UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

AMPLIACIÓN DE UN SIMULADOR DE TARJETA DE ADQUISICIÓN DE DATOS EN C

Proyecto Final de Carrera

Ingeniería Técnica en Informática de Sistemas

Autora: Rosa Mª Maciá Sempere

Tutor: Antonio Martí Campoy

Septiembre 2015

Resumen

El proyecto que se presenta tiene como punto de origen el realizado en junio de 2007 por Miguel Carro Pellicer, con título "PFC II-A-DISCA- 55/06: SIMULADOR DE TARJETA DE ADQUISICIÓN DE DATOS 'NuDAQ / NuIPC 9112 Series' ".

Con la simulación de esta tarjeta, también llamada tarjeta virtual, los alumnos de prácticas de la asignatura "Informatización Industrial" poseen una herramienta software con la cual realizar las prácticas sin necesidad de disponer de la tarjeta física.

Este nuevo proyecto pretende ampliar el abanico de funcionalidades de la tarjeta que se simulan actualmente. En la medida de lo posible, se intentará ser mínimamente intrusivo con el software original.

De entre las prestaciones de las que dispone la tarjeta emulada, se implementarán dos nuevos grupos de funciones: el Timer 8253 y la adquisición de datos analógicos en modo ráfaga.

En el caso del Timer 8253, se intentará realizar una simulación lo más cercana posible a la realidad, sin olvidar que se trata de una simulación software y que la inclusión de esta funcionalidad tiene un fin pedagógico.

La incorporación de funciones de adquisición de datos en ráfaga se debe a que actualmente, con la tarjeta virtual original, sólo es posible adquirir muestras de una en una. Se implementarán funciones con las cuales se puedan realizar n lecturas y registrarlas en un buffer de datos para su posterior uso.

Con el fin de asegurar la compatibilidad del software de virtualización de la tarjeta con los nuevos sistemas operativos Windows, se realizarán pruebas que verifiquen el correcto funcionamiento del mismo. Dichas pruebas se harán tanto del software ya disponible como del generado durante el desarrollo de este proyecto.

Tabla de contenidos

1 - Introducción

En junio de 2007, Miguel Carro Pellicer presentó su proyecto final de carrera “Simulador de tarjeta de adquisición de Datos ‘NuDAQ / NuIPC 9112 Series’ ”.

El objetivo de dicho proyecto era crear un software con el cual no fuera necesario el uso del hardware que se utilizaba hasta el momento en las prácticas de la asignatura "Informatización Industrial" (perteneciente al cuarto curso de la titulación "Ingeniería Industrial" de la Escuela Técnica Superior de Ingenieros Industriales (ETSII)) de la Universitat Politècnica de València.

Dicho hardware constaba de la ya mencionada tarjeta de adquisición de datos, más un emulador hardware de procesos conectado a ésta.

Tras la implementación del software, el alumnado ha podido continuar realizando las prácticas que hacían uso de la tarjeta física de forma equivalente (tareas de configuración y obtención de datos, con el uso de la librería de ésta), sin necesidad de disponer del costoso hardware y en cualquier PC: simplemente disponiendo del software y la librería suministrados para la realización de dichas prácticas.

Los elementos que conforman el sistema de herramientas virtuales son los siguientes:

- Una tarjeta virtual de adquisición de datos.
- Un simulador software de procesos (proceso virtual), herramienta con la cual generar entradas para la tarjeta virtual, y obtener las salidas provenientes de la tarjeta virtual.
- Librería de la tarjeta, modificada para su uso con la tarjeta virtual.
- Programa de prácticas del alumno.

Las funciones en la librería de la tarjeta virtual están implementadas de forma que se comuniquen con procesos virtuales en lugar de hacerlo con el hardware. Simulan el comportamiento de las funciones originales de la tarjeta y son utilizadas del mismo modo: el alumno realiza llamadas a funciones de mismo nombre que las reales y los resultados obtenidos son los que se recibirían del proceso hardware, pero en este caso se reciben del proceso virtual. Para poder acceder a dicho proceso, desde la librería se accede a la tarjeta virtual y ésta última es quien se comunica con él.

El esquema del sistema simulador original quedaría de la siguiente forma:

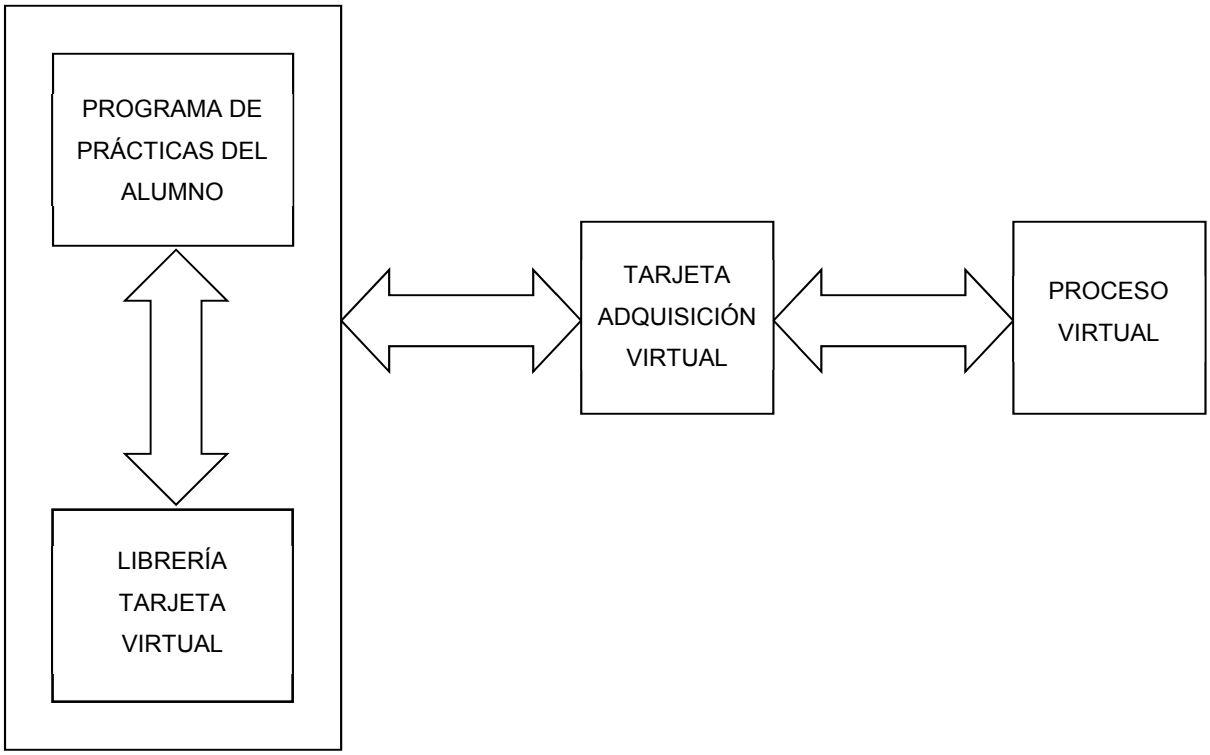

Imagen 1 – Simulador original

Este nuevo proyecto, que en este informe se describe, tiene como fin la ampliación de la colección de funcionalidades ya implementadas anteriormente en el proyecto origen, y disponer así de nuevas prestaciones de las que posee la original tarjeta 'NuDAQ / NuIPC 9112 Series'.

Dentro de estas nuevas funcionalidades se distinguen dos grupos:

- Funciones de Timer 8253: simularán el uso de otro elemento hardware de la tarjeta, como es el chip temporizador 8253, que consta de tres contadores descendentes e independientes de 16 bits y con una frecuencia de 2MHz.

- Funciones de captura de datos.

Con el fin de no ser muy intrusivo con el software ya existente, se han tomado las decisiones oportunas para no modificarlo, por lo que todas las ampliaciones han sido incluidas, o bien dentro de la librería ya existente, o creando una nueva aplicación (en el caso concreto del Timer 8253).

2 - Objetivos

Este proyecto tiene como primer objetivo añadir nuevas funcionalidades a las ya existentes en el simulador de la tarjeta de adquisición de datos, con el fin de que el alumnado pueda experimentar con otras prestaciones que el hardware posee.

Los cambios a realizar no deberán afectar al funcionamiento ni de la tarjeta virtual ni del simulador y serán compatibles con el programa de prácticas del alumnado, no siendo necesario un cambio del compilador utilizado, ni el uso de un PC concreto.

Dado el tiempo transcurrido desde la creación del software original, otro objetivo a llevar a cabo en este proyecto será verificar su compatibilidad con las nuevas versiones de sistemas operativos Windows. Para ello se realizarán pruebas de todo el conjunto de funcionalidades, aportadas tanto por el software original como el nuevo, en máquinas con Windows 7 y con Windows 8.

En resumen, los objetivos a abordar en este proyecto son los siguientes:

1 - Implementación de nuevas funcionalidades de la tarjeta NuDAQ / NulPC 9112 Series:

- Simulación de Timer hardware:
  - Creación de una aplicación wxWidget con Code::Blocks que simule el Timer.
  - Implementación de funciones de librería para el uso del Timer: Start, Status y Stop.
- Simulación de captura de datos A/D:
  - Ampliación de la librería de tarjeta para realizar captura continua de N muestras: Start, Status y Stop.

En ninguno de los casos es posible simular por software la frecuencia de reloj con la que trabaja el hardware original (2MHz) y dada la finalidad para la cual será creado este software, tampoco sería de utilidad dicha precisión. Se hará uso del reloj en tiempo real del ordenador y de un divisor de frecuencia para simular el comportamiento del temporizador de la tarjeta.

2 - Pruebas de compatibilidad del conjunto software en nuevos sistemas Windows:

- Compatibilidad con Windows 7.
- Compatibilidad con Windows 8.
- Cambios asociados a las pruebas: comprobación de aplicaciones ya lanzadas previa a una ejecución. Mejoras.

La incorporación de la nueva aplicación de Timer, añadida al sistema virtual existente quedaría de la siguiente forma:

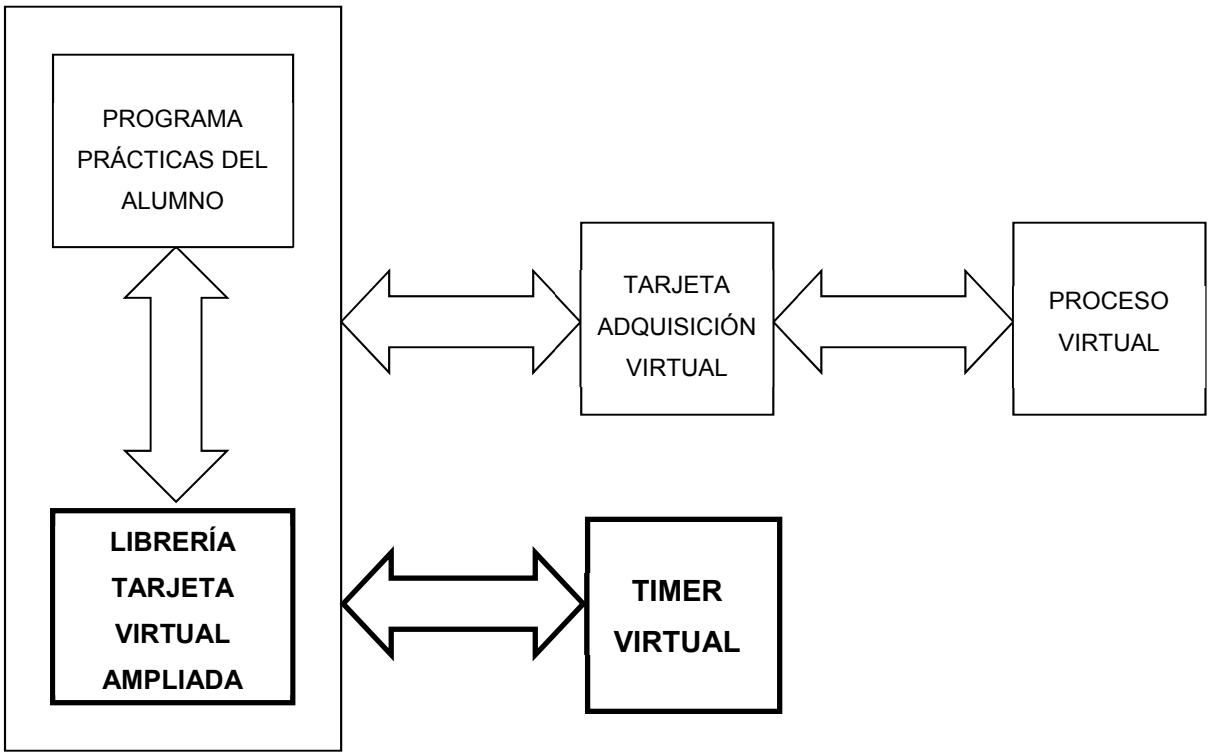

Imagen 2 – Simulador con incorporación de Timer Virtual

3 – Análisis de requisitos

Se definirán los requisitos para cada una de las partes en las que se diferencia el proyecto:

3.1 TIMER VIRTUAL

El timer virtual será una aplicación, ejecutada aparte de las ya existentes, con la que se podrá interactuar desde la librería virtual. Para no modificar el código del simulador virtual existente, esta aplicación será lanzada desde la librería, siendo también ésta quien se comunique con el Timer. Tras su ejecución, la aplicación devolverá, tanto en la consulta del estado del contador como en la solicitud de parada de éste, el valor del contador. En la librería virtual se añadirán las siguientes funciones correspondientes al uso del Timer. Los nombres de las funciones son los originales que aparecen en la librería de la tarjeta hardware. De esta forma se simplifica la adaptación del programa del alumno entre el uso de la tarjeta virtual y la real:

I16 _9112_TIMER_Start (U16 CardNumber, int timer_mode, int c0)

Con la llamada a esta función el usuario da la orden de comenzar la cuenta, a partir de los datos de inicio enviados como argumentos: número de tarjeta, modo de timer y valor de contador inicial. Desde la librería se avisará a la aplicación de Timer, que comenzará la cuenta.

