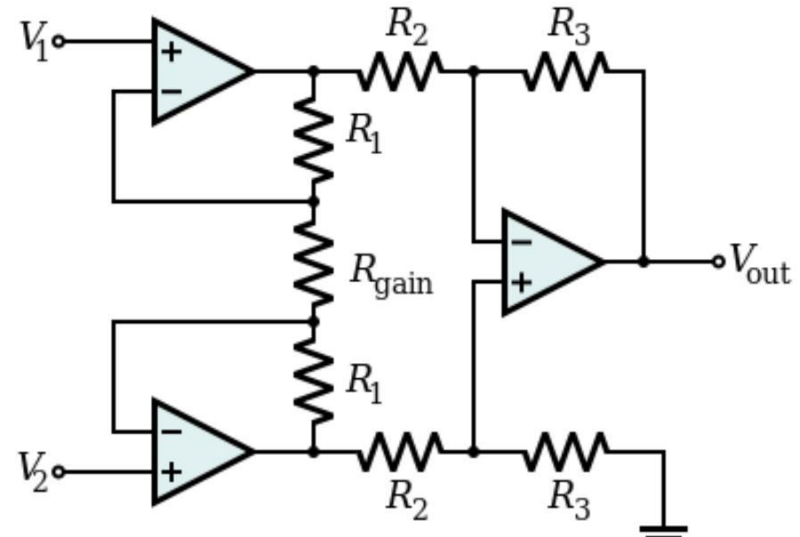
MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF PHYSICS

PRINCIPLES OF MEASUREMENT AND INSTRUMENTATION I

LABORATORY MANUAL



Prepared by

Tarek Elsebaei

Revised by

Deniz Orhun Boztemur

Lab Rules

1. In this course we will be using open-source **Arduino Software (IDE) 1.8.8** and **Python 3**. Moreover, it is definitely recommended to use your own PC and write the code of programs in the procedure of each experiment by your own before coming to the lab. You will also be graded for this arrangement as a bonus.

2. Before the experiment you should **read the laboratory manual carefully**. You may need to have some conceptual information about the related topics.

3. You must do the **preliminary work as briefly on a paper and hand in before the experiment**, you will also be graded for this arrangement.

4. At the end of the experiment you will be asked to prepare a report and submit it before the next experiment. **The reports may be written either by hand or a computer. But they must be delivered in the form of hard copy. Hence you need to print out your report document.**

5. The reports must include the **name/surname, student ID, experiment date and submission date** information on the cover page.

6. The report should be written in **Times New Roman** font and **12pt** font size.

7. **Report** to be prepared should be consist of the following sections:
   Objectives - **Introduction** - **Equipment** - **Procedure** - **Calculations** - **Discussion and Comments**. In the Discussion and Comments part, you should discuss how you did the experiment and what you have learnt as a result. If you wish to write your report on LaTeX, a template can be provided.

8. Make sure that each figure that you include in your report contains both the **figure number and related caption**. Example: "Fig.1. Response time of a photodiode"

9. Late reports may be accepted as long as you represent your valid excuse **BEFORE** the deadline of the report. On other circumstances, the grades will be cut off in the following manner:
   - 1 day of delay: **30%**
   - 2 days of delay: **60%**
   - 3 days of delay: **100%**

10. After your lab session, turn off all test equipment and return equipment, tools, and cables to their proper storage area.

11. In each experiment there will be some questions in the manual to be answered carefully and must be written in the corresponding report section.

12. You are going to **fail** from the lab (and also from the course) if you miss one section unless you have a medical report. So, please attend all the experiments at your assigned sections.

EXP.0 Introduction to Laboratory Measuring Devices

Purpose

- To learn how to use the electronic board and observe their properties.
- To learn how to construct a circuit on the bread board.
- To set up a typical oscilloscope for operation
- To learn how to read a signal and adjust the oscilloscope for optimum calibration.
- To learn how to use Signal Generator and Digital Multimeter.

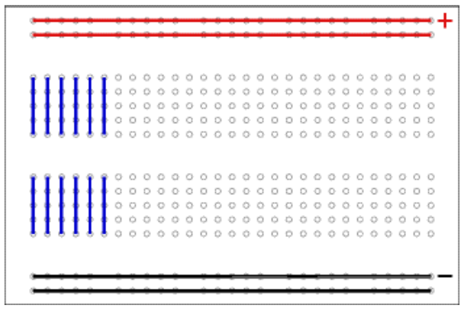
Equipment

Advanced Analog & Digital Electronic Design Workstation

The PB-507 Advanced Analog & Digital Electronic Design Workstation consists of various functional parts: Breadboard - DC Power Supply - AC Power Supply - Function Generator - Pulse Generator - Frequency Counter - Logic Indicators - Logic Probe - Hex to 7 Segments Decoder - Debounced Pushbuttons - Logic switches - SPDT switches - BNC connectors – Potentiometers – Speaker

Bread Board

A breadboard also known as protoboard is a type of solderless electronic circuit building. Inside the board, bars and rails allow to construct the circuits. It is well designed for easy mounting circuit elements such as op-amp's, resistors, transistors, seven segment displays, etc.

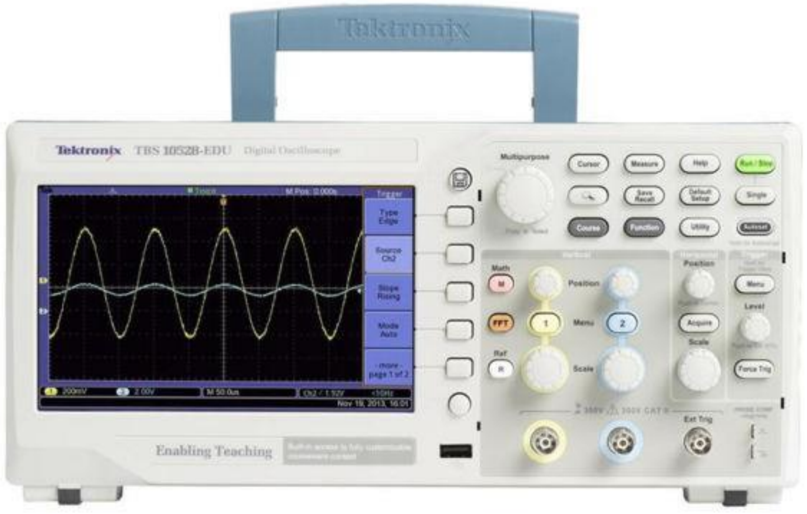


Arbitrary Function Generator AFG1022

The AFG1022 Arbitrary Function Generator provides a waveform generation tool. It includes dual channel, 25 MHz bandwidth and up-to 10 V_{p-p} output amplitude. The four run modes, 50 built-in frequently-used waveforms and the built-in 200 MHz frequency counter cover most waveform generation needs in your experiment and test jobs. The LCD, short-cut buttons, USB interface and PC software provide the most intuitive ways to configure the instrument.

Series Digital Storage Oscilloscope TBS1000B-EDU

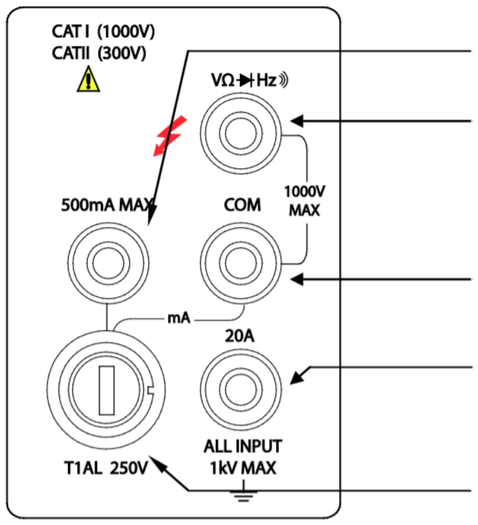
The oscilloscope is one of the most versatile electronic test instruments. Scope specifications concern the type of scope (single or dual trace), its bandwidth (its operating frequency range) and sensitivity. The instrument includes a 7-inch color display, up to 2 GS/s sampling rate, bandwidths from 50 MHz to 200 MHz, dual channel frequency counters.



Bench Multimeter B&K Precision 5491B

Measurement ranges:

- DC voltage from 50μV to 1000V
- AC (RMS) voltage from 1mV to 750V
- DC current from 100nA to 20A
- AC (RMS) current from 100nA to 20A
- Two -wire resistance from 10$m\Omega$ to 20 MΩ (50 MΩ for model 5491B)
- Frequency from 5Hz to 1MHz



2 mA-200 mA for 2831E, 5 mA-500 mA for model 5491B range DC/AC current input terminal for DC/AC current

Input-High terminal for Volts, ohms, Hz, period, diode and continuity measurements

Common terminal

2 A (5 A for model 5491B) and 20 A range current input terminal for DC/AC current measurement

T1AL/250 V type fuse for the mA measurement range

Input protection Limits

Function	Input Terminals	Maximum Allowable Input
DCV	VΩ→| to **COM**	1010V DC
ACV,HZ	VΩ→| to **COM**	757.5V AC RMS,1000V Peak
mA, HZ	500mA to **COM**	200mA (Model 5491B: 500 mA) DC or AC RMS
20A,HZ	20A to **COM**	20A DC or AC RMS
Ω	VΩ→| to **COM**	500V DC or AC RMS
→| , »))	VΩ→| to **COM**	500V DC or AC RMS
All functions	Any terminals to earth	1000V DC or 1000V peak AC

Power Supply Geratech 305D-2

A power supply is an electrical device that supplies electric power to an electrical load. Our dual channel power supply can provide output voltage up to 30 V and 5 A. The fixed 5 voltage output provides current of 2A.

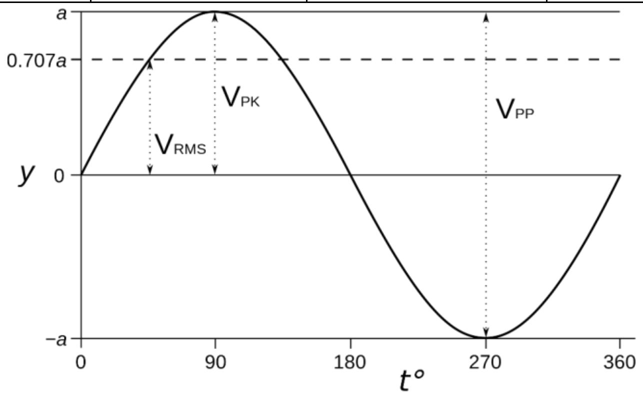
Procedure

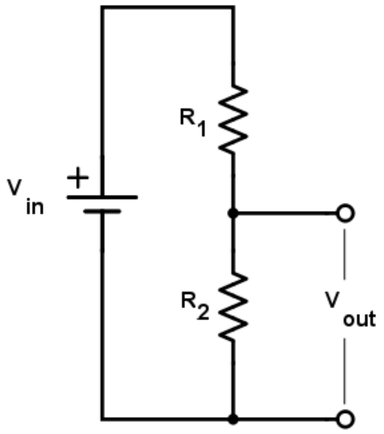
(**Attention:** Before power is on, make sure that there is no short-circuit, otherwise you may damage electronic devices. Switch the power to OFF while you are changing the Multimeter mode.)

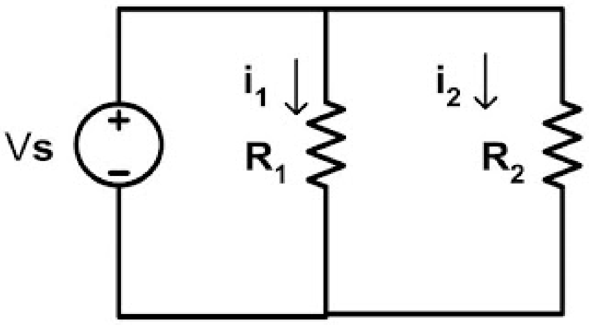
1- From the function generator, give five different frequency of sinus signals between 1Hz to 1 MHz and try to read their frequency and voltage from the oscilloscope. Record your frequencies and voltages. Do the same procedure for other wave forms.

2- For each frequency of signal note all V_p (peak voltage), V_{p-p} (peak to peak voltage) and calculate V_{rms} voltage. (V_{rms}=0.707 x V_p) Then, compare your results with measured values on the Multimeter.

Frequency					
V_p					
V_{p-p}					
V_{rms}					







3- Construct voltage divider and current divider circuits as shown above on the breadboard and calculate the desired voltages and currents compare your theoretical values with the measured ones on the Multimeter. Supply the circuits with 5V and Use 10kΩ for R_1 and R_2.

	Voltage Divider		Current Divider	
	I	V	I	V
Calculated				
Measured				

4- Using a potentiometer construct another voltage divider circuit and measure V_{out}. Calculate the expected value of resistance of the potentiometer using V_{in} and V_{out}. Then, measure the resistance of the potentiometer. Compare your results.

	Resistance
Calculated	Ω
Measured	Ω

There will be no report for Experiment 0. Be sure to hand in your preliminary work before conducting Experiment 1!

EXP.1 Methods of Signal Conversion: ADC & DAC

Preliminary Work

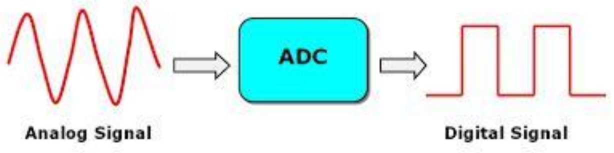
- Explain the basic process of converting an analog signal to digital.
- Describe the purpose of the sample-and-hold function.
- Explain what an operational amplifier is.
- Show how the op-amp can be used as an inverting amplifier or comparator.
- Show how the op-amp can be used to sum up voltage signals.
- Explain the operation of a binary-weighted-input DAC.
- Explain the operation of an R/2R ladder DAC.
- Explore the LM324 & LM741 datasheets, draw their schematic pin configuration.
- Print out the LM324 & LM741 datasheets.
- Use an electronic simulation software (ask the assistant for a copy of Proteus 8.8 if you don't have it) to construct and design 2-bit ADC as shown in Figure 2.2 by showing the values used for resistances and voltages at points A, B, C and D.

