Anexo 6.1

Descripción de los programas utilizados para caracterizar el sistema vigaflexible-actuador

A6.1.1 Introducción

- *¿Qué es LabVIEW?*

LabVIEW es un entorno de programación, mucho más moderno que C, BASIC o LabWindows/CVI. No obstante, LabVIEW es diferente de ellos en un aspecto importante. Otros sistemas de programación usan lenguajes *basados en texto* para crear líneas de código, mientras que LabVIEW utiliza un lenguaje de programación gráfico, **G**, para crear programas en forma de diagramas de bloques.

LabVIEW, como C o BASIC, es un sistema de programación de carácter general con amplias librerías de funciones para cualquier tarea que se desee programar. LabVIEW incluye librerías para la adquisición de datos, GPIB y el control de instrumentos, análisis de datos, presentación de datos y almacenamiento de datos. LabVIEW también incluye herramientas que afectan a la ejecución del programa de forma convencional. Éstas consisten en poder introducir puntos de pausa, animar la ejecución del programa para ver cómo fluyen los datos a través del programa y ejecutar paso a paso el programa; todo ello con el fin de poder suprimir posibles fallos cometidos durante la programación.

- *¿Cómo trabaja LabVIEW?*

Como se ha dicho anteriormente, LabVIEW es un sistema de programación de carácter general que incluye librerías de funciones y herramientas de desarrollo diseñadas específicamente para la adquisición de datos y el control de instrumentos. Los programas de LabVIEW son llamados *Virtual Instruments (VIs)* porque su apariencia y funcionamiento pueden imitar instrumentos físicos.

Un VI consta de una interface de usuario interactiva y de un diagrama de flujo de datos que sirve como código fuente, en el interior del cual se pueden encontrar las conexiones de iconos que permiten a los VIs ser llamado desde VIs de niveles superiores. Más concretamente, los VIs están estructurados de la siguiente forma:

- A la interface de usuario interactiva de un VI se le conoce como *panel frontal* porque simula el panel de un instrumento físico. El panel puede contener botones, pulsadores, gráficos y otros controles e indicadores. La entrada de datos se produce utilizando el ratón o el teclado, visualizando más tarde los resultados en la pantalla.

- El VI recibe instrucciones de un *diagrama de bloques*, que se debe construir en **G** (lenguaje de programación gráfico). El diagrama de bloques es una solución ilustrada a un problema de programación. Dicho diagrama es también el código fuente para el VI. En su interior podemos introducir los iconos que representan otros VIs. Hay que tener en cuenta que los VIs son jerárquicos y modulares. Es decir, es posible usarlos como programas de nivel superior o como subprogramas dentro de otro programa. A un VI dentro de otro VI se le denomina subVI. Los iconos y conectores de un VI trabajan como una lista de parámetros gráficos para que otros VIs pueden transferir datos a un subVI.

Con estas características, LabVIEW fomenta y respeta el concepto de programación modular. Es posible (casi indispensable) dividir una aplicación compleja en una serie de subtareas sencillas. Se programa un VI para ejecutar cada subtarea y entonces se combinan estos VIs en un diagrama de bloques superior para ejecutar la tarea compleja. Finalmente, el VI de nivel superior contiene un conjunto de subVIs que representan funciones de aplicación.

Resumiendo, los VIs (Instrumentos Virtuales) tienen dos partes principales: el panel frontal y el diagrama de bloques. El panel frontal representa la interface de usuario del VI. El diagrama de bloques es el código ejecutable. Siempre es posible utilizar un VI como un subVI en el diagrama de bloques de otro VI. Para conseguirlo, basta pegar en el diagrama de bloques del VI el icono del VI que actuará como subVI.

A6.1.2 Diseño de los programa de captura y generación de señales.

¿Cuál es la funcionalidad de la aplicación a desarrollar en LabVIEW?

La principal funcionalidad de la aplicación es adquirir y generar señales analógicas. Por *adquisición de señales* se entiende medir señales que procederán del sistema observado para posteriormente analizarlas. Por *generación de señales* se entiende enviar señales al sistema con el objetivo de controlar su funcionamiento. En ambos casos se trata de señales analógicas variables en el tiempo.

Adquisición de sañales

A la hora de adquirir las señales que representan el comportamiento del sistema, hay que tener en cuenta un aspecto importante, dependiendo de lo que se quiere hacer con los datos después de adquirirlos. En las primeras observaciones, los datos se analizarán una vez se hayan adquirido en su totalidad un número predeterminado de muestras. Esto evitará un tiempo de cálculo, innecesario por otra parte, mientras se realiza la adquisición de datos. De esta forma conseguiremos que la tarjeta de adquisición obtenga una captura lo más real posible impidiendo que el sistema entre en bucles de cálculo que bloqueen la adquisición de datos.

La forma más eficiente en la que LabVIEW puede trabajar, teniendo en cuenta las necesidades de esta aplicación, es utilizar una memoria de datos intermedia (*buffer*) a partir de la memoria "física" del ordenador. En la aplicación VI, se debe especificar el número de muestras que deben ser tomadas y el número de canales de los que se tomarán las muestras. A partir de esta información, LabVIEW asigna una zona de memoria que guarde un número de muestras igual al número de muestras por canal multiplicado por el número de canales. Mientras la adquisición continúa, el *buffer* se llena con los datos tomados; sin embargo, los datos no son accesibles hasta que LabVIEW adquiere todas las muestras. Una vez la adquisición de datos se ha completado, los datos que están en la memoria pueden ser analizados, almacenados en un fichero, o visualizados en pantalla a partir de la aplicación VI diseñada. Esta técnica es conocida con el nombre de **Simple-Buffered**, porque solo se crea una única zona de memoria.

Se puede dar el caso de que esta técnica no sea posible utilizarla, debido a que se necesite adquirir más datos de los que la memoria es capaz de almacenar o que se desee capturar una señal durante largos periodos de tiempo. Para este tipo de aplicaciones se debe utilizar la técnica **Circular-Buffered**. Con esta técnica, se puede configurar la tarjeta para que adquiera datos continuamente mientras LabVIEW recupera los datos adquiridos. Esta técnica difiere de la anterior sólo en cómo LabVIEW coloca los datos en el *buffer* y cómo los recupera de éste. El *buffer circular* se llena con los datos adquiridos, como si fuera un *buffer sencillo*; sin embargo, cuando éste se ha llenado, se vuelve al principio y llena el mismo *buffer* otra vez. Esto significa que los datos pueden ser leídos continuamente de la memoria del ordenador, pero únicamente se puede utilizar una cantidad de memoria definida. La aplicación VI deberá recuperar los datos en bloques, a partir de una posición determinada del *buffer*, mientras que los datos entran en el *buffer circular* en una posición diferente. De esta forma, los datos de la memoria que no han sido leídos no serán "machacados" por datos nuevos.

