Project_Progress

March 26, 2024

1 PHYS 319 Project Progress: 85888436 Harrison Li

2024 Mar 5 2:00 pm

1.1 Amplifier, Microphone circuit

Breadboard Connections: Vcc = +5V input: - mircophone : 47E to 48E - connect mic to ground : 48A to Ground - connect mic to amplifier : 47C to 30H

amplifier - connect op-amp LM386 pin(1, 2[-in], 3[+in], 4[GND], 5[V_out], 6[Vcc], 7, 8) -> (26E, 27E, 28E, 29E, 29F, 28F, 27F, 26F) - 10 kOhm resistor : 30G to 34D - 10 kOhm resistor : 33H to Ground - 1 microF capacitor : 34B to 36B - 1 microF capacitor : 34C to 36C - 1 microF capacitor : 29I to 31I - 1 microF capacitor : 31J to 33J - 10 microF capacitor : 30F to 28C - 10 microF capacitor : 27G to 22G - connect 28I to Vcc - connect 34A to Vcc - connect 22F to Ground - connect 27A to Ground - connect 29A to Ground - connect 36A to Ground

amplifier output at op-amp pin5: connect 29J to 20J

positive clamper circuit to add 2.5V offset to amplifier output - diode : 14F to Ground - 1 kOhm resistor : 14G to Ground - 10 kOhm resistor : 14I to 4I - 100 microF capacitor : 20H to 24H

final output signal at 4J

LaunchPad Connections: - Connect LanunchPad 5V to Vcc - Connect LanunchPad GND to ground - Connect LanunchPad P6.0 to 4J

1.2 C Code

[ ]:

```
#include <msp430f5529.h>
#include<stdio.h>

# define TXD  BIT4              // TXD on P4.4
# define RXD  BIT5              // RXD on P4.5
# define LED1 BIT0              // P1.0 LED
# define LED2 BIT7              // P4.7 LED
# define INPUT BIT0             // input from microphone  on P6.0

unsigned int TXByte;

int main(void) {
    WDTCTL = WDTPW + WDTHOLD;   // stop watchdog timer
    // configure ADC
    ADC12CTL0 = ADC12SHT02 + ADC12ON;   // Sampling time, ADC12 on
    ADC12CTL1 = ADC12SHP;               // sampling timer
    ADC12CTL0 |= ADC12ENC;              // ADC enable
    P6SEL |= 0x01;                      // P6.0 allow ADC on pin 6.0

    ADC12MCTL0 = ADC12INCH_0;
          //selects which input results are
          //stored in memory ADC12MEM0. Input
          //one is selected on reset so this line is not needed
          //Must be written before enabling conversions

    ADC12CTL0 |= ADC12ENC;              // ADC enable

    /* Configure hardware UART */
    UCA1CTL1 = UCSWRST;         // Recommended to place USCI in reset first
    P4SEL |= BIT4 + BIT5;
    UCA1CTL1 |= UCSSEL_2;       // Use SMCLK
    UCA1BR0 = 109;              // Set baud rate to 9600 with 1.048MHz clock␣
 ↪(Data Sheet 36.3.13)
    UCA1BR1 = 0;                // Set baud rate to 9600 with 1.048MHz clock
    UCA1MCTL = UCBRS_2;         // Modulation UCBRSx = 2
    UCA1CTL1 &= ~UCSWRST;       // Initialize USCI state machine
    /* if we were going to receive, we would also:
        IE2 |= UCA1RXIE; // Enable USCI_A1 RX interrupt
    */

    // output pin to computer
    P4DIR |= TXD;
    P4OUT |= TXD;

    // LED for function testing
//    P1DIR |= LED1;
//    P1OUT |= LED1;

    P4DIR |= LED2;
    P4OUT &= ~LED2;

    while (1) {
        ADC12CTL0 |= ADC12SC;           // Start sampling
        while (ADC12CTL1 & ADC12BUSY);  // while bit ADC12BUSY in register␣
 ↪ADC12CTL1 is high wait

        while (! (UCA1IFG & UCTXIFG)); // wait for TX buffer to be ready for␣
 ↪new data
```

```
        TXByte = ADC12MEM0/10;     // input voltage proportional to the volume␣
 ↪of the audio signal
        UCA1TXBUF = TXByte;     // Transmit TXByte;

//        P1OUT ^= LED1;          // toggle LED1
        P4OUT ^= LED2;          // toggle LED2

        _delay_cycles(10000);  // wait for 20 milliseconds before repeating
    }
}
```

1.3 Python Code to receive data and save it into a csv file with time

[ ]:

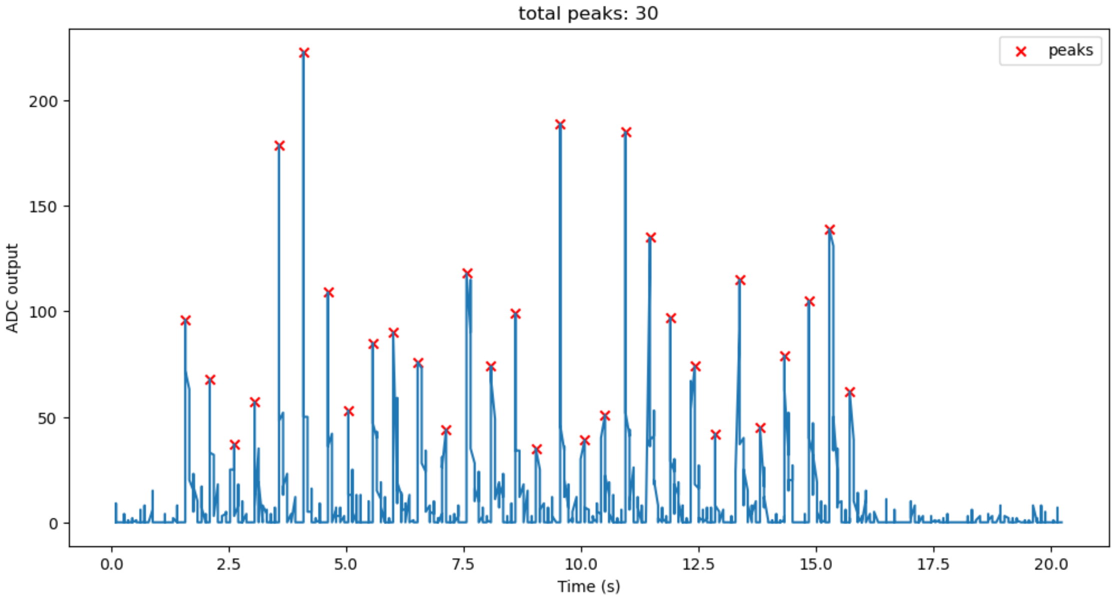
```
import matplotlib.pyplot as plt
import serial # for serial port
import numpy as np # for arrays, numerical processing
import time
import csv



#define the serial port
port = '/dev/tty.usbmodem112203'

audiodata = []
times = []
csv_savename = "audiodata3.csv"
create_plot = True  # set true to create updating plot


# try:
#     import IPython
#     shell = IPython.get_ipython()
#     shell.enable_matplotlib(gui='qt')
# except:
#     print("Need to enter '%matplotlib qt' manually in console.")
#     pass

# initialize figureç
if create_plot:
    plt.figure()
    plt.xlabel("Time (s)")
    plt.ylabel("ADC output")
    plt.title("Audio Signal ADC")
    # plt.ylim(0, 100)

try:
    with serial.Serial(port,9600,timeout = 0.1) as ser:
        print(ser.name)
        print("Flushing serial...")
        ser.flushInput()
        ser.flushOutput()
        print("Flushed")
        start_time = time.time()
        while(1): #loop forever
            data = ser.read(1)
            if len(data) > 0: #was there a byte to read?
                print(ord(data))
                # get current time and save in list
                current_time = time.time()-start_time
                times.append(current_time)
                audiodata.append(ord(data))

                # if create_plot:
                #     plt.pause(0.0001)# wait for plot to update (INCREASE IF PLOT ISN'T UPDATING PROPERLY)
                #     plt.clf()  # clear figure
                #     plt.plot(times, audiodata)



#on ctrl-c (in console), save to CSV in specified location
except KeyboardInterrupt:
    create_plot = False
    print("Collection stopped - saving to CSV...")
    startcsv = time.time()
    with open(csv_savename, "w", newline='', encoding='utf-8') as f:
        writer = csv.writer(f)
        rows = zip(times, audiodata)
        for row in rows:
            writer.writerow(row)
    endcsv = time.time()
    print("Data saved to " + csv_savename)
    print("Took " + str(endcsv - startcsv) + " s")
```