Debido a que no está implementada la gestión de interrupciones en la simulación de la tarjeta, sólo se implementará el Timer mode = 0. En este modo se realizará una cuenta descendente desde el valor inicial aportado hasta que el valor de contador sea igual a cero.

I16 _9112_TIMER_Read (U16 CardNumber, int *counter_value)

El usuario obtendrá el valor del contador en el momento de la consulta. La función recibirá como parámetros el número de tarjeta y una variable donde se devolverá el valor de contador. La librería remitirá la consulta a la aplicación Timer, que devolverá el valor inicial de contador menos el número de cuentas ya trascurridas, o cero si la cuenta hubiera llegado a este valor.

I16 _9112_TIMER_Stop (U16 CardNumber, int *counter_value)

Con esta orden el usuario podrá detener el proceso de timer y recibir el valor de contador en ese instante. La función recibe como parámetros el número de tarjeta y una variable donde se devolverá el valor de contador. Tras la recepción de la orden solicitada, la librería enviará el aviso de parada la cuenta y recibirá el valor de contador en dicho momento.

3.2 ADQUISICIÓN DE DATOS

La simulación de adquisición de datos en ráfaga será una colección de funciones de la librería virtual, que interactuarán con la tarjeta virtual.

Para su implementación también se seguirán estrictamente las especificaciones de la librería del fabricante.

Las nuevas funciones relacionadas con la captura de datos son las siguientes:

I16 _9112_AD_INT_Sart (U16 CardNumber, int auto_scan, int ad_ch_no, int ad_range,
int counter, unsigned long *ad_buffer, int c1, int c2);

Con la llamada a esta función el usuario puede obtener una colección de datos, como resultado de repetidas lecturas de las entradas analógicas. Con esta función se comenzará la captura de datos.

La función recibe como parámetros el número de tarjeta, el valor de autoscan, el número de canal de donde se obtendrán los datos, el rango, el contador, un buffer de datos donde se irán registrando los datos leídos y los valores de los divisores de frecuencia con los que se calculará el intervalo entre lecturas.

Desde la librería se lanzará un proceso en segundo plano (llamado AD_INT_Background), en el que se irán realizando llamadas a la función original AI_ReadChannel de lectura de canal. Cada medida obtenida se registrará en el buffer de datos para su posterior consulta.

I16 _9112_AD_INT_Status (U16 CardNumber, int *status, int *counter);

Con la llamada a esta función el usuario puede consultar si ha acabado por completo el proceso de adquisición de datos y, sea cual sea el resultado de dicha consulta, podrá consultar los datos registrados hasta ese momento, accediendo a la dirección de memoria del buffer, devuelta en la función Start.

La función recibirá como parámetros el número de tarjeta y dos variables donde se registrarán el estado del proceso y el número de registros capturados hasta ese momento.

I16 _9112_AD_INT_Stop (U16 CardNumber, int *counter);

Con la llamada a esta función el usuario ordena la finalización de la captura de datos, sea cual sea su estado y obtendrá los datos registrados hasta el momento de la consulta.

La función recibirá como parámetros el número de tarjeta y la variable donde se guardará el número de registros capturados. Al recibir la orden, la librería parará el proceso AD_INT_Background, de ese modo se parará el proceso de adquisición de datos, y accediendo a la dirección de memoria del buffer se podrá acceder a todos los registros almacenados.

4 – Diseño del proyecto

El lenguaje utilizado para desarrollar el proyecto ha sido C++, y se ha trabajado bajo el entorno de desarrollo Code::Blocks.

Con dicho entorno se han realizado las ampliaciones para la parte de la librería de la tarjeta virtual y para desarrollar la aplicación wxWidget que simula el Timer.

Como las partes implementadas no afectan al programa ya existente, que fue programado bajo el entorno de desarrollo Borland C++ Builder, se ha querido utilizar un entorno de desarrollo libre. Este es el motivo por el cual se ha elegido Code::Blocks, ya que se trata de un entorno integrado libre y licenciado bajo la Licencia pública general de GNU.

4.1 MECANISMOS DE INTERCOMUNICACIÓN DE PROCESOS

Debido a que este proyecto es una ampliación de otro ya existente, se ha continuado con la misma filosofía decidida por el antiguo proyectante (Miguel Carro Pellicer) a la hora de intercomunicar procesos:

Se utilizan tuberías con nombre (pipes) para las comunicaciones entre la librería de la tarjeta virtual y el Timer virtual. Para la sincronización de esta comunicación entre procesos se hará uso de eventos.

También se ha heredado del antiguo proyecto el protocolo de codificación de los mensajes de comunicación entre procesos: será una trama (de un máximo de veinte caracteres) que contendrá tres elementos separados por el carácter '#':

[CÓDIGO_OPERACIÓN # DIRECCIÓN o CANAL # VALOR]

A continuación se procederá a definir los diferentes recursos utilizados en este proyecto, que intervendrán en la intercomunicación de los procesos.

- TUBERÍAS:

Tubería "tubotimer"

Esta tubería será el medio de comunicación entre la librería y el timer virtual y será utilizado de modo bidireccional. La librería enviará por este medio las tramas con las órdenes a realizar por el timer, que serán esperadas por el hilo "Servidor de Timer". Tras la decodificación del mensaje recibido, se procederá a realizar la acción indicada en la trama.

Tubería "tubolib"

Esta tubería, heredada del anterior proyecto, será el medio de comunicación entre la tarjeta virtual y la librería y se utilizará de modo bidireccional. Las tramas serán esperadas por el hilo "Hilo Servidor de Librería" el cual, al recibir la trama, la decodificará y la tarjeta se encargará de realizar la operación que ésta indique.

- EVENTOS:

Evento "EventoTimer"

Este evento se utiliza por motivos de sincronización: cada operación de envío de una trama hacia el timer virtual será previamente notificada por la librería. El hilo "Servidor de Timer", que posteriormente se describirá, será el encargado de esperar este evento, quedando en suspensión mientras no se le notifique que va a recibir una trama a través de la tubería "tubotimer".

- HILOS:

Hilo de programación "Servidor de Timer"

Este hilo será el responsable de esperar continuamente tramas procedentes de la librería, a través de la tubería "tubotimer". El hilo quedará suspendido a la espera del evento "EventoTimer" y se activará cuando se vaya a recibir una trama. Tras recibirla, la aplicación decodificará el mensaje recibido y se procederá a realizar la operación de timer ordenada.

Hilo de programación "AD_INT_Background"

Con el fin de simular la captura de datos en ráfaga, y puesto que con el código original sólo se podía registrar una única lectura con la llamada a la función AI_ReadChanel, la librería lanzará este proceso en segundo plano, que será el encargado de realizar periódicamente consultas a la función original y registrando los resultados obtenidos. Con este proceso se simulará la captura de datos.

Hilo de programación "Hilo Servidor de Librería"

Este hilo, ya existente en el proyecto anterior, tiene como única función esperar tramas procedentes de la librería. Queda suspendido por el evento "EventoLibreria" y se activa cuando va a recibir una trama. Al recibirla, la tarjeta decodificará el mensaje y procederá a realizar las acciones solicitadas.

- MENSAJES:

Los mensajes de comunicación, como ya se ha explicado con anterioridad, constarán de tres elementos, separados por el carácter '#':

[CÓDIGO_OPERACIÓN # DIRECCIÓN o CANAL # VALOR]

A continuación se explicará cada uno de estos tres elementos, detallando a su vez los distintos valores que podrán tener:

- CÓDIGO_OPERACIÓN: Código que indica cuál de las funciones de la librería ha enviado la trama. Podrán recibirse los siguientes valores de código:

Código "DO"

Tubería: Tubolib.

Origen: Librería virtual.

Destino: Tarjeta virtual.

Descripción: Trama enviada por la función DO_WritePort indicándole a la tarjeta el nuevo valor de sus salidas digitales.

Efecto producido: Al recibir esta trama, la tarjeta actualiza sus registros internos y genera una trama DO con origen en la tarjeta virtual y destino en el proceso virtual, utilizando la tubería de nombre tubotest.

Código “AO”

Tubería: Tubolib.

Origen: Librería virtual.

Destino: Tarjeta virtual.

Descripción: Trama enviada por la función AO_WriteChannel indicándole a la tarjeta el nuevo valor de sus salidas analógicas.

Efecto producido: Al recibir esta trama, la tarjeta modificará el campo valor contenido en ella, para ajustarlo al formato requerido por el proceso virtual. Generará una trama AO con origen en la tarjeta virtual y destino en el proceso virtual, utilizando la tubería de nombre tubotest.

Código “DI”

Tubería: Tubolib.

Origen: Librería virtual.

Destino: Tarjeta virtual.

Descripción: Trama enviada por la función DI_ReadPort solicitándole a la tarjeta el valor de sus entradas digitales.

Efecto producido: Al recibir esta trama, la tarjeta genera una trama DI de respuesta, adjuntando el valor de las entradas digitales registradas en ese momento. El mensaje de respuesta volverá a la librería por la misma tubería tubolib.

Código “AI”

Tubería: Tubolib.

Origen: Librería virtual.

Destino: Tarjeta virtual.

Descripción: Trama enviada por la función AI_ReadChannel solicitándole a la tarjeta el valor de sus entradas analógicas.

Efecto producido: Al recibir esta trama, la tarjeta genera una trama AI de respuesta, adjuntando el valor de las entradas analógicas registradas en ese momento. La respuesta volverá a la librería por la misma tubería tubolib.

Código “XX”

Tubería: Tubotest2.

Origen: Proceso virtual.

Destino: Tarjeta virtual.

Descripción: Trama enviada por el proceso virtual con el valor de una entrada digital. Enviada si una entrada digital cambia su valor en el proceso virtual.

Efecto producido: La tarjeta virtual almacena el valor del dato recibido. No se genera ninguna trama.

Código “XZ”

Tubería: Tubotest2.

Origen: Proceso virtual.

Destino: Tarjeta virtual.

Descripción: Trama enviada por el proceso virtual con el valor analógico (número real) de una entrada analógica. Se envía cuando una entrada analógica cambia su valor en el proceso virtual.

Efecto producido: La tarjeta virtual convierte el valor analógico a entero simulando el convertidor ADC y lo almacena. No se genera ninguna trama.

Código “TS”

Tubería: Tubotimer.

Origen: Librería virtual.

Destino: Timer virtual.

Descripción: Trama enviada por la función _9112_TIMER_Start ordenándole al timer virtual el inicio del temporizador.

Efecto producido Al recibir esta trama, se extraerán de ella los valores de inicio de cuenta y se iniciará el temporizador.

Código “TR”

Tubería: Tubotimer.

Origen: Librería virtual.

Destino: Timer virtual.

Descripción: Trama enviada por la función _9112_TIMER_Read solicitándole al timer virtual el valor de actual de la cuenta.

Efecto producido: Al recibir esta trama, se calculará el valor de la cuenta y se generará una trama TR de respuesta, devuelta por la tubería tubotimer.

Código “TX”

Tubería: Tubotimer.

Origen: Librería virtual.

Destino: Timer virtual.

