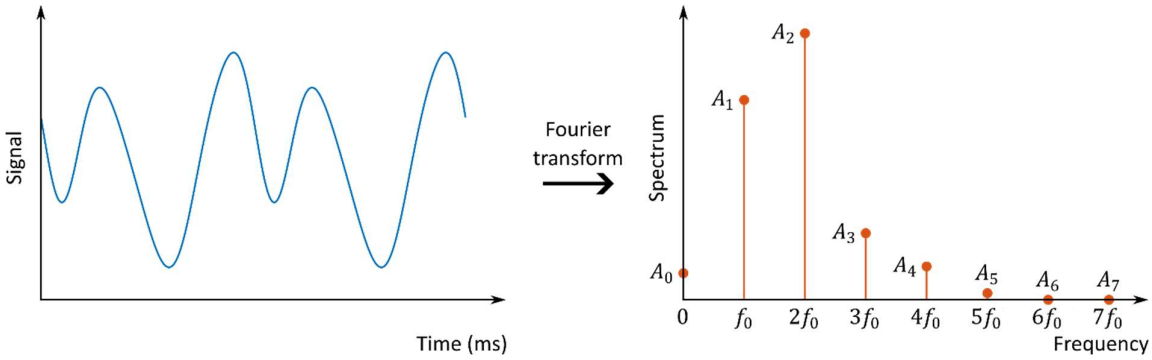


Fig. 3.1. Time and frequency domain representations of a waveform.

Spectral analysis is vital in applications like audio processing, telecommunications, signal processing, and vibration analysis. It provides frequency characteristics that aid in signal identification, noise analysis, and system performance evaluation.

3. Equipment and Components

Equipment

- Computer with ee2073 conda environment
- Controller unit
- USB cable with Type A and Type B Mini connectors

Components

- Resistors
- Jumper wires
- Breadboard

4. Project Tasks

4.1. Fourier transform and spectral analysis

Connect W1 and CH1 to a 20 kΩ resistor. Open a new Juypter Notebook and save it as *lab3_YourName.ipynb*. Execute the following lines in the notebook to open a communication channel, generate a sinusoidal waveform with 1 kHz frequency, 1 V amplitude, and 0 V offset at W1, read the signal via CH1, and plot the waveform.

```
# communication channel setup
import serial
import serial.tools.list_ports
import numpy as np
import plotly.graph_objs as go

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)')
```

Note that the line with the comment `# adjust` includes operations that force the waveform to have the set amplitude (1 V) and offset (0 V). These adjustments will not be needed once the controller unit is calibrated. Running the code will generate a plot of the waveform, as shown in Fig. 3.2.

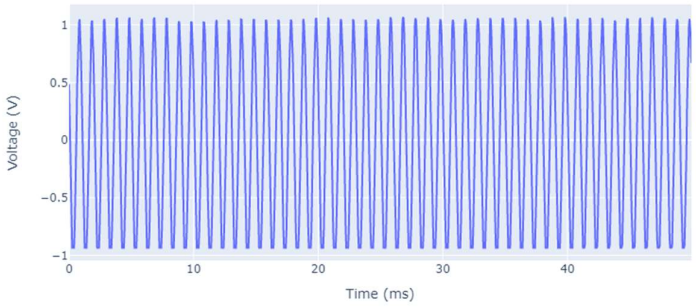


Fig. 3.2. Plot of a sinusoidal voltage signal taken from CH1.

To analyze the spectral components of the signal, we apply the Fourier transform. Since we are working with a discrete dataset, we use the discrete Fourier transform technique. The fast Fourier transform is a widely used algorithm for computing the discrete Fourier transform efficiently. In NumPy, the `rfft()` function is used to compute the fast Fourier transform for a real number array. The following lines of code create the corresponding frequency array, compute the discrete Fourier transform of the signal, and plot the amplitudes of the spectral components.

```
f = np.fft.rfftfreq(len(t), d=1/fs)  # frequency array
spec = np.abs(np.fft.rfft(sig))/len(f) # spectral amplitudes
fig = go.Figure()
fig.add_trace(go.Scatter(x=f*1e-3, y=spec))
fig.update_layout(xaxis_title='Frequency (kHz)', yaxis_title='Amplitude (a.u.)')
```

Here, we create the frequency array `f` based on the sampling frequency and the number of samples using the `rfftfreq()` function. The `rfft()` function returns an array of complex numbers where their magnitude and phase represent the amplitude and phase of the corresponding frequency component. We use the `abs()` function to take their magnitudes. The resulting plot is shown in Fig. 3.3. We observe a prominent peak with an amplitude of 1 at 1 kHz, which is consistent with the characteristics of the signal—a pure sinusoid oscillating at 1 kHz with an amplitude of 1.

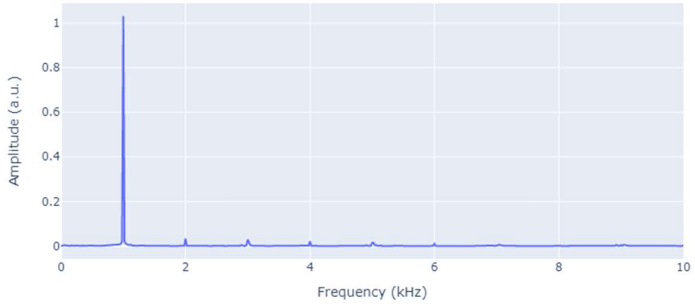


Fig. 3.3. Spectral amplitude versus frequency of the sinusoidal signal.

👉 Analyze the spectra of triangular and sawtooth waveforms using the same approach. Generate the triangular waveform in W1 and the sawtooth waveform in W2 and connect them to two separate

20 kΩ resistors. Use CH1 and CH2 to collect the two signals from the resistors simultaneously. Note that the same `f` array can be used on both samples, since the sampling rate and number of samples collected remain constant. Plot the spectral amplitudes of the two signals, discuss the similarities and differences between the sinusoidal, triangular, and sawtooth waveform spectra. For reference, the Fourier series of triangular and sawtooth waveforms are given by:

$$y(t) = \frac{8}{\pi^2} \sum_{n=1,3,5,\ldots}^{\infty} \frac{(-1)^{(n-1)/2}}{n^2} \sin(2\pi n f_0 t) \text{ and } y(t) = \frac{1}{2} - \frac{1}{\pi} \sum_{n=1}^{\infty} \frac{1}{n} \sin(2\pi n f_0 t), \quad (3.2)$$

respectively.

4.2. Amplitude modulation and its influence on spectrum

We can modulate the amplitude of a sinusoidal wave by multiplying it with an envelope that oscillates at a lower frequency. The mathematical expression for the modulation on a sinusoid of amplitude A and zero offset is given by:

$$y(t) = (1 + m \sin(2\pi f_m t)) \cdot A \sin(2\pi f_0 t), \quad (3.3)$$

where f_m is the modulation frequency and $0 < m < 1$ is the modulation depth. To demonstrate this modulation, let us first generate and read a sinusoidal signal by executing the following lines of code.

```
# waveform generation
device.write(bytes(cmd_gensin, 'utf-8')) # use the command string created earlier
```

```
# waveform detection
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, :])
```

To apply amplitude modulation on `sig` with a modulation frequency of 100 Hz and a modulation depth of 0.5, use the following lines of code. The resulting waveform is then plotted as shown in Fig. 3.4.

```
# 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.)')
```

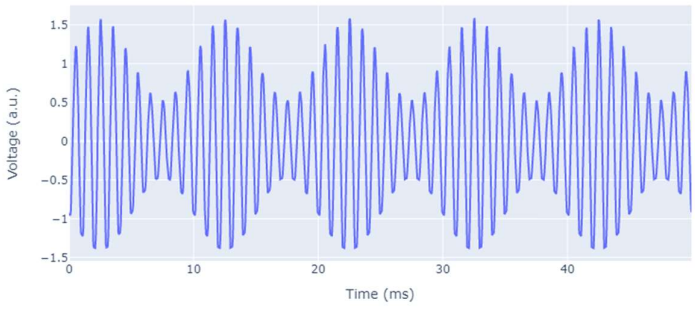


Fig. 3.4. Sinusoidal signal amplitude modulated at a frequency of 100 Hz and a modulation depth of 0.5.

To analyze the effect of modulation on the spectrum, apply the Fourier transform on the modulated signal using the following code. This provides the spectrum as presented in Fig. 3.5.

```
# 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.)')
```

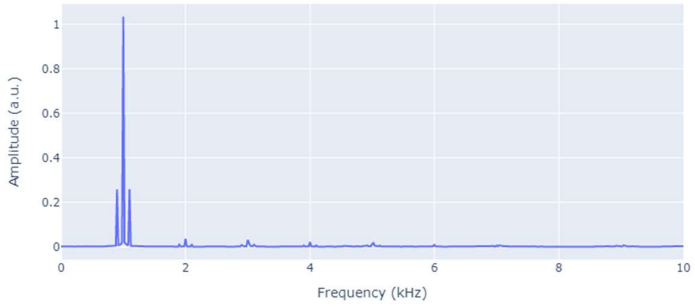


Fig. 3.5. Spectral amplitude of the amplitude modulated sinusoidal waveform.

Observe the formation of sidebands around the main frequency in the spectrum. Zoom in to verify that the sidebands are positioned ±100 Hz away from the main frequency. The appearance of sidebands is a distinct characteristic of amplitude modulation.

👉 Vary the modulation depth and frequency. Plot both the time signal and spectrum for each case and analyze the effects of these changes on the waveform.

👉 Apply amplitude modulations on triangular and sawtooth waveforms. Discuss the similarities and differences by examining the time signal and spectral plots.

Close the connection by issuing the `close()` command at the end of the laboratory session.

5. Open-Ended Questions

Generate a 1 kHz sinusoidal signal with 8 samples per cycle. Measure the signal using CH1 at a sampling frequency of 100 kHz. Repeat the measurement with a 20 kHz sinusoidal signal with the same number of samples per cycle. Comment on the results.

6. Notebook

Remember to save your notebook with all generated outputs and relevant discussions.

In your notebook for the session, consider including the following:

- Key points observed during examples and exercises
- Discussions on interesting trials and their outcomes

Print your notebook to a PDF file by selecting Print from the browser menu and choosing Save as PDF. Upload the PDF file to NTULearn before leaving the laboratory, regardless of whether you have finished all the tasks or not. It is important to upload everything you have for the day, and you are not permitted to change them once uploaded. Any unfinished work can be included in the notebook for the next laboratory session.

Lab 4: Overview of Pre-Built Instruments

1. Objective

To explore the functions of pre-built instruments comprising a dual DC voltage supply, two-channel waveform generator, and two-channel oscilloscope, and perform calibration of the DC offsets of the waveform generator and oscilloscope.

2. Introduction

Our previous approach involved lengthy Python code for tasks such as configuring DC voltage, generating waveforms, and displaying acquired signals. While insightful for learning, this becomes impractical for efficiently controlling an audio amplifier system that requires continuous execution of these processes. To construct the audio amplifier system, we will use pre-built instruments written in Python. These instruments incorporate graphical control elements and additional features to enhance usability.

Calibrating the controller unit is vital for reliable signal generation and accurate voltage reading. Due to the unique variations in the DC offsets of the waveform generator and oscilloscope, individual calibration is necessary for each controller unit.

3. Equipment and Components

Equipment

- Computer with ee2073 conda environment
- Controller unit
- USB cable with Type A and Type B Mini connectors
- Digital multimeter

Components:

- Resistors
- Jumper wires
- Breadboard

4. Project Tasks

4.1. Pre-built instruments

Download *instrument.ipynb* from NTULearn and open it in Jupyter Notebook. Connect the controller unit to the computer. Run all code cells in the notebook and ensure that no error messages are displayed. If any errors are encountered, try reconnecting the unit, restart the kernel, and run again. Once everything is set up correctly, a graphical user interface of the instruments that resembles Fig. 4.1 will be displayed. Altogether, there are three instruments. Save the notebook as *lab4_YourName.ipynb*.

At the top is the dual DC voltage supply. The slider allows us to set the magnitude of the DC voltage at VDCP and VDCN, ranging between ±5.5 and ±13.5 V.

The waveform generator has two channels, each offering options to select the waveform shape (sinusoidal, triangular, or sawtooth), amplitude (0–5 V), frequency (0.1–50 kHz), and offset (up to ±6 V). These settings allow for flexible configuration of the waveform output according to the desired parameters.

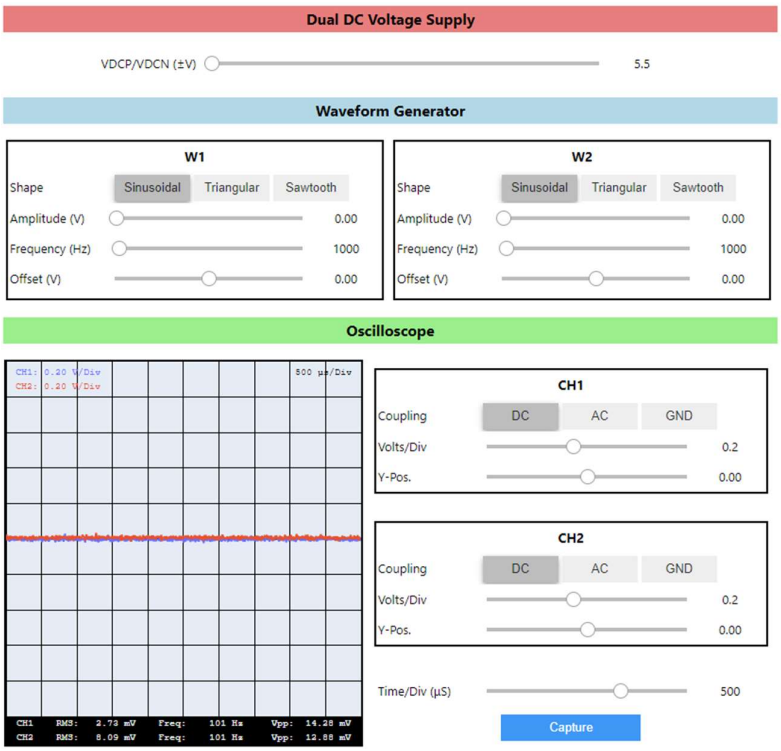


Fig. 4.1. Graphical user interface of the instruments.

