Modo streaming y contadores con la tarjeta DAQ LabJack U3-HV

Carlos Tomás Medrano Sánchez, Juan Carlos García López, Guillermo Palacios Navarro, Inmaculada Plaza García.

Objetivos: El objetivo de la práctica es utilizar las entradas y salidas de la DAQ para realizar operaciones con buena precisión temporal. Para acceder a ella utilizaremos Python, que nos permite acceder a alto nivel al driver de la tarjeta. Además veremos algunos aspectos del módulo de contadores.

Material necesario: Tarjetas DAQ LabJack U3-HV, protoboard.

Trabajo previo: Repasar las práctica 0 (conceptos de diccionarios y listas) y 1. Repasar la información de la tarjeta para conocer las características básicas de la tarjeta LabJack U3-HV. Se recomienda leer la práctica con antelación y traer hechos los códigos necesarios.

1. Concepto de Scan (modo buffered o streaming)

En la práctica 1 comprobamos que realizar operaciones de E/S con buena precisión temporal no se puede realizar por software (a menos que utilicemos un sistema operativo de tiempo real, pero no con los sistemas Windows o Linux habituales). Por ello, las tarjetas DAQ incorporan recursos para hacerlo de forma precisa:

- Señales de reloj para una gestión temporal correcta.
- Memoria para almacenar los datos.
- Configuraciones de inicio de la operación de E/S (por software, disparada por una señal externa).

Una secuencia de adquisición (scan) se refiere a la adquisición de varios canales a una determinada frecuencia de muestreo. La tarjeta LabJack U3-HV es una tarjeta de bajo coste, por lo que las frecuencias de muestreo no son elevadas y las opciones para realizar el scan son limitadas. La frecuencia de muestreo se puede aumentar si se pierde resolución en las lectura analógicas.

Recuerda que se supone que hemos importado el módulo, *import u3*, y abierto la tarjeta con *d=u3.U3()* para los comandos que veremos a continuación. La variable *d* representa la tarjeta. Y antes de salir de la consola conviene cerrar la tarjeta, *d.close()*.

Para configurar el modo streaming, utilizaremos el módulo *streamReader*. Podemos primero crear un objeto S*treamReader*, al que le pasamos la variable que indica la tarjeta abierta.

In[]: import streamReader as sr
In[]: stream = sr.StreamReader(d) #stream: nombre como otro cualquiera

Ahora podemos comenzar el streaming con el método *start*. Para configurarlo hay que pasarle una serie de parámetros:

- *NumChannels*, indica el número de canales.
- *PChannels* y *NChannels*, dos **listas**, que indican los canales que se van a leer. Incluso aunque sea un sólo valor debe estar como una lista (entre corchetes). Los valores que hay que indicar son los que se vieron en la anterior práctica (y también en la sección 2.6.1 del manual de usuario). Además, es posible adquirir también entradas digitales o valores de contadores y timers. Según la sección 3.2.1 del manual de usuario, para leer las entradas digitales basta poner como canal positivo el 193, y como negativo el 31. El valor devuelto corresponde a un número de 16 bits, siendo EIO los 8 bits superiores, y FIO los 8 bits inferiores.
- *ScanFrequency*, indica la frecuencia del scan en Hz. Dado que en cada scan hay que muestrear varios canales, las muestras se toman a una frecuencia que cumple:
  Sample Rate = ScanFrequency x NumChannels.
  Es conveniente que esta frecuencia no supere el valor de la sección 3.2.1 del manual de usuario. En particular si queremos obtener la máxima precisión en bits, no puede superar 2.5 kHz.

Por ejemplo, el siguiente comando inicia el streaming leyendo AIN0, AIN1 (ambas en modo Single Ended) y las entradas digitales, a 1 Hz:

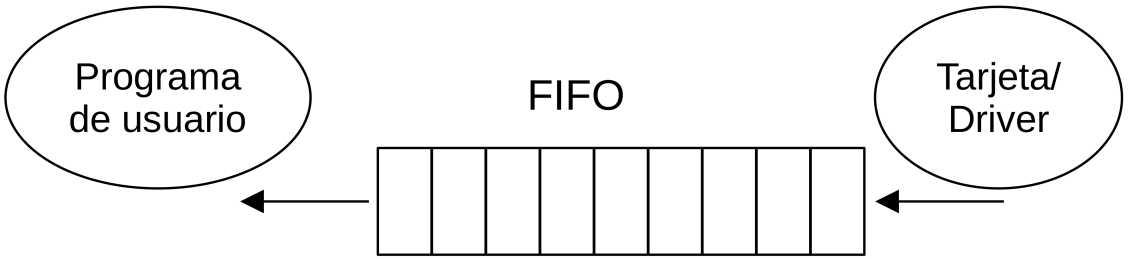
In[]: stream.start(NumChannels=3, PChannels=[0,1,193], NChannels=[31, 31, 31], ScanFrequency=1)

Para obtener los datos podemos llamar al método *get*. El método devuelve *None* si no hay ningún dato, o una variable de tipo diccionario si hay algún dato. El diccionario puede tener varios campos (*keys*), dependiendo de los canales que hayamos configurado al comenzar el streaming. En el ejemplo anterior los campos pueden ser 'AIN0', 'AIN1' y 'AIN193' (hay otros campos también que pueden informar acerca de posibles errores, aunque no los veremos en la práctica). El siguiente código permite comprobar si hay nuevos datos e imprime un mensaje distinto en cada caso:

```
In[]: res = stream.get()
In[]: if(res is not None):
…       if ('AIN0' in res.keys()): # Comprobamos si la key ‘AIN0’ está
…          # Si está, imprimimos los valores
…          print('Nuevo dato de AIN0', res['AIN0'])
…       elif('AIN1' in res.keys()):
…          print('Nuevo dato de AIN1', res['AIN1'])
…       elif('AIN193' in res.keys()):
…          print('Nuevo dato de AIN93', res['AIN193'])
…       else:
…          print('Las keys no están ??')
```

Cada vez que queremos obtener un nuevo dato debemos hacer otra llamada al método *get()*.

IMPORTANTE: Fíjate que comprobando si el valor devuelto por *get* es *None*, podemos saber si hay un nuevo valor y actuar en consecuencia. Si no hay un nuevo valor, podemos dedicar el programa de control a otras tareas hasta que volvamos a comprobarlo. Realmente los datos están en una memoria de tipo FIFO (First In First Out). La tarjeta y el driver producen los datos y los van almacenando. El programa del usuario, con *get*, los retira de la FIFO (consume los datos). A este esquema se la llama modelo productor-consumidor:



Para terminar el modo streaming usaremos el método *stop*. Tiene un parámetro *reset* que me indica si quiero borrar los datos de la FIFO (generalmente lo pondremos en *True*, si no la siguiente vez que comencemos el flujo de datos tendremos datos mezclados en la memoria de dos capturas distintas).

In[]: stream.stop(reset = True)

En el ejemplo anterior, cada uno de los campos contiene a su vez una **lista**. Es decir, *res['AIN0']* es una lista que puede contener varios valores. Es posible que se imprima uno sólo porque la frecuencia es baja. Puedes probar el ejemplo con una frecuencia más alta.

IMPORTANTE: si por algún motivo un flujo (stream) se empieza y no se termina antes de acabar el programa, puede que la tarjeta se quede en ese estado. Para sacarla, podemos:

- Salir de *ipython* y volver a entrar
- Desconectar la tarjeta del USB y volver a introducir el cable (lo más efectivo)

Ejercicio:

- Establece en el generador una onda de 0 a 1V, frecuencia 10 Hz. Compruébala en el osciloscopio.
- Introduce la onda en la entrada analógica 0, *AIN0*. No lo hagas directamente, hazlo a través de una R de 1kΩ.
- Descarga de Moodle el programa *capturaAlumnos.py* e intenta entender su estructura general. Los interrogantes son valores que tendrás que rellenar tú, o comentarios que debes hacer explicando el programa (ver siguiente punto).
- Modifica el programa (los símbolos ?? que se han dejado sin rellenar) para conseguir que la captura se realice por *AIN0* a 100 Hz, midiendo valores en modo Single Ended.
- Una vez que has modificado el programa, impórtalo desde una consola de ipython, llamando a la función para capturar 5 segundos de señal.
- Visualiza la lista de valores obtenidos. Razona que corresponde a una onda de 10 Hz. Nota: para visualizar una lista o un array de valores podemos usar el

módulo *matplotlib.pyplot*. Por ejemplo, si queremos visualizar una lista almacenada en la variable *y*, poniendo un punto en cada uno y uniendo con rayas los puntos, haremos esto:

```
import matplotlib.pyplot as plt
plt.ion()
y=[0,1,4,9,16]
plt.plot(y, 'o-')
```

