DISEÑO, IMPLEMENTACIÓN Y PUESTA A PUNTO DE UN SISTEMA DE CONTROL

PARA UN MOTOR DE CORRIENTE CONTINUA

ALUMNO: JESÚS BRUN CONEJOS
DIRECTOR: JORGE MÁS ESTELLES
INGENIERO TÉCNICO EN INFORMATICA DE SISTEMAS
25/08/2011

TABLA DE CONTENIDOS

1.RESUMEN

Este proyecto consiste en el diseño de un sistema para realizar el control de un motor de corriente continua. El proyecto se basa principalmente en la creación de una aplicación, que será la encargada de controlar el motor de corriente continua. Dicho motor se ha empleado para controlar un biorreactor utilizado en la regeneración de tejidos biológicos. La principal función de esta aplicación será realizar un muestreo de la intensidad consumida por el motor, al que estará conectada, para conocer la fuerza que hace el biorreactor. Para llevar a cabo el proyecto contaremos con una fuente de alimentación, que conectaremos al motor, y un adaptador para puerto GPIB, con el que conectaremos nuestra aplicación a la fuente de alimentación.

En esta memoria describiremos varios conceptos empleados para el desarrollo, así como el material utilizado y el código programado para la creación de la aplicación.

PALABRAS CLAVE:

Control Motor Intensidad Fuerza Corriente GPIB

2. INTRODUCCIÓN

El proyecto desarrollado de basa en el diseño e implementación de un sistema para el control de un motor de corriente continua, el cual se empleará para controlar un biorreactor utilizado en la regeneración de tejidos biológicos para fines terapéuticos.

Este motor de corriente continua será controlado a través de una fuente de alimentación, que le suministrará la corriente necesaria según las necesidades, así como también será la encargada de conocer los esfuerzos aplicados por el biorreactor. Para conocer este esfuerzo, el sistema debe ser capaz de saber en cada momento la intensidad consumida por el motor para poder obtener la fuerza ejercida.

La fuente empleada será conectada al sistema mediante el bus GPIB, por donde fluirá la información necesaria para establecer la comunicación. La fuente utilizada es una fuente programable mediante el estándar de ordenes SCPI empleado en aparatos de medida.

La aplicación, que se ha diseñado para llevar a cabo estas acciones, debe permitir programar el funcionamiento deseado sobre el biorreactor, así como la correcta toma de datos de fuerza aplicada, que serán almacenados en un fichero para que puedan ser posteriormente analizados. Esta aplicación se desarrollará en el lenguaje de programación C# utilizando el entorno de desarrollo Microsoft Visual Studio 2008.

En esta memoria primero realizaremos un repaso a los componentes y aparatos empleados en el proyecto, como son la fuente y al adaptador para la conexión de dicha fuente.
Luego veremos en qué se basa el funcionamiento del biorreactor, aunque esta parte no es competencia de este proyecto y veremos cómo funciona el estándar IEEE448, tanto a nivel físico como a nivel lógico. También repasaremos el control MSCOMM empleado en la comunicación con la fuente y explicaremos en qué se basa la aplicación diseñada para realizar el control, así como realizaremos una explicación de las fases del desarrollo llevadas a cabo en la realización del proyecto.

Por último, explicaremos cómo se emplea la aplicación diseñada y también explicaremos el código implementado para la creación de la aplicación.

3.FUENTE DE ALIMENTACIÓN

La fuente de alimentación emplea en el proyecto es el modelo PPT-1830 de la marca GW INSTEK. Esta es una fuente programable a través del puerto GPIB (IEEE448) mediante SCPI, que se trata de un estándar de comandos para instrumentos programables. Más adelante hablaremos sobre estos dos estándares.

3.1 Características de la fuente:

La fuente dispone de tres salidas, dos de ellas nos proporcionan de 0V a 18V con hasta 3A y una tercera salida que proporciona 0-6V con hasta 5A.

En la parte trasera encontramos el puerto GPIB, donde conectaremos nuestro adaptador para comunicarnos con la fuente.

1. Conector GPIB trasero de la fuente

En la parte frontal de la fuente vemos arriba una pantalla donde se nos muestra el voltaje, la intensidad y el canal que estamos utilizando. A la derecha tenemos un teclado numérico con el que se especifican algunos parámetros de configuración de la fuente. A la izquierda vemos en verde un botón que es el interruptor para apagar y encender la fuente. En la parte inferior se encuentran las conexiones de las 3 salidas de la fuente. En la parte central tenemos un conjunto de 12 botones con algunas de las funciones más utilizadas y principales que a continuación vamos a ampliar.

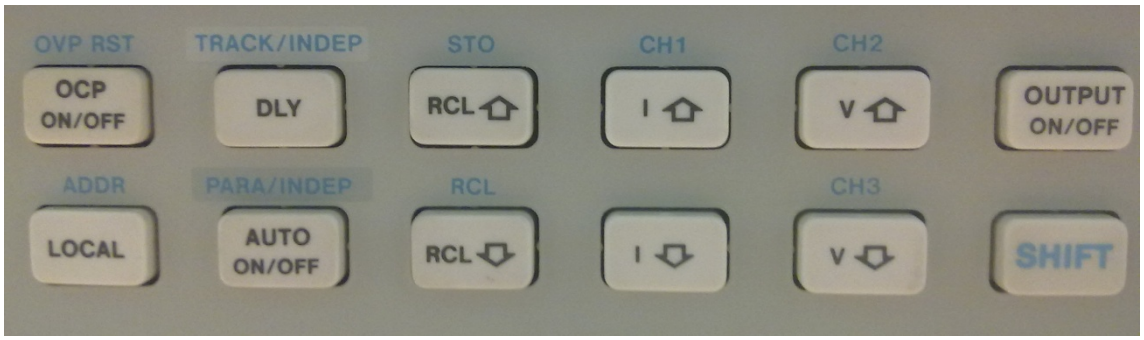
2 Frontal de la fuente

La fuente dispone de 2 modos: el local y el remoto. Si utilizamos el puerto GPIB para comunicarnos con la fuente, entonces se activará automáticamente el modo remoto y aparecerá en el display de la fuente las letras **RMT**. Con la fuente, en este modo, si pulsamos algún botón del panel frontal, no tendrá ningún efecto. Para trabajar con los botones del panel frontal la fuente debe estar en modo local, que más adelante veremos cómo se activa. Si no aparece en el display las letras RMT, es que estamos en el modo local.

3 Display con el modo remoto activado.

3.2 Conjunto de botones para las funciones principales

Vamos a ver las funciones de los botones que nos serán necesarios.



OUTPUT ON/OFF: Activa y desactiva las salidas de la fuente. Si no activamos las salidas, aunque hayamos configurado ya la corriente y el voltaje, no habrá salida real en la fuente. Este botón sí que funciona aunque estemos en modo remoto.

SHIFT: El botón SHIFT sirve para poder activar la segunda función que tienen algunos botones de este panel. Estos botones tienen la segunda función indicada en azul encima. Para activar estas funciones hay que pulsar primero SHIFT y luego el botón correspondiente.

I: Estos dos botones sirven para modificar el límite máximo de corriente que puede suministrar la fuente. La variación aparecerá en el display de la fuente que corresponde a la intensidad. El botón con la flecha hacia arriba aumenta la intensidad y con la flecha hacia abajo la disminuye. Si hemos pulsado antes SHIFT, el botón con la flecha hacia arriba pone a la fuente en el canal 1.

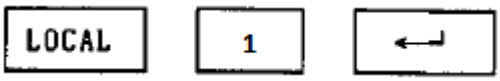
V: Formado también por 2 botones, sirven para modificar el voltaje que suministrará la fuente. La variación aparecerá en el display de la fuente que corresponde al voltaje. El botón con la flecha hacia arriba aumenta el voltaje y con la flecha hacia abajo la disminuye. Si hemos pulsado SHIFT antes estos botones ponen a la fuente en el canal 2 o 3.

LOCAL: Este botón hace que la fuente vuelva al modo local. Para que tengan efecto los botones de este panel, la fuente debe estar en este modo. Si hemos pulsado SHIFT antes, activaremos la función ADDR, que sirve para cambiar la dirección que tendrá la fuente en el bus GPIB y que servirá para comunicarnos de forma remota.

3.3 Modificación de la dirección de la fuente

Para modificar la dirección de la fuente dentro del bus GPIB, debemos pulsar el botón **SHIFT** y luego el botón **LOCAL,** o si estamos trabajando en modo Local, con pulsar directamente el botón **LOCAL** bastará. Una vez realizado esto, en el display nos aparecerá la dirección en la que está configurada la fuente.

Nosotros hemos de trabajar con la fuente en la dirección 1, por tanto para modificar este valor tenemos que pulsar el botón 1 del teclado numérico de la derecha y luego la flecha que aparece también en dicho teclado numérico. Con estos pasos ya tendremos configurada la fuente en la dirección 1.



Una vez configurada, no será necesario volver a configurarla cada vez que la encendamos porque la fuente guarda nuestra configuración.

3.4 Comandos SCPI

Para controlar la fuente de alimentación mediante el puerto GPIB se utilizan comandos SCPI. El conjunto de comandos para la fuente empleada son los siguientes:

COMMAND	RANGE	DESCRIPTION
[VOURce:] VOLTage	EFER TABLE 2.1	
[SOURce:] VOLTage?		
[SOURce:] VOLTage:PROTection	EFER TABLE 2.1	
[SOURce:] VOLTage:PROTection?		
[SOURce:] VOLTage:PROTection [:LEVel] ?		
[SOURce:] CURRent	EFER TABLE 2.1	
[SOURce:] CURRent?		
[SOURce:] CURRent:PROTection:STATe	1/0(on/off)	
[SOURce:] CURRent:PROTection:STATe?		
[SOURce:] CHANnel	1-3	triple output only
[SOURce:] CHANnel?		triple output only
[SOURce:] DTIMe?		
[SOURce:] DTIMe	0.0-255.59	(minute.second)
SYSTem:ERRor?		
SYSTem:VERSion?		
SYSTem:MEMory?		
SYSTem:AUTo	1/0(on/off)	
SYSTem:AUTo?		
SYSTem:TRAcking	1/0(on/off)	triple output only
SYSTem:PARAllel	1/0(on/off)	triple output only
STATus:OPERation?		
STATus:OPERation [:EVENt] ?		
STATus:OPERation:CONDition?		
STATus:OPERation:ENABle?		
STATus:OPERation:ENABle	0-65535	
STATus:QUEStionable?		
STATus:QUEStionable [:EVENT] ?		
STATus:QUEStionable:CONDition?		
STATus:QUEStionable:ENABle	0-65535	
STATus:QUEStionable:ENABle?		
STATus:PRESet		
STATus:QUEUe:ENABle		
OUTPut:STATe	1/0(on/off)	
OUTPut:STATe?		
OUTPut:PROTection:CLEar		
MEASure:VOLTage?		
:CURRent?		

En el proyecto utilizamos los siguientes comandos:

SOURce:VOLTage <valor>

Con este comando indicamos el voltaje que ha de suministrar la fuente.

MEASure:CURRent?

Con este comando pedimos a la fuente la corriente que se está consumiendo en ese instante.

OUTPut:STATe 1/0

Con este comando activamos con un 1 o desactivamos con un 0 la salida de la fuente de alimentación.

4.ADAPTADOR GPIB-USB

4.1 Adaptador

Para la comunicación entre el ordenador y la fuente programable hemos empleado un adaptador GPIB-USB modelo SMART488 de la casa ALCIOM.

La utilización de un adaptador con conexión USB al ordenador nos permite poder conectar la fuente a cualquier ordenador sin que tengamos que tener que utilizar ni tarjeta PCI ni ningún tipo de hardware aparte.

Lo más importante de este adaptador es que nosotros lo vemos como un puerto serie virtual, por tanto lo programaremos como si fuera un puerto RS-232 con sus respectivas librerías y componentes. Esto hace que el proceso sea trasparente para nosotros y se encargue el adaptador de realizar las distintos tareas necesarias para realizar una comunicación GPIB.

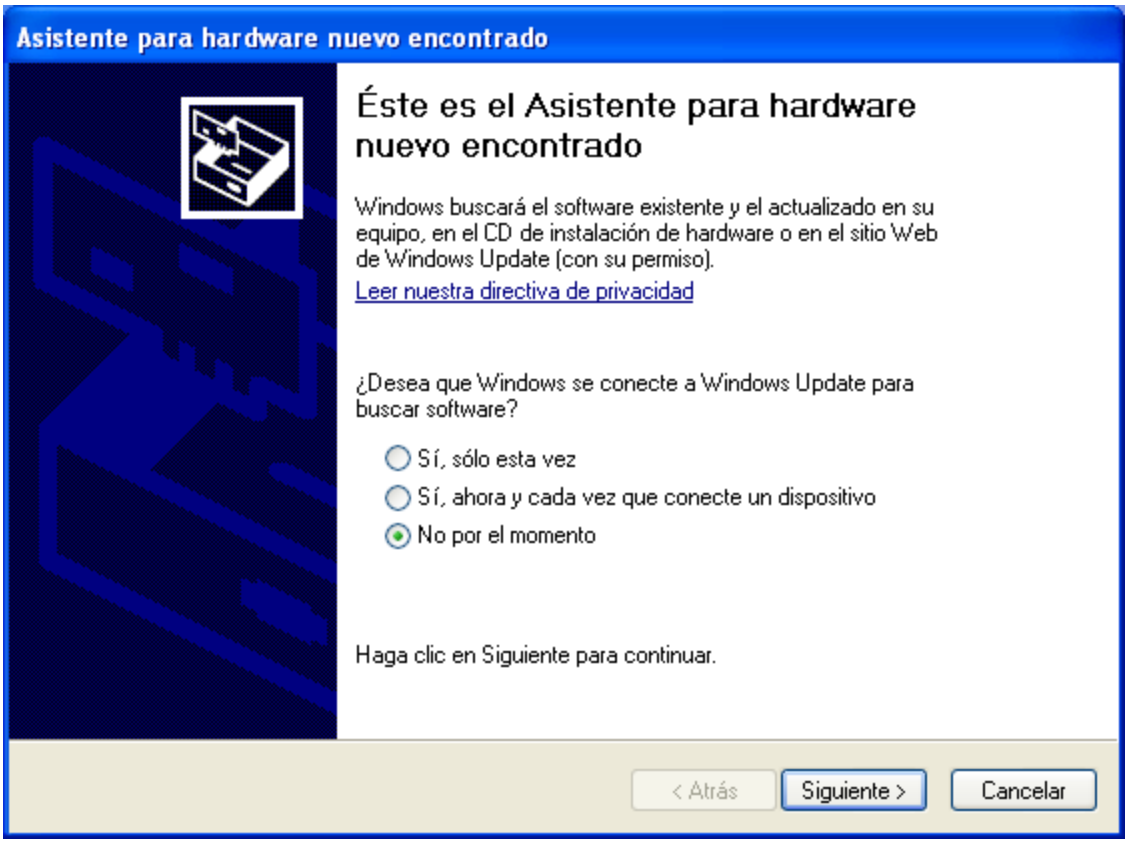
El adaptador contiene una serie de parámetros de configuración, los cuales se pueden modificar; pero para nuestro programa no debemos modificar ningún parámetro, ya que podría dejar de funcionar correctamente nuestro programa. Un parámetro modificable es, por ejemplo, la dirección del dispositivo que conectaremos, que en nuestro caso será la fuente de alimentación, que será la dirección 1.

Se trata de un dispositivo plug and play, por lo que lo único que hemos de hacer es conectarlo e instalar sus driver de la forma que explicamos a continuación.

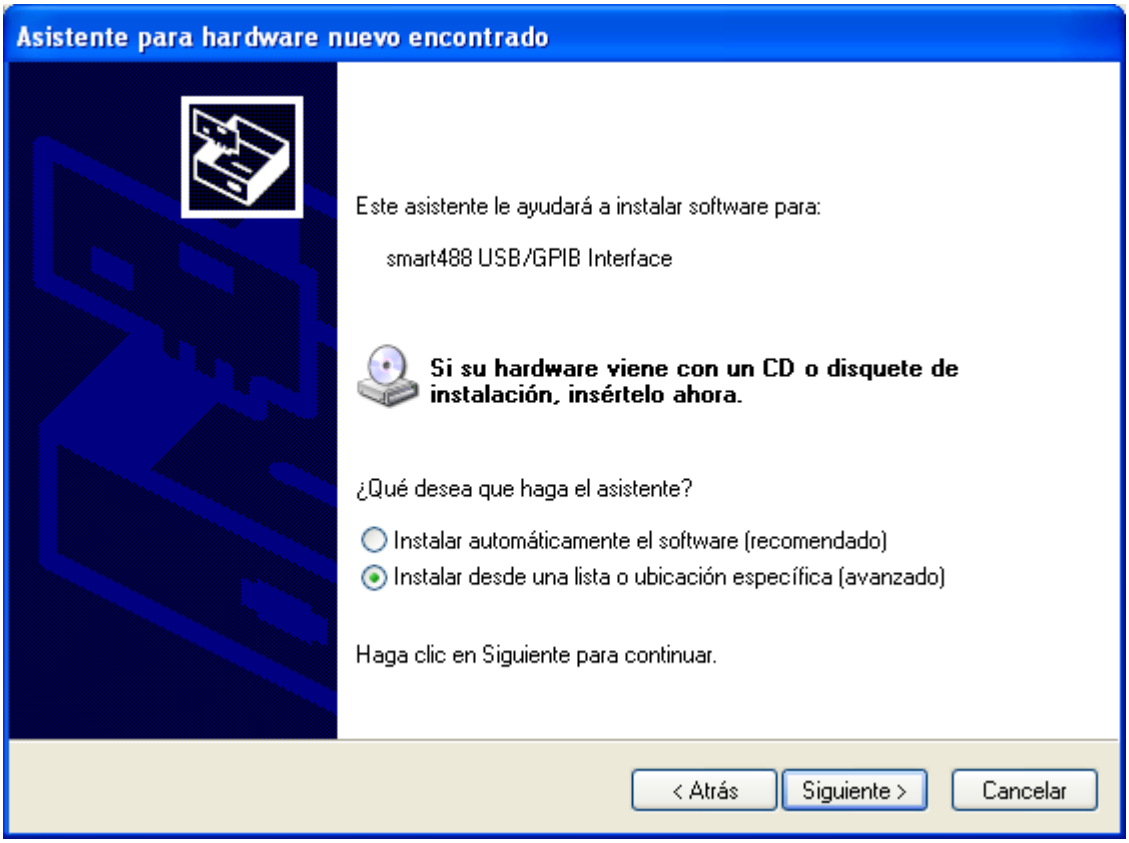
4.2 Instalación de los drivers del adaptador

1. Conectamos el adaptador al puerto USB que vayamos a emplear, ya que si luego lo conectamos a un USB diferente al que lo hayamos instalado, habrá que volver a instalar el adaptador para ese nuevo puerto USB.

