Pico Project Week three

Inputting sound

Following on from last week where we had built our own loudspeaker that played different sounds depending on which button was pressed. Our thought turned to how could we input sound into the Pico. An obvious choice is some form of microphone

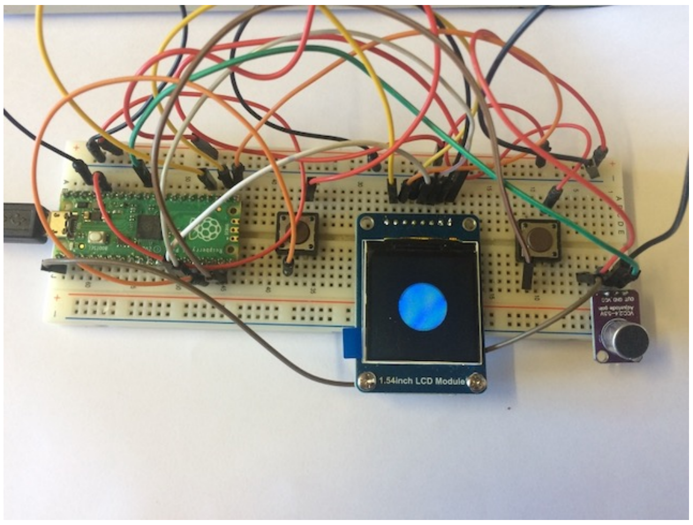
Microphone and ADC

We chose a low cost microphone with an inbuilt amplifier, the amplifier is needed to boost the tiny voltages produced by the microphone

A microphone produces an analogue signal whereas the Pico needs a digital signal to work with. The RP2040 processor has four ‘analogue to digital’ converters, three of which are available on the Pico. An analogue to digital converter, ADC, converts the analogue signal to a digital signal that can be read by the Pico. The Pico ADC’s are 12 bit, returns a value between 0 and 4095. However CircuitPython is written to work across a number of devices and returns a 16 bit value, 0 - 65535. We used GPIO 26 (GP26_A0 in CircuitPython)

Wiring

Out — GP26_A0)
VCC — 3.3V
GND — Ground



Coding

```
import board
import terminalio
import busio
import displayio
from vectorio import Rectangle, Circle
from adafruit_display_text import label
from adafruit_display_shapes import circle
from adafruit_st7789 import ST7789
import adafruit_imageload
import digitalio, analogio
import time, random
```

```
import audiomp3
import audiopwmio
```

```
mic = analogio.AnalogIn(board.GP26_A0)
```

```
# Release any resources currently in use for the displays
displayio.release_displays()
# SPI pins for display st7789
tft_cs = board.GP17
tft_dc = board.GP16
# setup spi bus
spi = busio.SPI(board.GP10, board.GP11)
# setup display
display_bus = displayio.FourWire(spi, command=tft_dc,
                                 chip_select=tft_cs, reset=board.GP18)
display = ST7789(display_bus,
                 width=240,
                 height=240,
                 rowstart=80,
                 auto_refresh=True)
# Make the display context
screen = displayio.Group()
```

```
palette = displayio.Palette(1)
palette[0] = 0x125690
```

```
circle = Circle(pixel_shader=palette, radius=25, x=120, y=120)
screen.append(circle)
display.show(screen)


buttonOne = digitalio.DigitalInOut(board.GP13)
buttonOne.switch_to_input(pull=digitalio.Pull.DOWN)
buttonTwo = digitalio.DigitalInOut(board.GP14)
buttonTwo.switch_to_input(pull=digitalio.Pull.DOWN)

def get_voltage(raw):
    return (raw * 3.3) / 65536.0

def mapFromTo(v, oldMin, oldMax, newMin, newMax):
   newV=(v-oldMin)/(oldMax-oldMin)*(newMax-newMin)+newMin
   return newV

def average(lst):
    return sum(lst) / len(lst)

oldAmp = mic.value  # holder of "old mic value" (so we can compare each loop)
buffer = 100        # a buffer to check between a range (old value - buffer |
new value | old value + buffer)
micArray = []       # keep an array of values so we can "smooth" an average ?
lowest = 10000      # set a high value for the lowest value (so we can
eventually whittle it down to the minimum value the mic spits out)
highest = 1         # set a low value for the maximum value (so we can build it
up to find the highest raw mic value)

while True:
    #get the mic value
    amp = mic.value
    #add the mic value to the micArray list
    micArray.append(amp)
    # if the micArray is larger than 10 items, pop the 1st item out
    if len(micArray) > 2:
        micArray.pop(0)

    # if the mic values are lower or higher than the previous min / max vlues -
change them accordingly (so we can get a true min/max range)
```

```
    if amp <= lowest:
        lowest = amp
    if amp >= highest:
        highest = amp
```

```
    # check to see if the new amp value is between the acceptable range of "no
change"
    if (oldAmp - buffer) <= amp <= (oldAmp + buffer):
        #print("no change, setting to convertedRad radius")
        circle.radius = convertedRad
        oldAmp = amp
    else:
        convertedRad = int(mapFromTo(average(micArray), lowest, highest-1, 2,
100))
        circle.radius = convertedRad
```

```
    #print("amp = {} & oldAmp = {} low = {} hi = {} av = {}".format(
    #    amp, oldAmp, lowest, highest, average(micArray)))
```

Conclusion

We set out what we wanted to do, connecting a microphone to the Pico, we realised however that we wanted some way to split the frequencies up and display separately, a bit like a spectrum analyser. It's a reasonably straight forward task using 'C' but not at the present time in CircuitPython. We started to look for a solution and what we found will be the basis for next weeks instalment.