Descripción: Trama enviada por la función _9112_TIMER_Stop solicitándole al timer virtual la detención de la cuenta.

Efecto producido: Al recibir esta trama, se calculará el valor de la cuenta en ese instante, se inicializarán los valores de cuenta, a la espera de una nueva orden de inicio, y se generará una trama TR de respuesta, devuelta por la misma tubería tubotimer.

- DIRECCIÓN o CANAL: En las operaciones digitales servirá para indicar el número de puerto y en las operaciones analógicas el número de canal.

Debido a que sólo existe un timer virtual, en las tramas pertenecientes a funciones de timer este valor siempre será 0. Si en el futuro se simulara más de un timer, en ese caso tendría sentido la modificación de este valor.

- VALOR: Determina el campo de datos de la trama, asociado a la operación especificada por el código de operación. En todos los casos se tratará de valores de 4 bytes de tamaño.

Dependiendo del tipo de datos que se manejen, se representarán o como un número entero, en las tramas de actualización de datos digitales y de simulación de timer, o bien como un número en formato decimal para la actualización de datos analógicos (también será un número entero en el caso de las tramas de código "AI" (lectura de canal), donde el valor será convertido posteriormente a decimal).

Este tipo de tramas son sencillas de codificar y decodificar, debido a que el separador determina cada uno de los datos que viajan en ellas.

El esquema de la situación, tras la incorporación de la nueva aplicación de Timer añadida, quedaría de la siguiente forma:

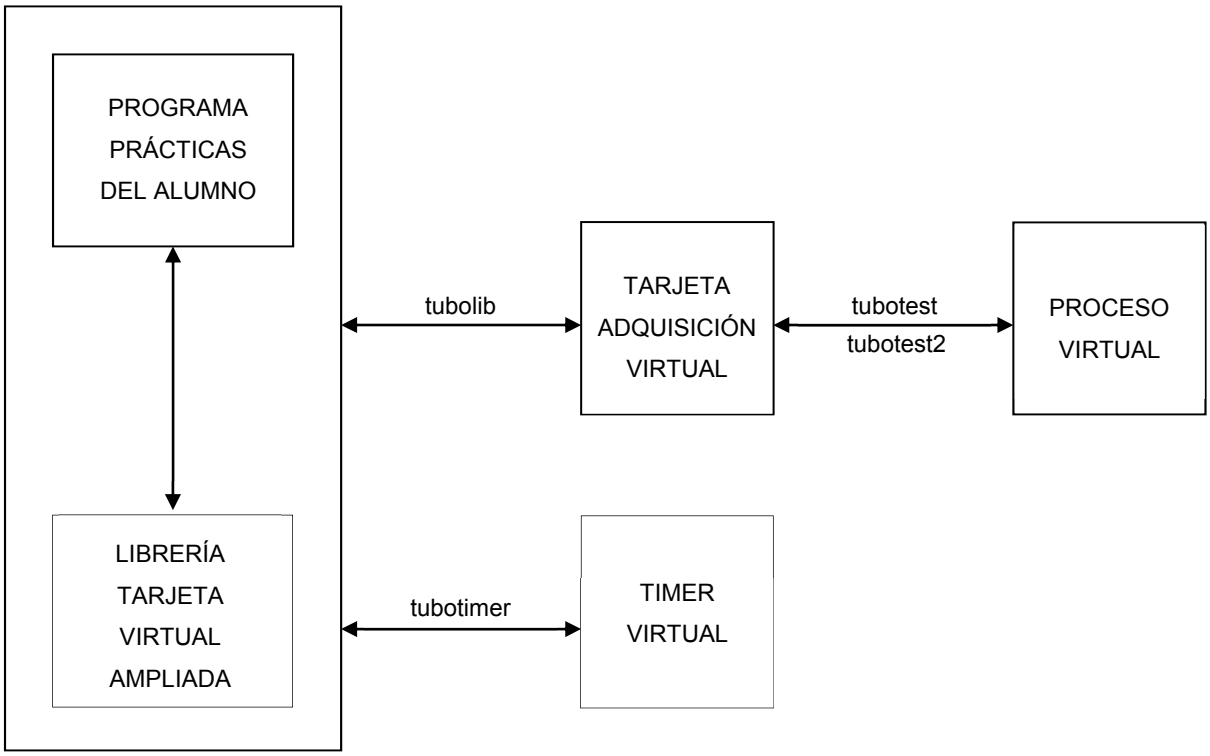

Imagen 3 – Incorporación de nuevo Timer Virtual

La estructura final, añadiendo los hilos de programación y los eventos quedará de la siguiente forma:

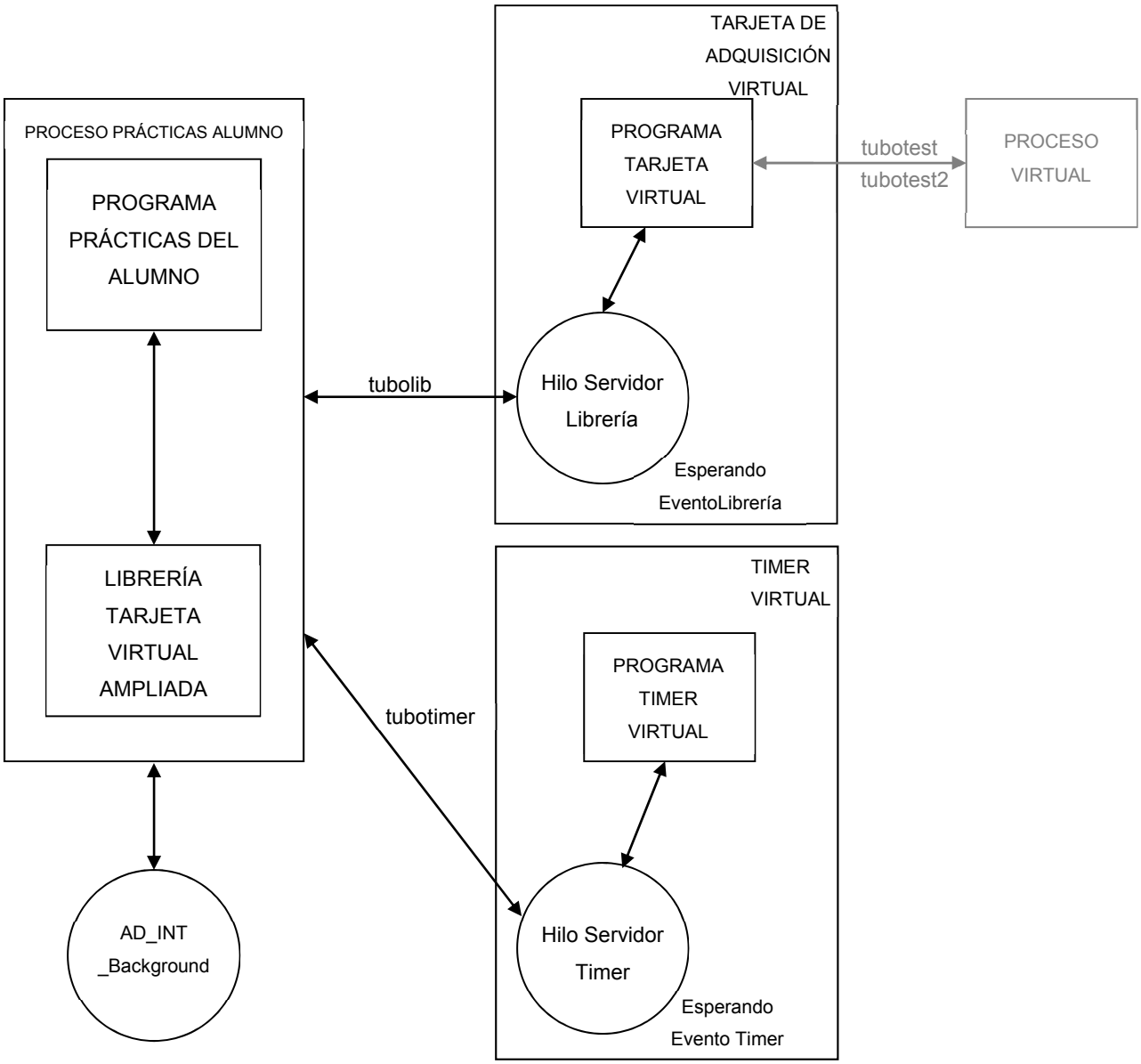

Imagen 4 – Estructura final con hilos y eventos

4.2 CAPA LÓGICA DE LA APLICACIÓN

Dentro de este apartado se detallará el planteamiento empleado en la lógica del programa, de su estructura y de los algoritmos de los protocolos utilizados. También se especificarán las variables necesarias para el almacenamiento de datos.

A continuación se mostrará en pseudocódigo el detalle de los programas implementados:

- LIBRERÍA DE TARJETA:

En primer lugar, se han utilizado las variables necesarias para la gestión de los recursos empleados: manejadores de tuberías e identificadores de eventos.

Otro grupo de variables definidas son las empleadas para conservar los datos utilizados en las diferentes funciones: en el caso del timer, el buffer que se usa para el envío de información a éste y el estado de la cuenta en curso.

Y en el caso de las funciones relativas a la captura de datos, el buffer donde registrar los resultados de las lecturas y las variables que indican el estado en el que se encuentra dicho proceso.

A la librería original se le agregarán las siguientes funciones:

- _9112_TIMER_Start

Construir la trama = "TS#canal#valor".

Comunicar evento de envío de datos al timer.

Enviar trama al timer a través de "tubotimer".

- _9112_TIMER_Read

Construir la trama = "TR#canal#valor".

Comunicar evento de envío de datos al timer.

Enviar trama al timer a través de "tubotimer".

Recibir datos de timer.

Retornar los datos al programa de prácticas.

- _9112_TIMER_Stop

    Construir la trama = "TX#canal#valor".

    Comunicar evento de envío de datos al timer.

    Envío de trama al timer a través de "tubotimer".

    Recibir datos de timer.

    Retornar los datos al programa de prácticas.

- _9112_AD_INT_Start

    Llamar función "inicializaRegistros".

    Lanzar proceso de lecturas "_AD_INT_Background".

- _AD_INT_Background

    Mientras esté la lectura activada y no se haya llenado el buffer de registros:

    Llamar a la función de librería AI_ReadChannel.

    Recibir el dato requerido.

    Registrar la medida obtenida en el buffer de datos.

    Repetir toda la operación en caso favorable.

- _9112_AD_INT_Status

    Comprobar si ha finalizado el proceso de adquisición de datos.

    Obtener resultado de la consulta y el buffer datos registrados.

    Retornar estado del proceso y buffer de datos al programa de prácticas.

- _9112_AD_INT_Stop

  Desactivar el flag de lectura en "_AD_INT_Background".

  Obtener buffer con los datos registrados.

  Eliminar el proceso "_AD_INT_Background".

  Retornar el buffer con los datos registrados al programa de prácticas.

- TIMER VIRTUAL:

Las variables que requerirá la aplicación serán, además de las necesarias para la gestión de los recursos empleados (manejadores de tuberías e identificadores de eventos) las utilizadas para guardar la información del valor contador, el tiempo inicial de la cuenta y el buffer con la trama proveniente de la librería.

Al iniciarse la aplicación se creará un hilo de programación que permanecerá a la espera de tramas procedentes de la librería.

La lógica de este hilo será la siguiente:

- Hilo Servidor de Timer:

Abrir evento para recibir tramas de la librería.

Esperar evento.

Evento recibido.

Lectura de datos de la librería.

Remitir el mensaje a la función auxiliar "DecodificarTrama".

- DecodificarTrama:

Enviar trama a "TokenizarMensaje".

Obtener campo "Valor" de la trama.

Dependiendo de la cabecera obtenida, proceder a la acción solicitada.

- "TS": Obtener tiempo inicial.

Comenzar cuenta: anotar hora del Sistema.

- "TR": Obtener valor del contador en ese momento.

Retornar el dato solicitado a la librería.

- "TX": Obtener valor del contador en ese momento.

Retornar el dato solicitado a la librería.

Inicializar variables de cuenta (tiempo inicial y contador).

4.3 CAPA DE LA INTERFAZ DE USUARIO

Para que el usuario del simulador de la tarjeta sea plenamente consciente de la utilización del timer virtual, se ha determinado que dicha aplicación de timer sea una aplicación visual.

En el inicio de la aplicación, además de las aplicaciones que ya se lanzaban, se abrirá una ventana como señal de que el proceso de timer está disponible para comenzar a realizar cuentas. Esta aplicación será lanzada por la librería virtual al comienzo de la sesión y ella misma llamará a su destrucción antes de cerrarse por completo las aplicaciones.