Purpose

To construct and design an Analog-to-Digital and Digital-to-Analog Converter circuits

Theory

We live in an analog world which is a time-varying "quantity" which convey some sort of information. These quantities are usually voltage that's changing over time. Signals are passed between devices in order to send and receive information, which might be video, audio, or some sort of encoded data. On the other hand, most commonly digital signals will be one of two values – like either 0 V or 5 V. As shown below, Analog waves are smooth and continuous, digital waves are stepping, square, and discrete.



Operational Amplifier

An op-amp is a linear amplifier that has two inputs (inverting and noninverting) and one output. It has a very high voltage gain and a very high input impedance, as well as a very low output impedance. The op-amp voltage comparator compares the magnitudes of two voltage inputs and determines which is the larger of the two. Due to this high open loop gain, the output from the comparator swings either fully to its positive supply rail, +Vcc or fully to its negative supply rail, -Vcc

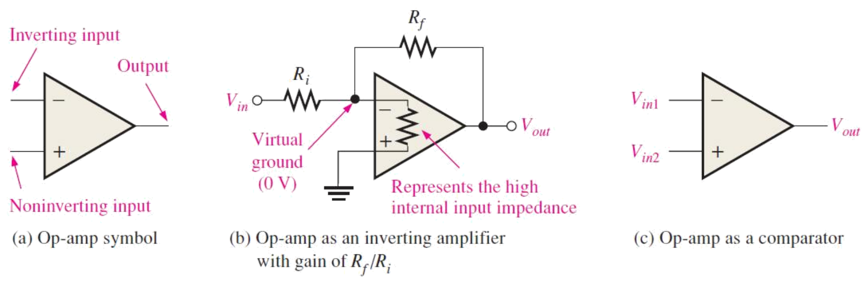


FIGURE 2.1 Operational Amplifier

Analog-to-Digital Converter

Analog-to-Digital converters (ADC) translate analog signals, real world signals like temperature, pressure, voltage, current, distance, or light intensity, into a digital representation of that signal. This digital representation can then be processed, manipulated, computed, transmitted or stored. We can build a voltage divided branches to adjust the reference voltages and then compare them with the analog input signal to be converted into a digital one.

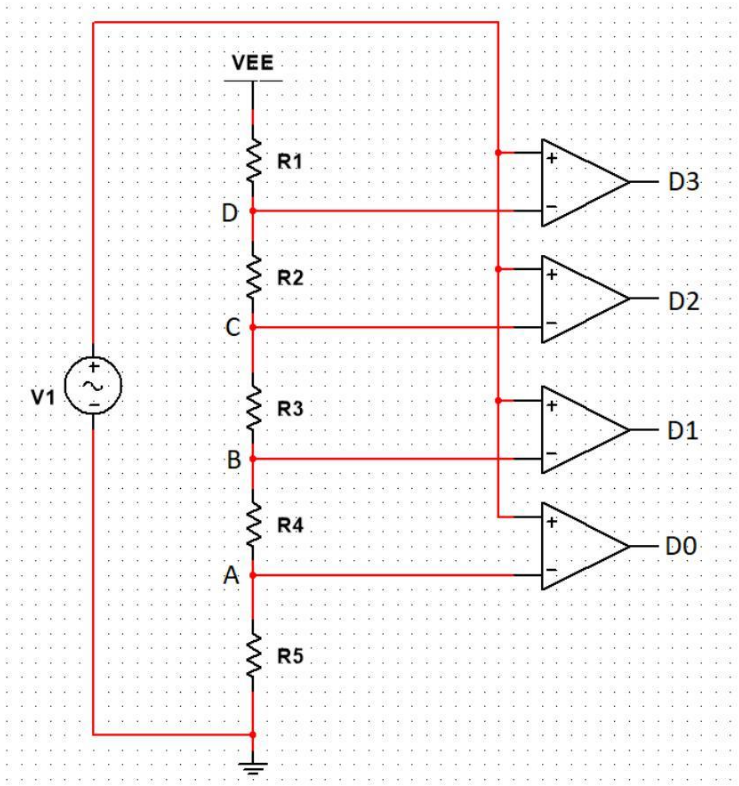


FIGURE 2.2 2-bit Analog-to-Digital Converter

Binary-Weighted-Input Digital-to-Analog Converter

One method of digital-to-analog conversion uses a resistor network with resistance values that represent the binary weights of the input bits of the digital code.

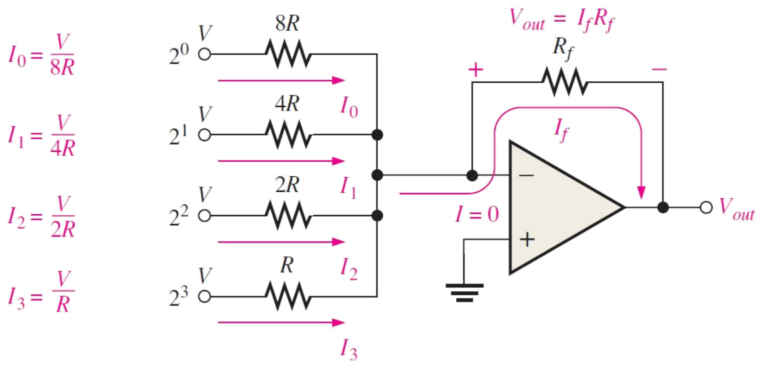


FIGURE 2.3 4-bit Binary-Weighted-Input Digital-to-Analog Converter

The R/2R Ladder Digital-to-Analog Converter

Another method of digital-to-analog conversion is the R/2R ladder, as shown in Figure 2.4 for four bits. It overcomes one of the problems in the binary-weighted-input DAC in that it requires only two resistor values.

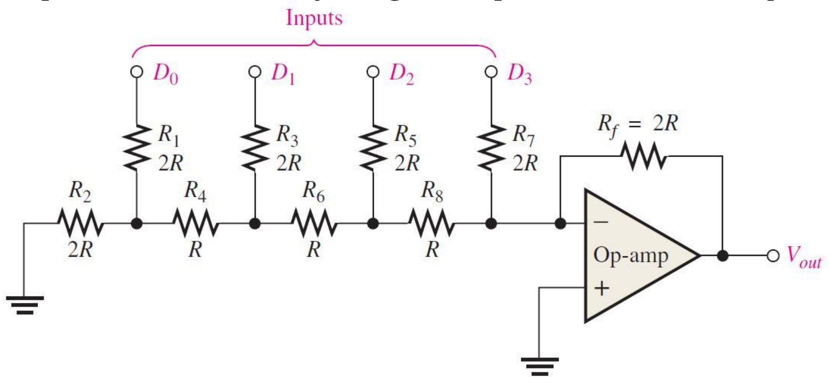


FIGURE 2.4 4-bit R/2R ladder Digital-to-Analog Converter

The ideal output is a straight-line stair step. As the number of bits in the binary code is increased, the resolution is improved. That is, the number of discrete steps increases, and the output approaches a straight-line linear ramp.

Equipment

- Multimeter
- Breadboard
- Oscilloscope
- Resistors
- Op-Amp IC LM741
- Quad Op-Amp IC LM324

Procedure

1. Construct the ADC as in figure 2.2 by using the quad Op-Amp IC LM324

   Hint: use equal resistor values to divide the voltage in equal terms

- Try high resistance values and low resistance values then compare the results with justification
- Record the voltage values at points A, B, C, D
- Record the values of voltage output D0, D1, D2, D3 as you change V1 from 0 v to 5 v
- Discuss the importance of connecting the output to a priority encoder and add its design
- Discuss testing ADCs for a missing code, incorrect code and offset
- Describe the purpose of an ADC

2. Construct the 4-bit Binary-Weighted-Input DAC as in figure2.5 using LM741

- Explain why this method is not preferable to be used for the Digital to Analog conversion
- Verify and note your experimental results

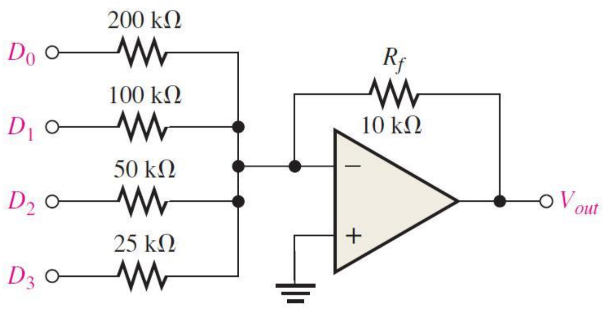


FIGURE 2.5 4-bit Binary-Weighted-Input Digital-to-Analog Converter

3. Construct the 4-bit R/2R ladder DAC as in figure2.4 using LM741 Op-Amp

- Choose a value for resistances R/2R and clarify the reasons behind your choice
- Observe Vout change while increasing the input D0, D1, D2 and D3
- Note the values of Vout for each D3D2D1D0 and then plot a graph for your results
- Determine the resolution of your DAC
- Discuss accuracy, linearity, monotonicity, and settling time of your DAC
- Explain why this method overcomes the problems in the Binary-Weighted-Input DAC
- Discuss the testing of DACs for nonmonotonicity, differential nonlinearity, low or high gain, and offset error

EXP.2 Arduino-Based Measurement Devices

Preliminary Work

- Explain the difference between microprocessor, microcomputer and microcontroller.
- Explain the difference between an analog signal and a digital signal.
- Explain the basic principle of the PWM. (Pulse Width Modulation).
- Describe the purpose of the following functions in Arduino:

analogRead() – analogWrite() – digitalRead() – digitalWrite() – pinMode() – Serial.begin() –

delay() – millis() – Serial.print() – Serial.println() – Serial.parseInt() – map() – random()

- What does Clock Speed mean for a microcontroller? How is it related to the time response of the
- microcontroller?
- Explain INTERRUPT on Arduino with an analogy.
- What is the resolution of Analog-to-Digital Converter in Arduino UNO?
- **Writing the codes before the experiment and handing them in with the preliminary report will get you 10 bonus points!**

Purpose

To explore Arduino Uno and Arduino based circuits. To learn the basic Arduino programming.

Theory

Microcontrollers are being used as simple solutions to tasks that previously utilized transistor-transistor-logic. With the profusion of these devices being used in industry as well as in hobby electronics, the ability to program them can be very useful for any project that may be attempted. While microprocessors, microcomputers and microcontrollers all share certain characteristics and the terms are often used interchangeably, there are certain distinctions that are used to classify them into separate categories.

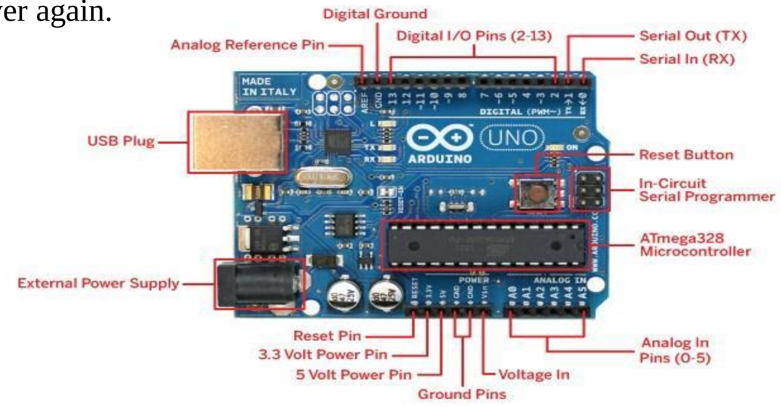
The simplest of the three categories is the **microprocessor**. Also known as a CPU (Central Processing Unit), these devices are generally found at the heart of a much larger system such as a desktop computer and are primarily used as data processors. They generally consist of an arithmetic logic unit (ALU), an instruction decoder, a number of registers and digital input/output (DIO) lines. Some processors also include memory spaces such as a cache or stack which can be used for more rapid temporary storage and retrieval of data than having to access system memory. Additionally, the processor must connect 1 to some form of data bus to access the memory and input/output peripherals external to the processor itself.

A **microcontroller** is, in some ways, a cross between a microprocessor and a microcomputer. Like microprocessors, the term microcontroller refers to a single device; however, it contains the entire microcomputer on that single chip. Therefore, a microcontroller will have a processor, on-board memory as well as a variety of IO devices. A microcontroller is a computer present in a single integrated circuit which is dedicated to perform one task and execute one specific application. It contains memory, programmable input/output peripherals as well as a processor. Microcontrollers are mostly designed for embedded applications and are heavily used in automatically controlled electronic devices such as cellphones, cameras, microwave ovens, washing machines, etc.

The **Arduino** environment has been designed to be easy to use for beginners who have no software or electronics experience. With Arduino, you can build objects that can respond to and/or control light, sound, touch, and movement. Arduino has been used to create an amazing variety of things, including musical instruments, robots, light sculptures, games, interactive furniture, and even interactive clothing. Arduino is an open-source electronic platform that use Atmel microcontroller chips based on easy-to-use hardware and software. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino Programming Language (based on Wiring), and the Arduino software IDE, based on Processing.

Arduino Uno

Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started. You can tinker with your UNO without worrying too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again.