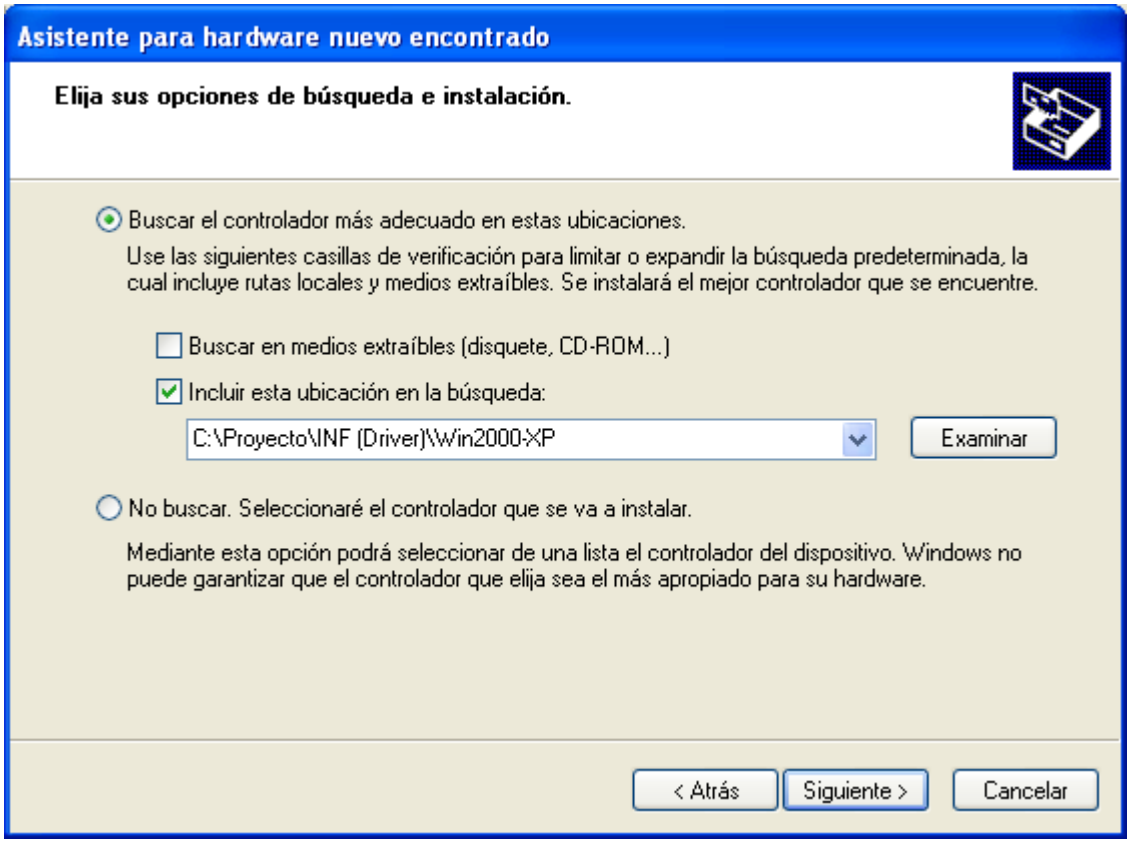
2. Una vez conectado, aparecerá un mensaje de nuevo dispositivo encontrado y nos aparecerá una ventana como esta, donde seleccionaremos ***No por el momento***



3. En la siguiente pantalla seleccionamos ***Instalar desde una lista o ubicación específica (avanzado)***

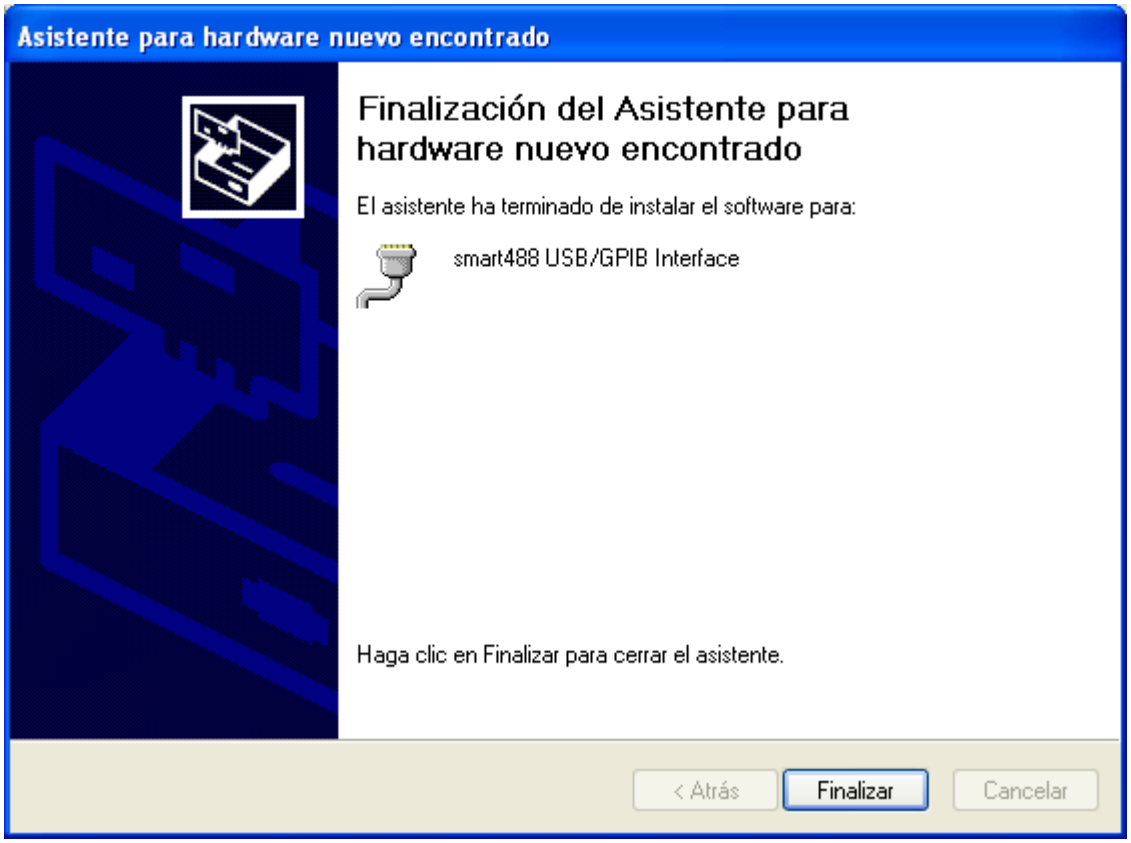


4. Luego indicamos la ubicación de los driver para windows xp, que están en la carpeta: **\anexos\Drivers\INF (Driver)\Win2000-XP**



5. Ahora esperamos a que se instalen los drivers. Si nos aparece una ventana como la siguiente, le damos a continuar.

6. Cuando finalice, ya podremos utilizar nuestro adaptador.



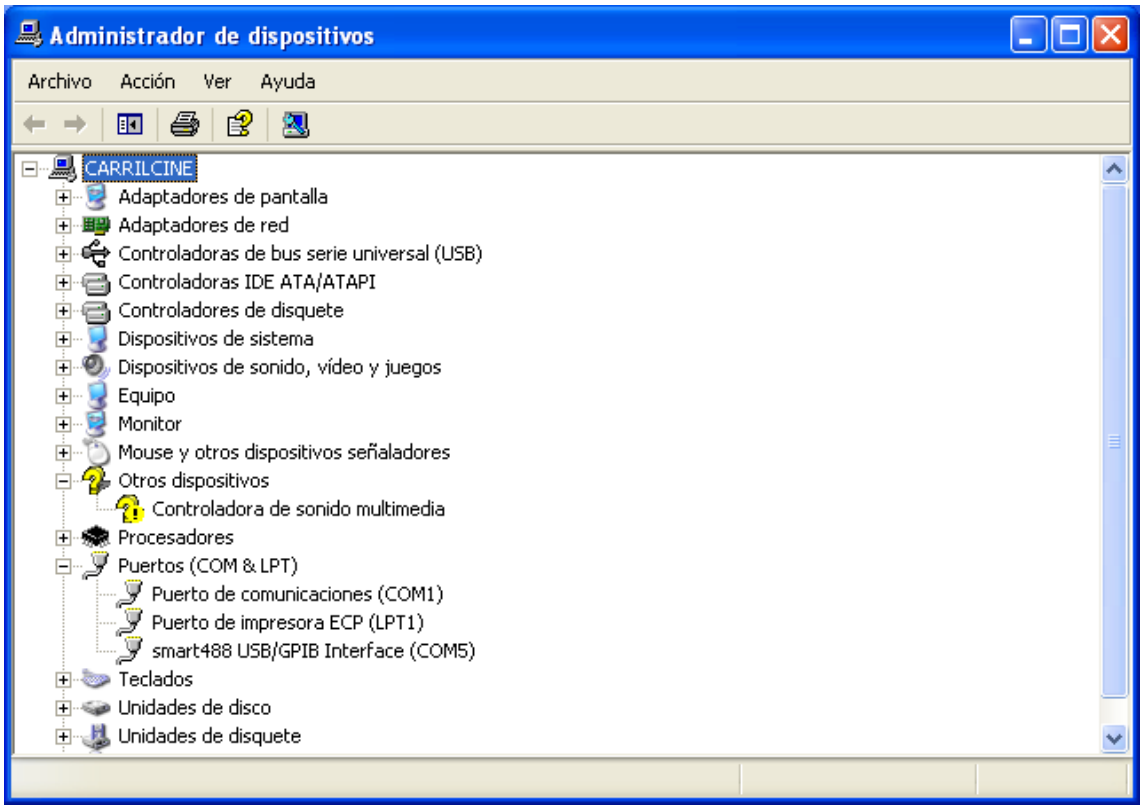
4.3 Conocer el puerto COM donde se ha instalado el adaptador

Como hemos comentado, el adaptador se instala como un puerto serie virtual; por tanto tiene una dirección COM, la cual hemos de conocer para después introducírsela al programa.

Para conocer esta dirección, primero tendremos que tener el adaptador conectado al ordenador y con los drivers ya instalados, y luego ir a:

Panel de control →Sistema →Pestaña →Administrador de dispositivos

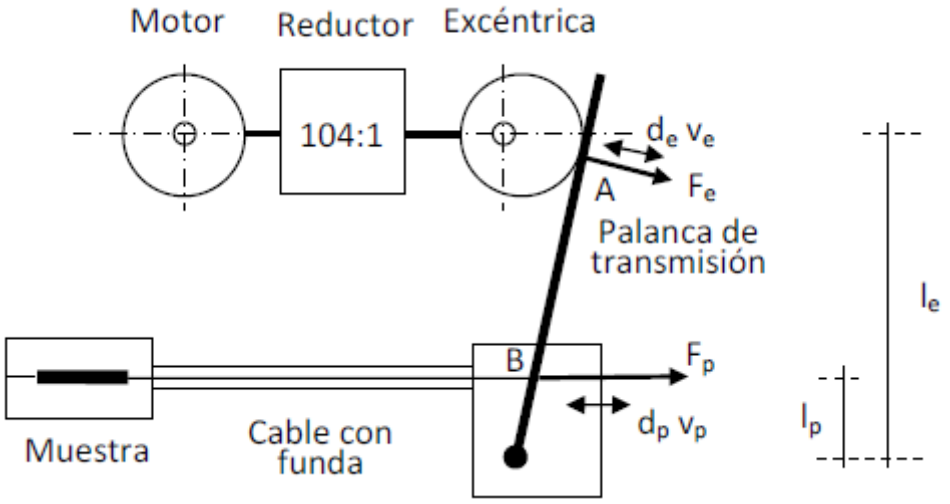
Luego, dentro de *Administrador de dispositivos,* hemos de ir a la sección *Puertos (COM & LPT)* y ahí nos aparecerá una entrada con el nombre del adaptador. En dicha entrada aparecerá, al final del nombre entre paréntesis, el número del puerto COM que en el caso de la imagen es el 5.



5.EL BIORREACTOR

Aunque la parte relacionada con el biorreactor no entra en este proyecto, vamos a incluir la explicación de cuál es su funcionamiento.

El biorreactor construido en el CBIT de la UPV consta de un motor eléctrico de corriente continua de imán permanente alimentado por una fuente de alimentación a una tensión V ajustable entre 0 y 12 V. El motor consume una corriente I relacionada con el par que desarrolla en cada instante, M_m. La corriente consumida sin muestra (corriente en vacío, I_0) es la corriente necesaria para vencer todos los rozamientos de la cadena cinemática. La velocidad de giro del motor w_m está relacionada con la fuerza



contraelectromotriz del motor, e'. La resistencia interna del motor, en nuestro caso, es de R_i=0,4 Ω. Finalmente, el parámetro k' que relaciona par con corriente y velocidad de giro con fuerza contraelectromotriz es k'=0,0155 Nm/A ó Vs.
Los valores de R_i y k' han sido calculados de forma experimental.

Las ecuaciones del motor son:

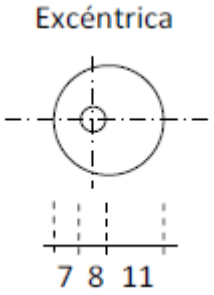
$$M_m = k'(I - I_0) \qquad \varepsilon' = V - IR_i = k'\omega_m$$

El motor mueve un reductor, cuya salida va conectada a una excéntrica, que mueve la palanca de transmisión. La relación de reducción del reductor es de r=104 a 1, por lo que el par y la velocidad de giro de la excéntrica serán:

$$M_e = rM_m \qquad \omega_e = \frac{\omega_m}{r}$$

La frecuencia f del movimiento aplicado a la prótesis de tendón es $f = \frac{\omega_e}{2\pi}$

Y el período T_e de dicho movimiento, $T_e = \frac{1}{f} = \frac{2\pi}{\omega_e}$.



Las dimensiones de la excéntrica determinan la amplitud del movimiento. Si su geometría es la del dibujo (valores en mm), la amplitud del movimiento del punto de la palanca de transmisión en contacto con ella (A) es d_e e = 4mm . La velocidad de dicho movimiento (asumiendo una gráfica desplazamiento frente a tiempo en forma de diente de sierra) es

$$v_e = \frac{2d_e}{T_e} = 2d_e f$$

Si consideramos la conservación de la potencia entre el movimiento de giro de la excéntrica y el movimiento alternativo, podemos calcular la fuerza F_e que la excéntrica produce en la palanca de transmisión en el punto A:

$$F_e v_e = M_e \omega_e \Rightarrow F_e = \frac{M_e \omega_e}{v_e} = \frac{r \cdot k'(I - I_0)2\pi}{2d_e f T_e} = \frac{r \cdot k'(I - I_0)\pi}{d_e}$$

Teniendo en cuenta las dimensiones l_e y l_p podemos calcular la fuerza aplicada a la muestra colocada en el biorreactor F_p:

$$F_e l_e = F_p l_p \Rightarrow F_p = F_e \frac{l_e}{l_p} = \frac{r \cdot k'(I - I_0)\pi}{d_e} \frac{l_e}{l_p}$$

Así pues, la fuerza aplicada a la prótesis sólo depende de la corriente consumida por la fuente que alimenta el motor y una serie de parámetros geométricos conocidos. Si consideramos los valores conocidos

$$r = 104 \qquad k' = 0{,}0155 \frac{N \cdot m}{A} \qquad I_0 = 0{,}8\ A \qquad \frac{l_e}{l_p} = 2 \qquad d_e = 4\ mm$$

entonces resulta que F_p=2532(I-0,8) N

Si la corriente en un instante dado es, por ejemplo, de 0,9 A, la fuerza aplicada al biorreactor sería de Fp=253 N; si hay colocadas dos muestras, cada una de ellas soportaría unos 125 N.

En realidad, estos valores deben ser minorados porque la cadena de transmisión no es completamente rígida, y al existir holguras, la fuerza realmente aplicada a la muestra es menor.

6. IEEE 448

Como hemos comentado antes, la comunicación con la fuente la realizamos mediante el bus GPIB. Veamos las características de este estándar.

6.1 INTRODUCCIÓN

GPIB es un estándar de conexión que permite la comunicación de un ordenador con instrumentos electrónicos de medida, como pueden ser fuentes de alimentación, osciloscopios, etc. Las siglas corresponden a General Purpose Interface Bus, pero a pesar de este nombre, fue diseñado específicamente para la conexión de instrumentos de medida.

La funcionalidad del estándar GPIB ha evolucionado a lo largo del tiempo y se encuentra descrito en las siguientes especificaciones:

- **IEEE 488.1 (1975):** Especificación que define las características de nivel físico (mecánico y eléctrico), así como sus características funcionales básicas.

- **IEEE 488.2 (1987):** Especificación que define las configuraciones mínimas, los comandos y formatos de datos básicos y comunes a todos los equipos, el manejo de errores y los protocolos que se siguen en las comunicaciones.

- **SCPI (Standard Commands for Programmable Instrumentation):** Especificación construida sobre el estándar IEEE 488.2, que define una estructura de comandos estándar aceptados por múltiples instrumentos de muchos fabricantes.

Componentes que se utilizan en el control de la instrumentación por GPIB

- **Equipo:** Instrumento del entorno que se controla. Debe estar dotado con una tarjeta hardware de conexión al bus GPIB. Para su control dispone de software interno de control que interpreta los mensajes que recibe por el bus GPIB e interacciona con el firmware propio del equipo. El “Parser” es el thread de gestión del intercambio de mensajes por el bus GPIB.

- **Nivel Físico (IEEE 488.1):** La comunicación entre los equipos se basa en un bus físico, compuesto por un conjunto de líneas con niveles lógicos bien definidos y con protocolos de comunicación basados en los estados lógicos de las líneas.

- **Nivel Operativo (IEEE 488.2):** El protocolo operativo básico dentro del que se encuadra el intercambio de información, datos e instrucciones básicas de control.

- **Driver GPIB (SCPI):** El computador interacciona con el bus GPIB a través de una tarjeta de control hardware que resuelve y atiende los dos protocolos anteriores. El propio fabricante ofrece una interfaz software implementada por un conjunto de funciones que permiten el acceso de los programas a la funcionalidad del bus. Puede ofertar una interfaz constituida por una librería de funciones que corresponde al lenguaje SCPI.

Características de Bus GPIB

Las características más relevantes de este bus GPIB son las siguientes:

a) Permite la interconexión de hasta 15 equipos, de los que uno de ellos es el controlador, que establece la función que debe ejercer cada uno de los otros.
b) Un dispositivo conectado al bus, puede enviar o recibir información hacia o desde cualquiera de los otros 14 equipos. A veces, la propia naturaleza de un equipo hace que sólo esté capacitado para recibir (p.e. una impresora), o sólo capacitado para enviar (p.e. un contador), o ambas cosas (p. e. un osciloscopio, o un computador)
c) El límite práctico de velocidad de intercambio de datos es de 500 Kbytes/s (o lo que es lo mismo 4 Mbits/s).
d) La interconexión entre equipos se realiza utilizando cables de 25 hilos, finalizados en conectores de doble boca (macho por un lado y hembra por el otro), que permite la interconexión de los equipos en cualquier configuración (estrella, línea, o cualquier combinación de ellas).
e) Las longitudes máximas permitidas en los cables es de 20 metros. Los cables que se comercializan son de 1, 2, 4 y 8 metros.