The oscilloscope has two channels each with selectable coupling modes (DC, AC, or GND), adjustable voltage per division scale, and vertical shift. The time per division scale is common to both channels. Changes in the oscilloscope settings or pressing the capture button instantly update the trace. Note that DC coupling mode is adequate for the project, and offset calibration will be performed in this mode only.

Verify the instrument functions by connecting them to a 20 kΩ resistor. Use a digital multimeter to check VDCP and VDCN voltages against the dual DC voltage supply. Ensure accurate representation of waveform generator's amplitude and frequency, as well as oscilloscope settings. To capture a trace, click "Download plot as a png", after hovering the mouse pointer over the trace. You can include the trace image in Jupyter notebook by copying the PNG file in File Explorer and paste it in the notebook Markdown cell.

4.2. Waveform generator DC offset calibration

Set both the amplitude and offset in W1 to zero in the waveform generator. Connect W1 to a 20 kΩ resistor and use a digital multimeter to monitor the DC bias voltage across the resistor. Take note of the voltage reading including the sign. Repeat the same procedure to measure the DC bias voltage in W2. Once the bias voltage readings from W1 and W2 are taken, go to the `VScopeBoard` class in the Jupyter Notebook file, and assign these values to `self.w1bias` and `self.w2bias`. Below is an example when the DC bias readings of −182 mV and 204 mV are taken from W1 and W2, respectively.

```
##########################################################################
# Enter your calibration data below
# W1/W2 DC bias
self.w1bias = -0.182 # waveform generator CH1 DC bias reading in V
self.w2bias = 0.204  # waveform generator CH2 DC bias reading in V
##########################################################################
```

Rerun all code cells in the notebook allowing the changes to take effect. Repeat the W1 and W2 DC bias measurements by following the procedure described earlier and verify that the voltage readings

on the digital multimeter are now close to zero for both channels. Also, check that the offset voltage sliders in the waveform generator are functioning as intended.

4.3. Oscilloscope DC offset calibration

Connect both CH1 and CH2 to GND through a breadboard. In the sixth code cell of the Jupyter Notebook file, there is an automated DC offset calibration routine for the oscilloscope that is commented out. Uncomment it to get the code below and activate the calibration routine.

```
# Calibrate CH1 and CH2 DC offset voltages
# Work only in DC coupling mode
# 1. Connect CH1 and CH2 pins to GND
# 2. Execute all cells above
# 3. Uncomment this cell and execute
# 4. Note down the new 'dc' values and change the look-up table in capture_oscscope()
# 5. Comment back this cell once completed

n_calibrate=10
vdiv_list=[0.02,0.05,0.1,0.2,0.5,1,2,5]
ch1_dc=list()
ch2_dc=list()
for vdiv in vdiv_list:
    _,_,_,y1,y2=vscope.capture_oscscope(1000,vdiv,'DC',vdiv,'DC')
    for i in range(n_calibrate-1):
        time.sleep(0.5)
        _,_,_,ch1,ch2=vscope.capture_oscscope(1000,vdiv,'DC',vdiv,'DC')
        y1+=ch1
        y2+=ch2
    y1/=n_calibrate
    y2/=n_calibrate
    ch1_dc.append(vscope.p1[vdiv]['dc']-np.mean(y1)/vscope.p1[vdiv]['amp'])
    ch2_dc.append(vscope.p2[vdiv]['dc']-np.mean(y2)/vscope.p2[vdiv]['amp'])
    print('Completed calibrating '+str(vdiv).rjust(4)+' V/Div')
for i, vdiv in enumerate(vdiv_list):
    print('\'dc\' CH1 '+str(vdiv).rjust(4)+' V/Div: '+str(round(ch1_dc[i],2)))
for i, vdiv in enumerate(vdiv_list):
    print('\'dc\' CH2 '+str(vdiv).rjust(4)+' V/Div: '+str(round(ch2_dc[i],2)))
```

Execute all code cells above this one in sequence, and then run the uncommented cell containing the calibration routine. It will take a few minutes to complete. Once finished, the newly calculated `'dc'` values, which should be incorporated into the look-up tables in the `VScopeBoard` class, will be printed.

```
'dc' CH1 0.02 V/Div: 1.84
'dc' CH1 0.05 V/Div: 1.81
'dc' CH1  0.1 V/Div: 1.79
'dc' CH1  0.2 V/Div: 1.79
'dc' CH1  0.5 V/Div: 1.93
'dc' CH1    1 V/Div: 1.9
'dc' CH1    2 V/Div: 1.82
'dc' CH1    5 V/Div: 1.79
'dc' CH2 0.02 V/Div: 1.84
'dc' CH2 0.05 V/Div: 1.82
'dc' CH2  0.1 V/Div: 1.79
'dc' CH2  0.2 V/Div: 1.79
'dc' CH2  0.5 V/Div: 1.93
'dc' CH2    1 V/Div: 1.9
'dc' CH2    2 V/Div: 1.82
'dc' CH2    5 V/Div: 1.79
```

Make a note of these values. Update the `'dc'` values for each V/div setting accordingly in the look-up tables `self.p1` and `self.p2` in the third code cell of the Jupyter Notebook file.

```
##########################################################################
# CH1/CH2: gain and offset look-up table
# 'dc' calibrate at each vscale
```

```
# 'dc'=-bias/'amp'+'dc'
self.p1={0.02 :{'gain':224,'amp':0.0950,'dc':1.84,'dco':130 if c1!=2 else 171, ...
         0.05 :{'gain':208,'amp':0.1500,'dc':1.81,'dco':130 if c1!=2 else 168, ...
         0.1  :{'gain':160,'amp':0.3900,'dc':1.79,'dco':130 if c1!=2 else 159, ...
         0.2  :{'gain':138,'amp':0.5500,'dc':1.79,'dco':130 if c1!=2 else 157, ...
         0.5  :{'gain':220,'amp':2.1700,'dc':1.93,'dco':130,'aco':0},
         1    :{'gain':212,'amp':2.8500,'dc':1.90,'dco':130,'aco':0},
         2    :{'gain':160,'amp':7.9000,'dc':1.82,'dco':130,'aco':0},
         5    :{'gain':106,'amp':19.100,'dc':1.79,'dco':130,'aco':0}}
self.p2={0.02 :{'gain':224,'amp':0.0950,'dc':1.84,'dco':130 if c2!=2 else 171, ...
         0.05 :{'gain':208,'amp':0.1500,'dc':1.82,'dco':130 if c2!=2 else 167, ...
         0.1  :{'gain':160,'amp':0.3900,'dc':1.79,'dco':130 if c2!=2 else 159, ...
         0.2  :{'gain':138,'amp':0.5500,'dc':1.79,'dco':130 if c2!=2 else 158, ...
         0.5  :{'gain':220,'amp':2.1700,'dc':1.93,'dco':130,'aco':0},
         1    :{'gain':212,'amp':2.8500,'dc':1.90,'dco':130,'aco':0},
         2    :{'gain':160,'amp':7.9000,'dc':1.82,'dco':130,'aco':0},
         5    :{'gain':106,'amp':18.800,'dc':1.79,'dco':130,'aco':0}}
################################################################################
```

Once the look-up table values are updated, comment back the code cell containing the calibration routine. Read the voltage signals generated in the waveform generator using the oscilloscope and confirm that the DC offsets are correctly measured in CH1 and CH2.

Save this file and use it to drive your controller unit in subsequent laboratory sessions.

5. Open-Ended Questions

Explain the meaning of `'amp'` and `'dc'` values in the look-up tables `self.p1` and `self.p2` and describe how the calibration routine works to find the new `'dc'` values for each V/div setting in CH1 and CH2.

6. Notebook

Remember to save your notebook with all generated outputs and relevant discussions.

In your notebook for the session, consider including the following:

- Key points observed during examples and exercises
- Discussions on interesting trials and their outcomes

Print your notebook to a PDF file by selecting Print from the browser menu and choosing Save as PDF. Upload the PDF file to NTULearn before leaving the laboratory, regardless of whether you have finished all the tasks or not. It is important to upload everything you have for the day, and you are not permitted to change them once uploaded. Any unfinished work can be included in the notebook for the next laboratory session.

Lab 5: Voltage-Controlled Amplifier Subsystem

1. Objective

To construct and study the transfer characteristic of the voltage-controlled amplifier (VCA) subsystem.

2. Introduction

The VCA is a key audio signal processing subsystem in the overall audio amplifier system, allowing control over gain/attenuation by applying a control voltage to its core element. The THAT 2180C (THAT Corporation) and OP 275 (Analog Devices) integrated circuits (ICs) serve as the VCA subsystem components. Figure 5.1 illustrates the equivalent circuit diagrams and terminal configurations of THAT 2180C and OP 275. We will construct and analyze a VCA circuit that is designed to operate with a standard dual power supply of ±13.5 V.

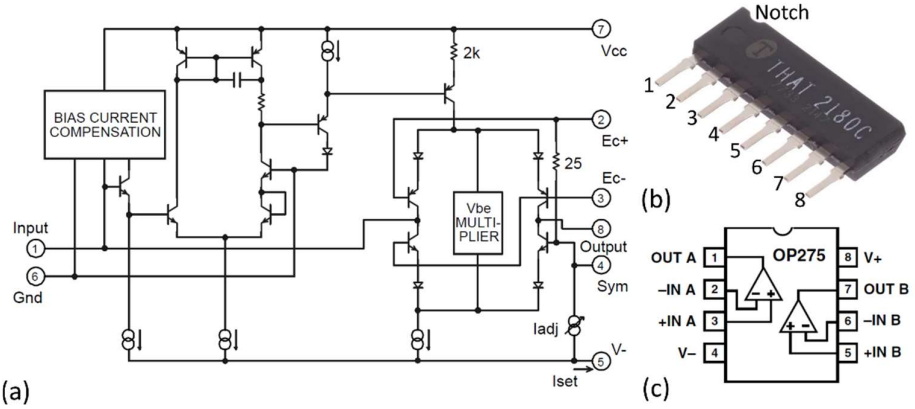


Fig. 5.1. Circuit diagrams and terminal configurations of (a, b) THAT 2180C and (c) OP275.

The specified audio signal gain of the VCA is given by:

$$G_{\text{dB}} = -\frac{V_C}{0.006(1 + 0.0033\Delta T)}, \tag{5.1}$$

where G_{dB} is the gain given in decibels, V_C is the control voltage, and ΔT is the temperature difference between the laboratory and the reference (25 °C). Equation (5.1) indicates that the specified gain sensitivity is −6 mV/dB.

3. Equipment and Components

Equipment

- Computer with ee2073 conda environment
- Controller unit
- USB cable with Type A and Type B Mini connectors

Components

- THAT 2180C voltage-controlled amplifier IC
- OP275 operational amplifier IC
- Resistors
- Capacitors
- Jumper wires
- Breadboard

4. Project Tasks

4.1. VCA gain measurement

Construct the circuit shown in Fig. 5.2 on the breadboard. Use only the left-half of the breadboard for ample space to accommodate other subsystems. To provide ±13.5 V and ground to the circuit, tap-out VDCP, VDCN, and GND from the controller unit onto the power rails of the breadboard. Connect W1 to V_{IN} for the input signal and W2 to V_a for the DC control voltage. Additionally, use the oscilloscope to monitor the input and output voltage signals by connecting CH1 and CH2 to V_{IN} and V_{OUT}.

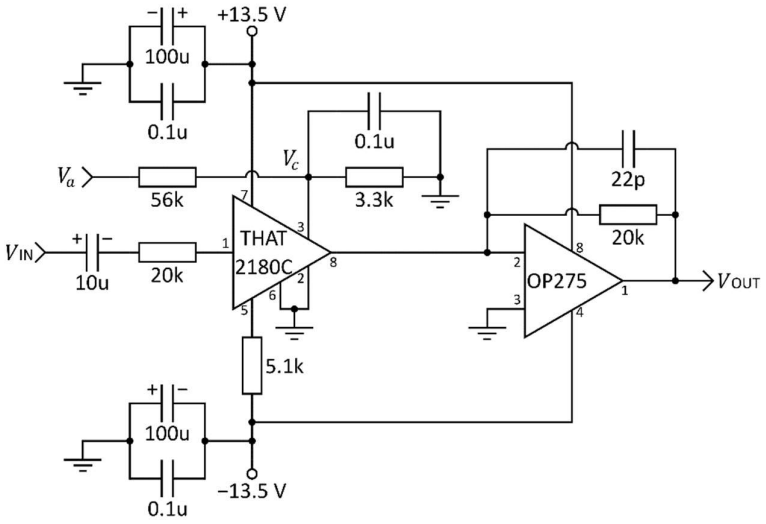


Fig. 5.2. VCA circuit configuration.

Observe the following precautions:

- Verify that all ground connections are correct.
- Ensure proper contact is made between all pins on the breadboard.
- Exercise caution when making wire connections, particularly when biasing with ±13.5 V and ground. Supplying only +13.5 V without the corresponding −13.5 V can lead to overheating and damage to the IC.
- If the IC becomes excessively hot, disconnect the controller unit from the computer. Before reconnecting it, double-check that all wires are correctly and securely connected.
- If a "Power surge on the USB port" warning message appears on the Windows notification area, immediately disconnect the controller unit. This message indicates a possible short circuit in the setup. Thoroughly examine all connections before reconnecting it.