Technical Specifications

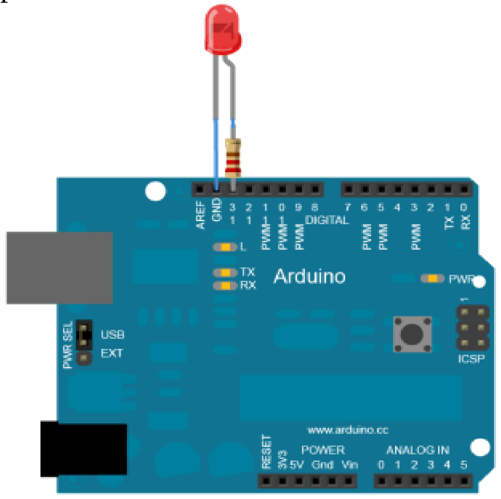
Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage(recommended)	7-12V
Input Voltage(limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

Equipment

- Electronic Board
- Arduino UNO
- Resistors
- LED
- Push Button
- Potentiometer

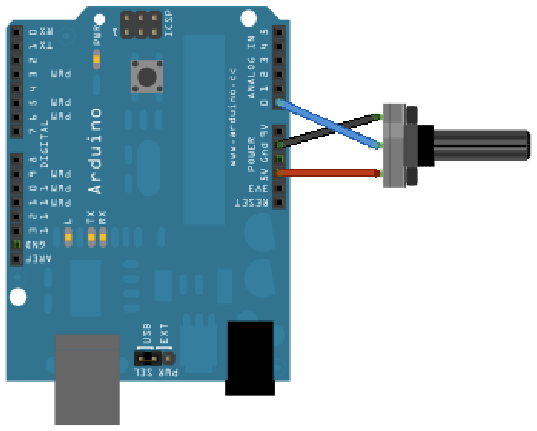
Procedure

1. Write a program on Arduino platform to blink an LED for one second by using the delay function.



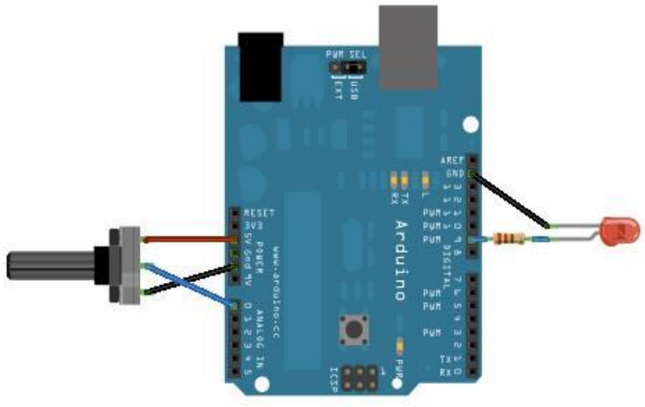
- Write a program to blink the LED without using the delay function
- Explain the reasons behind using a current-limiting resistor for the LED

2. By using the 1k ohm potentiometer implemented in the electronic board, connect one terminal to 5v and the other terminal to ground and connect the output of the potentiometer to one of the Arduino pins. Use analog read function and serial print to read the output.

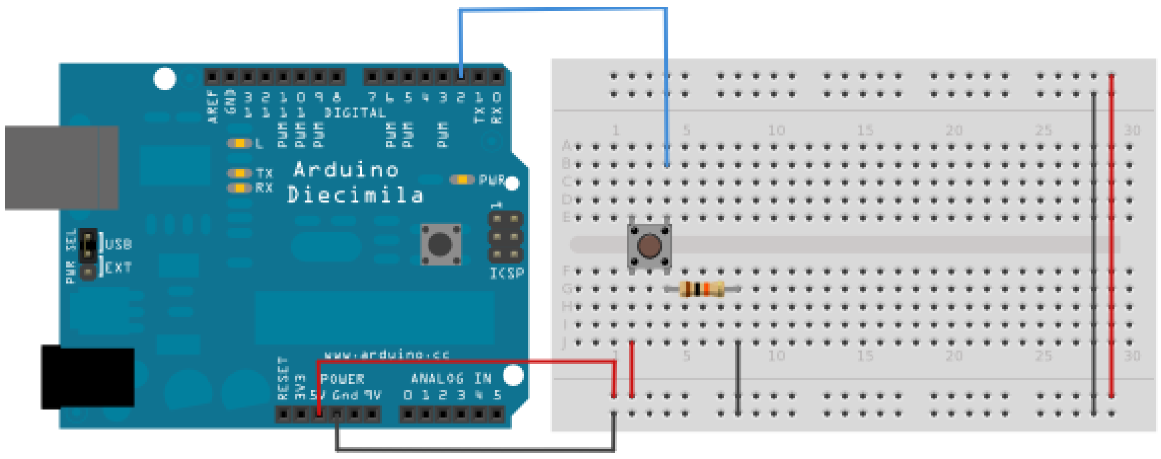


- What are your results? And what is the resolution of your Arduino analog-to-digital converter?
- By using map function in Arduino make your results scale to be from 0 to 5
- Write a program to read a float type of the potentiometer output from 0 to 5

3. By using the same circuit as in step 1 and 2, use the analog write function in Arduino to control the brightness of an LED.

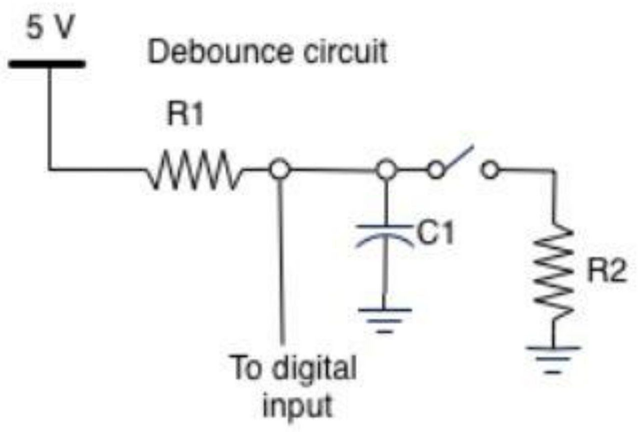


4. Write a program that can count the number of clicks on a push button by using Arduino interrupt service routine. Discuss your results.



Hint: choose the pin mode of the interrupt pin to be INPUT_PULLUP and add the reason of that to your report.

5. Solve the problem by constructing the circuit below.



- How the capacitor can solve this problem? (C1 = 100nF)
- Compare your results with the previous part.

EXP.3 Data Acquisition

Preliminary Work

- ➢ In Python, what do these functions do:

split() – encode() – decode() – len() – min() – max() – sorted() – sum() – type() – strip() – count() – clear() – range() – map() – str() – int() – repr() – index() – find() – enumerate() – next() – open() – round()

In the python pyserial package;

- ➢ Explain the purpose of the following functions:

write() – open() – close() – read() – readline() – flushInput() – flushOutput() – inWaiting()

- And define the following parameters:

port – baudrate – bytesize – parity

In the python pyvisa package;

- ➢ Explain the purpose of the following functions:

get_instruments_list() – instrument() – ask() – write() – trigger() – read_raw() - read() – clear() – query_values()

- And define what these messages mean:

*'*IDN?' – 'CURV?' – '*RST'*

- ➢ In the python struct package, what is the result of using *unpack()* function
- ➢ In the python NumPy package, explain the purpose of the following functions:

array() – arange() – linspace() – append() – indices() – shape() – reshape() – split() – delete()

- ➢ **Writing the codes before the experiment and handing them in with the preliminary report will get you 10 bonus points!**

Purpose

To learn how to interface with hardware using Python. We will start by interfacing Oscilloscope and achieve data acquisition using pyvisa library. We will then have Arduino interact with PC using pyserial library.

Theory

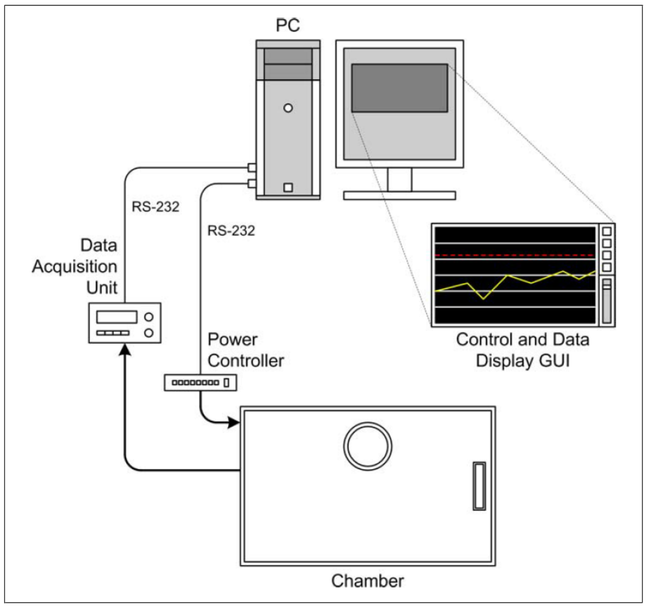
Data Acquisition (DAQ) is the process of measuring real-world conditions and converting these measurements into digital readings at fixed-time intervals (the data sample rate).

In the days of steam and brass, one might have monitored the pressure within a boiler or a pipe by means of a mechanical gauge. In order to capture data from the gauge, someone would have to write down the readings at certain times in a logbook or on a sheet of paper. Nowadays, we would use a transducer to convert the physical phenomenon of pressure into a voltage level that would then be digitized and acquired by a computer.

As implied above, some input data will already be in digital form, such as that from switches or other on/off–type sensors—or it might be a stream of bits from some type of serial interface (such as RS-232 or USB). In other cases, it will be analog data in the form of a continuously variable signal (perhaps a voltage or a current) that is sensed and then converted into a digital format.

From a computer's viewpoint, all data is composed of digital values, and all digital values are represented by voltage or current levels in the computer's internal circuitry. In the world outside of the computer, physical actions or phenomena that cannot be represented directly as digital values must be translated into either voltage or current, and then translated into a digital form. The ability to convert real-world data into a digital form is a vast improvement over how things were done in the past.

The next figure shows an example of an instrumentation system for controlling an environmental chamber. What is important are the instruments connected to it and how they, in turn, are interfaced to the computer? The data acquisition instrument is responsible for sensing and converting analog signals such as temperature. It might also monitor the electrical status of any heaters or coolers attached to the chamber. The power controller instrument is responsible for any heaters, coolers, cryogenic valves, or other controlled functions in the chamber.



In an electronics laboratory, or even a well-equipped hobbyist's workshop, it wouldn't be unusual to encounter oscilloscopes, logic analyzers, frequency meters, signal generators, and other such devices. While these are useful devices in their own right, when incorporated into an automated system they can become even more useful.

In order to use a piece of test equipment in an automated setup, there must be some type of control or acquisition interface available. Many modern instruments incorporate USB, Ethernet, GPIB, RS-232, or a combination of these. We will be using these measurement devices in this experiment and collect the data through USB serial communication to analyze or visualize it using Python.

The main objective of this experiment is to show you how you can extend Python to take advantage of existing binary library modules. Out of the box, Python cannot directly access the underlying hardware, nor can it interface directly with the software library modules provided by most hardware vendors. Python can, however, communicate with anything connected to a serial port or to a USB device that utilizes what is referred to as a virtual serial port (many USB–to–RS-485 converters use this technique). For those types of applications, an extension module typically isn't necessary.

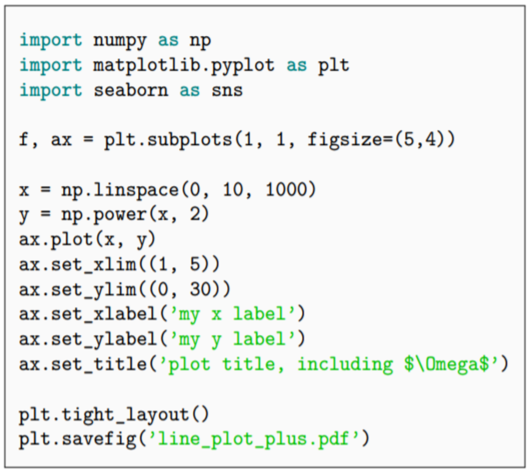
This course assumes some prior knowledge of Python programming. Here we will provide an introduction to Python modules commonly used in the field of Test and Measurements. Python is a great general-purpose programming language on its own, but with the help of a few popular libraries (numpy, scipy, matplotlib) it becomes a powerful environment for scientific computing. Python is a very clear and powerful programming language. It's easy for beginners, but goes a long way, even in a scientific context. Python also has a very elegant syntax. That makes it easy to read, and write, and importantly, to maintain your programs. Python has a large standard library, which supports many common programming tasks. It also has a very useful interactive mode for experimenting with code and with your ideas. Python is also highly portable. It runs on Windows, Mac OSX, UNIX, Linux, Solaris, and it even runs on your cell phone. Python is free to download and use, and it has a very large online user community.

NumPy & SciPy

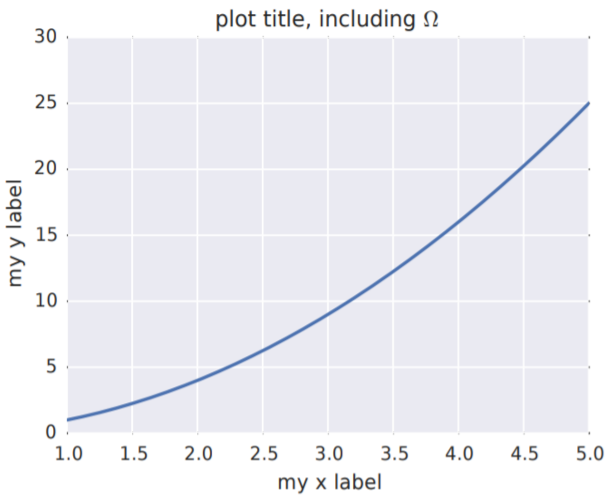
NumPy and SciPy are open-source add-on modules to Python that provide common mathematical and numerical routines in pre-compiled, fast functions. The NumPy (Numeric Python) package provides basic routines for manipulating large arrays and matrices of numeric data. The SciPy (Scientific Python) package extends the functionality of NumPy with a substantial collection of useful algorithms, like minimization, Fourier transformation, regression, and other applied mathematical techniques.