6.2 BUS GPIB NIVEL FISICO (IEEE-488.1)

Estructura del Bus GPIB

El bus GPIB está basado en 16 líneas activas, además de la tierra. Estas 16 líneas se organizan en tres buses:

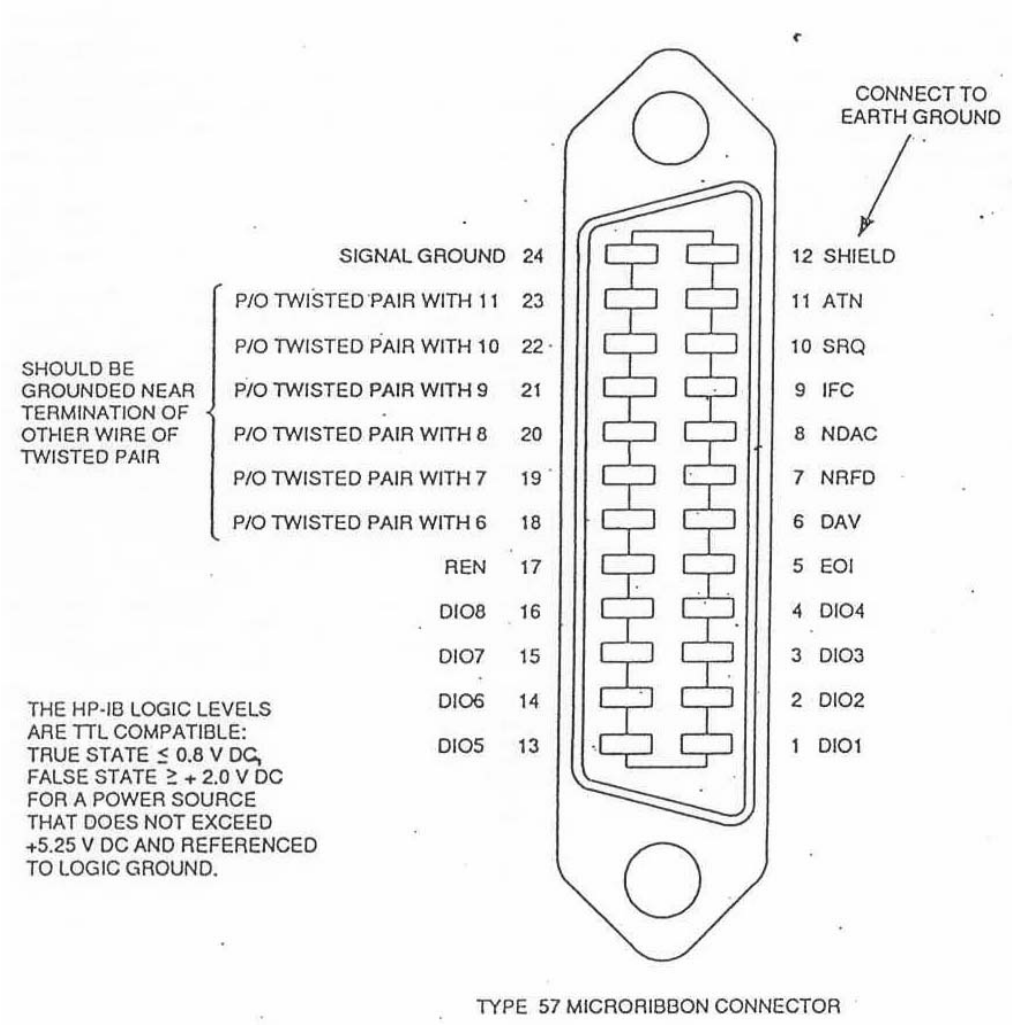
- **Bus de Datos (DIO1-DIO8)** (Data input/output): Es un bus bidireccional de 8 líneas orientado a la transferencia de bytes o de caracteres ASCII.

- **Bus de sincronización de la transferencia de datos:** Es un conjunto de tres líneas (**DAV**: Data valid, **NRFD**: Not Ready For Data y **NDAC**: Not Data ACcepted)

que se utilizan de forma coordinada para asegurar la transferencia de datos entre los equipos.

- **Bus de control**: Está constituido por 5 líneas (**ATN**: ATteNtion, **IFC**: InterFace Clear, **SRQ**: Service ReQuest, **REN**: Remote ENable, y **EOI**: End Or Identify) que se utilizan para transferir comandos entre los equipos, relativos al modo de interpretar los datos que se transfieren, o comandos básicos de gobierno de la interfaz del bus.

Conector estándar de acceso al Bus GPIB



Significado de las líneas del Bus GPIB

DATA LINE (DIO1 - DIO8):

Las 8 líneas de datos permiten que el único equipo establecido como "talker" (o en otros casos el "controller") envíe un byte en paralelo hacia todos aquellos equipos que en ese instante estén definidos como "listener".

DATA VALID (DAV):

Es una de las líneas de sincronización que permite la transferencia de datos por el bus, gobernada por los equipos establecidos como “talker”. Un TRUE lógico en esta línea significa que el equipo establecido como "talker" activo ha establecido unos datos válidos sobre el bus de datos, que deberán ser leídos por todos los equipos establecidos como "listener".

NOT READY FOR DATA (NRFD):

Es otra de las líneas de sincronización, que es gobernada por los equipos establecidos como "listener". Cuando esta línea está en estado lógico TRUE, significa que algún equipo de entre los "listener" no está aún dispuesto para aceptar nuevos datos. El que esta línea se encuentre en estado lógico TRUE inhibe al equipo "talker" a que inicie el envío de un nuevo dato. El que esta línea esté en estado lógico FALSE, significa que todos los equipos "listener" se encuentran a la espera de un nuevo dato, momento en el que el “talker” puede establecer el dato en el bus.

NO DATA ACCEPTED (NDAC):

Es la tercera línea de sincronización de datos, y es gobernada por los equipos que están establecidos como "listener". Cuando se encuentra en estado lógico TRUE, significa que alguno de los equipos establecidos como "listener" aún está pendiente de leer un dato, y en consecuencia, el “talker“ debe esperar aún para retirar los datos. Cuando esta línea se encuentra en estado lógico FALSE significa que ya todos los equipos establecidos como "listener" han leído el dato transferido y por tanto el “talker” puede retirar el dato del bus.

ATTENTION (ATN):

Es una señal que establece el "controller" para establecer, con un estado lógico TRUE en ella, que el dato que se envía por el bus de datos es un comando enviado por el "controller”. Cuando esta línea toma el estado lógico FALSE, indica que el byte del bus de datos debe ser considerado como un dato.

INTERFACE CLEAR (IFC):

Está bajo el exclusivo control del "system controller". Cuando es establecido en esta línea un estado lógico TRUE, todos los equipos conectados al bus deben ser reseteados, y todos ellos deben pasar a su estado base.

SERVICE REQUEST (SRQ):

Es utilizado por los equipos conectados al bus para comunicar al "controller" que requieren ser atendidos por alguna causa (ha concluido una actividad, se ha producido un error, existe algún dato para transferir, etc.). Cuando el "controller" detecta un estado lógico TRUE en esta línea, debe iniciar una encuesta (polling) para determinar qué equipo causó el requerimiento, y en el caso de que proceda, satisfacer su demanda.

REMOTE ENABLE (REN):

Es una línea con la que el "controller", al establecerla a un estado lógico TRUE, habilita a todos los equipos conectados al bus para que reciban datos o comandos.

END OR IDENTIFY (EOI):

Esta línea tiene dos funciones:

a) En primer lugar, el "talker" puede comunicar, poniendo a estado lógico TRUE esta línea, que concluye su envío de datos.

b) En segundo lugar, esta línea es utilizada por el "controller" para iniciar una encuesta paralela. En este caso el "controller" debe poner simultáneamente a estado lógico TRUE las señales ATN y EOI, y como respuesta a ello, los equipos que previamente hayan sido configurados para participar en la encuesta paralela, transfieren su bit de status sobre el bus.

Tipos de mensajes que intercambian los equipos

Entre los equipos conectados al bus GPIB se transfieren mensajes constituidos por secuencias de byte por transacción. De acuerdo con el estado de la señal de control "ATN", existen dos tipos de mensajes:

"Data": Mensaje que contiene información relativa a la funcionalidad de un equipo. Ejemplos son: instrucción de programación, resultado de medida, estatus de un equipo, etc.

"Command": Mensaje que tiene como función controlar el modo de operación del bus. Ejemplos son: Inicialización del bus, cambio del modo de operación de un equipo, transferencia del control, etc.

Modos de operación de un equipo

En cada momento, un equipo conectado al bus GPIB puede estar operando como uno o varios de los siguientes modos de comportamiento:

"Controller": Con capacidad de establecer quien envía o recibe datos y el modo de operación del bus (solo un equipo puede ser "controller").

"Talker": Con capacidad de enviar datos a otros equipos.
"Listener": Con capacidad de recibir datos de otros equipos.
"Idler": Sin ninguna capacidad respecto del bus.

"CONTROLLER": Equipo con capacidad de transferir mensajes de tipo Command a los otros equipos. Existen dos tipos de controller:

"**System Controller**": Tiene capacidad hardware de tomar el control del bus en todo momento, a través de las líneas "IFC" y "REN". En un bus sólo puede existir un único System Controller y está caracterizado por tener las capacidades hardware especiales de poder establecer el estado de las líneas "IFC" y "REN".

"Active controller": Tiene la capacidad de transferir mensajes de tipo Command para:

- Establecer los modos de operación "Listener" y "Talker" en los restantes equipos.
- Enviar los comandos de inicialización y sincronización del bus.
- Supervisar mediante encuesta el status de los equipos.

Cada bus puede tener conectado uno o más dispositivos capaces de asumir la función de "active controller", aunque en cada momento, solo uno de esos equipos puede operar como tal.

En una situación estándar, hay un computador conectado al bus que actúa a la vez como "system controller" y como único "active controller".
En algún momento, un equipo puede requerir del controlador del sistema su interés en convertirse en controlador activo, a fin de llevar a cabo una operación compleja, tal como transferir unos datos al "plotter", o almacenar un fichero en un disco, etc.. Como respuesta a este requerimiento el controlador de sistema transferirá el control al equipo que lo ha solicitado, el cual pasa a constituirse en controlador activo del bus. Cuando concluye su operación, retorna de nuevo el control del bus, al controlador de sistema.

Obsérvese que las capacidades propias de controlador de sistema no pueden ser transferidas.

Un sistema construido sobre el bus GPIB puede ser configurado en uno de los siguientes tres modos:

- **Sin "controller**" : En esta configuración, uno de los equipos debe tener capacidad para actuar solo como "talker", y los restantes solo como "listener". La transferencia de datos posibles es desde el "talker" a todos los "listener" simultáneamente.
- **Con "controller" único**: En esta configuración las transferencias de datos posibles son: Desde el "controller" a los equipos en modo comando y datos, de

un equipo al "controller" solo en modo datos, y de un equipo a otro equipo solo en modo datos.

- **Con múltiples "controller"**: En este caso tiene las mismas capacidades que la configuración anterior, sólo que en esta también es posible la transferencia entre equipos de la capacidad de operar como "controller activo".

TALKER: Equipo con capacidad de enviar Data.

En cada bus pueden existir uno o varios equipos con capacidad de enviar datos a otros equipos por el bus, pero en cada instante sólo uno de ellos puede ser establecido por el controller para que opere como Talker.

Es el único equipo (además del Active Controller) con capacidad de establecer el estado de las líneas DAV.

El equipo Talker sólo puede enviar un dato si todos los equipos que se encuentran en modo Listener esté en disposición de leerlo (Línea NRFD a valor lógico FALSE).

LISTENER: Equipo que recibe y lee todos los Data que se transfieren por el bus.

En cada bus pueden existir uno o varios equipos con capacidad de recibir datos desde el bus, y uno o varios de ellos se puede encontrar simultáneamente en modo “Listen”. El Active Controller es el que establece a través de un comando que un equipo pasa o deja de estar en modo Listen.

Todos los equipos que se encuentran en estado Listen reciben simultáneamente todos los datos que son transferidos por el bus.

IDLER: Estado base sin ninguna respuesta respecto del bus.

Comandos del Bus

Los comandos de bus (mensajes Command) son siempre enviados desde el controller a los otros equipos para sincronizar su estado de operación o para establecer su estado de operación.

En los comandos de bus se envían datos por el bus de datos (si la operación lo requiere) de igual modo que en la transferencia de datos, sólo que en estos casos la señal ATN es establecida a estado lógico TRUE por el "controller”. Con ello se indica que es un comando, y todos los equipos con independencia de que sean "talker" o "listener" reciben el comando, manteniendo el protocolo con el "controller".

El "controller" puede enviar cinco tipos de comandos de bus a los otros equipos: "addressed", "listen", "talk", "universal" y "secondary". Sólo los 7 bits menos significativos del bus son utilizados en los comandos de bus. Los tres bit b7, b6 y b5 definen la naturaleza de cada comando:

b7	b6	b5	Tipo comando
0	0	0	Addressed
0	0	1	Universal
0	1	x	Listen
1	0	x	Talk
1	1	x	Secondary

Los equipos conectados al bus GPIB tienen asignado un código o dirección de bus comprendido entre 0-30. Este código debe ser establecido en cada equipo, bien por medio de unos conmutadores hardware presentes en su panel trasero, o bien en algunos casos programando el equipo mediante su software interno. Normalmente, los cinco bits menos significativos de la línea de datos b4, b3, b2, b1, b0 se utilizan en un comando para establecer a qué equipos hace referencia un comando.
El código 31 ("11111") no es el código de un equipo, y suele estar reservado para hacer referencia a todos los equipos del bus en conjunto.

Comandos TALK/LISTEN

MTA	010ddddd	My Talk Address: Establece el modo Talker en el equipo "ddddd"
UNT	01011111	Untalk: El equipo en modo Talker pasa a modo Idler
MLA	001ddddd	My Listen Address: Establece el modo Listener en el equipo "ddddd"
UNL	00111111	Unlisten: Todos los equipos en modo Listener pasan a modo Idler

Mecanismos de finalización de mensajes de datos.
Hay tres mecanismos con el que el Talker puede comunicar que el dato que transmite es el último a enviar:
• **Método EOI**: Cuando envía el último dato el Talker establece la línea de control del bus EOI al estado lógico TRUE.
• **Método EOS**: El último carácter enviado por el Talker es el que previamente se ha establecido como End of Send. Habitualmente es CR (Ox0D) o LF (Ox0A). En este caso no es detectado por el hardware del bus sino por el software del driver.
• **Método de cuenta:** El controlador para al Talker cuando el número de byte del mensaje convenido (habitualmente establecido en la cabecera del mensaje) se han enviado. El Controlador bloquea al Talker estableciendo las líneas del control del bus NRFD y NDAC al estado lógico TRUE. La cuenta de bytes se realiza por el software del driver.

Comandos UNIVERSALES (UGC)

Mensajes enviados por el Controller a todos los equipos.

LLO	00010001	**Local Lockout**: Se deshabilitan los paneles de control de todos los equipos conectados al Bus.
DCL	00010100	**Device Clear**: Inicializa las interfaces hadware/software de los equipos.
PPU	00010101	**Parallel Poll Unconfigure**: Se cancela la programación previa de los equipos a fin de responder en la encuesta paralela (Pararell Poll)
SPE	00011000	**Serial Poll Enable**: Habilita a todos los equipos a fin de que respondan a la encuesta serie (Serie Poll).
SPD	00011001	**Serie Poll Disable**: Deshabilita el modo de encuesta serie.

Comandos ADDRESSED (ACG)

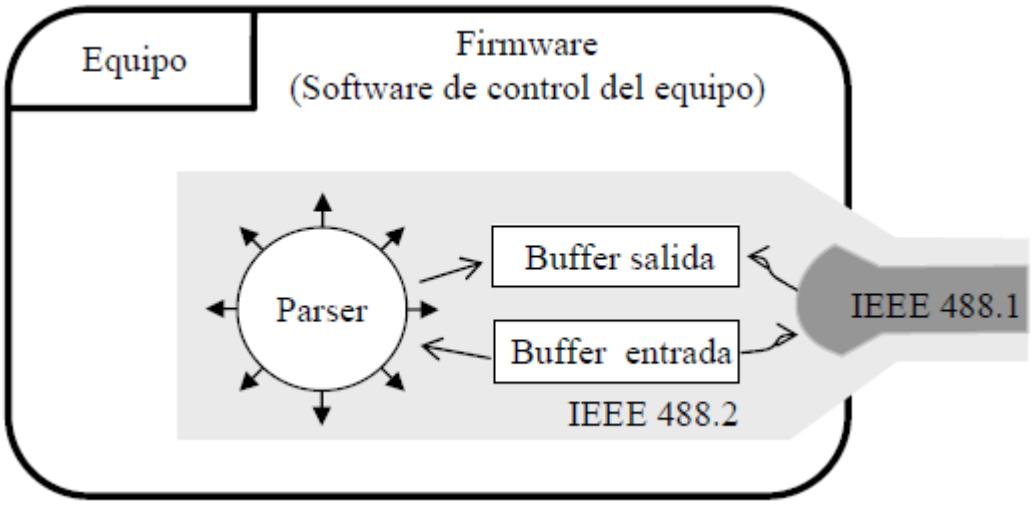
Van destinados y afectan únicamente a aquellos equipos que previamente han sido establecidos en "Listener".

GTL	00000001	**Go To Local**: Retorna el control de los paneles a todos los equipos en estado Listener.
SDC	00000100	**Selected Device Clear**: Inicializa las interfaces hadware/software de los equipos en estado Listener.
PPC	00000101	**Parallel Poll Configure**: Configura la respuesta a una encuesta paralela (Pararell Poll)los equipos en estado Listener. Los equipos quedan a la espera de un comando MSA.
MSA	011xcbbb	**My Secondary Address**: Establece la línea (bbb) y el estado (c=0 => request=FALSE) con la que responden a una encuesta paralela de los equipos Listener.
GET	00001000	**Group Trigger**: Dispara el trigger de los equipos en estado Listener.
TCT	00001001	**Take Control:** Establece como Active Controller al equipo que está establecido como Listener.

6.3 MODELO OPERATIVO DE LOS EQUIPOS (IEEE-488.2)

Protocolos

El estándar IEEE-488.2 define los modos de operación básicos de los equipos que lo satisfacen. Define los protocolos de intercambio de mensajes con el que el equipo y el controlador se comunican, así como facilidades básicas de control del instrumento.
Define:
a) Cuando los equipos están dispuestos a escuchar o a hablar a los otros equipos.
b) Qué ocurre cuando no se cumplen las normas establecidas en el protocolo.
Los equipos que satisfacen los modos operativos definidos en el estándar IEEE-488.2., necesariamente satisfacen los niveles de comunicación físicos definidos en el estándar IEEE-488.1. La implicación inversa no es requerida.
El modelo de intercambio de un equipo que satisface el estándar IEEE-488.2, incluye a los siguientes componentes:



Buffer de entrada: Es el área de memoria en la que las órdenes y requerimientos de entrada son almacenadas, antes de que sean interpretadas y ejecutadas por el Parser. El buffer de entrada permite que el equipo controlador del instrumento almacene en el buffer un string conteniendo una o varias órdenes que llevan un cierto tiempo esperando ser ejecutadas. Mientras esto ocurre puede realizar otras operaciones con otros equipo.

Buffer de salida: Es el área de memoria en el que los datos de salida (mensajes de salida) son almacenadas en espera de que el controlador decida leerlas.

Parser: Es el controlador interno del equipo que interpreta los mensajes completos almacenados en el buffer de entrada, los ejecuta, y en caso de que sean de requerimiento, deposita en la cola de salida los datos que resulten.

Protocolo básico

- El equipo y el controlador se comunican intercambiando mensajes de órdenes y mensajes de respuesta.

- Los mensajes de órdenes son enviados por el controlador, y pueden ser de dos tipos:

    Ordenes de control: Que requieren un cambio de estado del equipo, pero que no requieren ninguna respuesta.