La ventana, que permanecerá abierta durante todo el tiempo de uso de la tarjeta virtual, no interactuará de ninguna otra forma con el usuario, siendo un mero indicador de la nueva funcionalidad disponible en la virtualización de la tarjeta.

5 - Implementación

La implementación del código de este proyecto viene dividida en dos partes: la primera es la ampliación de la colección de funciones en la librería de la tarjeta y la segunda es el desarrollo de una aplicación que simula el timer 8253.

Las partes utilizadas del software, ya presentes al comienzo de este proyecto y referentes a la tarjeta virtual, el proceso virtual y las funciones de librería existentes en origen, no serán detalladas, puesto que ya fueron expuestas en el proyecto del autor del software: Miguel Carro Pellicer.

El código fuente estará disponible en los archivos fuente, adjuntos al proyecto.

A continuación se detallarán solamente las nuevas funciones implementadas.

5.1 MECANISMOS DE INTERCOMUNICACIÓN DE PROCESOS

- Funciones de creación de procesos e hilos:

· Desde la librería, utilizaremos la siguiente función para crear el nuevo proceso de simulador de Timer 8253:

```
BOOL CreateProcess(
        LPCTSTR lpApplicationName,
        LPTSTR lpCommandLine,
        LPSECURITY_ATTRIBUTES lpProcessAttributes,
        LPSECURITY_ATTRIBUTES lpThreadAttributes,
        BOOL bInheritHandles,
        DWORD dwCreationFlags,
        LPVOID lpEnvironment,
        LPCTSTR lpCurrentDirectory,
        LPSTARTUPINFO lpStartupInfo,
        LPPROCESS_INFORMATION lpProcessInformation);
```

De todos los argumentos indicados en la llamada a la función, se utilizarán los siguientes (dándoles un valor nulo a todos los demás):

- *lpCommandLine*: programa que se va a ejecutar (Timer virtual en nuestro caso).

- *bInheritHandles*: se le dará un valor TRUE, para que el nuevo proceso herede los manejadores heredables del proceso padre, la librería de tarjeta.

- *lpStartupInfo*: estructura en la que se registran los datos referentes a la ventana del proceso y los descriptores estándar en el momento de la creación de dicho proceso.

- *lpProcessInformation*: estructura que contendrá los descriptores e identificadores del nuevo proceso.

· La función que se utilizará para esperar a que se produzca un evento es:

DWORD **WaitForSingleObject**(

HANDLE hHandle,

DWORD dwMilliseconds);

En este proyecto será utilizada para que el timer virtual permanezca a la espera de eventos lanzados desde la librería. También será utilizada para que la librería espere a la terminación del proceso timer virtual tras lanzar la orden de parada de dicha aplicación.

- *hHandle*: referencia el evento que se espera.

- *dwMilliseconds*: intervalo de tiempo de espera del proceso que lo invoca, en los casos de este proyecto, se le dará un valor INFINITE.

· Para la creación de hilos de ejecución se utilizará la función:

HANDLE **CreateThread**(

LPSECURITY_ATTRIBUTES lpThreadAttributes,

DWORD dwStackSize,

LPTHREAD_START_ROUTINE lpStartAddress,

LPVOID lpParameter,

DWORD dwCreationFlags,

LPDWORD lpThreadId);

La función devolverá el manejador del hilo asociado.

De todos los argumentos indicados en la llamada a la función, se utilizarán los siguientes (dándoles un valor nulo a los demás):

- *lpStartAddress*: función que se debe ejecutar, en este proyecto se utilizará para ejecutar los procesos "Servidor de Timer" y "AD_INT_Background".

- *lpThreadId*: se retornará en esta variable el identificador de hilo.

· La función que se utilizará para cerrar los manejadores abiertos:

BOOL **CloseHandle**(

HANDLE hObject);

En este proyecto será llamada antes de la salida del programa principal, liberando así los recursos asignados. También será usada tras la llamada a finalización de la aplicación timer, por parte de la librería.

· La función utilizada para finalizar un proceso (y todos sus hilos):

BOOL WINAPI **TerminateProcess**(

HANDLE hProcess,

UINT uExitCode);

Será usado por la librería para ordenar la detención de la aplicación timer virtual.

Los argumentos indicados en la llamada a la función serán:

- *hProcess*: el manejador del proceso a terminar.

- *uExitCode*: código de salida, que será usado por el proceso.

- Funciones de creación de tuberías, lectura y escritura:

· En la aplicación de timer, utilizaremos la siguiente función para la creación de la tubería con nombre "tubotimer":

HANDLE **CreateNamedPipe**(

LPCTSTR lpName,

DWORD dwOpenMode,

DWORD dwPipeMode,

DWORD nMaxInstances,

DWORD nOutBufferSize,

DWORD nInBufferSize,

DWORD nDefaultTimeOut,

LPSECURITY_ATTRIBUTES lpSecurityAttributes);

Con esta llamada se crea una tubería con nombre, con la cual se comunicarán los procesos que hagan uso de ella. La función devolverá el identificador, que se utilizará posteriormente para el envío y la recepción de datos. En este proyecto se comunicarán mediante "tubotimer" la librería y el timer virtual.

A los argumentos indicados en la llamada a la función, se les asignará los siguientes valores:

- *lpName*: nombre de la tubería.

- *dwOpenMode*: Modo de apertura de la pipe, en este caso será PIPE_ACCESS_DUPLEX: bidireccional.

- *dwPipeMode*: Tipo de tubería. Será una pipe orientada a bytes, con modo de lectura orientado a bytes y con espera de lectura y escritura bloqueante. Para ello se enviará como argumento los flags:

PIPE_TYPE_BYTE | PIPE_READMODE_BYTE | PIPE_WAIT

- *nMaxInstances*: Número máximo de instancias posibles, en este caso se le dará el valor PIPE_UNLIMITED_INSTANCES (255).

- *nOutBufferSize*: Bytes a reservar para el buffer de salida (0).

- *nInBufferSize*: Bytes a reservar para el buffer de entrada (0).

- *nDefaultTimeOut*: Time-out por defecto, en este caso INFINITE.

- *lpSecurityAttributes*: Para que la tubería obtenga un descriptor de seguridad predeterminado y el manejador no pueda ser heredado, el valor asignado será NULL.

· La llamada que espera la conexión en el otro extremo de la tubería es:

BOOL **ConnectNamedPipe**(

HANDLE nNamedPipe,

LPOVERLAPPED lpo);

Será invocada por el timer virtual y, de manera bloqueante, esperará la conexión de la librería.

Se les asignará los siguientes valores a los argumentos de la función:

- *nNamedPipe*: manejador asociado a la tubería.

- *lpo*: sólo para E/S asíncronas (en nuestro caso valor NULL).

· Función utilizada para conectarse a una tubería, creada desde otro extremo:

BOOL **WaitNamedPipe**(

LPCTSTR lpNamedPipeName,

DWORD nTimeOut);

En este proyecto, será la librería quien se intente conectar a la tubería "tubotimer", que previamente deberá haber sido creada en el timer virtual.

Los valores de los argumentos de la llamada a la función serán:

- *lpNamedPipeName*: nombre de la tubería.

- *nTimeOut*: tiempo máximo de espera. Para usar el valor de tiempo de espera igual al que ya se indicó en la creación de la tubería, se usará como valor del argumento NMPWAIT_USE_DEFAULT_WAIT.

· Llamada para la desconexión de la tubería:

BOOL **DisconnectNamedPipe**(

HANDLE nNamedPipe);

Esta función será invocada desde el timer virtual, cuando vaya a desconectarse de la tubería, eliminando también la instancia asociada a ella.

En la llamada a la función sólo se envía el nombre del manejador asociado.

· Llamada para abrir la tubería, como si de un fichero se tratara:

HANDLE **CreateFile**(

LPCTSTR lpFileName,

DWORD dwDesiredAccess,

DWORD dwShareMode,

LPSECURITY_ATTRIBUTES lpSecurityAttributes,

DWORD dwCreationDistribution,

DWORD dwFlagsAndAttributes,

HANDLE hTemplateFile);

La función será llamada, desde la librería, tras la ejecución de la función WaitNamedPipe. Con ello se abrirá el descriptor de fichero asociado a la tubería, para poder escribir y leer en él.

Los valores de los parámetros pasados en la función son los siguientes:

- *lpFileName*: nombre de la tubería.

- *dwDesiredAccess*: tipo de acceso, en nuestro caso será para lectura y escritura: GENERIC_READ | GENERIC_WRITE.

- *dwShareMode*: si la tubería será de tipo compartido o no. En este caso no podrá ser compartida y, por tanto, su valor 0.

- *lpSecurityAttributes*: Para que el manejador devuelto por esta función no pueda ser heredado y el fichero asociado con este identificador obtenga un descriptor de seguridad predeterminado, el valor será NULL.

- *dwCreationDisposition*: Acción a tomar si la tubería existe o no. En este caso determinaremos la opción OPEN_EXISTING (abrir sólo si existe).

- *dwFlagsAndAttributes*: flags y atributos que se le asignarán al descriptor, en este caso FILE_ATTRIBUTE_NORMAL (sin atributos seleccionados).

- *hTemplateFile*: especifica un manipulador con tipo de acceso lectura a un fichero que se utilizará como plantilla. No se requiere y su valor será 0.

· Para leer los datos de la tubería se utilizará la siguiente función:

```
BOOL ReadFile(
        HANDLE hFile,
        LPVOID lpBuffer,
        DWORD nNumberOfBytesToRead,
        LPDWORD lpNumberOfBytesRead,
        LPOVERLAPPED lpOverlapped);
```

Esta llamada se realizará, en el caso del timer, para recibir las tramas con las órdenes a ejecutar y, en la parte de la librería, para obtener los datos retornados tras algunas de esas órdenes.

Los valores asignados a los parámetros en la llamada son:

- *hFile*: manejador de fichero de la tubería (obtenido en CreateFile).

- *lpBuffer*: buffer donde se recibirán los datos de la trama.

- *nNumberOfBytesToRead*: número de bytes a leer.

- *lpNumberOfBytesRead*: número de bytes leídos.

- *lpOverlapped*: dirección de la estructura para datos para E/S asíncronas. En este caso no lo son y, por tanto, el valor es NULL.

· Para escribir datos en la tubería se utilizará la siguiente función:

BOOL **WriteFile**(

HANDLE hFile,

LPCVOID lpBuffer,

DWORD nNumberOfBytesToWrite,

LPDWORD lpNumberOfBytesWritten,

LPOVERLAPPED lpOverlapped);

Esta llamada será utilizada por la librería para enviar las tramas con las órdenes a ejecutar y, en caso de ser necesario, por parte del timer virtual para enviar los datos de retorno, requeridos por la librería.

Los valores asignados a los parámetros en la llamada serán:

- *hFile*: manejador de fichero de la tubería (obtenido en CreateFile).

- *lpBuffer*: buffer con la trama a escribir.

- *nNumberOfBytesToWrite*: número de bytes a escribir.

- *lpNumberOfBytesWritten*: número de bytes escritos.

- *lpOverlapped*: para E/S asíncronas, en nuestro caso NULL.

- Funciones de creación de eventos y activación:

Se utilizarán eventos para sincronizar procesos, que permanecerán a la espera mientras otro proceso no active dicho evento.

· La función con la que se crearán los eventos es:

HANDLE **CreateEvent**(

LPSECURITY_ATTRIBUTES lpEventAttributes,

BOOL bManualReset,

BOOL bInitialState,

LPCTSTR lpName);

Se utilizará el evento "EventoTimer" para que la librería avise al timer virtual del envío de tramas. Mientras tanto, el timer permanecerá a la espera de este evento.

Los valores asignados a las variables en la llamada a la función CreateEvent son:

- *lpEventAttributes*: especifica los atributos de seguridad para el objeto evento. El valor será NULL, para que éste sea creado con un descriptor de seguridad por defecto y el manipulador resultante no sea heredable.

- *bManualReset*: si el evento es de reset manual o no. En este caso será FALSE, será el sistema operativo quien inicie su estado a "no señalizado".

- *bInitialState*: indica el estado inicial del evento. En este caso será FALSE, "no señalizado".

- *lpName*: nombre que se le dará al evento ("EventoTimer").

· Tras la creación del evento, para conectarse a éste utilizaremos la función:

HANDLE **OpenEvent**(

DWORD dwDesiredAccess,

BOOL bInheritHandle,

LPCTSTR lpName);

El evento "EventoTimer" será creado en el timer virtual y la librería ejecutará esta función para poder hacer uso del evento a la hora de avisar de envío de tramas.

