Mixed Signals Activity

You've received data from an experiment, but the signal is obscured with noise. You want to remove the noisy background to obtain your clear signal. To do this, you are going to build a potentiometer circuit, write and run code in Thonny to collect your data, and run code in Spyder to process your data. The circuit acts as a 'clean up' filter and the potentiometer allows you to vary the resistance of the component, thereby changing the output voltage. At a certain voltage, the noise in your data will be removed. You need to find the value of the resistor which gives you your clear signal.

There are three main parts to this activity:

- Building the circuit
- Writing and running the code
- Processing the data

1 Building the circuit

You will be building the circuit using a Raspberry Pi Pico, a potentiometer, a breadboard and some jumper cables. If you aren't familiar with how breadboards work, see the breadboard helpful tips box at the end of this section. The circuit you need to build can be found in Figure 1.

Figure 2 shows the pin-layout of the Raspberry Pi Pico. There are 40 pins in total, marked by the grey number beside the label. Most pins on the board are *General Purpose* pins, labelled as GP0 for example. You should notice that the GP-number does not correlate with the number in the grey box. When you are writing your code, you need to refer to a particular pin by its GP-number. If you get stuck building the circuit, step-by-step instructions can be found below.

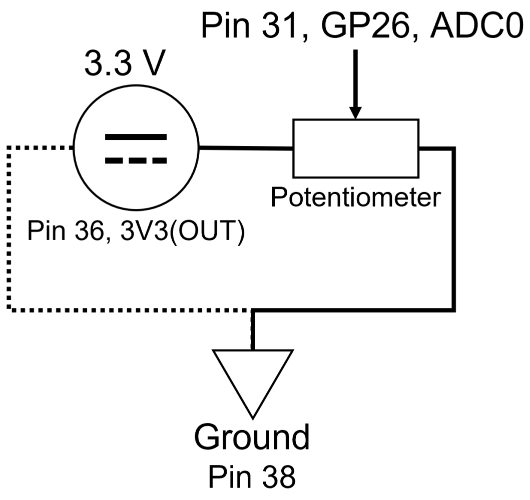

Figure 1: Potentiometer circuit diagram.

Task 1: Build the circuit according to Figure 1.

Step-by-step circuit building:

- Place the Raspberry Pi Pico into the breadboard (you may have to apply some pressure to ensure it is securely fitted).
- Connect the potentiometer to an unconnected space on the breadboard such that each of the three pins occupies a separate row.
- Connect pin 38 of the Pico (or any ground pin) to one side of the potentiometer ends.
- Connect pin 36 of the Pico to the other side of the potentiometer.
- Connect pin 31 (labelled GP26) to the middle pin of the potentiometer.
- Connect the micro-USB to the USB port on the Pico.

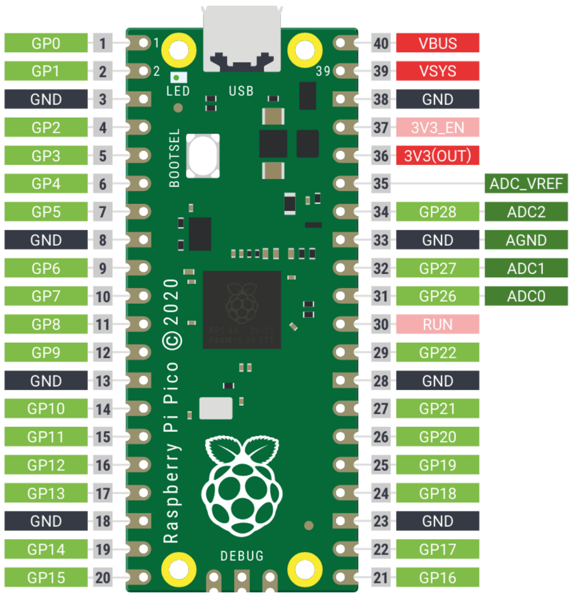

Figure 2: Raspberry Pi Pico pin layout. Each pin has a number in a grey box and an associated label. For example, pin 17 is General Purpose 13 (GP13), and pin 36 is a 3.3 V direct current output.

Pins 32 and 34 can be used instead of pin 31, however in the Thonny code, you must specify the connection by its GP-number. These three pins are unique because they are analogue-to-digital converters (ADC), which turn an analogue voltage signal into a numerical quantity - ADC counts. To convert between ADC counts and volts, the following equation is used:

$$V_{\text{out}} = V_{\text{in}} \frac{\text{ADC Counts}}{2^n - 1} \tag{1}$$

where in the Raspberry Pi Pico $V_{\text{in}} = 3.3$ V and $n = 16$. The ADC in the Pico is a 16-bit ADC meaning the voltage signal can be digitised into 65536 discrete steps.

