Sistemas Informático
Curso 2003-04

Drivers para tarjetas de adquisición de datos para Windows 2000

Primitivo Comendador Comendador
Jose Luis Ávila Asenjo

Dirigido por:
Prof. Jesús M. de la Cruz
DACYA

Facultad de Informática
Universidad Complutense de Madrid

ÍNDICE

ÍNDICE DE FUNCIONES

LISTA DE ACRÓNIMOS, ABREVIATURAS Y SÍMBOLOS

DLL Librerías de enlace dinámico.

DDK Driver Development Kit.

SDK Software Development Kit.

A/D Analógico/Digital.

D/A Digital/Analógico.

CA/D Entrada Analógica.

CD/A Salida Analógica.

N/D No disponible.

LSB Bits menos significativos.

MSB Bits más significativos.

DRDY Bit de dato listo.

PnP Plug and Play

INF Archivo de instalación

IRP paquete de peticion I/O

1. PRESENTACIÓN Y OBJETIVOS DEL PROYECTO

El objetivo principal del proyecto es el control de tarjetas de adquisición de dato (PCL-711B y PCL-812) bajo el entorno Windows 2000. El control y programación de estas tarjetas consiste básicamente es escrituras/lecturas de los registro de las tarjetas. Una vez instaladas las tarjetas en el PC, la lectura/escritura en estos registros se realiza mediante direcciones de memoria del espacio de direcciones del sistema operativo. Las direcciones de estos registros dependen de la configuración de la tarjeta, siendo todas ellas correlativas. Así, ambas tarjetas poseen 16 registros, con lo que contienen 16 direcciones de memoria consecutivas. Por ejemplo, en la configuración estándar de las tarjetas, la dirección inicial es la 220, con lo que el rango de direcciones de memoria para el control de la tarjeta será 220-22F.

En versiones anteriores a Windows 2000, como Windows98/Me, la política de seguridad permitía el acceso directo a direcciones de memoria, y al control de interrupciones. En un entorno de programación bajo Windows98/Me como por ejemplo Pascal o C++, mediante instrucciones de bajo nivel se permitía el acceso directo a memoria, así como la programación de las interrupciones. EL usuario final, por medio de estas instrucciones, puede programar y controlar la tarjeta.

En Windows 2000, y versiones posteriores, la nueva política de seguridad no permite el modo de operación anterior. Por lo tanto, para el control de las tarjetas en estos entornos, sería necesaria la utilización de un algún mecanismo para el control de las tarjetas. Concretamente, se pretendía el diseño de un drivers para Windows 2000, que si permite la lectura/escritura en direcciones de memoria (siempre y cuando sea un Kernel-driver).

Una vez diseñado el driver, el usuario final puede programar y controlar la tarjeta mediante el uso de este driver. A pesa de todo, el usuario final debía tener un amplio conocimiento sobre la funcionalidad de la tarjeta, registros utilizados y el modo de operar con ellos. Para solucionar este inconveniente, se propuso el desarrollo de una librería de enlace dinámico (DLL) para el control de la tarjeta. En esta DLL, se definen una serie de funciones que abarcan la funcionalidad básica de las tarjetas, permitiendo al usuario la programación de las tarjetas con un breve conocimiento sobre la funcionalidad de las mismas.

Finalmente, se propuso el desarrollo de una aplicación para el control interactivo de las tarjetas. Mediante esta aplicación, el usuario puede utilizar las funcionalidades de las tarjetas interactivamente, es decir, sin la necesidad de realizar programación alguna.

Para el desarrollo de este proyecto, era necesario el uso de un PC con el sistema operativo Windows 2000, las tarjetas de adquisición de datos, así como de otros elementos para el uso de las tarjetas (entrenador, osciloscopio, cables). Por su parte, para el desarrollo del driver y su posterior prueba era necesario tener permisos de administrador, además de para instalar las aplicaciones básicas para el desarrollo del driver (DDK, SDK). En un principio,

se nos asigno un PC en el laboratorio 7 de la facultad de informática. En este PC, no teníamos permisos de administrador, por lo que nos fue adjudicado u nuevo puesto en la facultad de físicas. En este PC, el sistema operativo instalado era Windows XP, por lo que el proyecto se ha desarrollado finalmente sobre esta plataforma. A pesar de ello, se ha creado también el driver correspondiente para Windows 2000. Los demás objetivos del proyecto (DLL y programa Control), son validos tanto para Windows 2000 como para Windows XP.

2. ESTRUCTURA DEL TRABAJO POR CAPÍTULOS

EL trabajo se ha llevado a cabo básicamente en cuatro fases, cada una de las cuales corresponde a un objetivo del proyecto. Para cada fase de desarrollo se ha incluido un capitulo donde se explica su desarrollo. Las cuatro fases o capítulos son los siguientes:

- **DESCRIPCIÓN DE LAS TARJETAS:** en esta fase se realizó un estudio de la funcionalidad y la forma de operar con las tarjetas. En este capitulo se incluye una breve descripción de las características de las tarjetas, así como de la forma de programar la tarjeta mediante la lectura/escritura de los registro de las mismas. Tras esta fase, adquirimos los conocimientos básicos para el manejo de las tarjetas, sin el cual no hubiera sido posible el desarrollo de los apartados siguientes.

- **DESARROLLO DEL DRIVER:** en este capitulo se incluye una introducción a la estructura de los drivers, y el modo de operación del sistema operativo con estos dispositivos. Paralelamente se ira explicando la estructura de nuestro driver, así como su funcionalidad. Por ultimo, se explicara la instalación del driver, así como su uso por parte del usuario final. En esta fase, hemos adquirido los conocimientos básicos sobre la estructura de un driver, su funcionalidad, así como de otras características tanto de los drivers como del sistema operativo Windows.