    Ordenes de requerimiento: Que solicitan información sobre el estado del equipo o sobre información que posee.

- El equipo sólo habla (envía un mensaje de salida) como respuesta de una orden de requerimiento.

- El controlador sólo admite un mensaje de salida (respuesta de una orden de requerimiento), y lo requiere antes de enviar un nuevo mensaje de órdenes. En caso contrario se genera una situación de bloqueo.

- La regla básica del protocolo es:
"El equipo solo habla cuando está dispuesto a ello, y en ese caso, tiene que hablar antes de que se le ordene hacer una cosa nueva"

- Cuando el equipo es encendido, o cuando recibe una orden de inicialización "*CLS" el buffer de entrada y de salida son inicializada, y el parser es inicializado a la raíz de su árbol de órdenes.

- El equipo y el controlador se comunican intercambiando mensajes de órdenes y de respuestas completos. Esto significa que el controlador siempre debe terminar un mensaje de órdenes, antes de intentar leer una respuesta.

- Si el equipo envía un mensaje de respuesta, el controlador debe siempre leer de forma completa el mensaje, antes de que envíe un nuevo mensaje de órdenes al equipo.

- El controlador puede enviar un mensaje conteniendo múltiples órdenes de requerimientos. A esto se le denomina un "requerimiento compuesto". Los diferentes requerimientos dentro del mensaje deben estar separados por el delimitador ";". Los mensajes de respuesta son encolados en la cola de salida, separados entre sí también, por el delimitador ";".

- Los comandos son ejecutados en el orden en que han sido recibidos.

Protocolos de excepción

Cuando un error ocurre en el intercambio de la información, este no termina en la forma normal, sino que sigue un protocolo de excepción:

a) Equipo direccionado para hablar sin nada en la cola:
- Si es consecuencia de que el equipo no haya recibido una orden de requerimiento, el equipo indicará un error de encolamiento, y no enviará ningún byte por el bus.
- Si es como consecuencia de que la orden de requerimiento previa no se ejecutó como consecuencia de un error, el equipo no indica ningún error de encolamiento, sino que el equipo espera a recibir el siguiente mensaje del controlador.

b) Equipo direccionado para hablar sin que ningún equipo escuche: En este caso, el equipo esperará o bien a que algún equipo escuche, o a que el controlador tome el control.

c) Error de orden: Se genera cuando se detecta un fallo de sintaxis o una orden no reconocible.

d) **Error de ejecución:** Se genera si un parámetro está fuera de rango, o si el equipo se encuentra en un estado que no permita la ejecución del comando requerido.

e) Error específico del equipo: Se produce cuando el equipo es incapaz de ejecutar una orden, como consecuencia de una razón estrictamente dependiente de él, y no del protocolo seguido por el bus.

f) Error de encolamiento: Se genera si no se sigue el protocolo de lectura de los datos de la cola de salida.

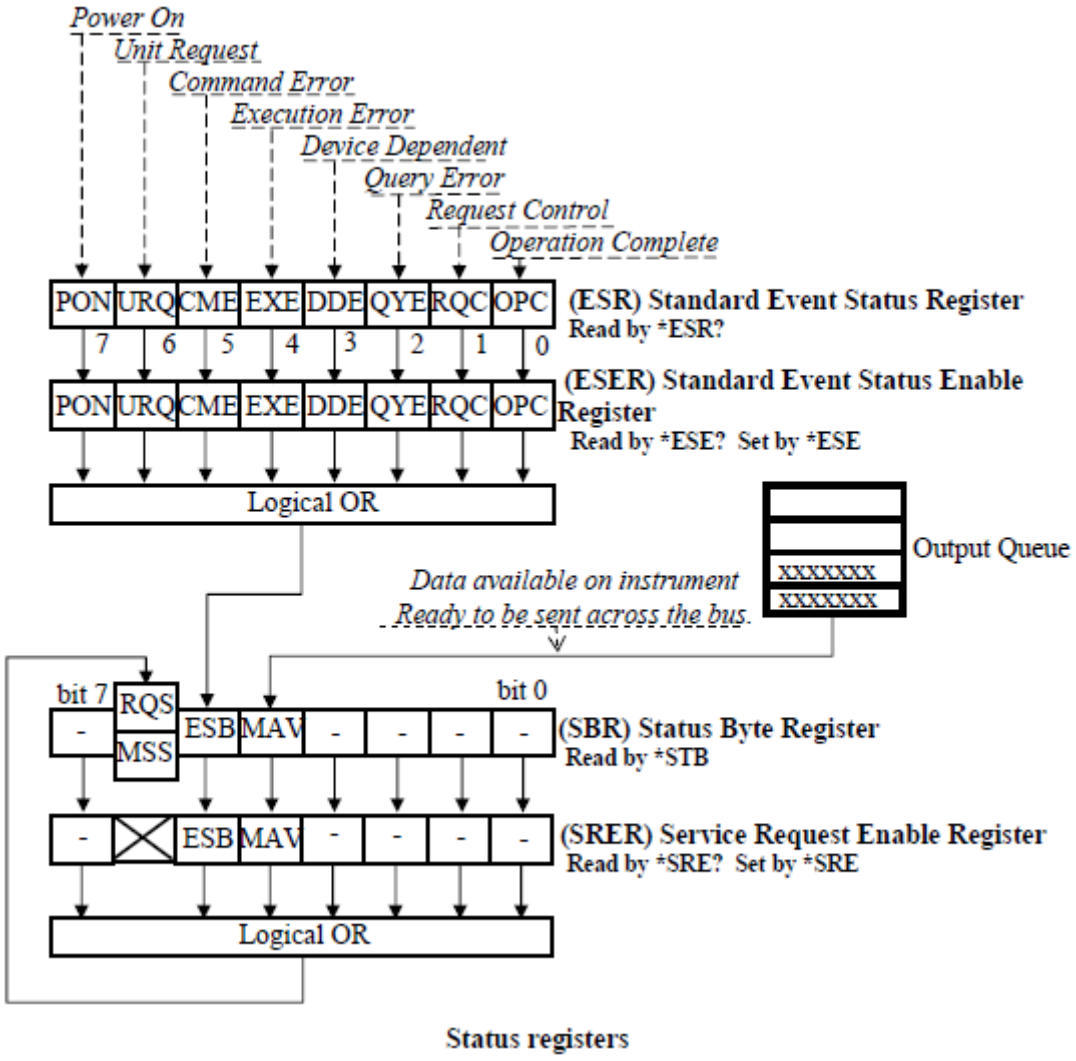
g) Condición inconclusa: Si el controlador intenta leer un mensaje de respuesta antes de que el programa haya concluido de ejecutar la orden que los genera. En este caso el Parser se inicializa a sí mismo, la respuesta ya elaborada es limpiada de la cola de salida, y ningún dato es transferido por el bus.

h) Condición interrumpida: Si el controlador no lee completamente el mensaje generado por un mensaje de requerimiento, y envía otro mensaje de orden, el equipo genera un error de encolamiento, y el segmento de mensaje de salida no leído es eliminado. La orden que interrumpe es inafectada.

i) Bloqueo de buffer: El equipo alcanza un estado de bloqueo si el buffer de entrada está lleno y también resulta llena la cola de salida. Esta situación ocurre si un mensaje de orden muy largo que contiene órdenes de requerimiento ha sido enviado, y genera un mensaje de salida superior al que puede contener la cola. El controlador no puede terminar de enviar el mensaje de entrada porque no cabe, y el buffer de entrada no se vacía porque espera que su cola de salida sea vaciada para concluir la orden en ejecución. En este caso el equipo rompe el bloqueo, limpiando la cola de salida y generando un error de encolamiento.

Estatus de un equipo por el protocolo IEEE 488.2

El estándar IEEE 488.2 ofrece un mecanismo estandarizado de presentar y mostrar el estado interno del equipo. A través de este mecanismo, se puede tener información de si el equipo tiene un dato dispuesto para transferir, así como si algún tipo de error ha ocurrido.



El mecanismo se basa en cuatro registros:

Status Byte Register (SBR): Cada bit está asociado con un tipo de estado específico del instrumento. Cuando el estado cambia, el instrumento establece el correspondiente bit a 1. Se puede habilitar e inhibir el efecto de cada bit del SBR a efectos de requerir atención (bit RQS), con el correspondiente bit del registro SRER. Se puede determinar qué eventos han ocurrido leyendo los establecidos en el registro SBR.

Bit	Label	Description
0-3	-	Instrument-specific summary messages.
4	MAV	The Message Available bit indicates if data is available in the Output Queue. MAV is 1 if the Output Queue contains data. MAV is 0 if the OutputQueue is empty.
5	ESB	The Event Status Bit indicates if one or more enabled events Have occurred. ESB is 1 if an enabled event occurs. ESB is 0 if no enabled events occur. You enable events with the Standar Event Status Enabled Register.
6	MSS	The Master Sumary Status sunarizes the ESB and MAV bits. MSS is 1 if either MAV or ESB is 1. MSS is 0 if both MAV and ESB are 0. This bits are obtained from *STB? Command.
	RQS	The ReQuest Service bit indicates that the instruments requests service from the GPIB controller. This bits is obtained by from a serial poll.
7	-	Instrument specific summary message.

Service Request Enable Register (SRER): Máscara de los bits correspondientes del registro SBR que va a determinar si se establece el Request Service (bit RQS).

Event Status Register (ESR): Cada bit está asociado con un tipo específico de evento. Cuando un evento ocurre, el instrumento establece el correspondiente bit a 1. Se puede habilitar o inhibir los eventos que van a ser proyectados sobre el bit ESB, a través de los correspondientes bits de máscara de registro ESR.

bit	Label	Description
0	OPC	The Operation Complete bit indicates that all commands have completed.
1	RQC	The Request Control bit is not used by most instruments.
2	QTE	The Query Error bit indicates that the instrument attempted to read an Empty output buffer, or that data in the output buffer was lost.
3	DDE	The Device Dependent Error bit indicates that a device error occurred (such as a self –test error).
4	EXE	The execution error bit indicates that an error occurred when the devices Was executing a command or query.
5	CME	The command Error bit indicates that a command syntax error occurred.
6	URQ	The User Request bit is not used by most instruments.
7	PON	The Power on bit indicates that the device is powered on.

Event Status Enable Register (ESER): Máscara de los bits del registro ESR que van a requerir el servicio a través del registro SBR (bit ESB).
Por ejemplo, si deseamos conocer cuando ocurre un error de ejecución, se debería establecer el bit 5 del registro ESER a fin de que el evento de error de interés (que establecerá el bit 5 del ESR a 1) sea reportado por el bit ESB del registro SBR.

Ordenes básicas

En el estándar IEEE-488.2 encontramos un conjunto de órdenes básicas que realizan funciones comunes a todos los equipos, con independencia de su naturaleza.

Orden	Nombre de la orden	Función
*CLS	Clear Status Command	*-Despeja el registro de estado y los registros de incidencia.*
*ESE	Event Status Enable Command	*-Habilita bits del registro de habilitación de incidencias*
*ESE?	Event Status Enable Query	*-Interroga el registró de habilitación de Incidencias estándar.*
*ESR?	Event Status Register Query	*-Interroga el registro de Incidencias estándar.*
*IDN	Identification Query	*-Identifica tipo de instrumento y versión software.*
*LRN?	Learn Device Setup Query	*-Requiere el estado actual del equipo.*
*OPC	Operation Complete Command	*-Fija el bit de "Operación completa" del registro estándar.*
*OPC?	Operation Complete Query	*-Responde con "1" si se han ejecutado ordenes previas.*
*OPT?	Option Identification Query	*-Requiere la opción instalada en el equipo.*
*RCL	Recall Command	*-Restaura el estado del equipo del registro save/recall.*
*RST	Reset Command	*-Sitúa al equipo en el estado básico de referencia.*
*SAV	Save Command	*-Almacena el estado actual en un registro save/recall.*
*SRE	Service Request Enable Command	*- Habilita los bits del registro de habilitación de Byte de estado.*
*SRE?	Service Request Enable Query	*- Requiere el contenido del registro SER de habilitación del byte de estado*
*STB?	Read Status Byte Query	*- Requiere el estado del registro resumido del Byte de estado.*
*TRG	Trigger Command	*- Arranca o dispara la operación del equipo de forma remota.*
*TST?	Selft-Test Query	*-Requiere el resultado del autotest del equipo.*
*WAI	Wait-to-Continue Command	*- Espera a que se realicen todas las operaciones pendientes.*

6.4 SCPI

A pesar de los estándares IEEE 488.1 y 488.2, existía libertad para que cada fabricante eligiera los comandos de sus instrumentos, por eso en 1990 un grupo de empresas fabricantes de instrumentos acordaron crear un conjunto de órdenes con una sintaxis común, que fue llamada SCPI (Comandos Estándar para Instrumentos Programables).
Lógicamente, SCPI se construyó respetando los principios del anterior 488.2.
Si dos instrumentos, de fabricantes distintos, se adhieren al estándar SCPI, es teóricamente posible intercambiarlos con mínimas modificaciones en el programa de control. Los comandos SCPI se escriben como texto ASCII, y tienen una estructura jerárquica por niveles, separados por dos puntos:

SOURce:VOLTage <valor>

En esta instrucción SOURce es la raíz del comando y VOLTage es nuestro primer nivel.
Las instrucciones pueden tener más de un nivel. Los caracteres en mayúsculas son necesarios para especificar la orden, mientras que los que están en minúsculas pueden suprimirse, sirviendo sólo para facilitar la lectura de programas por usuario. El comando anterior es equivalente a SOUR:VOLT <valor>.

Los comandos en sí pueden ser escritos indistintamente en mayúsculas o minúsculas.
Una ventaja del estándar SCPI es la definición homogénea de comandos para todos los aparatos de una misma clase.

7.CONTROL MSCOMM

Como hemos comentado en la sección del adaptador GPIB-USB, desde el ordenador veremos el adaptador como un puerto serie, por tanto para la comunicación entre el ordenador y el adaptador deberemos utilizar herramientas que trabajen con el puerto serie; en nuestro caso emplearemos el control MSCOMM.

7.1 Características

El control *MSComm* proporciona comunicaciones serie para que su aplicación pueda transmitir y recibir datos a través de un puerto serie.
Es una herramienta integrada en entornos de desarrollo de Microsoft. El icono que lo representa en la caja de herramientas es el siguiente:

En el formulario solamente es visible en tiempo de diseño.

El control *MSComm* proporciona dos formas diferentes de tratamiento de las comunicaciones:

- Las comunicaciones controladas por eventos, en las que cuando aparece un evento, por ejemplo cuándo llega algún carácter, se produce un evento del tipo *OnComm* del control *MSComm* para interceptar y tratar la información según el tipo de evento. El evento *OnComm* también detecta y trata los errores en las comunicaciones. En la propiedad *CommEvent* puede ver una lista completa de todos los eventos y errores posibles en las comunicaciones. Esta es la forma que utilizaremos para la implementación.

- También se puede sondear los eventos y errores si comprueba el valor de la propiedad *CommEvent* después de cada función crítica de su programa.

Cada uno de los controles *MSComm* que use corresponde a un puerto serie. Si necesita tener acceso a más de un puerto serie en su aplicación, debe usar más de un control *MSComm*.
El control *MSComm* tiene muchas propiedades de las cuales detallamos a continuación las más importantes.

7.2 Propiedades

Existen propiedades que pueden establecerse en tiempo de diseño o en tiempo de ejecución, y otras que solamente se pueden ejecutar o consultar en solamente en tiempo de ejecución. Estas propiedades tienen que ser configuradas correctamente para que la comunicación sea posible.

CommPort

Indica el número del puerto serie usado. Admite los valores de 1 a 255. Cambiando esa propiedad, podemos cambiar el puerto de comunicación que vamos a usar. Si le damos a este valor un número de puerto inexistente, dará error. Deberemos saber el puerto de nuestro adaptador para configurar esta propiedad en nuestro programa.

Settings

Su sintaxis es: Velocidad, Paridad, Bits de información, Bits parada
Indica la velocidad, paridad, número de bits y bits de stop (parada) que se van a usar en la comunicación.
Los valores posibles para *velocidad* (en baudios) son 50, 100, 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200 y 28800

Los valores posibles para *paridad* son:
N: No envía bit de paridad ni hace comprobación de paridad en la recepción.
O: Envía y comprueba paridad, con el criterio de paridad IMPAR
E: Envía y comprueba paridad, con criterio de paridad PAR

Los valores para el parámetro *Bits de Información* pueden ser :
7: Se envían / reciben 7 bits por trama de información.
8: Se envían / reciben 8 bits por trama de información
5: Se envían / reciben 5 bits por trama de información. Este valor de 5 bits es el típico del sistema *Baudot* para transmisión telegráfica (Teletipos) que se ha conservado en las comunicaciones informáticas por pura tradición. Si se elige 5 bits, los bits de parada se ponen automáticamente a 1,5 (Típico también del sistema Baudot.)

Los valores para el parámetro *Bits de parada* pueden ser:
1: Se envía un bit de parada
2: Se envían 2 bits de parada
No es posible programar 1,5 bits de parada. Sólo lo hace cuando se programan 5 bits de información y lo hace automáticamente.

Nuestra configuración será “9600, n, 8, 1"

Handshaking

Especifica el método de control sobre el flujo de información. En una comunicación serie se necesita conocer si el puerto puede enviar información y necesita también

indicarle a la fuente que él está preparado para recibir información. A este proceso se le denomina *Handshaking*. (Control de Flujo)
El Control de Flujo puede hacerse de dos formas: una mediante las señales auxiliares del puerto (RTS, CTS, DSR, DTR), que son cables adicionales que tendrán una tensión positiva respecto a los 0 V del equipo si esa señal está activada, o una tensión negativa si no lo está.
Existe otra forma de controlar el flujo de información: mediante señales especiales que se envían por los dos cables que transportan la información. Mediante estas dos señales podemos controlar que el ordenador envíe información o deje de enviarla. De igual forma, podemos indicarle al módem que envíe o no envíe. Estas señales especiales se denominan X-ON y X-OFF.
La propiedad *Handshaking* controla la forma de realizar este proceso. Puede tomar los siguientes valores :
0: No existe Control de Flujo
1: Control de Flujo mediante XON - XOFF
2: Control de Flujo mediante Request To Send (RTS) y Clear To Send (CTS)
3: Control de Flujo mediante XON - XOFF y RTS – CTS

Nosotros utilizaremos el control de flujo mediante Request To Send (RTS) y Clear To Send (CTS).

InBufferSize
Mediante esta propiedad establecemos el tamaño del Buffer (almacén de datos) de entrada. Este Buffer sirve para poder recibir datos sin que tenga que intervenir la aplicación continuamente para controlar el puerto de entrada.
Puede conocerse el número de caracteres presentes en el Buffer de entrada consultando el valor de la propiedad *InBufferCount*. Nuestra aplicación no recibe una gran cantidad de datos por tanto con 1024 bits es suficiente.

OutBufferSize
Mediante esta propiedad controlamos el tamaño del Buffer de salida.
No enviamos muchos bits a la vez; por tanto con 512 bits es suficiente.
Se puede conocer el número de caracteres presentes en el Buffer de salida (los que aún están por transmitir) consultando el valor de la propiedad *OutBufferCount*.