Matplotlib

It is a Plotting library for Python. It Works well with Numpy and has similar syntax as of Matlab.



```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

f, ax = plt.subplots(1, 1, figsize=(5,4))

x = np.linspace(0, 10, 1000)
y = np.power(x, 2)
ax.plot(x, y)
ax.set_xlim((1, 5))
ax.set_ylim((0, 30))
ax.set_xlabel('my x label')
ax.set_ylabel('my y label')
ax.set_title('plot title, including $\Omega$')

plt.tight_layout()
plt.savefig('line_plot_plus.pdf')
```



PyVISA

PyVISA is a Python package that enables you to control all kinds of measurement devices independently of the interface (e.g. GPIB, RS232, USB, and Ethernet). As an example, reading self-identification from a Keithley Multimeter with GPIB number 12 is as easy as three lines of Python code:

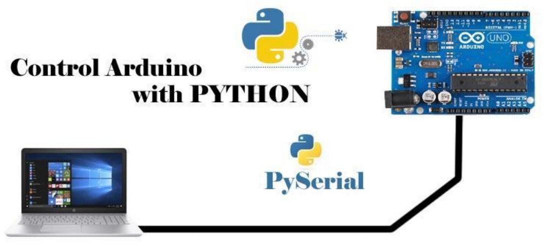
Control Equipments
with PYTHON
PyVISA

```python
import visa #This imports the PyVisa Library.
#Next line is used to connect to the resource manager file.
rm = visa.ResourceManager(r'C:\Windows\System32\visa32.dll')

rm.list_resources() #This line lists all the connected devices.
#To Print the first element or first connected device.
print (rm.list_resources()[0])
#Next line connects the first device available.
myFirstInstrument = rm.open_resource(rm.list_resources()[0])
#To Print the first elements make and model number
print(myFirstInstrument.query('*IDN?'))
#Connecting to the device.
myDevice = rm.get_instrument(rm.list_resources()[0])
myDevice.write("*RST")  #Resets the device connected.
```

PySerial

<u>Serial Communication using Python and Arduino</u>

```python
import serial #This imports the PySerial Library.
#You can find which COM port has been used from Device Manager.
#And the Baud rate is selected here as 9600.

arduino = serial.Serial('COM3', 9600)

arduino.write('C')
#This command will send a character ‘C’ through the serial port to Arduino.
#Before that you have to close the Arduino IDE if it is connected.
#Note: Provide proper time delay between sending two consecutive data .
data=arduino.readline()
#This command reads a line data from serial port
#and stores in a variable named ‘data’.
```



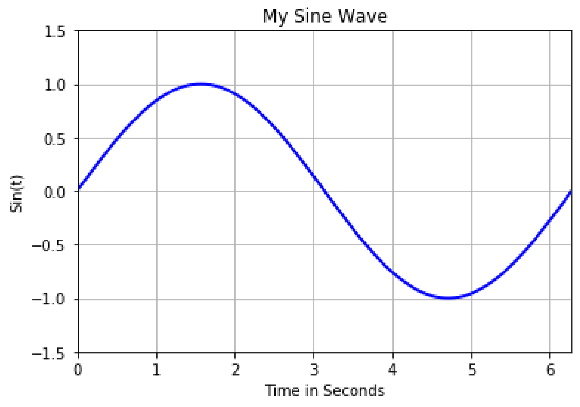
Equipment

- PC
- Oscilloscope
- Arduino UNO
- Thermistor

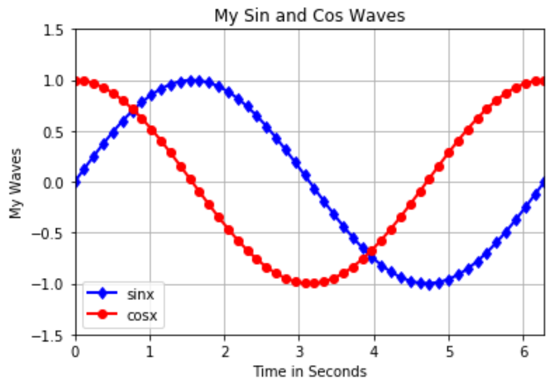
Procedure

1. In this part, we will try to run some programs to learn how to plot graphs using Matplotlib

```python
import numpy as np
import matplotlib.pyplot as plt

x=[]
y=[]


for i in np.arange(0,2*np.pi,2*np.pi/1000):
    x.append(i)
    y.append(np.sin(i))

plt.plot(x,y, 'b-', linewidth=2)
plt.grid(True)
plt.axis([0,2*np.pi,-1.5,1.5])
plt.title('My Sine Wave')
plt.xlabel('Time in Seconds')
plt.ylabel('Sin(t)')
plt.show()
```

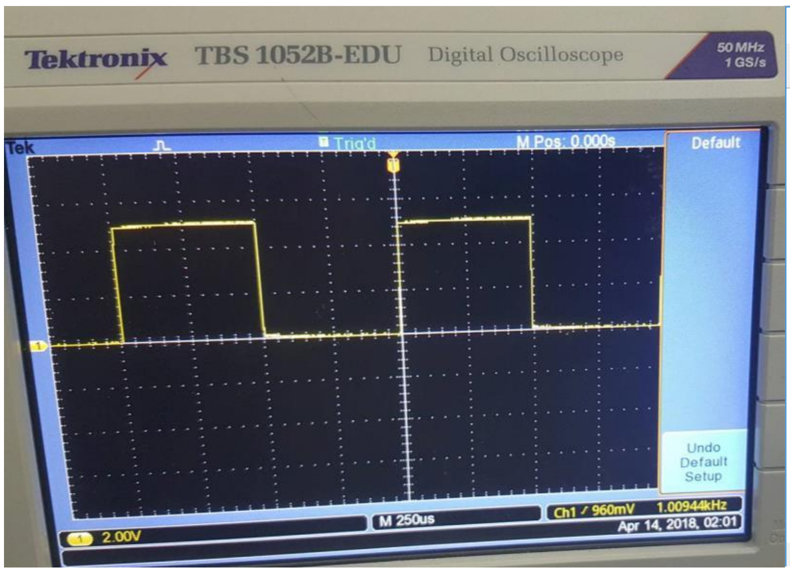


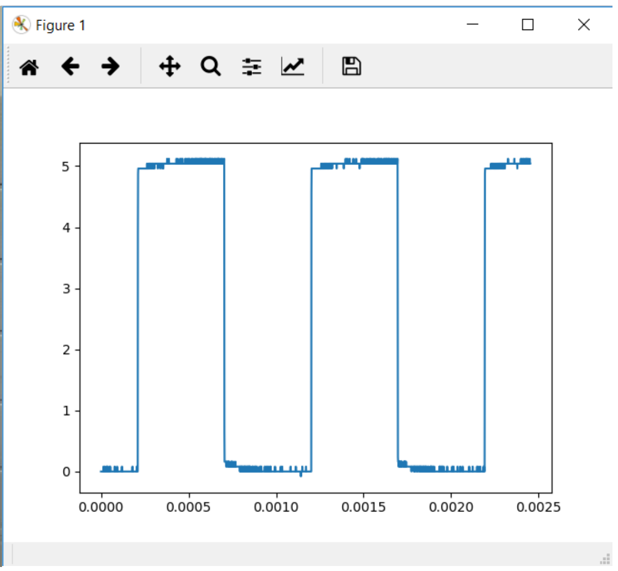
```python
import numpy as np #import numpy library
import matplotlib.pyplot as plt #import matplotlib

x= np.linspace( 0, 2*np.pi, 50)#create your x array
y= np.sin(x) #create y array
z= np.cos(x) #create z array
plt.plot(x,y, 'b-d', linewidth=2, label='sinx') #plot y
plt.plot(x,z, 'r-o', linewidth=2, label='cosx') #plot z
plt.grid(True) #display background grid
plt.axis([0,2*np.pi,-1.5,1.5]) #set range on axis
plt.title('My Sin and Cos Waves') #chart title
plt.xlabel('Time in Seconds') #label x axis
plt.ylabel('My Waves') #label y axis
plt.legend() #show legend
plt.show() #show the plot
```



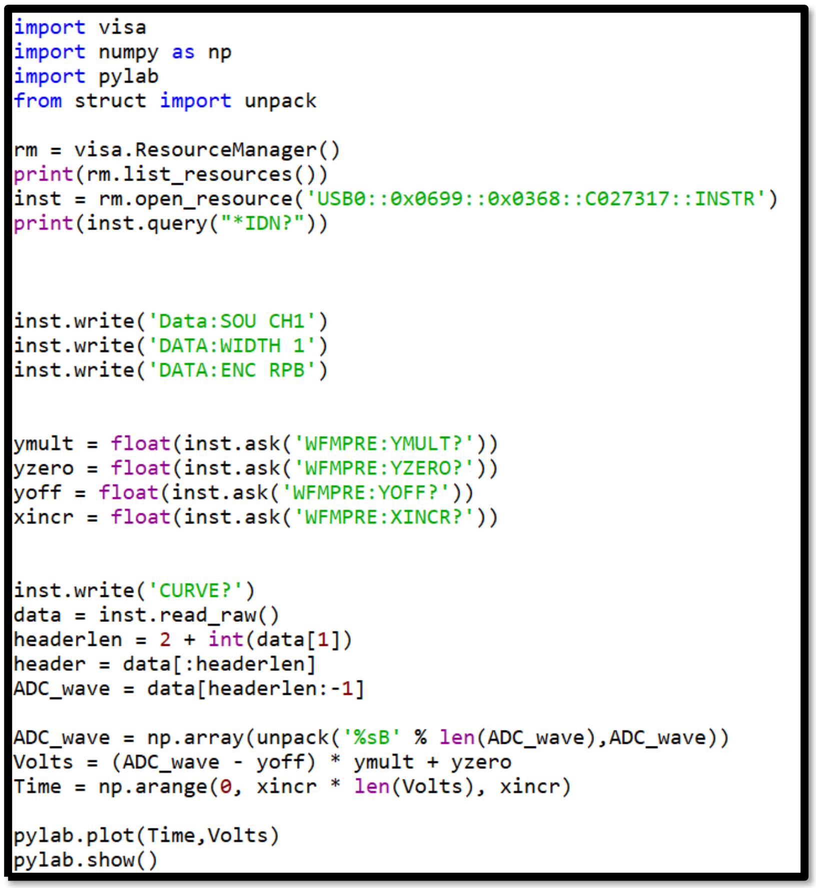
2. Now use pip to install the drawnow to plot live data coming from Arduino using Python and Matplotlib
Hint: Write in the console: *pip install drawnow*

3. By using PyVISA, write the next code on your Python editor to get a picture from your oscilloscope.





Hint: You should download and install NI VISA library as well.



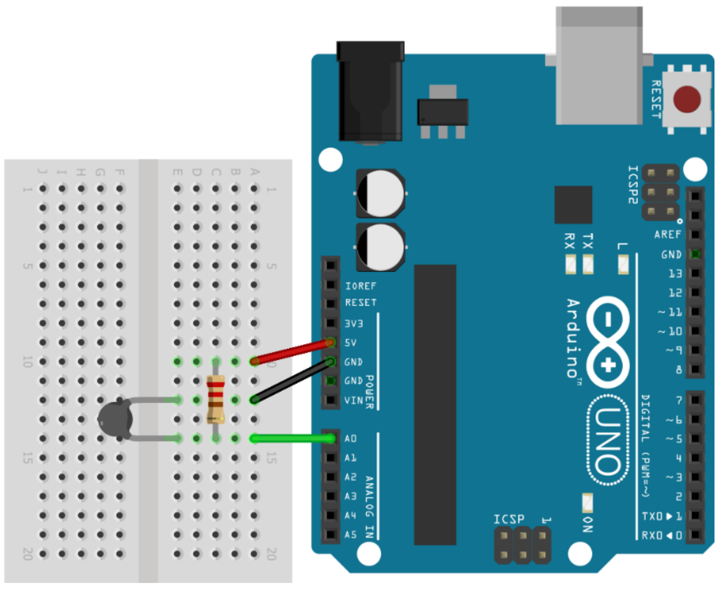
```
import visa
import numpy as np
import pylab
from struct import unpack

rm = visa.ResourceManager()
print(rm.list_resources())
inst = rm.open_resource('USB0::0x0699::0x0368::C027317::INSTR')
print(inst.query("*IDN?"))



inst.write('Data:SOU CH1')
inst.write('DATA:WIDTH 1')
inst.write('DATA:ENC RPB')


ymult = float(inst.ask('WFMPRE:YMULT?'))
yzero = float(inst.ask('WFMPRE:YZERO?'))
yoff = float(inst.ask('WFMPRE:YOFF?'))
xincr = float(inst.ask('WFMPRE:XINCR?'))


inst.write('CURVE?')
data = inst.read_raw()
headerlen = 2 + int(data[1])
header = data[:headerlen]
ADC_wave = data[headerlen:-1]

ADC_wave = np.array(unpack('%sB' % len(ADC_wave),ADC_wave))
Volts = (ADC_wave - yoff) * ymult + yzero
Time = np.arange(0, xincr * len(Volts), xincr)

pylab.plot(Time,Volts)
pylab.show()
```

Hint: change the green underlined part of the code below with your instrument's one.

inst = rm.open_resource('USB0::0x0699::0x0368::C027317::INSTR')

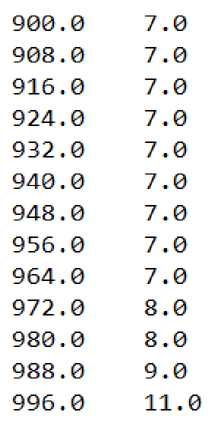
4. Write a program that can read the data of both the two channels of the oscilloscope

5. By using a thermistor, construct the circuit below. Use analog read function and serial print to read the output.



6. Use PySerial library to read the data in the previous part by transferring data through USB serial port to a PC. And then visualize it using Matplotlib

7. Add a code that can record the Volts and Time in a txt file in two columns as shown in the following manner



```
900.0    7.0
908.0    7.0
916.0    7.0
924.0    7.0
932.0    7.0
940.0    7.0
948.0    7.0
956.0    7.0
964.0    7.0
972.0    8.0
980.0    8.0
988.0    9.0
996.0    11.0
```

8. Use pyserial library to send a command to Arduino to blink an LED

EXP.4 Communication Protocols: I^2C & SPI Compatible Sensors

Preliminary Work

- Discuss the working principle of a digital potentiometer and its design
- Discuss the thermoelectric effect and the pyroelectric effect
- Explore the datasheet of your digital potentiometer and describe its principle of operation and programming
- Describe the working principle of a thermocouple and its types, advantages and disadvantages.
- Explain how accelerometers measure acceleration
- Show the difference in the working principle between Thermostat, Thermistor, Resistive Temperature
- Detectors (RTD), Semiconductor-based temperature sensor.
- **Writing the codes before the experiment and handing them in with the preliminary report will get you 10 bonus points!**

Purpose

- To build a controllable-gain amplifier circuit using a digital potentiometer
- To measure the temperature using I^2C or SPI compatible temperature sensor module To measure the acceleration using I^2C or SPI compatible accelerometer module

Theory

A protocol is a set of rules and conventions that govern the exchange of information between devices (end points) in a communication network. In all IoT ecosystems, devices communicate using binary protocols, meaning that all information is transmitted as endless trails of 0s and 1s (zeroes and ones).