Launch the instruments in Jupyter Notebook, i.e., *lab4_YourName.ipynb* which has W1, W2, CH1, and CH2 offsets calibrated in the last laboratory session. Configure the DC voltage supply to generate ±13.5 V. Set W1 to generate a sinusoidal wave with a frequency of 1 kHz and an amplitude of 1 V. Keep the amplitude of W2 at 0 and use the offset to adjust its DC level. The acceptance range of V_C in THAT 2180C is between −0.15 V and 0.50 V. To set V_C within a broader range of V_a, we utilize a voltage divider. The relation between them is given by:

$$V_C = \frac{3.3\ \text{k}\Omega}{56\ \text{k}\Omega + 3.3\ \text{k}\Omega} V_a. \tag{5.2}$$

Adjust the amplitude of V_{IN} and DC voltage at V_a according to the values specified in Table 5.1. Assign the last digit x of the amplitudes using the digits from your matriculation number, starting from left and repeating as needed. Record the peak-to-peak values of V_{IN} and V_{OUT}, and complete the table.

Amplitude (V)	V_a (V)	V_{IN}-pp (V)	V_{OUT}-pp (V)
3.0	5.4		
2.x	4.5		
2.x	3.6		
2.x	2.7		
1.x	1.8		
1.x	0.9		
1.x	0		
0.9x	−0.9		
0.8x	−1.8		
0.8x	−2.7		

Table 5.1. VCA gain measurement.

During the measurements, capture exemplary oscilloscope traces for inclusion and discussion in your notebook.

4.2. VCA gain characterization

We want to verify that the VCA is functioning as specified in Eq. (5.1). Open a new Juypter Notebook and save it as *lab5_YourName.ipynb*. Create NumPy arrays of V_a and measured peak-to-peak voltages of V_{IN} and V_{OUT}. The measured gain in decibels is given by:

$$G_{\text{dB}} = 20 \log_{10}\left(\frac{V_{\text{OUT}}}{V_{\text{IN}}}\right). \tag{5.3}$$

Plot V_C versus measured gain. Check whether the relationship is linear. Discuss possible reasons if it is not linear. Below is a code snippet of how to achieve this.

```
import numpy as np
import plotly.graph_objs as go

v_a = np.array([5.4, 4.5, 3.6, 2.7, 1.8, 0.9, 0, -0.9, -1.8, -2.7])
v_IN = np.array([your data corresponding to v_a separated by comma])
v_OUT = np.array([Your data corresponding to v_a separated by comma])

v_C = 3.3/(56 + 3.3)*v_a # Eq. (5.2)
gain = 20*np.log10(v_OUT/v_IN) # Eq. (5.3)

fig = go.Figure()
fig.add_trace(go.Scatter(x=v_C, y=gain, mode='markers', name='Measured gain'))
fig.update_layout(xaxis_title='V<sub>C</sub> (V)', yaxis_title='Gain (dB)')
```

Use NumPy's `polyfit()` function to determine the coefficients of the linear fit, i.e., the slope and vertical intercept. Consider excluding data points that deviate from the linear line, such as those affected by gain saturation. Incorporate these coefficients to add the line of best fit. For a first-order

polynomial, `polyfit()` returns two outputs: the slope and vertical intercept. Additionally, include the specified gain given in Eq. (5.1) for comparison. Refer to Fig. 5.3 as an example of the expected plot to produce.

```
coef = np.polyfit(v_C, gain, deg=1) # deg=1 for order 1 polynomial (linear)
fit = coef[0]*v_C + coef[1]

lab_temp = 25 # Laboratory temperature
gain_spec = -v_C/(0.006*(1 + 0.0033*(lab_temp - 25))) # Eq. (5.1)

fig.add_trace(go.Scatter(x=v_C, y=fit, mode='lines', line=dict(dash='dash'),
  name='Best fit'))
fig.add_trace(go.Scatter(x=v_C, y=gain_spec, mode='lines', line=dict(dash='dot'),
  name='Specified gain'))
```

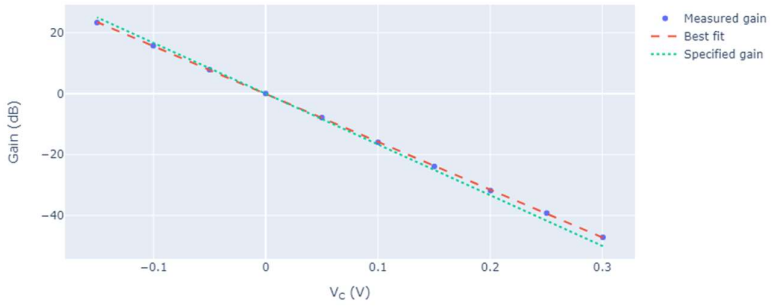


Fig. 5.3. Gain versus V_C.

Compare and discuss the measured gain sensitivity value with the specified one of −6 mV/dB. Note that the measured gain sensitivity corresponds to the inverse of the slope in the linear fit.

👉 Apply small input signal amplitudes. What happens to the input? What happens to the output signal when the amplification factor is high?

👉 Apply high signal amplitudes and high VCA gains simultaneously. Discuss the impact on the output signal and provide an explanation for the observation.

5. Open-Ended Questions

Discuss the function of the VCA in the overall automated volume-controlled audio amplifier system.

6. Notebook

Remember to save your notebook with all generated outputs and relevant discussions.

In your notebook for the session, consider including the following:

- Key points observe during the experiment
- Discussion of interesting trends and their implications
- Captured images of waveforms
- Graphs of measurements and theoretical calculations

Print your notebook to a PDF file by selecting Print from the browser menu and choosing Save as PDF. Upload the PDF file to NTULearn before leaving the laboratory, regardless of whether you have finished all the tasks or not. It is important to upload everything you have for the day, and you are not permitted to change them once uploaded. Any unfinished work can be included in the notebook for the next laboratory session.

Lab 6: Power Amplifier Subsystem

1. Objective

To construct and study the frequency response of the power amplifier (PA) subsystem.

2. Introduction

The PA is the central component responsible for delivering the necessary gain and power amplification to drive the load. A suitable integrated circuit (IC) for the PA subsystem is the LM380N whose equivalent circuit diagram and terminal configuration are shown in Fig. 6.1. The voltage gain of the PA is internally fixed at 34 dB.

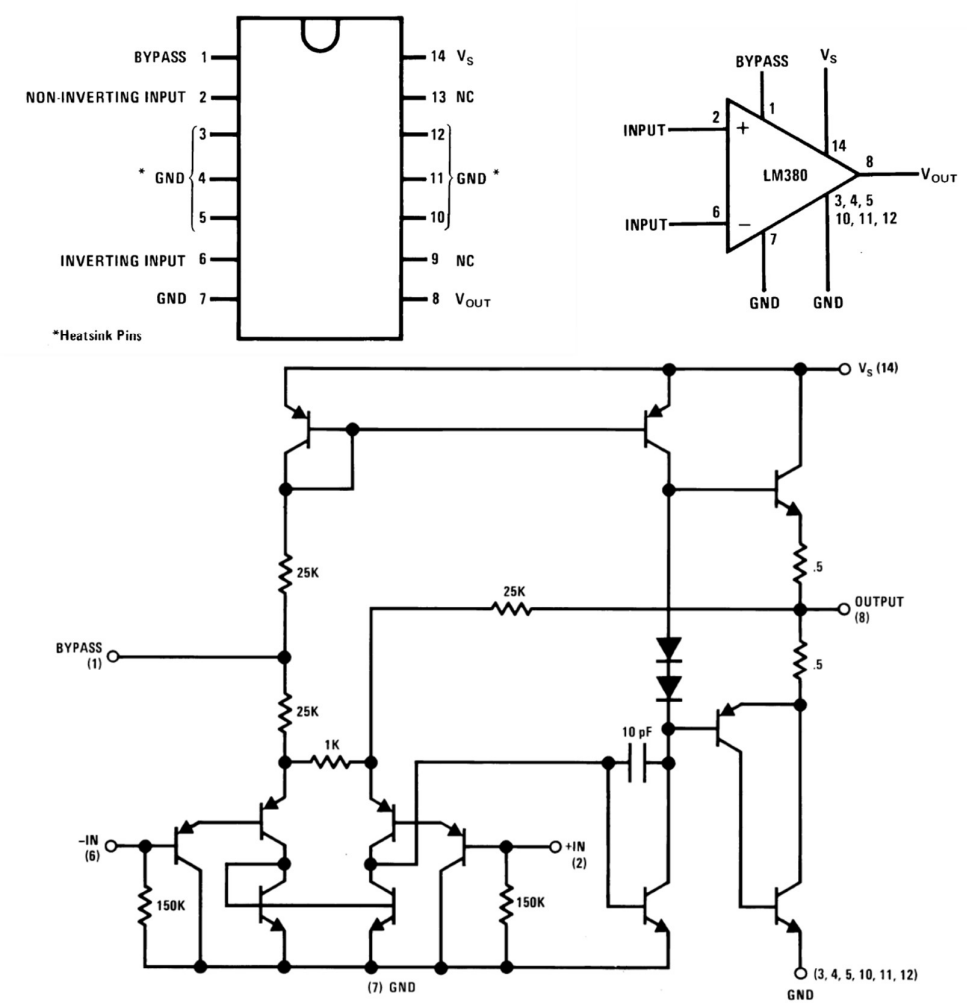


Fig. 6.1. LM380N power amplifier IC.

3. Equipment and Components

Equipment

- Computer with ee2073 conda environment
- Controller unit
- USB cable with Type A and Type B Mini connectors

Components
- LM380N power amplifier IC
- Resistors
- Capacitors
- Jumper wires
- Breadboard

4. Project Tasks

4.1. Frequency response measurement

Construct the circuit illustrated in Fig. 6.2 on the breadboard. Supply +13.5 V and ground to the circuit by tapping out VDCP and GND from the controller unit onto the power rails of the breadboard. Take note to ground pin 7 of LM380N separately from the other grounded pins. This isolates the input stage ground (pin 7) from the output stage grounds (pins 3, 4, 5, 10, 11, and 12) in LM380N. The current in the output stage is very high and any output stage current flowing back to the input stage may cause oscillations due to the high PA gain. Use two power rails on the breadboard, dedicating one (GND1) for grounding pin 7 and the other (GND2) for grounding the rest. Connect W1 to V_{IN} for the input signal and monitor V_{IN} and V_{OUT} with CH1 and CH2.

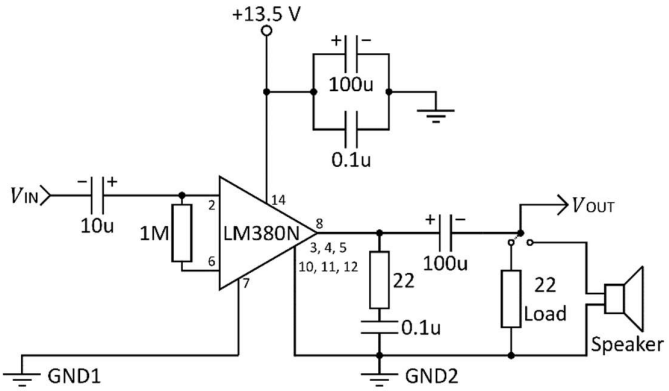


Fig. 6.2. PA circuit configuration.

Launch the instruments in Jupyter Notebook, i.e., your *lab4_YourName.ipynb* file. Set the DC voltage supply to generate +13.5 V and W1 to generate a sinusoidal waveform with an amplitude of 0.1 V. Note that such a small signal amplitude may result in a low signal-to-noise ratio. Adjust the frequency of V_{IN} according to the values provided in Table 6.1. Record the peak-to-peak values of V_{IN} and V_{OUT} to complete the table.

Repeat the measurements with the same set of frequency values, but this time increase the signal amplitude to 0.2 V. Record the peak-to-peak values of V_{IN} and V_{OUT} and complete another table of the same form for 0.2 V amplitude. Observe and discuss the differences in the output waveforms between the two sets of measurements.

Frequency (Hz)	V_{IN}-pp (V)	V_{OUT}-pp (V)
100		
200		
500		

1,000	
2,000	
5,000	
10,000	
20,000	
50,000	

Table 6.1. PA frequency response measurement.

During the measurements, capture exemplary oscilloscope traces for inclusion and discussion in your notebook.

4.2. PA frequency response characterization

Open a new Juypter Notebook and save it as *lab6_YourName.ipynb*. Create NumPy arrays of the measured peak-to-peak values of V_{IN} and V_{OUT} for both 0.1 V and 0.2 V input amplitude sets. Calculate the measured gain in decibels for each data point. The gain is given by:

$$G_{\text{dB}} = 20 \log_{10} \left(\frac{V_{\text{OUT}}}{V_{\text{IN}}} \right). \tag{6.1}$$

Plot the gain versus frequency graph for the two measurement sets in a single graph. Use a logarithmic scale for the frequency axis. This can be achieved by executing `fig.update_xaxes(type='log')` after creating the `fig` object.

Give a brief discussion of the results. Recall that the specified voltage gain of the PA is 34 dB, and comment how the circuit performs against this across the tested frequency range. Relate the results to the expected audio quality at the PA output.

5. Open-Ended Questions

How much current is flowing through the output stage ground (GND2) when the PA is driven with the loudest volume?

6. Notebook

Remember to save your notebook with all generated outputs and relevant discussions.

In your notebook for the session, consider including the following:

- Observation of key points in experiments
- Discussion of interesting trends and their implications
- Captured images of waveforms
- Graphs of measurements and theoretical calculations

Print your notebook to a PDF file by selecting Print from the browser menu and choosing Save as PDF. Upload the PDF file to NTULearn before leaving the laboratory, regardless of whether you have finished all the tasks or not. It is important to upload everything you have for the day, and you are not permitted to change them once uploaded. Any unfinished work can be included in the notebook for the next laboratory session.

Lab 7: Volume Unit Meter Subsystem

1. Objective

To construct and study the function of the volume unit meter (VU Meter).

2. Introduction

In automatic volume control, the VU Meter serves the purpose of measuring the output signal amplitude from the PA subsystem. This measurement is fed back to the controller unit for adjusting the input amplitude to the PA. To construct the VU Meter, the CA3140 operational amplifier integrated circuit (IC) is employed, depicted with its pin diagram in Fig. 7.1.