Existen dos posibles problemas a la hora de crear la aplicación VI. Puede ser que dicha aplicación intente recuperar los datos del *buffer* más rápidamente de lo que los datos son colocados en éste. En este caso en el que la VI intenta leer datos del *buffer* que todavía no han sido recopilados, LabVIEW espera esos datos que fueron solicitados para ser adquiridos y entonces los recupera.

También podría darse el caso que la aplicación VI no recuperase los datos del *buffer* lo suficientemente rápido antes de que LabVIEW "machacase" parte de esos datos. En este caso, LabVIEW retorna un mensaje de error advirtiendo de que los datos recuperados del buffer son datos "machacados".

Generación de señales

Se trata de generar señales variables con el tiempo. Al igual que sucede con la adquisición de datos, podemos generar señales utilizando esas mismas técnicas.

La primera de ellas se conoce como *generación de señales basada en un único buffer*. En este caso debemos almacenar un ciclo de la función que se desea generar en un vector de LabVIEW y programar la tarjeta de adquisición de datos para que genere los valores de ese vector una sola vez o ininterrumpidamente de uno en uno a una frecuencia establecida.

La segunda de ellas se conoce como *generación de señales basada en un buffer circular*. Este caso es útil cuando se quiere generar una señal que varia continuamente o cuando la cantidad de datos de esa función es demasiado extensa para que quepa en una memoria *buffer*. Por ejemplo, se dispone de un gran fichero almacenado en el disco que contiene datos que se quieren extraer. Debido a que LabVIEW es incapaz de almacenar toda esa señal completa en un único buffer, lo que se debe hacer es cargar continuamente los datos en el buffer durante la generación de la señal.

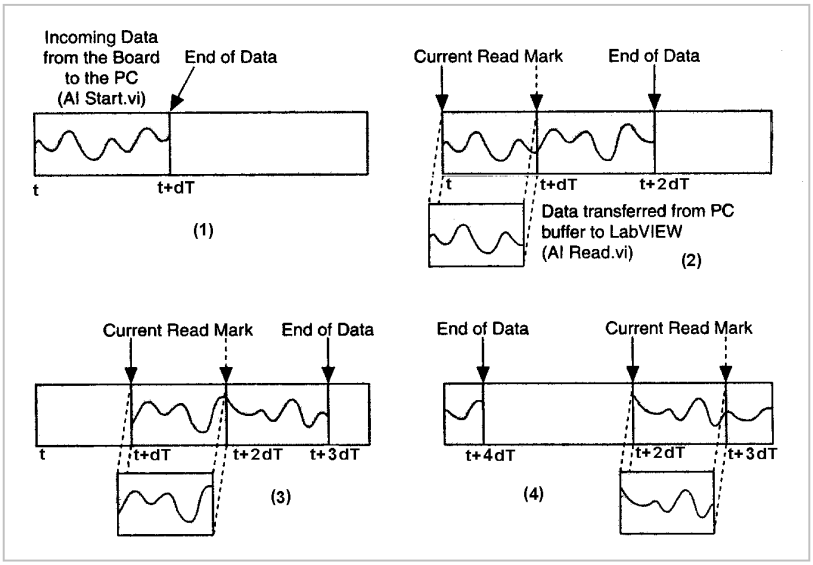

Figura A6.1.1Forma de trabajar del Buffer Circular

A6.1.3Descripción del programa de captura y generación de señales

Antes de pasar a la descripción detallada del panel frontal y del diagrama de bloques del programa, se comentará la filosofía de este. El programa ha sido diseñado para realizar la adquisición y generación de datos de forma *simultánea*, es decir, ambas se realizan en el mismo periodo de tiempo. Tanto para la generación como la adquisición de datos, el tipo de

buffer utilizado es *único* (simple-buffered), ya que se trata de una aplicación en la que no es necesario utilizar un *buffer circular*. Pese a que la adquisición y generación de señales es simultanea, éstas *no están sincronizadas*. Esto significa que existe un desfase temporal de milésimas de segundo entre el inicio de la adquisición y de la generación. Esta característica la podemos observar en el diagrama de bloques: el subVI AOStart.vi es llamado antes que el AIStart.vi debido a la dependencia de datos de error. Esta dependencia de datos provoca que la adquisición se inicie después de que la generación de datos haya empezado.

Descripción del panel frontal.

A continuación puede verse la interface de usuario del programa.