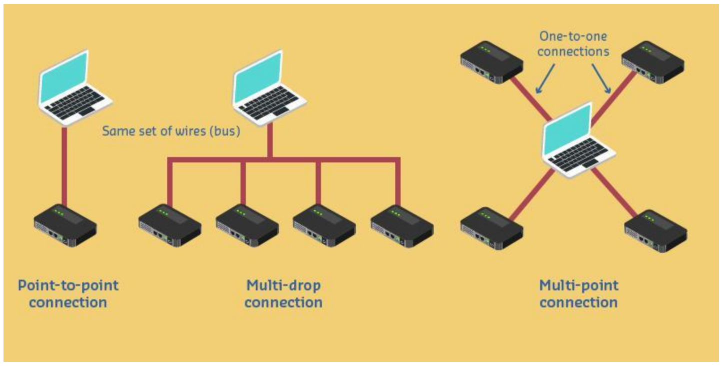
Synchronous systems use a common clock that has to be transmitted on one of the wires. On the other hand, asynchronous systems do not require transmitting the clock over the cable. In asynchronous communication, devices run their own clock, all at the same speed, and the data lines are used to achieve synchronization.

In some communication networks only one device, called the master, can talk to all the others, the slaves. Slaves cannot talk to each other directly; information transfers between slaves have to go through the master. In most transactions, the master initiates the exchange of information and determines when it finishes. To use the communication channel, slaves have to wait until the master grants them access, otherwise more than one slave could be transmitting information at the same, causing a collision. Moreover, some master-slave protocols allow for more than one device to act as master. In these instances, an arbiter decides which of the masters will take control of the channel and for how long.

The maximum speed of a communication channel depends on the processing speed of the end points and the physical properties of the medium (capacitance, inductance). The maximum length of a cable is determined by its physical properties (resistance, capacitance, and inductance) and the speed at which information is transmitted.

High speeds imply transmitting high frequency signals over the wires. At high frequencies, normal wires behave like filters and antennae, deteriorating the quality of the transmitted signals.

Devices use wires in different ways to connect to a network. The simplest connection is between two devices using a dedicated set of wires. This is called **point-to-point** communication. Sometimes many devices can share the same set of wires. This connection mode is usually referred to as a bus. In other networks, one master can connect to more than one slave using dedicated wires. This mode is called **multi-point**. Multi-drop refers to the number of devices connected to a bus, while multi-point refers to the number of one-to-one connections present in a network. The following figures show examples of these configurations.



I^{2}C

I^{2}C is a popular master-slave, multi-drop communication protocol used to exchange information between devices over very short distances, typically on the same printed circuit board. One of the attractive features of this bus is the low number of wires needed to interconnect devices. Only two lines are involved: the Serial Data line (SDA) and the Serial Clock line (SCL).

Note: You will find I^{2}C also written as I2C and also IIC. It is referred to as I squared C or I two C or I-I-C. It all comes from 'Inter-Integrated Circuit' as in communication between two integrated circuit boards.

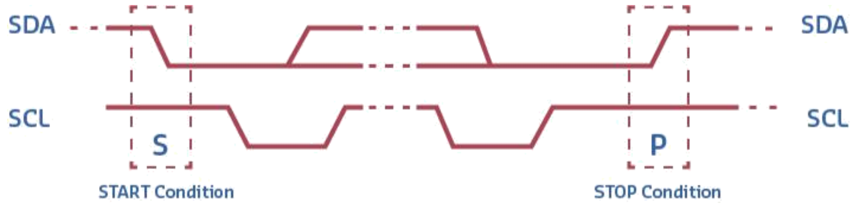
In I^{2}C communication, each device takes a designated role: either a **master**, or a **slave**. The master device pulses the clock for each transmitted bit, and the slave has to follow this speed. It is possible for a slave to hold down the clock line to pause communication until it is ready to continue. This mechanism is referred to as **clock stretching** and ensures that a slave device has the ability to protect itself from being overwhelmed.

Handshake exchange and timing diagram

I^{2}C defines the following basic types of messages, each of which begins with a START and ends with a

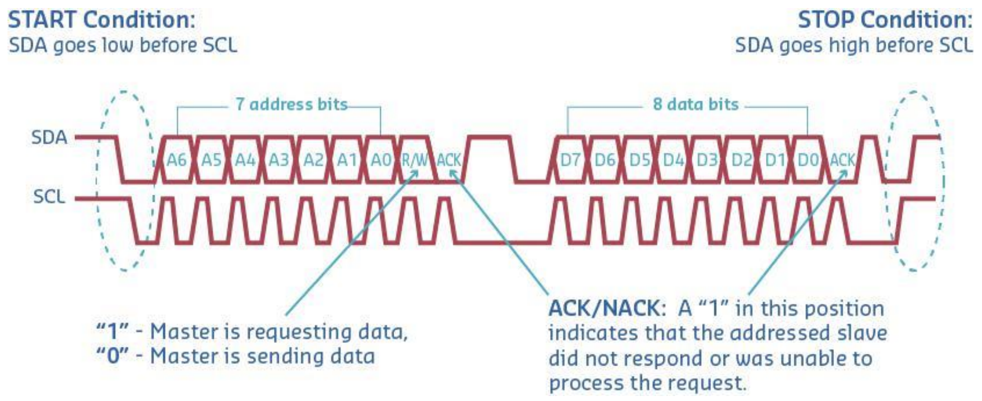
STOP:

- Single message where a master writes data to a slave; Single message where a master reads data from a slave;
- Combined messages, where a master issues at least two reads and/or writes to one or more slaves.
  - A HIGH to LOW transition on the SDA line while SCL is HIGH indicates a START condition. This is always initiated by the Master.
  - A LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition. See the following timing diagram.



In I^{2}C information is transmitted one byte (8 bits) at a time, each followed by an acknowledge bit. If a slave is busy, it holds SCL LOW to force the master into a wait state. Data transfer continues when the slave releases SCL.

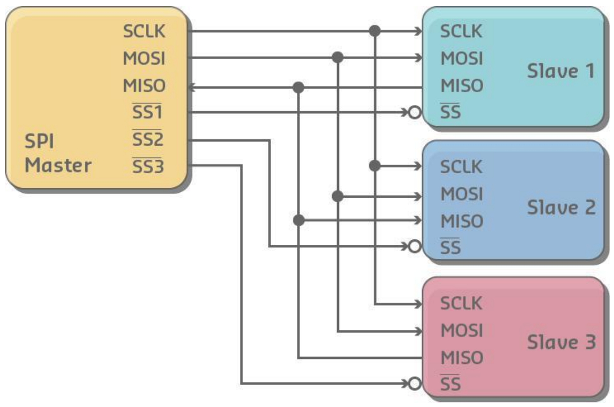
Devices are identified with a 7-bit address, therefore, at least theoretically, up to 128 devices can be connected on the same bus. In practice however, some addresses are reserved for special purposes and only 112 are available. The following timing diagram shows the transmission of one byte.


START Condition:
SDA goes low before SCL
STOP Condition:
SDA goes high before SCL
7 address bits
8 data bits
SDA
SCL
A6 A5 A4 A3 A2 A1 A0 R/W ACK
D7 D6 D5 D4 D3 D2 D1 D0 ACK
"1" - Master is requesting data,
"0" - Master is sending data
ACK/NACK: A "1" in this position indicates that the addressed slave did not respond or was unable to process the request.

The Serial Peripheral Interface (SPI)

SPI is a synchronous, master-slave, full-duplex interface that is very popular for communicating between devices over short distances. In SPI, there is no specification for the protocol used between devices, so that message sizes and data rates can vary as required. Acknowledgment of transmission by the receiver is not required either, but can be built in if it is useful.

SPI requires only four physical wires to communicate between two devices. One will act as the 'master' and one as the 'slave'. For each extra slave added to the bus, only one extra, separate connection at the master is needed, as shown in the following figure.

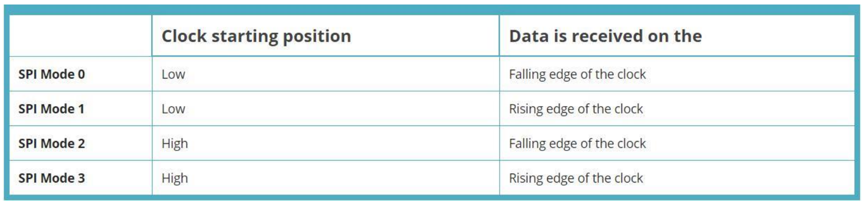


The lines involved in the SPI interface are:

- SCLK - Serial Clock. This is the clock signal being sent by the master to all slaves.
- MOSI - Master Out, Slave In. This line is for sending data from master to a slave.
- MISO - Master In, Slave Out. This line is for sending data from a slave to the master.
- SS - Slave Select. There is a separate SS line between a master and each slave.

Clock mechanism

SPI is so versatile that every combination of the clock with the data lines can be adopted. These are called the SPI modes, and are shown in the following table.



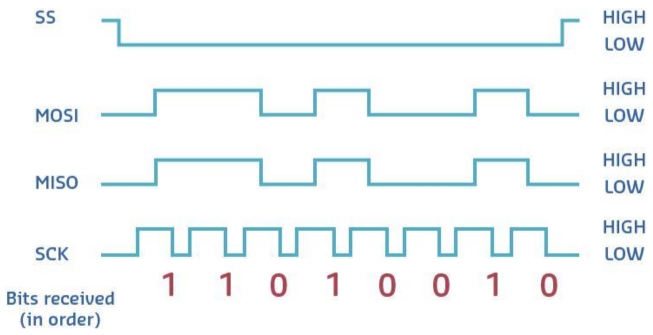
	Clock starting position	Data is received on the
SPI Mode 0	Low	Falling edge of the clock
SPI Mode 1	Low	Rising edge of the clock
SPI Mode 2	High	Falling edge of the clock
SPI Mode 3	High	Rising edge of the clock

Devices will use the moment of either the rising or falling edge (low-to-high or high-to-low) of the clock line to read data, and use the opposite edge to make any changes to data on the line they are sending.

Handshake exchange and timing diagram

- SS - Slave Select. There is a separate SS line between a master and each slave. These lines are normally held high but are lowered to indicate when a data line should be written to or read from. Some manufacturers call this line Chip Select (CS)
- MOSI - Master Out, Slave In. This line is for sending data from master to a slave. It's held high or low by the master before a clock edge to represent a '1' or '0'.
- MISO - Master In, Slave Out. This line is for sending data from a slave to the master. It's held high or low by the slave before a clock edge to represent a '1' or '0'.
- SCLK - Serial Clock. This is the clock signal being sent by the master to any slaves.

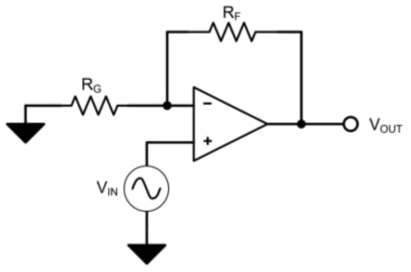
The following timing diagram shows the sequence of signals involved in the transmission of 8 bits using the SPI mode '0'.



1-Programmable Gain Instrumentation Amplifier

Transducers have a very wide range of output voltages. High gain is needed for a small sensor voltage, but with a large output, a high gain will cause the amplifier or ADC to saturate. So, some type of predictably controllable gain device is needed. Amplifiers with programmable gain have a variety of applications. Such a device has a gain that is controlled by a dc voltage or, more commonly, a digital input. This device is known as a *variable gain amplifier (VGA), or programmable gain amplifier (PGA)*.

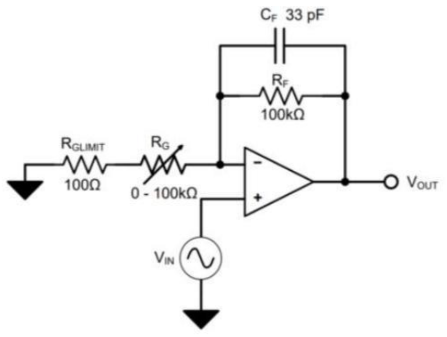
A standard non-inverting amplifier is created from an op amp, a feedback resistor, RF, and an input, or gain-setting resistor, RG, as shown in the following Figure:



The transfer function for this standard op amp building block is shown in the following Equation:

$$V_{OUT} = V_{IN} * \left(1 + \frac{R_F}{R_G}\right)$$

To turn the non-inverting amplifier into a digitally controlled programmable gain amplifier, either the RF or RG resistance must be variable based on a digital control signal. In the following figure, the circuit and transfer function displayed in this design. RGLIMIT is included in the circuit to set the maximum gain in the circuit, preventing an unbounded gain condition as the variable RG resistance approaches 0 Ω.



$$V_{OUT} = V_{IN} * \left(1 + \frac{R_F}{R_{GLIMIT} + R_G}\right)$$

For the non-inverting topology, it is advantageous to control the RG resistance for two reasons. First, with a fixed RF resistance the feedback network resistance remains constant. Therefore, the output current delivered to the feedback network doesn't change with gain and RF can be configured based on the circuit's current consumption and noise requirements. Second, if a bandwidth limiting capacitor, CF, is placed into the circuit to limit the bandwidth, the cutoff frequency, f(-3dB), won't vary as the gain changes. The cutoff frequency equation is shown in the following Equation.