Los valores tomados por los argumentos de la función son:

- *dwDesiredAccess*: tipo de acceso al evento, en este caso se seleccionará la opción todos: EVENT_ALL_ACESS

- *bInheritHandle*: si el manejador será heredable, FALSE en nuestro caso.

- *lpName*: nombre que se le dará al evento ("EventoTimer").

· Para activar el evento creado anteriormente se usará la función:

BOOL **SetEvent**(

HANDLE hEvent);

El evento "EventoTimer" será activado por la librería, para que el timer (que permanecerá a la espera del evento, ejecutando la función WaitForSingleObjet) reciba la señal de leer de la tubería lo que se ha escrito.

En la llamada a la función sólo se envía el nombre del manejador asociado al evento, obtenido tras la llamada a CreateEvent.

5.2 CAPA LÓGICA DE LA APLICACIÓN

En el apartado referido a la capa lógica, de la sección de diseño de la aplicación, ya ha sido explicado con bastante detalle el funcionamiento de la misma, no creyendo necesario concretar más sobre este respecto.

A nivel de implementación, comentar que se han utilizado funciones del lenguaje de programación C, dentro del entorno de desarrollo Code::Blocks, por lo que tampoco será necesario una explicación completa de cómo está estructurado.

Se podría destacar, como detalle de la implementación, el método elegido para la simulación del Timer 8253:

El timer 8253, incluido en la tarjeta, es un dispositivo hardware que consiste en 3 contadores descendentes de 16 bits cada uno. Funciona de forma permanente, decrementando su cuenta en cada flanco activo de reloj. Este temporizador posee 6 modos programables de contador, en este proyecto se simulará el Modo 0 o de “Interrupción al final de la cuenta”. En este modo se realiza una cuenta descendente, desde el valor indicado por el usuario hasta llegar a cero, finalizando así la cuenta.

A la hora de decidir cómo abarcar la implementación de Timer 8253 se ha tenido en cuenta que los ordenadores personales ya disponen de varios temporizadores hardware, que permiten realizar tareas de administración o simplemente mantener la hora y fecha (como, por ejemplo; el High Precision Event Timer (HPET)), que emplearemos para nuestra simulación.

Otro detalle a considerar es que el hecho de tener un proceso realizando una cuenta constantemente consumiría muchos recursos, de manera que no sería viable su uso. Este es el motivo por el cual se ha tomado como solución realizar un cálculo del estado de la cuenta en el momento de la consulta, sin necesidad de haber ido realizando una cuenta como tal. Para ello se llamará a la función clock() en el momento de la llamada a _9112_TIMER_Start, de este modo quedará registrado el dato del tiempo inicial en la variable “tiempoInicial”. Cuando se solicite el valor del estado de la cuenta (con las llamadas a las funciones _9112_TIMER_Read y _9112_TIMER_Stop) se llamará nuevamente a clock() y, usando el dato anterior, se realizarán los cálculos necesarios para obtener el valor del contador en ese instante.

Como ya se ha comentado con anterioridad, no es necesario ni posible simular los 2MHz del timer real, por lo que se ha decidido considerar una frecuencia de reloj inferior, configurable por el usuario (para ello se utilizará el fichero de configuración timerVirtual.cfg). Los valores de frecuencia que se estiman razonables para percibir visualmente los cambios en la cuenta se encuentran en el intervalo comprendido entre 1 y 100Hz.

En resumen, con esta virtualización se dispondrá de un timer de 16 bits descendente (igual que el Timer 8253 original), pero que se limitará a emular el modo de cuenta TIMER_MODE0 y cuya frecuencia de reloj será configurable por el usuario.

Otro detalle de la implementación a destacar es el método elegido para la simulación de la adquisición de múltiples muestras:

La función original de la tarjeta realiza un número N de conversiones A/D solicitadas, con interrupción de transferencia de datos al finalizar las conversiones. Las conversiones se pueden detener en cualquier instante mediante otra función que da la correspondiente orden a la tarjeta, estando disponibles las muestras tomadas hasta entonces.

Para simular esta adquisición de múltiples muestras se ha decidido utilizar un proceso, lanzado desde la llamada de inicio (_9112_AD_INT_Start), que permanecerá ejecutándose en segundo plano. Este proceso, llamado AD_INT_Background, se dedicará a realizar tantas llamadas a la función AI_ReadChannel como número de lecturas se haya ordenado al comienzo del muestreo. La función AI_ReadChannel sólo realiza una lectura por llamada. Cada una de las lecturas se irá registrando en un buffer de datos (bufferAD) para su posterior uso.

El tiempo de espera entre muestras será controlado por la función Sleep(nseg), donde nseg será el número de segundos calculado a partir de los valores de los dos divisores de frecuencia indicados en la llamada a la función de inicio de muestreo. Dependiendo del valor de dichos parámetros, se conseguirán frecuencias de reloj menores y, por tanto, el muestreo será más lento y perceptible por el usuario.

5.3 CAPA DE LA INTERFAZ DE USUARIO

La única interfaz gráfica que ha sido realizada en este proyecto es la aplicación "Timer Virtual". Se trata de una aplicación wxWidget realizada en el entorno de desarrollo Code::Blocks y la única finalidad que tiene dentro del proyecto es que el usuario pueda comprobar de manera visual que el Timer está en funcionamiento, es un testigo que indica que la cuenta se está produciendo.

La interfaz que puede visualizar el usuario tiene el siguiente diseño:

Imagen 5 – Interfaz de Timer Virtual

Para realizar esta aplicación se han utilizado el objeto de Code::Blocks:

- wxStaticBitmap: Este objeto se utilizará para poder cargar una imagen desde fichero. Dicha imagen será en este caso una imagen en formato .bmp y se utilizará como fondo de la ventana de la aplicación

5.4 HISTORIAL DE VERSIONES

A continuación se describirá la evolución del proyecto, tomando como referencia las diferentes versiones de la aplicación que se han realizado y los hitos a los que se ha llegado en cada una de estas versiones:

VERSIÓN 21 DE MAYO DE 2015:

- Ampliación de la librería de la tarjeta con las funciones de simulación de timer.

- Comprobación de la comunicación entre la librería y la aplicación timer mediante nueva tubería y eventos asociados. Se muestran mensajes de log en diferentes puntos del programa, para esa verificación.

- Se trabaja con una versión básica de la aplicación de timer, para la comprobación de tráfico correcto de tramas. Se lanza con la llamada a TimerStart y se cierra la aplicación con la llamada a TimerStop.

VERSIÓN 1 DE JUNIO DE 2015

- Creación de la versión completa de la aplicación timer virtual.

- Correcta codificación de los mensajes de timer previa a su envío.

- Correcta decodificación de las tramas de los mensajes recibidos por el timer.

- Creación de un programa de pruebas con el cual revisar al completo las nuevas prestaciones.

VERSIÓN 15 DE JUNIO DE 2015

- Ampliación de la librería, añadiendo las funciones que simulan la adquisición de múltiples muestras. En esta primera versión la función sólo realiza una lectura, siendo prioritaria la implementación de funciones y su funcionamiento.

- Ampliación del programa de pruebas para integrar las nuevas funcionalidades.

VERSIÓN 2 DE AGOSTO DE 2015

- Implementación del programa que registra las lecturas en segundo plano.

- Se determina un tamaño máximo de 65.000 muestras y queda pendiente probar si es excesivo o no.

VERSIÓN 15 DE SEPTIEMBRE DE 2015

- Debido al problema detectado al lanzar más de una tarjeta virtual, se añade una comprobación previa al lanzamiento de la tarjeta. Si ya existe una en ejecución, se devolverá un mensaje de error al usuario avisando de la existencia del problema.

- Por el mismo motivo, se decide modificar el código de las funciones de timer para que se lance la aplicación timer virtual al inicio de la sesión y se destruya al final de ésta, no realizándolo ya las funciones TimerStart y TimerStop.

- Eliminación de los mensajes de log utilizados para la depuración de errores.

6 - Pruebas

Como se ha indicado en varias ocasiones a lo largo del documento, un requisito esencial a la hora de ampliar la colección de funciones de la tarjeta ha sido que no se viera afectada ninguna de las funcionalidades ya existentes en el antiguo simulador. Por este motivo, las pruebas se han dedicado tanto a comprobar el funcionamiento esperado de las nuevas aplicaciones como a verificar que no se vieran afectadas las ya existentes.

El programa de pruebas está programado en C (lenguaje que utilizan los alumnos de prácticas para comunicarse con la tarjeta virtual) y se ha ido modificando con el tiempo, para cumplir los requisitos existentes en cada una de las versiones del proyecto. El código del programa de pruebas final es el siguiente:

```
void mostrar_menu(int *x);

int main(int argc, char* argv[]) {

  int card,res=0,contadorTimer,prueba=1;
  int canal=0,i,estado,iter,contador=15,registros=-1;
  char parar;
  unsigned long ad_buffer[256];
  unsigned long dato;
  float voltios;


  card=Register_Card(PCI_9112,num_card);
  if(card<0)
    printf("Error en la tarjeta. Avisa al profesor\n");
  else
    printf("Tarjeta inicializada.\n");

  printf("\nBIENVENIDO AL PROGRAMA DE PRUEBAS\n");

  mostrar_menu(&prueba);
  while(prueba!=0) {

    switch(prueba){
    case(1): {     //Opcion 1: Probar el Timer virtual.

      printf("\nLANZANDO EL TIMER VIRTUAL\n");
      contadorTimer=200;
      res=_9112_TIMER_Start(card,0,contadorTimer);
      Sleep(1500);

      printf("Timer ejecutandose con contador c0=%d\n",contadorTimer);
      printf("Pulsa Intro para LEER el valor del timer\n");
      getchar();
      getchar();
```

```
  res=_9112_TIMER_Read(card,&contadorTimer);
  printf("Tras la llamada a timerRead valor devuelto=|%d|
        (PulsosDeReloj/valorC0)\n",contadorTimer);

  printf("Pulsa Intro para LEER timer de nuevo\n");
  getchar();
  res=_9112_TIMER_Read(card,&contadorTimer);
  printf("Tras llamada a timerRead valor devuelto=|%d|\n",contadorTimer);

  printf("Pulsa Intro para PARAR el timer virtual: ");
  getchar();
  res=_9112_TIMER_Stop(card,&contadorTimer);
  printf("Tras llamada timerStop valor de timer=|%d|.\n",contadorTimer);

  printf("Prueba finalizada. Pulsa cualquier tecla\n ");
  getchar();
  break;
}
case(2): {      //Opcion 2: Probar la captura de datos A/D

  parar='0';
  printf("\nVA A COMENZAR LA CAPTURA DE %d DATOS A/D EN EL CANAL
         %d\n\n",contador,canal);
  printf("Pulsa 1 si quieres que se PARE en mitad de la captura:");
  getchar();
  scanf("%c",&parar);

  res=_9112_AD_INT_Sart(card,0,canal,0,contador,ad_buffer,0,0);
  Sleep(1000);
  estado=-1;
  iter=0;

  while(estado!=0 && iter<=contador) {
    //Si se ha elegido la opción de parar en mitad del muestreo,
    //finalizará tras realizarse 8 lecturas
    if(iter==8 && parar=='1') {
      printf("Has PARADO el proceso antes de capturar los %d datos\n",
             contador);
      break;
    }
  res=_9112_AD_INT_Status(card,&estado,&registros,ad_buffer);
  printf("Registros almacenados |%d| - Proceso finalizado |%d|
        (1-No, 0-Si)\n\n",registros,estado);
  Sleep(500);
  iter++;
  }
  res=_9112_AD_INT_Stop(card,&registros,ad_buffer);
  printf("Obtenidos %d registros de los %d solicitados:\n", registros,
        contador);
  printf(" BUFF | VOLT\n");
  //Se mostraran todas las lecturas registradas
  for(i=0;i<registros;i++) {
    dato = ad_buffer[i] >> 4;
    voltios = (10.0*dato/4095)-5;
    printf("%ld - %.2f\n",ad_buffer[i],voltios);
  }
  break;
  }
```

```
    case 3: {      //Opcion: Uso original de la Tarjeta virtual

      printf("\nACTIVA LOS INTERRUPTORES DEL PROCESO VIRTUAL\n");
      while(!kbhit()) {
      res=DI_ReadPort(card,0,&dato);
      printf("Activados los interruptores: %ld\n",dato);
      if(dato&1)
        res=DO_WritePort(card,0,0xF0);
      else
        res=DO_WritePort(card,0,0x0F);
      Sleep(1000);
      }
      break;
    }

    default: {     //Opcion no valida
      printf("Pulsa Intro para elegir nueva prueba:\n ");
      getchar();
      break;
    }
  }
  mostrar_menu(&prueba);
  }

  printf("Finalizado el programa de pruebas. Pulsa una tecla para salir");
  getchar();
  res=Disconnect_Card(PCI_9112,num_card);
  return res;
}

void mostrar_menu(int *x) {
  printf("Elije una de las siguientes opciones:\n");
  printf("\t1 - Probar el Timer virtual.\n");
  printf("\t2 - Probar la captura de datos A/D.\n");
  printf("\t3 - Uso original de la Tarjeta virtual.\n");
  printf("\t0 - Salir de la aplicacion.\n");
  do{
    printf("Y la opcion elegida es: ");
    scanf("%d",x);
  }while(*x<0 || *x>3);
}
```

Al inicio del programa se mostrará el siguiente menú:

Imagen 6 – Menú de programa de pruebas

Con cada opción se podrá elegir qué grupo de funciones examinar. Dentro de cada apartado se realizan pruebas de todas las funciones dedicadas a esa simulación en particular. Los resultados de las diferentes pruebas han sido satisfactorios, y con ello se puede afirmar que, tanto el nuevo desarrollo implementado en este proyecto como el antiguo simulador, son totalmente funcionales.

