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

Carlos Tomás Medrano Sánchez, Inmaculada Plaza García, Raúl Igual Catalán, Iván García-Magariño 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, resistencias y condensadores.

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) está formada 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.

Nota: 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.

La configuración del modo streaming se realiza utilizando la función *d.streamConfig*, que posee varios parámetros, que describimos a continuación:

- *NumChannels*, indica el número de canales.
- *PChannels* y *NChannels*, dos listas de canales, que indican los canales que se van a leer. 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.

Una vez hemos configurado el scan, hay que comenzar la adquisición. En esta tarjeta, esto sólo se puede hacer a través del comando *d.streamStart()*. Los resultados se recogen con *d.streamData()*, pero esto requiere una explicación más detallada. s*treamData()* devuelve un objeto que se denomina en Python un generador. Básicamente, lo que significa es que admite el método next(), que devuelve uno o varios datos. Por ejemplo, vamos a suponer que leemos los canales analógicos 0, 1 (que son siempre analógicos), y los digitales a 5 Hz. Deberíamos hacer lo siguiente para configurar la tarjeta y comenzar la adquisición:

In[]: *d.configIO(FIOAnalog = 0x0f)*
In[]: *d.streamConfig(NumChannels = 3, PChannels = [0, 1, 193], NChannels = [31, 31, 31], ScanFrequency=5)*
In[]: *d.streamStart()*
In[]: *res=d.streamData()*

Los datos 'están' en la variable *res*. Para obtener los datos, llamaremos a *res.next()*, que los eliminar del buffer de memoria driver/tarjeta para que podamos usarlos: *res.next()* devuelve un diccionario:

In[]: *print res.next()*

que imprimirá algo parecido a esto:

{'AIN0': [0.4614079999999987, 0.4614079999999987, 0.44131199999999815],
'AIN1': [0.895926784, 0.892352608],
'AIN193': [(240, 255), (240, 255)],
'errors': 0,
'firstPacket': 0,
'missed': 0,
'numPackets': 1,
'result': ….}

En *'AIN0'*, tendremos varias lecturas del canal 0, idem para *'AIN1'*. En *'AIN193'* tenemos la lectura de las entradas digitales. Ya se separan la parte de *FIO* y de *EIO*, de forma que se devuelven en pares (*FIO7-0, EIO7-0*).

Conviene también fijarse en los campos '*errors*' y '*missed*'. Dado que la memoria de buffer está limitada, si no leemos los datos puede llegarse a una situación de llenado. En ese caso, la tarjeta no captura más datos, pero al enviar la memoria al usuario indicará el

problema a través de esos campos (número de errores y paquetes que se han dejado de capturar).

Recuerda que podemos acceder a los campos de un diccionario con las palabras clave. Por ejemplo, si queremos imprimir los datos obtenidos en el canal 0 haremos esto:

```
In[]: data = res.next()
In[]: for nn in range(0, len(data['AIN0'])):
          print data['AIN0'][nn]
```

Nota: *data* en el ejemplo anterior es un diccionario. *data['AIN0']* es el valor almacenado con la palabra clave *'AIN0'*. *data['AIN0']* resulta ser una lista de valores y *data['AIN0'][nn]* es el nn-ésimo valor de la lista. La función *len()* permite obtener el número de elementos de una lista.
Es también interesante que la variable res también puede ser iterada en un bucle. De esta forma podemos acceder a todos los datos hasta que acabemos el bucle interrumpiendo con CTRL+C.

```
In[]: for data in res:
          print data['AIN0']
```

Finalmente, para que la adquisición en modo streaming termine, hay que usar *d.streamStop()*:

```
In[]: d.streamStop()
```

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)

Además, si el flujo ya estaba abierto, conviene comprobarlo antes de abrirlo. Lo podemos hacer a través de la variable booleana *d.streamStarted*, de forma que para abrirlo haremos:

```
In[]: if(not d.streamStarted): d.streamStart()
```

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 *pylab*. 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:

  In[]: *import pylab*
  In[]: *pylab.ion()*
  In[]: y=[0, 1, 4, 9, 16]
  In[]: *pylab.plot(y, 'o-')*

  Para otras opciones de dibujo hacer *help(pylab.plot)*.
- No olvides subir el código modificado a Moodle.

Es fácil realizar visualizaciones en tiempo real de los valores obtenidos en la tarjeta, a modo de osciloscopio. Puedes consultar al profesor para ver algunos ejemplos de códigos que permiten dicha visualización.

2. 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?