Universidad del Valle
Departamento de Física

Trabajo de Grado presentado como requisito parcial
para optar por el título de Físico

Director : Otto Vergara García, Dr.Rer.Nat

Programa de manejo y driver de comunicación para el analizador multicanal Mossbauer AMCMB96

Christian David Latorre González
<christian.latorre@correounivalle.edu.co>

Santiago de Cali, Septiembre de 2014

AGRADECIMIENTOS

- Al grupo de instrumentación y Física aplicada (GIFA), por haberme acogido como un integrante más.
- Al grupo de investigación en espectroscopía Mössbauer, por brindarme el apoyo logístico necesario.
- Al profesor Otto Vergara, por su guía, amistad, acompañamiento e infinita paciencia.
- Al profesor Alberto Sánchez Asseff, por su apoyo, sus enseñanzas, amistad y numerosos consejos.
- Al profesor Jesús Tabares, por su inagotable fe en la terminación de este proyecto.
- A mis amigos, quienes hicieron llevadera y amena mi estadía en este proceso. En especial: Eduardo, Marco, Oscar, Jaiver, Enrique, Juan Andrés, Luis y Gina.
- A quienes de una u otra forma contribuyeron con su amistad y compañía durante mi estadía en la Universidad.
- A Valentina, por ser la luz de mis días.

RESUMEN

Se considera que el espectrómetro AMCMB96 es un instrumento fundamental para el trabajo de investigación en espectroscopía Mössbauer en la Universidad del Valle, y que dicho instrumento es difícilmente reemplazable en el corto a mediano plazo. En este proyecto se diseñó y programó un programa de manejo y driver de comunicación para el espectrómetro Mössbauer AMCMB96, obteniendo una aplicación funcional y compatible con los formatos de datos hasta ahora utilizados por los usuarios del instrumento en la Universidad. Se demostró el correcto funcionamiento del nuevo programa para los modos de análisis de altura de pulsos y analizador multicanal, usados cotidianamente en el AMCMB96. El programa desarrollado soporta su ejecución en cualquier plataforma $Windows$ posterior a $WindowsXP$, puede generar espectros gráficos en diversos formatos de imagen y fue probado exitosamente en la toma de datos de varias muestras de calibración típicamente usadas en el laboratorio de espectroscopía Mössbauer de la Universidad del Valle.

OBJETIVOS

Objetivo general

- Diseñar e implementar una interfaz de comunicación y control de tipo instrumento virtual, capaz de manejar y administrar todas las funcionalidades del espectrómetro mössbauer AMCMB96.

Objetivos específicos

- Reemplazar el actual programa de manejo del espectrómetro Mössbauer AMCMB96 con una nueva interfaz de control y comunicación, con soporte para todas las funcionalidades del programa original.
- Llevar a cabo la implementación del programa en el laboratorio de espectroscopía Mössbauer de la Universidad, de modo tal que en adelante los espectrómetros AMCMB96 disponibles sean gestionados con el mismo.

Índice general

Capítulo 1

Introducción

El grupo de investigación en Espectrocopía Mössbauer de la Universidad del Valle ha sido, durante muchos años, semillero de numerosos investigadores y egresados de la carrera de Física. En muchas ocasiones los análisis llevados a cabo por los estudiantes del grupo sobre muestras de estudio han tenido como principal instrumento de medición el espectrómetro Mössbauer AMCMB96, diseñado y puesto en marcha en 1996 como parte de un trabajo doctoral [1]. Se considera entonces que éste es un instrumento fundamental en el mencionado laboratorio, y que siendo un instrumento diseñado para las necesidades específicas de investigación y análisis de datos requeridas por el grupo, no existen equipos de similar funcionalidad en el mercado nacional, teniendo éstos que ser importados a un alto costo para la institución. Estos factores hacen al espectrómetro AMCMB96 un instrumento difícilmente reemplazable en el corto a mediano plazo. Por otra parte, la notable evolución tecnológica experimentada en los sistemas computacionales usados en laboratorio ha llevado a la necesidad de actualización de los métodos de manejo del AMCMB96, principalmente en lo referido a nuevos requerimientos de software no cubiertos por el programa en uso, que completa ya 17 años de funcionamiento continuo. Es absolutamente necesario solucionar las incompatibilidades de orden tecnológico y las dificultades de actualización de la interfaz de programa en uso que impiden la gestión del AMCMB96 desde sistemas computacionales modernos.

Como solución a las necesidades mencionadas, en el presente trabajo se expone el dise-

ño, programación e implementación de una interfaz de comunicación y control para el espectrómetro Mössbauer AMCMB96, sobre una plataforma de programación Visual C#, que cumple con los criterios de facilidad de uso, flexibilidad y conservación de funcionalidad del programa original diseñado para el AMCMB96. La nueva interfaz cuenta con soporte para los modos de operación usados normalmente en el AMCMB96 (análisis de altura de pulsos y analizador multicanal), soporte para ejecución desde cualquier plataforma $Windows$ posterior a $WindowsXP$, compatibilidad con los formatos de datos espectrales usados en el laboratorio, generación de espectros gráficos en los formatos de imagen más utilizados en la actualidad (JPG, PNG, EPS) y con un modo de depuración y comunicación en bruto para diagnóstico básico de la respuesta del aparato.

La estructura del documento es la siguiente: En el capítulo de Marco Teórico se brinda información acerca del funcionamiento y comunicación del AMCMB96. En el capítulo de Metodología se explican los modos de servicio del AMCMB96, junto con la descripción y detalle de los comandos que no dependen de un modo de servicio particular. Se explica también el funcionamiento esencial del programa original de manejo del instrumento, incluyendo los detalles correspondientes a los 2 modos de servicio y el procesamiento de los datos en el sistema DOS. En el capítulo siguiente se describe la interfaz del nuevo programa y los paradigmas bajo los cuales ha sido construida y diseñada. Posteriormente se describen y documentan de forma detallada las rutinas de funcionamiento del programa y la estructura del mismo, y por último se explica el modo de implementación y consideraciones adicionales, tales como el sistema de desarrollo colaborativo a futuro. El último capítulo es el correspondiente a las conclusiones del proyecto y se hacen también algunas sugerencias para aquellos interesados en continuar con este trabajo.

Capítulo 2

Marco Teórico

En este capítulo se dará a conocer al lector cuales son las tecnologías, protocolos de comunicación y dispositivos que utilizamos. Aquí se describe el flujo de funcionamiento del AMCMB96 y la forma en como son recogidos y procesados los datos desde la generación de los mismos, contador proporcional y la posterior escritura en memoria para el análisis de los espectros.

2.1. Protocolos de comunicación

El espectrómetro Mössbauer AMCMB96 se comunica con una PC mediante el protocolo de comunicación RS-232, con una modificación para el establecimiento de un puente entre los pines 1, 4 y 6 con el fin de mantener el puerto continuamente habilitado por hardware. En la declaración de rutinas del programa, además, debe especificarse un control de flujo RTS/CTS, toda vez que los puertos correspondientes deben encontrarse conectados de forma cruzada para la correcta comunicación del aparato. Esto significa que la comunicación entre el AMCMB96 y la PC deberá llevarse a cabo mediante un cable especialmente confeccionado para ello. Las tecnologías disponibles en el desarrollo del programa original obligaron al desarrollo de un módulo denominado COMROM, sin el cual era imposible manejar las interrupciones al chipset del sistema operativo y mediante el cual se implementaba un sistema de buffers que hacía posible la

transferencia de datos sin pérdida de caracteres. Este controlador ha quedado obsoleto y ha sido completamente reemplazado por las rutinas de comunicación existentes para los puertos seriales en el lenguaje $C\#$. En la tabla 2.1 se observa el listado de funciones del conector DB9, con la respectiva descripción de la convención de uso de cada uno de los pines.

Pin $DB-9$	Nombre	Función
1	DCD	DETECCIÓN DE PORTE DE DATOS (ENTRADA)
2	RXD	RECEPCIÓN DE DATOS (ENTRADA)
3	TXD	TRANSMISIÓN DE DATOS (SALIDA)
4	DTR	TERMINAL DE DATOS LISTO (SALIDA)
5	COMÚN	COMÚN COMÚN (REFERENCIA)
6	DSR	DISPOSITIVO DE DATOS LISTO (ENTRADA)
7	RTS	PETICIÓN DE ENVÍO (SALIDA)
8	CTS	DISPUESTO PARA ENVIAR (ENTRADA)
9	RI	INDICADOR DE LLAMADA (ENTRADA)

Cuadro 2.1: Asignación de pines de conectores $DB-9$

En la comunicación por puerto serial, la velocidad con la que los datos se transmiten es uno de los parámetros más importantes. Para el caso del $RS-232$ utilizado con el AMCMB96, se debe transmitir a una velocidad de 9600 Baudios ($1Baudio = 1bit/seg$).

La comunicación por puerto serial se utiliza frecuentemente para el establecimiento de comunicación asíncrona, es decir, sin un tiempo preestablecido de inicio. Los datos llegan en paquetes de información $8bits$ o $1byte$ (equivalente a un caracter ASCII), en el caso del AMCMB96, los comandos deben ser enviados carácter por carácter y finalizar con el caracter de fin de comando $< CR >$

2.2. Estructura de los comandos

Los comandos de control y manejo del AMCMB96 que son ejecutados por el usuario desde la PC tienen una estructura bien definida. Se componen de caracteres ASCII específicos que definen el tipo de servicio y subcomando a ejecutar, seguidos de valores hexadecimales de 1 o 2 bytes de longitud que deben ser declarados y enviados al instrumento en estricto orden para la correcta ejecución del comando. Así pues, la estructura general de un comando a ejecutar desde la PC mediante un programa de manejo del AMCMB96 debe ser la siguiente:

Índice del campo												
0	1	2	3	4	5	6	7	8-9	10-11	12-13	...	<CR>
[	longitud	servicio	tipo	byte	byte	byte	byte	2 bytes	2 bytes	2 bytes	...	FIN

El primer campo de 1 byte ("[") es el iniciador de comando, en el campo 1 está definida la longitud del comando, mientras que los campos 2 y 3 corresponden al tipo de comando y subservicio a ejecutar. Desde el campo 4 en adelante se pueden encontrar diferentes valores, los cuales dependen del tipo de comando y el subservicio. Estos valores (bien sean del campo 4 al 7, de un byte, o de ahí en adelante de 2 bytes) deben ser declarados y enviados en formato hexadecimal. Sin embargo, debe anotarse que el campo de longitud no es enviado al AMCMB96, y solamente tiene propósitos de control del tamaño de los comandos en las rutinas de construcción de las cadenas a enviar.

2.2.1. Matriz de Comandos

De acuerdo con lo indicado en 2.2 podemos resumir la matriz completa de los comandos que controlan la operación del AMCMB96.

Índice del campo												
0	1	2	3	4	5	6	7	8-9	10-11	12-13	...	...
[	2	?	Chequeo de presencia y ID del AMC									
[	2	0	I/G	Activar modo PHA/AAP								
[	2	1	I/G	Activar modo multicanal (AMC), muestreos cortos								
[	2	2	I/G	Activar modo multicanal (AMC), muestreos largos								
[	3	C	P/M	D/A	Control del retorno de interrupción							
[	2	L	P/M	Limpieza de los bancos de datos PHA/AMC								
[	2	E	P	Envío del banco de datos modo PHA/AAP								
[	10	E	M	0h	0h	0h	0h	00h	longitud		Envío del banco de datos modo AMC	
[	10	R	S	0h	0h	0h	0h	00h	2*canales+1		Recepción señal analógica	
[	2	V	E	Envío de los niveles de ventana								
[	4	V	R	LIh	LSh	Recibir niveles de ventana en hex						
[	9	P	yy	mm	dd-m	dd-s	hh	mm	ss	cc	<CR>	Poner fecha y hora en formato BCD
[	4	T	I/E	0/1	Leer fecha y hora							
[	22	D	0	E/I	0	0	0	00	00	00	00	Datos AMC

Tabla 1. Matriz de comandos del AMCMB96

Aunque en capítulos posteriores se explicará minuciosamente el funcionamiento de cada una de las órdenes descritas en 2.2.1 y su implementación en el programa, el detalle de la estructura de cada comando contenido en esta matriz puede ser consultado en [1].

Capítulo 3

Diseño y programación del nuevo programa de manejo - MOSSCOMM

3.1. Interfaz de programa

MOSSCOMM, el programa de manejo y driver de comunicación para el espectrómetro Mossbauer AMCMB96, es una aplicación nativa para sistemas *Windows*, compatible con *Windows XP/Vista*/7/8 y *Windows Server* 2003/2008/2012. Fue programada en la plataforma de desarrollo *Visual Studio* 2010, en el lenguaje de programación *Visual C#*.

De acuerdo con los requerimientos de funcionalidad del AMCMB96, se portaron a la mencionada plataforma todas las rutinas de comunicación e interacción del instrumento con el PC de manejo, manteniéndose por tanto inalterado el componente de hardware del espectrómetro. Las rutinas de software necesarias para el manejo del aparato de medición fueron, por tanto, reprogramadas en lenguaje *C* y adaptadas a los estándares de manejo de recursos de hardware administrados por plataformas modernas. Esto significó, de entrada, la eliminación del requerimiento de un controlador propio

($COMROM$) para la interacción de interrupciones a la CPU y al puerto serial del sistema operativo, toda vez que los controladores modernos de tales puertos poseen mecanismos apropiados para el control de interrupción. A continuación se describen los componentes principales de la aplicación de control del espectrómetro.

3.1.1. Interfaz gráfica

La interfaz gráfica fue diseñada en $Visual\ C$, y es la encargada de manejar la interacción con el usuario final. Se reciben en ésta las órdenes definidas por el usuario y con base en dichas órdenes se despliegan los componentes correspondientes al modo de operación elegido por el usuario, y se envían y/o reciben los datos por el puerto seriado en los casos apropiados. Las rutinas del programa asociadas a cada comando de operación del AMCMB96 toman los datos necesarios de entrada, los transforman en registros transmitibles y los envían por el puerto de comunicación. El código fuente completo de la interfaz se anexa al final de este documento y se divide en principalmente en dos componentes: la primera es la relacionada con la comunicación seriada, en la cual es posible establecer una serie de parámetros de operación necesarios para permitir la comunicación entre la PC y el AMCMB96. La segunda corresponde a la interfaz de control, en la cual se han establecido una serie de componentes gráficos para facilitar al usuario la operación de todas las funcionalidades del espectrómetro AMCMB96. Las figuras 3.1 y 3.2 muestran las características de ambas interfaces.

3.1.2. Modos de operación

El programa puede funcionar en 2 modos de operación: el primero es el modo de analizador de altura de pulsos (PHA/AAP), en el cual la entrada de datos consiste en pulsos provenientes del AMCMB96 con una resolución de 256 canales, correspondientes a una conversión análoga-digital de $8bits$. El segundo modo de operación es el modo de recepción de datos Mössbauer (AMC). En este modo se lee un banco de datos

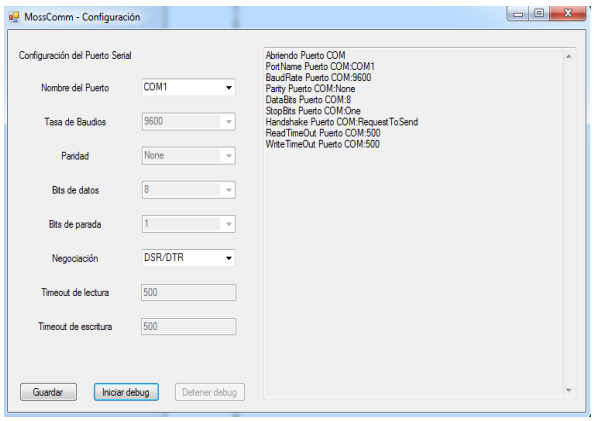

Figura 3.1: Interfaz de configuración de comunicación

cuya longitud depende del tiempo de muestreo elegido por el usuario al establecer los parámetros de medición en la toma de datos. Mientras en el modo PHA/AAP no se requiere de prácticamente ninguna intervención del usuario en cuanto a los parámetros de trabajo, en el modo AMC se requiere el establecimiento previo de parámetros de funcionamiento del instrumento, tales como la frecuencia, forma de onda y tiempo de muestreo.