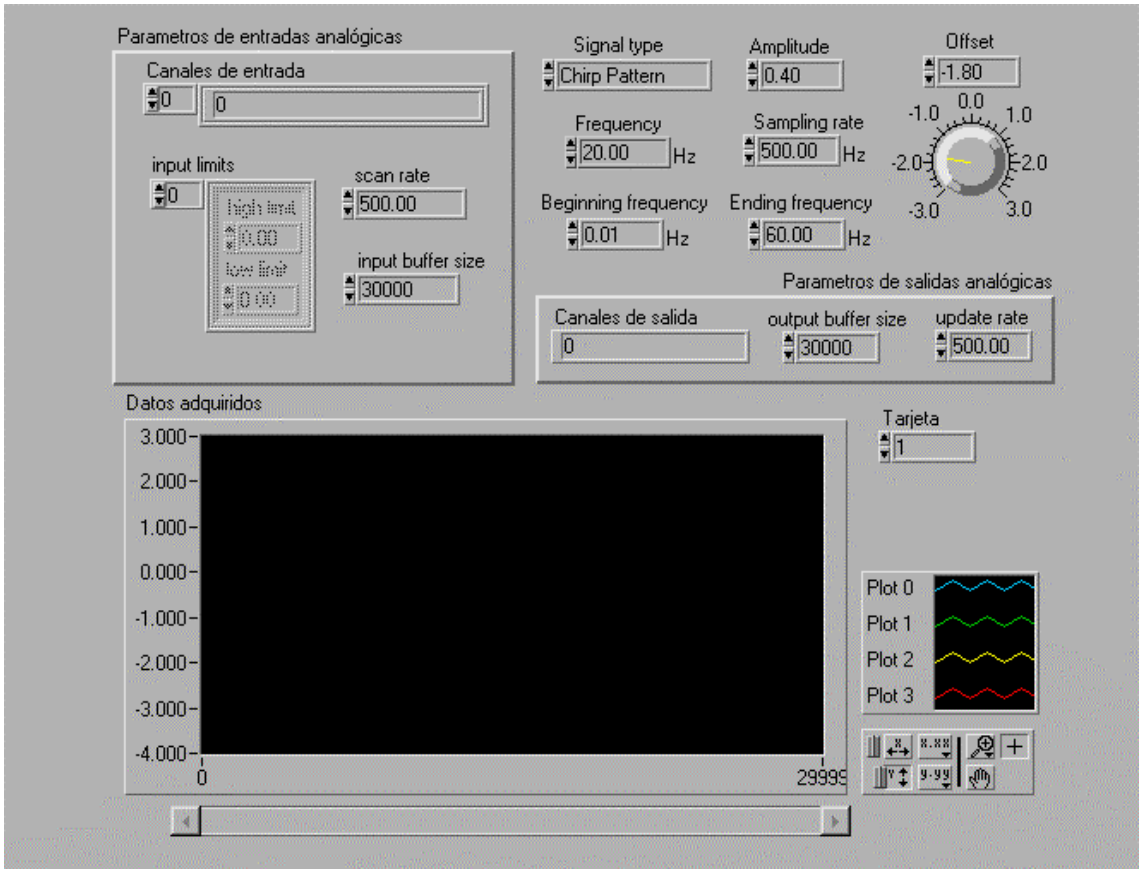

Figura A6.1.2Panel frontal del Programa de adquisición y generación de datos

Ésta consta de cuatro partes bien definidas; que son las siguientes:

1. Parámetros de la entrada analógica.

2. Parámetros de la salida analógica.
3. Señal generada.
4. Visualización de los datos captados.

En la primera de ellas encontramos todos aquellos parámetros que hacen referencia a entradas analógicas. Son los siguientes:

Input buffer size: hace referencia al tamaño de la memoria intermedia (buffer) que asignamos a todos los canales de la tarjeta de entradas analógicas. En el caso de la tarjeta de adquisición utilizada (PCI-1200) el tamaño de éste no puede ser superior a 64Kbytes. Como se deben utilizar dos canales de entrada analógica para capturar las señales, cada uno de ellos debe reservarse un tamaño inferior a 32Kbytes.

Input channels: con este parámetro se asignan los canales de entrada que realizarán captura de datos. Hay que introducir, a través del control, el número y nombre de canal que se definieron previamente utilizamos al configurar los canales físicos de entrada de la tarjeta.

Scan rate: define el número de lecturas por segundo que LabVIEW captura de todos y cada uno de los canales de entrada asignados por el usuario. Por ejemplo, con un *scan rate* de 10 Hz, LabVIEW muestrea cada canal de lectura 10 veces por segundo. Lo podemos traducir como frecuencia de muestreo.

Input limits: define los niveles de tensión superior e inferior para cada canal. Con ello se limita el rango de lectura de cada canal y permite detectar condiciones de lectura anormales.

Los parámetros de salida analógica son los siguientes:

Output buffer size: hace referencia al tamaño de la memoria intermedia (buffer) que asignada a todos los canales correspondientes a las salidas analógicas. No puede ser superior a 64Kbytes. En este caso, únicamente se utiliza un canal de salida.

Output channel: con este parámetro se define el canal de salida que realizará la escritura de datos. Se ha de introducir, a través del control, el nombre de canal que previamente fue definido al configurar los canales físicos de salida de la tarjeta.

Update rate: hace referencia al número de escrituras por segundo que LabVIEW

realiza en todos los canales de salida asignados por el usuario. Por ejemplo, con un *update rate* de 10 Hz, LabVIEW escribe en el canal seleccionado 10 veces por segundo. Cabe destacar que para obtener unas muestras lo más representativas posible, el valor de scan rate y update rate serán iguales.

Los parámetros de la señal generada son los siguientes:

Tipo de señal: con este control es posible seleccionar el tipo de onda que se generará. Se puede escoger entre una señal senoidal, cuadrada, triangular, diente de sierra o chirp.

Amplitud y Offset: con estos controles se define el valor de la amplitud de la señal generada así como también su nivel de offset, ambos expresados en Voltios.

Frecuencia: con este control sse define la frecuencia de la señal para todos los tipos de señal excepto la chirp, que está parametrizada por las *Frecuencia inicial* y *Frecuencia final*. Todas ellas expresadas en Hz (ciclos/seg).

Sampling rate: define la frecuencia de muestreo con la que se crea la señal de salida. Es decir, si se introduce un sampling rate de 500 Hz, la tarjeta actualizará el valor de la salida analógica 500 veces por segundo.

Por último, se ha previsto un gráfico para poder visualizar los datos capturados a través de los dos canales de entrada.

Descripción del diagrama de bloques.

A continuación puede verse el diagrama de bloques del programa en la figura A6.1.3.

El diagrama de bloque está dividido en cuatro partes principales. La primera de ellas, situada en la parte superior, incluye todos aquellos subVIs relacionados con los canales de entrada analógica (adquisición de señales). La segunda, situada en la parte central, incluye todos aquellos subVIs relacionados con los canales de salida analógica (generación de señales). La tercera, situada en la parte inferior izquierda, incluye el subVI diseñado para la generación de la señal. Por último, en la parte inferior derecha está el VI que permite almacenar los datos obtenidos.