RThreshold, SThreshold
Estas dos propiedades especifican el número de caracteres que deben estar presentes en los Buffers de Recepción y Transmisión respectivamente, para que se produzca el evento *OnComm* relativo a recepción y transmisión de caracteres(Eventos *EvReceive* y *EvSend*). Si el valor de una de estas propiedades está a 0, no se produce el evento *OnComm* correspondiente.
El valor que se debe dar a estas dos propiedades depende de la aplicación y del tiempo que queramos que la aplicación está atendiendo al puerto de comunicaciones. Si ponemos en RThreshold un valor corto (1 es el mínimo), cada vez que reciba un

carácter se producirá el evento *OnComm*. Al producirse este evento, ejecutará el procedimiento asociado a él, lo que hará perder tiempo a la aplicación, impidiéndole realizar otras funciones. Si se pone un valor muy alto, el puerto no avisará que tiene caracteres recibidos hasta que reciba un número igual al programado en esta propiedad, por lo que no podremos procesar los datos recibidos hasta que el buffer tenga ese número de caracteres en su interior. El número adecuado para nuestra aplicación será de 10 caracteres en RThreshold, ya que es el número de caracteres que envía la fuente cuando le pedimos la corriente consumida, y un 0 en SThreshold.

InputLen
Por defecto, cuando se lee el Buffer de recepción, se leen todos los caracteres, quedando el Buffer vacío. Si se le asigna a esta propiedad un valor distinto de 0, cada vez que leamos el Buffer de recepción leerá un número de caracteres igual a esa cantidad, permaneciendo los caracteres restantes en el Buffer a la espera de una nueva lectura. Asignándole el valor 0 (Valor por defecto), el buffer se lee completo que será el valor que utilizaremos nosotros.

RTSEnable
Activa (Pone a 1) la señal *RTS* (Request To Send - Petición de envío). Esta señal debe ponerse a 1 para indicar a la fuente que va a recibir nuestra comunicación que deseamos enviar datos. Debe estar activada durante toda la transmisión de datos.

DTREnable
Activa (Pone a 1) la salida DTR (Data Terminal Ready - Terminal de Datos Listo). Esta señal se emplea para decirle a la fuente que ordenador está preparado para recibir datos.

7.3 Propiedades propias del tiempo de ejecución

PortOpen
Abre el puerto de comunicación. Puede tener los valores *True* (Para abrirlo) y *False* (Para cerrarlo). Ejemplo apertura del puerto:

```
puertoGPIB.PortOpen = true;
```

InBufferCount
Nos permite averiguar cuántos caracteres tenemos en el Buffer de entrada.

OutBufferCount
Nos permite conocer cuántos caracteres quedan por transmitir en el Buffer de salida.

Output

Envía caracteres al Buffer de salida. Debe existir un signo igual (=) entre *Output* y lo que se envía al Buffer. Ejemplo de envío de comando a la fuente:

```
puertoGPIB.Output = "MEAS:CURR?\r";
```

Input

Lee el Buffer de recepción. El número de caracteres leídos dependerá del valor de la propiedad *InputLen*. Cuando la propiedad *InputLen* tiene el valor 0, el Buffer se lee completo. Si *InputLen* tiene un valor distinto de 0, se leerá un número de caracteres igual al valor de esta propiedad. Ejemplo:

```
caracteresRecibidos=puertoGPIB.Input;
```

CTSHolding

Devuelve el estado de la línea de control *CTS* (Dispuesto para enviar). Si es *True*, esa entrada está activada, si es *False*, la entrada está desactivada.

8.LA APLICACIÓN

8.1 Utilidad

El programa diseñado es una aplicación que sirve para realizar el control de varios parámetros de la fuente de alimentación utilizada, y para realizar un muestreo de la intensidad consumida por la fuente cada determinado tiempo y guardarlos en un fichero. Los parámetros de la fuente, que podemos modificar mediante la aplicación, son el voltaje de salida de la fuente y la activación o desactivación de la salida de la fuente.
Sobre el muestreo, se permite realizar dos tipos de muestreo: un muestreo normal, en el que se programa el tiempo total del muestreo y el intervalo de muestreo de la intensidad, o un muestreo por ciclos, donde podemos ir alternando un ciclo de funcionamiento con un ciclo de parada programando los aspectos anteriores más el tiempo de funcionamiento y el tiempo de parada que queramos. También la aplicación permite realizar una pequeña configuración de algún aspecto del adaptador y del programa.

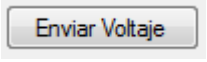
8.2 Diseño

El programa ha sido diseñado en el lenguaje de programación C#, utilizando el entorno de desarrollo Visual Studio 2008 de Microsoft. El proyecto ha sido creado como una aplicación de Windows Forms, donde para la creación de las ventanas se utilizan las librerías de Windows.

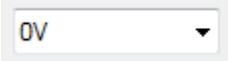
Windows Forms

El diseño mediante Windows Forms permite crear el entorno gráfico de una forma gráfica sin tener que escribir el código fuente. Para ello se tiene un conjunto de herramientas como son botones, texto plano o menús desplegables, las cuales se insertan en la ventana a diseñar arrastrándolas y soltándolas sobre ella. Las herramientas o componentes utilizados en este proyecto son:

Button: Se trata de un simple botón en el cual al pulsar sobre él se asocia una acción a realizar.

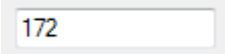


ComboBox: Se trata de un menú desplegable donde en cada entrada se escriben los datos permitidos.



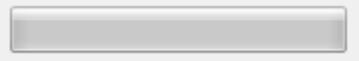
GroupBox: Muestra un marco con un título alrededor de un grupo de controles.

TextBox: Recuadro donde se permite especificar un texto. Este texto se puede emplear como entrada de algún dato.



Label: Muestra un texto descriptivo.

ProgessBar: Muestra una barra que se va cargando para mostrar el progreso.



Microsoft Communications Control: Se trata del control MSCOMM para la comunicación con el puerto COM.

8.3 Fichero de datos del muestreo

Los datos sobre intensidad consumida, recibidos de la fuente, son almacenados en un fichero . La ruta donde se guarda se puede especificar en la ventana de configuración que más adelante se explicará. La extensión de este fichero se puede indicar cuando se especifica la ruta, escribiéndola a continuación del nombre del archivo. Por ejemplo, si queremos guardarlo en un fichero de texto normal escribiremos: *nombredelfichero***.txt**

Si no se define formato, luego habrá que ponerle la extensión o abrirlo directamente desde un programa que lo lea correctamente.

Los datos se guardan en el fichero de la siguiente forma:

Intensidad consumida (A) Número de muestra Tiempo (seg)

Cada línea en el fichero es un dato muestreado. La intensidad se almacena en amperios y el tiempo en segundos.

9.FASES DEL DESARROLLO DEL PROYECTO

Antes de comenzar con el desarrollo de la aplicación, empecé con el aprendizaje de los distintos aparatos que se utilizan como son la fuente y el adaptador. Para ello realicé la lectura de los manuales de estos aparatos hasta comprender su funcionamiento. Lo siguiente fue intentar comunicar el ordenador con la fuente; para ello se empleó una aplicación que incluía el adaptador. Esta aplicación era una especie de terminal para comunicación por el puerto COM, en el que todo lo que se escribía lo enviaba por el puerto y todo lo que recibía lo sacaba por pantalla. Los comandos utilizados para la comunicación son comandos SCPI, por lo que tuve que buscar información acerca de cómo utilizarlos. Una vez entendidos los comandos SCPI, realicé la comunicación con la fuente mediante la aplicación anteriormente mencionada.

El paso siguiente fue intentar comunicarse mediante un programa en C#, que es el lenguaje que utilizaría para crear la aplicación. Aquí el problema surgió en buscar una forma de poder comunicarse con el adaptador mediante el puerto serie. Para ello le pregunté al profesor Ángel Rodas Jordá del DISCA sobre cómo podría realizar la comunicación y él fue el que me aconsejó realizarla con el control MSCOMM de Microsoft, el cual se habla en la sección 6.

Una vez conocido este control, comencé a realizar un programa de prueba con el que poder comunicarme y realizar diferentes acciones en la fuente. Después de bastante tiempo logré controlar las funciones que quería de la fuente, como es la modificación de la intensidad y la recepción de la intensidad enviada por la fuente.

Ahora ya tenía una gran parte del proyecto resuelta, que era lograr la comunicación. El siguiente paso fue comenzar a realizar la aplicación que formaría el proyecto. Comencé con la creación de un menú inicial donde elegir realizar un ensayo o configurar el adaptador. Luego implementé la ventana de configuración necesaria para que funcionara correctamente la aplicación y más tarde realicé el ensayo simple, donde se realiza ya el muestreo de la intensidad. Aquí fui añadiendo los componentes necesarios hasta tener listo todo lo necesario para el funcionamiento. Cuando ya estaba esto hecho, comencé a hacer pruebas para ver su correcto funcionamiento, pero me dí cuenta que había un fallo y era que no recibía todos los muestreos de intensidad que realizaba y se perdían muchos datos.

Después de probar diferentes cosas logré solucionar el problema de la pérdida de datos realizando una petición a la fuente para que se pusiera en modo escucha para luego pedirle el dato de intensidad. Este fue uno de los problemas que mayor tiempo me llevó solucionar.

Con el ensayo simple en correcto funcionamiento, comencé a realizar el ensayo por ciclos, que era más complejo, pero en el que me podía servir de la mayoría del código utilizado en el ensayo simple y luego a partir de este realizar la ampliación. También realice la introducción del tiempo tanto en el ensayo simple como en el de por ciclos.

Cuando ya finalicé y comprobé que funcionaba el ensayo por ciclos, lo que hice fue repasar todo el código implementado e ir corrigiendo errores y excepciones que se podían efectuar con el fin de intentar tener una aplicación sin errores.

10. INSTRUCCIONES DE USO

1) Instalación del adaptador GPIB

1) Realizaremos la instalación del adaptador como se explica en la sección 3 sobre el **ADAPTADOR GPIB-USB** en el apartado ***Instalación de los drivers del adaptador.***

Una vez realizada la instalación correctamente, ya no tendremos que realizar otra vez la instalación, sólo tendremos que conectar el adaptador en el mismo puerto USB.

2) Conexión de la fuente de alimentación

1) Conectaremos el conector GPIB del adaptador a la fuente. El conector GPIB de la fuente se encuentra en la parte trasera del mismo.
2) Lo siguiente será encenderlo del botón verde que hay en la parte frontal.

Para más información ver la sección *FUENTE DE ALIMENTACIÓN.*

3) Copia de archivos necesarios para el control MSCOMM

1) En la carpeta Controles MSCOMM, que se encuentra dentro de la carpeta **anexos**, encontramos 3 archivos necesarios para utilizar el control MSCOMM. Debemos copiarlos y pegarlos en la carpeta ***C:\Windows\System32***

4) Instalación programa visual studio

Para la ejecución del programa necesitamos una serie de componentes que deben estar instalados en nuestro ordenador. Estos componentes los podemos encontrar en las distribuciones de Visual Studio de Microsoft. Si no tenemos instalado en nuestro ordenador ninguna versión de Microsoft Visual Studio, deberemos instalarla si tenemos una en nuestra propiedad y si no, podemos instalar la versión libre Microsoft Visual Studio C# 2008 Express Edition. Esta edición se puede encontrar en internet de forma libre en páginas de descarga de software como softonic o en la propia página de Microsoft.

5) Utilización del programa

1) Creación de un acceso directo para ejecutar el programa

El icono para ejecutar la aplicación se encuentra en la carpeta:

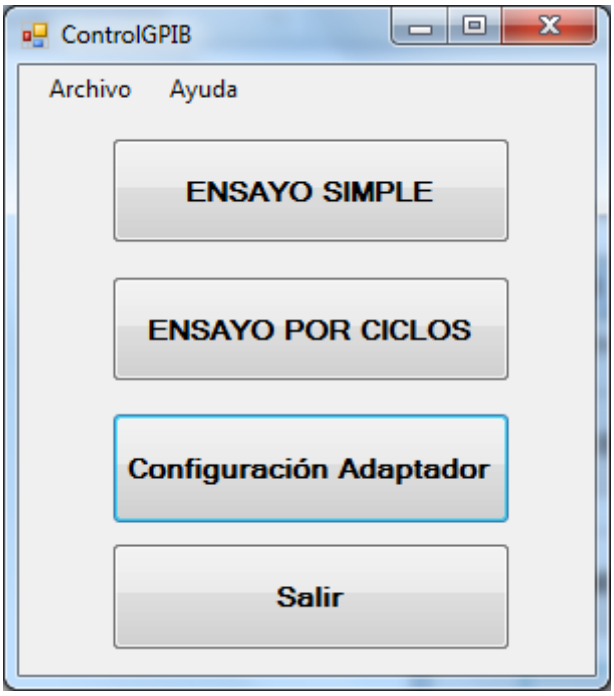
Index →ControlGPIB →ControlGPIB →bin →Release →ControlGPIB.exe

Hay que tener cuidado porque hay 2 archivos que se llaman ControlGPIB pero el que nosotros queremos es tipo aplicación es decir un .exe.

Podemos ejecutar la aplicación desde esta ubicación, pero para hacerlo más sencillo podemos crear un acceso directo a esta aplicación y luego ponerlo donde queramos y así acceder mediante este acceso directo al programa.

2) Ventana de inicio

Una vez entremos en la aplicación encontraremos la ventana de inicio.



En esta ventana inicial encontramos cuatro botones con distintas funciones:

ENSAYO SIMPLE: Se nos abrirá una nueva ventana donde podremos realizar un ensayo simple, en el que configuraremos un periodo de muestreo y la duración. Si no se puede abrir el puerto COM para la comunicación, porque no está conectado el adaptador o

porque no está configurado correctamente, en la ventana *Configuración Adaptador* aparecerá un mensaje error y no podremos abrir la ventana.

ENSAYO POR CICLOS: Si lo pulsamos, abriremos la ventana de ensayo por ciclos, donde configuraremos un ensayo en el pueda haber periodos de inactividad y periodos de muestreo. Al igual que el ensayo por ciclos, no podremos abrir la ventana si no está conectado el adaptador.

Configuración Adaptador: Abriremos una ventana donde configuraremos parámetros del adaptador y de la aplicación, como el puerto COM utilizado, la dirección donde guardar el fichero de datos del muestreo, o también podremos establecer parámetros por defecto de la aplicación.

Salir: Con este botón cerraremos la aplicación.

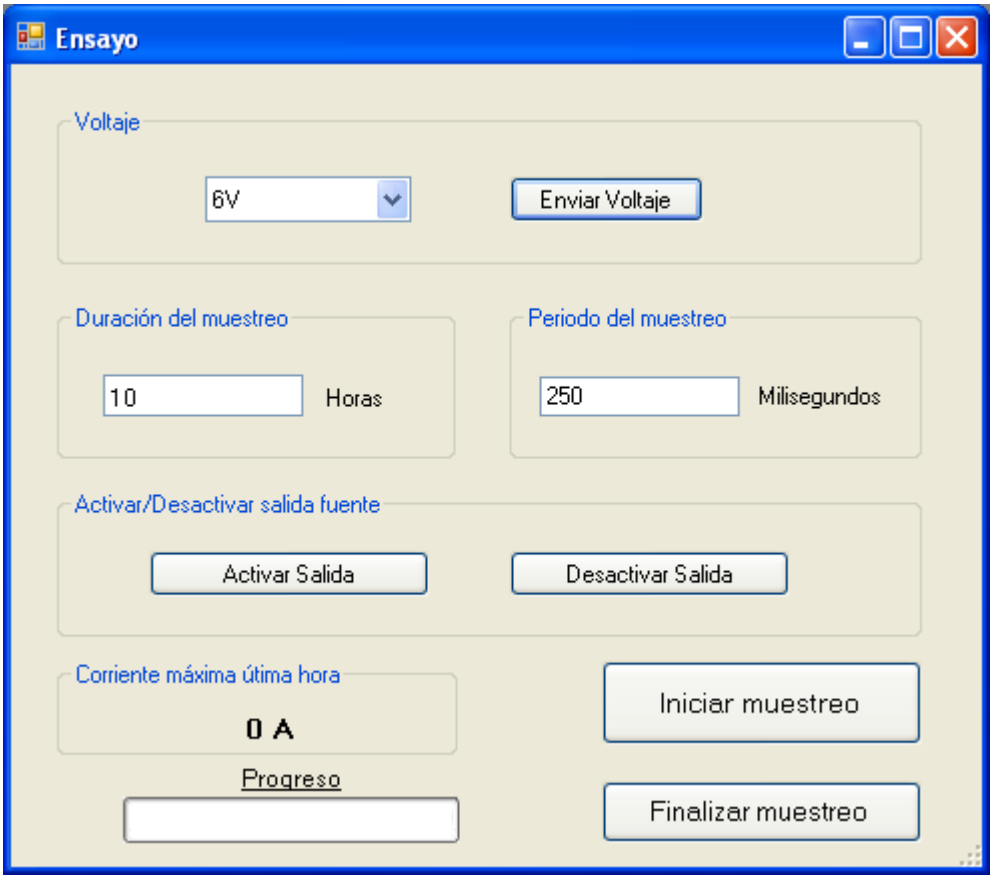
Archivo: Encontramos los cuatro opciones anteriormente explicadas pero en forma de menú desplegable.

Ayuda: Encontramos 3 opciones en el menú desplegable:

> *Biorreactor*: Muestra un mensaje indicando donde se encuentra el pdf con la información sobre el biorreactor.
> *Aplicación*: Muestra un mensaje indicando donde se encuentra información sobre la aplicación.
> *Acerca de*: Se muestra una pequeña información donde aparece algún dato sobre el proyecto.

3) Ventana de ensayo simple

Podremos realizar un ensayo programando una duración y un período de muestreo. Los datos serán guardados en un fichero, en la ubicación definida en la ventana de configuración de la ventana de inicio.



Explicación de cada bloque o botón:

Voltaje: Tenemos un menú desplegable donde seleccionaremos el voltaje que queremos que suministre la fuente. Una vez seleccionado, hay que enviarle a la fuente el voltaje pulsando el botón ***Enviar Voltaje***.

Duración del muestreo: Debemos escribir en el recuadro el número de horas que queremos que dure el muestreo. Debe estar entre 1 y 596 horas.

Periodo del muestreo: Escribiremos en el recuadro el período de muestreo, es decir cada cuántos milisegundos queremos leer la intensidad consumida. El periodo mínimo de muestreo es de 200ms.

Activar/Desactivar salida fuente: Estos dos botones sirven para activar o desactivar la salida de la fuente. No será necesario activar la salida antes de iniciar el muestreo, ya que activa automáticamente al pulsar el botón *Iniciar muestreo*. Hace la misma función que el botón OUTPUT ON/OFF de la fuente.

Corriente máxima última hora: Muestra la corriente máxima consumida por el dispositivo que conectamos a la fuente.

Progreso: La barra muestra el progreso del muestreo programado una vez lo iniciemos.

Iniciar muestreo: Con este botón iniciaremos el muestreo, es decir se activará la salida de la fuente y empezaremos a tomar valores de intensidad consumida.