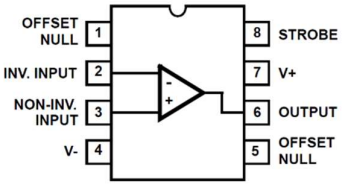


Fig. 7.1. CA3140 pin diagram.

3. Equipment and Components

Equipment

- Computer with ee2073 conda environment
- Controller unit
- USB cable with Type A and Type B Mini connectors

Components

- CA3140 operational amplifier IC
- Resistors
- Capacitors
- Diodes
- Jumper wires
- Breadboard

4. Project Tasks

4.1. VU Meter gain measurement

Construct the circuit illustrated in Fig. 7.2 on the breadboard. Build it on the same side as the PA subsystem, sharing the same power rails for the +13.5 V supply and ground. Supply +13.5 V and ground to the circuit by tapping out VDCP and GND from the controller unit onto the breadboard power rails. Connect W1 to V_{IN} for the input signal and monitor it using CH1. Use CH2 to observe the output at three different points in the circuit, namely V_{OUT1}, V_{OUT2}, and V_{OUT3}.

Launch the instruments in Jupyter Notebook, i.e., your *lab4_YourName.ipynb* file. Set the DC voltage supply to generate +13.5 V and W1 to generate a sinusoidal waveform with a frequency of 1 kHz. Adjust the amplitude of V_{IN} according to the values provided in Table 7.1. Assign the last digit x of the amplitudes using the digits from your matriculation number, starting from left and recycling as necessary. Record the peak-to-peak values of V_{IN}, V_{OUT1} and V_{OUT2}, as well as the root-mean-squared (RMS) values of V_{OUT3}, to complete the table.

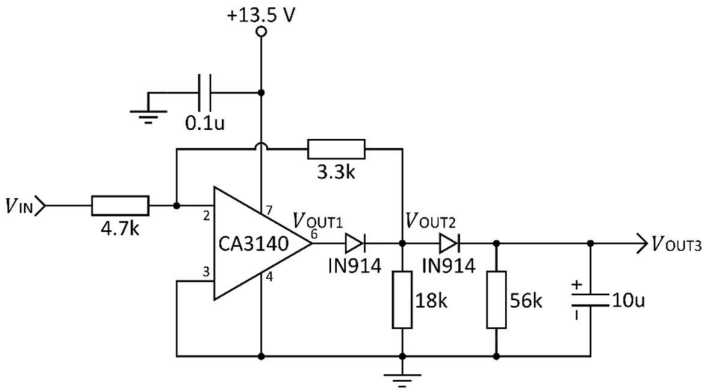


Fig. 7.2. VU Meter circuit.

Amplitude (V)	V_{IN}-pp (V)	V_{OUT1}-pp (V)	V_{OUT2}-pp (V)	V_{OUT3}-RMS (V)
4.x				
3.x				
3.x				
2.x				
2.x				
1.x				
1.x				
0.x				
0.x				

Table 7.1. VU Meter gain measurement.

During the measurements, capture exemplary oscilloscope traces for inclusion and discussion in your notebook.

4.2. VU Meter gain characterization

Open a new Juypter Notebook and save it as *lab7_YourName.ipynb*. Create NumPy arrays of the measured peak-to-peak values of V_{IN}, V_{OUT1}, V_{OUT2}, as well as RMS values of V_{OUT3}. Calculate the gain at V_{OUT2} for each data point. The gain is given by:

$$G = \frac{2V_{OUT2}}{V_{IN}}. \tag{7.1}$$

Plot the graph of measured gain versus V_{IN}-pp. Compare and discuss the measured gain with the theoretical one given by $-R_2/R_1 = -3.3\text{k}\Omega/4.7\text{k}\Omega = -0.70$.

Next, plot the graph of V_{OUT3}-RMS versus V_{IN}-pp, and discuss whether the function of the VU Meter could be met.

5. Open-Ended Questions

What does the two diodes do in the VU Meter? Explain using the waveforms observed at V_{OUT1} and V_{OUT2}.

What happens to V_{OUT3} if there is a DC offset at V_{IN}?

At what volume, i.e., PA output amplitude, does the VU Meter fails to function?

6. Notebook

Remember to save your notebook with all generated outputs and relevant discussions.

In your notebook for the session, consider including the following:

- Observation of key points in experiments
- Discussion of interesting trends and their implications
- Captured images of waveforms
- Graphs of measurements and theoretical calculations

Print your notebook to a PDF file by selecting Print from the browser menu and choosing Save as PDF. Upload the PDF file to NTULearn before leaving the laboratory, regardless of whether you have finished all the tasks or not. It is important to upload everything you have for the day, and you are not permitted to change them once uploaded. Any unfinished work can be included in the notebook for the next laboratory session.

Lab 8: Integration of Audio Amplifier System

1. Objective

To integrate the subsystems constructed so far and test a complete audio amplifier system.

2. Introduction

The complete audio amplifier system is built by connecting the voltage-controlled amplifier (VCA), power amplifier (PA), and volume unit meter (VU Meter) subsystems as shown in Fig. 8.1. The combined system is a manual volume-controlled system where an operator needs to adjust V_a to control the loudness based on the VU Meter reading. As there is no feedback path, it can be considered as an open-loop system.

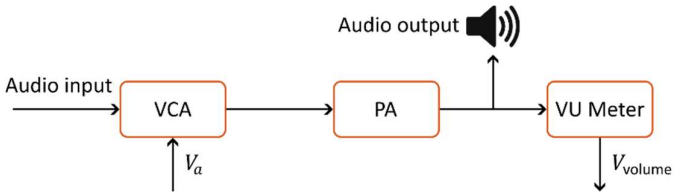


Fig. 8.1. Block diagram of the manual volume-controlled audio amplifier system.

3. Equipment and Components

Equipment

- Computer with ee2073 conda environment
- Controller unit
- USB cable with Type A and Type B Mini connectors

Components

- VCA subsystem
- PA subsystem
- VU Meter subsystem
- Speaker
- Audio cable
- Jumper wires
- Inductor

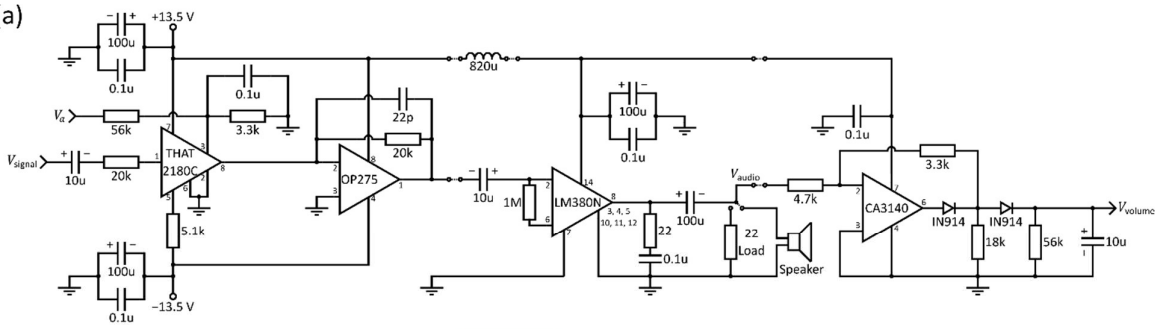
4. Project Tasks

4.1. Integration and testing

Integrate the subsystems as shown in Fig. 8.2. Tap-out VDCP, VDCN, and GND from the controller unit onto the power rails of the breadboard. Use separate power rails for the +13.5 V DC voltage supply to the VCA and PA, with a 820 µH inductor placed between them. Remember to isolate pin 7 of LM380N from the other grounded pins in the same chip. Connect W1 to V_{signal} for the input signal and W2 to V_a for VCA control. Use CH1 and CH2 to monitor V_{signal} and V_{audio} with the oscilloscope.

Launch the instruments in Jupyter Notebook, i.e., your *lab4_YourName.ipynb* file. Set the DC voltage supply to generate ±13.5 V. Configure W1 to generate a sinusoidal waveform with a frequency of 300 Hz. Set the amplitude of W2 to zero and adjust the control voltage using the offset. When the input amplitude is 0.5 V and V_a is +2 V, we can expect the signal at V_{audio} to resemble the waveform shown in Fig. 8.3.

(a)



(b)

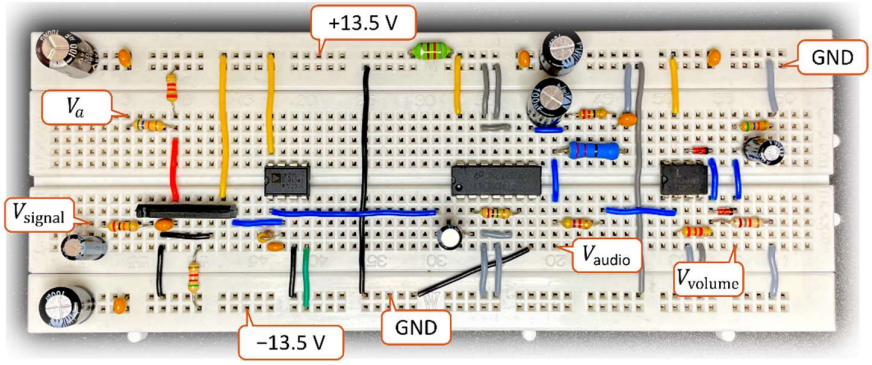


Fig. 8.2. (a) Audio amplifier system circuit. (b) Image of the circuit on breadboard.

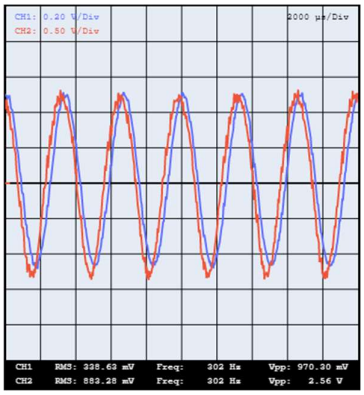


Fig. 8.3. V_{signal} and V_{audio} signals when the input amplitude is set at 0.5 V and V_a is +2V.

Vary the amplitude of V_{signal} and the voltage at V_a according to the values provided in Table 8.1. Assign the last digit x of the amplitudes using the digits from your matriculation number, starting from left and recycling as necessary. Record the peak-to-peak values of V_{signal} and V_{audio}, as well as the root-mean-squared (RMS) values of V_{volume} to complete the table. Pay attention to the V_{audio} waveform shape when adjusting the input amplitude or V_a.

Amplitude (V)	V_a (V)	V_{signal}-pp (V)	V_{audio}-pp (V)	V_{volume}-RMS (V)
4.x	5.4			
3.x	4.5			
2.x	3.6			
1.x	2.7			

$0.8x$	1.8
$0.3x$	0.9
0.09	0
0.04	−0.9

Table 8.1. Audio amplifier system testing.

During the measurements, capture exemplary oscilloscope traces for inclusion and discussion in your notebook. Also, note down the RMS of V_{volume} when the peak-to-peak V_{audio} is at its maximum.

4.2. Audio amplifier characterization

Open a new Juypter Notebook and save it as *lab8_YourName.ipynb*. Create NumPy arrays of V_a and measured peak-to-peak values of V_{signal} and V_{audio}, as well as RMS values of V_{volume}. Obtain V_C from the V_a using:

$$V_C = \frac{3.3\ \text{k}\Omega}{56\ \text{k}\Omega + 3.3\ \text{k}\Omega} V_a. \tag{8.1}$$

Also, calculate the system gain at V_{audio} via:

$$G_{\text{dB}} = 20 \log_{10}\left(\frac{V_{\text{audio}}}{V_{\text{signal}}}\right). \tag{8.2}$$

Plot the graph of the gain versus V_C, and discuss whether the output performance of the audio amplifier system meets the desired objective.

4.3. Playing music on audio amplifier system

Disconnect W1 from V_{signal} and connect an audio cable from the computer's audio output port to V_{signal} of the amplifier system. Replace the 22 Ω load resistor in the PA subsystem with a speaker. Take note of ground loop to ensure the speaker ground connection is localized to the high current ground. Play your favorite tunes from YouTube, making sure that the audio is not muted in the Windows Sound settings. You should now be able to enjoy your music through the speaker. Verify that the volume can be controlled by adjusting the voltage at V_a.

5. Open-Ended Questions

Describe the problems you encountered when integrating the circuit and manually controlling the volume. How did you troubleshoot the problems?

6. Notebook

Remember to save your notebook with all generated outputs and relevant discussions.

In your notebook for the session, consider including the following:

- Observation of key points in experiments
- Discussion of interesting trends and their implications
- Captured images of waveforms
- Graphs of measurements and theoretical calculations

Print your notebook to a PDF file by selecting Print from the browser menu and choosing Save as PDF. Upload the PDF file to NTULearn before leaving the laboratory, regardless of whether you have

finished all the tasks or not. It is important to upload everything you have for the day, and you are not permitted to change them once uploaded. Any unfinished work can be included in the notebook for the next laboratory session.

Lab 9: Automatic Volume Control for Audio Amplifier System

1. Objective

To implement and test the automatic volume control for audio amplifier system.

2. Introduction

The automatic volume control for the audio amplifier is a closed-loop system as shown in Fig. 9.1. The gain control in volume-controlled amplifier (VCA) acts as the actuator, adjusting the signal amplitude that is fed into the power amplifier (PA). The volume unit meter (VU Meter) serves as the sensor, measuring the output audio signal level. A control algorithm executed by the computer processes the feedback path. A simple step-up-down controller can be implemented in a computer program. Ideally, the automatic volume control should maintain a constant audio output level regardless of the input signal level by increasing or decreasing the VCA gain when the audio signal level is below or above the set point.