3.1.2.1. Modo de altura de pulsos (AAP)

Para iniciar el programa en el modo de altura de pulsos (PHA/AAP) se debe acceder a la opción de menú especificada en la figura 3.3

El modo PHA/AAP es activado mediante la opción "$ActivarModo$" especificada en la interfaz de programa. Una vez activado el modo AAP en el programa, el AMCMB96 puede seguir trabajando de forma autónoma y la ejecución constante del programa es innecesaria. Cuando se ingresa en el modo PHA/AAP el programa inhabilita la posibilidad de ingresar al modo AMC, de forma tal que para ingresar al modo de operación AMC primero debe desactivarse el modo PHA/AAP en el menú. La rutina siguiente ilustra la activación del modo AAP y correspondiente desactivación del modo

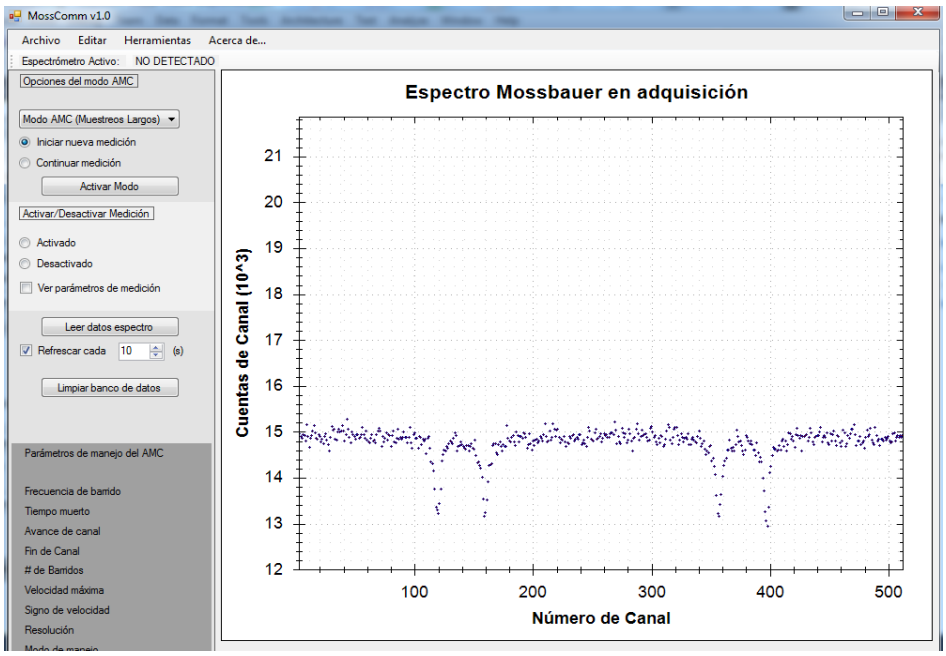

Figura 3.2: Interfaz gráfica de usuario MOSSCOMM

AMC a través de la accesibilidad de los menúes correspondientes.

```
private void MenuModoAAP_Click(object sender, EventArgs e)
      {
          MenuModoAAP.Checked = !MenuModoAAP.Checked;
          if (MenuModoAAP.Checked == true)
          {
              MenuModoAMC.Checked = false;
              MenuModoAMC.Enabled = false;
              panelModoAAP.Enabled = true;
              panelModoAAP.Location = new Point(0, 44);
              panelModoAAP.Visible = true;
              CreateGraphAAP(graphMain);
              ActualiceGraficoAAP();
              graphMain.Show();
          }
          if (MenuModoAAP.Checked == false)
          {
              MenuModoAMC.Enabled = true;
```

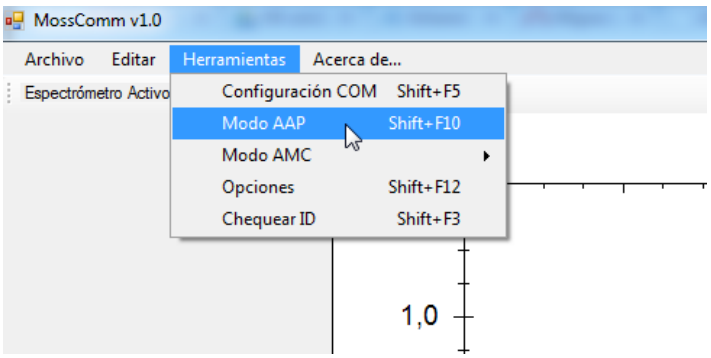

Figura 3.3: Acceso al modo PHA/AAP.

```
            panelModoAAP.Enabled = false;
            panelModoAAP.Visible = false;
            buttonActivarAAP.Enabled = true;
            graphMain.Hide();
        }
    }
```

3.1.2.2. Modo de analizador multicanal (AMC)

Para iniciar el programa en el modo de altura de pulsos (*AMC*) se debe acceder a la opción de menú especificada en la figura 3.4

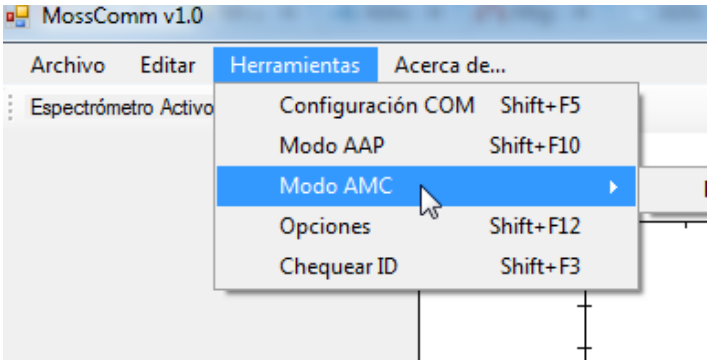

Figura 3.4: Acceso al modo AMC.

Si bien el modo AMC puede ser activado de forma idéntica al modo PHA/AAP (mediante la opción "$ActivarModo$") especificada en la interfaz de programa, para la toma de datos Mössbauer se requiere una serie de parámetros de trabajo que deben ser establecidos previamente por el usuario. Estos parámetros pueden ser configurados en el submenú *Parámetros de Medición* disponible bajo el menú $ModoAMC$. De no definirse parámetro alguno por parte del usuario, el modo AMC se inicia con parámetros de trabajo de $10Hz$, 256 canales de resolución y forma de onda triangular. Una vez activado el modo AMC en el programa, el AMCMB96 puede seguir trabajando de forma autónoma y la ejecución constante del programa es innecesaria. Cuando se ingresa en el modo AMC el programa inhabilita la posibilidad de ingresar al modo PHA/AAP, de forma tal que para ingresar al modo de operación PHA/AAP primero debe desactivarse el modo AMC en el menú. La rutina siguiente ilustra la activación del modo AMC y correspondiente desactivación del modo AAP a través de la accesibilidad de los menúes correspondientes. Se puede observar que en este caso, se establecen los parámetros de trabajo por defecto desde el mismo momento en que se accede al modo AMC por ingreso al menú.

```
private void MenuModoAMC_Click(object sender, EventArgs e)
          /*Rutina que inicia el despliegue gráfico del modo AMC.
           * Cuando se pone check en este ítem, se deshabilitan las opciones de modo AAP
           * y se hacen visibles los paneles y gráficos del modo AMC.
           * El modo gráfico más importante es el correspondiente a ZedGraph,
           * invocado en CreateGraphAMC() y ActualiceGraficoAMC()
           */
      {
          MenuModoAMC.Checked = !MenuModoAMC.Checked;
          if (MenuModoAMC.Checked == true)
          {
              MenuModoAAP.Checked = false;
              MenuModoAAP.Enabled = false;
              panelModoAMC.Enabled = true;
              panelModoAMC.Visible = true;
              CreateGraphAMC(zedGraphControlMainAMC);
```

```
            ActualiceGraficoAMC();
            //zedGraphControlMainAMC.Show();
            buttonActivarAMC.Enabled = true;
            buttonActivarAMC.Show();
            Properties.Settings.Default.ModoManejo = 1;
            Properties.Settings.Default.FrecuenciaBarrido = 10;
            Properties.Settings.Default.TiempoMuerto = 20;
            Properties.Settings.Default.Resolucion = 256;
            Properties.Settings.Default.MaxVelocidad = 8;
            Properties.Settings.Default.FactorEscala = 1;
            Properties.Settings.Default.SignoVelocidad = 0;
            Properties.Settings.Default.ModoAvance = Convert.ToChar("I");
            MessageBox.Show("ATENCIÓN:"
            + Environment.NewLine
            + "Si no modifica los parámetros de trabajo, se usarán los siguientes:"
            + Environment.NewLine
            + Environment.NewLine + "Modo de manejo:" +
                Properties.Settings.Default.ModoManejo.ToString()
            + Environment.NewLine + "Frecuencia de barrido:" +
                Properties.Settings.Default.FrecuenciaBarrido.ToString() + " Hz"
            + Environment.NewLine + "Resolución:" +
                Properties.Settings.Default.Resolucion.ToString() + " Canales"
            + Environment.NewLine + "Factor de escala:" +
                Properties.Settings.Default.FactorEscala.ToString()
            + Environment.NewLine + "Signo de velocidad:" +
                Properties.Settings.Default.SignoVelocidad.ToString()
            + Environment.NewLine + "Modo de avance:" +
                Properties.Settings.Default.ModoAvance.ToString()
            );
            zedGraphControlMainAMC.Show();
        }
        if (MenuModoAMC.Checked == false)
        {
            MenuModoAAP.Enabled = true;
            panelModoAMC.Enabled = false;
            panelModoAMC.Visible = false;
            //buttonActivarAMC.Enabled = true;
            zedGraphControlMainAMC.Hide();
        }
    }
```

3.2. Rutinas principales de manejo del AMCMB96

A continuación se detallan las rutinas principales de funcionamiento y manejo del AMCMB96. En particular, se hace énfasis en el proceso necesario para extraer de forma correcta los datos requeridos en una medición y en el envío de los parámetros de comando explicados en los primeros apartes de este documento.

3.2.1. Solicitud y Recepción del banco de datos (AAP)

El banco de datos de altura de pulsos es generado por un conversor de $8bits$ $ADC0804$, teniendo por tanto la posibilidad de obtener 256 diferentes datos de altura de pulsos. Se requieren, por tanto, 256 canales para guardar dicha información. En la CPU del $AMCMB96$ se determina en 3 bytes consecutivos la cantidad de memoria utilizada por cada canal, lo cual indica que el tamaño del banco de datos AAP es de 768 bytes. El comando para solicitar el banco de datos AAP tiene como código de uso EP y se detalla en la rutina $instanciarPuertoSerial$:

```
public void instanciarPuertoSerial()
    /*
     * Si el menú de un modo está activo, se envía la orden para recibir el espectro
     * del respectivo modo. Los modos son excluyentes así que la rutina es segura.
     * Se envía la orden, se espera el buffer y con el evento de recepción de datos
     * se invocan los Handlers para procesar los buffers y escribirlos en los respectivos
     * archivos.
     * Por el motivo anterior se declaran aquí los nombres de los archivos.
     */
{
    string archivoAMC = "currentAMC_Dev" + amcID.ToString() + ".data";
    string archivoAAP = "currentAAP_Dev" + amcID.ToString() + ".data";
    ...
    ...
    ...

    if (MenuModoAAP.Checked == true)
```

```
        {
            string bufaap = "EP";
            EnviarOrdenEsperarRespuesta(bufaap);
            serialPortMain.DataReceived += new
                SerialDataReceivedEventHandler(serialPortMain_BuffAAPReceived);
            //envia [EP (envío único del modo PHA/AAP)
            //El handler serialPortMain_BuffAAPReceived maneja el retorno de los datos PHA/AAP.
            //serialPortMain.Close();

        }
    }
```

Una vez enviada la solicitud de envío del banco de datos de altura de pulsos, la rutina se suscribe al evento $serialPortMain_BuffAAPReceived$, que es activado en cuanto se recibe un dato por el puerto serial después del envío de la solicitud descrita. La rutina correspondiente al evento $serialPortMain_BuffAAPReceived$ se muestra a continuación.

```
public void serialPortMain_BuffAAPReceived(object sender, SerialDataReceivedEventArgs e)
    {
        System.Threading.Thread.Sleep(6000);
        int numdatosBuferEntrada;
        numdatosBuferEntrada = serialPortMain.BytesToRead;
        Byte[] datoBuffAAP = new Byte[numdatosBuferEntrada];
        long[] datoPulsosBufAAP = new long[256];
        serialPortMain.Read(datoBuffAAP, 0, numdatosBuferEntrada);
        ...
        Array.Clear(datoPulsosBufAAP, 0, datoPulsosBufAAP.Length);
        for (i = 1, j = 0; i < (256 * 3); j++, i += 3)
        {
            long dato1n = Int64.Parse(datoBuffAAP[i].ToString(), NumberStyles.HexNumber);
            long dato2n = Int64.Parse(datoBuffAAP[i + 1].ToString(), NumberStyles.HexNumber);
            long dato3n = Int64.Parse(datoBuffAAP[i + 2].ToString(), NumberStyles.HexNumber);
            datoPulsosBufAAP[j] = dato1n + 256 * dato2n + 65536 * dato3n;
            dato1n = 0;
            dato2n = 0;
            dato3n = 0;
        }
```

```
        string archivoAAPdraw = "currentAAP_Dev" + amcID.ToString() + ".data";
        StreamWriter espeaap = new System.IO.StreamWriter(archivoAAPdraw);
        for (i = 1; i < datoPulsosBufAAP.Length; i++)
        {
            espeaap.WriteLine(datoPulsosBufAAP[i].ToString());
        }
        espeaap.Close();
        serialPortMain.DataReceived -= new
             SerialDataReceivedEventHandler(serialPortMain_BuffAAPReceived);
        ...
    }
```

Como se puede observar, se recibe un buffer de 768 bytes y se genera un vector entero de 256 componentes, todas puestas a cero. En el bucle de generación del espectro de pulsos se toman grupos de 3 bytes consecutivos y se realiza la correspondiente operación de conversión de los valores desde hexadecimal hacia números enteros, además de la conversión del número de 24 bits a un número entero correspondiente al número de cuentas del canal medido. Esto es:

$$C_i = n_{3i} + 256(n_{3i+1}) + 65536(n_{3i+2}), i = 0, 1, 2... \tag{3.1}$$

De este modo, el espectro de altura de pulsos AAP es recibido y procesado por el programa, que procede a escribirlo en un archivo temporal, que luego será procesado por la rutina de generación de archivos de espectro, la cual será descrita posteriormente.

3.2.2. Solicitud y Recepción del banco de datos (AMC)

La solicitud del banco de datos de espectros Mössbauer (AMC) es ligeramente diferente a la del espectro AAP. En este caso, el número de canales (siempre y cuando se trabaje con una señal triangular o senoidal) esta dado por el doble del parámetro de resolución establecido para el modo de trabajo del instrumento.

$$C = 2r \tag{3.2}$$

En el caso de una señal tipo diente de sierra, el número de canales se ve disminuido por el intervalo de tiempo muerto de la señal, de modo tal que la proporción entre dicho tiempo muerto y el ciclo completo de la señal determina el número total de canales ocupados en la medición, en la siguiente forma

$$C = (1 + p/100)r \tag{3.3}$$

En donde p corresponde al porcentaje de tiempo muerto de la señal diente de sierra y r es la resolución definida. En cualquiera de los casos, la resolución deberá corresponder a una potencia de 2 $(2, 4, ..,256, 512, 1024)$. Los parámetros de trabajo más comunes en el ámbito de desarrollo del programa corresponden a una resolución de 256 (512 canales) y forma de onda triangular, lo cual significa que con datos de 24 bits (3 bytes por canal) tendremos un tamaño de buffer dado por

$$bufamc = 3 * C \tag{3.4}$$

Lo cual para el banco de datos significa una longitud total de 1536 bytes a recibir. El comando para solicitar el banco de datos AAP tiene como código de uso EM y se detalla en la rutina $instanciarPuertoSerial$:

```
public void instanciarPuertoSerial()
{
    string archivoAMC = "currentAMC_Dev" + amcID.ToString() + ".data";
    string archivoAAP = "currentAAP_Dev" + amcID.ToString() + ".data";

    if (MenuModoAMC.Checked == true)
    {
        InicializarPuerto(serialPortMain);
        byte[] control_interrumpir = new byte[3] { 0x43, 0x4D, 0x41 };
        string inicomando = "[";
        serialPortMain.Write(inicomando);
        for (int i = 0; i < control_interrumpir.Length; i++)
        {
            byte[] chain = new byte[1] { control_interrumpir[i] };
```

```
            serialPortMain.Write(chain, 0, chain.Length);
            System.Threading.Thread.Sleep(50);
        }
        string fincr = "\r";
        serialPortMain.Write(fincr);
        byte[] bufamc = new byte[10] { 0x45, 0x4D, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
             0x06 };
        serialPortMain.Write(inicomando);
        for (int i = 0; i < bufamc.Length; i++)
        {
            byte[] chain = new byte[1] { bufamc[i] };
            serialPortMain.Write(chain, 0, chain.Length);
            System.Threading.Thread.Sleep(100);
        }
        serialPortMain.Write(fincr);
        //Envía [EM000001537 (1536/3 = 512 = N, y 3N + 1 = 1537)
        //Envía ( 0x45 (E), 0x4D (M), 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 (4 campos de 1 byte,
             uno de 2 bytes), 0x0601 (Longitud)
        //El handler serialPortMain_BuffAMCReceived maneja el retorno de los datos del espectro.
       serialPortMain.DataReceived += new
            SerialDataReceivedEventHandler(serialPortMain_BuffAMCReceived);
    }
    ...
    }
}
```

Es importante mencionar que previamente al envío de la solicitud del envío del banco de datos AMC debe enviarse un comando de activación del control de interrupción. Esto se debe a que el servicio del modo AMC debe tener claramente especificado si al término de una interrupción para el envío de datos se debe continuar con la medición o se debe detener la misma. De no especificar el estado del control de interrupción antes de la solicitud de envío, la medición se detendrá. Una vez enviada la solicitud de envío del banco de datos AMC, la rutina se suscribe al evento $serialPortMain_BuffAMCReceived$, que es activado en cuanto se recibe un dato por el puerto serial después del envío de la solicitud descrita. La rutina correspondiente al evento $serialPortMain_BuffAMCReceived$ se muestra a continuación.

```
void serialPortMain_BuffAMCReceived(object sender, SerialDataReceivedEventArgs e)

      {
          System.Threading.Thread.Sleep(2000);
          int numdatosEntradabtr = serialPortMain.BytesToRead;
          Byte[] datoBuffAMCReal = new Byte[numdatosEntradabtr];
          serialPortMain.Read(datoBuffAMCReal, 0, numdatosEntradabtr);
          string respuestaBuffAMCReal = System.Text.Encoding.UTF8.GetString(datoBuffAMCReal);
          Byte[] datoBuffAMC = new Byte[1540];
          long[] datoEspecAMC = new long[512];
          int i = 0;
          int j = 0;
          Array.Clear(datoEspecAMC, 0, datoEspecAMC.Length);
          for (i = 2, j = 0; i < (512 * 3); j++, i += 3)
          {
              int dato1n = Convert.ToInt32(datoBuffAMCReal[i]);
              int dato2n = Convert.ToInt32(datoBuffAMCReal[i+1]);
              int dato3n = Convert.ToInt32(datoBuffAMCReal[i+2]);
              datoEspecAMC[j] = dato1n + (256 * dato2n) + (65536 * dato3n);
              dato1n = 0;
              dato2n = 0;
              dato3n = 0;
          }
          string archivoAMCdraw = "currentAMC_Dev" + amcID.ToString() + ".data";
          StreamWriter espeamc = new System.IO.StreamWriter(archivoAMCdraw);
          long numbarridos = datoEspecAMC[datoEspecAMC.Length - 1];
          for (i = 1; i < datoEspecAMC.Length - 1; i++)
          {
              espeamc.WriteLine(datoEspecAMC[i].ToString());
          }
          espeamc.WriteLine(datoEspecAMC[510].ToString());
          espeamc.Close();
          serialPortMain.DataReceived -= new
               SerialDataReceivedEventHandler(serialPortMain_BuffAMCReceived);
          CreateGraphAMC(zedGraphControlMainAMC);
          ActualiceGraficoAMC();
      }
```

Como se puede observar, se recibe un buffer de 1536 bytes y se genera un vector

entero de 512 componentes, todas puestas a cero. En el bucle de generación del espectro de pulsos se toman grupos de 3 bytes consecutivos y se realiza la correspondiente operación de conversión de los valores desde hexadecimal hacia números enteros, además de la conversión del número de 24 bits a un número entero correspondiente al número de cuentas del canal medido, con un método similar al usado para el espectro AAP. De este modo, el espectro de altura de pulsos AMC es recibido y procesado por el programa, que procede a escribirlo en un archivo temporal, que luego será procesado por la rutina de generación de archivos de espectro, la cual será descrita posteriormente.