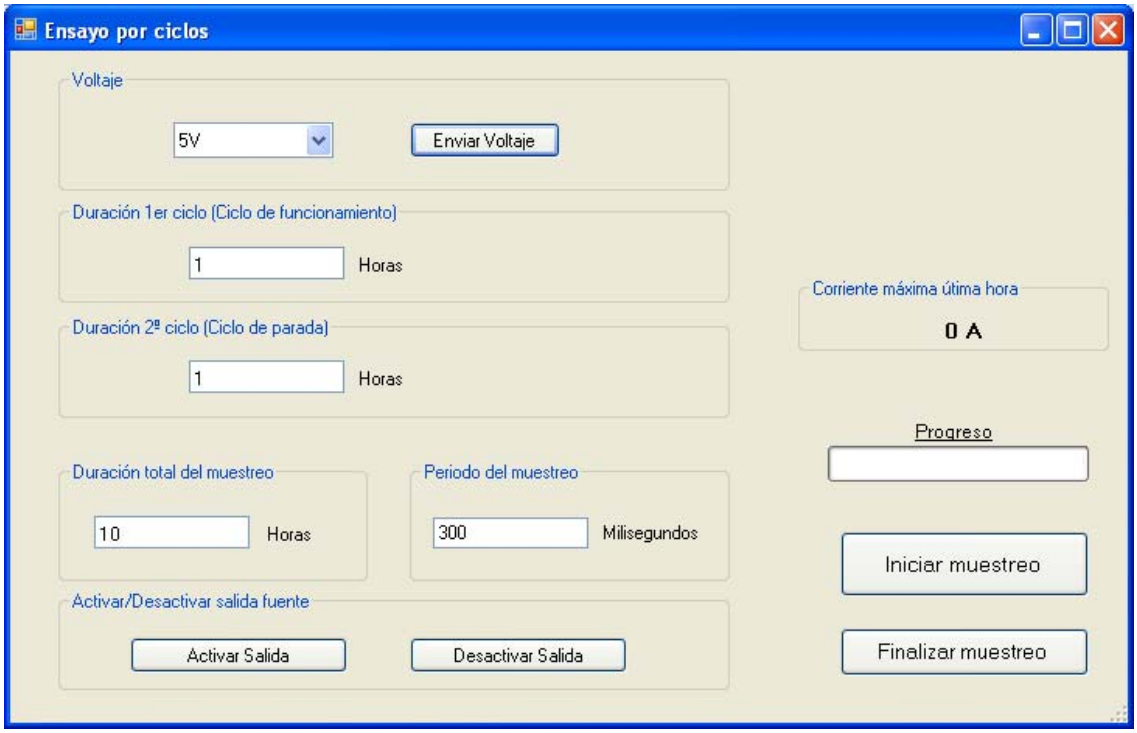
Finalizar muestreo: Con este botón finalizaremos el muestreo aunque no se haya completado el tiempo programado.

Pasos a seguir para realizar el ensayo simple:

- Primero seleccionaremos el voltaje, y una vez seleccionado pulsaremos el botón *Enviar voltaje**.***
- Luego escribiremos la duración en horas y el período de muestreo en milisegundos.
- A continuación, pulsaremos el botón *Iniciar muestreo* para comenzar el muestreo.
- Si deseamos finalizar el ensayo antes de que se cumpla el tiempo establecido anteriormente, pulsaremos el botón *Finalizar muestreo.*

4) Ventana de ensayo por ciclos

Podremos realizar un ensayo por ciclos en el que programaremos la duración de un primer ciclo, donde estará activado el muestreo, y un segundo ciclo de inactividad, donde se parará el muestreo y la salida de la fuente. Este ciclo de actividad y luego inactividad se irá repitiendo hasta que se cumpla la duración programada. Los datos serán guardados en un fichero, el cual definiremos su ubicación en la ventana de configuración de la ventana de inicio.



Explicación de cada bloque o botón:

Voltaje: Tenemos un menú desplegable donde seleccionaremos el voltaje que queremos que suministre la fuente. Una vez seleccionado, hay que enviarle a la fuente el voltaje pulsando el botón ***Enviar Voltaje***.

Duración 1er ciclo (Ciclo de funcionamiento): Indicaremos en el recuadro el número de horas del primer ciclo, que es el ciclo de funcionamiento donde se realiza el muestreo.

Duración 2er ciclo (Ciclo de parada): Indicaremos el número de horas del segundo ciclo, que es el ciclo donde no realizamos ningún muestreo. Se desconecta también la salida de la fuente de alimentación.

Duración del muestreo: Debemos escribir, en el recuadro, el número de horas que queremos que dure el muestreo. Debe estar entre 1 y 596 horas.

Periodo del muestreo: Escribiremos, en el recuadro, el periodo de muestreo, es decir cada cuántos milisegundos queremos leer la intensidad consumida. El periodo mínimo de muestreo es de 200ms.

Activar/Desactivar salida fuente: Estos dos botones sirven para activar o desactivar la salida de la fuente. No será necesario activar la salida antes de iniciar el muestreo ya que activa automáticamente al pulsar el botón *Iniciar muestreo*. Hace la misma función que el botón OUTPUT ON/OFF de la fuente.

Corriente máxima última hora: Muestra la corriente máxima consumida por el dispositivo que conectamos a la fuente.

Progreso: La barra muestra el progreso del muestreo programado una vez lo iniciemos.

Iniciar muestreo: Con este botón iniciaremos el muestreo, es decir se activará la salida de la fuente y empezaremos a tomar valores de intensidad consumida.

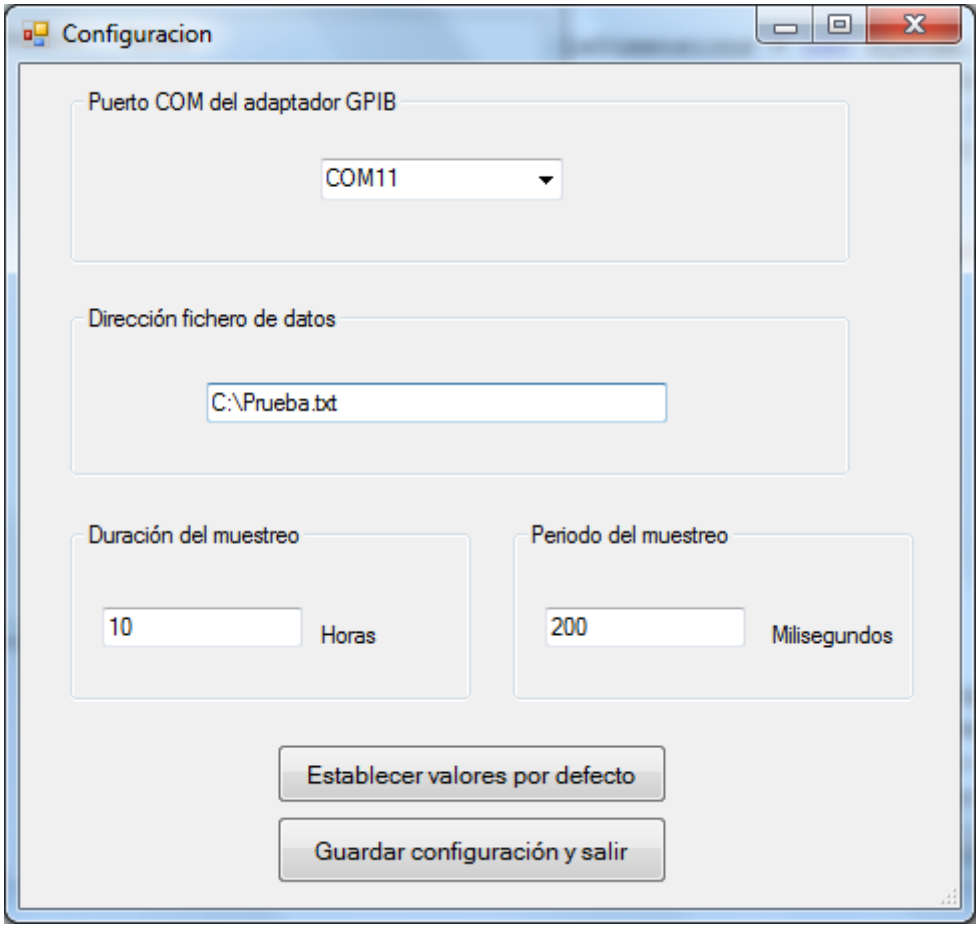
Finalizar muestreo: Con este botón finalizaremos el muestreo aunque no se haya completado el tiempo programado.

Pasos a seguir para realizar el ensayo por ciclos:

- Primero seleccionaremos el voltaje y una vez seleccionado pulsaremos el botón *Enviar voltaje**.***
- Luego escribiremos la duración del ciclo de funcionamiento, de parada y la duración total del muestreo. También programaremos el período de muestreo en milisegundos.
- A continuación pulsaremos el botón *Iniciar muestreo* para comenzar el muestreo.
- Si deseamos finalizar el ensayo antes de que se cumpla el tiempo establecido anteriormente pulsaremos el botón *Finalizar muestreo.*

5) Ventana configuración adaptador

En esta ventana podremos configurar aspectos importantes, como el puerto COM donde está conectado el adaptador y la dirección donde guardar el fichero de datos leídos del muestreo.



<u>Explicación de cada bloque o botón:</u>

Puerto COM del adaptador GPIB: En este menú desplegable debemos seleccionar el número del puerto COM donde hemos instalado y conectado el adaptador. Deberemos configurar el puerto COM correctamente para poder acceder a los ensayos. Para saber cómo conocer el puerto donde está instalado nuestro adaptador hay que ir al apartado 3.3 sección 3 donde se explica.

Dirección fichero de datos: En este recuadro hay que escribir la ruta donde queremos guardar el fichero de datos, así como el nombre del fichero que le queremos dar. Por ejemplo, si queremos guardar el fichero *ensayo1.txt* en el disco duro *C* dentro de la carpeta *datos,* que se encuentra dentro de la carpeta *proyecto* escribiremos: *C:\proyecto\datos\ensayo1.txt*

Hay que recordar que si no ponemos extensión al fichero, creará un fichero que no será reconocido por ningún programa y por tanto, no lo podremos abrir directamente pulsando sobre él, por lo que habrá que ponerle una extensión o abrirlo desde un programa que pueda tratar el texto almacenado en el fichero. La ruta debe existir, porque si no existe se mostrará un error.

Duración del muestreo y Periodo del muestreo: En estos recuadros escribiremos la duración y el período del muestreo que queremos que aparezca cuando abrimos las ventanas de ensayo simple y por ciclos. Cuando vayamos a realizar los ensayos, estos parámetros podrán ser modificados.

Establecer valores por defecto: Este botón guarda en un fichero llamado *Parametros.txt,* que se encuentra en la carpeta /anexos/C*ontrolGPIB/ControlGPIB/bin/Release,* los 4 parámetros que hemos explicado anteriormente para que la siguiente vez que se inicie el programa cargue la configuración de este fichero y no tengamos que escribirla cada vez que iniciemos el programa. Luego, para que la configuración tenga efecto en la sesión abierta habrá que salir pulsando el botón *Guardar configuración y salir.*

Guardar configuración y salir: Este botón guarda los parámetros anteriormente explicados en el programa y cierra la ventana de configuración. Este botón, a diferencia del anterior, no guarda la configuración para la siguiente vez que iniciemos el programa, solo lo guarda para la sesión actual abierta del programa.

Pasos a seguir

- Conocer el puerto COM donde se ha instalado el adaptador. En el apartado 3.3 de la sección 3 se explica cómo conocer el número de puerto.
- Escribiremos la dirección donde queramos guardar nuestro fichero de datos del muestreo y modificaremos, si queremos, los parámetros.
- Modificaremos, si queremos, la duración y el período de muestreo que queramos que aparezca en la pantalla de los ensayos.
- Si queremos que estos parámetros se guarden para cuando volvamos a utilizar el programa, pulsaremos el botón *Establecer valores por defecto.*
- Para salir y guardar la configuración, pulsar *Guardar configuración y salir;* si se quiere salir sin guardar, pulsar la cruz superior.

11. CODIGO IMPLEMENTADO

En esta sección repasaremos el código implementado en los ficheros ***Inicio.cs***, ***Configuracion.cs***, ***Ensayo.cs*** y ***Ensayo_ciclos.cs***, realizando pequeñas explicaciones sobre el código. En estos ficheros se ha implementado el código de las funcionalidades de cada una de las ventanas que tiene el programa.

11.1 Inicio.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
```

Librerias que se cargan al iniciar las cuales son necesarias para la ejecución.

```
namespace ControlGPIB
{
    public partial class Inicio : Form
    {
        public static String dir="C:\\Prueba.txt";
        public static int puerto=11;
        public static int tiempo_muestreo=10;
        public static int periodo_muestreo=200;
```

Definición de las variables estáticas que guardan la dirección donde se guarda el fichero, el puerto del adaptador y el tiempo y período de muestreo, que se mostrará al abrir las ventanas de ensayo.

```
//-------------------------------------------------------------------//

        public Inicio()
        {
            InitializeComponent();
            obtener_configuracion_por_defecto("Parametros.txt");
        }
```

Inicialización de la ventana de inicio y llamada al método `obtener_configuracion_por_defecto`, pasándole como parámetro el nombre del fichero donde están guardados los datos de las cuatro variables anteriores.

```
        private void ConfiguracionAdaptador_Click(object sender,
EventArgs e)
        {
            Configuracion c = new Configuracion(); ;
            c.ShowDialog();

        }
```

Método que se ejecuta cuando se pulsa el botón *Configuración Adaptador*, que abre la ventana de configuración.

```
//-----------------------------------------------------------------//

        private void prueba_Click(object sender, EventArgs e)
        {
            Ensayo p = new Ensayo();
            try
            {
                p.ShowDialog();

            }
            catch (Exception ex) { }

        }
```

Método que se ejecuta cuando se pulsa el botón *Ensayo Simple* y que abre la ventana del ensayo simple.

```
//-----------------------------------------------------------------//


        private void salir_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
```

Método que se ejecuta cuando se pulsa el botón *Salir* y que cierra la aplicación.

```
//-----------------------------------------------------------------//


        private void obtener_configuracion_por_defecto(String
NombreFichero)
        {   Boolean ok;
            String ruta;
            int port, tiemp_muestreo, frec_muestreo;

            Accesofichero fich =
Accesofichero.accederFichero(NombreFichero);

            if (fich == null) {
```

```
                MessageBox.Show(this, "No se encuentra el fichero
Parametros.txt", "Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Asterisk);
            }
            else
            {
                ok = fich.ObtenerParametros(out ruta, out port, out
tiemp_muestreo, out frec_muestreo);
                if (ok == false)
                {
                    MessageBox.Show(this, "El fichero Parametros.txt
no contiene los parametros correctamente", "Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Asterisk);
                }
                else
                {
                    dir = ruta;
                    puerto = port;
                    tiempo_muestreo = tiemp_muestreo;
                    periodo_muestreo = frec_muestreo;
                    fich.cerrarFichero();
                }
            }
        }
```

Método que abre el fichero pasado como parámetro, que en nuestro caso es el fichero con los parámetros de configuración del programa.

```
//-------------------------------------------------------------------//

        private void ensayo_ciclico_Click(object sender, EventArgs e)
        {
            Ensayo_por_ciclos ec = new Ensayo_por_ciclos();
            try
            {
                ec.ShowDialog();

            }
            catch (Exception ex) { }

        }
```

Método que se ejecuta cuando se pulsa el botón *Ensayo Por Ciclos* y que abre la ventana del ensayo por ciclos.

```
        private void salirToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Application.Exit();
        }
```

Método que se ejecuta cuando se abre la pestaña *archivo* y se pulsa el botón *Salir.* Este método cierra la aplicación.

```
//-----------------------------------------------------------------//

        private void simpleToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Ensayo p = new Ensayo();
            try
            {
                p.ShowDialog();

            }
            catch (Exception ex) { }
        }
```

Método que se ejecuta cuando se abre la pestaña *archivo*, luego la pestaña *ensayo* y se pulsa el botón *Simple.* Este método abre la ventana de ensayo simple.

```
//-----------------------------------------------------------------//


        private void porCiclosToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Ensayo_por_ciclos ec = new Ensayo_por_ciclos();
            try
            {
                ec.ShowDialog();

            }
            catch (Exception ex) { }

        }
```

Método que se ejecuta cuando se abre la pestaña *archivo*, luego la pestaña *ensayo* y se pulsa el botón *Por ciclos.* Este método abre la ventana de ensayo por ciclos.

```
        private void configuraciónToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            Configuracion c = new Configuracion(); ;
            c.ShowDialog();
        }
```

Método que se ejecuta cuando se abre la pestaña *archivo* y se pulsa el botón *Configuracion.* Este método abre la ventana configuración del adaptador.

```
//-------------------------------------------------------------------//


        private void acercaDeToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            MessageBox.Show(this, "Proyecto final de carrera   \r
Jesús Brun   \r                                UPV", "Acerca de",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Asterisk);
        }
```

Método que se ejecuta cuando se abre la pestaña A*yuda* y se pulsa el botón *Acerca de.*

```
    }
}
```

11 .2 Configuración.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace ControlGPIB
{
    public partial class Configuracion : Form
    {
        public static String direccion;
        public static int com;
        private StreamWriter sw;
        public int temp_muestreo;
        public int per_muestreo;
```

Definición de las variables estáticas que guardan la dirección donde se guarda el fichero, el puerto del adaptador, el tiempo y período de muestreo, y un descriptor para abrir ficheros.

```
//-------------------------------------------------------------------//

        public Configuracion()
        {
            InitializeComponent();
            textBox1.Text = Inicio.dir;
            comboBox1.Text = "COM"+Inicio.puerto;
            textBox2.Text = Inicio.tiempo_muestreo.ToString();
            textBox3.Text = Inicio.periodo_muestreo.ToString();

        }
```

Inicializa la ventana de configuración escribiendo en cada *textBox* y *comboBox* el dato correspondiente almacenado en las variables globales de *inicio.cs*

```
        private void guardar_configuracion_Click(object sender,
EventArgs e)
        {
            try
            {

                sw = new StreamWriter(textBox1.Text);
                sw.Close();
                File.Delete(textBox1.Text);

                Inicio.dir = textBox1.Text;

                string c = comboBox1.SelectedItem.ToString();
                int pos = c.Length - 3;
                Inicio.puerto = Int32.Parse(c.Substring(3, pos));

                Inicio.tiempo_muestreo = temp_muestreo;
                Inicio.periodo_muestreo = per_muestreo;

                Close();
            }


            catch (DirectoryNotFoundException exc)
            {

                MessageBox.Show(this, "Ruta de fichero errónea",
"Error",
                       MessageBoxButtons.OK,
                       MessageBoxIcon.Asterisk);

                textBox1.Text = Inicio.dir;
            }
            catch (NullReferenceException exc)
            {
                MessageBox.Show(this, "No ha indicado correctamente el
puerto\rDebe selecionarlo de la lista", "Error",
                     MessageBoxButtons.OK,
                     MessageBoxIcon.Asterisk);

                comboBox1.Text = "COM" + Inicio.puerto;
            }

        }
```

Este método guarda los parámetros que se han establecido en la ventana de configuración y sale de la ventana. Se ejecuta al pulsar el botón *Guardar configuración y salir.* Muestra una pequeña ventana con un error si la ruta no existe.

```
        private void establecer_valores_por_defecto_Click(object
sender, EventArgs e)
        {   string c= comboBox1.SelectedItem.ToString();
            int pos = c.Length-3;
            Accesofichero
af=Accesofichero.accederEscribir("Parametros.txt");

            Bool
ok=af.guardarParametros(textBox1.Text,Int32.Parse(c.Substring(3,
pos)),Int32.Parse(textBox2.Text), Int32.Parse(textBox3.Text));
                if (ok == false)
                {
                    MessageBox.Show(this, "Error al escribir en
Parametros.txt", "Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Asterisk);
                }
                af.cerrarEscribir();
        }
```

Método que guarda en el fichero Parametros.txt la configuración establecida en la ventana de configuración; así, podremos recuperar esta información cuando volvamos a abrir el programa. Muestra un error si no se puede escribir en *Parametros.txt*

```
//-----------------------------------------------------------------//

        private void tiempo_muestreo_modificado(object sender,
EventArgs e)
        {
            try
            {
                if (textBox2.Text == "")
                    Inicio.tiempo_muestreo = 0;
                else if (Int32.Parse(textBox2.Text) > 596 ||
Int32.Parse(textBox2.Text) < 1)
                    throw new Exception();
                else
                    temp_muestreo = Int32.Parse(textBox2.Text);
            }
            catch (Exception ex)
            {
               MessageBox.Show(this, "Numero incorrecto\rEl numero
debe estar entre 1 y 596\ry no puede contener caracteres que no sean
numeros ", "Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Asterisk);


                textBox2.Text = Inicio.tiempo_muestreo.ToString();

            }
        }
```

Se ejecuta cuando se modifica el tiempo de muestreo en la ventana de configuración. Comprueba que el número de horas esté entre 1 y 596 y que sólo se hayan introducido caracteres numéricos. Mostrará un error si no se cumple lo anterior.

```
        private void frecuencia_muestreo_modificada(object sender,
EventArgs e)
        {
            try
            {
                if (textBox3.Text == "")
                    Inicio.periodo_muestreo = 0;
                else if (Int32.Parse(textBox3.Text) < 1)
                    throw new Exception();
                else
                    per_muestreo = Int32.Parse(textBox3.Text);
            }
            catch (Exception ex)
            {


                MessageBox.Show(this, "Numero incorrecto\rEl numero
debe estar entre 200 y 2147483647\ry no puede contener caracteres que
no sean numeros ", "Error",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Asterisk);


                textBox3.Text = Inicio.periodo_muestreo.ToString();

            }
        }

    }
}
```

Se ejecuta cuando se modifica la frecuencia de muestreo en la ventana de configuración. Comprueba que el número de milisegundos esté entre 200 y 2147483647 y que sólo se hayan introducido caracteres numéricos. Mostrará un error si no se cumple lo anterior.