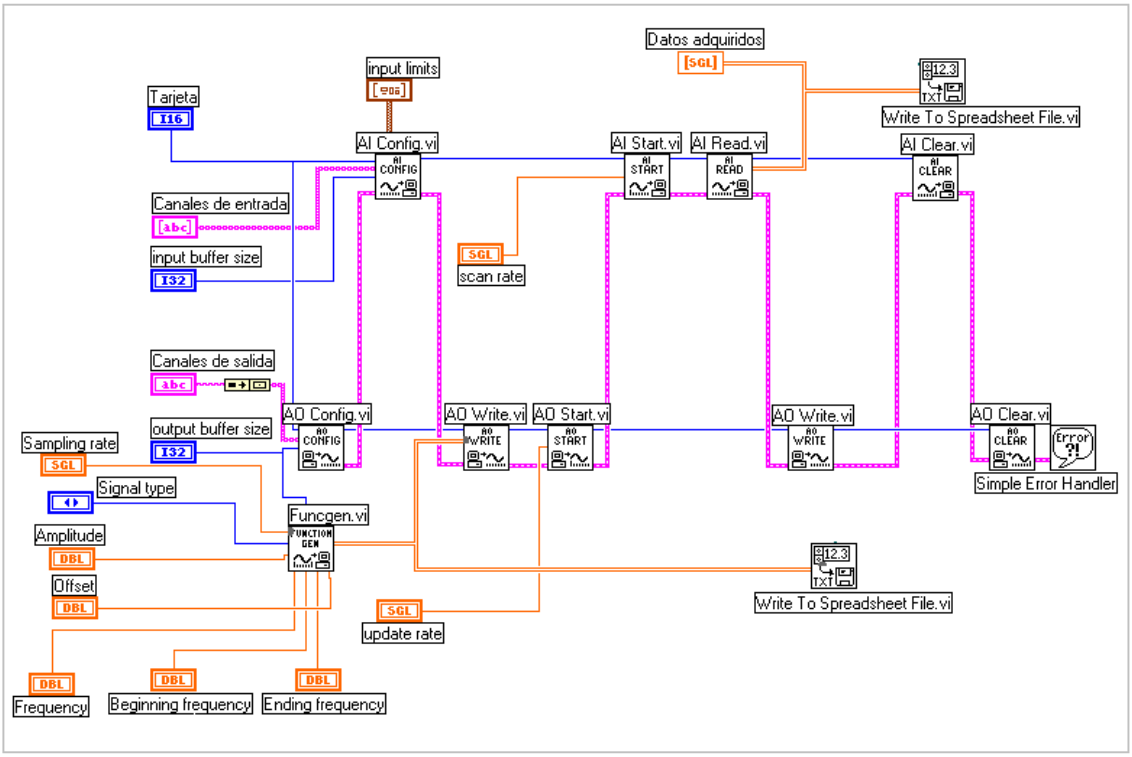

Figura A6.1.3 Diagrama de bloques del Programa de adquisición y generación de datos

A continuación, se indican detalladamente los parámetros utilizados de cada una de las librerías utilizadas. Cabe destacar que los parámetros configurados en todas las librerías utilizadas son los más básicos. Es posible configurar parámetros algo más complejos que permitirían tener un control más preciso sobre las librerías. Sin embargo, con los parámetros normales ya es posible realizar las tareas deseadas. En caso de que algún parámetro tenga asignado un número entre paréntesis, éste indica el valor por defecto.

Librerías de entradas analógicas.

La descripción que se encuentra en las siguientes líneas se realiza siguiendo el orden de ejecución de los VI.

1.AIConfig.vi

Lo primero que hace el VI es configurar una serie de parámetros necesarios para realizar la operación de adquisición de datos. Así se configura el hardware y se define un buffer para la captura de datos. Los parámetros configurables por este VI se ven en la Figura A6.1.4.

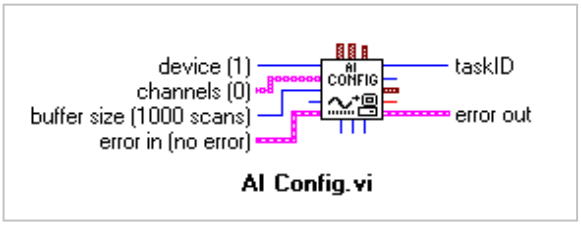

Figura A6.1.4 Librería AIConfig.vi

- **Device:** es el número que ha recibido la tarjeta en la configuración inicial.
- **Channels:** es una lista de los canales analógicos que serán utilizados para la adquisición de señales, es decir, es una "declaración" de los canales que van a ser leídos.
- **Buffer size:** es el número de "scans" que serán almacenados en el buffer de entrada. Es decir, define el tamaño del buffer de entrada. Hay que tener en cuenta que el número de muestras de entrada en un "scan" es igual al número de canales de entrada; así, un pulso del reloj de "scan" produce lee una nueva muestra en cada canal de entrada.
- **Error in:** describe condiciones de error que ocurren antes de que el VI se ejecute. Si un error ha ocurrido ya, el VI retorna el valor de ese error en **error out**.
- **Task ID:** es un número generado por Labview que identifica el tipo de operación que se realiza. Hay varios códigos de función, de entre todos ellos, los utilizados para realizar la aplicación son los siguientes:

Código de función	Operación de E/S
1	Entrada analógica
2	Salida analógica

Tabla A6.1.1 Códigos de función

2. AIStart.vi

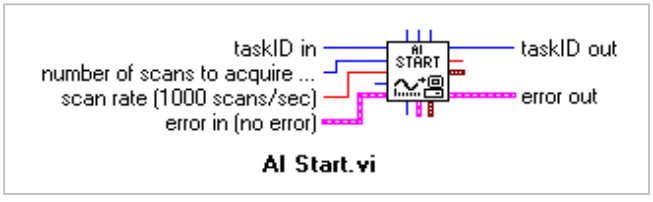

Figura A6.1.5 Librería AIStart.vi

A continuación el programa fija varios parámetros para iniciar la captura de datos. Una vez se han configurado dichos parámetros, este VI inicia la lectura de los canales de entrada. Los parámetros que configuramos en este caso se ven en la figura anterior:

- **Task ID in:** se utiliza para identificar el tipo de operación que se realiza.
- **Number of scans to acquire:** hace referencia al número total de "scans" que Labview captura antes de que la adquisición esté completa. Si este parámetro se configura con el valor **–1**, el existente por defecto, Labview captura exactamente un buffer de datos. Esto quiere decir que captura tantos datos como lo fijado en el tamaño del buffer (**buffer size**) de la librería AIConfig.vi.
- **Scan rate:** hace referencia a la frecuencia con la que se toman los datos. Indica el número de "scans" que adquiere por segundo.
- **Error in:** describe condiciones de error que ocurren antes de que el VI se ejecute. Si un error ha ocurrido ya, el VI retorna el valor de ese error en **error out**.
- **Task ID out:** tiene el mismo valor que **Task ID in**.

3. AIRead.vi

Lo siguiente que hacemos es fijar varios parámetros para recuperar los datos del buffer de entrada. Los parámetros que configuramos en este caso se ven en la siguiente figura:

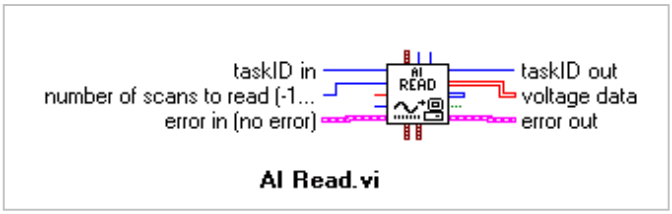

Figura A6.1.6 Librería AIRead.vi

- **Task ID in:** se utiliza para identificar el tipo de operación que se realiza.
- **Number of scans to read:** hace referencia al número total de "scans" que Labview debe recuperar del buffer de adquisición. Si este parámetro se configura con el valor **–1**, el existente por defecto, Labview recupera exactamente el mismo número de "scans" que se adquirieron con la librería AIStart.vi.