- **DESCRIPCIÓN DE LA DLL:** en este capítulo se incluye la descripción de las funciones de la DLL y su uso por parte del usuario final. Por ultimo se incluyen unas nociones de programación para el uso de la DLL y unos ejemplos de programas.

- **PROGRAMA CONTROL:** en este capítulo se explica la utilización del programa control, que permite al usuario interactuar con la tarjeta. Este programa permite las funcionalidades básicas de las tarjetas, como son las entradas/salidas digitales y analógicas.

4. CAPÍTULOS DE DESARROLLO Y RESULTADOS

4.1.- DESCRIPCIÓN DE LAS TARJETAS

4.1.1.- PC- LABCARD 711B

- CARACTERÍSTICAS:
    - 8 entradas analógicas con 12 bits de resolución en conversión A/D
    - Conversión disparada:
        - Por software
        - Por temporizador programable
        - Por señal externa
    - Nivel de interrupción IRQ programable para transferencia del dato A/D
    - 1 canal salida D/A de 12 bits de resolución con rango de salida 0-5 V ó 0-10 V
    - 16 entradas digitales
    - 16 salidas digitales

Especificaciones de la entrada analógica (CA/D)

Canales:	8 entradas single-ended
Resolución:	12 bits, conversión mediante aproximaciones sucesivas programable por software a: ±5V, ±2.5V, ±1.25V, ±0.625V y ±0.3125V
Conversor:	AD574 o equivalente
Tiempo de conversión:	25µs max.
Fiabilidad:	0.015% de la lectura ±1 LSB
No linealidad:	±1 bit
Modo de disparo:	Por software, por temporizador programable o por una señal externa
Transferencia de datos:	Por software o por interrupción
Nivel de IRQ:	IRQ2 a IRQ7
Sobrevoltaje:	±30V de continua

Especificaciones de la salida analógica (CD/A)

Canales:	1 salida analógica
Resolución:	12 bits
Rango de salida:	0-5V ó 0-10V
Conversor:	PM7548GP o equivalente
Tiempo de asentamiento:	30µs
Capacidad de salida:	±5mA max.
Voltaje de referencia:	Interna –5V y –10V (±0.05V)
No linealidad:	±1/2 LSB

Entradas digitales

Canales: 16 bits, compatibles TTL
Voltajes de entrada: Low → 0.8V max.
High → 2.0V min.
Carga de entrada: Low → 0.4mA max. @0.5V max.
High → 0.05mA max. @2.4V min.

Salidas digitales

Canales: 16 bits, compatibles TTL
Voltajes de salida: Low (sink) → 8mA @0.5V max.
High (source) → 0.4mA @2.4V mix.

- CONTROL DE LA TARJETA:

Mapa de direcciones de E/S

UBICACIÓN	LECTURA	ESCRITURA
Base + 0	Contador 0	Contador 0
Base + 1	Contador 1	Contador 1
Base + 2	Contador 2	Contador 2
Base + 3	N/D	Control del contador
Base + 4	A/D low byte	D/A low byte
Base + 5	A/D high byte	D/A high byte
Base + 6	D/I low byte	N/D
Base + 7	D/I high byte	N/D
Base + 8	N/D	“Clear” estado interrupción
Base + 9	N/D	Control de ganancia
Base + 10	N/D	Control multiplexor A/D
Base + 11	N/D	Control modo e interrupción
Base + 13	N/D	Disparador software A/D
Base + 14	N/D	D/O low byte
Base + 15	N/D	D/O high byte
Base + 16	N/D	N/D

Conversión A/D

- Registros de datos A/D (Base + 4, Base + 5)

Base + 4 Byte Low de datos del CA/D (Lectura)

D7	D6	D5	D4	D3	D2	D1	D0
AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0=LSB

Base + 5 Byte High de datos del CA/D (Lectura)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	DRDY	AD11=MSB	AD10	AD9	AD8

AD0-AD11 representan los bits de datos del menos significativo (LSB) al más (MSB). El bit DRDY (data ready bit) se pone a 1 cuando la conversión AD está en marcha y se pone a 0 cuando ha terminado. Se vuelve a poner a 1 cuando se lee el byte low de datos.

- Registro del multiplexor de entradas A/D (Base + 10)

Base + 10 Registro de control del multiplexor de los canales para A/D (Escritura)

D7	D6	D5	D4	D3	D2	D1	D0
-	-	-	-	-	C2	C1	C0

C2	C1	C0	Canal
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Se pueden multiplexar hasta 8 canales de entradas analógicas. El canal del que se desea realizar una entrada para conversión A/D se pone en este registro.

- Registro de control de modo y de interrupción (Base + 11)

La conversión A/D se puede disparar:

- Por software: escribiendo cualquier dato en el registro Base + 12, lo que genera un pulso de disparo al conversor A/D
- Por el temporizador de la tarjeta: la tarjeta lleva un Timer/Counter 8253 de Intel para generar salidas precisas de temporización. La frecuencia que se obtiene está entre 0.5MHz y 35 minutos por pulso.

- Por pulso externo: se puede usar una señal externa por la entrada D/I 0 de la tarjeta como pulso de disparo de la conversión. La conversión se inicia con el flanco de subida de la señal externa.

La tarjeta proporciona dos métodos de transferencia para el valor convertido por el A/D a una variable:

- Por control software (foreground): una vez se ha disparado al CA/D se testea el bit DRDY del registro Base + 5 hasta que se detecta que se ha puesto a 0. Entonces se leen por programa los registros Base + 4 y Base + 5 para obtener el dato.