11.3 Ensayo.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace ControlGPIB
{
    public partial class Ensayo : Form
    {
        int tiempoMuestreo=Inicio.tiempo_muestreo;
        int frecuenciaMuestreo=Inicio.periodo_muestreo;
        private int puerto=Inicio.puerto;
        private String dir=Inicio.dir;
        private Boolean activo=false;

        StreamWriter sw;
        Timer tReloj, final, fuerza_hora, retraso_inicial_fuerza_hora;

        int numero_muestra=0;
        double fuerza_max_hora = 0;
```

Definición de variables globales, como son los "timers" que se utilizaran para controlar el tiempo, la variable que guarda la fuerza máxima durante la última hora, el descriptor de fichero con el que se guardaran los datos y la variable que almacena el número de muestra. También se pasan los 4 parámetros de configuración de la clase *inicio* a la clase actual.

Timers o relojes utilizados:

`tReloj`: Marca cada vez que se ha de pedir la intensidad a la fuente. Se programa con el valor del período de muestreo introducido. Cada vez que se cumple el tiempo, llama al método *recibir_intensidad.*

`final`: Marca el final del muestreo. El tiempo al cual se programa es el de la variable *tiempoMuestreo.* Cada vez que se cumple el tiempo, llama al método *finalizar_muestreo* .

`fuerza_hora`: Sirve para controlar cada hora la intensidad máxima consumida. Llama al método *borrar_fuerza_max.*

`retraso_inicial_fuerza_hora`: Se utiliza para introducir un retraso de 3 segundos al comienzo del muestreo antes de activar el reloj de fuerza_hora. Llama al método *iniciar_fuerza_hora.*

```
        public Ensayo()
        {

            InitializeComponent();
            textBox1.Text = tiempoMuestreo.ToString();
            textBox2.Text = frecuenciaMuestreo.ToString();

            iniciaPuerto();

        }
```

Inicializa la ventana de ensayo, escribe en los *textBox* el tiempo de muestreo y la frecuencia de muestreo y llama al método *iniciaPuerto*.

```
//-------------------------------------------------------------------//


        private void iniciaPuerto()
        {
            puertoGPIB.RTSEnable = true;

            puertoGPIB.CommPort = (short)puerto;
            puertoGPIB.RThreshold = 10;
            puertoGPIB.Settings = "9600,n,8,1";
            puertoGPIB.DTREnable = true;
            puertoGPIB.Handshaking =
MSCommLib.HandshakeConstants.comRTS;
            puertoGPIB.InputLen = 0;
            puertoGPIB.NullDiscard = false;
            puertoGPIB.OnComm += new System.EventHandler(this.OnComm);

            try
            {
                puertoGPIB.PortOpen = true;
            }
            catch (Exception excp)
            {
                MessageBox.Show(this, "El puerto no se puede abrir",
"Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Asterisk);
                Close();


            }

        }
```

Método encargado de iniciar el puerto; es decir, configura los parámetros del puerto COM para que funcione correctamente. La variable *puertoGPIB* es el control MSCOMM definido en tiempo de diseño en la ventana de Windows Form. Para ver el significado de cada una de los parámetros de puertoGPIB, ver la sección 6. Si no se puede abrir correctamente, muestra un error.

```
        private void OnComm(object sender, EventArgs e)
        {
            if (puertoGPIB.CommEvent ==
(short)MSCommLib.OnCommConstants.comEvReceive)
            {

                    dar_formato((string)puertoGPIB.Input);

            }
        }
```

Este método se ejecuta cada vez se produzca un evento en el puerto COM. En nuestro caso sólo nos interesa el evento de tipo *comEvReceive,* que se produce cuando llega algún carácter al buffer de entrada del puerto, como ocurre cuando se recibe la intensidad que se ha pedido a la fuente. Una vez comprobado que el evento sea de este tipo, se pasarán los caracteres recibidos al método *dar_formato,* donde se manipularán los caracteres para representarlo en el formato correcto.

```
//-----------------------------------------------------------------//
```

```
        private void guardar_en_fichero(String e)
        {
            double tiempo=(numero_muestra *
((Double)frecuenciaMuestreo / 1000)) - ((Double)frecuenciaMuestreo /
1000);

            sw.WriteLine(e + "     " + numero_muestra + "    " +
tiempo);
        }
```

Método encargado de guardar en el fichero los datos de intensidad consumida que recibimos de la fuente. Para ello utiliza el descriptor de fichero *sw* que se ha definido al principio. El formato en el que se escribe en el fichero es:

Intensidad consumida Numero de muestra Tiempo

```
        private void dar_formato(String e)
        {

            if (e[7] == '+' || e[6] == '+')
            {
                int posicion_del_punto = e.IndexOf(".");
                int posicion_de_la_E = e.IndexOf("E");
                double aux =
double.Parse(e.Substring(posicion_del_punto + 1, (posicion_de_la_E -
posicion_del_punto) - 1));
                int exp = Int32.Parse(e[posicion_de_la_E +
2].ToString());
                aux = aux / Math.Pow(10, ((posicion_de_la_E -
posicion_del_punto) - 1) - exp);
                aux = (Int32.Parse(e[0].ToString()) * Math.Pow(10,
exp)) + aux;
                guardar_en_fichero(aux.ToString());
                if (aux > fuerza_max_hora) fuerza_max_hora = aux;

            }

        }
```

Método encargado de recibir los caracteres que representan la intensidad en el formato que la envía la fuente de alimentación; este formato es exponencial, y lo pasa a un formato más legible en amperios. Una vez que ya tiene el dato en el formato correcto se lo envía al método *guardar_en_ fichero* para que lo escriba en el fichero.

```
//-----------------------------------------------------------------//


        private void inicia_muestreo_Click(object sender, EventArgs e)
        {
            if (activo==false)
            {
                if (frecuenciaMuestreo >= 200)
                {

                    try
                    {
                        sw = new StreamWriter(dir);
                    }
                    catch (Exception exc)
                    {
                        MessageBox.Show(this, "Ruta de fichero
erronea", "Error",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Asterisk);
                        puertoGPIB.PortOpen = false;
                        Close();
                    }



                    tReloj = new Timer();
                    tReloj.Interval = frecuenciaMuestreo;
```

```
                    tReloj.Tick += new
EventHandler(this.recibir_intensidad);


                    final = new Timer();
                    final.Interval = tiempoMuestreo * 3600000;
                    final.Tick += new
EventHandler(this.finalizar_muestreo);

                    fuerza_hora = new Timer();
                    fuerza_hora.Interval = 3600000;
                    fuerza_hora.Tick += new
EventHandler(this.borrar_fuerza_max);

                    retraso_inicial_fuerza_hora = new Timer();
                    retraso_inicial_fuerza_hora.Interval = 3000;
                    retraso_inicial_fuerza_hora.Tick += new
EventHandler(this.iniciar_fuerza_hora);

                    progreso.Value = 0;
                    numero_muestra = 0;

                    activo = true;

                    puertoGPIB.Output = "REN\r";
                    puertoGPIB.Output = "OUTP:STAT 1\r";
                    retraso_inicial_fuerza_hora.Start();
                    tReloj.Start();
                    final.Start();

                }
                else
                {

                    MessageBox.Show(this, "El periodo de muestreo debe
ser como mínimo de 200 ms\rA menor periodo mayor probabilidad de
perdida de algun dato y \rmenor precisión en el tiempo de muestreo.",
"Advertencia",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Asterisk);
                }
            }
            else
            {
                MessageBox.Show(this, "Ya se ha iniciado el muestreo",
"Error",
                 MessageBoxButtons.OK,
                 MessageBoxIcon.Asterisk);
            }
        }
```

Se ejecuta al pulsar el botón *Iniciar muestreo.* Este método está encargado de iniciar el muestreo. Para ello abre el descriptor de fichero donde se guardaran los datos, activa la salida de la fuente, configura los relojes *tReloj, fuerza_hora, final y retraso_inicial_fuerza_hora* e inicia todos los relojes menos el *fuerza_hora,* que se iniciará en 3s cuando que finalice el reloj *retraso_inicial_fuerza_hora.*

```
        private void recibir_intensidad(object sender, EventArgs e)
        {
            puertoGPIB.Output = "a\r";
            puertoGPIB.Output = "MEAS:CURR?\r";


            numero_muestra++;

            try
            {
                progreso.Value = (int)(numero_muestra * 100) /
(tiempoMuestreo * 3600000 / (frecuenciaMuestreo));
            }
            catch (Exception exc) { progreso.Value = 100; }

            label_fuerza.Text = fuerza_max_hora.ToString() + " A";


        }
```

Envía la orden a la fuente para que le suministre la corriente consumida en ese instante. Muestra también, mediante un *label,* la corriente máxima consumida en la última hora, aumenta la variable que guarda el número de muestra y actualiza la barra de progreso.

```
//-------------------------------------------------------------------//

        private void finalizar_muestreo(object sender, EventArgs e)
        {
            if (activo == true)
            {
                tReloj.Stop();
                final.Stop();
                fuerza_hora.Stop();

                puertoGPIB.Output = "REN\r";
                puertoGPIB.Output = "OUTP:STAT 0\r";



                progreso.Value = 100;
                activo = false;
                sw.Close();

                puertoGPIB.Output = "REN\r";
                puertoGPIB.Output = "SOUR:VOLT 0\r";

                MessageBox.Show(this, "Muestreo finalizado", "OK",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Asterisk);
            }
            else
            {
                MessageBox.Show(this, "No hay ningun muestreo activo",
"Error",
```

```
                MessageBoxButtons.OK,
                MessageBoxIcon.Asterisk);
            }
        }
```

Método que finaliza el muestreo. Para ello para el reloj de muestreo, el reloj que controla el tiempo de muestreo y el reloj de fuerza máxima durante la última hora. También cierra el descriptor de fichero abierto, pone la variable activo a false y envía a la fuente la orden de desactivar la salida y ponga a 0 el voltaje. Luego muestra un mensaje diciendo que ha finalizado el muestreo.

```
//-----------------------------------------------------------------//

        private void envia_voltaje_Click(object sender, EventArgs e)
        {
            String volt = "0";

            try
            {
                volt = comboBox1.SelectedItem.ToString();
                int pos = volt.IndexOf("V");
                volt = volt.Substring(0, pos);

                puertoGPIB.Output = "REN\r";
                puertoGPIB.Output = "SOUR:VOLT " + volt + "\r";


            }
            catch (Exception excp)
            {
                MessageBox.Show(this, "No ha indicado el voltaje",
"Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Asterisk);
            }
        }
```

Se ejecuta al pulsar el botón *Envia Voltaje* . Envía a la fuente a través del puerto GPIB el voltaje indicado para que lo suministre la fuente.

```
//-----------------------------------------------------------------//

        private void activa_salida_Click(object sender, EventArgs e)
        {
            puertoGPIB.Output = "REN\r";
            puertoGPIB.Output = "OUTP:STAT 1\r";
        }
```

Envía a la fuente, a través del puerto GPIB, la orden de activar la salida de la fuente de alimentación.

```
        private void desactiva_salida_Click(object sender, EventArgs
e)
        {
            puertoGPIB.Output = "REN\r";
            puertoGPIB.Output = "OUTP:STAT 0\r";
        }
```

Envía a la fuente, a través del puerto GPIB, la orden de desactivar la salida de la fuente de alimentación.

```
//-----------------------------------------------------------------//
```

```
        private void tiempo_muestreo_modificado(object sender,
EventArgs e)
        {
            try
            {
                if (textBox1.Text == "")
                    tiempoMuestreo = 0;
                else if (Int32.Parse(textBox1.Text) > 596 ||
Int32.Parse(textBox1.Text) < 1)
                    throw new Exception();
                else
                    tiempoMuestreo = Int32.Parse(textBox1.Text);
            }
            catch (Exception ex)
            {

                MessageBox.Show(this, "Numero incorrecto\rEl numero
debe estar entre 1 y 596\ry no puede contener caracteres que no sean
numeros ", "Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Asterisk);

                tiempoMuestreo = Inicio.tiempo_muestreo;
                textBox1.Text = tiempoMuestreo.ToString();

            }
        }
```

Se ejecuta cuando se modifica el tiempo de muestreo en la ventana de configuración. Comprueba que el número de horas esté entre 1 y 596 y que sólo se hayan introducido caracteres numéricos. Mostrará un error si no se cumple lo anterior.

```
        private void frecuencia_muestreo_modificada(object sender,
EventArgs e)
        {
            try
            {
                if (textBox2.Text == "")
                    frecuenciaMuestreo = 0;
                else if (Int32.Parse(textBox2.Text) < 1)
                    throw new Exception();
                else
                frecuenciaMuestreo = Int32.Parse(textBox2.Text);
                }
                catch (Exception ex)
                {



                    MessageBox.Show(this, "Numero incorrecto\rEl numero
debe estar entre 200 y 2147483647\ry no puede contener caracteres que
no sean numeros ", "Error",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Asterisk);

                        frecuenciaMuestreo = Inicio.periodo_muestreo;
                        textBox2.Text = frecuenciaMuestreo.ToString();

                }
        }
```

Se ejecuta cuando se modifica la frecuencia de muestreo en la ventana de configuración. Comprueba que el número de milisegundos esté entre 200 y 2147483647 y que sólo se hayan introducido caracteres numéricos. Mostrará un error si no se cumple lo anterior.

```
//-----------------------------------------------------------------//
```

```
        private void iniciar_fuerza_hora(object sender, EventArgs e)
        {
            fuerza_hora.Start();
            fuerza_max_hora = 0;
            retraso_inicial_fuerza_hora.Stop();
        }
```

Inicia el reloj que controla la fuerza máxima durante la última hora, pone a cero la variable fuerza máxima y para el reloj del retraso inicial, utilizado antes de iniciar el reloj fuerza_hora.

```
        private void borrar_fuerza_max(object sender, EventArgs e)
        {
            fuerza_max_hora = 0;
        }
```

Método que borra la fuerza máxima durante la última hora, almacenada en la variable *fuerza_max_hora*.

```
//-----------------------------------------------------------------//

        private void cierre_del_form(object sender,
FormClosingEventArgs e)
        {
            try
            {
                if(activo==true) finalizar_muestreo(sender,e);
                puertoGPIB.PortOpen = false;
            }
            catch (Exception ex) { }

        }
```

Método que se ejecuta cuando se cierra la ventana. Si el muestreo está activado, llama al método *finalizar_muestreo* para finalizarlo y parar los relojes. También cierra el puerto COM que hemos abierto.

```
//-----------------------------------------------------------------//

        private void fin_muestreo_Click(object sender, EventArgs e)
        {
            finalizar_muestreo(sender,e);
        }


    }
}
```

Se ejecuta al pulsar el botón *Finalizar muestreo.* Hace una llamada *finalizar_muestreo* para finalizarlo.

11 .4 Ensayo_ciclos.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;

namespace ControlGPIB
{
    public partial class Ensayo_por_ciclos : Form
    {

        int tiempoMuestreo = Inicio.tiempo_muestreo;
        int frecuenciaMuestreo = Inicio.periodo_muestreo;
        int duracionCiclo1 = 1;
        int duracionCiclo2 = 1;
        private int puerto = Inicio.puerto;
        private String dir = Inicio.dir;
        private Boolean activo = false;


        int ciclo=1;
        double tiempo=0;
        double periodo_en_segundos;

        StreamWriter sw;
        Timer tReloj, final,
fuerza_hora,ciclo1,ciclo2,retraso_inicial_fuerza_hora, barra_progreso;

        int numero_muestra = 0;
        double fuerza_max_hora = 0;
```

Definición de variables globales, como son los “timers” que se utilizarán para controlar el tiempo, la variable que guarda la fuerza máxima durante la última hora, el descriptor de fichero con el que se guardaran los datos y la variable que almacena el número de muestra, el tiempo y el ciclo en el que se encuentra. También se pasan los 4 parámetros de configuración de la clase *inicio* a la clase actual.

Timers o relojes utilizados:

`tReloj`: Marca cada vez que se ha de pedir la intensidad a la fuente. Se programa con el valor del período de muestreo introducido. Cada vez que se cumple el tiempo llama al método *recibir_intensidad.*

`final`: Marca el final del muestreo. El tiempo al cual se programa es el de la variable *tiempoMuestreo.* Cada vez que se cumple el tiempo, llama al método *finalizar_muestreo* .

`fuerza_hora:` Sirve para controlar cada hora la intensidad máxima consumida. Llama al método *borrar_fuerza_max.*

`retraso_inicial_fuerza_hora:` Se utiliza para introducir un retraso de 3 segundos al comienzo del muestreo antes de activar el reloj de fuerza_hora. Llama al método *iniciar_fuerza_hora.*

`ciclo1:` Controla la duración del primer ciclo, que es el ciclo que está activo. El tiempo al cual se programa, es el tiempo introducido en el recuadro *Duración 1er ciclo (Ciclo de funcionamiento).* Llama al método *fin_Ciclo1* .