1.4 Python Code to analyze the data from saved csv file

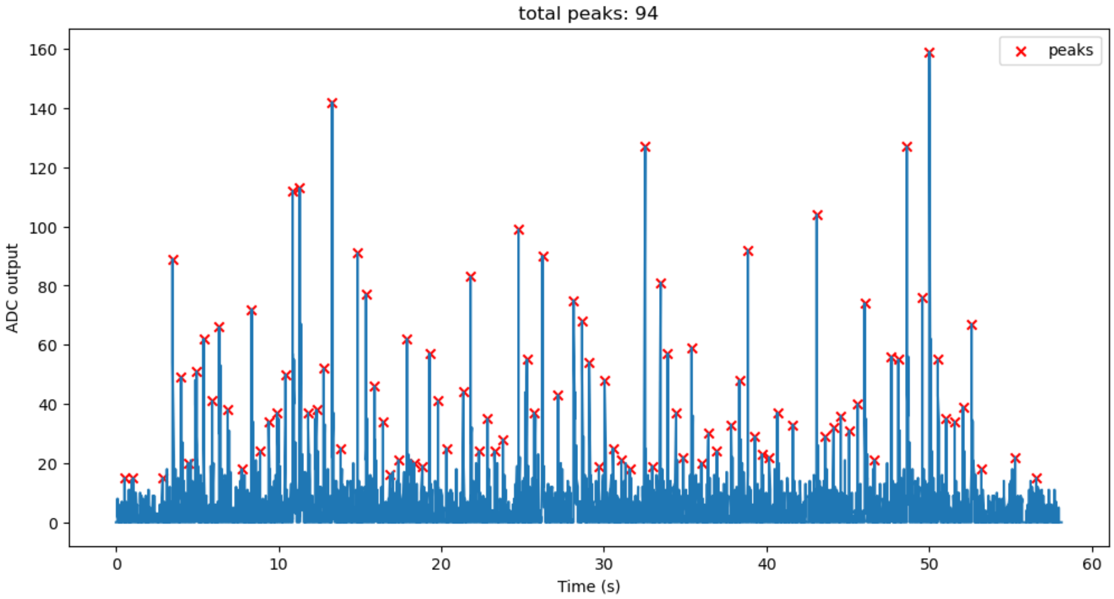
[ ]:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp

# import data
def count_peaks(filename):
    print("processing file: {}".format(filename))
    data = np.loadtxt(filename, delimiter=',')
    time = data[:, 0]
    power = data[:, 1]

    # print(np.mean(power), np.max(power))
    fs = 1/(time[1]-time[0]) # sampling frequency

    # find peaks in smoothed signal
    # peaks, props = sp.signal.find_peaks(power, distance=0.2*fs, height=np.
 ↪max(power[-500:]))
    # find peaks in noisy signal using wavelet decomposition
    # peaks = sp.signal.find_peaks_cwt(power, widths=np.arange(5, 15))
    peaks, props = sp.signal.find_peaks(power, distance=40, height=np.
 ↪mean(power)+1*np.std(power))
    print("time_arr, power_arr length", time.shape, power.shape)
    print("peak array shape", peaks.shape)
    print("fs=" ,fs)
    print("peaks idx", peaks)

    plt.figure(figsize=(12, 6))
    plt.plot(time, power)
    plt.scatter(time[peaks], power[peaks], marker='x', c='red', label="peaks")
    plt.legend()
    plt.xlabel("Time (s)")
    plt.ylabel("ADC output")
    plt.title("total peaks: {}".format(len(peaks)))
    plt.show()

    return len(peaks)

file1 = "audiodata_approx30_peaks.csv"
file2 = "audiodata_approx100_peaks.csv"
count_peaks(file1)
count_peaks(file2)
```

```
processing file: audiodata_approx30_peaks.csv
time_arr, power_arr length (2097,) (2097,)
peak array shape (30,)
fs= 53092.45569620253
peaks idx [ 160  210  264  313  363  417  472  520  571  620  674  729  782  834
  884  935  982 1035 1080 1129 1180 1227 1278 1326 1378 1430 1483 1535
 1583 1628]
```



```
processing file: audiodata_approx100_peaks.csv
time_arr, power_arr length (6040,) (6040,)
peak array shape (94,)
fs= 10.704124132298897
peaks idx [   51   101   289   355   407   458   507   552   605   656   710   798   862   913
   967  1026  1073  1121  1169  1219  1270  1318  1372  1428  1536  1589  1642  1698
  1738  1799  1848  1897  1945  1997  2050  2101  2206  2257  2311  2360  2412  2461
  2559  2611  2662  2712  2813  2915  2968  3015  3073  3114  3168  3220  3271  3367
  3419  3466  3515  3565  3617  3668  3734  3774  3827  3924  3977  4025  4070  4121
  4167  4216  4315  4468  4522  4573  4624  4675  4727  4777  4840  4943  4993  5046
  5145  5193  5245  5294  5349  5402  5459  5520  5735  5876]
```



```
[ ]: 94
```

1.5 Progress summary

Hardware: done. The circuit are working properly, but sometime generates a lot of noise randomly, but everything goes back to normal after I connected it to the oscilloscope.

Software: I finished most of the code, and is trying to optimize the peak finding algorithm. The code for the real time plot in the heartbeat sensor doesn't seems to work for my project, it creates a huge delay in the plot.