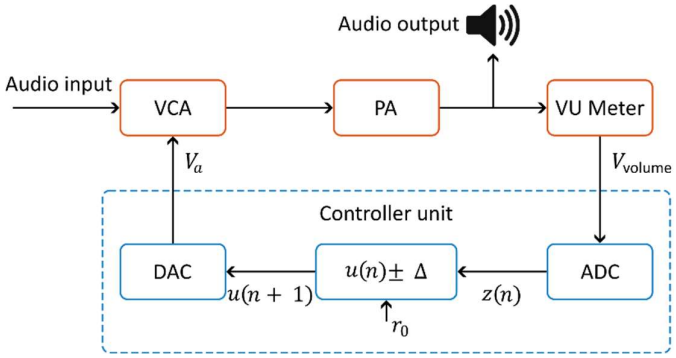


Fig. 9.1. Block diagram of the automatic volume-controlled audio amplifier system.

3. Equipment and Components

Equipment

- Computer with ee2073 conda environment
- Controller unit
- USB cable with Type A and Type B Mini connectors

Components

- Integrated audio amplifier system
- Speaker
- Audio cable
- Jumper wires

4. Project Tasks

4.1. Preparation

Tap-out VDCP, VDCN, and GND from the controller unit onto the power rails of the breadboard that has the integrated audio amplifier system. Connect the speaker to V_{audio} for sound output, W2 to V_a for VCA control, and PC0 to V_{volume} for monitoring the output audio signal level. Connect the computer's audio output port to V_{signal} with an audio cable and ensure that the audio is not muted in the Windows Sound settings.

The automatic volume control program can be developed on top of your *lab4_YourName.ipynb* file. Open this file and save it as *lab9_YourName.ipynb*. Since we will not be using the instruments for this session, comment out the last code cell that displays the user interface elements of the instruments as shown below.

```
# display(ps_title,ps_ui)
# display(wg_title,wg_ui)
# display(osc_title,osc_ui)
```

For the automatic volume control panel, we create two slider elements. One slider lets the user to set the desired level of audio volume, while the other indicates the current audio volume based on the root-mean-squared (RMS) voltage reading at V_{volume}. Additionally, for convenience, we include a toggle button to switch the audio amplifier on and off.

```
set = FloatSlider(description='Set', continuous_update=False, value=10)
vol = FloatSlider(description='Vol', disabled=True)
switch = ToggleButtons(options=['On', 'Off'], value='Off')
```

Note the default minimum and maximum values of the `FloatSlider` object are 0 and 100, respectively. Details on Jupyter Widgets can be found at https://ipywidgets.readthedocs.io/en/latest/.

Next, we need to decide on the required parameters for the automatic volume control. First, we should establish the range of V_a that VCA can take to control its gain. This range should align with the acceptance range of V_C in THAT 2180C, which is between −0.15 V and 0.50 V. To determine the range of V_a, use the relationship:

$$V_C = \frac{3.3\ \text{k}\Omega}{56\ \text{k}\Omega + 3.3\ \text{k}\Omega} V_a. \tag{9.1}$$

Another important parameter to determine is the upper limit of V_{volume} at maximum loudness. Note that the maximum voltage that can be measured on the PC0 and PC1 pins is 3.3 V. This should be close to the RMS of V_{volume} when the audio is at its loudest.

```
va_min = -2.7    # vC must be between -0.15 V (loudest) and 0.5 V (softest)
va_max = 9.0
max_volume = 3.3 # maximum voltage that can be read by PC0 and PC1
```

4.2. Implementing step-up-down controller

The step-up-down controller is represented as:

$$u(n+1) = \begin{cases} u(n) - \Delta, & z(n) < r_0 \\ u(n) + \Delta, & z(n) > r_0 \end{cases}, \tag{9.2}$$

where $u(n)$ is the actuator control voltage calculated in the last iteration and $u(n+1)$ is the updated value in the current iteration. In each iteration, the control voltage will either be decremented or incremented by the step size, Δ, depending on whether the current audio signal level, $z(n)$, is smaller than or larger than the set point value, r_0.

👉 Implement the step-up-down controller in Python based on the algorithm described above. Use the following code skeleton and fill in the missing part responsible for adjusting V_a depending on V_{volume} and the user set volume level.

```
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
    ##########################
    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
```

The `measure_volt()` function in the `VScopeBoard` class provides instantaneous voltage measurements at PC0 and PC1, producing two outputs, i.e., voltage readings at PC0 and PC1, respectively. The `auto` object is an instance of the `RepeatTimer` class. When it is initiated with `auto.start()`, the `control()` function is called every 10 ms. Consequently, the step-up-down controller continue to read V_{volume} and calculate new V_a at regular intervals. For the definition of the `RepeatTimer` class, refer to the fourth code cell of the Jupyter Notebook file.

Test the automatic volume control by playing a sound tone in the computer. Use an online tone generator available in https://onlinetonegenerator.com/ as the input audio signal to begin with. Switch to a music input after checking that the automatic volume control is functioning correctly. Observe and discuss the performance of the automatic volume control with different types of music.

👉 Vary the V_a step size and observe the impact on the system behavior. Discuss the observations and the effectiveness of different step sizes in maintaining a consistent audio output level.

5. Open-Ended Questions

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

6. Notebook

Remember to save your notebook with all generated outputs and relevant discussions.

In your notebook for the session, consider including the following:

- Observation of key points in experiments
- Discussion of interesting trends and their implications

Print your notebook to a PDF file by selecting Print from the browser menu and choosing Save as PDF. Upload the PDF file to NTULearn before leaving the laboratory, regardless of whether you have finished all the tasks or not. It is important to upload everything you have for the day, and you are not permitted to change them once uploaded. Any unfinished work can be included in the notebook for the next laboratory session.

Appendix A: VScope Ver1.4.r1 Circuit Diagrams and PCB Layouts

1. DC boost circuit from +5 V VCC to nominal ±15 V and to nominal ±5 V

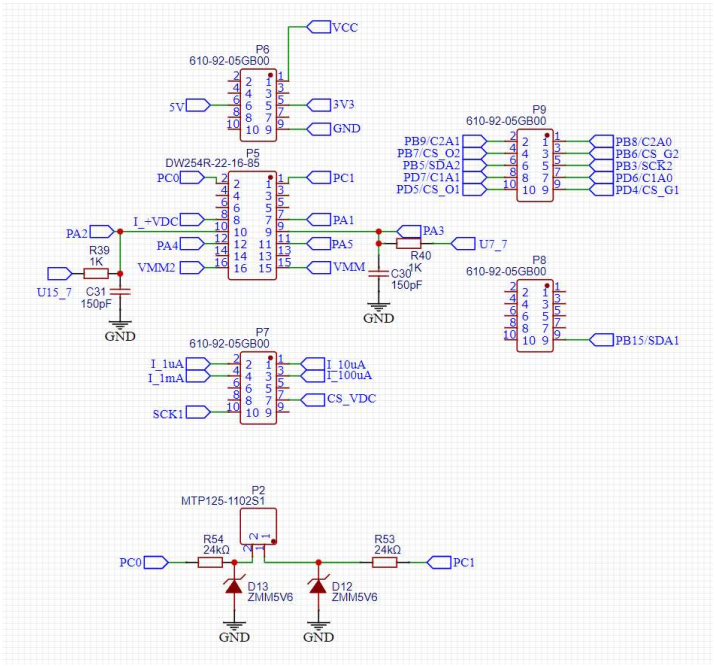


Fig. A.1. Header pin connections and wire labels.

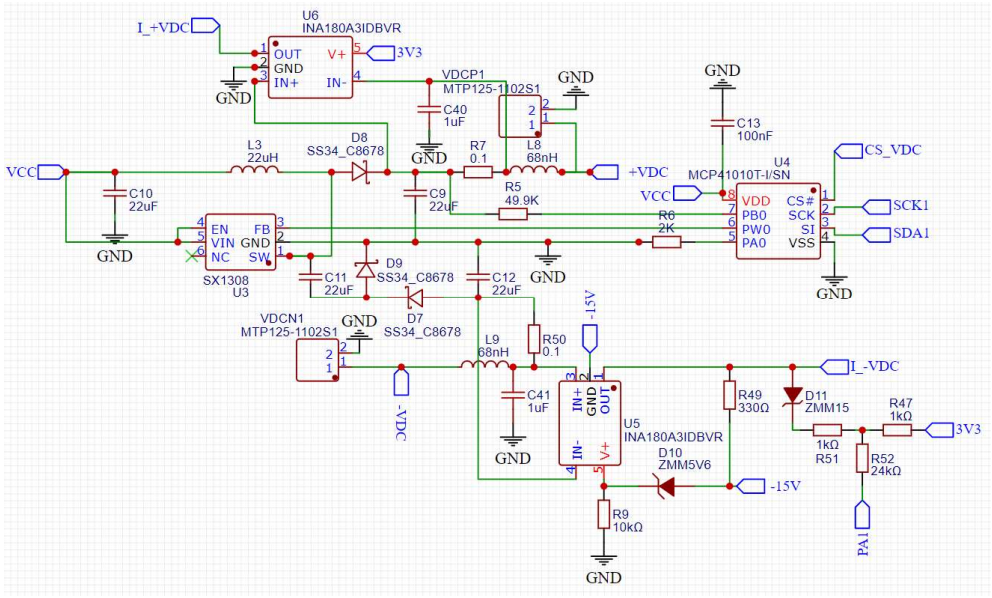


Fig. A.2. Circuit diagram of digitally controlled variable DC voltage supply with output current monitor.

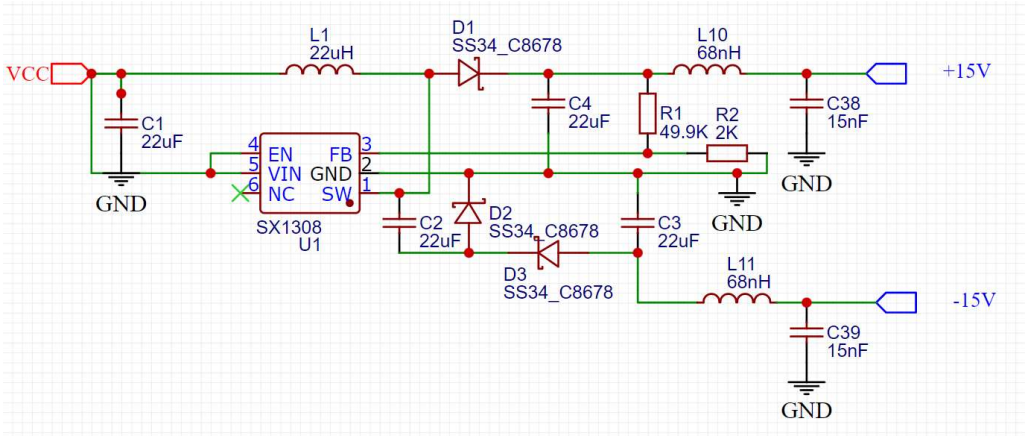


Fig. A.3. Circuit diagram of DC voltage supply with ±15 V output and +5 V VCC input.

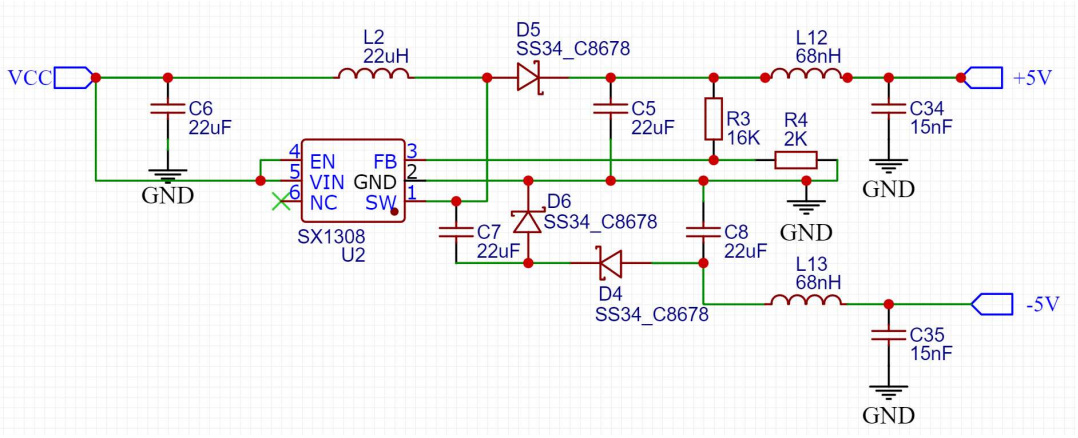


Fig. A.4. Circuit diagram of DC voltage supply with ±5.2 V output and +5 V VCC input.

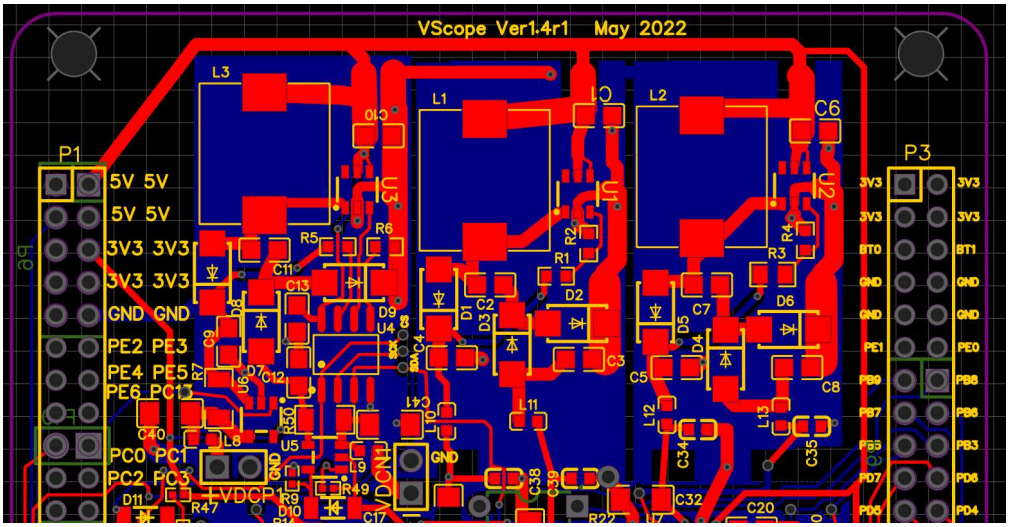


Fig. A.5. PCB layout of DC voltage supply circuits without the components.