`ciclo2:` Controla la duración del segundo ciclo, que es el ciclo que está activo. El tiempo al cual se programa, es el tiempo introducido en el recuadro *Duración 2er ciclo (Ciclo de parada).* Llama al método *fin_Ciclo2* .

barra_progreso: Sirve para aumentar el valor de la barra que muestra el progreso. El tiempo, al cual se programa, es a una veinteava parte del tiempo total del muestreo. Llama al método *aumentar_progreso* .

```
//-------------------------------------------------------------------//
        public Ensayo_por_ciclos()

        {
            InitializeComponent();
            textBox1.Text = tiempoMuestreo.ToString();
            textBox2.Text = frecuenciaMuestreo.ToString();
            textBox3.Text = duracionCiclo1.ToString();
            textBox4.Text = duracionCiclo2.ToString();

            iniciaPuerto();
        }
```

Inicializa la ventana de ensayo por ciclos, escribe en los *textBox* el tiempo de muestreo y la frecuencia de muestreo y llama método *iniciaPuerto*.

```
//-------------------------------------------------------------------//
        private void iniciaPuerto()
        {
            puertoGPIB.RTSEnable = true;
            puertoGPIB.CommPort = (short)puerto;
            puertoGPIB.RThreshold = 10;
            puertoGPIB.Settings = "9600,n,8,1";
            puertoGPIB.DTREnable = true;
            puertoGPIB.Handshaking =
MSCommLib.HandshakeConstants.comRTS;
            puertoGPIB.InputLen = 0;
            puertoGPIB.NullDiscard = false;
```

```
            puertoGPIB.OnComm += new System.EventHandler(this.OnComm);

            try
            {
                puertoGPIB.PortOpen = true;
            }
            catch (Exception excp)
            {
                MessageBox.Show(this, "El puerto no se puede abrir",
"Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Asterisk);
                Close();


            }

        }
```

Método encargado de iniciar el puerto, es decir configura los parametros del puerto COM para que funcione correctamente. La variable *puertoGPIB* es el control MSCOMM definido en tiempo de diseño en la ventana de WIndos Form. Para ver el significado de cada una de los parametros de *puertoGPIB* ver la seccion 6. Si no se puede abrir correctamente muestra un error.

```
//-------------------------------------------------------------------//

        private void OnComm(object sender, EventArgs e)
        {
            if (puertoGPIB.CommEvent ==
(short)MSCommLib.OnCommConstants.comEvReceive)
            {

                dar_formato((string)puertoGPIB.Input);

            }

        }
```

Este método se ejecuta cada vez se produzca un evento en el puerto COM. En nuestro caso sólo nos interesa el evento de tipo *comEvReceive,* que se produce cuando llega algún carácter al buffer de entrada del puerto, como ocurre cuando se recibe la intensidad que se ha pedido a la fuente. Una vez comprobado que sea de este tipo el evento, se pasarán los caracteres recibidos al método *dar_formato* donde se manipularán los caracteres para representarlo en el formato correcto.

```
        private void guardar_en_fichero(String e)
        {


            sw.WriteLine(e + "      " + numero_muestra + "     " +
Math.Round(tiempo,2));
        }
```

Método encargado de guardar en el fichero los datos de intensidad consumida que recibimos de la fuente; para ello utiliza el descriptor de fichero *sw* que se ha definido al principio. El formato en el que se escribe en el fichero es:

Intensidad consumida Numero de muestra Tiempo

```
//-------------------------------------------------------------------//


        private void dar_formato(String e)
        {

            if (e[7] == '+' || e[6] == '+')
            {
                int posicion_del_punto = e.IndexOf(".");
                int posicion_de_la_E = e.IndexOf("E");
                double aux =
double.Parse(e.Substring(posicion_del_punto + 1, (posicion_de_la_E -
posicion_del_punto) - 1));
                int exp = Int32.Parse(e[posicion_de_la_E +
2].ToString());
                aux = aux / Math.Pow(10, ((posicion_de_la_E -
posicion_del_punto) - 1) - exp);
                aux = (Int32.Parse(e[0].ToString()) * Math.Pow(10,
exp)) + aux;
                guardar_en_fichero(aux.ToString());
                if (aux > fuerza_max_hora)
                { fuerza_max_hora = aux;
                label_fuerza.Text = fuerza_max_hora.ToString() + " A";
                }

            }

        }
```

Método encargado de recibir los caracteres que representan la intensidad en el formato que la envía la fuente de alimentación; este formato es exponencial, y lo pasa a un formato más legible en amperios. Una vez que ya tiene el dato en el formato correcto se lo envía al método *guardar_en_ fichero* para que lo escriba en el fichero.

```
        private void inicia_muestreo_Click(object sender, EventArgs e)
        {
            if (activo == false)
            {
                if (frecuenciaMuestreo >= 200)
                {


                try
                {
                    sw = new StreamWriter(dir);
                }
                catch (Exception exc)
                {
                    MessageBox.Show(this, "Ruta de fichero erronea",
"Error",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Asterisk);
                    puertoGPIB.PortOpen = false;
                    Close();
                }

                tReloj = new Timer();
                tReloj.Interval = frecuenciaMuestreo;
                tReloj.Tick += new
EventHandler(this.recibir_intensidad);


                final = new Timer();
                final.Interval = tiempoMuestreo * 3600000;
                final.Tick += new
EventHandler(this.finalizar_muestreo);

                fuerza_hora = new Timer();
                fuerza_hora.Interval = 3600000;
                fuerza_hora.Tick += new
EventHandler(this.borrar_fuerza_max);

                retraso_inicial_fuerza_hora = new Timer();
                retraso_inicial_fuerza_hora.Interval = 3000;
                retraso_inicial_fuerza_hora.Tick += new
EventHandler(this.iniciar_fuerza_hora);


                ciclo1 = new Timer();
                ciclo1.Interval = duracionCiclo1 * 3600000;
                ciclo1.Tick += new EventHandler(this.fin_Ciclo1);

                ciclo2 = new Timer();
                ciclo2.Interval = duracionCiclo2 * 3600000;
                ciclo2.Tick += new EventHandler(this.fin_Ciclo2);

                barra_progreso = new Timer();
                barra_progreso.Interval=(tiempoMuestreo * 3600000)/20;
                barra_progreso.Tick += new
EventHandler(this.aumentar_progreso);



                progreso.Value = 0;
                numero_muestra = 0;
```

```
                ciclo = 1;
                tiempo = 0;

                activo = true;
                periodo_en_segundos=((Double)frecuenciaMuestreo /
1000);
                tiempo = tiempo - periodo_en_segundos;


                puertoGPIB.Output = "REN\r";
                puertoGPIB.Output = "OUTP:STAT 1\r";

                tReloj.Start();
                retraso_inicial_fuerza_hora.Start();
                barra_progreso.Start();
                ciclo1.Start();
                final.Start();

                }
                else
                {

                    MessageBox.Show(this, "El periodo de muestreo debe
ser como mínimo de 200 ms\rA menor periodo mayor probabilidad de
perdida de algun dato y \rmenor precisión en el tiempo de muestreo.",
"Advertencia",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Asterisk);
                }

            }
            else
            {
                MessageBox.Show(this, "Ya se ha iniciado el muestreo",
"Error",
                 MessageBoxButtons.OK,
                 MessageBoxIcon.Asterisk);
            }
        }
```

Se ejecuta al pulsar el botón *Iniciar muestreo.* Este método está encargado de iniciar el muestreo; para ello abre el descriptor de fichero donde se guardaran los datos, activa la salida de la fuente y configura los relojes *tReloj, fuerza_hora, final*, ciclo1, ciclo2, barra_progreso *y retraso_inicial_fuerza_hora* . Se inician todos menos el *fuerza_hora,* que se iniciará en 3s cuando finalice el reloj *retraso_inicial_fuerza_hora,* y ciclo2, que se iniciará al finalizar el ciclo1.

```
        private void recibir_intensidad(object sender, EventArgs e)
        {
            tiempo = tiempo + periodo_en_segundos;
            numero_muestra++;

            puertoGPIB.Output = "a\r";
            puertoGPIB.Output = "MEAS:CURR?\r";

        }
```

Envía la orden a la fuente para que le suministre la corriente consumida en ese instante. Aumenta también la variable que guarda el número de muestra.

```
//-----------------------------------------------------------------//

        private void fin_Ciclo1(object sender, EventArgs e)
        {
            desactiva_salida_Click(sender, e);
            tReloj.Stop();
            ciclo1.Stop();
            ciclo2.Start();
            fuerza_hora.Stop();
            tiempo = (ciclo * (duracionCiclo1 + duracionCiclo2)) -
((Double)frecuenciaMuestreo / 1000);
            desactiva_salida_Click(sender, e);

        }
```

Se ejecuta cuanto finaliza el reloj que marca el primer ciclo. Llama a *desactiva_salida_Click* para desactivar la salida de la fuente, pone en marcha el reloj ciclo2, que marca el tiempo que ha de estar parado, y para los relojes *tReloj, fuerza_hora* y *ciclo1.*

```
//-----------------------------------------------------------------//

        private void fin_Ciclo2(object sender, EventArgs e)
        {
            activa_salida_Click(sender,e);
            ciclo2.Stop();
            retraso_inicial_fuerza_hora.Start();
            ciclo++;
            ciclo1.Start();
            activa_salida_Click(sender, e);
            tReloj.Start();

        }
```

Se ejecuta cuanto finaliza el reloj que marca el segundo ciclo. Llama a *activa_salida_Click* para activar la salida de la fuente, para el reloj *ciclo2* y pone en marcha los relojes ciclo1 y tReloj.

```
        private void finalizar_muestreo(object sender, EventArgs e)
        {
            if (activo == true)
            {
                barra_progreso.Stop();
                tReloj.Stop();
                final.Stop();
                fuerza_hora.Stop();
                ciclo1.Stop();
                ciclo2.Stop();

                puertoGPIB.Output = "REN\r";
                puertoGPIB.Output = "OUTP:STAT 0\r";



                progreso.Value = 100;
                activo = false;
                sw.Close();

                puertoGPIB.Output = "REN\r";
                puertoGPIB.Output = "SOUR:VOLT 0\r";

                MessageBox.Show(this, "Muestreo finalizado", "OK",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Asterisk);
            }
            else
            {
                MessageBox.Show(this, "No hay ningun muestreo activo",
"Error",
                 MessageBoxButtons.OK,
                 MessageBoxIcon.Asterisk);
            }

        }
```

Método que finaliza el muestreo. Para ello, para el reloj que controla el tiempo de muestreo y el reloj de fuerza máxima durante la última hora. También cierra el descriptor de fichero abierto, pone la variable activo a false y envía a la fuente la orden de desactivar la salida y ponga a 0 el voltaje. Luego muestra un mensaje diciendo que ha finalizado el muestreo.

```
//-----------------------------------------------------------------//

        private void envia_voltaje_Click(object sender, EventArgs e)
        {
            String volt = "0";

            try
            {
                volt = comboBox1.SelectedItem.ToString();
                int pos = volt.IndexOf("V");
                volt = volt.Substring(0, pos);

                puertoGPIB.Output = "REN\r";
                puertoGPIB.Output = "SOUR:VOLT " + volt + "\r";
```

```
            }
            catch (Exception excp)
            {
                MessageBox.Show(this, "No ha indicado el voltaje",
"Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Asterisk);
            }
        }
```

Se ejecuta al pulsar el botón *Envia Voltaje* . Envía a la fuente a través del puerto GPIB el voltaje indicado para que lo suministre la fuente.

```
//-------------------------------------------------------------------//
        private void activa_salida_Click(object sender, EventArgs e)
        {
            puertoGPIB.Output = "REN\r";
            puertoGPIB.Output = "OUTP:STAT 1\r";
        }
```

Envía a la fuente a través del puerto GPIB la orden de activar la salida de la fuente de alimentación.

```
//-------------------------------------------------------------------//

        private void desactiva_salida_Click(object sender, EventArgs
e)
        {
            puertoGPIB.Output = "REN\r";
            puertoGPIB.Output = "OUTP:STAT 0\r";
        }
```

Envía a la fuente a través del puerto GPIB la orden de desactivar la salida de la fuente de alimentación.

```
//-------------------------------------------------------------------//
        private void tiempo_muestreo_modificado(object sender,
EventArgs e)
        {
            try
            {
                if (textBox1.Text == "")
                    tiempoMuestreo = 0;
                else if (Int32.Parse(textBox1.Text) > 596 ||
Int32.Parse(textBox1.Text) < 1)
                    throw new Exception();
                else
                    tiempoMuestreo = Int32.Parse(textBox1.Text);
            }
            catch (Exception ex)
            {
```

```
                MessageBox.Show(this, "Numero incorrecto\rEl numero
debe estar entre 1 y 596\ry no puede contener caracteres que no sean
numeros ", "Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Asterisk);

                tiempoMuestreo = Inicio.tiempo_muestreo;
                textBox1.Text = tiempoMuestreo.ToString();

            }
        }
```

Se ejecuta cuando se modifica el tiempo de muestreo en el *textbox* de la ventana de configuración. Comprueba que el número de horas esté entre 1 y 596 y que sólo se hayan introducido caracteres numéricos. Mostrara un error si no se cumple lo anterior.

```
//-------------------------------------------------------------------//
        private void frecuencia_muestreo_modificada(object sender,
EventArgs e)
        {
            try
            {
                if (textBox2.Text == "")
                    frecuenciaMuestreo = 0;
                else if (Int32.Parse(textBox2.Text) < 1)
                    throw new Exception();
                else
                    frecuenciaMuestreo = Int32.Parse(textBox2.Text);
            }
            catch (Exception ex)
            {



                MessageBox.Show(this, "Numero incorrecto\rEl numero
debe estar entre 200 y 2147483647\ry no puede contener caracteres que
no sean numeros ", "Error",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Asterisk);

                frecuenciaMuestreo = Inicio.periodo_muestreo;
                textBox2.Text = frecuenciaMuestreo.ToString();

            }
        }
```

Se ejecuta cuando se modifica la frecuencia de muestreo en el *textbox* de la ventana de configuración. Comprueba que el número de milisegundos esté entre 200 y 2147483647 y que sólo se hayan introducido caracteres numéricos. Mostrará un error si no se cumple lo anterior.

```
        private void duracion_ciclo1(object sender, EventArgs e)
        {
            try
            {
                if (textBox3.Text == "")
                    duracionCiclo1 = 0;
                else if (Int32.Parse(textBox3.Text) > 596 ||
Int32.Parse(textBox3.Text) < 1)
                    throw new Exception();
                else
                    duracionCiclo1 = Int32.Parse(textBox3.Text);
            }
            catch (Exception ex)
            {

                MessageBox.Show(this, "Numero incorrecto\rEl numero
debe estar entre 1 y 596\ry no puede contener caracteres que no sean
numeros ", "Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Asterisk);

                duracionCiclo1 = 10;
                textBox3.Text = duracionCiclo1.ToString();

            }
        }
```

Método que se ejecuta cuando se modifica la duración del primer ciclo en el *textbox* correspondiente de la ventana de configuración. Comprueba que el número de horas esté entre 1 y 596 y que sólo se hayan introducido caracteres numéricos. Mostrará un mensaje de error si no se cumple lo anterior.

```
//-----------------------------------------------------------------//

        private void duracion_ciclo2(object sender, EventArgs e)
        {
            try
            {
                if (textBox4.Text == "")
                    duracionCiclo2 = 0;
                else if (Int32.Parse(textBox4.Text) > 596 ||
Int32.Parse(textBox4.Text) < 1)
                    throw new Exception();
                else
                    duracionCiclo2 = Int32.Parse(textBox4.Text);
            }
            catch (Exception ex)
            {

                MessageBox.Show(this, "Numero incorrecto\rEl numero
debe estar entre 1 y 596\ry no puede contener caracteres que no sean
numeros ", "Error",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Asterisk);

                duracionCiclo2 = 10;
                textBox4.Text = duracionCiclo2.ToString();
```

```
        }
    }
```

Método que se ejecuta cuando se modifica la duración del segundo ciclo en el *textbox* correspondiente de la ventana de configuración. Comprueba que el número de horas esté entre 1 y 596 y que sólo se hayan introducido caracteres numéricos. Mostrará un mensaje de error si no se cumple lo anterior.

```
//-----------------------------------------------------------------//

    private void iniciar_fuerza_hora(object sender, EventArgs e)
    {
        fuerza_hora.Start();
        fuerza_max_hora = 0;
        retraso_inicial_fuerza_hora.Stop();
    }
```

Inicia el reloj que controla la fuerza máxima durante la última hora, pone a cero la variable fuerza máxima y para el reloj del retraso inicial utilizado antes de iniciar el reloj fuerza_hora.

```
//-----------------------------------------------------------------//

    private void aumentar_progreso(object sender, EventArgs e)
    {
        try
        {
            progreso.Value = progreso.Value + 5;
        }
        catch (Exception ex) { progreso.Value = 100; }
    }
```

Aumenta la barra de barra de progreso mostrada en la ventana.

```
//-----------------------------------------------------------------//

    private void borrar_fuerza_max(object sender, EventArgs e)
    {
        fuerza_max_hora = 0;
    }
```

Método que borra la fuerza máxima durante la última hora, almacenada en la variable *fuerza_max_hora*.

```
        private void cierre_del_form(object sender,
FormClosingEventArgs e)
        {
            try
            {
                if (activo == true) finalizar_muestreo(sender, e);
                puertoGPIB.PortOpen = false;
            }
            catch (Exception ex) { }

        }
```

Método que se ejecuta cuando se cierra la ventana. Si el muestreo está activado, llama al método *finalizar_muestreo* para finalizarlo y parar los relojes. También cierra el puerto COM que hemos abierto.

```
//-----------------------------------------------------------------//

        private void fin_muestreo_Click(object sender, EventArgs e)
        {
            finalizar_muestreo(sender, e);
        }



    }
}
```

Se ejecuta al pulsar el botón *Finalizar muestreo.* Hace una llamada *finalizar_muestreo* para finalizar.

12. CONCLUSIONES

El desarrollo de este proyecto me ha servido para madurar en el mundo de la informática y ampliar mis conocimientos en diversos campos, como son la programación, el desarrollo de software o la manipulación y programación de dispositivos. También ha ampliado mi experiencia al tener que enfrentarme a un problema real, ya que había que desarrollar desde cero un sistema para la vida real sin contar con casi ayuda.

Este ha sido el trabajo más amplio que he realizado hasta el momento y el más complejo, ya que antes los trabajos realizados habían sido bastante simples y habían sido guiados contando con bastante información, pero en este, el único material con el que contaba era con el manual de la fuente y del adaptador y he tenido que ir buscando información sin tener una guía que seguir.

Antes de realizar este proyecto no conocía nada sobre el estándar GPIB, ni sobre los comandos SCPI, ni sobre la programación de aparatos como fuentes o similares ni sobre muchos conceptos utilizados. Anteriormente, mi experiencia en programación con aparatos había sido con pequeños y simples dispositivos como teclados y displays de segmentos digitales.

Esta experiencia me ha servido para ver que cuando uno empieza un proyecto parece que no vas a ser capaz de realizarlo, ya que al principio no tienes idea de casi nada, pero a medida que vas investigando y haciendo pruebas vas comprendiendo las cosas y consiguiendo el objetivo. También ha habido momentos en los que pasas mucho tiempo intentando conseguir algo y no te sale y parece que es imposible pero al final se consigue.

Quiere agradecer a Jorge Más Estelles el haber ofrecido este proyecto así como agradecerle el material y la ayuda prestada. También agradecer a Ángel Rodas Jordá del DISCA que me haya ayudado en algún aspecto en el que le he pedido ayuda.