- Por interrupción (background): cuando se ha completado la conversión A/D se produce una interrupción y debe ser la rutina de servicio de la interrupción la que se encargue de todo lo necesario para transferir el dato a las variables del programa.

A continuación vemos el formato del registro Base + 11 donde se especifican el modo de operación y el nivel de interrupción (IRQ):

Base + 11 Registro de control del modo y del nivel de interrupción (Escritura)

D7	D6	D5	D4	D3	D2	D1	D0
-	I2	I1	I0	-	S2	S1	S0

S2	S1	S0	Modo de operación
0	0	0	Disparo SW, transferencia SW
0	0	1	
0	1	0	Disparo externo, transferencia SW
0	1	1	Disparo externo, transferencia INT
1	0	0	Disparo temporizador, transferencia SW
1	0	1	Reservado
1	1	0	Disparo temporizador, transferencia INT
1	1	1	Reservado

I2	I1	I0	Nivel de interrupción
0	0	0	IRQ2
0	0	1	N/D
0	1	0	IRQ2
0	1	1	IRQ3
1	0	0	IRQ4
1	0	1	IRQ5
1	1	0	IRQ6
1	1	1	IRQ7

- Registro de estado de la interrupción (Base + 8)

Si la tarjeta está en el modo de transferencia de datos por interrupción se activa un bit de estado después de cada conversión. El usuario tiene que poner

a 0 dicho flag escribiendo cualquier dato en este registro de estado para permitir que la tarjeta acepte la siguiente interrupción.

- Registro de disparo software de la conversión (Base + 12)

Al escribir cualquier dato en este registro se dispara al conversor A/D.

Conversión D/A

Base + 4 Byte Low de datos del CD/A (Escritura)

D7	D6	D5	D4	D3	D2	D1	D0
DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0=LSB

Base + 5 Byte High de datos del CD/A (Escritura)

D7	D6	D5	D4	D3	D2	D1	D0
-	-	-	-	DA11=MSB	DA10	DA9	DA8

Se debe escribir primero el primer byte y después el segundo (Base + 5). La salida del CD/A no cambia hasta que no se actualiza el registro Base + 5.

Entradas y salidas digitales

Base + 6 Byte Low de datos D/I (Lectura)

D7	D6	D5	D4	D3	D2	D1	D0
DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0

Base + 7 Byte High de datos D/I (Lectura)

D7	D6	D5	D4	D3	D2	D1	D0
DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8

Base + 13 Byte Low de datos D/O (Escritura)

D7	D6	D5	D4	D3	D2	D1	D0
DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

Base + 14 Byte High de datos D/O (Escritura)

D7	D6	D5	D4	D3	D2	D1	D0
DO15	DO14	DO13	DO12	DO11	DO10	DO9	DO8

4.1.2.- PC- LABCARD 812

- CARACTERÍSTICAS:

- 16 entradas analógicas con 12 bits de resolución en conversión A/D
- Conversión disparada:
  - Por software
  - Por temporizador programable
  - Por señal externa
- Nivel de interrupción IRQ programable para transferencia del dato A/D
- 2 canales de salida D/A de 12 bits de resolución con rango de salida 0-5 V ó +/-10 V con referencia externa
- 16 entradas digitales
- 16 salidas digitales

Especificaciones de la entrada analógica (CA/D)

Canales:	16 entradas single-ended
Resolución:	12 bits, conversión mediante aproximaciones sucesivas
Rangos de entrada:	+/-1V, +/-2V, +/-5V, +/-10V,
Conversor:	AD574 o equivalente
Tiempo de conversión:	30KHz max.
Fiabilidad:	0.015% de la lectura ±1 LSB
No linealidad:	±1 bit
Modo de disparo:	Por software, por temporizador programable o por una señal externa
Transferencia de datos:	Por software o por interrupción
Nivel de IRQ:	IRQ2 a IRQ7
Sobrevoltaje:	±30V de continua

Especificaciones de la salida analógica (CD/A)

Canales:	2 salidas analógicas
Resolución:	12 bits
Rango de salida:	0-5V ó 0-10V con referencia externa
Conversor:	PM7548GP o equivalente
Tiempo de asentamiento:	30µs
Capacidad de salida:	±5mA max.
Voltaje de referencia:	Interna –5V y –10V (±0.05V)
No linealidad:	±1/2 LSB

Entradas digitales

Canales:	16 bits, compatibles TTL

Voltajes de entrada: Low → 0.8V max.
High → 2.0V min.
Carga de entrada: Low → 0.4mA max. @0.5V max.
High → 0.05mA max. @2.7V min.

Salidas digitales

Canales: 16 bits, compatibles TTL
Voltajes de salida: Low (sink) → 8mA @0.5V max.
High (source) → 0.4mA @2.4V mix.

- CONTROL DE LA TARJETA:

Mapa de direcciones de E/S

UBICACIÓN	LECTURA	ESCRITURA
Base + 0	Contador 0	Contador 0
Base + 1	Contador 1	Contador 1
Base + 2	Contador 2	Contador 2
Base + 3	N/D	Control del contador
Base + 4	A/D low byte	CH1 D/A low byte
Base + 5	A/D high byte	CH1 D/A high byte
Base + 6	D/I low byte	CH2 D/A low byte
Base + 7	D/I high byte	CH2 D/A high byte
Base + 8	N/D	“Clear” estado interrupción
Base + 9	N/D	N/D
Base + 10	N/D	Control multiplexor A/D
Base + 11	N/D	Control modo e interrupción
Base + 12	N/D	Disparador software A/D
Base + 13	N/D	D/O low byte
Base + 14	N/D	D/O high byte
Base + 15	N/D	N/D

Conversión A/D

- Registros de datos A/D (Base + 4, Base + 5)

Base + 4 Byte Low de datos del CA/D (Lectura)

D7	D6	D5	D4	D3	D2	D1	D0
AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0=LSB

Base + 5 Byte High de datos del CA/D (Lectura)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	DRDY	AD11=MSB	AD10	AD9	AD8

AD0-AD11 representan los bits de datos del menos significativo (LSB) al más (MSB). El bit DRDY (data ready bit) se pone a 1 cuando la conversión AD está en marcha y se pone a 0 cuando ha terminado. Se vuelve a poner a 1 cuando se lee el byte low de datos.