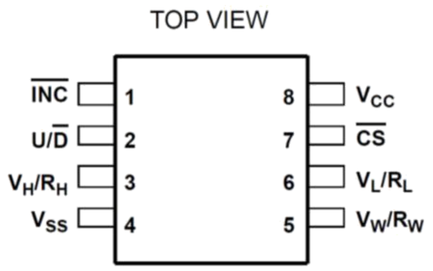
$$f_{(-3dB)} = \frac{1}{2\pi R_F C_F}$$

It is important to note that because of the non-inverting topology, the filtering effect will reduce the gain of the circuit down to 1V/V, but will not create a true single-pole filtering effect as created with a feedback capacitor in the inverting topology. This is because at high frequencies the CF capacitor will short out the RF resistance resulting in a feedback impedance, ZF, near 0 **Ω**. However, based on the transfer function for a non-inverting amplifier the gain will never decrease below 1V/V.

$$\text{Gain} = \frac{V_{OUT}}{V_{IN}} = \left(1 + \frac{R_F}{R_{GLIMIT} + R_G}\right) = \left(1 + \frac{0}{R_{GLIMIT} + R_G}\right) = 1V/V$$

Digitally Controlled Potentiometer

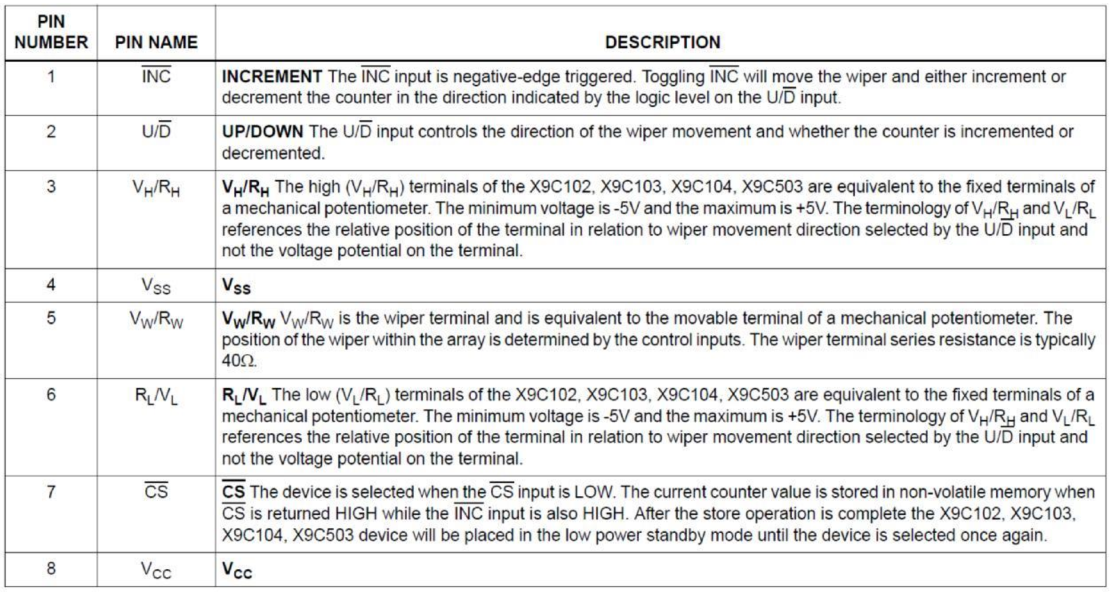
<u>Example: X9C104 = 100k Ω</u>



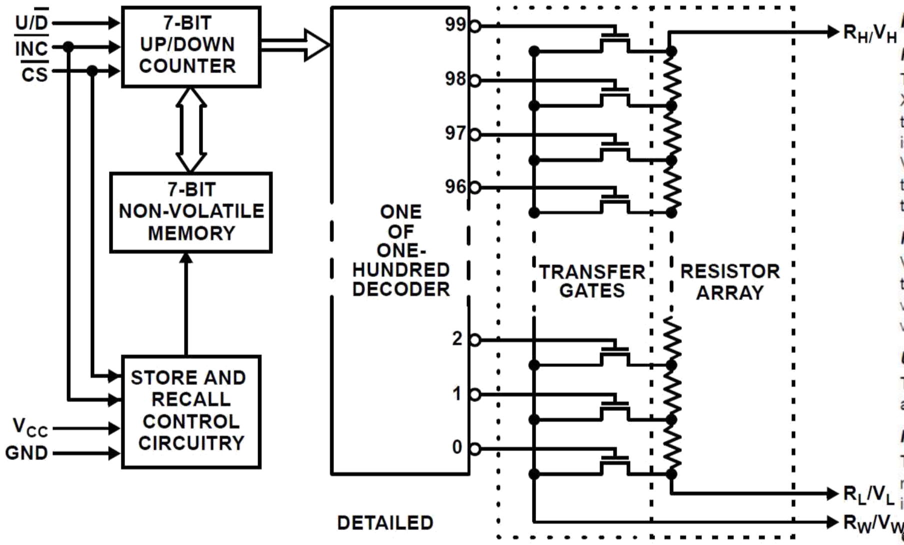
The X9C102, X9C103, X9C104, X9C503 are Intersils' digitally controlled (XDCP) potentiometers. The device consists of a resistor array, wiper switches, a control section, and non-volatile memory. The wiper position is controlled by a three-wire interface.

The potentiometer is implemented by a resistor array composed of 99 resistive elements and a wiper switching network. Between each element and at either end are tap points accessible to the wiper terminal. The position of the wiper element is controlled by the CS, U/D, and INC inputs. The position of the wiper can be stored in non-volatile memory and then be recalled upon a subsequent power-up operation. The device can be used as a three-terminal potentiometer or as a two-terminal variable resistor in a wide variety of applications ranging from control to signal processing to parameter adjustment.

Block Diagram



PIN NUMBER	PIN NAME	DESCRIPTION
1	$\overline{INC}$	**INCREMENT** The $\overline{INC}$ input is negative-edge triggered. Toggling $\overline{INC}$ will move the wiper and either increment or decrement the counter in the direction indicated by the logic level on the $U/\overline{D}$ input.
2	$U/\overline{D}$	**UP/DOWN** The $U/\overline{D}$ input controls the direction of the wiper movement and whether the counter is incremented or decremented.
3	V_H/R_H	**V_H/R_H** The high (V_H/R_H) terminals of the X9C102, X9C103, X9C104, X9C503 are equivalent to the fixed terminals of a mechanical potentiometer. The minimum voltage is -5V and the maximum is +5V. The terminology of V_H/R_H and V_L/R_L references the relative position of the terminal in relation to wiper movement direction selected by the $U/\overline{D}$ input and not the voltage potential on the terminal.
4	V_{SS}	**V_{SS}**
5	V_W/R_W	**V_W/R_W** V_W/R_W is the wiper terminal and is equivalent to the movable terminal of a mechanical potentiometer. The position of the wiper within the array is determined by the control inputs. The wiper terminal series resistance is typically 40Ω.
6	R_L/V_L	**R_L/V_L** The low (V_L/R_L) terminals of the X9C102, X9C103, X9C104, X9C503 are equivalent to the fixed terminals of a mechanical potentiometer. The minimum voltage is -5V and the maximum is +5V. The terminology of V_H/R_H and V_L/R_L references the relative position of the terminal in relation to wiper movement direction selected by the $U/\overline{D}$ input and not the voltage potential on the terminal.
7	$\overline{CS}$	**$\overline{CS}$** The device is selected when the $\overline{CS}$ input is LOW. The current counter value is stored in non-volatile memory when $\overline{CS}$ is returned HIGH while the $\overline{INC}$ input is also HIGH. After the store operation is complete the X9C102, X9C103, X9C104, X9C503 device will be placed in the low power standby mode until the device is selected once again.
8	V_{CC}	**V_{CC}**



Pin Descriptions

R_H/V_H and R_L/V_L

The high (V_H/R_H) and low (V_L/R_L) terminals of the ISLX9C102, X9C103, X9C104, X9C503 are equivalent to the fixed terminals of a mechanical potentiometer. The minimum voltage is -5V and the maximum is +5V. The terminology of V_H/R_H and V_L/R_L references the relative position of the terminal in relation to wiper movement direction selected by the $U/\overline{D}$ input and not the voltage potential on the terminal.

R_W/V_W

V_W/R_W is the wiper terminal, and is equivalent to the movable terminal of a mechanical potentiometer. The position of the wiper within the array is determined by the control inputs. The wiper terminal series resistance is typically 40Ω.

Up/Down ($U/\overline{D}$)

The $U/\overline{D}$ input controls the direction of the wiper movement and whether the counter is incremented or decremented.

Increment ($\overline{INC}$)

The $\overline{INC}$ input is negative-edge triggered. Toggling $\overline{INC}$ will move the wiper and either increment or decrement the counter in the direction indicated by the logic level on the $U/\overline{D}$ input.

Chip Select ($\overline{CS}$)

The device is selected when the $\overline{CS}$ input is LOW. The current counter value is stored in non-volatile memory when $\overline{CS}$ is returned HIGH while the $\overline{INC}$ input is also HIGH. After the store operation is complete the ISLX9C102, X9C103, X9C104, X9C503 device will be placed in the low power standby mode until the device is selected once again.

2- Thermocouple

Thermocouples have no electronics inside them, they are simply made by welding together two metal wires. Because of a physical effect of two joined metals, there is a slight but measurable voltage across the wires that increases with temperature. The type of metals used affect the voltage range, cost and sensitivity, which is why we have a few different kinds of thermocouples. The main improvement of using a thermocouple over a semiconductor sensor or thermistor is that the temperature range is very much increased.

Thermocouples are often used in HVAC systems, heaters and boilers, kilns, etc. There are a few different kinds but, in this experiment, we will discuss K type, which are very common and easier to interface with.

One difficulty in using them is that the voltage to be measured is very small, with changes of about 50 µV per ^{o}C (a µV is 1/1000000 Volts). While it is possible to read these voltages using a clean power supply and nice op-amps, there are other complications such as a non-linear response (it's not always 50µV/°C) and cold-temperature compensation (the effect measured is only a differential and there must be a reference, just as ground is a reference for voltage). For that reason, we suggest only using an interface chip that will do the heavy lifting for you, allow you to easily integrate the sensor without as much pain. In this experiment we will use a MAX6675 K-thermocouple interface chip which doesn't even require an ADC, spitting out a nice digital data signal of the temperature.

Example: Cold-Junction-Compensated K-Thermocouple to-Digital Converter (0°C to +1024°C) MAX6675

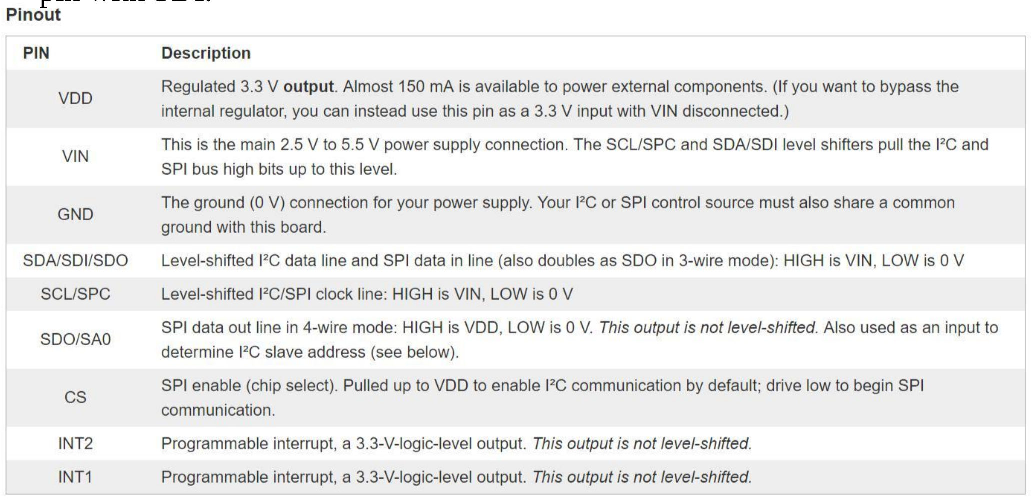
The MAX6675 performs cold-junction compensation and digitizes the signal from a type-K thermocouple. The data is output in a 12-bit resolution, SPI™-compatible, read-only format. This converter resolves temperatures to 0.25°C, allows readings as high as +1024°C, and exhibits thermocouple accuracy of 8LSBs for temperatures ranging from 0°C to +700°C.

3- Accelerometer

Accelerometers are devices that measure acceleration, which is the rate of change of the velocity of an object. They measure in meters per second squared (m/s2) or in G-forces (g). A single G-force for us here on planet Earth is equivalent to 9.8 m/s2, but this does vary slightly with elevation (and will be a different value on different planets due to variations in gravitational pull). Accelerometers are useful for sensing vibrations in systems or for orientation applications. Accelerometers can measure acceleration on one, two, or three axes. 3-axis units are becoming more common as the cost of development for them decreases. Generally, accelerometers contain capacitive plates internally. Some of these are fixed, while others are attached to minuscule springs that move internally as acceleration forces act upon the sensor. As these plates move in relation to each other, the capacitance between them changes. From these changes in capacitance, the acceleration can be determined. Other accelerometers can be centered on piezoelectric materials. These tiny crystal structures output electrical charge when placed under mechanical stress (e.g. acceleration).

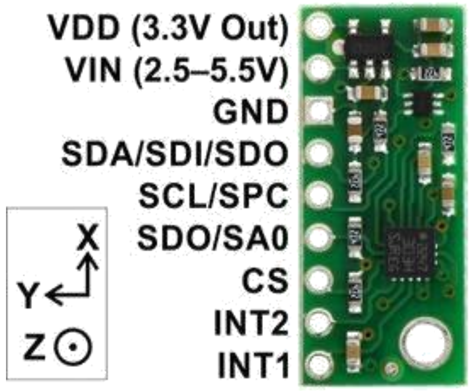
Example: LSM303D

The LSM303D combines a digital 3-axis accelerometer and 3-axis magnetometer into a single package that is ideal for making a tilt-compensated compass. The six independent readings, whose sensitivities can be set in the ranges of ±2 to ±16 g and ±2 to ±12 gauss, are available through I²C and SPI interfaces. This LSM303 carrier board includes a 3.3 V voltage regulator and integrated level shifters that allows operation from 2.5 to 5.5 To use the LSM303D in SPI mode, four logic connections are typically used: SPC, SDI, SDO, and CS. These should be connected to an SPI bus operating at the same logic level as VIN. The SPI interface operates in 4-wire mode by default, with SDI and SDO on separate pins, but it can be configured to use 3-wire mode so that SDO shares a pin with SDI.



