

Proyecto 3

Laboratorio de Microcontroladores ADCs y Puerto Serial

Marco Antonio Montero Chavarria Carné: A94000
Francisco Molina Carné: B14194

14 de octubre de 2015

1. Cambios en la solución propuesta

Para elaborar el graficador de señales efectivamente se implementó el esquema planeado de la figura 1, para ello lo primero que se trabajó fueron los bloques CDC ACM y ADC junto con la aplicación Minicom en linux para probar que efectivamente había comunicación serial entre el stm32 y la computadora. Posteriormente se elaboró un pequeño programa en python de prueba para recibir con pyserial los mismo datos pero en nuestra propia aplicación.

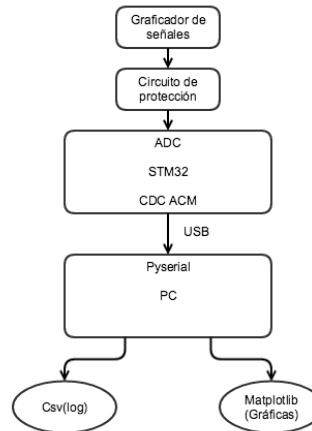


Figura 1: Esquema

Una vez probado que se recibían los datos se pasó a montar el circuito de la figura 2 como protección para el microcontrolador dando como resultado el circuito de la figura 3, donde encontramos el MCP6022 y resistencias de 680Ω y 470Ω para realizar el divisor de tensiones de forma que la tensión que llega a la salida V_o es igual a $V_o = \frac{470}{680} \cdot V_i = 0,69V_i$ con lo que el máximo $V_i = 5$ daría un $V_o = 3,45$, sin embargo al tomar medidas con el multímetro, la entrada máxima del V_i era de aproximadamente 4,3 con lo que la salida máxima del V_o alcanzaba los 2,97 V.

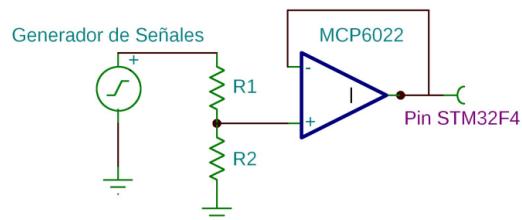


Figura 2: Circuito de protección

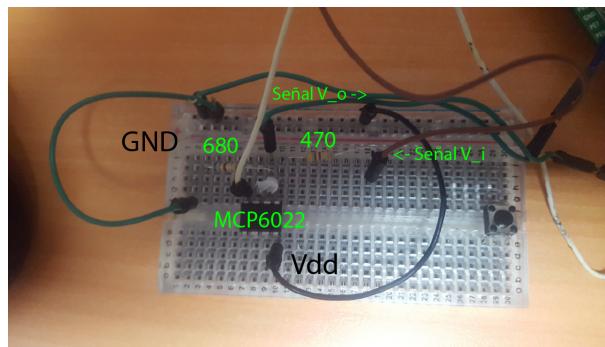


Figura 3: Circuito de protección

Ya con la tensión máxima asegurada por debajo de 3 V. se procedió a usar el graficador de matplotlib, junto con los datos recibidos de pyserial y con csv para guardar el log con los datos tomados. Como primera prueba se utilizó el código del archivo transmi2.py mostrado en la figura 4 junto con el generador de ondas, lo que devolvió como resultado la figura 5 al muestrear claramente a más del doble de la frecuencia de la onda.

```

9   size = 300 #Tamaño de la muestra
10  arrtension = [size]
11  arrtiempo = [size]
12  starttime = time.time() #Definir el tiempo inicial
13
14  ser = serial.Serial('/dev/cu.usbmodem1411', 115200) #60A78A81 open serial port
15
16
17 #Formato del nombre del archivo log
18 log_file = open("l"+datetime.datetime.now().ctime() + "data "+".csv", "wb")
19 #Formato del writer, lo que se escribe al log
20 writer = csv.writer(log_file, delimiter=' ', quotechar=' ', quoting=csv.QUOTE_MINIMAL)
21
22 #Ciclo de muestreo
23 for x in range (0, size):
24
25     #Tome el tiempo del momento
26     tiempo = time.time()-starttime
27     #Tome la tensión del momento
28     tension = ser.readline()
29     #Adjunte al arreglo que se pasara a plt
30     arrtension.append(tension)
31     arrtiempo.append(tiempo)
32     #Escriba en el log
33     writer.writerow(str(tiempo)+str(tension))
34
35
36 #Cierre el puerto
37 ser.close()
38
39 tension_flotante = map(float, arrtension)
40 #Acomode y grafique
41 plt.plot(arrtiempo, arrtension, 'ro')
42 plt.ylim([-5,5])
43 plt.xlim([0,0.1])
44 plt.ylabel('Tension en Volt')
45 plt.xlabel('Tiempo')
46 plt.show()
47