- **Error in:** describe condiciones de error que ocurren antes de que el VI se ejecute. Si un error ha ocurrido ya, el VI retorna el valor de ese error en **error out**.
- **Task ID out:** tiene el mismo valor que **Task ID in**.
- **Voltage data:** es una matriz de 2D que contiene los datos capturados. Cada fila contiene los datos de un "scan", es decir, contiene un valor capturado (muestra). Cada columna contiene todos los datos capturados de un canal.

	Canal de entrada 0	**Canal de entrada 1**	**…**
Scan 1			
Scan 2			
Scan 3			
…			

Tabla A6.1.2 Matriz 2D con los datos capturados

4. AIClear.vi

Por último, lo que hace el programa es desconfigurar la asignación realizada de todos los buffers de entrada y otros recursos utilizados para la captura de datos. Los parámetros que configuramos en este caso se ven en la siguiente figura:

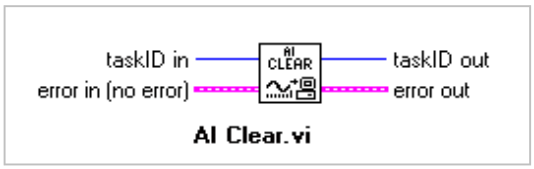

Figura A6.1.7 Librería AIClear.vi

- **Task ID in:** se utiliza para identificar el tipo de operación que se realizó.
- **Error in:** describe condiciones de error que ocurren antes de que el VI se ejecute. Si un error ha ocurrido ya, el VI retorna el valor de ese error en **error out**.
- **Task ID out:** tiene el mismo valor que **Task ID in**.

Librerías de salidas analógicas.

Al igual que antes, la descripción que se encuentra en las siguientes líneas se realiza por orden de ejecución.

1. AOConfig.vi

Lo primero que hace el programa es configurar una serie de parámetros para realizar la operación de escritura de datos. Con ello se configura el hardware y se asigna un buffer para la operación de salida de datos. Los parámetros que configurados se ven en la Figura A6.1.8.

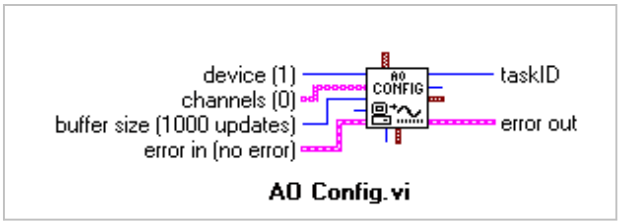

Figura A6.1.8 Librería AOConfig.vi

- **Device:** es el número que ha recibido la tarjeta en la configuración inicial.
- **Channels:** es una lista de los canales analógicos que se utilizarán para la generación de datos, es decir, es una "declaración" de los canales en los que escribiremos la señal generada.
- **Buffer size:** es el número de "updates" que almacena el buffer de salida. Es decir, define el tamaño del buffer de salida. Hay que tener en cuenta que el número de muestras de salida en un "update" es igual al número de canales de salida; por ejemplo, un pulso del reloj de "update" produce una escritura de una nueva muestra en cada canal de salida.
- **Error in:** describe condiciones de error que ocurren antes de que el VI se ejecute. Si un error ha ocurrido ya, el VI retorna el valor de ese error en **error out**.
- **Task ID:** es un número generado por Labview que identifica el tipo de operación que se realiza. Hay varios códigos de función, de entre todos ellos, los utilizados para realizar la aplicación son los siguientes:

Código de función	Operación de E/S
1	Entrada analógica
2	Salida analógica

Tabla A6.1.3 Códigos de función

2. AOWrite.vi

Lo siguiente que hace el programa es fijar varios parámetros para escribir los datos en el buffer de salida. Los parámetros que configura en este caso se ven en la siguiente figura:

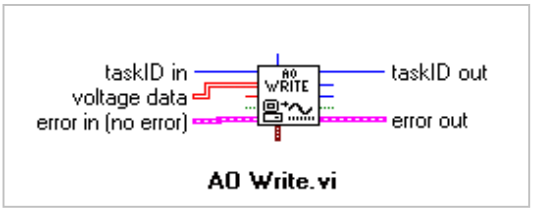

Figura A6.1.9 Librería AOWrite.vi

- **Task ID in:** se utiliza para identificar el tipo de operación que se realiza.
- **Voltage data:** es una matriz de 2D que contiene los datos que queremos enviar hacia el exterior. Cada fila contiene los datos de un "update", es decir, contiene un valor generado (muestra). Cada columna contiene todos los datos generados para un canal determinado.

	Canal de salida 0	**Canal de salida 1**	**…**
Update 1			
Update 2			
Update 3			
…			

Tabla A6.1.4 Matriz 2D con los datos a enviar

- **Error in:** describe condiciones de error que ocurren antes de que el VI se ejecute. Si un error ha ocurrido ya, el VI retorna el valor de ese error en **error out**.
- **Task ID out:** tiene el mismo valor que **Task ID in**.

3. AOStart.vi

Lo siguiente que hace el programa es fijar varios parámetros para iniciar la escritura de los datos en los respectivos canales de salida. Los parámetros que se configuran en este caso se ven en la Figura A6.1.10.

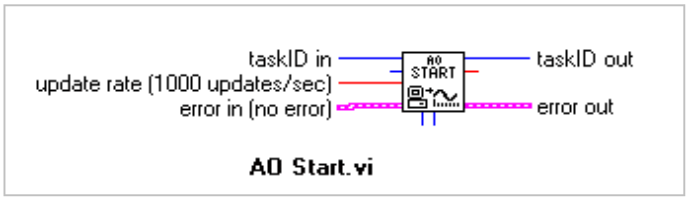

Figura A6.1.10 Librería AOStart.vi

- **Task ID in:** se utiliza para identificar el tipo de operación que se realiza.
- **Update rate:** hace referencia a la frecuencia con la que se escriben los datos en los canales de salida. Indica el número de "updates" que se escriben por segundo.
- **Error in:** describe condiciones de error que ocurren antes de que el VI se ejecute. Si un error ha ocurrido ya, el VI retorna el valor de ese error en **error out**.
- **Task ID out:** tiene el mismo valor que **Task ID in**.

4. AOClear.vi

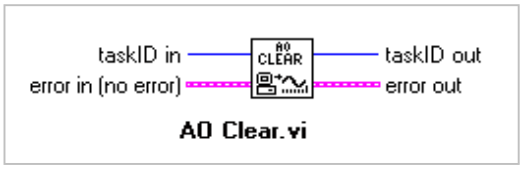

Figura A6.1.11 Librería AOClear.vi

Por último, lo que hace el programa es "limpiar" la asignación realizada con todos los buffers de salida y otros recursos utilizados para la escritura de datos. Los parámetros configurados en este caso se ven en la Figura:A6.1.11.