Pinout

PIN	Description
VDD	Regulated 3.3 V **output**. Almost 150 mA is available to power external components. (If you want to bypass the internal regulator, you can instead use this pin as a 3.3 V input with VIN disconnected.)
VIN	This is the main 2.5 V to 5.5 V power supply connection. The SCL/SPC and SDA/SDI level shifters pull the I²C and SPI bus high bits up to this level.
GND	The ground (0 V) connection for your power supply. Your I²C or SPI control source must also share a common ground with this board.
SDA/SDI/SDO	Level-shifted I²C data line and SPI data in line (also doubles as SDO in 3-wire mode): HIGH is VIN, LOW is 0 V
SCL/SPC	Level-shifted I²C/SPI clock line: HIGH is VIN, LOW is 0 V
SDO/SA0	SPI data out line in 4-wire mode: HIGH is VDD, LOW is 0 V. *This output is not level-shifted.* Also used as an input to determine I²C slave address (see below).
CS	SPI enable (chip select). Pulled up to VDD to enable I²C communication by default; drive low to begin SPI communication.
INT2	Programmable interrupt, a 3.3-V-logic-level output. *This output is not level-shifted.*
INT1	Programmable interrupt, a 3.3-V-logic-level output. *This output is not level-shifted.*



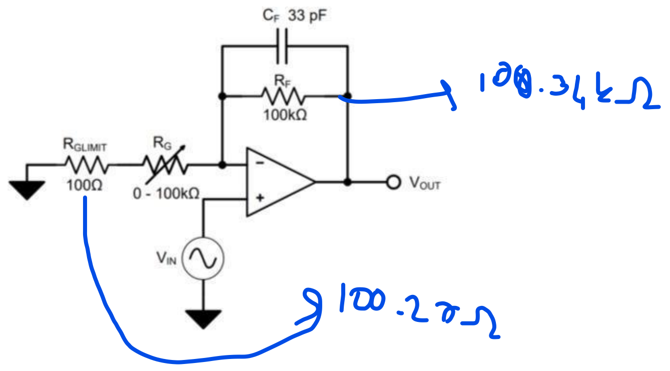
Equipment

- LM741 Op-Amp
- Function Generator
- Oscilloscope
- Arduino UNO
- Mechanical Potentiometer of 100 KΩ
- X9C104 digital potentiometer module
- MAX6675 temperature sensor module
- LSM303D accelerometer module
- Candle
- Liquid Crystal Display LCD with I^2C Serial Interface Board Module

Procedure

1. Design of Programmable Controllable-Gain Amplifier Circuit

1) Build a controllable gain amplifier circuit shown below using a mechanical potentiometer of 100 KΩ. Use the input signal from your function generator and observe the output using your oscilloscope.



- Observe the change of the output signal before and after connecting the feedback capacitor CF. Verify your explanation.

Observe the change in the gain while changing the resistor value and record your results gain vs. RG

- Compare your results with the theoretical vales of the gain vs. RG

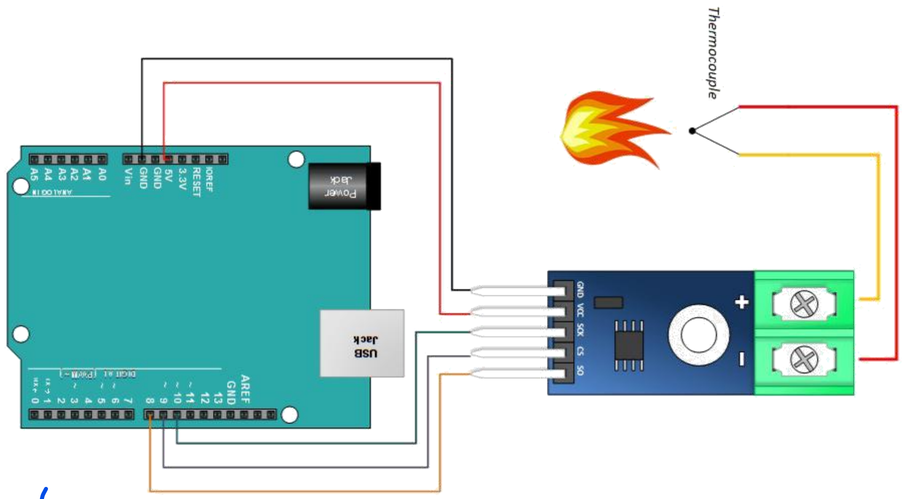
$$V_{OUT} = V_{IN} * \left(1 + \frac{R_F}{R_{GLIMIT} + R_G}\right)$$

- Observe any change while change the input signal frequency

2) Use X9C104 digital potentiometer and program it using Arduino platform and the FastX9CXXX.h library. Write a program that can change the resistor value
3) Repeat part one in the experiment but using your X9C104 digital potentiometer

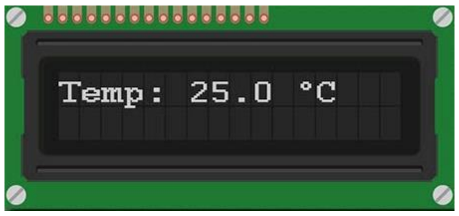
2. Design of Digital Thermometer Using a Thermocouple Sensor

1) Use the Arduino max6675.h library to read the temperature measured by the thermocouple MAX6675. Use a candle or a lighter to observe the change in the temperature readings.



What can be done with thermocouple

- By using Liquid Crystal Display LCD with I^{2}C Serial Interface Board Module, show the measured temperature on the screen



- Add the program you wrote to your report and discuss your results found
- Discuss the applications that can be done using this thermocouple
-

3. Measurement of Mechanical Vibration using 3D Digital Accelerometer

Use the Arduino LSM303.h library to read the acceleration in all x, y, and z directions using LSM303D accelerometer.

EXP.5 Characterization of a Thermistor

Preliminary Work

- Explain briefly what a thermistor is.
- Explain what β is.
- Explain the difference between NTC and PTC thermistors.
- Explain the Steinhart and Hart equation.
- Explain what an energy gap is.
- Explain what ”sensitivity” is.
- Give examples on where thermistors can be used.
- **Writing the codes before the experiment and handing them in with the preliminary report will get you 10 bonus points!**

Purpose

To effectively characterize a thermistor and find a use for it.

Equipment

- Arduino
- MAX6675 K-type Thermocouple with module (or a mercury thermometer)
- Unknown Thermistor
- Breadboard
- Jumper wires
- Personal Computer
- Various resistors

Theory

A thermistor is, etymologically, the combination of **therm**ally sensitive res**istor**. It is essentially a temperature dependent resistor. It is generally a match-head sized epoxy-coated element with two electrical leads (or wires) coming out of it. The epoxy is a protective coating to the sensing element inside of it. They are normally made out of oxides of manganese, nickel, copper and/or cobalt. As a result, the material is technically a semiconductor with an effective energy gap E_g.
Resistance is proportional to the reciprocal of conductivity, therefore;

$$R \propto e^{\frac{E_g}{2kT}}$$

Where E_g is the effective energy gap, k is the Boltzmann’s constant (in EV/K unit!), T is the temperature (in Kelvin!). If we define

$$\beta \simeq \frac{E_g}{2k}$$

Then, we can say that

$$R = Ae^{\frac{\beta}{T}}$$

For a temperature T_0, we can say that the resistance is R_0. Then,

$$R_0 = Ae^{\frac{\beta}{T_0}}$$

So;

$$R = R_0 e^{(\frac{\beta}{T} - \frac{\beta}{T_0})}$$

$$ln(R) = \frac{\beta}{T} + \left[ln(R_0) - \frac{\beta}{T_0}\right]$$

The slope of R(T) can be defined as

$$\frac{dR}{dT} = \left[R_0 e^{\beta\left(\frac{1}{T} - \frac{1}{T_0}\right)}\right] \left(\frac{-\beta}{T^2}\right.$$

As a general measure of sensitivity of resistance to temperature, we can say;

$$\alpha = \frac{1}{R}\frac{dR}{dT}$$

Steinhart and Hart equation has been determined to be the best mathematical expression for the relationship between temperature and resistance for NTC thermistors. The most common form of it is;

$$T = \frac{1}{A + Bln(R) + Cln(R)^3}$$

Where *A*, *B* and *C* are coefficients derived as;

$$A = \frac{1}{T_1} - ln(R_1)[B + C(ln(R_1)^2)]$$

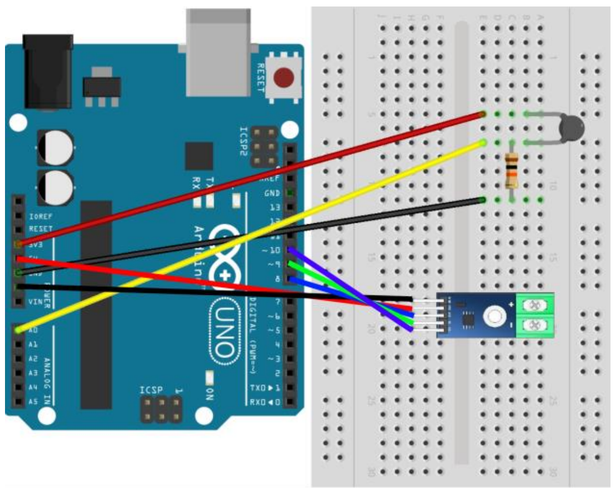
$$B = \frac{1}{T_2} - C[ln(R_1)^2 + ln(R_1)ln(R_2) + ln(R_2)^2]$$

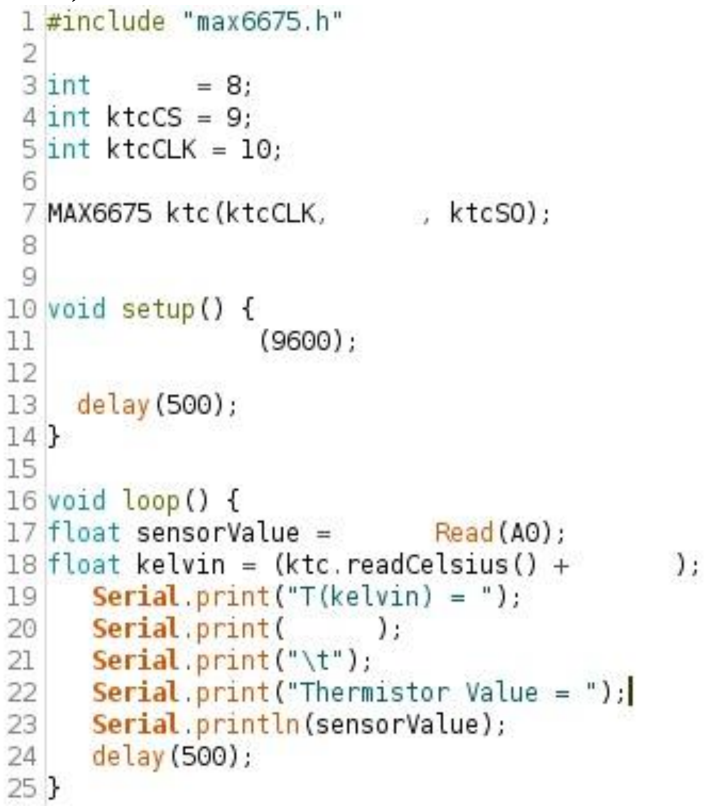
$$C = \left(\frac{\frac{1}{T_3} - \frac{1}{T_2}}{ln(R_3) - ln(R_2)}\right)(ln(R_1) + ln(R_2) + ln(R_3))^{-1}$$

Procedure

Circuitry & Code

Construct the circuit shown below, then write & execute the following code. Fill in the code's blanks! (HINT: Choose the resistor the same as your thermistor value!)





```
#include "max6675.h"

int        = 8;
int ktcCS = 9;
int ktcCLK = 10;

MAX6675 ktc(ktcCLK,      , ktcSO);


void setup() {
                 (9600);

  delay(500);
}

void loop() {
float sensorValue =        Read(A0);
float kelvin = (ktc.readCelsius() +       );
   Serial.print("T(kelvin) = ");
   Serial.print(      );
   Serial.print("\t");
   Serial.print("Thermistor Value = ");
   Serial.println(sensorValue);
   delay(500);
}
```

Data

	Temperature	Resistance
Cold		
Room		
Hot		

	β
Cold	
Hot	
$\Delta\beta$	

	Hot	Cold
e_g		
A		
$\frac{dR}{dT}$		

Sensitivity	
Cold	
Room	-
Hot	

	A	B	C	$T_{expected}$	% Error
Cold					
Room					
Hot					

1. Measure temperature & resistance for three different temperatures by putting both the thermocouple and the thermistor inside the said medium. They should be at least $10^{\circ}C$ away. Once both sensors are in the medium, wait for approximately 5 minutes for the sensors to adjust. Resistance is calculated as;

$$V_{out} = V_{in}\frac{R}{R+10k}$$

2. Calculate β for both Room-Cold and Room-Hot couplings. Then, find the average β value.
3. Calculate the energy gap.
4. Calculate the coefficient, A. (equation 1.3)
5. Calculate the derivative of R(T).
6. Calculate sensitivity for Cold and Hot temperatures.
7. Plot ln(R) vs 1000/T graph and find the slope.
8. Use the resistance values you measured to find A, B, C and the expected value of temperature. (Steinhart-Hart equation)

Discussion

1. Explain why we use a resistor in the circuitry.
2. Explain what kind of output we get as the Thermistor Value.
3. Explain what each line of code does in comments.
4. How sensitive is your thermistor? What is the reason behind the sensitivity change?
5. How well does your calculated β value agree with the slope you found in part 7? Why?
6. How good of a thermometer would this thermistor make? Calculate the percent error.

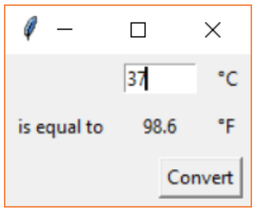
$$(\%Error = \left|\frac{\#exp-\#curve}{\#curve}\right| \cdot 100)$$

7. Why did we use both a thermistor and a thermocouple?
8. What were the reasons for errors in this experiment? Explain.
9. What did you learn from this experiment? Explain in a few sentences.