3.2.3. Solicitud de identidad

La primera acción que debe llevarse a cabo para la utilización del programa consiste en el chequeo de identidad. El chequeo de identidad permite al programa conocer el número de identificación del instrumento particular utilizado en la medición, de forma tal que al usarse varios instrumentos de forma simultánea no existan problemas de sobreescritura de archivos temporales y/o datos de medición de ninguno de los instrumentos por parte del programa. Al iniciar $MOSSCOMM$, se encontrarán deshabilitadas las entradas de menú correspondientes al Modo AAP y Modo AMC. Para activarlas, deberá realizarse el chequeo de identidad como es indicado en la figura 3.5

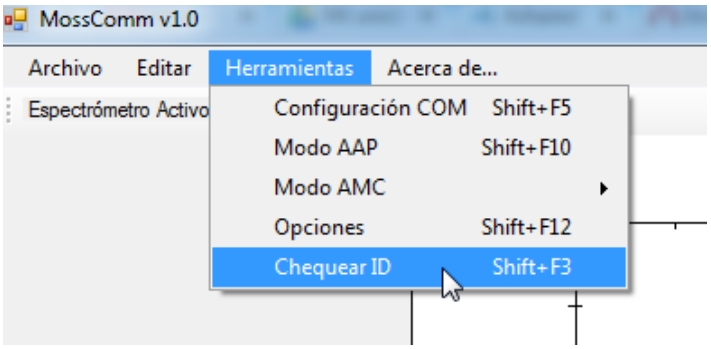

Figura 3.5: Acceso al chequeo de solicitud de identidad.

De ejecutarse adecuadamente el chequeo de identidad, el número del espectrómetro activo aparecerá en pantalla y las entradas de menú de los modos principales de operación serán accesibles.

3.2.4. Parámetros del modo multicanal (AMC)

El modo de trabajo *AMC* requiere el establecimiento de una serie de parámetros previos al inicio de la medición, los cuales deben ser especificados mediante comando al instrumento. Para tal propósito se programó la rutina *EnviarDatosOperacionAMC*, la cual determina y envía los parámetros de operación necesarios para las mediciones en el modo de trabajo *AMC*. A continuación se describen los parámetros necesarios para la construcción del comando necesario para la activación del modo de trabajo *AMC*.

- *Avance*: Este parámetro define si el modo de avance de canal del AMC es interno o externo. Los modelos de AMCMB96 disponibles en el entorno de trabajo solamente disponen de modo interno. Los valores posibles son *E* (externo), *I* (interno).

- *Longitud*: Este parámetro define la longitud necesaria en bytes para el almacenamiento del banco de datos que se obtenga en la medición.

- *Canales*: Este parámetro está definido de acuerdo con la resolución escogida por el usuario para la medición.

- *Barridos*; Este parámetro define el número de barridos a realizar en la medición. Por defecto se pone a cero, lo que significa infinitos barridos.

- *Tiempo de muestreo*; Este parámetro define el tiempo de muestreo por canal (de acuerdo con la constante de carga del instrumento). Está definida a cero.

- *Número de tiempo de muestreo*: Este parámetro se extrae a partir del cálculo de la señal, en la subrutina *CalculaTiempos*. Puede ser 0 o 1.

Estos parámetros se utilizan como argumentos en las rutinas de envío y cálculo de la señal, activación del servicio del modo AMC y en el cálculo de la forma de onda enviada en el banco de señal analógica. La rutina encargada del control y establecimiento, como se mencionó anteriormente, es *EnviarDatosOperacionAMC*, la cual se describe a continuación:

```
public void EnviarDatosOperacionAMC(Variables_Manejo_AMC manejo)
      {
          manejo.mb_barridos = 0;
          byte[] avance = BitConverter.GetBytes(manejo.avance);
          byte[] longi = BitConverter.GetBytes(Convert.ToInt16(manejo.longitud));
          byte[] canales = BitConverter.GetBytes(Convert.ToInt16(manejo.canales));
          byte[] barridos = BitConverter.GetBytes(Convert.ToInt16(manejo.mb_barridos));
          byte[] tiempo_muestreo = BitConverter.GetBytes(Convert.ToInt16(manejo.tiempo_muestreo));
          byte[] numero_ti_muestreo =
               BitConverter.GetBytes(Convert.ToInt16(manejo.numero_ti_muestreo));
          byte[] datos_oper_byte = new byte[18] { 0x44, 0x00, avance[0], 0x00, 0x00, 0x00, 0x00,
               0x00, longi[0], longi[1], canales[0], canales[1], barridos[0], barridos[1],
               tiempo_muestreo[0], tiempo_muestreo[1], numero_ti_muestreo[0], numero_ti_muestreo[1] };
          InicializarPuerto(serialPortMain);
          string inicomando = "[";
          serialPortMain.Write(inicomando);
          for (int i = 0; i < datos_oper_byte.Length; i++)
          {
              byte[] chain = new byte[1] { datos_oper_byte[i] };
              serialPortMain.Write(chain, 0, chain.Length);
              System.Threading.Thread.Sleep(300);
          }
          string fincr = "\r";
          serialPortMain.Write(fincr);
          serialPortMain.DataReceived += new
               SerialDataReceivedEventHandler(serialPortMain_ConfirmParamsReceived);
      }
```

Cuando se envían los parámetros de trabajo al AMCMB96, se espera confirmación de recepción exitosa por parte del instrumento con el caracter !. La suscripción al evento $serialPortMain_{C}onfirmParamsReceived$ procesa la recepción de dicho caracter y retorna a la ejecución de la rutina con código de transmisión exitosa.

3.2.5. Banco de señal analógica y generación de la forma de onda

En el caso de trabajar en el modo AMC, antes de activar dicho modo se deberán tener en cuenta los correspondientes parámetros de trabajo, sin los cuales el instrumento no puede llevar a cabo la activación del servicio. Entre estos parámetros, el más complejo e importante corresponde a la forma de onda y la respectiva generación de la señal. Antes de iniciarse el modo AMC, el instrumento debe preparar la recepción de un banco de señal analógica concordante con la forma de la señal escogida. Este banco será utilizado por el AMCMB96 para la generación de la señal que dará al transductor de velocidad del instrumento su movimiento específico, y por tanto es de vital importancia en la medición. La preparación de la recepción del banco de señal analógica tiene como cadena de instrucción RS, y como está explicado en la matriz de comandos, se acompaña de diversos parámetros para definir la longitud de dicha señal. Así pues, el correcto inicio del modo AMC está ligado a la generación y el envío de la señal analógica al instrumento. La rutina de cálculo de la señal analógica se muestra a continuación:

```
...
 {
                case 1:
                    k = maxresol_1 / prms_amc.resolucion;
                    j = (maxresol_1 - (k * prms_amc.resolucion)) / 2;
                    vrmanejo.canales = 2 * prms_amc.resolucion;
                    vrmanejo.longitud = 3 * vrmanejo.canales;
                    senal = new int[1024];
```

```
    for (i = 0; i < prms_amc.resolucion; i++)
    {
        senal[i] = j;
        j += k; //256 => 0,4,8,12... 512 => 0,2,4,6,8... 1024 => 0,1,2,3,4....
    }
    fact_vel = Properties.Settings.Default.FactorEscala;
    switch (fact_vel)
    {
        case 1:
            j = 0;
            break;
        case 2:
            j = 256;
            break;
        case 4:
            j = 384;
            break;
        default:
            j = 0;
            break;
    }

    for (i = 0; i < prms_amc.resolucion; i++)
    {
        int xquot = senal[i] / fact_vel;
        senal[i] = xquot + j;
        senal[vrmanejo.canales - 1 - i] = senal[i];
    }
    z = frecu / (vrmanejo.canales * prms_amc.frec_barrido);
    vrmanejo.tiempo_muestreo = CalculaTiempos(z, vrmanejo, out n);
    vrmanejo.numero_ti_muestreo = n;
    if (vrmanejo.numero_ti_muestreo == 1) vrmanejo.modo = 1; //buffer corto
    else vrmanejo.modo = 2; //buffer largo
    break;
/////////CASO DIENTE DE SIERRA/////////
case 2:
    k = maxresol_1 / prms_amc.resolucion; //esto es el quotient
    j = (maxresol_1 - (k * prms_amc.resolucion)) / 2; //esto es el quotient j (siempre
        es 0)
    vrmanejo.canales = (int)(prms_amc.resolucion*(1 + prms_amc.tiempo_muerto/100));
    i1 = vrmanejo.canales - prms_amc.resolucion;
```

```
vrmanejo.longitud = 3 * vrmanejo.canales;
senal = new int[1024];
for (i = 0; i < prms_amc.resolucion; i++)
{
    senal[i] = j+(i*k);
}
k = (prms_amc.resolucion - 1) * (k / (i1 + 1));
for (i = 0; i < i1; i++)
{
    senal[vrmanejo.canales - 1 - i] = j + (k * i);
}
if (prms_amc.signo_velocidad == -1)
{
    for (i = 0; i < vrmanejo.canales; i++)
    {
        senal[i] = maxresol_1 - 1 - senal[i];
    }
}
switch (fact_vel)
{
    case 1:
        j = 0;
        break;
    case 2:
        j = 256;
        break;
    case 4:
        j = 384;
        break;
    default:
        j = 0;
        break;
}
for (i = 0; i < vrmanejo.canales; i++)
{
    int xquot = senal[i] / fact_vel;
    senal[i] = xquot + j;
}
z = frecu / (vrmanejo.canales * prms_amc.frec_barrido);
vrmanejo.tiempo_muestreo = CalculaTiempos(z, vrmanejo, out n);
vrmanejo.numero_ti_muestreo = n;
```

```
            if (vrmanejo.numero_ti_muestreo == 1) vrmanejo.modo = 1; //buffer corto
            else vrmanejo.modo = 2; //buffer largo
            break;
            ///////////////CASO SINUSOIDAL///////////////
        case 3:
            z1 = pi / prms_amc.resolucion;
            vrmanejo.canales = 2 * prms_amc.resolucion;
            i1 = vrmanejo.canales / 4;
            vrmanejo.longitud = 3 * vrmanejo.canales;
            for (i = 0; i < prms_amc.resolucion; i++)
            {
                senaldouble[i] = 511*(1.0 - Math.Cos(z1*(i+0.5)));
            }
            switch (fact_vel)
            {
                case 1:
                    j = 0;
                    break;
                case 2:
                    j = 256;
                    break;
                case 4:
                    j = 384;
                    break;
                default:
                    j = 0;
                    break;
            }
            for (i = 0; i < prms_amc.resolucion; i++)
            {
                double xquot = (double)senaldouble[i] / fact_vel;
                senaldouble[i] = xquot + j;
                senaldouble[vrmanejo.canales - 1 - i] = senaldouble[i];
            }
            z = frecu / (vrmanejo.canales * prms_amc.frec_barrido);
            vrmanejo.tiempo_muestreo = CalculaTiempos(z, vrmanejo, out n);
            vrmanejo.numero_ti_muestreo = n;
            if (vrmanejo.numero_ti_muestreo == 1) vrmanejo.modo = 1; //buffer corto
            else vrmanejo.modo = 2; //buffer largo
            break;
    }
```

...

Una vez la señal es calculada, debe procederse al envío de la misma. En este caso, se envía el comando para preparar al AMCMB96 para la recepción del banco de señal analógica, y se espera confirmación de recepción exitosa por parte del instrumento para proceder con el envío de la señal en sí misma. La rutina de preparación es la siguiente:

```
public void instanciarRecepcionBDatos(Variables_Manejo_AMC manejo)
      {
            int longitud = 2*manejo.canales + 1;
            string longi = longitud.ToString();
            byte[] datos_senal_byte = new byte[10] { 0x52, 0x53, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
                 0x01, 0x04};
            InicializarPuerto(serialPortMain);
            string inicomando = "[";
            serialPortMain.Write(inicomando);
            for (int i = 0; i < datos_senal_byte.Length; i++)
            {
                byte[] chain = new byte[1] { datos_senal_byte[i] };
                serialPortMain.Write(chain, 0, chain.Length);
     }
            string fincr = "\r";
            serialPortMain.Write(fincr);
            serialPortMain.DataReceived += new
                 SerialDataReceivedEventHandler(serialPortMain_ConfirmBDReceived);
      }
```

Como se puede observar, la rutina se suscribe al evento $serialPortMain_{C}onfirmBDReceived$ en espera de la confirmación del instrumento para la recepción del banco de señal. Este evento invoca la rutina *EnviarSenal*, la cual se detalla a continuación:

```
public void EnviarSenal(Variables_Manejo_AMC manejo, int[] senal)
      {
          int i, j;
          int xquot, xrem;
          int[] senalalta = new int[1024];
          int[] senalbaja = new int[1024];
          int chks = 0;
          string alta, baja;
```

```
        byte[] altabyte, bajabyte;
        for (i = 0; i < manejo.canales; i++)
        {
            xquot = Math.DivRem(senal[i], 4, out xrem);
            senalalta[i] = xquot;
            chks += senalalta[i];
            senalbaja[i] = xrem;
            chks += senalbaja[i];
        }
        chks += chks;
        chks = (~chks) + 1;
        instanciarRecepcionBDatos(manejo);
        System.Threading.Thread.Sleep(2000);
        for (i = 0; i < manejo.canales; i++)
        {
            alta = senalalta[i].ToString();
            baja = senalbaja[i].ToString();
            altabyte = new byte[] { Convert.ToByte(senalalta[i]) };
            bajabyte = new byte[] { Convert.ToByte(senalbaja[i]) };

            serialPortMain.Write(altabyte, 0, 1);
            serialPortMain.Write(bajabyte, 0 ,1);
        }
        byte[] chkszero = new byte[1] { 0x00 };
        serialPortMain.Write(chkszero,0,1);
    }
```

Una vez enviado el banco de señal, se puede proceder con el comando de inicio del modo AMC. Es importante resaltar que la rutina de envío de la señal analógica no espera confirmación alguna.

3.2.6. Control del retorno de interrupción

Las subrutinas de espera de interrupciones del AMCMB96 siempre tienen su retorno de interrupción especificado de tal forma que cualquiera de los modos de servicio del instrumento deben inscribir un código de control en una de las variables internas del AMCMB96. De esta forma, el espectrómetro controla el retorno a un modo de

trabajo específico después de una interrupción. En particular, el control del retorno de interrupción es utilizado para indicar al instrumento si una medición debe continuar o debe ser detenida. La cadena de instrucción a enviar para el modo AAP es CPD para detener la medición y CPA para activarla. En el caso del modo Mössbauer, las cadenas de instrucción para activar y desactivar la medición son respectivamente CMA y CMD. Debe anotarse que el valor de la variable de control de retorno de interrupción debe estar definida obligatoriamente, toda vez que la lectura misma de un banco de datos constituye una interrupción de la medición y si este control no está bien definido, la medición bien se detendrá después de una primera lectura o bien llevará al instrumento a un estado de $HALT$.

3.2.7. Limpieza de los bancos de datos

El AMCMB96 posee ubicaciones de memoria específicas para los bancos de datos tanto del modo AAP como del modo AMC. Al iniciar una nueva medición o reiniciar una medición existente, el contenido de estos bancos de datos debe ser puesto a cero. En el caso del banco de datos del modo de altura de pulsos, la cadena de instrucción es LP, mientras que en el caso del banco de datos AMC la cadena de instrucción a enviar es LM. Para este comando no se espera respuesta alguna, toda vez que el AMCMB96 limpia sus bancos de memoria de forma autónoma al recibir la orden. Las rutinas encargadas de la limpieza de bancos de datos se detallan a continuación.

```
private void buttonLimpiarBancoAMC_Click(object sender, EventArgs e)
      {
          if (MessageBox.Show("La limpieza del banco de datos eliminará cualquier dato no guardado"
              + Environment.NewLine + "Está seguro de ejecutar la limpieza del banco de datos AMC?",
              "Precaución - Posible pérdida de Información", MessageBoxButtons.OKCancel) ==
                  DialogResult.OK)
          {
              string limbufamc = "LM";
              EnviarOrden(limbufamc);
```

```
            return;
        }
        else return;
    }
private void buttonLimpiarBancoAAP_Click(object sender, EventArgs e)
    {
        if (MessageBox.Show("La limpieza del banco de datos eliminará cualquier dato no guardado"
            + Environment.NewLine + "Está seguro de ejecutar la limpieza del banco de datos AAP?",
            "Precaución - Posible pérdida de Información", MessageBoxButtons.OKCancel) ==
                DialogResult.OK)
        {
            string limbufaap = "LP";
            EnviarOrden(limbufaap);
            return;
        }
        else return;
    }
```

Como se puede observar, la solicitud de limpieza de un banco de datos cualquiera origina la aparición de un diálogo de confirmación advirtiendo de la posible pérdida de información en caso de que los datos de la última medición no se hubiesen guardado.

3.2.8. Generación y formato de datos

Durante la programación de $MOSSCOMM$ se determinó la conveniencia de mantener un formato de datos compatible con los archivos generados por el programa original de manejo del $AMCMB96$, de forma tal que los traumatismos en el procesamiento de los datos obtenidos con el nuevo programa fuesen mínimos o nulos. A continuación se detalla el formato de dichos archivos y el método de generación de los mismos.