- **Task ID in:** se utiliza para identificar el tipo de operación que se realizó.
- **Error in:** describe condiciones de error que ocurren antes de que el VI se ejecute. Si un error ha ocurrido ya, el VI retorna el valor de ese error en **error out**.
- **Task ID out:** tiene el mismo valor que **Task ID in**.

SubVI Generación de señales (Funcgen.vi)

Este subprograma es el encargado de generar las señales que recibe el canal de salida. A continuación puede verse la interface de usuario del subprograma.

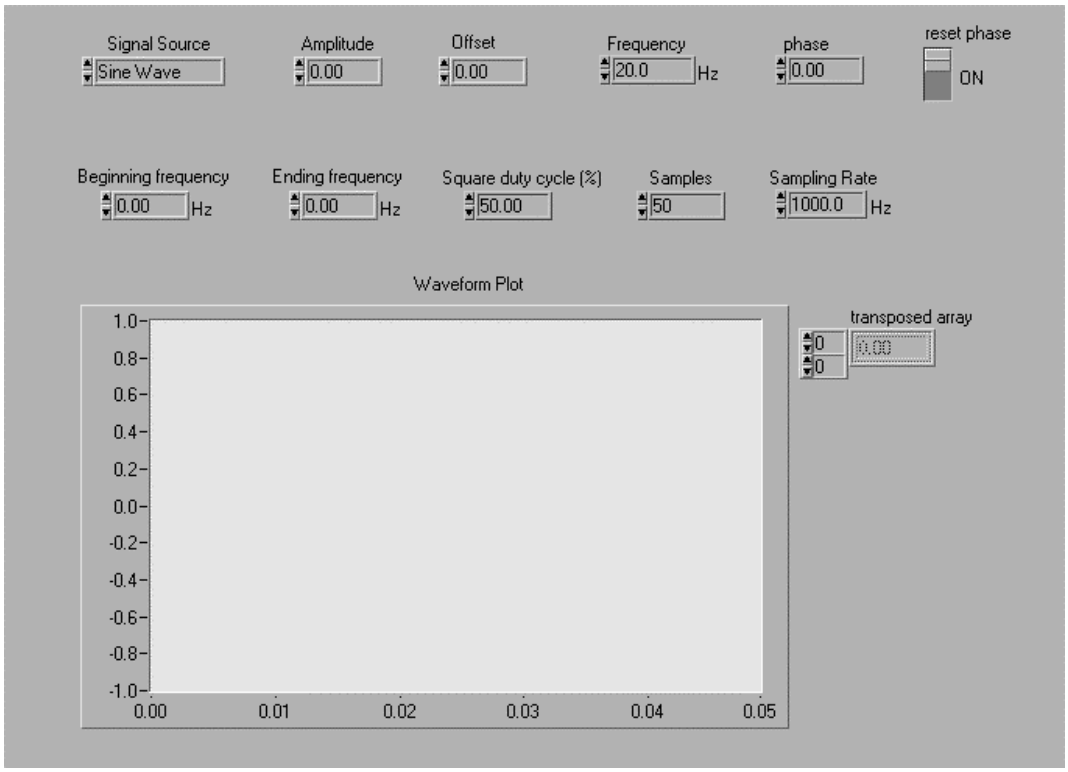

Figura A6.1.12 Panel frontal del Programa de generación de señales

Y también pueden verse sus diagramas de bloques para cada una de las señales que se generan:

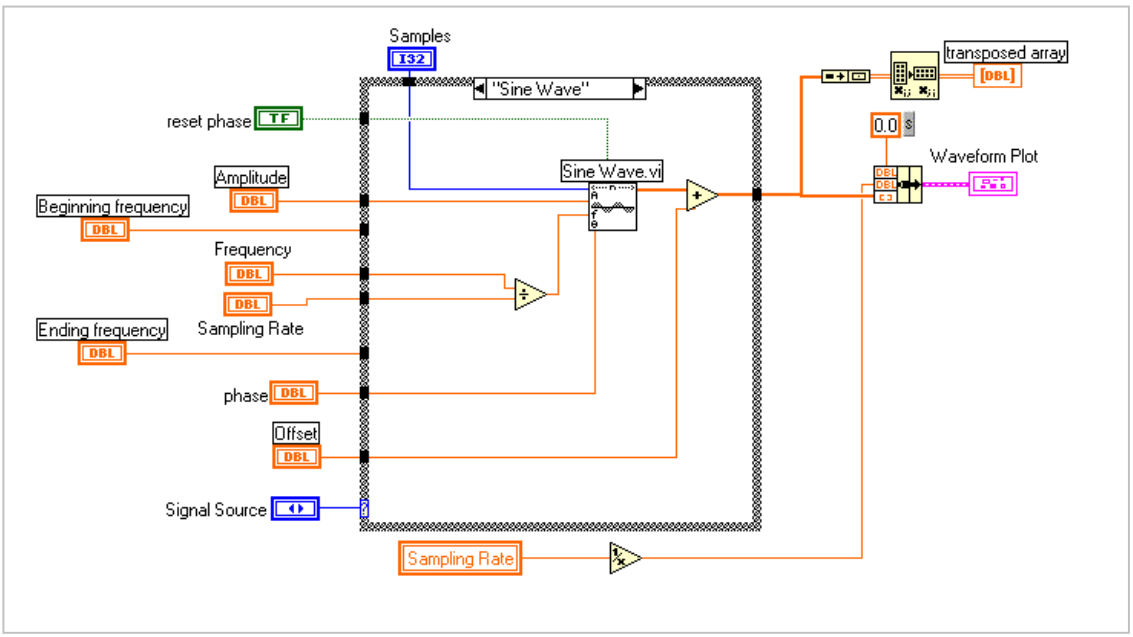

Figura A6.1.13 Diagramas de bloques del Programa de generación de señales: Señal Senoidal