7 – Tests en diferentes plataformas

La finalidad del software desarrollado es que el alumnado pueda realizar las prácticas sin necesidad de disponer del hardware de la tarjeta y se pueda trabajar en cualquier PC. Debido a que el software original ya data del año 2007, surge la necesidad de verificar si existirán problemas de compatibilidad entre los nuevos sistemas operativos Windows y el antiguo software de virtualización. Por extensión, también será necesario probar el software desarrollado en este proyecto.

La idea final de estas pruebas es adelantarse a los problemas que puedan surgirle a los alumnos y buscar las posibles soluciones en cada caso, para así conocer de primera mano si el software de virtualización está o no disponible para trabajar en alguna plataforma Windows concreta.

Pasamos a detallar los resultados obtenidos en las tres distribuciones más utilizadas de Windows en el momento de la realización de este proyecto:

- WINDOWS 8:

Al ser el sistema operativo más actual de los tres probados, se decidió elegir este entorno para realizar el desarrollo de las nuevas funcionalidades, no sin antes asegurar que el simulador virtual anterior no sufría ninguna incompatibilidad.

Esta fue en realidad la primera fase que se abordó del proyecto: Para tener una idea clara del funcionamiento de la tarjeta virtual y para verificar que se podía trabajar sin problemas en Windows 8, se optó por realizar las sesiones de prácticas de la asignatura “Informatización industrial”, para la cual se creó la aplicación.

Las pruebas fueron satisfactorias y, a su vez se tuvo claro el funcionamiento del software de virtualización original, para así poder comenzar a implementar las soluciones decididas.

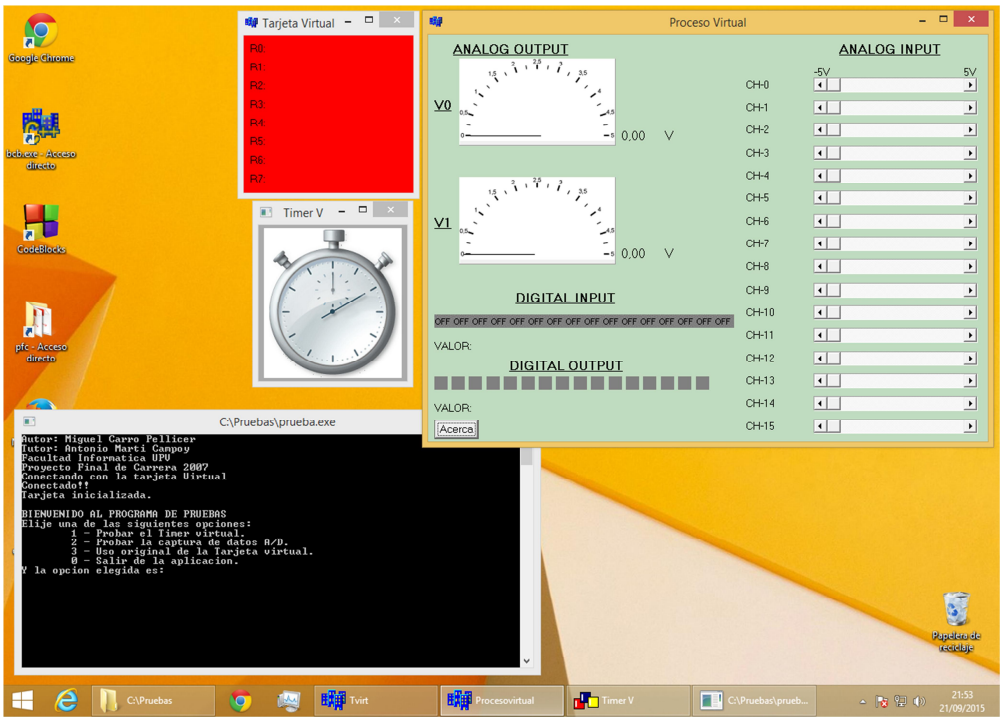

Imagen 7 – Pruebas en Windows 8

- WINDOWS 7:

Partiendo del conocimiento de que el software original era funcional en el sistema operativo sucesor (Windows 8), se estimó preferente implementar todo el desarrollo y proceder más tarde a realizar las pruebas globales, de todo el conjunto de software final. Los resultados en este caso también fueron satisfactorios, siendo completamente operativo el software original tras la incorporación del nuevo.

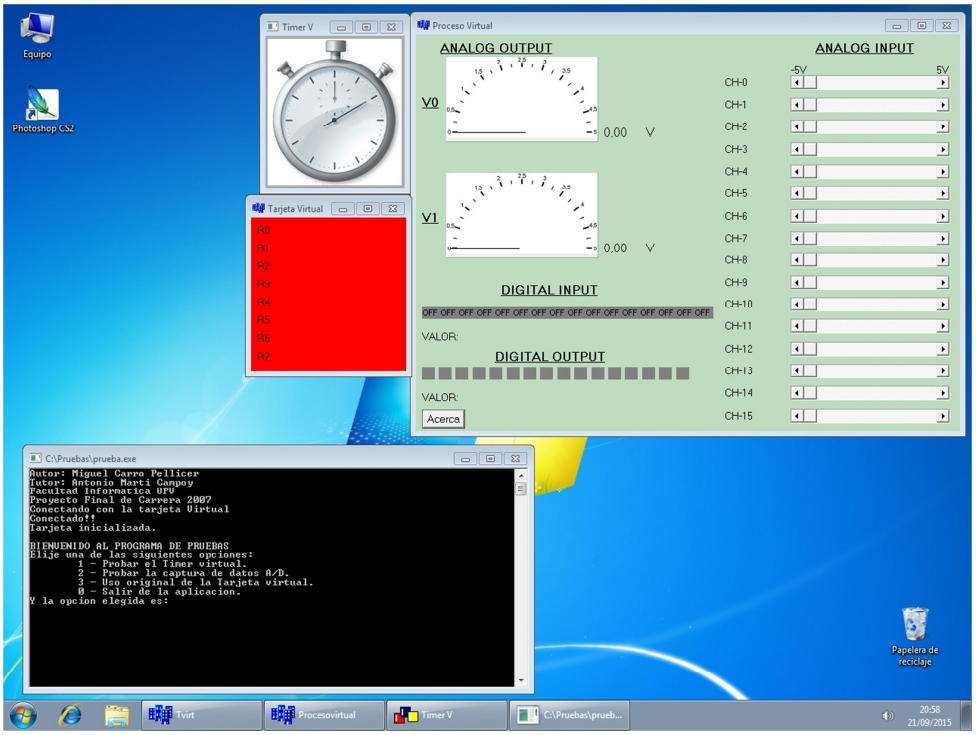

Imagen 8 – Pruebas en Windows 7

- WINDOWS XP:

A pesar de ser un sistema operativo sin soporte técnico desde el año 2014, todavía se pueden encontrar ordenadores con esta plataforma. Otro motivo por el cual realizar pruebas en este sistema operativo es que ha sido el utilizado en las prácticas los años anteriores, confirmándose con ello que el software de virtualización original era totalmente funcional.

En esta plataforma lo que se ha verificado es que el nuevo software añadido funciona sin problemas y no se ha visto afectada ninguna funcionalidad existente en el anterior software. Para ello se ha ejecutado el programa de pruebas expuesto en el apartado 6, confirmando que los resultados eran los esperados.

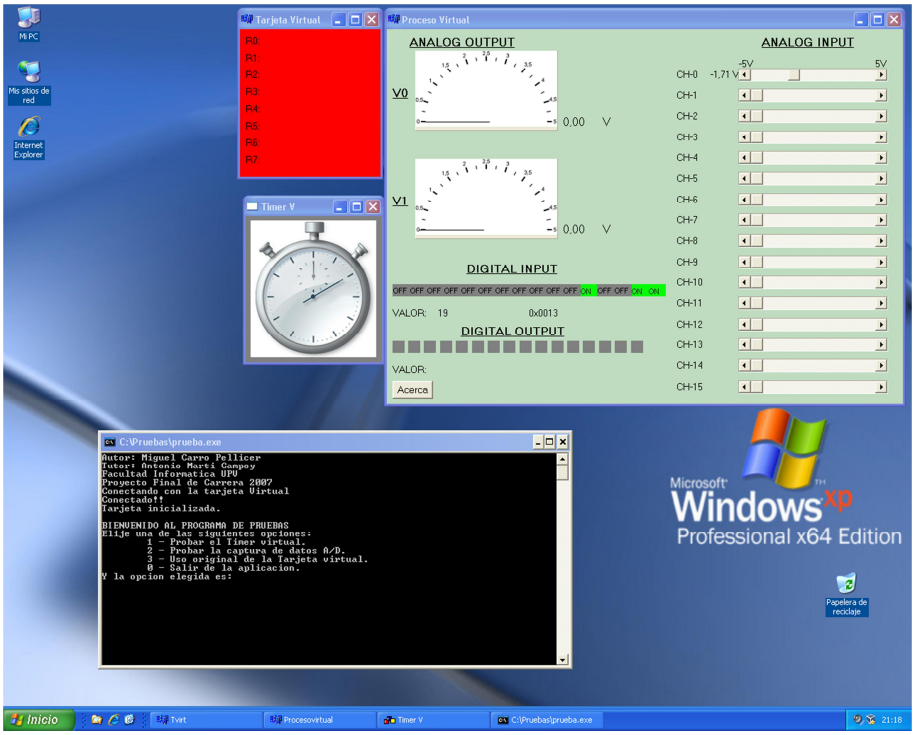

Imagen 9 – Pruebas en Windows XP

- MEJORAS PLANTEADAS TRAS LAS PRUEBAS:

Aunque no se encontraron problemas de compatibilidad con el software, sí se detectaron algunos conflictos durante el proceso de testeo.

1 - Al ejecutar la tarjeta virtual no se comprueba antes si ya existe una aplicación lanzada. En este caso quedan inoperativas tanto la primera tarjeta virtual como la segunda, teniendo que cerrarlas para poder comenzar a trabajar con una. Esto también ocurre si, por ejemplo, el proceso virtual falla y queda una tarjeta "huérfana".

Tras detectarse este problema, se realizaron pruebas también en el lanzamiento de la aplicación timer virtual, confirmándose que, si no se llamaba a timerStop para cerrar una aplicación ya en ejecución, se podían lanzar más timers virtuales, quedando inoperativos todos.

2 - Problemas para la creación de tuberías al lanzar la aplicación en máquinas virtuales lentas. El tiempo que tarda en lanzarse una aplicación difiere de un ordenador a otro y en el caso de entornos virtualizados este proceso se lentifica aún más. Por este motivo, si una máquina es lo bastante lenta, podrá ocurrir que no se termine de lanzar una aplicación cuando ya el programa principal intente conectarse a ésta mediante una tubería, no pudiéndose realizar la conexión.

Para intentar sortear el problema, se ha añadido tiempos de espera algo mayores entre la orden de lanzamiento de las aplicaciones y el intento de conexión mediante tuberías. Dicho tiempo tampoco se ha fijado a un valor muy alto, para no ralentizar demasiado el inicio de la aplicación.

8 – Futuras implementaciones

Como en este proyecto se han añadido nuevas funcionalidades a la tarjeta virtual, un trabajo futuro sería continuar por esta vía y, de entre las funciones de la librería del fabricante, seleccionar otras y ampliar la colección de funciones de la tarjeta real que se desee simular.