3.2.8.1. Archivos .DAT

Los archivos $.DAT$ son los archivos generados durante la ejecución de una medición de espectro en modo AMC. Dichos archivos poseen las siguientes características básicas.

- Son archivos de texto simple.

- Su longitud es de exactamente 512 líneas, cada una correspondiente a un canal.
- El mayor número posible de cuentas por canal es de 2^24, esto es un total de 16777216 cuentas.
- Los valores del último y penúltimo canal son idénticos, dado que en el último canal de la medición se almacena un contador cuya utilidad en el espectro es nula.

El archivo de datos en formato $.DAT$ es generado siempre a solicitud del usuario, y es una copia bit a bit del último archivo temporal de la medición. Esto significa que los datos de medición guardados en un archivo $.DAT$ generado por el usuario están tan actualizados como la última solicitud de envío del banco de datos del AMCMB96 por parte del programa. Para guardar un espectro Mössbauer en formato $.DAT$, el usuario, estando en el modo de trabajo AMC, deberá ingresar a la opción "$Archivo => GuardarComo...$", la cual desplegará el tradicional árbol de archivos de $Windows$ para guardar los datos en la ubicación deseada por el usuario. La rutina de guardado de los archivos, como fue mencionado, realiza una copia bit a bit del último archivo temporal generado por la solicitud de envío del banco de datos del modo de servicio en ejecución, y se detalla a continuación.

```
private void guardarComoToolStripMenuItem_Click(object sender, EventArgs e)
      {
          string datafile;
          SaveFileDialog guardarEspecDialog = new SaveFileDialog();
          ...
          {
              if (MenuModoAMC.Checked == true && MenuModoAAP.Checked == false)
              {
                  guardarEspecDialog.Filter = "Archivo de espectro Mössbauer|*.dat|Todos los archivos
                       |*.txt";
                  guardarEspecDialog.Title = "Guardar archivo de espectro Mössbauer";
                  datafile = @"currentAMC_Dev" + amcID.ToString() + ".dataplot";
```

```
            }
            ...
        }
        guardarEspecDialog.ShowDialog();
        if (guardarEspecDialog.FileName != "")
        {
            using (FileStream fsout = (FileStream)guardarEspecDialog.OpenFile(), input =
                 File.OpenRead(datafile))
            {
            switch (guardarEspecDialog.FilterIndex)
            {
                case 1:
                    int read = -1;
                    byte[] buffer = new byte[4096];
                    while (read != 0)
                    {
                        read = input.Read(buffer, 0, buffer.Length);
                        fsout.Write(buffer, 0, read);
                    }
                break;
                case 2:
                    read = -1;
                    buffer = new byte[4096];
                    while (read != 0)
                    {
                        read = input.Read(buffer, 0, buffer.Length);
                        fsout.Write(buffer, 0, read);
                    }
                    break;
            }
            fsout.Close();
            }
        }
    }
```

3.2.8.2. Archivos .AAP

Los archivos *.AAP* son los archivos generados durante la ejecución de una medición de espectro en modo AAP. Dichos archivos poseen las siguientes características básicas.

- Son archivos de texto simple.
- Su longitud es de exactamente 256 líneas, cada una correspondiente a un canal.
- El mayor número posible de cuentas por canal es de 2^24, esto es un total de 16777216 cuentas.
- El valor del primer canal es nulo (incluso en trabajo con todos los canales abiertos), dado que en dicho canal se almacena un contador sin relevancia en la medición.

El archivo de datos en formato $.AAP$, al igual que los archivos $.DAT$, es generado siempre a solicitud del usuario, y es una copia bit a bit del último archivo temporal de la medición. Esto significa que los datos de medición guardados en un archivo $.AAP$ generado por el usuario están tan actualizados como la última solicitud de envío del banco de datos de altura de pulsos del AMCMB96 por parte del programa. Para guardar un espectro de altura de pulsos en formato $.AAP$, el usuario, estando en el modo de trabajo AAP, deberá ingresar a la opción "$Archivo => GuardarComo...$", la cual desplegará el tradicional árbol de archivos de $Windows$ para guardar los datos en la ubicación deseada por el usuario. La rutina de guardado del espectro de altura de pulsos es la misma rutina encargada del guardado del espectro AMC, y se puede consultar en la sección correspondiente.

3.3. Detalles de programación

El desarrollo de $MOSSCOMM$ tuvo particularidades inherentes a la programación en sistemas modernos, y diversas consideraciones en razón a los requerimientos de funcionalidad, compatibilidad y accesibilidad del código. En esta sección se describen estas consideraciones surgidas durante el desarrollo.

3.3.1. El despliegue gráfico de los datos

El despliegue gráfico de los datos se programó con ayuda de la librería ZedGraph, programada en C# y con licenciamiento LGPL, lo cual permite modificar, adaptar y redistribuir el código de la misma a necesidades locales. En este caso, utilizarla sin más restricción que su mención en el código de *MOSSCOMM*. El programa dispone de 2 controles de usuario principales; uno para el despliegue de los espectros de altura de pulsos y el otro para el despliegue de los espectros AMC. La librería cuenta con la funcionalidad de realización de zoom dinámico sobre datos de la gráfica desplegada, la capacidad de enviar una gráfica directamente a impresión (siempre y cuando se tenga una impresora instalada), el marcaje de puntos específicos de las series de datos (pudiendo ver en los mismos los valores de los ejes) y la posibilidad de exportar las gráficas del programa directamente a formatos *JPG*, *PNG* y *EPS*, apropiados para su inserción directa en documentos y artículos.

3.3.2. Hilos de eventos y procesamiento de datos entrantes

Uno de los principales problemas que se encontraron durante el desarrollo y programación de *MOSSCOMM* estuvo en los tiempos que debían manejarse entre el envío de datos, la recepción de los mismos y las validaciones de comunicación exitosa entre el AMCMB96 y la PC, problemas que se debían principalmente a la diferencia de velocidades de procesamiento entre la PC que contenía el programa de manejo original y el moderno servidor que se adquirió para el proyecto. Para solucionar problemas de temporización se implementó una serie de hilos y eventos de comunicación asíncrona, de forma tal que la apertura del puerto serial y el respectivo funcionamiento del protocolo se dieran únicamente por orden de programa y se tuvieran tiempos de espera razonables para la recepción de datos y confirmaciones de éxito de comunicación por parte del AMCMB96. De esta forma, en rutinas como la recepción de los bancos de

datos AMC y AAP se programaron tiempos muertos del orden de 3-5 segundos con el fin de que el AMCMB96 pudiese enviar el buffer de datos y con plena seguridad de que dicho buffer se encontraría completo. De igual manera, se determinó que para el envío de órdenes de programa con caracteres ASCII (prácticamente todas) debía dejarse un intervalo de tiempo muerto de $50ms$ por cada caracter enviado, tardando por tanto los comandos más largos alrededor de $800ms$ para enviarse completamente.

3.3.2.1. Suscripción y desuscripción a eventos

En cada rutina en la cual se espera alguna respuesta por parte del AMCMB96 se definió un manejador de eventos para la recepción de dicha respuesta y el posterior procesamiento de la misma. Desde el chequeo de identidad hasta la recepción de los bancos de datos, pasando por la recepción de los límites de ventana de comparación y las confirmaciones de éxito de recepción de datos, cada uno de los posibles eventos de salida de datos del AMCMB96 tienen su correspondiente manejador de eventos en el código de programa. Todas las rutinas que reciben datos invocan la ejecución del manejador con base en la recepción del evento respectivo, y en el código del propio manejador está programada la desuscripción del evento de datos, de forma tal que al recibir una serie de datos ésta sea procesada únicamente por el manejador adecuado y se eviten conflictos de funcionamiento en el programa.

3.3.2.2. El sistema GIT para desarrollo colaborativo

El programa ha sido desarrollado en Visual C# y todo el código es completamente abierto. Cualquier usuario está explícitamente autorizado a copiar, modificar y redistribuir el programa, con la única obligación de dar los respectivos créditos al autor del mismo si utiliza el código del programa en todo o en parte. Por tal motivo, la totalidad del código del programa se encuentra disponible en el repositorio GIT con la siguiente dirección URL:

https://github.com/crislato/amcmb96

El sistema GIT permite que múltiples desarrolladores puedan trabajar en un mismo código a través de sincronizaciones similares a las utilizadas en sistemas Unix estándar. Cada desarrollador puede probar nuevas funcionalidades en el código, agregar sus propios aportes al programa, probar cualquier cambio programado y consolidar resultados con otros participantes del proyecto. La trazabilidad del código, por tanto, es absoluta; cualquier cambio en el código, por mínimo que sea, deberá ser justificado, comentado y registrado en el árbol de código del proyecto, permitiendo así determinar puntos de inflexión, mejoras importantes y posibles fallas durante los procesos de desarrollo a mediano y largo plazo en el proyecto. De acuerdo con estos principios, se espera que el desarrollo de $MOSSCOMM$ sea constante y que se puedan realizar constantes mejoras al código, siempre en beneficio de los usuarios del programa.

3.4. Implementación y pruebas.

El programa fue probado en diversas ocasiones y con diferentes espectrómetros en el laboratorio de espectroscopía Mössbauer de la Universidad del Valle. La metodología de prueba de correcto funcionamiento consistió en comparar el resultado de la lectura de datos de $MOSSCOMM$ frente a la de $ESPE96$, teniendo que ser idénticos para las mismas muestras y/o el mismo instrumento. Las figuras 3.6 (Muestra de Calibración con ^{57}Fe) y 3.7 (Muestra de C_5N_6OFe) ilustran los espectros obtenidos mediante la lectura de datos con $MOSSCOMM$

Después de comparar los espectros leídos por $MOSSCOMM$ con los leídos por el programa original $ESPE96$, se determinó que no existe ninguna diferencia entre la lectura de los datos por parte de uno u otro programa. Después de estas pruebas de funcionamiento, los espectrómetros AMCMB96 disponibles en el laboratorio de espec-

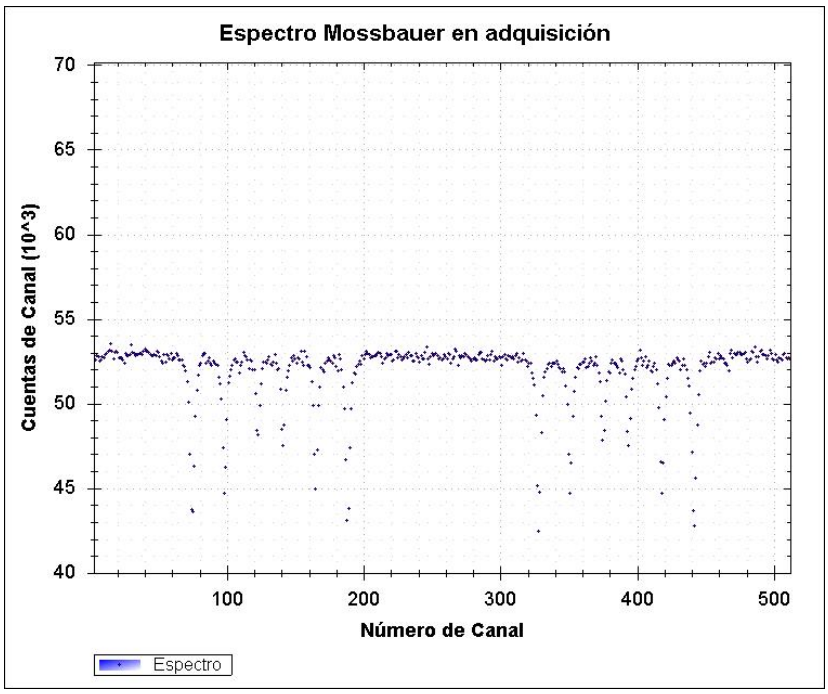

Figura 3.6: Muestra de calibración ^{57}Fe.

troscopía Mössbauer en la Universidad del Valle están permanentemente conectados al nuevo servidor y son manejados y administrados desde la interfaz de $MOSSCOMM$, con lo cual se cumple el objetivo principal de este proyecto.

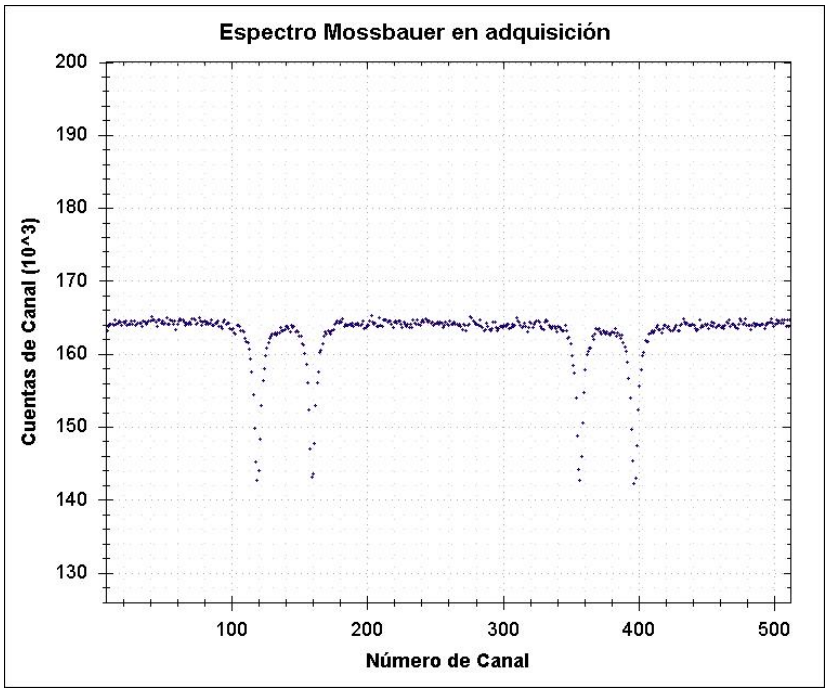

Figura 3.7: Muestra de C_5N_6OFe

Capítulo 4

Conclusiones

En este proyecto se diseñó y programó un manejador (driver) de comunicación entre una PC con sistema operativo tipo *Windows* y el espectrómetro Mössbauer AMCMB96, obteniendo resultados satisfactorios en cuanto a desempeño y confiabilidad de las lecturas de datos generados por varios ejemplares del instrumento estudiado.

El driver de comunicación y programa de manejo desarrollado en este proyecto cumple con todas las funciones de su predecesor y es apto para el trabajo de medición y toma de espectros en ambiente de producción en el laboratorio de espectroscopía Mössbauer de la Universidad del Valle.

Se deja copia de la totalidad del código escrito en lenguaje de alto nivel (C#) tanto en medio local como en un repositorio robusto y remoto. El código es abierto, trazable y completamente adaptable a nuevas necesidades y requerimientos cuya funcionalidad quiera desarrollarse en el grupo de investigación.

Capítulo 5

Perspectivas

A futuro pueden tenerse en cuenta las siguientes propuestas para continuar con el desarrollo de este proyecto:

1. Implementar código apropiado para realizar el procesamiento posterior de los datos (foldeo y ajuste), o por lo menos para exportar los datos directamente a un programa de ajuste.

2. Implementar código apropiado para portar el programa a plataformas POSIX.

3. Implementar código apropiado para el trazado de usuarios y sesiones en el caso en que se usen múltiples espectrómetros de forma simultánea.

4. Implementar código para almacenar los datos de espectros tomados y bitácoras de toma de datos directamente en una base de datos incorporada en el programa.

Apéndice A

Código fuente del programa (Visual C#)

A continuación se muestra la totalidad del código del programa, organizado por clases.