- Registro del multiplexor de entradas A/D (Base + 10)

Base + 10 Registro control del multiplexor de los canales para A/D (Escritura)

D7	D6	D5	D4	D3	D2	D1	D0
-	-	-	-	C3	C2	C1	C0

C3	C2	C1	C0	CH
0/1	0	0	0	0/8
0/1	0	0	1	1/9
0/1	0	1	0	2/10
0/1	0	1	1	3/11
0/1	1	0	0	4/12
0/1	1	0	1	5/13
0/1	1	1	0	6/14
0/1	1	1	1	7/15

Se pueden multiplexar hasta 8 canales de entradas analógicas. El canal del que se desea realizar una entrada para conversión A/D se pone en este registro.

- Registro de control de modo y de interrupción (Base + 11)

La conversión A/D se puede disparar:

- Por software: escribiendo cualquier dato en el registro Base + 12, lo que genera un pulso de disparo al conversor A/D
- Por el temporizador de la tarjeta: la tarjeta lleva un Timer/Counter 8253 de Intel para generar salidas precisas de temporización. La frecuencia que se obtiene está entre 0.5MHz y 35 minutos por pulso.
- Por pulso externo: JP1 en EXT y entrada 1 (EX.TRG) en CN5.

La tarjeta proporciona tres métodos de transferencia para el valor convertido por el A/D a una variable:

- Por control software (foreground): una vez se ha disparado al CA/D se testea el bit DRDY del registro Base + 5 hasta que se detecta que se ha puesto a 0. Entonces se leen por programa los registros Base + 4 y Base + 5 para obtener el dato.

- Por interrupción (background): cuando se ha completado la conversión A/D se produce una interrupción y debe ser la rutina de servicio de la interrupción la que se encargue de todo lo necesario para transferir el dato a las variables del programa.

- Por DMA

El modo de operación y el nivel de interrupción (IRQ) se especifican en el JP4. Poniendo JP4 a X, al final de la cuenta del temporizador inicia la conversión y el programa. Testeando DRDY puede determinar el final de la conversión (DRDY a 0).

Base + 11 Registro de control del modo y del nivel de interrupción (Escritura)

D7	D6	D5	D4	D3	D2	D1	D0
-	-	-	-	-	S2	S1	S0

A. Con condición de disparo interno (JP1: INT con TRG)

S2	S1	S0	Modo de operación
0	0	0	Inhibe disparo SW y por temporizador
0	0	1	Permite sólo disparo SW
0	1	0	Permite sólo disparo por temporizador usando DMA
1	1	0	Permite disparo por temporizador usando temporizador y transferencia SW con JP4 a X, o por interrupción según JP4

B. Con condición de disparo externo (JP1: EXT con TRG)

S2	S1	S0	Modo de operación
0	0	X	Inhibe disparo SW y por temporizador
0	1	0	Permite sólo disparo externo usando DMA
1	1	0	Permite disparo externo usando transferencia SW o por interrupción

- Registro de estado de la interrupción (Base + 8)

Si la tarjeta está en el modo de transferencia de datos por interrupción se activa un bit de estado después de cada conversión. El usuario tiene que poner

a 0 dicho flag escribiendo cualquier dato en este registro de estado para permitir que la tarjeta acepte la siguiente interrupción.

- Registro de ganancia programable (Base + 9)

En este modelo de tarjeta (812) la selección del rango +/-5V, +/-10V se hace mediante un switch.

- Registro de disparo software de la conversión (Base + 12)

Al escribir cualquier dato en este registro se dispara al conversor A/D.

Conversión D/A

Base + 4 Byte Low de datos del CD/A (Escritura)

D7	D6	D5	D4	D3	D2	D1	D0
DA7	DA6	DA5	DA4	DA3	DA2	DA1	DA0=LSB

Base + 5 Byte High de datos del CD/A (Escritura)

D7	D6	D5	D4	D3	D2	D1	D0
-	-	-	-	DA11=MSB	DA10	DA9	DA8

Se debe escribir primero el primer byte y después el segundo (Base + 5). La salida del CD/A no cambia hasta que no se actualiza el registro Base + 5. Sería igual para el caso del otro canal (CH2).

Entradas y salidas digitales

Base + 6 Byte Low de datos D/I (Lectura)

D7	D6	D5	D4	D3	D2	D1	D0
DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0

Base + 7 Byte High de datos D/I (Lectura)

D7	D6	D5	D4	D3	D2	D1	D0
DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8

Base + 13 Byte Low de datos D/O (Escritura)

D7	D6	D5	D4	D3	D2	D1	D0
DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

Base + 14 Byte High de datos D/O (Escritura)

D7	D6	D5	D4	D3	D2	D1	D0
DO15	DO14	DO13	DO12	DO11	DO10	DO9	DO8

4.3.- DESCRIPCIÓN DEL DRIVER.