Otra evolución que se podría plantear en la virtualización sería la inclusión de interrupciones entre distintas aplicaciones (para avisar de la finalización de un proceso, por ejemplo). Con esta mejora se podrían abarcar nuevos proyectos que requieran del uso de interrupciones para que su funcionamiento sea lo más cercano a la realidad.

Dentro de la aplicación de timer virtual, y en el caso en el que se pudiera trabajar con interrupciones, se podrían añadir nuevos modos de contador, que sí requieren de interrupciones en su ejecución.

Otra futura ampliación del timer virtual sería el que se pudiera trabajar con más de una aplicación. Ya en el diseño de la trama de comunicación del timer se ha dejado abierta esta posibilidad, teniéndose que enviar el código de la aplicación timer con la que se desea comunicar.

9 – Conclusiones

Tras finalizar la implementación acordada para este proyecto, la herramienta de la que disponían los alumnos de la asignatura “Informatización industrial” queda ampliada y, por tanto se podrán añadir las nuevas funcionalidades en la programación de las prácticas de dicha asignatura.

Con la revisión exhaustiva que se ha realizado del funcionamiento del software de virtualización en diferentes máquinas, es posible asegurar que esta herramienta podrá ser utilizada en cualquiera de las tres distribuciones Windows en las que ha sido probada, no siendo el sistema operativo causante de ninguna disfunción de la herramienta.

En el inicio del proyecto se acordaron los hitos a alcanzar, revisándose y/o modificándose las versiones de éste en las sucesivas reuniones con el director del proyecto.

Tanto para el desarrollo del software como para el plan de pruebas ya comentados ha habido una comunicación fluida, quedando claras las tareas a realizar.

10- Bibliografía

- Miguel Carro Pellicer. ***Simulador de tarjeta de adquisición de datos Nudaq/Nuipc 9112 Series*** [Proyecto final de carrera] ETSINF-UPV, 2007

- ***Wikipedia for Code::Blocks*** [Wiki en Internet]. The Code::Blocks team.
Disponible en: http://wiki.codeblocks.org

- ***wxWidgets: Cross-Platform GUI Library*** [biblioteca online]. The wxWidgets team.
Disponible en: https://www.wxwidgets.org/

- ***MSDN Library - Microsoft*** [biblioteca online]. Microsoft Corporation
Disponible en: https://msdn.microsoft.com/es-es/library/ms123401.aspx

- ADLINK. ***NuDAQ / NuIPC 9112 Series Multi-function DAS Cards for PCI / 3U CompactPCI User's Guide*** [Manual de usuario]. Disponible en:
http://www.adlinktech.com/publications/manual/NuIPC3UIO/CP9112_50-11111-201.pdf

- INTEL. ***8254 Programmable Interval Timer*** [Hoja de datos del fabricante]

Disponible en: http://www.scs.stanford.edu/10wi-cs140/pintos/specs/8254.pdf

- ***Timer 8253*** En: Wikipedia: the free enciclopedia [Wiki en Internet]. St. Petersburg (FL): Wikimedia Foundation, Inc. 2001.
Disponible en: https://en.wikipedia.org/wiki/Intel_8253

11 – Agradecimientos

En primer lugar quiero expresar mi más sincero agradecimiento al director del proyecto, Antonio Martí Campoy, por todo el tiempo dedicado, sin importar ni el día de la semana ni el período de vacaciones. Agradecer también la celeridad al responder a mis dudas y toda la atención prestada. Ha sido una gran suerte para mí encontrar a una persona tan profesional y que demuestre tanto interés por su trabajo.

También quiero darle las gracias a mi marido Nicolás que, al regalarme el primer ordenador con la placa base defectuosa, hizo que se abriera ante mí un mundo hasta entonces desconocido, que ahora es mi vida y mi profesión. "Si no puedes con tu enemigo, únete a él" y en esas estamos, unida a la informática para siempre.

Una mención especial a Rosa Mari, mi hija, que no ha entendido por qué mamá no le ha hecho todo el caso que ella quería, pero que cuando crezca comprenderá que vale la pena realizar ciertos sacrificios...

Anexo 1: Manual de usuario

En este anexo se indicará cómo utilizar el software de virtualización, para poder trabajar con él como si se estuviera en posesión del hardware.

Antes de ejecutar cualquier programa que interactúe con este software, será necesario tener lanzado el Proceso Virtual (ejecutando ProcesoVirtual.exe):

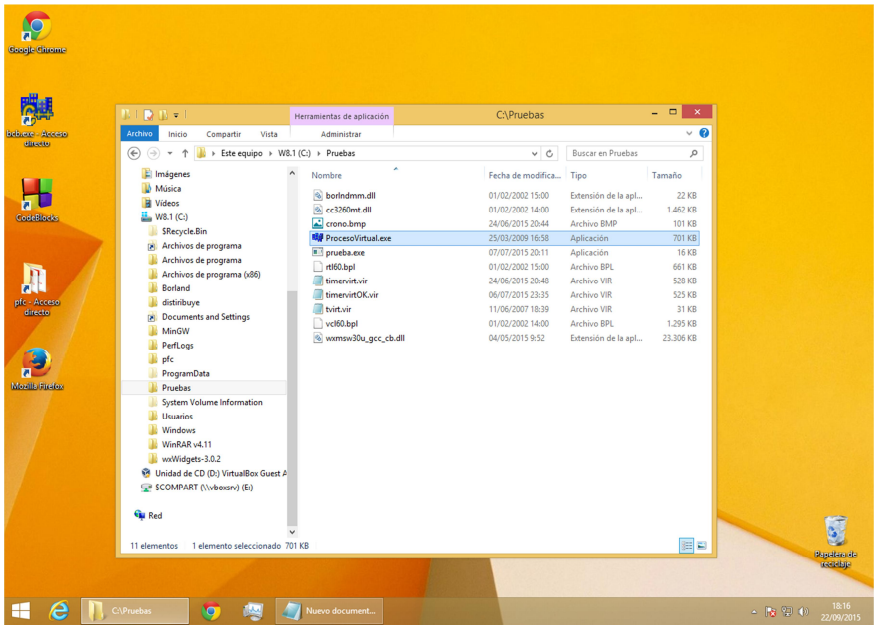

Imagen 10 – Lanzar el Proceso Virtual

Junto con el Proceso Virtual se lanzarán automáticamente la Tarjeta Virtual y el Timer Virtual, quedando todo listo para ser utilizado:

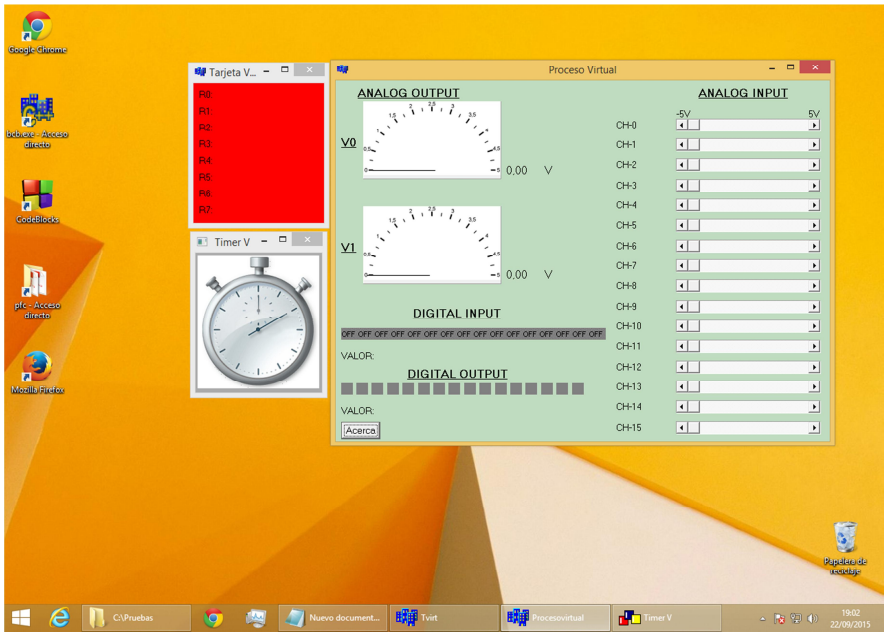

Imagen 11 – Software de virtualización listo para ser usado

Será necesario realizar un programa que utilice la librería de la tarjeta virtual. A continuación se describirán los pasos para realizar un proyecto utilizando el entorno de desarrollo Code::Blocks.

Se creará un nuevo proyecto, de tipo “Console application”:

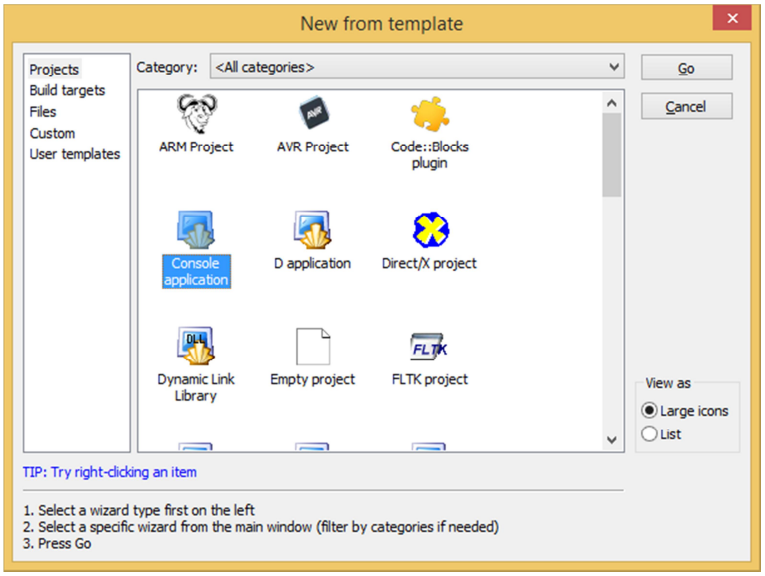

Imagen 12 – New Console application

Seleccionaremos el lenguaje que se utilizará, en este caso basta que sea en C:

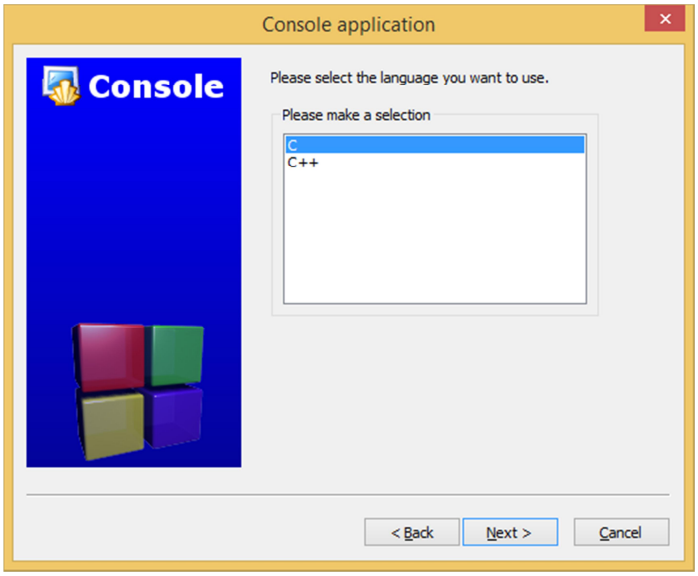

Imagen 13 – El programa de usuario será en C

Una vez creado el proyecto, para poder utilizar las funciones de la librería, necesitaremos incluir el fichero con los prototipos de las funciones (***LibTarjeta.h***, facilitado por el profesor) en el programa:

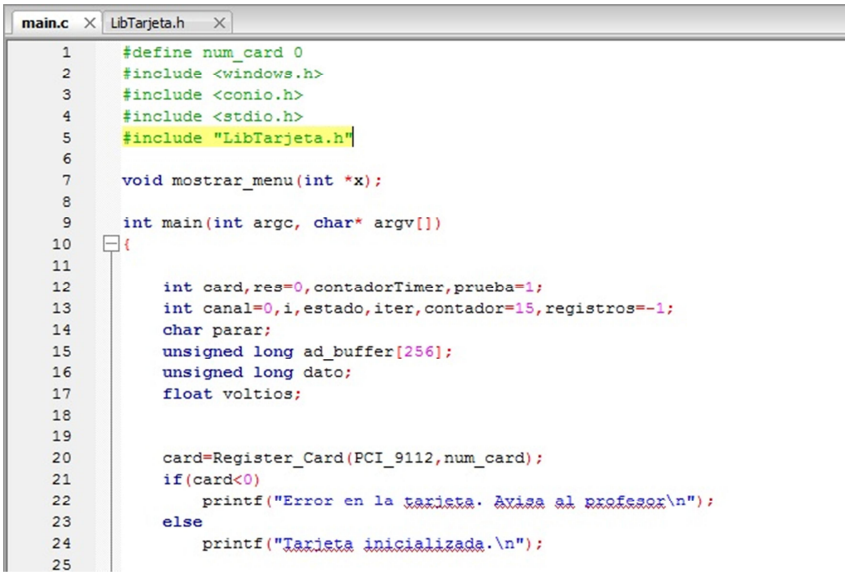

Imagen 14 – Incluir el fichero con las cabeceras

Después habrá que enlazar a nuestro proyecto la librería dinámica facilitada, cuyo nombre es ***libTarjeta.a***. Para ello habrá que ir a *Project -> Build options*