```

Figura 4: Graficación normal

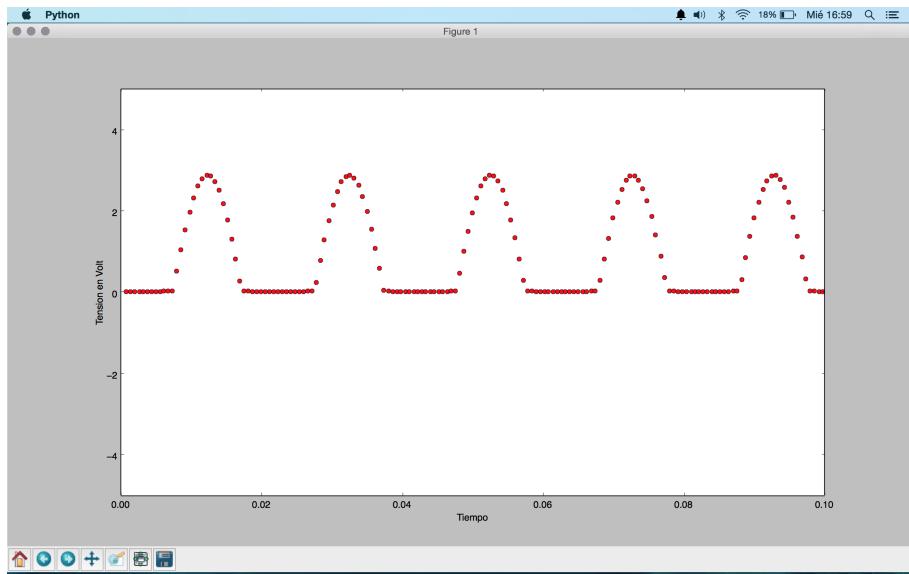


Figura 5: Onda senoidal

2. Innovación

Como innovación del proyecto se implementaron 2 mejoras, la primera fue utilizar un sensor de tacto externo mostrado en la figura 6 para verificar que se puede muestrear señales de cualquier periférico siempre y cuando se haga la conversión necesaria. La segunda fue implementar un código para muestrear en tiempo real los datos, de forma que se obtenga la señal en pantalla conforme se van tomando los datos. El código del archivo transmi.py para esto lo podemos encontrar en la figura 7 mientras que el sistema funcionando lo podemos encontrar en el video: <https://youtu.be/9R3dfunGKa4>



Figura 6: Sensor

```

9
10 size = 300 #Tamaño de la muestra
11 arrtension = [size]
12 arrtiempo = [size]
13 starttime = time.time() #Definir el tiempo inicial
14
15 ser = serial.Serial('/dev/cu.usbmodem1411',115200) #60A78A81 open serial port
16
17 #Formato del nombre del archivo log
18 log_file = open("["+str(datetime.datetime.now().ctime()) + "data "+".csv", "wb")
19 #Formato del writer, lo que se escribe al log
20 writer = csv.writer(log_file, delimiter=' ', quotechar=' ', quoting=csv.QUOTE_MINIMAL)
21
22 #Ejes del grafico de matplotlib
23 plt.axis([0, 100, -1, 6])
24 #Plot en modo interactivo
25 plt.ion()
26 #Mostrar el grafico
27 plt.show()
28
29 #Ciclo de muestreo
30 for x in range (0, size):
31
32     #Tome el tiempo del momento
33     tiempo = time.time()-starttime
34     #Tome la tensión del momento
35     tension = ser.readline()
36     #Imprima para pruebas
37     print tension
38     print tiempo
39     #Escriba en el log
40     writer.writerow(str(tiempo)+str(tension))
41     #Adjunte al gráfico plt
42     plt.scatter(tiempo, tension)
43     plt.draw()
44     #Espere un tiempo para tomar otro dato
45     time.sleep(0.001)
46
47 #Cierre el puerto
48 ser.close()
49

```

Figura 7: Tiempo real

3. Justificación de resultados

El hecho de ir realizando poco a poco los diferentes pasos, y atacando los errores uno por uno sirvió para que el proyecto fuera realizado con éxito, en cuanto a las señales y la graficación en la primera parte se obtuvo lo deseado, sin embargo no se realizó la conversión de vualta a la escala 0V-5V, pero esto queda en recomendaciones para futuros proyectos, ya que es simplemente multiplicar el valor final. En cuanto a tiempos, solo hizo falta preocuparse por el tiempo de muestreo ya que los datos se tomaban y posteriormente se imprimían.

Para la parte de innovación, especialmente en la sección de graficación en tiempo real, la temporización fue muy importante, ya que se notó un apilamiento de los datos en por parte de la librería pyserial que se resolvió bajando la frecuencia de muestreo del STM para sincronizar la toma de datos con el muestreo, sin embargo lo preferible sería que el STM tome los datos y que el muestreo sea independiente de la frecuencia del STM. Por ello se hace referencia en las recomendaciones, verificar la temporización y el apilamiento de pyserial como el ciclo de atención a interrupciones del contador del STM.

En conclusión se lograron los objetivos del proyecto y se exploró un poco más los alcances tanto de las librerías como del microcontrolador, esto nos acerca al objetivo del proyecto final de forma que muestrear señales de un potenciómetro es realizable para el controlador que queremos elaborar.

4. Recomendaciones

- En el caso de graficación en tiempo real, verificar los tiempos de muestreo del script en python y las frecuencias de trabajo del STM para la atención a interrupciones.
- Agregar el multiplicador a la salida del archivo .py para recuperar la señal original.
- Verificar la toma de datos y no el apilamiento, explorar las funcionalidades de pyserial.
- Tener sumo cuidado con el circuito de protección, probar que las tensiones sean las adecuadas antes de conectarlo al STM, así como revisar los pines que se están utilizando.
- Realizar pequeñas pruebas antes de programar cada parte y descomponer el problema en problemas más sencillos.