En este capitulo se incluye una introducción al sistema I/O de Windows 2000, que es el encargado de la conexión entre dispositivos hardware y sistema por medio de dirvers. En este apartado, se describen los componentes que conforma este sistema, como interactúan entre si para el manejo de dispositivos hardware, y como el usuario a través de este sistema puede manejar tales dispositivos.

En nuestro caso, para el control de las tarjetas necesitamos acceder a direcciones de memoria para lectura/escritura, además de manejar las interrupciones producidas por las tarjetas. En una primera fase de desarrollo, se ha creado un driver para lectura/escritura en memoria, sin el manejo de las interrupciones. Tras la elaboración de este driver, se propuso la implementación de la DLL y del programa Control, dejando en último lugar el manejo de las interrupciones por parte del driver. Este último apartado no se ha podido llevar a cabo, por lo que básicamente nuestro driver solo soporta lectura/escritura en memoria.

Para la elaboración de nuestro driver se ha utilizado la plataforma DDK (Driver Development Kit) junto con su documentación. Esta plataforma incluye numerosos ejemplos de drivers, de los cuales hemos utilizado uno como esqueleto para nuestro propio driver. Concretamente estamos hablando del driver Portio (Generic Port I/O Device Driver), que es un driver PnP que permite la lectura/escritura tanto de memoria, como de registros del procesador. Por consiguiente, hemos desarrollado un driver PnP siguiendo este ejemplo. Los drivers PnP y otros tipos de drivers se explicarán posteriormente, así como las rutinas y funcionalidad del driver.

En resumen, en este apartado hemos desarrollado un driver PnP (llamado PCLDriver), para lectura/escritura en direcciones de memoria. Estas direcciones de memoria han sido reservadas por el driver mediante su instalación con un fichero INF.

4.3.1.- SISTEMA I/O DE WINDOWS 2000.

El sistema I/O de Windows 2000 consiste en varios componentes para el manejo de los dispositivos hardware proporcionando una interfaz para aplicaciones y el propio sistema. El sistema I/O esta formado por varios componentes, así como de dispositivos drivers, tal y como muestra la Figura 1.

- **I/O manager:** conecta las aplicaciones y componentes del sistema con dispositivos virtuales, lógicos y físicos, y define la infraestructura que soportan los dispositivos drivers.

- **Dispositivos drivers:** proporcionan una interfaz I/O para un determinado dispositivo. Los dispositivos drivers reciben comandos enviados por el I/O manager, e informan al I/O manager cuando ese comando ha sido completado.

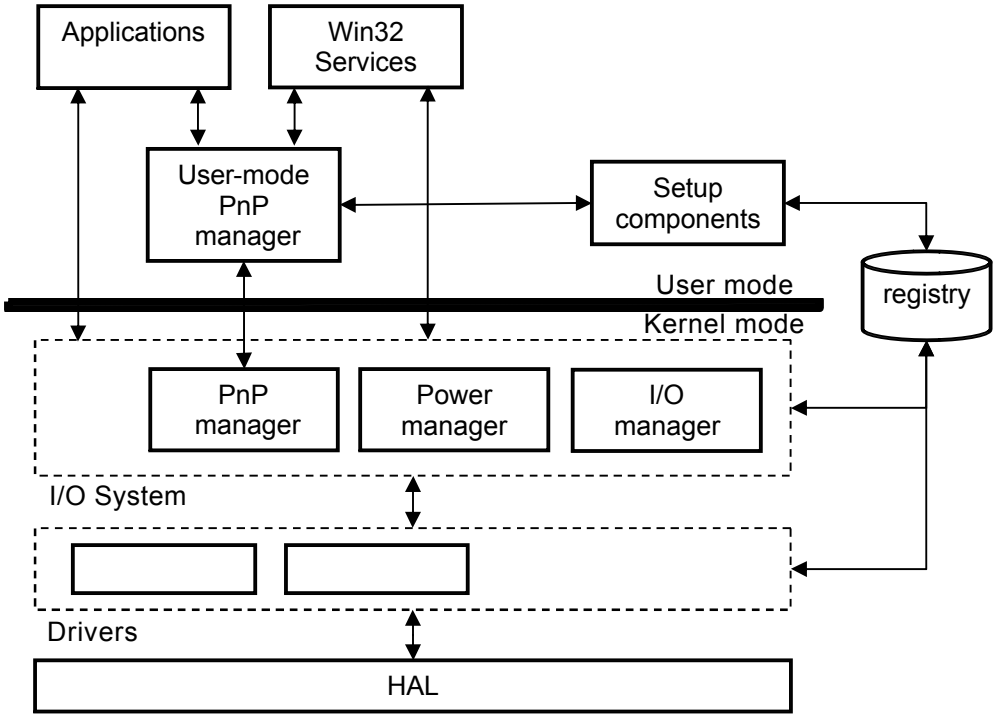


Figura 1: Componentes del sistema I/O.

- **PnP manager:** trabaja junto con el I/O manager y un tipo de drivers llamados bus-drivers. El PnP manager y los bus-drivers son los responsables de cargar el driver adecuado cuando el dispositivo ha sido detectado.

- **Power manager:** trabaja junto con el I/O manager para guiar al sistema, así como a un dispositivo driver, a través de los estados de energía.

- **Registro:** sirve como una base de datos que almacena una descripción de los dispositivos hardware instalados en el sistema, así como su inicialización y opciones de configuración.

4.3.1.1.- I/O MANAGER.

El I/O Manager define el modo con que las peticiones I/O son enviadas a los dispositivos drivers. La mayoría de las peticiones I/O son representadas mediante un I/O request packet (IRP), que son enviadas de un componente de sistema I/O a otro. Una IRP es una estructura que contiene una completa información de una petición I/O.