A.1. Código de programa

A.1.1. Clase principal FormMain.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;
using ZedGraph;
using System.IO.Ports;
using System.Timers;
using System.Globalization;

namespace SerialControlAMCMB96
```

```
{
    public partial class FormMain : Form
    {
        string bancoDatosAMC;
        string bancoDatosAAP;
        string amcIniciar;
        int amcID = 0;
        int liminf = 0;
        int limsup = 0;
        System.Timers.Timer refreshTimer = new System.Timers.Timer();

        public FormMain()
        {
            InitializeComponent();

        }

        //Rutina que se encarga de abrir el puerto, validar y enviar el comando especificado en cada
        //método asociado a un evento de click y selección de radio en un botón


        public void InicializarPuerto(System.IO.Ports.SerialPort Puerto)
        /*Variables de configuración del puerto, guardadas en principio por el programa.
          DTR y RTS deben ir a Enable TRUE (Control por hardware)
        */
        {
            if (Puerto.IsOpen) serialPortMain.Close();
            Puerto.PortName = Properties.Settings.Default.COMPortName;
            Puerto.BaudRate = Properties.Settings.Default.COMBaudRate;
            Puerto.Parity = Properties.Settings.Default.COMParity;
            Puerto.DataBits = Properties.Settings.Default.COMDataBits;
            //Puerto.StopBits = Properties.Settings.Default.COMStopBits;
            Puerto.StopBits = System.IO.Ports.StopBits.One;
            Puerto.Handshake = Properties.Settings.Default.COMHandshake;
            Puerto.ReadTimeout = Properties.Settings.Default.COMReadTimeout;
            Puerto.WriteTimeout = Properties.Settings.Default.COMWriteTimeout;

            try
            {
                Puerto.Open();
                Puerto.DtrEnable = true;
```

```
        Puerto.RtsEnable = true;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }

}

public void EnviarOrden(string comando)
//Envío de un comando. Cadena de inicialización empieza con "[".
//El fin de comando se considera con el terminador \r. El \n es agregado por el SO.
//En la rutina se inicializa el puerto, se envía el iniciador, la cadena argumento y el
     terminador.
{
    InicializarPuerto(serialPortMain);
    string inicomando = "[";
    string fincr = "\r";
    serialPortMain.Write(inicomando);
    serialPortMain.Write(comando);
    serialPortMain.Write(fincr);
    //serialPortMain.Close();

}
//Fin de rutina EnviarOrden(comando);

public void EnviarOrdenEsperarRespuesta(string comando)
//Envío de un comando. Cadena de inicialización empieza con "[".
//El fin de comando se considera con el terminador \r. El \n es agregado por el SO.
//En la rutina se inicializa el puerto, se envía el iniciador, la cadena argumento y el
     terminador.
//En esta rutina se espera respuesta por parte del AMCMB96, por eso no se cierra el puerto.
//Si el puerto no existe se presenta una excepción
{
    InicializarPuerto(serialPortMain);
    string inicomando = "[";
    string fincr = "\r";
    serialPortMain.Write(inicomando);
    serialPortMain.Write(comando);
    serialPortMain.Write(fincr);
```

```
}

public void EnviarOrdenByteEsperarRespuesta(byte[] bytecomando)
//Rutina para enviar comandos en formato byte (requisición de espectros)
//El programa espera respuesta por parte del AMCMB96
{
    InicializarPuerto(serialPortMain);
    string inicomando = "[";
    string fincr = "\r";
    serialPortMain.Write(inicomando);
    serialPortMain.Write(bytecomando, 0, bytecomando.Length);
    serialPortMain.Write(fincr);

}

public void EnviarOrdenByte(byte[] bytecomando)
//Rutina para enviar comandos en formato byte (requisición de espectros)
//El programa NO espera respuesta por parte del AMCMB96
{
    InicializarPuerto(serialPortMain);
    string inicomando = "[";
    string fincr = "\r";
    serialPortMain.Write(inicomando);
    serialPortMain.Write(bytecomando, 0, bytecomando.Length);
    serialPortMain.Write(fincr);
    serialPortMain.DiscardInBuffer();
    //serialPortMain.Close();

}

private void ActualiceID()
{
    this.Invoke(new EventHandler(DisplayAMCid));
}

private void DisplayAMCid(object sender, EventArgs e)
{
    labelNumAMC.Text = amcID.ToString();
}
```

```
private void menuChequeoID_Click(object sender, EventArgs e)
/*
 * Cuando se da click en el menú de chequear el ID del espectrómetro activo se procede
 * a enviar el comando "?", el cual debe esperar como respuesta el ID.
 * El handler espera la respuesta y crea los archivos de datos necesarios para la adquisición
 * de la información del espectrómetro y el guardado de sus parámetros de funcionamiento.
 * El Handler se activa al recibir el ID y luego es invocado para actualizar el mismo ID en
 * el hilo correspondiente a las rutinas DisplayAMCid() y ActualiceID()
 */
{
    if (menuChequeoID.Checked == false)
    {
        string comandoID = "?";
        EnviarOrdenEsperarRespuesta(comandoID);
        serialPortMain.DataReceived += new
             SerialDataReceivedEventHandler(serialPortMain_IDReceived);

        //serialPortMain.Close();
        MenuModoAMC.Enabled = true;
        MenuModoAAP.Enabled = true;
        string dataAMC = "currentAMC_Dev" + amcID.ToString() + ".data";
        string dataAAP = "currentAAP_Dev" + amcID.ToString() + ".data";
        if (!File.Exists(dataAMC) | !File.Exists(dataAAP))
        {
            File.Create(dataAMC).Dispose();
            File.Create(dataAAP).Dispose();
        }
    }

}

void serialPortMain_IDReceived(object sender, SerialDataReceivedEventArgs e)
/*
 * Este es el handler que se activa al recibir el ID del espectrómetro.
 * El dato es leído por la rutina y de inmediato se invoca el manejador del handler
 * contenido en ActualiceID(), el cual a su vez invoca a DisplayAMCid para actualizar
 * el valor del ID que es visible al usuario.
 * El handler se desuscribe del evento de datos de puerto serial al terminar su ejecución.
 */
{
    int numdatosEntrada;
```

```
numdatosEntrada = serialPortMain.BytesToRead;

Byte[] datoID = new Byte[numdatosEntrada];
serialPortMain.Read(datoID, 0, numdatosEntrada);
string respuestaId = System.Text.Encoding.UTF8.GetString(datoID);
//MessageBox.Show("Se ha detectado el espectrómetro con ID:" + respuestaId);

    if (respuestaId.Equals("!1", StringComparison.Ordinal))
    {
        amcID = 1;
        ActualiceID();
    }
    if (respuestaId.Equals("!2", StringComparison.Ordinal))
    {
        amcID = 2;
        ActualiceID();
    }
    if (respuestaId.Equals("!3", StringComparison.Ordinal))
    {
        amcID = 3;
        ActualiceID();
    }
    if (respuestaId.Equals("!4", StringComparison.Ordinal))
    {
        amcID = 4;
        ActualiceID();
    }
    if (respuestaId.Equals("!5", StringComparison.Ordinal))
    {
        amcID = 5;
        ActualiceID();
    }
    if (respuestaId.Equals("!6", StringComparison.Ordinal))
    {
        amcID = 6;
        ActualiceID();
    }
    if (respuestaId.Equals("!7", StringComparison.Ordinal))
    {
        amcID = 7;
        ActualiceID();
```

```
        }
        if (respuestaId.Equals("!8", StringComparison.Ordinal))
        {
            amcID = 8;
            ActualiceID();
        }
    serialPortMain.Close();
    serialPortMain.DataReceived -= serialPortMain_IDReceived;
    return;
}


private void salirToolStripMenuItem_Click(object sender, EventArgs e)
//Pregunta de confirmación para salir del programa.
{
    if (MessageBox.Show("Desea salir del programa?", "Salir", MessageBoxButtons.OKCancel) ==
         DialogResult.OK)
    {
        Application.Exit();
    }
}

private void MenuCOM_Click(object sender, EventArgs e)
    //Opciones de comunicación. Se despliega el formulario de opciones.
{
    serialPortMain.Close();
    FormCOMOptions opcionesComm = new FormCOMOptions();
    opcionesComm.Show();
}

private void MenuModoAMC_Click(object sender, EventArgs e)
    /*Rutina que inicia el despliegue gráfico del modo AMC.
     * Cuando se pone check en este ítem, se deshabilitan las opciones de modo AAP
     * y se hacen visibles los paneles y gráficos del modo AMC.
     * El modo gráfico más importante es el correspondiente a ZedGraph,
     * invocado en CreateGraphAMC() y ActualiceGraficoAMC()
     */
{
    MenuModoAMC.Checked = !MenuModoAMC.Checked;
    if (MenuModoAMC.Checked == true)
    {
```

```
    MenuModoAAP.Checked = false;
    MenuModoAAP.Enabled = false;
    panelModoAMC.Enabled = true;
    panelModoAMC.Visible = true;
    CreateGraphAMC(zedGraphControlMainAMC);
    ActualiceGraficoAMC();
    //zedGraphControlMainAMC.Show();
    buttonActivarAMC.Enabled = true;
    buttonActivarAMC.Show();
    Properties.Settings.Default.ModoManejo = 1;
    Properties.Settings.Default.FrecuenciaBarrido = 10;
    Properties.Settings.Default.TiempoMuerto = 20;
    Properties.Settings.Default.Resolucion = 256;
    Properties.Settings.Default.MaxVelocidad = 8;
    Properties.Settings.Default.FactorEscala = 1;
    Properties.Settings.Default.SignoVelocidad = 0;
    Properties.Settings.Default.ModoAvance = Convert.ToChar("I");
    MessageBox.Show("ATENCIÓN:"
    + Environment.NewLine
    + "Si no modifica los parámetros de trabajo, se usarán los siguientes:"
    + Environment.NewLine
    + Environment.NewLine + "Modo de manejo:" +
        Properties.Settings.Default.ModoManejo.ToString()
    + Environment.NewLine + "Frecuencia de barrido:" +
        Properties.Settings.Default.FrecuenciaBarrido.ToString() + " Hz"
    + Environment.NewLine + "Resolución:" +
        Properties.Settings.Default.Resolucion.ToString() + " Canales"
    + Environment.NewLine + "Factor de escala:" +
        Properties.Settings.Default.FactorEscala.ToString()
    + Environment.NewLine + "Signo de velocidad:" +
        Properties.Settings.Default.SignoVelocidad.ToString()
    + Environment.NewLine + "Modo de avance:" +
        Properties.Settings.Default.ModoAvance.ToString()
    );
    zedGraphControlMainAMC.Show();

}
if (MenuModoAMC.Checked == false)
{
    MenuModoAAP.Enabled = true;
    panelModoAMC.Enabled = false;
```

```
        panelModoAMC.Visible = false;
        //buttonActivarAMC.Enabled = true;
        zedGraphControlMainAMC.Hide();
    }
}

public struct Parametros_AMC
{
    public char num_identif;
    public double frec_barrido;
    public double tiempo_muerto;
    public char avance_canal;
    public char fin_canal;
    public int barridos;
    public double max_velocidad;
    public int signo_velocidad;
    public int resolucion;
    public int modo_manejo; //1:Triangular; 2:DienteSierra; 3:Sinusoidal
}

public struct Variables_Manejo_AMC
{
    public int modo;
    public char avance;
    public int longitud;
    public int canales;
    public int tiempo_muestreo;
    public int mb_barridos;
    public double numero_ti_muestreo;
}

public void InicieAMC()
{
    double z, z1;
    int i, i1, i2, j, k, n;
    int maxresol_1 = 1024;
    int maxresol_2 = 512;
    int fact_vel = Properties.Settings.Default.FactorEscala;
    int[] senal = new int[1024];
    double[] senaldouble = new double[1024];
    double frecu = 4915200;
```

```
double pi = 3.141592;
Parametros_AMC prms_amc;
prms_amc.modo_manejo = Properties.Settings.Default.ModoManejo;
prms_amc.frec_barrido = Properties.Settings.Default.FrecuenciaBarrido;
prms_amc.tiempo_muerto = Properties.Settings.Default.TiempoMuerto;
prms_amc.resolucion = Properties.Settings.Default.Resolucion;
prms_amc.max_velocidad = Properties.Settings.Default.MaxVelocidad;
prms_amc.signo_velocidad = Properties.Settings.Default.SignoVelocidad;
prms_amc.avance_canal = Properties.Settings.Default.ModoAvance;

Variables_Manejo_AMC vrmanejo = default(Variables_Manejo_AMC); //poner a default, sino
     aparece como unassigned
vrmanejo.avance = prms_amc.avance_canal;
switch (prms_amc.modo_manejo)
{
    case 1:
        k = maxresol_1 / prms_amc.resolucion; //esto es el quotient res 256 => k=4; 512 =>
             k=2, 1024 => k=1
        j = (maxresol_1 - (k * prms_amc.resolucion)) / 2; //esto es el quotient j (siempre
             es 0)
        vrmanejo.canales = 2 * prms_amc.resolucion;
        vrmanejo.longitud = 3 * vrmanejo.canales;
        senal = new int[1024];
        for (i = 0; i < prms_amc.resolucion; i++)
        {
            senal[i] = j;
            j += k; //256 => 0,4,8,12... 512 => 0,2,4,6,8... 1024 => 0,1,2,3,4....
        }
        fact_vel = Properties.Settings.Default.FactorEscala;
        switch (fact_vel)
        {
            case 1:
                j = 0;
                break;
            case 2:
                j = 256;
                break;
            case 4:
                j = 384;
                break;
            default:
```

```
            j = 0;
            break;
        }

        for (i = 0; i < prms_amc.resolucion; i++)
        {
            //Rampa dividida entre 4 para res 256=>0,1,2... res 512=>0,0.5,1,1.5... res
                1024=>0,0.25,0.5,0.75...
            int xquot = senal[i] / fact_vel;
            //Para fact_vel = 1 Rampa es res 256=>0,1,2... res 512=>0,0.5,1,1.5... res
                1024=>0,0.25,0.5,0.75...
            //Para fact_vel = 2 Rampa es res 256=>256,257,258... res
                512=>256,256.5,257,257.5... res 1024=>256,256.25,256.5,256.75...
            //Para fact_vel = 4 Rampa es res 256=>384,385,386... res
                512=>384,384.5,385,385.5... res 1024=>384,384.25,384.5,384.75...
            senal[i] = xquot + j;
            //simetria de la señal. senal[255] == senal[1], senal[254] == senal[2],
                senal[253] == senal[3]....
            senal[vrmanejo.canales - 1 - i] = senal[i];
        }
        z = frecu / (vrmanejo.canales * prms_amc.frec_barrido);
        vrmanejo.tiempo_muestreo = CalculaTiempos(z, vrmanejo, out n);
        vrmanejo.numero_ti_muestreo = n;
        if (vrmanejo.numero_ti_muestreo == 1) vrmanejo.modo = 1; //buffer corto
        else vrmanejo.modo = 2; //buffer largo
        break;
    /////////CASO DIENTE DE SIERRA/////////
    case 2:
        k = maxresol_1 / prms_amc.resolucion; //esto es el quotient
        j = (maxresol_1 - (k * prms_amc.resolucion)) / 2; //esto es el quotient j (siempre
            es 0)
        vrmanejo.canales = (int)(prms_amc.resolucion*(1 + prms_amc.tiempo_muerto/100));
        i1 = vrmanejo.canales - prms_amc.resolucion;
        vrmanejo.longitud = 3 * vrmanejo.canales;
        senal = new int[1024];
        for (i = 0; i < prms_amc.resolucion; i++)
        {
            senal[i] = j+(i*k);
        }
        k = (prms_amc.resolucion - 1) * (k / (i1 + 1));
        for (i = 0; i < i1; i++)
```

```
    {
        senal[vrmanejo.canales - 1 - i] = j + (k * i);
    }
    if (prms_amc.signo_velocidad == -1)
    {
        for (i = 0; i < vrmanejo.canales; i++)
        {
            senal[i] = maxresol_1 - 1 - senal[i];
        }
    }
    switch (fact_vel)
    {
        case 1:
            j = 0;
            break;
        case 2:
            j = 256;
            break;
        case 4:
            j = 384;
            break;
        default:
            j = 0;
            break;
    }
    for (i = 0; i < vrmanejo.canales; i++)
    {
        int xquot = senal[i] / fact_vel;
        senal[i] = xquot + j;
    }
    z = frecu / (vrmanejo.canales * prms_amc.frec_barrido);
    vrmanejo.tiempo_muestreo = CalculaTiempos(z, vrmanejo, out n);
    vrmanejo.numero_ti_muestreo = n;
    if (vrmanejo.numero_ti_muestreo == 1) vrmanejo.modo = 1; //buffer corto
    else vrmanejo.modo = 2; //buffer largo
    break;

    ///////////////CASO SINUSOIDAL///////////////
case 3:
    z1 = pi / prms_amc.resolucion;
    vrmanejo.canales = 2 * prms_amc.resolucion;
```

```
        i1 = vrmanejo.canales / 4;
        vrmanejo.longitud = 3 * vrmanejo.canales;
        for (i = 0; i < prms_amc.resolucion; i++)
        {
            senaldouble[i] = 511*(1.0 - Math.Cos(z1*(i+0.5)));
        }
        MessageBox.Show("La señal es:" + string.Join(",", senaldouble));
        switch (fact_vel)
        {
            case 1:
                j = 0;
                break;
            case 2:
                j = 256;
                break;
            case 4:
                j = 384;
                break;
            default:
                j = 0;
                break;
        }
        for (i = 0; i < prms_amc.resolucion; i++)
        {
            double xquot = (double)senaldouble[i] / fact_vel;
            senaldouble[i] = xquot + j;
            senaldouble[vrmanejo.canales - 1 - i] = senaldouble[i];
        }
        z = frecu / (vrmanejo.canales * prms_amc.frec_barrido);
        vrmanejo.tiempo_muestreo = CalculaTiempos(z, vrmanejo, out n);
        vrmanejo.numero_ti_muestreo = n;
        if (vrmanejo.numero_ti_muestreo == 1) vrmanejo.modo = 1; //buffer corto
        else vrmanejo.modo = 2; //buffer largo
        break;
}
MessageBox.Show("Voy a enviar los siguientes datos:"
    + Environment.NewLine + vrmanejo.avance.ToString()
    + Environment.NewLine + vrmanejo.longitud.ToString()
    + Environment.NewLine + vrmanejo.canales.ToString()
    + Environment.NewLine + vrmanejo.mb_barridos.ToString()
    + Environment.NewLine + vrmanejo.tiempo_muestreo.ToString()
```

```
        + Environment.NewLine + vrmanejo.numero_ti_muestreo.ToString()
        );

    EnviarDatosOperacionAMC(vrmanejo);
    System.Threading.Thread.Sleep(1000);
    if (prms_amc.modo_manejo == 1 || prms_amc.modo_manejo == 2)
    {
        EnviarSenal(vrmanejo, senal);
    }
    else EnviarSenalDouble(vrmanejo, senaldouble);

    System.Threading.Thread.Sleep(5000);
    string modo_oper = vrmanejo.modo.ToString();
    amcIniciar = modo_oper + "I";
    //string amcIniciar = "1I";
    MessageBox.Show("Comando enviado para iniciar: " + amcIniciar);
    EnviarOrden(amcIniciar);
    MessageBox.Show("Flag iniciar");
    serialPortMain.DataReceived += new
         SerialDataReceivedEventHandler(serialPortMain_ConfirmParamsReceived);
    //Byte[] confirmacion = new Byte[1];
    //serialPortMain.Read(confirmacion, 0, 1);


}

public void EnviarDatosOperacionAMC(Variables_Manejo_AMC manejo)
{
    manejo.mb_barridos = 0;
    byte[] avance = BitConverter.GetBytes(manejo.avance);
    byte[] longi = BitConverter.GetBytes(Convert.ToInt16(manejo.longitud));
    byte[] canales = BitConverter.GetBytes(Convert.ToInt16(manejo.canales));
    byte[] barridos = BitConverter.GetBytes(Convert.ToInt16(manejo.mb_barridos));
    byte[] tiempo_muestreo = BitConverter.GetBytes(Convert.ToInt16(manejo.tiempo_muestreo));
    byte[] numero_ti_muestreo =
         BitConverter.GetBytes(Convert.ToInt16(manejo.numero_ti_muestreo));
    //Estructura del comando = 0x44 (D), 0x00, 0x49 (I), 0x00, 0x00, 0x00 (3 campos de 1 byte a
         0), 0x00, 0x00 (campo 2 bytes a 0), longitud (2 bytes),
    //canales (2 bytes), barridos (2 bytes), tiempo_muestreo (2 bytes / z), numero_ti_muestreo
         (2 bytes)
```