2. Waveform generator circuit and oscilloscope CH2 circuit

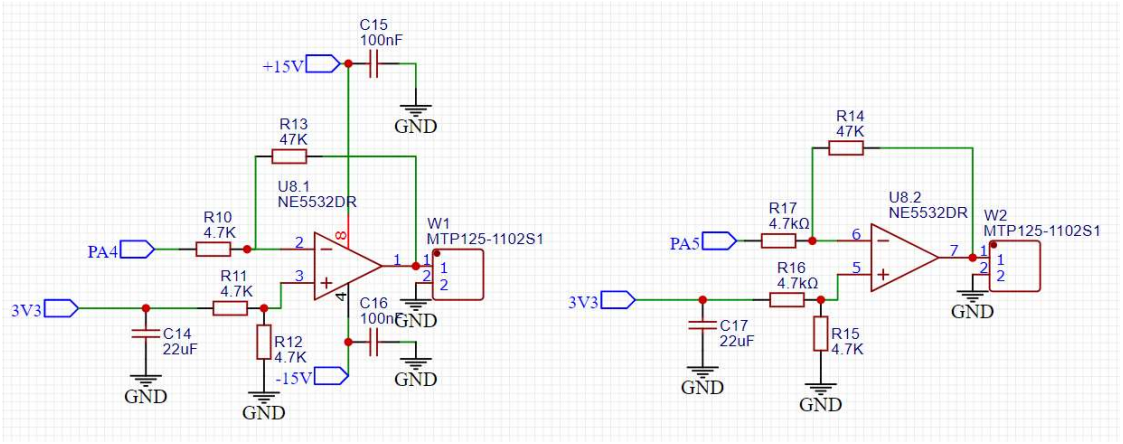


Fig. A.6. Circuit diagrams of waveform generator amplifiers W1 and W2.

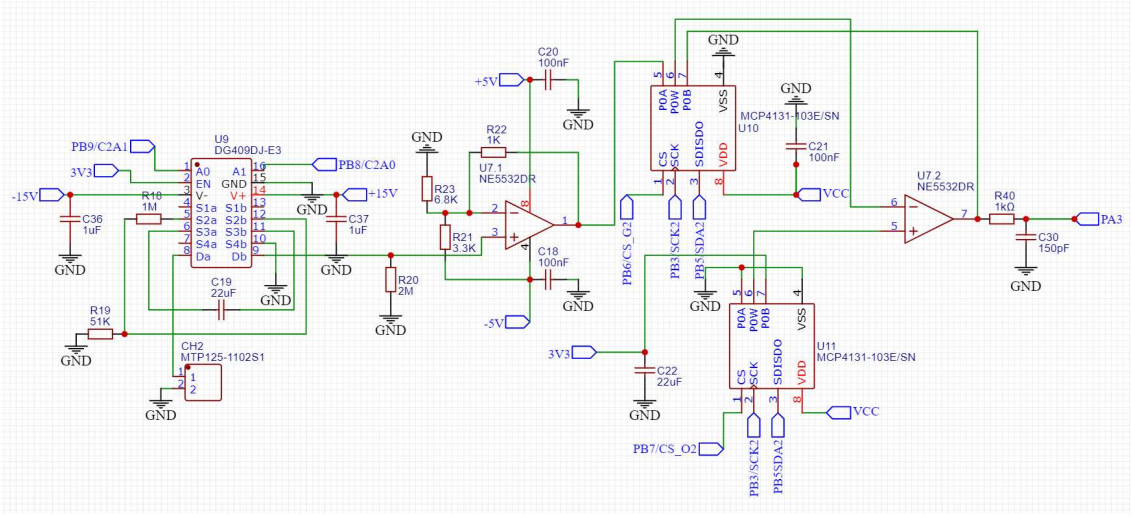


Fig. A.7. Circuit diagram of oscilloscope CH2 input section.

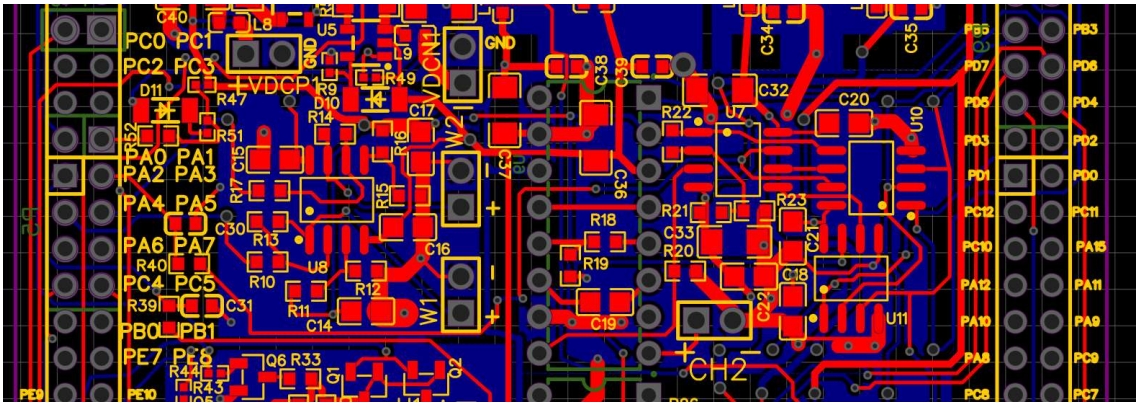


Fig. A.8. PCB layout of waveform generator circuit and oscilloscope CH2 circuit.

3. Multimeter circuit and oscilloscope CH1 circuit

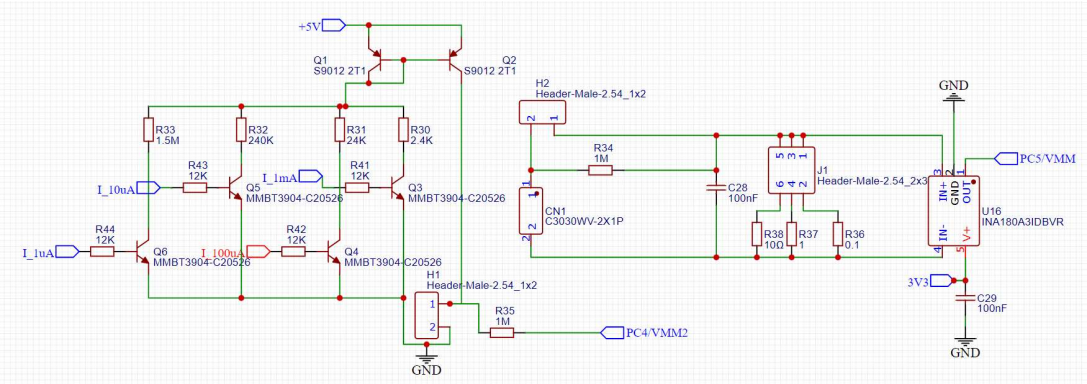


Fig. A.9. Circuit diagram of multimeter.

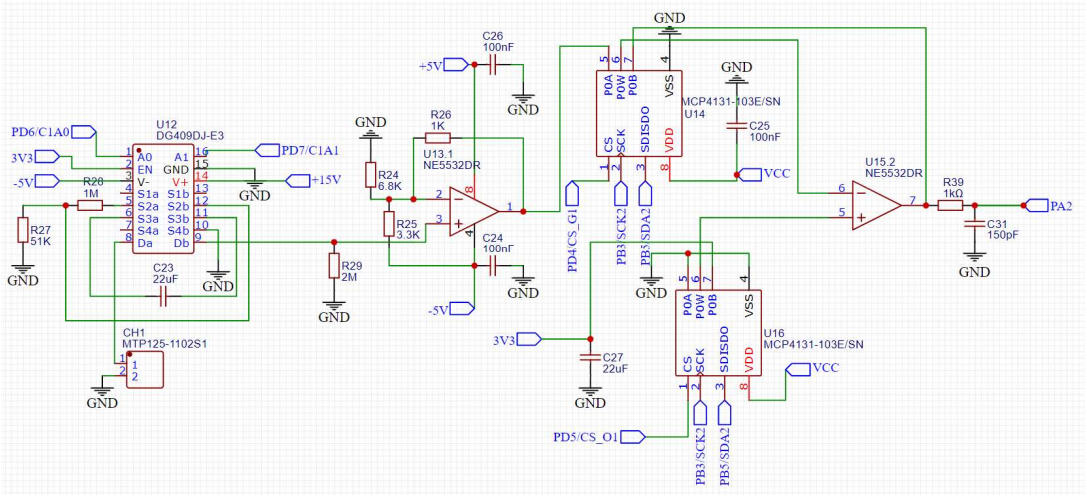


Fig. A.10. Circuit diagram of oscilloscope CH1 input section.

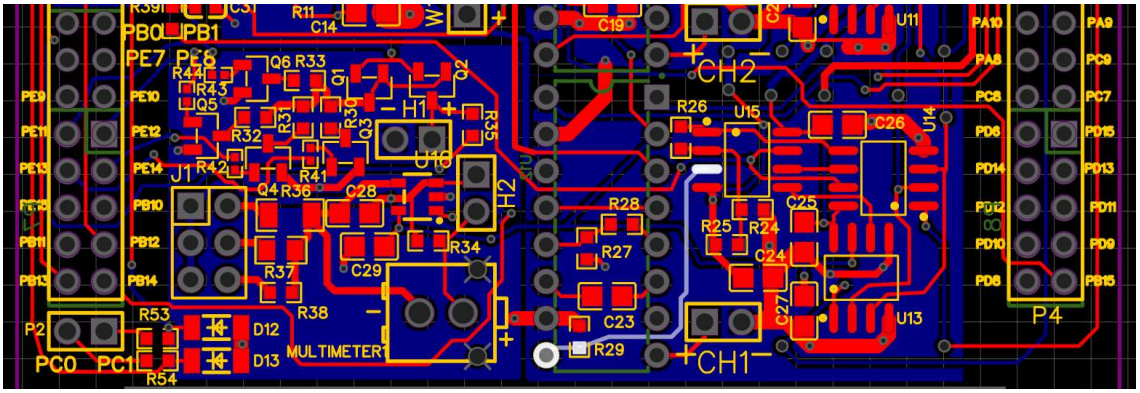


Fig. A.11. PCB layout of multimeter circuit and oscilloscope CH1 circuit.