2. Visualización de datos: osciloscopio en el PC

- En cualquier lenguaje, existen numerosas librerías de visualización de datos. En nuestro caso, vamos a hacer uso de una sencilla basada en *pygame*. El programa *oscope_labjacku3.py* se basa en ella (lo puedes descargar de Moodle). Toma datos de *AIN0* en el rango de -10V a 10V, a una frecuencia máxima de 20 kHZ, y los visualiza. La visualización se controla con las teclas:

  f - trigger en el flanco de bajada, *r* - trigger en el flanco de subida
  s - sin trigger
  flecha hacia arriba / abajo: aumentar / disminuir el nivel del trigger
  9 o *0* - aumenta /disminuye la escala de tiempos
  q - sale

  Para ejecutar el programa desde una consola de linux haz:
  $ *python oscope_labjacku3.py*

- Configura el generador de señales para obtener una señal entre -0.5 y 0.5 V, con una frecuencia de 100 Hz. Comprueba en el osciloscopio la señal.
- Introduce la señal, a través de una R de 1 kΩ, en la entrada AIN0. Observa la onda en la pantalla del PC. Comprueba como, si cambias la frecuencia, se observa en la pantalla.

3. Contadores y temporizadores

La tarjeta U3-HV tiene 2 “timers” y 2 contadores. Los timers pueden servir para medir períodos, anchuras de pulso, entradas en cuadratura o generar PWM. También admiten la posibilidad de contar pulsos externos. Los contadores, permiten contar flancos de bajada de un pin externo. Para configurar estos sistemas hay que:

- Habilitar o deshabilitar cada uno de ellos.
- Indicar los pines en los que estarán, siempre dentro de FIO4-7 (o en EIO0-7, el conector DB15 que no usamos). Siempre aparecerán los que estén habilitados en este orden: Timer0, Timer1, Counter0, Counter1. Para indicar el pin tenemos el argumento *TimerCounterPinOffset*.

Ejemplo: Timer 0 habilitado, Timer1 deshabilitado, Counter0 deshabilitado, Counter1 habilitado, y *TimerCounterPinOffset*=6: Esto da lugar a ésta configuración de pines:

FIO6 = Timer0
FIO7 = Counter1

Al configurar un pin como timer/counter, ya se configura automáticamente como digital y de salida/entrada según el uso del timer/counter. Conviene recordar también que en la tarjeta U3-HV los pines FIO0-3 están reservados a entradas analógicas HV.

Como ejemplo de uso de un contador, vamos a contar los flancos de bajada en el pin FIO6. Para ello, configuramos el contador 0 con este comando:

In[]: d.configIO(EnableCounter0 = True, TimerCounterPinOffset = 6, FIOAnalog = 0x0f)

Debido a la resistencia interna de pull-up, el valor por defecto de la entrada es de '1'. Si atornillas un cable a FIO6 y tocas con la otra punta la señal de tierra de la tarjeta, puedes generar pulsos de bajada. Debes tener en cuenta que habrá muchos rebotes. Para leer el valor del contador 0 haremos:

In[]: d.getFeedback(u3.Counter0(Reset = False))

que devuelve el valor de la cuenta. El argumento *Reset* sirve para anular o no el valor del contador tras la lectura. Si no se anula, las cuentas se van acumulando.

Ejercicio:

- Comprueba que el contador cambia de valor cada vez que tocas tierra con el cable. Comprueba que si el argumento *Reset* es True, el contador vuelve a 0 cada vez que se lee.
- Comprueba el efecto de tener el parámetro *Reset* como *True* o como *False*, haciendo varias pruebas de provocar flancos en la entrada y leer los valores.
- Monta un circuito RC a la entrada de FIO6 para eliminar rebotes y comprueba que los elimina en parte. ¿Qué circuito digital permitiría eliminar rebotes también? ¿Cómo se configuraría?