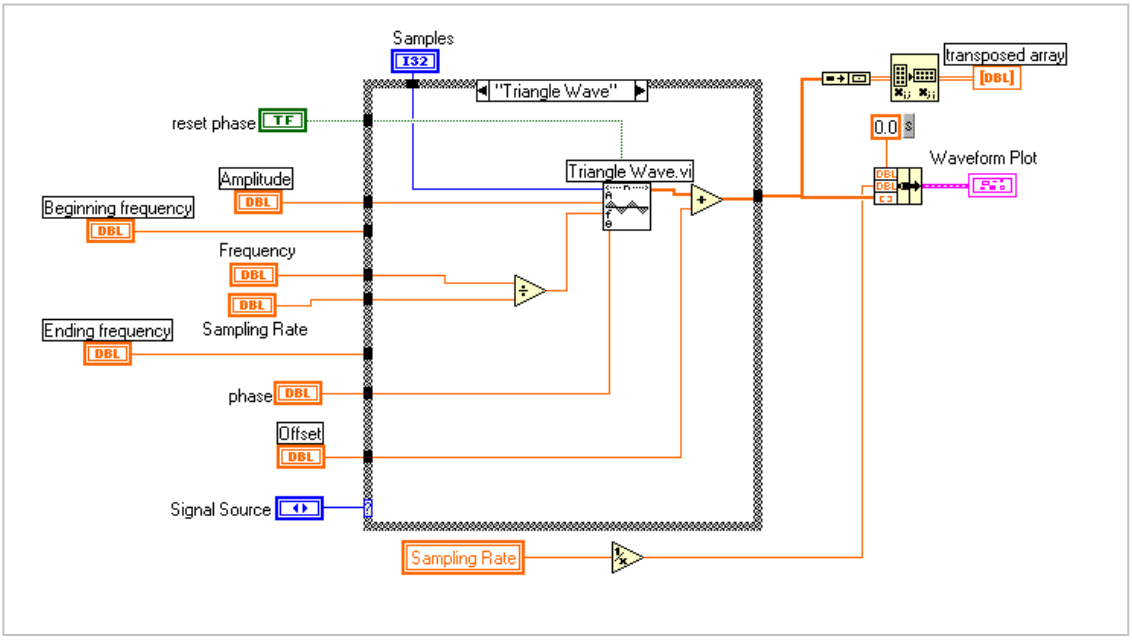

Figura A6.1.14 Diagramas de bloques del Programa de generación de señales: Señal Triangular

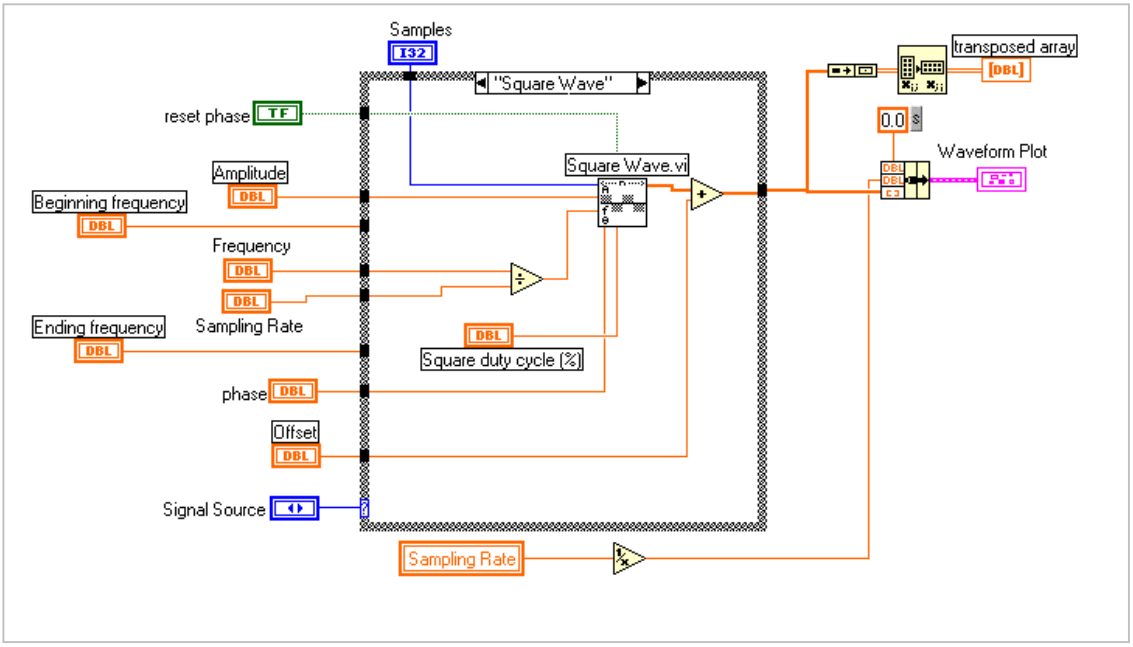

Figura A6.1.15 Diagramas de bloques del Programa de generación de señales: Señal Cuadrada

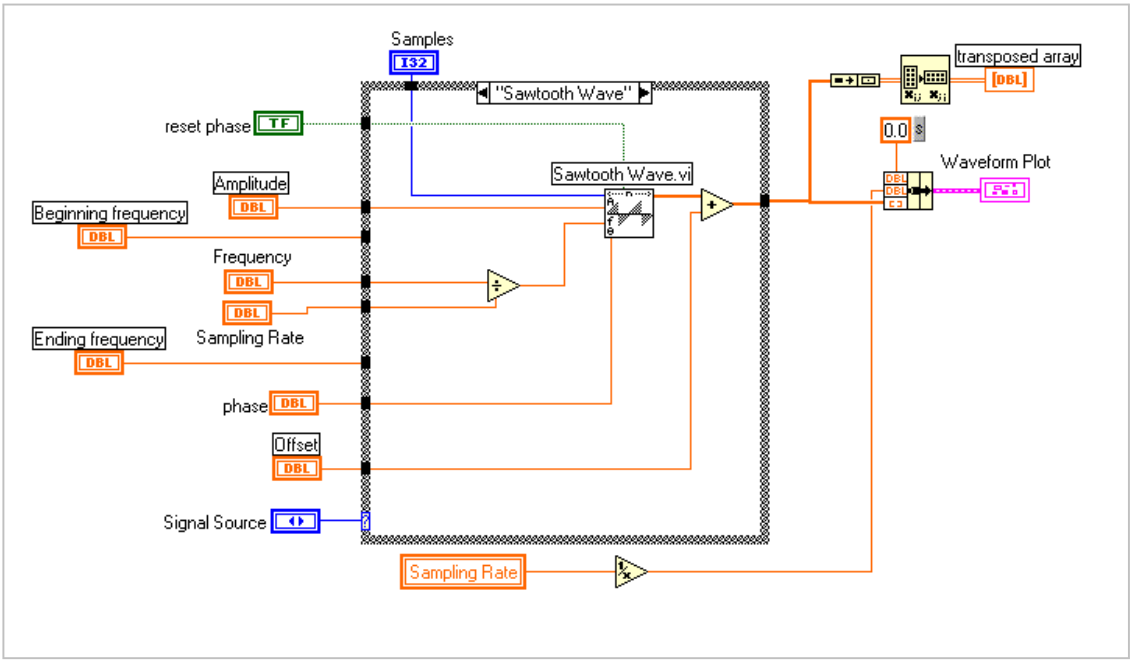

Figura A6.1.16 Diagramas de bloques del Programa de generación de señales: Señal Diente de sierra