Appendix B: *main.py*

```
# Micropython firmware for STM32F4VE with VScope 1.4r1
# Law Choi Look      ecllaw@ntu.edu.sg          07 Mar 2022
# Wonkeun Chang      wonkeun.chang@ntu.edu.sg   22 Jul 2023

from pyb import Pin
from pyb import ADC
from pyb import DAC
from pyb import USB_VCP
from pyb import Timer

from machine import SPI

from array import array

import micropython
import math

BUFFERSIZE=1000

micropython.alloc_emergency_exception_buf(50)
usb=USB_VCP()

# Initialize SPI port for control of digital potentiometer of dual DC voltage supply
spi_dc=SPI(sck=Pin('PB13',Pin.OUT),mosi=Pin('PB15',Pin.OUT),miso=Pin('PB14',Pin.IN))
dz=Pin('PB12',Pin.OUT) #dc_voltage set select

# Initialize SPI ports for control of digital potentiometer of ADC buffer amplifiers and input coupling
spi=SPI(sck=Pin('PB3',Pin.OUT),mosi=Pin('PB5',Pin.OUT),miso=Pin('PB4',Pin.IN))
g1=Pin('PD4',Pin.OUT)    # channel 1 gain select
dc1=Pin('PD5',Pin.OUT)   # channel 1 dc_offset select
g2=Pin('PB6',Pin.OUT)    # channel 2 gain select
dc2=Pin('PB7',Pin.OUT)   # channel 2 dc_offset select
c1va0=Pin('PD6',Pin.OUT) # select V/Div # a1a0='00' select s1=DC, '01' select s2=ATT, '10' select s3=AC', '11' select s4=GND
c1va1=Pin('PD7',Pin.OUT) # select channel 1 # s1=0.5V/Div, s2=1V/Div, s3=2V/Div, s4=5V/Div
c2va0=Pin('PB8',Pin.OUT) # select V/Div
c2va1=Pin('PB9',Pin.OUT) # select channel 2

# Dual DC voltage supply
def dcsupply(volt):
    y=312-1020/volt
    dz.value(0)
    spi_dc.write(b'\x11')
    spi_dc.write(bytes((int(y),)))
    dz.value(1)

# Arbitrary waveform generator
def agen(ch,freq,typ,amp,os,ns):
    if amp==0: # Use write() instead of write_timed() for DC signal
        dac=DAC(ch,bits=12,buffering=True)
        dac.write(int((4095/330)*os))
    else: # Use write_timed() for time-varying signal
        dac=DAC(ch,bits=12)  # DAC output: ch1 at pin PA4 and ch2 at pin PA5
        if typ=='sin':
            buf=array('H',[int((4095/330)*(os+amp*math.sin(2*math.pi*i/ns))) for i in range(ns)])
        elif typ=='cos':
            buf=array('H',[int((4095/330)*(os+amp*math.cos(2*math.pi*i/ns))) for i in range(ns)])
        elif typ=='saw':
            buf=array('H',[int((4095/330)*(os-amp*(1.0-2.0*i/ns))) for i in range(ns)])
        elif typ=='tri':
            k=int(ns/2)
            buf=array('H',[int((4095/330)*(os-amp*(1.0-4.0*i/ns))) for i in range(k)])
            buft=array('H',[int((4095/330)*(os+amp*(1.0-4.0*(-0.5+i/ns)))) for i in range(k,ns)])
            buf=buf+buft
        dac.write_timed(buf,Timer(5+ch,freq=freq*len(buf)),mode=DAC.CIRCULAR)

# Initialize ADC for time based measurements
adc0=ADC('PC0') # pin input = PC0
adc1=ADC('PC1') # pin input = PC1
adc2=ADC('PA2') # pin input = PA2
adc3=ADC('PA3') # pin input = PA3
adc_buf2=array('H',(0 for i in range(BUFFERSIZE)))
adc_buf3=array('H',(0 for i in range(BUFFERSIZE)))

# Calibration parameters for waveform generators
w1_gain=10.8
w1_dc=179.3 # unit in x10mV
w2_gain=10.8
w2_dc=179.5 # unit in X10mV

# Initialize DC voltage
dcsupply(5.5)

# Initialize waveform generators output
freq1=1000
freq2=1000
wtype1='sin'
wtype2='sin'
amp1=0
amp2=0
os1=w1_dc
os2=w2_dc
ns1=64
ns2=64
agen(1,freq1,'sin',amp1,os1,ns1)
```

```
agen(2,freq2,'sin',amp2,os2,ns2)

while True:
    mode=input()
    if (mode[0:2]=='m1'): # oscilloscope mode; CH1 and CH2 voltage time series
        fs=int(mode[2:8])
        c1=int(mode[8:9])
        gain1=int(mode[9:12])
        ofs1=int(mode[12:15])
        c2=int(mode[15:16])
        gain2=int(mode[16:19])
        ofs2=int(mode[19:22])
        c1va0.value(c1&(0x1)) # a1a0= '00' DC ; '01' ATT by 20.6 times ; '10' AC ; '11' GND
        c1va1.value((c1&(0x2))>>1)
        c2va0.value(c2&(0x1))
        c2va1.value((c2&(0x2))>>1)
        g1.value(0) # setting ch1 gain
        spi.write(b'\x11')
        spi.write(bytes((gain1,)))
        g1.value(1) # activate new gain
        dc1.value(0) # setting ch1 dc offset
        spi.write(b'\x11')
        spi.write(bytes((ofs1,)))
        dc1.value(1) # activate new dc offset
        g2.value(0) # setting ch2 gain
        spi.write(b'\x11')
        spi.write(bytes((gain2,)))
        g2.value(1) # activate new gain
        dc2.value(0) # setting ch2 dc offset
        spi.write(b'\x11')
        spi.write(bytes((ofs2,)))
        dc2.value(1) # activate new dc offset
        adc_samp=Timer(9,freq=fs)
        ADC.read_timed_multi((adc2,adc3),(adc_buf2,adc_buf3),adc_samp)
        usb.write(adc_buf2)
        usb.write(adc_buf3)
    if (mode[0:2]=='m2'):
        v0=adc0.read()
        v1=adc1.read()
        for i in range(9):
            v0+=adc0.read()
            v1+=adc1.read()
        v0/=10
        v1/=10
        usb.write(array('H',[int(v0)]))
        usb.write(array('H',[int(v1)]))
    if (mode[0:2]=='s1'):
        wnum1=int(mode[2:4])
        ns1=int(mode[4:7])
        freq1=int(mode[7:14])
        amp1=int(mode[14:18])
        amp1=amp1/w1_gain
        os1=int(mode[18:])
        os1=-os1/10+w1_dc
        if wnum1==0:
            wtype1='sin'
        elif wnum1==1:
           wtype1='cos'
        elif wnum1==10:
             wtype1='tri'
        elif wnum1==11:
             wtype1='saw'
        else:
             wtype1='none'
        agen(1,freq1,wtype1,amp1,os1,ns1) #output at PA4
        agen(2,freq2,wtype2,amp2,os2,ns2) #output at PA5  # reduce the occurrence of W1/W2 going chaotic
    if (mode[0:2]=='s2'):
        wnum2=int(mode[2:4])
        ns2=int(mode[4:7])
        freq2=int(mode[7:14])
        amp2=int(mode[14:18])
        amp2=amp2/w2_gain
        os2=int(mode[18:])
        os2=-os2/10+w2_dc
        if wnum2==0:
            wtype2='sin'
        elif wnum2==1:
            wtype2='cos'
        elif wnum2==10:
            wtype2='tri'
        elif wnum2==11:
            wtype2='saw'
        else:
            wtype2='none'
        agen(2,freq2,wtype2,amp2,os2,ns2) #output at PA5
        agen(1,freq1,wtype1,amp1,os1,ns1) #output at PA4 # reduce the occurrence of W1/W2 going chaotic
    if (mode[0:2]=='dz'):
        vV=int(mode[2:6])
        dcsupply(vV/100)
```