```
    //byte[] datos_oper_byte = new byte[18] { 0x44, 0x00, 0x49, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x06, 0x00, 0x02, 0x00, 0x00, 0xC0, 0x03, 0x01, 0x00 };
    byte[] datos_oper_byte = new byte[18] { 0x44, 0x00, avance[0], 0x00, 0x00, 0x00, 0x00,
        0x00, longi[0], longi[1], canales[0], canales[1], barridos[0], barridos[1],
        tiempo_muestreo[0], tiempo_muestreo[1], numero_ti_muestreo[0], numero_ti_muestreo[1] };
    InicializarPuerto(serialPortMain);
    string inicomando = "[";
    serialPortMain.Write(inicomando);
    for (int i = 0; i < datos_oper_byte.Length; i++)
    {
        byte[] chain = new byte[1] { datos_oper_byte[i] };
        serialPortMain.Write(chain, 0, chain.Length);
        System.Threading.Thread.Sleep(300);

    }
    string fincr = "\r";
    serialPortMain.Write(fincr);
    MessageBox.Show("Flag envio datos");
    serialPortMain.DataReceived += new
        SerialDataReceivedEventHandler(serialPortMain_ConfirmParamsReceived);
}

public void EnviarSenal(Variables_Manejo_AMC manejo, int[] senal)
{
    int i, j;
    int xquot, xrem;
    int[] senalalta = new int[1024];
    int[] senalbaja = new int[1024];
    int chks = 0;
    string alta, baja;
    byte[] altabyte, bajabyte;

    for (i = 0; i < manejo.canales; i++)
    {
        xquot = Math.DivRem(senal[i], 4, out xrem);
        senalalta[i] = xquot;
        chks += senalalta[i];
        senalbaja[i] = xrem;
        chks += senalbaja[i];
        //Ver operación de CHKS (sumas de chequeo, bitwise comparison chks = ~chks +1)
    }
```

```
    chks += chks;
    chks = (~chks) + 1;

    instanciarRecepcionBDatos(manejo);
    System.Threading.Thread.Sleep(2000);
    //InicializarPuerto(serialPortMain);

    for (i = 0; i < manejo.canales; i++)
    {
        alta = senalalta[i].ToString();
        baja = senalbaja[i].ToString();
        altabyte = new byte[] { Convert.ToByte(senalalta[i]) };
        bajabyte = new byte[] { Convert.ToByte(senalbaja[i]) };

        serialPortMain.Write(altabyte, 0, 1);
        serialPortMain.Write(bajabyte, 0 ,1);


    }
    //MessageBox.Show("Escribí:" + string.Join(",", senalalta));
    //MessageBox.Show("Escribí:" + string.Join(",", senalbaja));
    //byte[] chksbyte = new byte[] { Convert.ToByte(chks)};
    byte[] chkszero = new byte[1] { 0x00 };
    serialPortMain.Write(chkszero,0,1);
    //serialPortMain.Close();
    //serialPortMain.ReadByte();
}

public void EnviarSenalDouble(Variables_Manejo_AMC manejo, double[] senaldouble)
{
    int i, j;
    int xquot;
    double xrem;
    int[] senalalta = new int[1024];
    double[] senalbaja = new double[1024];
    int chks = 0;
    string alta, baja;
    byte[] altabyte, bajabyte;

    for (i = 0; i < manejo.canales; i++)
    {
```

```
            xquot = Convert.ToInt32(senaldouble[i]) / 4;
            senalalta[i] = xquot;
            chks += senalalta[i];
            xrem = senaldouble[i] % 4;
            senalbaja[i] = xrem;
            chks += Convert.ToInt32(senalbaja[i]);
            //Ver operación de CHKS (sumas de chequeo, bitwise comparison chks = ~chks +1)
        }
        //MessageBox.Show("La señal es:" + string.Join(",", senalalta));
        //MessageBox.Show("Escribí:" + string.Join(",", senalbaja));

        chks += chks;
        chks = (~chks) + 1;

        instanciarRecepcionBDatos(manejo);
        System.Threading.Thread.Sleep(2000);
        //InicializarPuerto(serialPortMain);

        for (i = 0; i < manejo.canales; i++)
        {
            alta = senalalta[i].ToString();
            baja = senalbaja[i].ToString();
            altabyte = new byte[] { Convert.ToByte(senalalta[i]) };
            bajabyte = new byte[] { Convert.ToByte(senalbaja[i]) };

            serialPortMain.Write(altabyte, 0, 1);
            serialPortMain.Write(bajabyte, 0, 1);


        }
        //MessageBox.Show("Valor CHKS Final:" + chks.ToString());
        byte[] chkszero = new byte[1] { 0x00 };
        serialPortMain.Write(chkszero, 0, 1);
        //serialPortMain.Close();
        //serialPortMain.ReadByte();
    }

    public int CalculaTiempos( double zz, Variables_Manejo_AMC manejo, out int numerotimuestreo)
    {
        double origvalue = zz / (double)65536;
        double intpart = Math.Floor(origvalue);
```

```
    manejo.numero_ti_muestreo = 1 + intpart;
    numerotimuestreo = (int)manejo.numero_ti_muestreo;
    if ((zz / (double)manejo.numero_ti_muestreo) > 65536)
    {
double origvalue2 = (zz / (double)manejo.numero_ti_muestreo) - 65536;
        manejo.tiempo_muestreo = (int)Math.Floor(origvalue2);
        //MessageBox.Show("Bucle 1 CalculaTiempo, tiempo_muestreo:" + manejo.tiempo_muestreo);
        return manejo.tiempo_muestreo;
    }
    else
    {
        double origvalue2 = (zz / (double)manejo.numero_ti_muestreo); //10Hz => 960 /1 = 960
        manejo.tiempo_muestreo = (int)Math.Floor(origvalue2);
        //MessageBox.Show("Bucle 2 CalculaTiempo, tiempo_muestreo:" + manejo.tiempo_muestreo);
        return manejo.tiempo_muestreo;
    }
}

private void MenuModoAAP_Click(object sender, EventArgs e)
/*Rutina que inicia el despliegue gráfico del modo AAP.
 * Cuando se pone check en este ítem, se deshabilitan las opciones de modo AMC
 * y se hacen visibles los paneles y gráficos del modo AAP.
 * El modo gráfico más importante es el correspondiente a ZedGraph,
 * invocado en CreateGraphAMC() y ActualiceGraficoAMC()
 */
{
    MenuModoAAP.Checked = !MenuModoAAP.Checked;
    if (MenuModoAAP.Checked == true)
    {
        MenuModoAMC.Checked = false;
        MenuModoAMC.Enabled = false;
        panelModoAAP.Enabled = true;
        panelModoAAP.Location = new Point(0, 44);
        panelModoAAP.Visible = true;
        CreateGraphAAP(graphMain);
        ActualiceGraficoAAP();
        graphMain.Show();
    }

    if (MenuModoAAP.Checked == false)
    {
```

```
            MenuModoAMC.Enabled = true;
            panelModoAAP.Enabled = false;
            panelModoAAP.Visible = false;
            buttonActivarAAP.Enabled = true;
            graphMain.Hide();
        }
    }

    private void buttonActivarAMC_Click(object sender, EventArgs e)
    {
        if (radioButtonAMCInit.Checked == true)
        {
            string limbuf = "LM";
            EnviarOrden(limbuf);
            InicieAMC();
            //InicializarPuerto(serialPortMain);
            //byte[] control_interrumpir = new byte[3] { 0x43, 0x4D, 0x41 };
            //string inicomando = "[";
            //serialPortMain.Write(inicomando);
            //for (int i = 0; i < control_interrumpir.Length; i++)
            //{
            //    byte[] chain = new byte[1] { control_interrumpir[i] };
            //    serialPortMain.Write(chain, 0, chain.Length);
            //    System.Threading.Thread.Sleep(50);

            //}
            //string fincr = "\r";
            //serialPortMain.Write(fincr);



        }
        if (radioButtonAMCContinue.Checked == true)
        {
            string amcCortoSeguir = "1G";
            EnviarOrden(amcCortoSeguir);
        }

/*          if (comboBoxAMCMode.SelectedIndex == 0 & radioButtonAMCInit.Checked == true)
        {
            string amcCortoIniciar = "1I";
```

```
            string limbuf = "LM";
            EnviarOrden(limbuf);
            EnviarOrden(amcCortoIniciar);
        }
        if (comboBoxAMCMode.SelectedIndex == 0 & radioButtonAMCContinue.Checked == true)
        {
            string amcCortoSeguir = "1G";
            EnviarOrden(amcCortoSeguir);
        }
        if (comboBoxAMCMode.SelectedIndex == 1 & radioButtonAMCInit.Checked == true)
        {
            string amcLargoIniciar = "2I";
            string limbuf = "LM";
            EnviarOrden(limbuf);
            EnviarOrden(amcLargoIniciar);
        }
        if (comboBoxAMCMode.SelectedIndex == 1 & radioButtonAMCContinue.Checked == true)
        {
            string amcLargoSeguir = "2G";
            EnviarOrden(amcLargoSeguir);
        }
*/
        buttonActivarAMC.Enabled = true;
    }

    private void buttonActivarAAP_Click(object sender, EventArgs e)
    {
        if (comboBoxModoAAP.SelectedIndex == 0 & radioButtonAAPInit.Checked == true)
        {
            string aapIniciar = "0I";
            string limbuf = "LP";
            EnviarOrden(limbuf);
            EnviarOrden(aapIniciar);
        }
        if (comboBoxModoAAP.SelectedIndex == 0 & radioButtonAAPContinue.Checked == true)
        {
            string aapSeguir = "0G";
            EnviarOrden(aapSeguir);
        }
        buttonActivarAAP.Enabled = true;
    }
```

```
//Comando de control de interrupción, rutina que en este momento no hace nada
//debe ponerse un botón más amigable y/o eliminar para poner una forma más entendible
//de pausa y reanudación de las mediciones
private void buttonEnviarComandoIntAAP_Click(object sender, EventArgs e)
{
    if (radioButtonControlIntAAPAct.Checked == true)
    {
                }
    if (radioButtonControlIntAAPDesact.Checked == true)
    {

    }
}

//Limpieza de buffer. Esta rutina debe usarse con cuidado, escribe ceros en el banco de memoria.
private void buttonLimpiarBancoAMC_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("La limpieza del banco de datos eliminará cualquier dato no guardado"
        + Environment.NewLine + "Está seguro de ejecutar la limpieza del banco de datos AMC?",
        "Precaución - Posible pérdida de Información", MessageBoxButtons.OKCancel) ==
            DialogResult.OK)
    {
        string limbufamc = "LM";
        EnviarOrden(limbufamc);
        amcIniciar = "1I";
        EnviarOrden(amcIniciar);
        return;
    }
    else return;
}

//Limpieza de buffer. Esta rutina debe usarse con cuidado, escribe ceros en el banco de memoria.
private void buttonLimpiarBancoAAP_Click(object sender, EventArgs e)
{
    if (MessageBox.Show("La limpieza del banco de datos eliminará cualquier dato no guardado"
        + Environment.NewLine + "Está seguro de ejecutar la limpieza del banco de datos AAP?",
        "Precaución - Posible pérdida de Información", MessageBoxButtons.OKCancel) ==
            DialogResult.OK)
    {
        string limbufaap = "LP";
```

```
        EnviarOrden(limbufaap);
        return;
    }
    else return;
}

public void instanciarPuertoSerial()
    /*
     * Si el menú de un modo está activo, se envía la orden para recibir el espectro
     * del respectivo modo. Los modos son excluyentes así que la rutina es segura.
     * Se envía la orden, se espera el buffer y con el evento de recepción de datos
     * se invocan los Handlers para procesar los buffers y escribirlos en los respectivos
     * archivos.
     * Por el motivo anterior se declaran aquí los nombres de los archivos.
     */
{
    string archivoAMC = "currentAMC_Dev" + amcID.ToString() + ".data";
    string archivoAAP = "currentAAP_Dev" + amcID.ToString() + ".data";

    if (MenuModoAMC.Checked == true)
    {
        InicializarPuerto(serialPortMain);
        byte[] control_interrumpir = new byte[3] { 0x43, 0x4D, 0x41 };
        string inicomando = "[";
        serialPortMain.Write(inicomando);
        for (int i = 0; i < control_interrumpir.Length; i++)
        {
            byte[] chain = new byte[1] { control_interrumpir[i] };
            serialPortMain.Write(chain, 0, chain.Length);
            System.Threading.Thread.Sleep(50);

        }
        string fincr = "\r";
        serialPortMain.Write(fincr);
        byte[] bufamc = new byte[10] { 0x45, 0x4D, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
             0x06 };
        //byte[] buflargo = new byte[11] { 0x45, 0x4D, 0x00, 0x00, 0x00, 0x00, 0x00, 0x31, 0x35,
             0x33, 0x37 };
        serialPortMain.Write(inicomando);

        for (int i = 0; i < bufamc.Length; i++)
```

```
        {
            byte[] chain = new byte[1] { bufamc[i] };
            serialPortMain.Write(chain, 0, chain.Length);
            System.Threading.Thread.Sleep(100);

        }
        serialPortMain.Write(fincr);
        //Envía [EM000001537 (1536/3 = 512 = N, y 3N + 1 = 1537)
        //Envía ( 0x45 (E), 0x4D (M), 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 (4 campos de 1 byte,
             uno de 2 bytes), 0x0601 (Longitud)
        //El handler serialPortMain_BuffAMCReceived maneja el retorno de los datos del espectro.
        //serialPortMain.Close();

        //EnviarOrdenByteEsperarRespuesta(buflargo);
        serialPortMain.DataReceived += new
             SerialDataReceivedEventHandler(serialPortMain_BuffAMCReceived);
        //System.Threading.Thread.Sleep(1000);
        //byte[] control_reanudar = new byte[3] { 0x43, 0x4D, 0x41 };
        //serialPortMain.Write(inicomando);
        //for (int i = 0; i < control_reanudar.Length; i++)
        //{
        //    byte[] chain = new byte[1] { control_reanudar[i] };
        //    serialPortMain.Write(chain, 0, chain.Length);
        //    System.Threading.Thread.Sleep(100);

        //}
        //serialPortMain.Write(fincr);

    }
    if (MenuModoAAP.Checked == true)
    {
        string bufaap = "EP";
        EnviarOrdenEsperarRespuesta(bufaap);
        serialPortMain.DataReceived += new
             SerialDataReceivedEventHandler(serialPortMain_BuffAAPReceived);
        //envia [EP (envío único del modo PHA/AAP)
        //El handler serialPortMain_BuffAAPReceived maneja el retorno de los datos PHA/AAP.
        //serialPortMain.Close();

    }
}
```

```
public void instanciarRecepcionBDatos(Variables_Manejo_AMC manejo)
{
        int longitud = 2*manejo.canales + 1;
        string longi = longitud.ToString();
        string prepbdatos = "RS00000" + longi;
        MessageBox.Show("Comando enviado recepcion señal datos:" + prepbdatos);

        byte[] datos_senal_byte = new byte[10] { 0x52, 0x53, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
             0x01, 0x04};
        InicializarPuerto(serialPortMain);
        string inicomando = "[";
        serialPortMain.Write(inicomando);
        for (int i = 0; i < datos_senal_byte.Length; i++)
        {
            byte[] chain = new byte[1] { datos_senal_byte[i] };
            serialPortMain.Write(chain, 0, chain.Length);

        }
        string fincr = "\r";
        serialPortMain.Write(fincr);
        serialPortMain.DataReceived += new
             SerialDataReceivedEventHandler(serialPortMain_ConfirmBDReceived);
    //Recepción del banco de datos.
    //Por ahora, se espera confirmación del banco y se pasan los datos recibidos por evento
    //hacia el handler serialPortMain_ConfirmBDReceived
        //serialPortMain.Close();

}



private void parametrosAMCMenuItem_Click(object sender, EventArgs e)
{
    FormParamAMC parametrosAMC = new FormParamAMC();
    parametrosAMC.Show();
}


private void buttonPonerLimites_Click(object sender, EventArgs e)
```

```
{
    byte liminferior = (byte)numericUpDownLimInf.Value;
    byte limsuperior = (byte)numericUpDownLimSup.Value;
    byte[] comventana = new byte[6] { 0x56, 0x52, 0x30, 0x30, liminferior, limsuperior};
    //Envía [VR00XY (X = valor byte del límite inferior, Y = valor byte límite superior
    InicializarPuerto(serialPortMain);
    string inicomando = "[";
    serialPortMain.Write(inicomando);
    for (int i = 0; i < comventana.Length; i++)
    {
        byte[] chain = new byte[1] { comventana[i] };
        serialPortMain.Write(chain, 0, chain.Length);
        System.Threading.Thread.Sleep(300);

    }
    string fincr = "\r";
    serialPortMain.Write(fincr);
    string verventana = "VE";
    EnviarOrdenEsperarRespuesta(verventana);
    serialPortMain.DataReceived +=new
         SerialDataReceivedEventHandler(serialPortMain_LimsReceived);
    //Envía [VE y espera la respuesta. El flujo de datos de respuesta es manejado por
    //el handler serialPortMain_LimsReceived, que invoca una rutina para escribir los lims en
         la GUI.