Tips - Breadboards

Figure 3 shows a breadboard highlighting:

- All *sockets* in a *single row* are connected.
- Rows and columns are not connected, except for the '+' and '-'-rails
- The '+' rail and the '-' rail are each connected along the length of the breadboard.
- The left and right sides of the breadboard are *not* connected

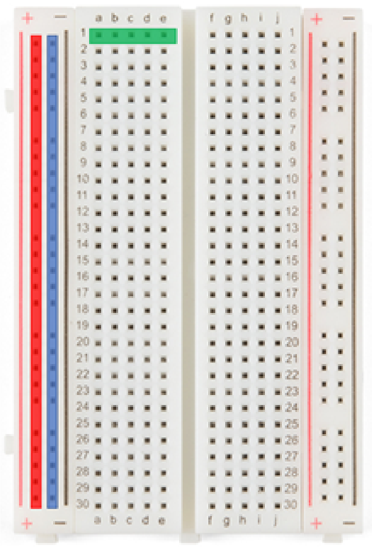

Figure 3: The red box shows all the holes in the '+' rail are connected, and the same for the blue box but the '-' rail. The green box highlights that holes a-e in row 1 are connected.

2 Writing and running the code

The code operating the Raspberry Pi Pico is written in Python, using the Thonny programme. In Python, comments can be added by using a '#'. When running the code, if a line is commented then the programme will not process whatever is written on that line as code; it is particularly useful to include comments to explain what your code is doing.

The code below will measure the voltage across your potentiometer as you adjust it.

```
#### THONNY CODE ###
# Import the 'Pin' and 'ADC' modules from the 'machine' library
from machine import Pin, ADC

#Import the utime module
import utime

#Create an empty list to store ADC values
store=[]

#Create a variable 'adc_reader' which will measure and convert the voltage of
    GP26 (pin 31)
adc_reader = ADC(Pin(26, mode=Pin.IN))

#Create variable 'start' which records the time at that moment
start=utime.time()

#Create a while loop which will run the code inside while the condition is
    true
#Here the loop will run until 30 seconds has elapsed
while (utime.time()-start)<30:

    #Measure the voltage
    count=adc_reader.read_u16()

    #Print the measured ADC
    print(count)

    #Save the ADC count to the list
    store.append(count)

    #Set how often to measure the voltage
    #Here the voltage is measured every 0.5 seconds
    utime.sleep(0.5)

#Save the list of stored files to a textfile: Output.txt
with open('Output.txt','w') as file:
    for item in store:
        file.write("%s\n" % item)
```

Task 2: Write the Thonny code in the Thonny window.

Once you have written your code, save the Python file to the Raspberry Pi Pico - do this by pressing save or Ctrl+S and then select Raspberry Pi Pico. Press the green play button to run the code. If your code is written correctly, you should see a column of numbers - these are the digitised values (ADC counts) of the voltage in the potentiometer. Now, while running the code turn the knob on the potentiometer, you should see the numbers change as your adjust the resistance.

At this point, the text file containing the stored ADC counts is saved on the Raspberry Pi Pico memory. To save the file to the computer, right-click the Output.txt item under the Files tab on the left side of the Thonny window and click "Download to ...". You should select the folder which contains your PicoFunctions.py and Spectrum.txt files.

Task 3: Run your code and save the data to the computer.

3 Data processing

To complete the data processing aspect of this activity, you will be using a different Python programme called Spyder. Spyder allows for a different selection of modules to be installed, and additionally offers graphing capabilities. A module called ‘PicoFunctions’ has been created to process your data. The code below will read in your data, and plot the results.

```
### SPYDER CODE ###
#Import the ‘PicoFunctions’ module with a shortcut name
import PicoFunctions as PF

#Call the function ‘main()’ from PicoFunctions and assign it to a variable ‘run’
#The main() function takes your Output.txt file as its only input
run=PF.main(‘Output.txt’)
```

Task 4: Write and run the Spyder code.

The desired signal will be shown in the top plot of the figure, and your noisy signal which is being filtered by your ADC values is shown in the middle plot. The bottom plot will show the percentage difference between your ADC values and the value necessary to obtain the desired signal.

If any of your values are in the acceptable range, a green region will appear which will help you narrow down your final value. Once you are sure of a final ADC value, you should use it to determine the potentiometer's resistance at this point.

The potentiometer is essentially a voltage divider circuit where:

$$V_{\text{out}} = V_{\text{in}} \frac{R_2}{R_1 + R_2} \tag{2}$$

The total resistance of the potentiometer is 10 kΩ meaning $R_1 + R_2 = 10,000$. Using Equation 1, Equation 2, and the relationship between R_1 and R_2, determine the value of R_2.

Task 5: Determine the value of the resistor necessary to clean up your signal.

Tips - Calculating the resistance
- Calculate V_{out} using your final ADC value and Equation 1
- Rearrange Equation 2 for R_2
- Substitute V_{out}, V_{in} and $(R_1 + R_2)$ into your rearranged equation
- $V_{\text{in}} = 3.3$ V
- $(R_1 + R_2)$ =10,0000