Appendix C: *instrument.ipynb*

```
# For use with VScope v1.4r1
# Law Choi Look     ecclaw@ntu.edu.sg           07 Mar 2022 v1.3
# Wonkeun Chang     wonkeun.chang@ntu.edu.sg    22 Jul 2023 v2.7
```

```
import serial
import serial.tools.list_ports

import threading
import time

import plotly
from ipywidgets import
interactive_output,fixed,Button,ToggleButtons,SelectionSlider,IntSlider,FloatSlider,HTML,HBox,VBox,Label,Layout
from IPython.display import display

import numpy as np

VID=61525
PID=38912
BAUDRATE=115200
BUFFERSIZE=1000
```

```
# VScopeBoard class

class VScopeBoard:

    def __init__(self,vid=VID,pid=PID,baudrate=BAUDRATE):
        ports=serial.tools.list_ports.comports()
        for p in ports:
            if p.vid==vid and p.pid==pid:
                self.device=serial.Serial(p.device,baudrate=baudrate)
        if not hasattr(self,'device'):
            raise Exception('No controller unit detected')
        self.device.close()

        ##########################################################################
        # Enter your calibration data below
        # W1/W2 DC bias
        self.w1bias=0 # W1 DC bias reading in V; set it to 0 before calibrating
        self.w2bias=0 # W2 DC bias reading in V; set it to 0 before calibrating
        ##########################################################################
        # CH1/CH2: gain and offset look-up table
        # 'dc' calibrate at each vscale
        # 'dc'=-bias/'amp'+'dc'
        self.p1={0.02 :{'gain':224,'amp':0.0950,'dc':1.90,'dco':130,'dco_c2':171,'aco_c2':-0.005},
                 0.05 :{'gain':208,'amp':0.1500,'dc':1.86,'dco':130,'dco_c2':168,'aco_c2':-0.000},
                 0.1  :{'gain':160,'amp':0.3900,'dc':1.81,'dco':130,'dco_c2':159,'aco_c2':-0.020},
                 0.2  :{'gain':138,'amp':0.5500,'dc':1.80,'dco':130,'dco_c2':157,'aco_c2':-0.050},
                 0.5  :{'gain':220,'amp':2.1700,'dc':1.99,'dco':130,'dco_c2':130,'aco_c2':-0.000},
                 1    :{'gain':212,'amp':2.8500,'dc':1.94,'dco':130,'dco_c2':130,'aco_c2':-0.000},
                 2    :{'gain':160,'amp':7.9000,'dc':1.83,'dco':130,'dco_c2':130,'aco_c2':-0.000},
                 5    :{'gain':106,'amp':19.100,'dc':1.80,'dco':130,'dco_c2':130,'aco_c2':-0.000}}
        self.p2={0.02 :{'gain':224,'amp':0.0950,'dc':1.91,'dco':130,'dco_c2':171,'aco_c2':-0.025},
                 0.05 :{'gain':208,'amp':0.1500,'dc':1.86,'dco':130,'dco_c2':167,'aco_c2':-0.020},
                 0.1  :{'gain':160,'amp':0.3900,'dc':1.81,'dco':130,'dco_c2':159,'aco_c2':-0.040},
                 0.2  :{'gain':138,'amp':0.5500,'dc':1.80,'dco':130,'dco_c2':158,'aco_c2':-0.070},
                 0.5  :{'gain':220,'amp':2.1700,'dc':1.98,'dco':130,'dco_c2':130,'aco_c2':-0.000},
                 1    :{'gain':212,'amp':2.8500,'dc':1.94,'dco':130,'dco_c2':130,'aco_c2':-0.000},
                 2    :{'gain':160,'amp':7.9000,'dc':1.83,'dco':130,'dco_c2':130,'aco_c2':-0.000},
                 5    :{'gain':106,'amp':18.800,'dc':1.80,'dco':130,'dco_c2':130,'aco_c2':-0.000}}
        ##########################################################################

    # Set DC power supply voltage VDCP and VDCN
    def set_vdc(self,voltage):
        cmd='dz'+str(int(voltage*100)).zfill(4)+'\r'
        self.device.open()
        self.device.reset_output_buffer()
        self.device.write(bytes(cmd,'utf-8'))
        self.device.close()

    # Generate waveforms on W1 and W2
    def generate_wave(self,channel,shape,amp,freq,offset):
        if channel==1:
            offset-=self.w1bias
            cmd='s1'
        else:
            offset-=self.w2bias
            cmd='s2'
        ns=64
        if shape=='Triangular':
            cmd+=str(10).zfill(2)
        elif shape=='Sawtooth':
            cmd+=str(11).zfill(2)
        else:
            cmd+=str(0).zfill(2)
        cmd+=str(ns).zfill(3)+str(freq).zfill(7)+str(int(amp*100)).zfill(4)+str(int(offset*100)).zfill(4)+'\r'
        self.device.open()
        self.device.reset_output_buffer()
        self.device.write(bytes(cmd,'utf-8'))
        self.device.close()

    # Capture oscilloscope traces on CH1 and CH2
    def capture_oscscope(self,tbase,vscale1,coupling1,vscale2,coupling2):
```

```
        if tbase>1100:
            fs=50000
        elif tbase>510:
            fs=100000
        else:
            fs=200000 # maximum sampling rate=210 kHz
        if coupling1=='DC':
            c1=0 if vscale1<0.4 else 1
        else:
            c1=2 if coupling1=='AC' else 3
        if coupling2=='DC':
            c2=0 if vscale2<0.4 else 1
        else:
            c2=2 if coupling2=='AC' else 3
        if c1==2:
            dco1=self.p1[vscale1]['dco_c2']
            aco1=self.p1[vscale1]['aco_c2']
        else:
            dco1=self.p1[vscale1]['dco']
            aco1=0
        if c2==2:
            dco2=self.p2[vscale1]['dco_c2']
            aco2=self.p2[vscale1]['aco_c2']
        else:
            dco2=self.p2[vscale1]['dco']
            aco2=0
        cmd='m1'+str(fs).zfill(6)+str(c1)+str(self.p1[vscale1]['gain']).zfill(3)+str(dco1).zfill(3)+str(c2)+\
            str(self.p2[vscale2]['gain']).zfill(3)+str(dco2).zfill(3)+'\r'
        bytedata=bytearray(BUFFERSIZE*4)
        self.device.open()
        self.device.reset_output_buffer()
        self.device.reset_input_buffer()
        self.device.write(bytes(cmd,'utf-8'))
        self.device.readline()
        self.device.readinto(bytedata)
        self.device.close()
        data=np.frombuffer(bytedata,dtype='uint16').reshape((2,BUFFERSIZE))
        raw1=aco1+self.p1[vscale1]['amp']*(self.p1[vscale1]['dc']-1.5*data[0,:]/1700)
        raw2=aco2+self.p2[vscale2]['amp']*(self.p2[vscale2]['dc']-1.5*data[1,:]/1700)
        # Whittaer-Shannon interpolation for signal reconstruction on a finer grid within the tbase*10 range
        n=2**9
        t=np.arange(n)*10*tbase*1e-6/n
        ch1=np.sum(np.multiply(raw1,np.transpose(np.sinc(t*fs-np.reshape(np.arange(BUFFERSIZE),(BUFFERSIZE,1)))),axis=1)
        ch2=np.sum(np.multiply(raw2,np.transpose(np.sinc(t*fs-np.reshape(np.arange(BUFFERSIZE),(BUFFERSIZE,1)))),axis=1)
        return t,ch1,ch2,raw1,raw2

    # Measure DC voltage on PC0 and PC1
    def measure_volt(self):
        cmd='m2\r'
        bytedata=bytearray(4)
        self.device.open()
        self.device.reset_output_buffer()
        self.device.reset_input_buffer()
        self.device.write(bytes(cmd,'utf-8'))
        self.device.readline()
        self.device.readinto(bytedata)
        self.device.close()
        data=np.frombuffer(bytedata,dtype='uint16')
        v1=3.3*data[0]/4095
        v2=3.3*data[1]/4095
        return v1,v2
```

```
# RepeatTimer class

class RepeatTimer:

    def __init__(self,interval,function,*args,**kwargs):
        self.interval=interval
        self.function=function
        self.args=args
        self.kwargs=kwargs
        self.is_running=False

    def _run(self):
        self.is_running=False
        try:
            self.function(*self.args,**self.kwargs)
        except:
            pass
        self.start()

    def start(self):
        if not self.is_running:
            self._timer=threading.Timer(self.interval,self._run)
            self._timer.start()
            self.is_running=True

    def stop(self):
        try:
            self._timer.cancel()
            self.is_running=False
        except:
            pass
```

```
vscope=VScopeBoard()
```

```
# # Calibrate CH1 and CH2 DC offset voltages
# # Work only in DC coupling mode
# # 1. Connect CH1 and CH2 pins to GND
```

```
# # 2. Execute all cells above
# # 3. Uncomment this cell and execute
# # 4. Note down the new 'dc' values and incorporate in the look-up table in capture_oscscope()
# # 5. Comment back this cell once completed

# n_calibrate=10
# vdiv_list=[0.02,0.05,0.1,0.2,0.5,1,2,5]
# ch1_dc=list()
# ch2_dc=list()
# for vdiv in vdiv_list:
#     _,_,_,y1,y2=vscope.capture_oscscope(1000,vdiv,'DC',vdiv,'DC')
#     for i in range(n_calibrate-1):
#         time.sleep(0.5)
#         _,_,_,ch1,ch2=vscope.capture_oscscope(1000,vdiv,'DC',vdiv,'DC')
#         y1+=ch1
#         y2+=ch2
#     y1/=n_calibrate
#     y2/=n_calibrate
#     ch1_dc.append(vscope.p1[vdiv]['dc']-np.mean(y1)/vscope.p1[vdiv]['amp'])
#     ch2_dc.append(vscope.p2[vdiv]['dc']-np.mean(y2)/vscope.p2[vdiv]['amp'])
#     print('Completed calibrating '+str(vdiv).rjust(4)+' V/Div')
# for i, vdiv in enumerate(vdiv_list):
#     print('\'dc\' CH1 '+str(vdiv).rjust(4)+' V/Div: '+str(round(ch1_dc[i],2)))
# for i, vdiv in enumerate(vdiv_list):
#     print('\'dc\' CH2 '+str(vdiv).rjust(4)+' V/Div: '+str(round(ch2_dc[i],2)))
```

```
# DC voltage supply interface

vdc=FloatSlider(min=5.5,max=13.5,step=0.1,value=5.5,continuous_update=False,readout_format='.1f',layout=Layout(width='500px'))
vdc_ui=HBox([Label(value='VDCP/VDCN (±V)',layout=Layout(width='105px')),vdc],
       layout=Layout(justify_content='center',margin='5px 5px 5px 5px'))

ps_html='<h1 style=\'text-align:center;font-size:16px;background-color:lightcoral\'>Dual DC Voltage Supply</h1>'
ps_title=HTML(value=ps_html,layout=Layout(width='820px'))
ps_ui=HBox([vdc_ui],layout=Layout(justify_content='space-around',width='820px',height='44px'))

interactive_output(vscope.set_vdc,{'voltage':vdc})
```

```
# Waveform generator interface

shape1=ToggleButtons(options=['Sinusoidal','Triangular','Sawtooth'],value='Sinusoidal',style={'button_width':'80px'})
amp1=FloatSlider(min=0,max=5,step=0.01,value=0,readout_format='.2f',continuous_update=False)
freq1=IntSlider(min=100,max=50000,step=100,value=1000,continuous_update=False)
offset1=FloatSlider(min=-6,max=6,step=0.01,value=0,readout_format='.2f',continuous_update=False)

shape2=ToggleButtons(options=['Sinusoidal','Triangular','Sawtooth'],value='Sinusoidal',style={'button_width':'80px'})
amp2=FloatSlider(min=0,max=5,step=0.01,value=0,readout_format='.2f',continuous_update=False)
freq2=IntSlider(min=100,max=50000,step=100,value=1000,continuous_update=False)
offset2=FloatSlider(min=-6,max=6,step=0.01,value=0,readout_format='.2f',continuous_update=False)

shape1_ui=HBox([Label(value='Shape',layout=Layout(width='105px',display='flex',justify_content='flex-start')),shape1])
amp1_ui=HBox([Label(value='Amplitude (V)',layout=Layout(width='105px',display='flex',justify_content='flex-start')),amp1])
freq1_ui=HBox([Label(value='Frequency (Hz)',layout=Layout(width='105px',display='flex',justify_content='flex-start')),freq1])
offset1_ui=HBox([Label(value='Offset (V)',layout=Layout(width='105px',display='flex',justify_content='flex-start')),offset1])

shape2_ui=HBox([Label(value='Shape',layout=Layout(width='105px',display='flex',justify_content='flex-start')),shape2])
amp2_ui=HBox([Label(value='Amplitude (V)',layout=Layout(width='105px',display='flex',justify_content='flex-start')),amp2])
freq2_ui=HBox([Label(value='Frequency (Hz)',layout=Layout(width='105px',display='flex',justify_content='flex-start')),freq2])
offset2_ui=HBox([Label(value='Offset (V)',layout=Layout(width='105px',display='flex',justify_content='flex-start')),offset2])

wg1_ui=VBox([HTML('<h3 style=\'text-align:center;font-size:14px;margin-top:0px;margin-bottom:0px\'>W1</h3>'),shape1_ui,amp1_ui,
       freq1_ui,offset1_ui],layout=Layout(border='solid 2px',margin='5px 5px 5px 5px'))
wg2_ui=VBox([HTML('<h3 style=\'text-align:center;font-size:14px;margin-top:0px;margin-bottom:0px\'>W2</h3>'),shape2_ui,amp2_ui,
       freq2_ui,offset2_ui],layout=Layout(border='solid 2px',margin='5px 5px 5px 5px'))

wg_ui=HBox([wg1_ui,wg2_ui],layout=Layout(justify_content='space-around',width='820px',height='180px'))

wg_html='<h1 style=\'text-align:center;font-size:16px;background-color:lightblue\'>Waveform Generator</h1>'
wg_title=HTML(value=wg_html,layout=Layout(width='820px'))

interactive_output(vscope.generate_wave,{'channel':fixed(1),'shape':shape1,'amp':amp1,'freq':freq1,'offset':offset1})
interactive_output(vscope.generate_wave,{'channel':fixed(2),'shape':shape2,'amp':amp2,'freq':freq2,'offset':offset2})
```

```
# Oscilloscope interface

fig=plotly.graph_objs.FigureWidget()
fig.update_layout(width=380,height=410,margin=dict(l=2,r=2,t=2,b=32),paper_bgcolor='black',showlegend=False)
fig.update_layout(xaxis=dict(showticklabels=False,showgrid=True,gridwidth=1,gridcolor='black',zeroline=True,
  zerolinecolor='black'))
fig.update_layout(yaxis1=dict(tickmode='array',tickvals=[0],ticks='inside',tickwidth=2,
  tickcolor=plotly.colors.qualitative.Plotly[0],showticklabels=False,showgrid=False,zeroline=False))
fig.update_layout(yaxis2=dict(tickmode='array',tickvals=[0],ticks='inside',tickwidth=2,
  tickcolor=plotly.colors.qualitative.Plotly[1],showticklabels=False,showgrid=False,zeroline=False,overlaying='y1'))
fig.update_layout(yaxis3=dict(range=[-5,5],dtick=1,showticklabels=False,showgrid=True,gridwidth=1,gridcolor='black',
  zeroline=True,zerolinewidth=2,zerolinecolor='black',overlaying='y1'))
fig.add_trace(plotly.graph_objs.Scatter(x=[],y=[],name='CH1',yaxis='y1'))
fig.add_trace(plotly.graph_objs.Scatter(x=[],y=[],name='CH2',yaxis='y2'))
fig.add_trace(plotly.graph_objs.Scatter(x=[],y=[],yaxis='y3'))
fig.add_annotation(dict(font=dict(family='Courier New, monospace',color='white',size=10),x=0.5,y=-0.04,showarrow=False,text='',
  textangle=0,xanchor='center',xref='paper',yref='paper'))
fig.add_annotation(dict(font=dict(family='Courier New, monospace',color='white',size=10),x=0.5,y=-0.08,showarrow=False,text='',
  textangle=0,xanchor='center',xref='paper',yref='paper'))
fig.add_annotation(dict(font=dict(family='Courier New, monospace',color=plotly.colors.qualitative.Plotly[0],size=10),x=0.02,
  y=0.99,showarrow=False,text='',textangle=0,xanchor='left',xref='paper',yref='paper'))
fig.add_annotation(dict(font=dict(family='Courier New, monospace',color=plotly.colors.qualitative.Plotly[1],size=10),x=0.02,
  y=0.95,showarrow=False,text='',textangle=0,xanchor='left',xref='paper',yref='paper'))
fig.add_annotation(dict(font=dict(family='Courier New, monospace',color='black',size=10),x=0.98,y=0.99,showarrow=False,
  text='',textangle=0,xanchor='right',xref='paper',yref='paper'))
```

```
coupling1=ToggleButtons(options=['DC','AC','GND'],value='DC',style={'button_width':'80px'})
vscale1=SelectionSlider(options=[0.02,0.05,0.1,0.2,0.5,1,2,5],value=0.2,continuous_update=False)
vpos1=FloatSlider(min=-5,max=5,step=0.01,value=0,readout_format='.2f',continuous_update=False)

coupling2=ToggleButtons(options=['DC','AC','GND'],value='DC',style={'button_width':'80px'})
vscale2=SelectionSlider(options=[0.02,0.05,0.1,0.2,0.5,1,2,5],value=0.2,continuous_update=False)
vpos2=FloatSlider(min=-5,max=5,step=0.01,value=0,readout_format='.2f',continuous_update=False)

tbase=SelectionSlider(options=[20,50,100,200,500,1000,2000],value=500,continuous_update=False)

capture=Button(description='Capture',button_style='primary')

coupling1_ui=HBox([Label(value='Coupling',layout=Layout(width='105px',display='flex',justify_content='flex-start')),coupling1])
vscale1_ui=HBox([Label(value='Volts/Div',layout=Layout(width='105px',display='flex',justify_content='flex-start')),vscale1])
vpos1_ui=HBox([Label(value='Y-Pos.',layout=Layout(width='105px',display='flex',justify_content='flex-start')),vpos1])

coupling2_ui=HBox([Label(value='Coupling',layout=Layout(width='105px',display='flex',justify_content='flex-start')),coupling2])
vscale2_ui=HBox([Label(value='Volts/Div',layout=Layout(width='105px',display='flex',justify_content='flex-start')),vscale2])
vpos2_ui=HBox([Label(value='Y-Pos.',layout=Layout(width='105px',display='flex',justify_content='flex-start')),vpos2])

tbase_ui=HBox([Label(value='Time/Div (µS)',layout=Layout(width='105px',display='flex',justify_content='flex-start')),tbase],
         layout=Layout(margin='7px 7px 7px 7px'))

osc1=VBox([HTML('<h3 style=\'text-align:center;font-size:14px;margin-top:0px;margin-bottom:0px\'>CH1</h3>'),coupling1_ui,
     vscale1_ui,vpos1_ui],layout=Layout(border='solid 2px',margin='5px 5px 5px 5px'))
osc2=VBox([HTML('<h3 style=\'text-align:center;font-size:14px;margin-top:0px;margin-bottom:0px\'>CH2</h3>'),coupling2_ui,
     vscale2_ui,vpos2_ui],layout=Layout(border='solid 2px',margin='5px 5px 5px 5px'))
osc3=VBox([tbase_ui,capture],layout=Layout(align_items='center'))
osc_ui=HBox([VBox([fig],layout=Layout(justify_content='space-around')),VBox([osc1,osc2,osc3],
       layout=Layout(justify_content='space-around'))],
       layout=Layout(justify_content='space-around',width='820px',height='420px'))

osc_html='<h1 style=\'text-align:center;font-size:16px;background-color:lightgreen\'>Oscilloscope</h1>'
osc_title=HTML(value=osc_html,layout=Layout(width='820px'))

def calchar(y,dt,tbase):
    n_avg=5
    y=np.convolve(y,np.ones(n_avg),mode='valid')/n_avg # moving average
    rms=np.sqrt(np.mean(y**2))
    ptp=np.ptp(y)
    if tbase<510:
        y=np.pad(y,(0,int(0.01*len(y)/(10*tbase*1e-6))-len(y)),'constant') # ensure frequency resolution < 100 Hz
    freq=np.abs(np.fft.fftfreq(len(y))[np.argmax(np.abs(np.fft.fft(y-np.mean(y))))]/dt)
    return rms,freq,ptp

def txtchar(ch,rms,freq,ptp):
    text='CH1    ' if ch==1 else 'CH2    '
    text=text+'RMS: {0:6.2f} mV    '.format(rms*1e3) if rms<1 else text+'RMS: {0:6.2f} V     '.format(rms)
    text=text+'Freq: {0:6.3f} kHz    '.format(freq*1e-3) if freq>1e3 else text+'Freq: {0:6.0f} Hz     '.format(freq)
    text=text+'Vpp: {0:6.2f} mV'.format(ptp*1e3) if ptp<1 else text+'Vpp: {0:6.2f} V '.format(ptp)
    return text

def capture_oscscope(empty=None):
    if coupling1.value=='AC' and vscale1.value>0.4:
        print('Only DC coupling is available for V/Div > 0.5. Selecting DC coupling for CH1.')
        coupling1.value='DC'
    if coupling2.value=='AC' and vscale2.value>0.4:
        print('Only DC coupling is available for V/Div > 0.5. Selecting DC coupling for CH2.')
        coupling2.value='DC'
    t,ch1,ch2,_,_=vscope.capture_oscscope(tbase.value,vscale1.value,coupling1.value,vscale2.value,coupling2.value)
    fig.data[0]['x']=fig.data[1]['x']=t
    fig.data[0]['y']=ch1
    fig.data[1]['y']=ch2
    fig.update_layout(xaxis=dict(range=[0,10*tbase.value*1e-6],dtick=tbase.value*1e-6))
    fig.update_layout(yaxis1=dict(range=[vscale1.value*(-vpos1.value-5),vscale1.value*(-vpos1.value+5)]))
    fig.update_layout(yaxis2=dict(range=[vscale2.value*(-vpos2.value-5),vscale2.value*(-vpos2.value+5)]))
    rms1,freq1,ptp1=calchar(ch1,t[1],tbase.value)
    rms2,freq2,ptp2=calchar(ch2,t[1],tbase.value)
    fig.update_layout(annotations=[dict(text=txtchar(1,rms1,freq1,ptp1)),dict(text=txtchar(2,rms2,freq2,ptp2)),
      dict(text='CH1: {0:4.2f} V/Div'.format(vscale1.value)),dict(text='CH2: {0:4.2f} V/Div'.format(vscale2.value)),
      dict(text=str(tbase.value)+' µs/Div')])

capture_oscscope()

capture.on_click(capture_oscscope)
tbase.observe(capture_oscscope,'value')
vscale1.observe(capture_oscscope,'value')
vpos1.observe(capture_oscscope,'value')
coupling1.observe(capture_oscscope,'value')
vscale2.observe(capture_oscscope,'value')
vpos2.observe(capture_oscscope,'value')
coupling2.observe(capture_oscscope,'value')
```

```
display(ps_title,ps_ui)
display(wg_title,wg_ui)
display(osc_title,osc_ui)
```