}

void serialPortMain_LimsReceived(object sender, SerialDataReceivedEventArgs e)
{
    System.Threading.Thread.Sleep(1000);
    int numdatosEntrada;
    numdatosEntrada = serialPortMain.BytesToRead - 1;
    //MessageBox.Show("Tengo para leer estos datos: " + numdatosEntrada);
    Byte[] datoLims = new Byte[numdatosEntrada];
    serialPortMain.ReadByte();
    serialPortMain.Read(datoLims, 0, numdatosEntrada);
    //string respuestaLims = BitConverter.ToString(datoLims).Replace("-", " ");
    //MessageBox.Show("Toma tus datos:" + respuestaLims);
    byte[] limsupbyte = BitConverter.GetBytes(datoLims[0]);
    byte[] liminfbyte = BitConverter.GetBytes(datoLims[1]);
    liminf = BitConverter.ToInt16(limsupbyte, 0);
```

```
    limsup = BitConverter.ToInt16(liminfbyte, 0);
    ActualiceLimites();
    serialPortMain.DataReceived -=new
        SerialDataReceivedEventHandler(serialPortMain_LimsReceived);
    //Desuscripción al hilo de eventos.
}

private void ActualiceLimites()
{
    this.Invoke(new EventHandler(DisplayAAPLims));
}

private void DisplayAAPLims(object sender, EventArgs e)
{
    labelLimInfValue.Text = liminf.ToString();
    labelLimSupValue.Text = limsup.ToString();
    //este handler es invocado por ActualiceLimites(), y es el que modifica los valores en la
        GUI
}

void serialPortMain_ConfirmParamsReceived(object sender, SerialDataReceivedEventArgs e)
{
    System.Threading.Thread.Sleep(1000);
    int numdatosEntrada;
    numdatosEntrada = serialPortMain.BytesToRead;

    Byte[] datoOK = new Byte[numdatosEntrada];
    serialPortMain.Read(datoOK, 0, numdatosEntrada);
    string respuestaOK = System.Text.Encoding.UTF8.GetString(datoOK);
    //MessageBox.Show("Respuesta Params:" + respuestaOK);

    /*if (respuestaOK.Equals("!", StringComparison.Ordinal))
    {
        MessageBox.Show("Correcta la respuesta Params");
    }
    */
    serialPortMain.DataReceived -= new
        SerialDataReceivedEventHandler(serialPortMain_ConfirmParamsReceived);
}

void serialPortMain_ConfirmBDReceived(object sender, SerialDataReceivedEventArgs e)
```

```
{
    //System.Threading.Thread.Sleep(1000);
    int numdatosEntrada;
    numdatosEntrada = serialPortMain.BytesToRead;
    Byte[] datoOK = new Byte[numdatosEntrada];
    serialPortMain.Read(datoOK, 0, numdatosEntrada);
    string respuestaOK = System.Text.Encoding.UTF8.GetString(datoOK);
    //MessageBox.Show("Respuesta BDatos:" + respuestaOK);

    /*if (respuestaOK.Equals("!", StringComparison.Ordinal) || respuestaOK.Equals("@",
         StringComparison.Ordinal))
    {
        MessageBox.Show("Correcta la respuesta Bdatos");
    }
    */
    serialPortMain.DataReceived -= new
         SerialDataReceivedEventHandler(serialPortMain_ConfirmBDReceived);
}

void serialPortMain_BuffAMCReceived(object sender, SerialDataReceivedEventArgs e)

{
    System.Threading.Thread.Sleep(2000);
    //int numdatosEntrada;
    int numdatosEntradabtr = serialPortMain.BytesToRead;
    //numdatosEntrada = 1548;
    //MessageBox.Show("Tengo para leer estos datos: "+ numdatosEntradabtr);
    Byte[] datoBuffAMCReal = new Byte[numdatosEntradabtr];
    serialPortMain.Read(datoBuffAMCReal, 0, numdatosEntradabtr);

    //Actualiza.
    string respuestaBuffAMCReal = System.Text.Encoding.UTF8.GetString(datoBuffAMCReal);
    //MessageBox.Show("Toma tus datos:" + String.Join(",", respuestaBuffAMCReal));
    Byte[] datoBuffAMC = new Byte[1540];
    long[] datoEspecAMC = new long[512];
    //serialPortMain.Read(datoBuffAMC, 0, numdatosEntrada);
    int i = 0;
    int j = 0;
    Array.Clear(datoEspecAMC, 0, datoEspecAMC.Length);
    for (i = 2, j = 0; i < (512 * 3); j++, i += 3)
    {
```

```
        //Debug
        //long dato1n = int.Parse(datoBuffAMCReal[i].ToString(), NumberStyles.HexNumber);
        //long dato2n = int.Parse(datoBuffAMCReal[i + 1].ToString(), NumberStyles.HexNumber);
        //long dato3n = int.Parse(datoBuffAMCReal[i + 2].ToString(), NumberStyles.HexNumber);
        int dato1n = Convert.ToInt32(datoBuffAMCReal[i]);
        int dato2n = Convert.ToInt32(datoBuffAMCReal[i+1]);
        int dato3n = Convert.ToInt32(datoBuffAMCReal[i+2]);
        //MessageBox.Show("Datos línea " + j + ": " + dato1n.ToString() + "," +
             dato2n.ToString() + "," + dato3n.ToString());
        datoEspecAMC[j] = dato1n + (256 * dato2n) + (65536 * dato3n);
        dato1n = 0;
        dato2n = 0;
        dato3n = 0;
    }
    string respuestaBuffAMC = BitConverter.ToString(datoBuffAMCReal).Replace("-", " ");
    string arrayLong = String.Join(",", datoEspecAMC.Select(p => p.ToString()).ToArray());
    //MessageBox.Show("Toma tus datos:" + respuestaBuffAMC);
    //MessageBox.Show("Vector de datos:" + arrayLong);
    string archivoAMCdraw = "currentAMC_Dev" + amcID.ToString() + ".data";
    StreamWriter espeamc = new System.IO.StreamWriter(archivoAMCdraw);
    long numbarridos = datoEspecAMC[datoEspecAMC.Length - 1];
    for (i = 1; i < datoEspecAMC.Length - 1; i++)
    {
        espeamc.WriteLine(datoEspecAMC[i].ToString());
    }
    espeamc.WriteLine(datoEspecAMC[509].ToString());
    espeamc.WriteLine(datoEspecAMC[510].ToString());
    espeamc.Close();
    serialPortMain.DataReceived -= new
         SerialDataReceivedEventHandler(serialPortMain_BuffAMCReceived);

    //serialPortMain.Close();
    CreateGraphAMC(zedGraphControlMainAMC);
    ActualiceGraficoAMC();



}

/* Esta rutina procesa el evento de recepción de datos del modo PHA/AAP para luego invocar el
 * modo gráfico.
```

```
 * Primero se recibe el buffer, se genera un vector de tipo long, 256 componentes.
 * Se hace un bucle doble hasta el final del vector de datos original proveniente del AMCMB96,
 * que tiene 770 bytes (256*3 de datos + la confirmación "!"). De acuerdo con esta operación
 * se genera el vector de pulsos del buffer AAP.
 * Se escriben los datos en el archivo del buffer, se invoca la desuscripción al evento y se
 * llama la ejecución del despliegue de la gráfica del espectro.
 *
 */
public void serialPortMain_BuffAAPReceived(object sender, SerialDataReceivedEventArgs e)
{
    System.Threading.Thread.Sleep(6000);
    int numdatosBuferEntrada;
    numdatosBuferEntrada = serialPortMain.BytesToRead;
    //MessageBox.Show("Tengo para leer estos datos: " + numdatosBuferEntrada);
    Byte[] datoBuffAAP = new Byte[numdatosBuferEntrada];//770

    long[] datoPulsosBufAAP = new long[256];
    serialPortMain.Read(datoBuffAAP, 0, numdatosBuferEntrada);
    int i = 0;
    int j = 0;
    Array.Clear(datoPulsosBufAAP, 0, datoPulsosBufAAP.Length);
    for (i = 1, j = 0; i < (256 * 3); j++, i += 3)
    {
        long dato1n = Int64.Parse(datoBuffAAP[i].ToString(), NumberStyles.HexNumber);
        long dato2n = Int64.Parse(datoBuffAAP[i + 1].ToString(), NumberStyles.HexNumber);
        long dato3n = Int64.Parse(datoBuffAAP[i + 2].ToString(), NumberStyles.HexNumber);
        datoPulsosBufAAP[j] = dato1n + 256 * dato2n + 65536 * dato3n;
        dato1n = 0;
        dato2n = 0;
        dato3n = 0;
    }
    string respuestaBuff = BitConverter.ToString(datoBuffAAP).Replace("-", " ");
    string arrayLong = String.Join(",", datoPulsosBufAAP.Select(p=>p.ToString()).ToArray());
    //MessageBox.Show("Toma tus datos:" + respuestaBuff);
    //MessageBox.Show("Vector de datos:" + arrayLong);
    string archivoAAPdraw = "currentAAP_Dev" + amcID.ToString() + ".data";
    StreamWriter espeaap = new System.IO.StreamWriter(archivoAAPdraw);
    for (i = 1; i < datoPulsosBufAAP.Length; i++)
    {
        espeaap.WriteLine(datoPulsosBufAAP[i].ToString());
    }
```

```
    espeaap.Close();
    serialPortMain.DataReceived -= new
         SerialDataReceivedEventHandler(serialPortMain_BuffAAPReceived);
    //serialPortMain.Close();
    CreateGraphAAP(graphMain);
    ActualiceGraficoAAP();

}

//Como es una actualización del componente gráfico, debe invocarse un EventHandler.
private void ActualiceGraficoAAP()
{
    this.Invoke(new EventHandler(DisplayAAPGraph));
}

//Este handler es invocado por ActualiceGraficoAAP.
private void DisplayAAPGraph(object sender, EventArgs e)
{
    graphMain.Location = new Point(227, 44);
    // Leave a small margin around the outside of the control
    graphMain.Size = new Size(ClientRectangle.Width - 250,
                          ClientRectangle.Height - 100);
    //graphMain.Show();
}

//Como es una actualización del componente gráfico, debe invocarse un EventHandler.
private void ActualiceGraficoAMC()
{
    this.Invoke(new EventHandler(DisplayAMCGraph));
}

//Handler idéntico al de PHA/AAP, pero para AMC.
private void DisplayAMCGraph(object sender, EventArgs e)
{
    zedGraphControlMainAMC.Location = new Point(227, 44);
    // Leave a small margin around the outside of the control
    zedGraphControlMainAMC.Size = new Size(ClientRectangle.Width - 250,
                          ClientRectangle.Height - 100);
}

private void buttonLeerEspecAMC_Click(object sender, EventArgs e)
```

```
{
    instanciarPuertoSerial();
}

private void refreshTimer_Elapsed(object sender, ElapsedEventArgs e)
{
    instanciarPuertoSerial();
    graphMain.Refresh();
    CreateGraphAAP(graphMain);
    ActualiceGraficoAAP();
}


private void buttonLeerDatosAAP_Click(object sender, EventArgs e)
{
    instanciarPuertoSerial();
}


private void FormMain_Resize(object sender, EventArgs e)
{
    ActualiceGraficoAAP();
}

/*
private void SetSize()
{
    zedGraphControlMain.Location = new Point(227, 44);
    // Leave a small margin around the outside of the control
    zedGraphControlMain.Size = new Size(ClientRectangle.Width - 250,
                         ClientRectangle.Height - 100);
}
*/

private void CreateGraphAAP(ZedGraphControl graphMainAAP)
{
    // get a reference to the GraphPane
    GraphPane PaneAAP = graphMainAAP.GraphPane;
    // Titulos y ejes
    PaneAAP.Title.Text = "Espectro de Pulsos en adquisición";
    //PaneAAP.Title.FontSpec.Family =
```

```
PaneAAP.XAxis.Title.Text = "Número de Canal";
PaneAAP.YAxis.Title.Text = "Cuentas de Canal";
PaneAAP.XAxis.MajorGrid.IsVisible = true;
PaneAAP.YAxis.MajorGrid.IsVisible = true;
PaneAAP.XAxis.MajorGrid.Color = Color.DarkGray;
PaneAAP.YAxis.MajorGrid.Color = Color.DarkGray;
PaneAAP.XAxis.MinorGrid.IsVisible = true;
PaneAAP.YAxis.MinorGrid.IsVisible = true;
PaneAAP.XAxis.MinorGrid.Color = Color.LightGray;
PaneAAP.YAxis.MinorGrid.Color = Color.LightGray;
//PaneAAP.Legend.Position = ZedGraph.LegendPos.Bottom;
PaneAAP.XAxis.Scale.MaxGrace = 0.01;
PaneAAP.XAxis.Scale.MinGrace = 0.01;
PaneAAP.YAxis.Scale.MaxGrace = 0.05;
PaneAAP.YAxis.Scale.MinGrace = 0.05;
// Make up some data arrays based on the Sine function
int x, y1;
int i = 1;
string cuenta;
string archivoAAPdraw = "currentAAP_Dev" + amcID.ToString() + ".data";
string aaplotfile_current = archivoAAPdraw + "plot";
File.Copy(archivoAAPdraw, aaplotfile_current, true);
StreamReader archivoActualAAP = new StreamReader(aaplotfile_current);
PointPairList list1 = new PointPairList();

while ((cuenta = archivoActualAAP.ReadLine()) != null)
{
    x = i;
    y1 = int.Parse(cuenta);
    list1.Add(x, y1);
    y1++;
    i++;
}

archivoActualAAP.Close();
// Agregar un handler para cuando se detecte que el archivo cambia, y asi ejecutar de nuevo
// la generacion de la gráfica.
// Generar la curvaa
PaneAAP.CurveList.Clear();

LineItem myCurve = PaneAAP.AddCurve("Espectro", list1, Color.DarkBlue, SymbolType.Diamond);
```

```
    myCurve.Symbol.Size = 3.0F;
    myCurve.Symbol.Fill = new Fill(Color.DarkRed);
    myCurve.Line.IsVisible = true;
    myCurve.Label.IsVisible = false;
    // Tell ZedGraph to refigure the
    // axes since the data have changed
    graphMainAAP.AxisChange();
    graphMainAAP.Invalidate();
}

private void CreateGraphAMC(ZedGraphControl graphMainAMC)
{
    // get a reference to the GraphPane
    GraphPane PaneAMC = graphMainAMC.GraphPane;
    // Titulos y ejes
    PaneAMC.Title.Text = "Espectro Mossbauer en adquisición";
    //PaneAAP.Title.FontSpec.Family =
    PaneAMC.XAxis.Title.Text = "Número de Canal";
    PaneAMC.YAxis.Title.Text = "Cuentas de Canal";
    PaneAMC.XAxis.MajorGrid.IsVisible = true;
    PaneAMC.YAxis.MajorGrid.IsVisible = true;
    PaneAMC.XAxis.MajorGrid.Color = Color.DarkGray;
    PaneAMC.YAxis.MajorGrid.Color = Color.DarkGray;
    PaneAMC.XAxis.MinorGrid.IsVisible = true;
    PaneAMC.YAxis.MinorGrid.IsVisible = true;
    PaneAMC.XAxis.MinorGrid.Color = Color.LightGray;
    PaneAMC.YAxis.MinorGrid.Color = Color.LightGray;
    //PaneAMC.Legend.Position = ZedGraph.LegendPos.Bottom;
    PaneAMC.Legend.IsVisible = false;
    PaneAMC.XAxis.Scale.MaxGrace = 0.01;
    PaneAMC.XAxis.Scale.MinGrace = 0.01;
    PaneAMC.YAxis.Scale.MaxGrace = 0.05;
    PaneAMC.YAxis.Scale.MinGrace = 0.05;
    PaneAMC.XAxis.Scale.Max = 512;

    int x, y1;
    int i = 1;
    string cuenta;
    string archivoAMCdraw = "currentAMC_Dev" + amcID.ToString() + ".data";
    string amcplotfile_current = archivoAMCdraw + "plot";
```

```
    File.Copy(archivoAMCdraw, amcplotfile_current, true);
    StreamReader archivoActualAMC = new StreamReader(amcplotfile_current);
    PointPairList list1 = new PointPairList();
    while ((cuenta = archivoActualAMC.ReadLine()) != null)
    {
        x = i;
        y1 = int.Parse(cuenta);
        list1.Add(x, y1);
        y1++;
        i++;
    }

    archivoActualAMC.Close();
    // Agregar un handler para cuando se detecte que el archivo cambia, y asi ejecutar de nuevo
    // la generacion de la gráfica.
    // Generar la curva
    PaneAMC.CurveList.Clear();
    var vec_list = list1.Select(pointPair => pointPair.Y).ToArray();
    LineItem myCurve = PaneAMC.AddCurve("Espectro", list1, Color.DarkBlue, SymbolType.Diamond);
    myCurve.Symbol.Size = 1.0F;
    myCurve.Symbol.Fill = new Fill(Color.DarkRed);
    myCurve.Line.IsVisible = false;
    myCurve.Line.Fill = new Fill(Color.Blue, Color.White, 45F);
    myCurve.Label.IsVisible = false;
    PaneAMC.YAxis.Scale.Max = 2*vec_list[10];
    // Tell ZedGraph to refigure the
    // axes since the data have changed
    graphMainAMC.AxisChange();
    graphMainAMC.Invalidate();
}


private void FormMain_Load(object sender, EventArgs e)
{
    InicializarPuerto(serialPortMain);
    //MenuModoAAP.Enabled = false;
    //  MenuModoAMC.Enabled = false;
}

private void radioButtonControlIntAAPAct_CheckedChanged(object sender, EventArgs e)
{
```

```
    string activarMedAAP = "CPA";
    EnviarOrden(activarMedAAP);

}

private void radioButtonControlIntAAPDesact_CheckedChanged(object sender, EventArgs e)
{
    string desactivarMedAAP = "CPD";
    EnviarOrden(desactivarMedAAP);
}

private void radioButtonControlIntAct_CheckedChanged(object sender, EventArgs e)
{
    if (radioButtonControlIntAct.Checked == true)
    {
        radioButtonControlIntAct.Enabled = false;
        radioButtonControlIntDesact.Enabled = true;
        InicializarPuerto(serialPortMain);
        byte[] control_interrumpir = new byte[3] { 0x43, 0x4D, 0x41 };
        string inicomando = "[";
        serialPortMain.Write(inicomando);
        for (int i = 0; i < control_interrumpir.Length; i++)
        {
            byte[] chain = new byte[1] { control_interrumpir[i] };
            serialPortMain.Write(chain, 0, chain.Length);
            System.Threading.Thread.Sleep(50);

        }
        string fincr = "\r";
        serialPortMain.Write(fincr);