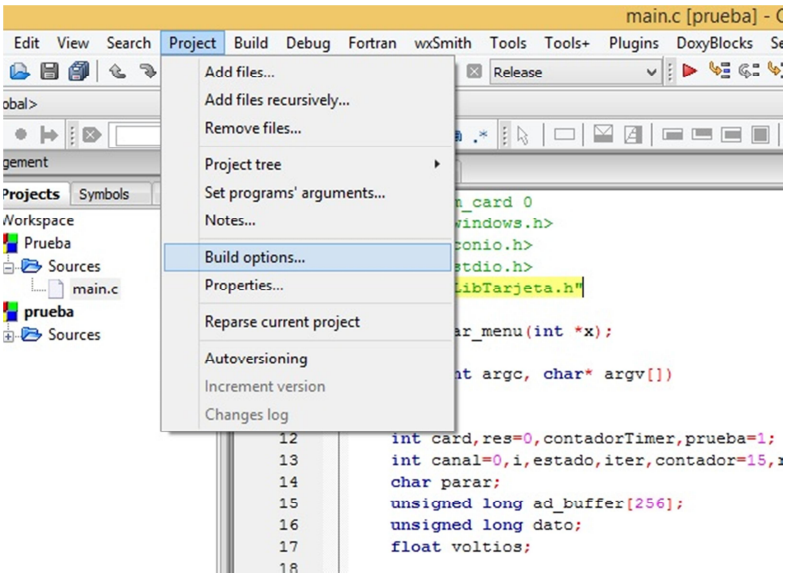

Imagen 15 – Enlazar la librería dinámica facilitada

Ir a la pestaña "*Linker settings*" y añadir la ruta donde se encuentra la librería:

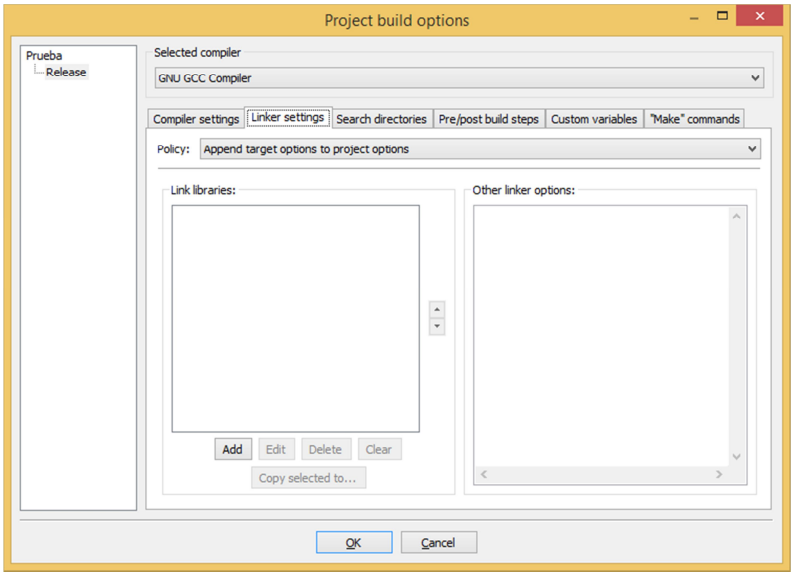

Imagen 16 – Ir a la pestaña "Linker settings"

MUY IMPORTANTE: Cuando seleccionemos la ruta donde se ubica la librería, nos consultará si la queremos guardar como una ruta relativa, para no tener problemas futuros, seleccionar **NO**.

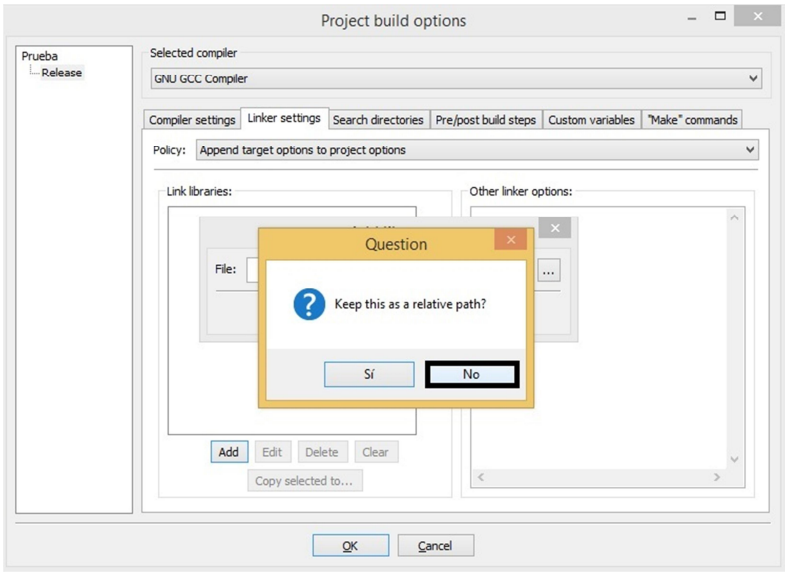

Imagen 17 – No guardar la ruta de la librería como relativa

A la hora de utilizar las funciones de Timer o las capturas de datos en ráfaga, existe la posibilidad trabajar con una frecuencia de reloj configurable por el usuario. En caso de querer trabajar con un valor de frecuencia diferente al asignado por defecto (que será de 10Hz), se le asignará el valor en Hz al parámetro ***fclk*** del fichero de configuración de nombre "***param.cfg***". Por ejemplo, para una frecuencia de reloj de 100Hz debe quedar así:

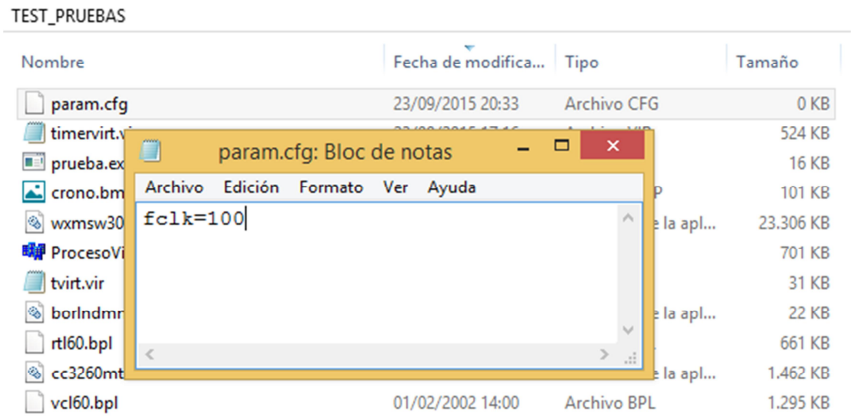

Imagen 18 – Frecuencia de reloj configurable

El fichero debe estar ubicado en la misma ruta que el ejecutable y su nombre será el indicado en este documento. En el caso en el que no se cumplan estas premisas o no se pueda obtener un dato coherente de este fichero, se aplicará la frecuencia de reloj por defecto (10 Hz). El valor del parámetro fclk se cargará al registrar la tarjeta virtual desde el programa del usuario y será necesario reiniciar dicho programa si se desea modificar su valor.

Ya se estará en disposición de ejecutar el programa, en este caso se muestra un programa de ejemplo en el que parpadearán las salidas digitales impares cada segundo (simulando unos leds intermitentes):

main.c | LibTarjeta.h

```c
#define num_card 0
#include <windows.h>
#include <conio.h>
#include <stdio.h>
#include "LibTarjeta.h"

void mostrar_menu(int *x);

int main(int argc, char* argv[]) {

    int card,res=0;

    card=Register_Card(PCI_9112,num_card);
    if(card<0)
        printf("Error en la tarjeta. Avisa al profesor\n");
    else
        printf("Tarjeta inicializada.\n");

    printf("\nLEDS INTERMITENTES EN EL PROCESO VIRTUAL\n");
    while(!kbhit()) {

        res=DO_WritePort(card,0,0x55);
        Sleep(1000);
        res=DO_WritePort(card,0,0x00);
        Sleep(1000);
    }
    res=Disconnect_Card(PCI_9112,num_card);
    return res;
}
```

Imagen 19 – Programa de pruebas

Ejecutando este programa se observa que las salidas digitales indicadas en el programa parpadean, como se ha ordenado:

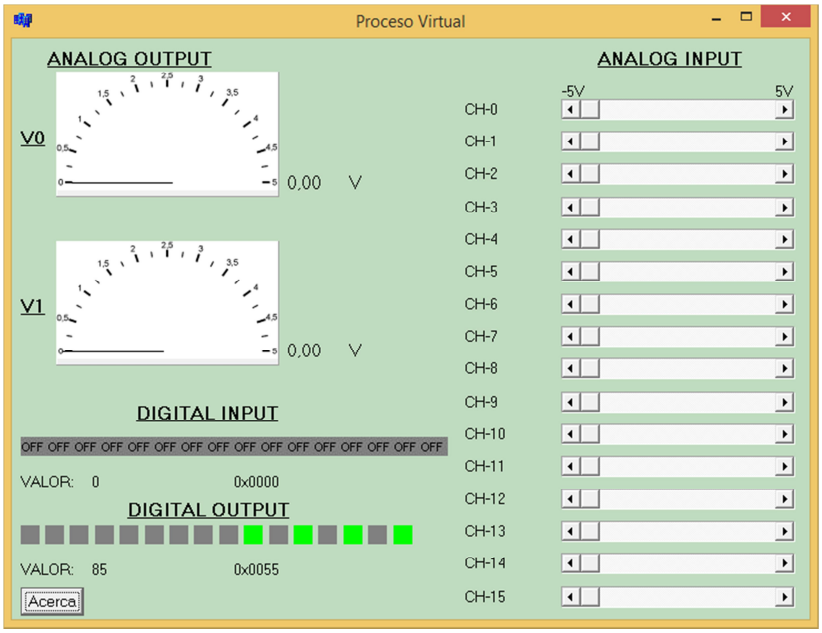

Imagen 20 – Salidas digitales impares activadas

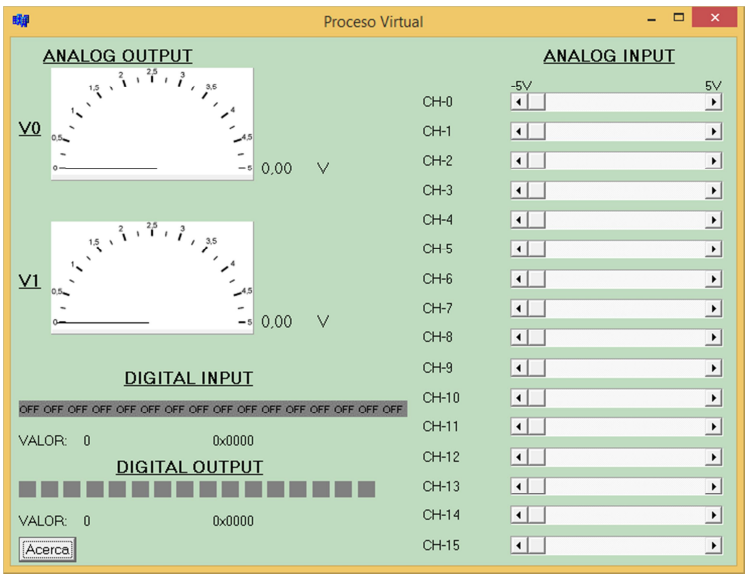

Imagen 21 – Salidas digitales impares desactivadas

Cuando en el programa del usuario se llama a la función que desconecta la tarjeta (Disconnect_Card), no se cierran las aplicaciones software lanzadas. Se podría conectar de nuevo la tarjeta ejecutando otra vez el programa de pruebas y estaría totalmente funcional (siempre y cuando la anterior se haya desconectado correctamente)

Para finalizar su utilización basta con cerrar la aplicación Proceso Virtual, con ello se cerrarán a su vez la Tarjeta virtual y el Timer Virtual, lanzados al comienzo de la sesión.