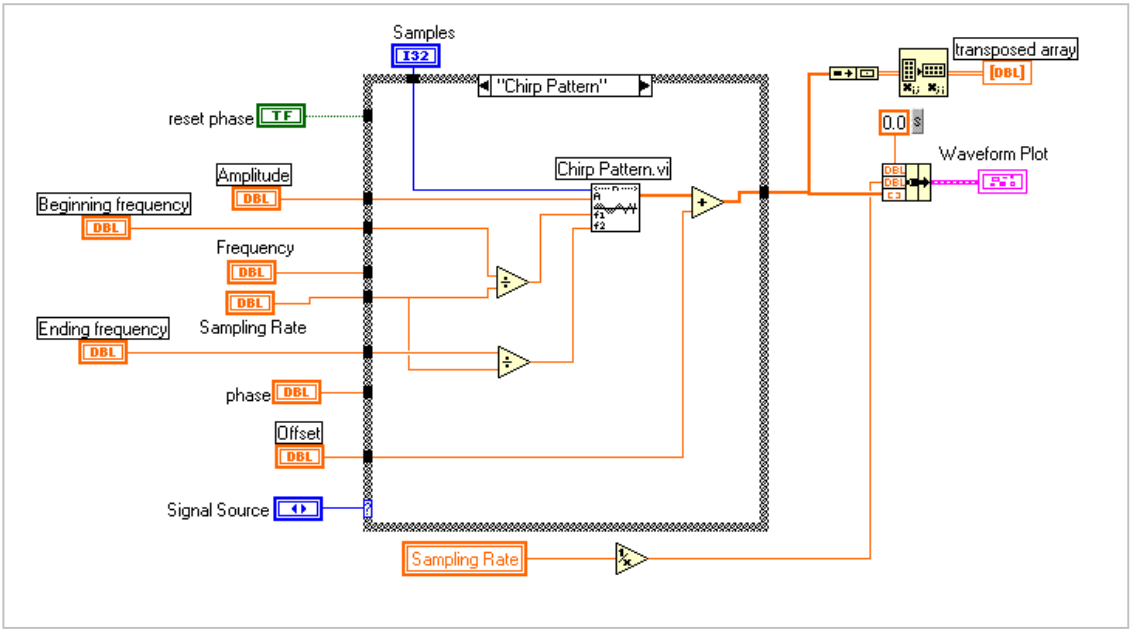

Figura A6.1.17 Diagramas de bloques del Programa de generación de señales: Señal Chirp

Como puede verse en los diagramas de bloques anteriores, el programa tiene la posibilidad de generar hasta cinco **tipos de señales** distintas, que son:

- senoidal,
- cuadrada,
- triangular,
- diente de sierra o
- chirp.

Tiene la posibilidad también de variar la **amplitud** y el nivel de **offset** de la señal, ambos expresados en Voltios, así como también su **frecuencia**, expresada ésta en Hz (ciclos/seg). Para el caso de la señal chirp, no se tiene en cuenta el valor del parámetro *Frecuencia*, sino el de las *Frecuencia inicial* y *Frecuencia final*, ambas también expresadas en Hz. Otro parámetro que es posible ajustar es el **sampling rate**, que hace referencia a la frecuencia de muestreo con la que se crea la señal. Conviene que este valor sea igual a los parámetros *scan rate* y *update rate* anteriormente descritos con el fin de que la forma real de la señal de salida se actualice con la misma frecuencia que se actualizan las lecturas de los canales de entrada.

Quizá de todos los aspectos de este programa, el más confuso sea el que hace referencia a como introducir en las librerías de generación de señal (por ejemplo, en la SineWave.vi) el parámetro de la frecuencia. El tipo de dato exigido por éstas es el de *frecuencia normalizada*. Sus unidades son ciclos/muestra, y se define de la siguiente forma:

$$\text{frecuencia normalizada}\left(\text{ciclos}/\text{muestra}\right) = \frac{\text{frecuencia analógica}\left(\text{ciclos}/\text{seg}\right)}{\text{frecuencia de muestreo}\left(\text{muestras}/\text{seg}\right)}$$

La frecuencia analógica hace referencia a la frecuencia de la señal que generamos, y se ajusta mediante el parámetro *Frecuencia*, mientras que la frecuencia de muestreo hace referencia a la frecuencia de las muestras en la señal que generamos, y se ajusta mediante el *sampling rate*.

Librerías de almacenamiento de datos

A la hora de almacenar los datos, se debe utilizar un subVI propio para ello. De todos los formatos disponibles para almacenar los datos, quizá el más idóneo es el que permite almacenarlo en formato hoja de cálculo, ya que se trata del formato más stándard para un posterior tratamiento de datos. Además, es compatible con Matlab, pues este programa dispone de una librería de comunicación que permite cargar en su zona de trabajo una matriz en formato hoja de cálculo Excel.

Esta librería es **Write To Spreadsheet File.vi**. Los parámetros de configuración se pueden ver en la siguiente figura:

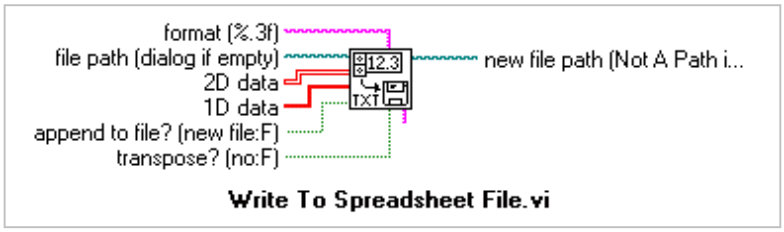

Figura A6.1.18 Librería Write To Spreadsheet File.vi

Los parámetros de configuración más relevantes son:

- **2D data:** este parámetro incluye los datos en formato de matriz de 2D.
- **transpose? (no: F):** este parámetro permite hacer la traspuesta de la matriz de datos que se almacenan. El programa se ha configurado para no hacer esta operación.
- **file path:** este parámetro permite definir la localización del fichero que se genera así como también el nombre de éste.

Los datos que se almacenan son tanto los generados como los capturados.