El I/O Manager crea una IRP que representa una operación I/O, pasando un puntero a IRP al driver correcto y dispone del paquete cuando la operación I/O esta completada. Por otro lado, un driver recibe una IRP, realiza la operación especificada por la IRP y devuelve la IRP al I/O Manager, para su

finalización o para ser enviada a otro driver. Este proceso se puede observa en la Figura 2.

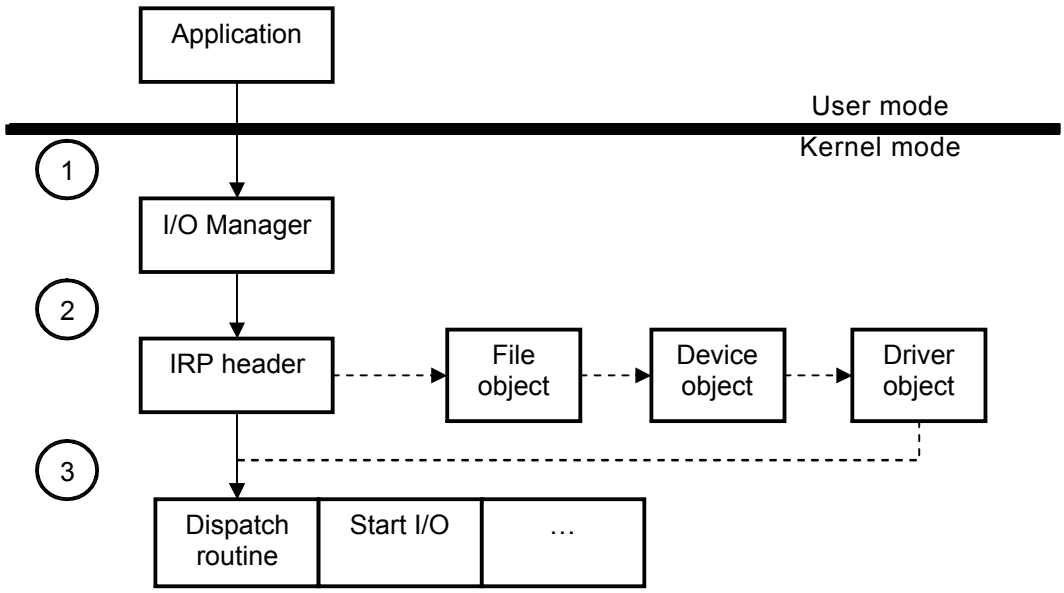


Figura 2: Petición I/O.

Una aplicación, realiza una llamada al sistema I/O pasando un descriptor de fichero al I/O manager (1). El I/O manager crea una IRP (2) y usa el objeto driver para localizar la rutina correspondiente, pasándole la IRP.

4.3.1.2.- DISPOSITIVOS DRIVERS.

Para integrar el I/O Manager con otros componentes de sistemas I/O, un dispositivo driver debe implementar las directivas especificadas por el tipo de dispositivo que maneja y el papel que desempeña el driver. A continuación veremos los diferentes tipos de drivers soportados por Windows 2000, así como su estructura.

Windows 2000 soporta un amplio rango de diferentes dispositivos drivers. En este apartado, nos dedicaremos a los tipos de drivers que trabajan en modo Kernel, que pueden ser divididos en las siguientes categorías, según vemos en la Figura 3.

- **Dispositivos Drivers Virtuales (VDD):** son componente que trabajan en modo usuario y permiten a las aplicaciones basadas en DOS acceder al hardware en plataformas x86.

- **Driver de sistemas de ficheros:** que implementan el modelo estándar de sistema de ficheros del PC.

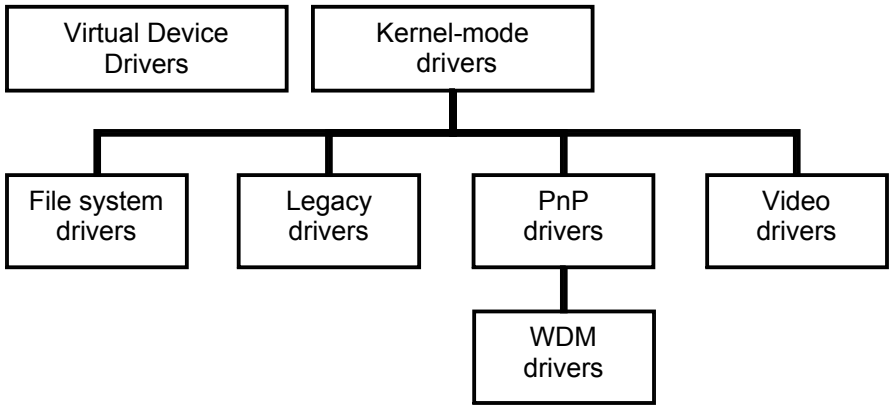


Figura 3: Tipos de drivers en Windows 2000.

- **Drivers PnP:** driver que entiende el protocolo Plug and Play de Windows 2000.

- **Dispositivo Drivers Heredados:** son kernerl-drivers que controlan directamente el hardware sin la ayuda de otros drivers. Esta categoría incluye los driver para versiones anteriores de Windows NT que se ejecutan sin cambios en Windows 2000.

- **Drivers de video:** dispositivos kernel para la visualización e impresión de datos.

- **Driver WDM:** es un driver PnP que también entiende el protocolo de energía y es compatible con lo sistemas Windows 98 y Windows 2000.

Como hemos dicho anteriormente, nuestro driver se trata de un driver PnP, por lo que además de la posibilidad de lectura/escritura de memoria, nuestro driver ha de implementar el protocolo PnP. Esto se vera más adelante.