EXP.6 User Interfaces

Preliminary Work

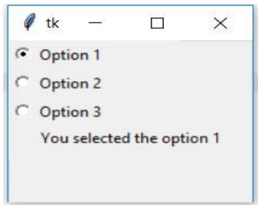
- Read the Documentation on Tkinter. https://docs.python.org/3/library/tk.html
- After installing tkinter package in python, **write a program that converts Celsius temperature to Fahrenheit by getting the user input** using the Tkinter Entry class.
  Your program should look something like this:



37 °C

is equal to 98.6 °F

- Write a program using Tkinter package that choose between different Radiobuttons and show which option is being chosen using Tkinter Label class.



You selected the option 1

- Repeat the previous program but choose Buttons instead of Radiobuttons
- Calculate the current limiting resistor for a red LED.
- Explain in details the working principle of the H-Bridge circuit and how to operate it to control

the speed and direction of rotation of a DC motor.

- Explore the L298N Motor driver shield and then write a program in Arduino platform that can control a DC motor speed and direction.
- Recommended: to learn more about Tkinter user interfaces
  http://msdl.cs.mcgill.ca/presentations/02.02.Tkinter/python-intro.pdf
- **Writing the codes before the experiment and handing them in with the preliminary report will get you 10 bonus points!**

Purpose

To build a graphical user interface that can send and receive data or commands from and to Arduino by using pyserial, tkinter and Serial Command libraries. And to build an interface that can control the speed level and direction of a DC motor.

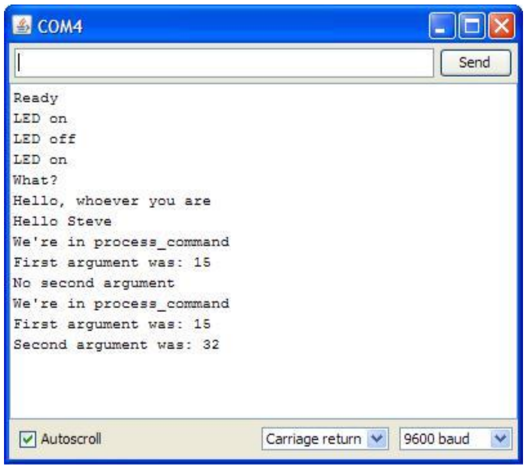
Theory

User interfaces are what allows end users to interact with an application. An application can be excellent, but without a good user interface, it becomes more difficult to use, and less enjoyable. It is thus very important to design good user interfaces. Designing user interface takes place at two different levels: the graphical level and the event level. Graphical elements of a user interface are called widgets. Widgets are basic components like buttons, scrollbars, etc. But user interfaces involve more than a collection of widgets placed in a window. The application must be able to respond to mouse clicks, keyboard actions or system events such as minimizing the window. For this to happen, events must be associated to some pieces of code. This process is called binding

SerialCommand Library on Arduino

It is to be used for interacting with a host computer to do certain tasks. You can send one argument or several arguments on the serial monitor window. The Arduino boards have a very pleasant, very easy to use USB-to-Serial connection on them, so you just have to write your host program to send serial bytes and you can communicate with the Arduino board. Frequently what I end up needing is the Arduino to respond to some commands sent from the host computer. Things like "turn off that relay" or "process this" or "move this motor to this position and stop" or "holy s**t the building is burning down, everybody run!" All the commands become "number, something, something …" and the number depends on what order you've added the handlers, so you have to do your own bookkeeping to know which number belongs to the enum of the command handlers.

A SerialCommand object maintains a char[] buffer, that fills with characters from the Serial() stream. When the terminator character is received (in this case, the carriage return, or '\r'), it then scans the buffer looking for the first word in the buffer to be the command. The buffer is of fixed size (configurable), and just wraps around if you overwrite the end, so it doesn't crash but might give unintended results if you try to receive a command larger than the buffer. There is a .clearBuffer() routine, which you can use to zero out the buffer to be sure. The buffer also zeros out when a command gets recognized and processed. In the handler, you can call the .next() function to parse out the next token. It comes out as a pointer to a char string, so you can use atoi() or whatever to convert to a numeric format or do further parsing. You can keep calling next() until it returns NULL, meaning no more arguments.



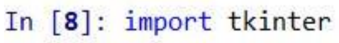
```
Ready
LED on
LED off
LED on
What?
Hello, whoever you are
Hello Steve
We're in process_command
First argument was: 15
No second argument
We're in process_command
First argument was: 15
Second argument was: 32
```

Tkinter Programming

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit. Creating a GUI application using tkinter is an easy task. We cannot show you everything about GUI application development in just one experiment, but we will give you a very solid introduction to it. Tk is not the latest and greatest, nor does it have the most robust set of GUI building blocks, but it is fairly simple to use, and with it, you can build GUIs that run on most platforms. All you need to do is perform the following steps;

1- Import the tkinter module.
2- Create the GUI application main window.
3- Add one or more of the above-mentioned widgets to the GUI application.
4- Enter the main event loop to take action against each event triggered by the user.

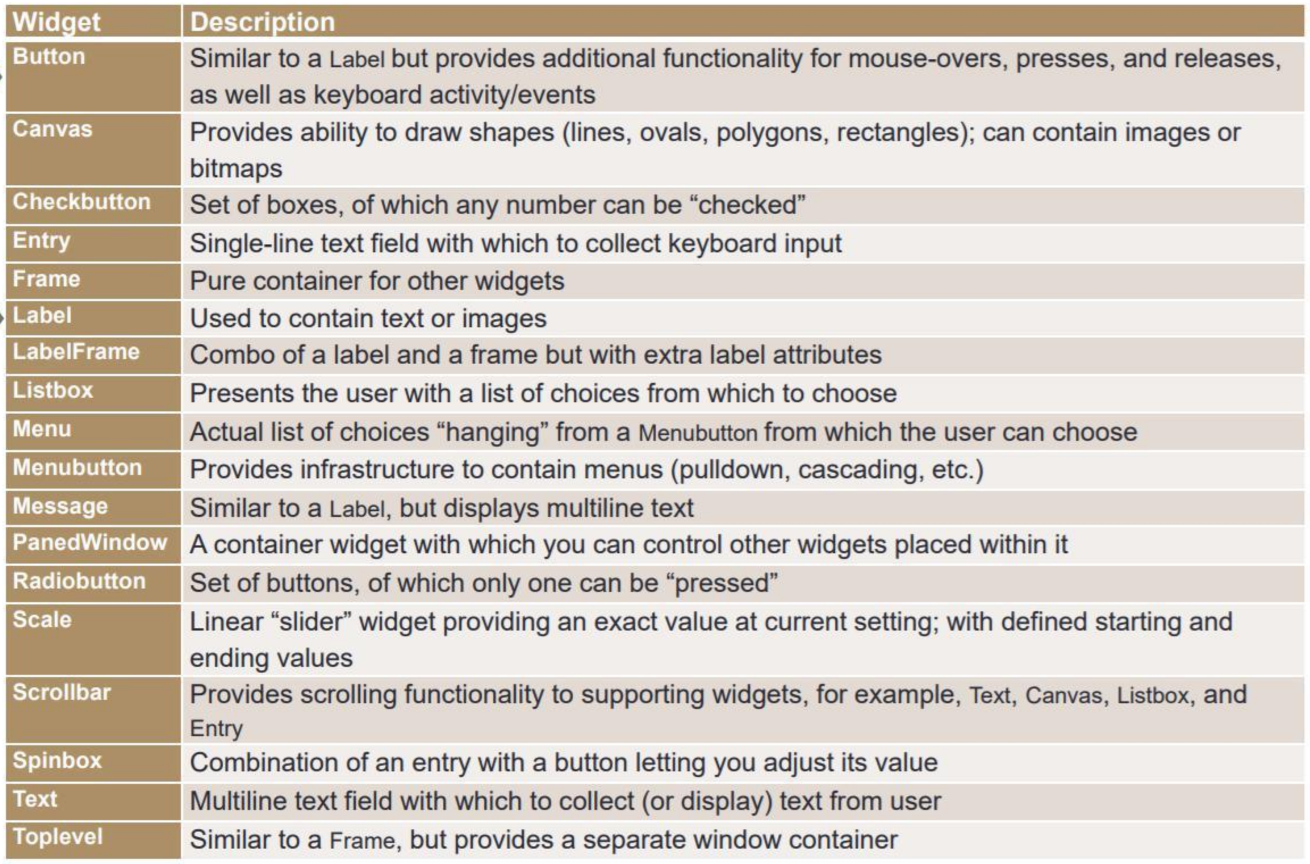
Tkinter is not necessarily turned on by default on your system. You can determine whether Tkinter is available for your Python interpreter by attempting to import the Tkinter module (in Python 1 and 2; renamed to tkinter in Python 3). If Tkinter is available, then no errors occur, as demonstrated in the following:



```
In [8]: import tkinter
```

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets. Widget is a graphical component on the screen (button, text label, drop-down menu, scroll bar, picture, etc…)

There are currently 15 types of widgets in Tkinter.



Widget	Description
Button	Similar to a Label but provides additional functionality for mouse-overs, presses, and releases, as well as keyboard activity/events
Canvas	Provides ability to draw shapes (lines, ovals, polygons, rectangles); can contain images or bitmaps
Checkbutton	Set of boxes, of which any number can be "checked"
Entry	Single-line text field with which to collect keyboard input
Frame	Pure container for other widgets
Label	Used to contain text or images
LabelFrame	Combo of a label and a frame but with extra label attributes
Listbox	Presents the user with a list of choices from which to choose
Menu	Actual list of choices "hanging" from a Menubutton from which the user can choose
Menubutton	Provides infrastructure to contain menus (pulldown, cascading, etc.)
Message	Similar to a Label, but displays multiline text
PanedWindow	A container widget with which you can control other widgets placed within it
Radiobutton	Set of buttons, of which only one can be "pressed"
Scale	Linear "slider" widget providing an exact value at current setting; with defined starting and ending values
Scrollbar	Provides scrolling functionality to supporting widgets, for example, Text, Canvas, Listbox, and Entry
Spinbox	Combination of an entry with a button letting you adjust its value
Text	Multiline text field with which to collect (or display) text from user
Toplevel	Similar to a Frame, but provides a separate window container

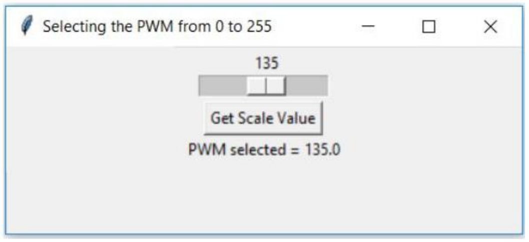
The Need for New Variables

When the Tkinter variables are used for simple tasks, they do not behave differently than the regular Python variables, apart from the fact that their values have to be handled through method calls, such as **get()** and **set()**. The choice behind the implementation of this master variable class is therefore not obvious. However, by taking a closer look at what the Variable class offers, the motivation behind it becomes much more apparent. Tkinter variables provide, in true Tkinter style, the possibility to add a callback which will be executed when read and write operations are performed on the variable, and also when the value associated with the variable is undefined. This allows, for example, to bind a **StringVar()** instance to a Label widget, which then automatically updates its caption when the value of the variable changes. Tkinter provides four descendants of the Variable class:

• StringVar
• IntVar
• DoubleVar
• BooleanVar

Example1: Given below an example on how to use tkinter to build a scale from 0 to 255. This example will help you in controlling the brightness of an LED later in this experiment.

```
from tkinter import *    #Import the Tkinter module


def sel():
   selection = "PWM selected = " + str(var.get())
   label.config(text = selection)

#to get a place where you put all your widgets
root = Tk()

#automatically updates its caption when the value of the variable changes
var = DoubleVar()

#To set the title of your window
root.title("Selecting the PWM from 0 to 255")

#To put a horizontal scale or slider with a certain limits
scale = Scale( root,orient=HORIZONTAL,from_ = 0, to= 255, variable = var )

#To arrange the geometry of the scale
scale.pack(anchor=CENTER)

#To put a button that achieves a certain function
button = Button(root, text="Get Scale Value", command=sel)
button.pack(anchor=CENTER)

#To put a label that shows the variable selection
label = Label(root)
label.pack()

#to enter the aforementioned infinite main loop
root.mainloop()
```



PWM selected = 135.0

Equipment

- Arduino
- LED
- Resistor
- PC
- DC Motor
- L298N Motor driver shield

Procedure

After installing SerialCommand Library on Arduino Platform and tkinter and pyserial libraries on Python; (https://github.com/kroimon/Arduino-SerialCommand/blob/master/SerialCommand.h)

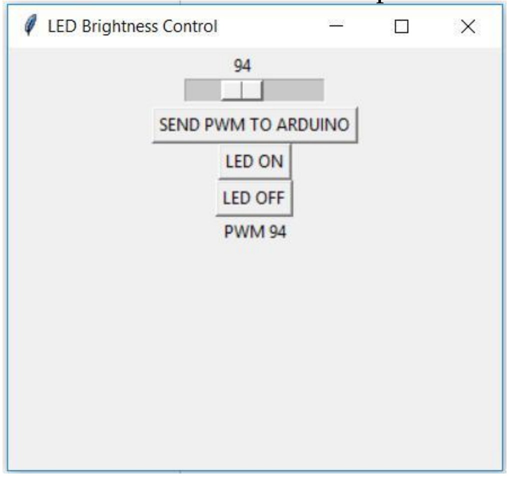
1- Design a user interface that controls the brightness of an LED by using Serial command and Pyserial libraries using Arduino

   Hint: Use Serial Command Library on Arduino platform to define each command sent from python

   Hint: You can use the Serial Command Library example "SerialCommandHardwareOnlyExample.ino" to help you figure out how to send the commands from Python

   Hint: Each command sent from Python should end with '\r'

   Hint: Make sure that your LED is connected to an Arduino pin that can provide PWM



2- Design a user interface that can control the speed and direction of rotation of a DC motor.

   Hint: Your Program should contain several speed levels, Stop, and brake buttons

   Hint: Your Program should contain two Radiobuttons to control the direction of rotation

   Hint: Your Program should contain a scale from 0 to 255 to manually control the PWM given to the DC motor