    }
    else return;
}

private void radioButtonControlIntDesact_CheckedChanged(object sender, EventArgs e)
{
    if(radioButtonControlIntDesact.Checked == true)
    {
        radioButtonControlIntAct.Enabled = true;
        radioButtonControlIntDesact.Enabled = false;
```

```
        InicializarPuerto(serialPortMain);
        byte[] control_interrumpir = new byte[3] { 0x43, 0x4D, 0x44 };
        string inicomando = "[";
        serialPortMain.Write(inicomando);
        for (int i = 0; i < control_interrumpir.Length; i++)
        {
            byte[] chain = new byte[1] { control_interrumpir[i] };
            serialPortMain.Write(chain, 0, chain.Length);
            System.Threading.Thread.Sleep(50);

        }
        string fincr = "\r";
        serialPortMain.Write(fincr);

    }
    else return;
}

private void guardarComoToolStripMenuItem_Click(object sender, EventArgs e)
{
    string datafile;
    SaveFileDialog guardarEspecDialog = new SaveFileDialog();
    if (MenuModoAAP.Checked == true && MenuModoAMC.Checked == false)
    {
        guardarEspecDialog.Filter = "Archivo de altura de pulsos|*.pha";
        guardarEspecDialog.Title = "Guardar archivo de altura de pulsos";
        datafile = @"currentAAP_Dev" + amcID.ToString() + ".dataplot";
    }
    else
    {
        if (MenuModoAMC.Checked == true && MenuModoAAP.Checked == false)
        {
            guardarEspecDialog.Filter = "Archivo de espectro Mössbauer|*.dat|Todos los archivos
                |*.txt";
            guardarEspecDialog.Title = "Guardar archivo de espectro Mössbauer";
            datafile = @"currentAMC_Dev" + amcID.ToString() + ".dataplot";
        }
        else
        {
            MessageBox.Show("No se detectó el modo de operación para guardar archivo");
            return;
```

```
        }
    }

    guardarEspecDialog.ShowDialog();

    // If the file name is not an empty string open it for saving.
    if (guardarEspecDialog.FileName != "")
    {
        using (FileStream fsout = (FileStream)guardarEspecDialog.OpenFile(), input =
             File.OpenRead(datafile))
        {

        switch (guardarEspecDialog.FilterIndex)
        {
            case 1:
                int read = -1;
                byte[] buffer = new byte[4096];
                while (read != 0)
                {
                    read = input.Read(buffer, 0, buffer.Length);
                    fsout.Write(buffer, 0, read);
                }
            break;

            case 2:
                read = -1;
                buffer = new byte[4096];
                while (read != 0)
                {
                    read = input.Read(buffer, 0, buffer.Length);
                    fsout.Write(buffer, 0, read);
                }
                break;
        }

        fsout.Close();
        }
    }


}
```

```
        private void FormMain_FormClosed(object sender, FormClosedEventArgs e)
        {
            Application.Exit();
        }
        System.Windows.Forms.Timer tmrref;

        private void checkBoxRefreshAMC_CheckedChanged(object sender, EventArgs e)
        {
            if (checkBoxRefreshAMC.Checked == true)
            {

                tmrref = new System.Windows.Forms.Timer();
                //set time interval 3 sec
                tmrref.Interval = 1000*Convert.ToInt32(numericUpDownAMCRate.Value.ToString());
                //starts the timer
                tmrref.Start();
                tmrref.Tick += tmr_Tick;
                buttonLeerEspecAMC.Enabled = false;
            }
            else buttonLeerEspecAMC.Enabled = true;
        }

        void tmr_Tick(object sender, EventArgs e)
        {
            //after 3 sec stop the timer
            //tmrref.Stop();
            //display mainform
            instanciarPuertoSerial();

        }


    }
}
```

A.1.2. Funciones de comunicación. FormCOMOptions.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.Configuration;
using SerialControlAMCMB96.Properties;
using System.IO;
using System.IO.Ports;

namespace SerialControlAMCMB96
{
    public partial class FormCOMOptions : Form
    {
        string sBuffer = String.Empty;
        List<byte> bBuffer = new List<byte>();
        string RxString;
        string ErrorNoCOM = "Imposible abrir puerto de comunicación";
        public FormCOMOptions()
        {
            InitializeComponent();
            comboBoxPortName.Items.Clear();
            foreach (string s in System.IO.Ports.SerialPort.GetPortNames())
            {
                comboBoxPortName.Items.Add(s);
            }

        //Establecimiento de valores por defecto para la inicialización del puerto

        comboBoxPortName.Text = Properties.Settings.Default.COMPortName;
        comboBoxBaudrate.Text = Convert.ToString(Properties.Settings.Default.COMBaudRate);
        comboBoxParity.SelectedIndex = 0;
        comboBoxStopBits.SelectedIndex = 0;
        comboBoxDataBits.Text = Convert.ToString(Properties.Settings.Default.COMDataBits);
        comboBoxHandshake.SelectedIndex = 3;
        textBoxReadTimeout.Text = Convert.ToString(Properties.Settings.Default.COMReadTimeout);
        textBoxWriteTimeout.Text = Convert.ToString(Properties.Settings.Default.COMWriteTimeout);
        comboBoxPortName.Enabled = true;
        comboBoxBaudrate.Enabled = false;
```

```
comboBoxParity.Enabled = false;
comboBoxStopBits.Enabled = false;
comboBoxDataBits.Enabled = false;
comboBoxHandshake.Enabled = true;
textBoxReadTimeout.Enabled = false;
textBoxWriteTimeout.Enabled = false;
}


private void buttonIniciar_Click(object sender, EventArgs e)
{
    serialPort1.PortName = comboBoxPortName.Text;
    serialPort1.BaudRate = int.Parse(comboBoxBaudrate.Text);
    if (comboBoxParity.SelectedIndex == -1)
    {
        MessageBox.Show("Paridad Inválida");
        return;
    }
    if (comboBoxParity.SelectedIndex == 0)
    {
        serialPort1.Parity = System.IO.Ports.Parity.None;
    }
    if (comboBoxParity.SelectedIndex == 1)
    {
        serialPort1.Parity = System.IO.Ports.Parity.Odd;
    }
    if (comboBoxParity.SelectedIndex == 2)
    {
        serialPort1.Parity = System.IO.Ports.Parity.Even;
    }
    if (comboBoxParity.SelectedIndex == 3)
    {
        serialPort1.Parity = System.IO.Ports.Parity.Mark;
    }
    if (comboBoxParity.SelectedIndex == 4)
    {
        serialPort1.Parity = System.IO.Ports.Parity.Space;
    }

    serialPort1.DataBits = int.Parse(comboBoxDataBits.Text);
```

```
if (comboBoxStopBits.SelectedIndex == -1)
{
    MessageBox.Show("Imposible definir Bits de parada");
    return;
}
if (comboBoxStopBits.SelectedIndex == 1)
{
    serialPort1.StopBits = System.IO.Ports.StopBits.One;
}
if (comboBoxStopBits.SelectedIndex == 2)
{
    serialPort1.StopBits = System.IO.Ports.StopBits.None;
}
if (comboBoxStopBits.SelectedIndex == 3)
{
    serialPort1.StopBits = System.IO.Ports.StopBits.Two;
}

if (comboBoxHandshake.SelectedIndex == -1)
{
    MessageBox.Show("Imposible definir modo de negociación");
    return;
}
if (comboBoxHandshake.SelectedIndex == 1)
{
    serialPort1.Handshake = System.IO.Ports.Handshake.None;
}
if (comboBoxHandshake.SelectedIndex == 2)
{
    serialPort1.Handshake = System.IO.Ports.Handshake.XOnXOff;
}
if (comboBoxHandshake.SelectedIndex == 3)
{
    serialPort1.Handshake = System.IO.Ports.Handshake.RequestToSend;
}
if (comboBoxHandshake.SelectedIndex == 4)
{
    serialPort1.Handshake = System.IO.Ports.Handshake.RequestToSendXOnXOff;
}

serialPort1.ReadTimeout = int.Parse(textBoxReadTimeout.Text);
```

```
    serialPort1.WriteTimeout = int.Parse(textBoxWriteTimeout.Text);

    textBox1.AppendText("Abriendo Puerto COM" + Environment.NewLine);
    textBox1.AppendText("PortName Puerto COM:" + serialPort1.PortName + Environment.NewLine);
    textBox1.AppendText("BaudRate Puerto COM:" + serialPort1.BaudRate + Environment.NewLine);
    textBox1.AppendText("Parity Puerto COM:" + serialPort1.Parity + Environment.NewLine);
    textBox1.AppendText("DataBits Puerto COM:" + serialPort1.DataBits + Environment.NewLine);
    textBox1.AppendText("StopBits Puerto COM:" + serialPort1.StopBits + Environment.NewLine);
    textBox1.AppendText("Handshake Puerto COM:" + serialPort1.Handshake + Environment.NewLine);
    textBox1.AppendText("ReadTimeOut Puerto COM:" + serialPort1.ReadTimeout +
         Environment.NewLine);
    textBox1.AppendText("WriteTimeOut Puerto COM:" + serialPort1.WriteTimeout +
         Environment.NewLine);

    try
    {
        serialPort1.Open();
        //serialPort1.RtsEnable = true;
        serialPort1.DtrEnable = true;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }

    if (serialPort1.IsOpen)
    {
        buttonIniciar.Enabled = false;
        buttonStop.Enabled = true;
        textBox1.ReadOnly = false;
    }
    else
    {
        textBox1.AppendText(ErrorNoCOM);
    }
}

private void buttonStop_Click(object sender, EventArgs e)
{
    if (serialPort1.IsOpen)
```

```
        {
            serialPort1.Close();
            buttonIniciar.Enabled = true;
            buttonStop.Enabled = false;
            textBox1.AppendText("Cerrando puerto COM" + Environment.NewLine);
            textBox1.ReadOnly = true;
        }
    }

    public void serialPort1_DataReceived(object sender, SerialDataReceivedEventArgs e)
    {
        int numdatosEntrada;
        numdatosEntrada = serialPort1.BytesToRead;
        Byte[] dato = new Byte[numdatosEntrada];
        serialPort1.Read(dato, 0, numdatosEntrada);
        RxString = BitConverter.ToString(dato).Replace("-", " ");
        this.Invoke(new EventHandler(DisplayText));
    }


    private void ProcessBuffer(List<byte> bBuffer, int len)
    {
        // Create a byte array buffer to hold the incoming data
        byte[] buffer = bBuffer.ToArray();

        // Show the user the incoming data // Display mode
        for (int i = 0; i < buffer.Length; i++)
        {
            textBox1.AppendText("Dato: " + (bBuffer[i].ToString()));
        }
/*          string Data1 = "";
        string sData = "";
        int i = 0;
        while (i < len)
        {
            //Data1 = String.Format("{0:X}", buf[i++]); //no joy, doesn't pad
            Data1 = bBuffer[i++].ToString("X").PadLeft(2, '0'); //same as "%02X" in C
            sData += Data1;

        }
        return sData;
```

```
            // Look in the byte array for useful information
*/          // then remove the useful data from the buffer
        }

        private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
            // If the port is closed, don't try to send a character.

            if (!serialPort1.IsOpen) return;

            // If the port is Open, declare a char[] array with one element.
            char[] buff = new char[1];

            // Load element 0 with the key character.

            buff[0] = e.KeyChar;

            // Send the one character buffer.
            serialPort1.Write(buff, 0, 1);
            serialPort1.DataReceived +=new SerialDataReceivedEventHandler(serialPort1_DataReceived);

            // Set the KeyPress event as handled so the character won't
            // display locally. If you want it to display, omit the next line.
            //e.Handled = true;
        }

        private void DisplayText(object sender, EventArgs e)
        {
            textBox1.AppendText("\n" + RxString);
        }

        private void FormCOMOptions_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (serialPort1.IsOpen) serialPort1.Close();
        }

        private void textBox1_TextChanged(object sender, EventArgs e)
        {

        }
```

```
        private void splitContainer1_Panel1_Paint(object sender, PaintEventArgs e)
        {

        }

        private void buttonSaveConf_Click(object sender, EventArgs e)
        {
            Properties.Settings.Default.COMPortName = comboBoxPortName.Text;
            Properties.Settings.Default.COMBaudRate = int.Parse(comboBoxBaudrate.Text);
            if (comboBoxParity.SelectedIndex == 0)
            {
                Properties.Settings.Default.COMParity = System.IO.Ports.Parity.None;
            }
            if (comboBoxParity.SelectedIndex == 1)
            {
                Properties.Settings.Default.COMParity = System.IO.Ports.Parity.Odd;
            }
            if (comboBoxParity.SelectedIndex == 2)
            {
                Properties.Settings.Default.COMParity = System.IO.Ports.Parity.Even;
            }
            if (comboBoxParity.SelectedIndex == 3)
            {
                Properties.Settings.Default.COMParity = System.IO.Ports.Parity.Mark;
            }
            if (comboBoxParity.SelectedIndex == 4)
            {
                Properties.Settings.Default.COMParity = System.IO.Ports.Parity.Space;
            }
            Properties.Settings.Default.COMDataBits = int.Parse(comboBoxDataBits.Text);
            Properties.Settings.Default.COMStopBits = serialPort1.StopBits;
            Properties.Settings.Default.COMHandshake = serialPort1.Handshake;
            Properties.Settings.Default.COMReadTimeout = int.Parse(textBoxReadTimeout.Text);
            Properties.Settings.Default.COMWriteTimeout = int.Parse(textBoxWriteTimeout.Text);
            Properties.Settings.Default.Save();
            this.Close();
        }

    }
}
```

A.1.3. Parámetros del modo AMC. FormParamAMC.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.Configuration;
using SerialControlAMCMB96.Properties;
using System.IO;
using System.IO.Ports;

namespace SerialControlAMCMB96
{
    public partial class FormParamAMC : Form
    {
        public FormParamAMC()
        {
            InitializeComponent();

            comboBoxModoManejo.SelectedIndex = 0;
            numericUpDownFrecuencia.DecimalPlaces = 1;
            comboBoxTiempoMuerto.SelectedIndex = 0;
            comboBoxResolucion.SelectedIndex = 0;
            comboBoxMaxVel.SelectedIndex = 0;
            comboBoxFactorEscala.SelectedIndex = 0;
            comboBoxSignoVelocidad.SelectedIndex = 0;
            comboBoxModoAvance.SelectedIndex = 0;
            numericUpDownFrecuencia.Value = 10.0M;

            //Properties.Settings.Default.ModoManejo = comboBoxModoManejo.SelectedText;
            /*Properties.Settings.Default.ModoManejo = 1;
            Properties.Settings.Default.FrecuenciaBarrido = 10;
            Properties.Settings.Default.TiempoMuerto = 20;
            Properties.Settings.Default.Resolucion = 256;
            Properties.Settings.Default.MaxVelocidad = 10;
            Properties.Settings.Default.FactorEscala = 1;
            //Properties.Settings.Default.SignoVelocidad = comboBoxSignoVelocidad.SelectedText;
```

```
            //Properties.Settings.Default.ModoAvance = comboBoxModoAvance.SelectedText;
            Properties.Settings.Default.SignoVelocidad = 0;
            Properties.Settings.Default.ModoAvance = Convert.ToChar("I");
            */
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void buttonSaveConf_Click(object sender, EventArgs e)
        {
            Properties.Settings.Default.ModoManejo = Convert.ToInt16(comboBoxModoManejo.SelectedValue);
            Properties.Settings.Default.FrecuenciaBarrido =
                Convert.ToDouble(numericUpDownFrecuencia.Value);
            Properties.Settings.Default.TiempoMuerto =
                Convert.ToInt16(comboBoxTiempoMuerto.SelectedValue);
            Properties.Settings.Default.Resolucion = Convert.ToInt16(comboBoxResolucion.SelectedValue);
            Properties.Settings.Default.MaxVelocidad = Convert.ToInt16(comboBoxMaxVel.SelectedValue);
            Properties.Settings.Default.FactorEscala =
                Convert.ToInt16(comboBoxFactorEscala.SelectedValue);
            Properties.Settings.Default.SignoVelocidad =
                Convert.ToInt16(comboBoxSignoVelocidad.SelectedValue);
            Properties.Settings.Default.ModoAvance = Convert.ToChar(comboBoxModoAvance.SelectedValue);
            Properties.Settings.Default.Save();
            this.Close();
        }




    }
}
```

A.1.4. Splash Screen. SplashScr.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;

namespace SerialControlAMCMB96
{
    public partial class SplashScr : Form
    {
        public SplashScr()
        {
            InitializeComponent();
        }
        Timer tmr;
        private void SplashScr_Shown(object sender, EventArgs e)
        {
            tmr = new Timer();
            //set time interval 3 sec
            tmr.Interval = 5000;
            //starts the timer
            tmr.Start();
            tmr.Tick += tmr_Tick;
        }

        void tmr_Tick(object sender, EventArgs e)
        {
            //after 3 sec stop the timer
            tmr.Stop();
            //display mainform
            FormMain mf = new FormMain();
            mf.Show();
            //hide this form
            this.Hide();
        }


    }
}
```

Bibliografía

[1] A.J. Sanchez [et al.], *Hyperfine Interact.* **110**, 135-141 (1997)

[2] A.J. Sanchez, *Diseño y construcción de un analizador multicanal Mössbauer y su aplicación al estudio de sistemas metálicos*, PhD. Phys. Thesis, Universidad del Valle, Departamento de Física, 1997.

[3] F. Varret, J. Teillet, **MOSFIT** program, no publicado.

[4] J.D. Betancur, *Manual de ajuste de espectros Mössbauer con el programa MOSFIT*, no publicado, Universidad del Valle, 2007.

[5] A.A. Velásquez [et al.], *Design and Construction of an Autonomous Low-Cost Pulse Height Analyzer and a Single Channel Analyzer for Mössbauer Spectroscopy*, Proceedings of PAM 2003, La Jolla, 2003.

[6] M. Cardoso [et al.] *A Very Low-Cost Portable Multichannel Analyzer*, http://lei.fis.uc.pt/pdfs/Com002.pdf