4.3.1.3.- ESTRUCTURA DE UN DRIVER.

Los dispositivos drivers consisten en un conjunto de rutinas que son llamador por diferentes estados en una petición I/O. Algunas de estas rutinas, las más importantes, se describen a continuación.

- **Rutina de inicialización:** el I/O Manager ejecuta una rutina de inicialización, normalmente llamada DriverEntry, cuando carga el driver en el sistema operativo. Esta rutina se encarga de rellenar las estructuras de datos necesarias para registrar el resto de rutinas del drivers.

- **Rutina add-device:** esta rutina ha de ser implementada por los dispositivos driver que soporten Plug and Play. En esta rutina, el driver inicializa un objeto device que representa al dispositivo.

- **Rutinas dispatch:** estas rutinas son las funciones propias del driver, es decir, la funcionalidad que el driver ofrece. Algunos ejemplos son la lectura o escritura en sistemas de fichero o en memoria. Cuando se produce una llamada a una operación I/O, el I/O Manager genera una IRP y llama al driver a través de alguna de estas rutinas.

- **Rutinas Start I/O:** el driver puede utilizar una rutina start I/O para realizar una transferencia de un dato a otro dispositivo. Esta rutina es definida solo para los drivers que confían en el I/O Manager para la serialización de las IRP.

- **Rutinas I/O Completion:** estas rutinas notifican al sistema el resultado de la operación realizada por el dispositivo driver tras haber recibido una IRP.

- **Rutinas de Cancelación:** si una operación puede ser cancelada, un driver debe definir una o más rutina de cancelación. Cuando un driver recibe una IRP que puede ser cancelada, asigna a la IRP la rutina de cancelación.
- **Rutina de descarga:** esta rutina es la encargada de liberar los recursos de un drivers en uso, para que así el I/O manager pueda liberarlo de memoria.

4.3.1.4.- ESTRUCTURAS DE DATOS I/O.

Hay tres estructuras primarias I/O: objeto file, objeto driver, objeto device. Cada una de estas estructuras está definida en el fichero de cabecera Ntddk.h de la plataforma DDK.

- **Objeto file:** son los descriptores de ficheros o dispositivos en el modo kernel. Cuando se abre un fichero o un simple dispositivo, el I/O Manager devuelve un descriptor al objeto fichero.

- **Objeto driver:** representa un driver en el sistema. El I/O manager obtiene la dirección de cada rutina del driver a partir de este objeto.

- **Objeto device:** representa un dispositivo lógico o físico del sistema y describe sus características.

El I/O manager crea un objeto driver cuando un driver es cargado en el sistema, y es cuando llama a la rutina de inicialización del driver (por ejemplo, DriverEntry), que se encarga de inicializar el objeto con los valores de las demás direcciones de las rutinas del driver.

Después de cargar un driver, el driver puede crear un objeto device para representar al dispositivo, mediante una llamada a IoCreateDevice. Sin embargo, la mayoría de los driver realizan esta operación mediante la rutina add-device, que es invocada cuando el PnP manager localiza el dispositivo.

4.3.2.- USO DEL DRIVER.

Una vez explicado la forma de operar el sistema con los drivers, y la estructura de los mismos, vamos a ver la instalación del driver, así como su uso por parte del usuario final.

4.3.2.1.- INSTALACIÓN DEL DRIVER.

- **Windows 2000:**

  - Doble click en “Añadir Nuevo Hardware” en el panel de control.
  - Selecciona “Next”.
  - Selecciona “No, seleccionaré el hardware de una lista”.
  - Selecciona “Otros dispositivos” y click en “Next”.
  - Click en “Usar disco” y selecciona el directorio que contiene el archivo inf.

- **Windows XP:**

  - Doble click en “Añadir Nuevo Hardware” del panel de control
  - En el cuadro de bienvenida selecciona “Next”
  - Selecciona “Si, ya he conectado el hardware”, y pulsa “Next”
  - Selecciona “Agregar un nuevo dispositivo de hardware” de la lista, y pulsa “Next”
  - Selecciona “Instalar el hardware seleccionado de una lista (avanzado)”, y pulsar “Next”
  - Selecciona mostrar todos los dispositivos, y pulsa “Next”
  - Click en “Usar disco” y selecciona el directorio que contiene el archivo inf.

4.3.2.2.- USO DEL DRIVER.

- **Apertura del driver:** una vez instalado el driver, este puede ser usado por el usuario. Primeramente ha de abrir el dispositivo driver, mediante una llamada al sistema (CreateFile), que devuelve un descriptor del driver. Así, para nuestro driver, el código para la apertura del driver sería:

```
hndFile = CreateFile("\\\\.\\PCLDriver",
          GENERIC_ALL, FILE_SHARE_WRITE, NULL,
          OPEN_EXISTING, 0, NULL);
if (hndFile == INVALID_HANDLE_VALUE)
     return -1;
else {
     return 0;
}
```

- **Lectura/escritura en memoria:** una vez abierto el driver, el usuario puede leer direcciones de memoria mediante el uso del driver, pasándole un mensaje. El código para las operaciones de lectura/escritura sería:

```
// Lectura de memoria
// Código IOCTL para la operación de lectura.
IoctlCode = IOCTL_PCL_READ_PORT_UCHAR;
DataLength = sizeof(byte);
ULONG PortNumber = 4; // Direccion Base + 4
// Paso de mensaje al driver.
IoctlResult = DeviceIoControl(hndFile, IoctlCode,
&PortNumber, sizeof(PortNumber),
&byte, DataLength, &ReturnedLength, NULL);
if (IoctlResult) {
   dato.dato = byte;
return EXITO;
} else {
    return ERROR_LECTURA_BYTE;
}

// Escritura de memoria
InputBuffer.PortNumber = 4; Direccion base + 4
DataValue = 256;
// Codigo IOCTL para la operación de lectura.
IoctlCode = IOCTL_PCL_WRITE_PORT_UCHAR;
InputBuffer.CharData = (UCHAR)DataValue;
DataLength = offsetof(PCL_WRITE_INPUT, CharData) +
                      sizeof(InputBuffer.CharData);
// Paso de mensaje al driver.
IoctlResult = DeviceIoControl(hndFile, IoctlCode,
              &InputBuffer, DataLength, NULL, 0,
              &ReturnedLength, NULL);
if (IoctlResult)
     return EXITO;
else
     return ERROR_ESCRITURA_BYTE;
```

4.2.- DLLs PARA CONTROL DE TARJETAS PCL 711B Y 812.

4.2.1.- INTRODUCCIÓN.

El driver para tarjetas de adquisición de datos PCLDriver incluye unas librerías de enlace dinámico (DLLs), que contienen un conjunto de funciones y estructuras asociadas que pueden ser usadas en diferentes entornos de programación (Microsoft Visual C++, C++ Builder) , permitiendo así el control de la tarjeta por parte del usuario final. Concretamente, se incluye dos DLLs:

- DLLVisual: DLL para el entorno de programación de Microsoft Visual C++.

- DLLBorland: DLL para el entorno de programación de C++ Builder.

A pesar de que la funcionalidad de ambas DLLs es la misma, ha sido necesaria la creación de sendas DLLs por que los entornos de programación utilizan diferentes DLLs. A partir de ahora, nos referiremos a una sola DLL, ya que el código es el mismo para ambos casos.

La funcionalidad de la DLL se puede agrupar en cinco grupos, cada uno de los cuales se encarga de una funcionalidad diferente.

- **Funciones de dispositivo:** funciones para la apertura y cierre del driver, así como de escritura/lectura en puerto de memoria.

- **Entradas digitales:** funciones para el control de entradas digitales.

- **Salidas digitales:** funciones para el control de salidas digitales.

- **Entradas analógicas:** funciones para el control de entradas analógicas. Implementa funciones para la conversión analógico/digital.

- **Salidas analógicas:** funciones para el control de salidas analógicas. Implementa funciones para la conversión digital/analógico.

Además, la DLL utiliza una serie de estructuras y constantes utilizadas en la funciones, para hacer más legible su uso. Tanto las funciones, como las estructuras y constantes utilizadas se definen a continuación.

4.2.2.- ESTRUCTURAS DLLs.

- Registro para lectura/escritura de un byte en memoria.

```
typedef struct {
     UCHAR puerto;
```

```
     UCHAR dato;
} ByteDato;
```

Donde Base + puerto es la dirección de memoria sobre la que se realiza la operación le lectura/escritura, y dato es el dato que se lee o escribe en memoria,

- Registro para la lectura/escritura de un entrada/salida digital.

```
typedef struct {
     UCHAR numero;
     BOOL dato;
} BitDato;
```

Donde número es el número de entrada digital que se lee o escribe, y dato es el dato leído o escrito.

4.2.3.- CONSTANTES DLLs.

- Tipos de tarjetas.

PCL711B - Tarjeta PCL-711B.
PCL812 - Tarjeta PCL-812.

- Canales de entrada

PCL_AI_0 - Canal de entrada 0.
PCL_AI_1 - Canal de entrada 1.
PCL_AI_2 - Canal de entrada 2.
PCL_AI_3 - Canal de entrada 3.
PCL_AI_4 - Canal de entrada 4.
PCL_AI_5 - Canal de entrada 5.
PCL_AI_6 - Canal de entrada 6.
PCL_AI_7 - Canal de entrada 7.
PCL_AI_8 - Canal de entrada 8. Solo para PCL-812.
PCL_AI_9 - Canal de entrada 9. Solo para PCL-812.
PCL_AI_10 - Canal de entrada 10. Solo para PCL-812.
PCL_AI_11 - Canal de entrada 11. Solo para PCL-812.
PCL_AI_12 - Canal de entrada 12. Solo para PCL-812.
PCL_AI_13 - Canal de entrada 13. Solo para PCL-812.
PCL_AI_14 - Canal de entrada 14. Solo para PCL-812.
PCL_AI_15 - Canal de entrada 15. Solo para PCL-812.

- Valores de ganancia

PCL_GANANCIA_X1 - ganancia x 1. Solo para PCL-711B.
PCL_GANANCIA_X2 - ganancia x 2. Solo para PCL-711B.
PCL_GANANCIA_X4 - ganancia x 4. Solo para PCL-711B.
PCL_GANANCIA_X8 - ganancia x 8. Solo para PCL-711B.
PCL_GANANCIA_X16 - ganancia x 16. Solo para PCL-711B.

- Valores del control

PCL_DISPARO_SW_MODO_SW - Solo para PCL-711B.
PCL_DISPARO_EX_MODO_SW - Solo para PCL-711B.
PCL_DISPARO_EX_MODO_INT - Solo para PCL-711B.
PCL_DISPARO_TEMP_MODO_SW - Solo para PCL-711B.
PCL_DISPARO_TEMP_MODO_INT - Solo para PCL-711B.
PCL_DISPARO_SW - Solo para PCL-812.
PCL_DISPARO_TEMP - Solo para PCL-812.
PCL_DISPARO_EXT - Solo para PCL-812.

- Modo del contador

  DISPARO_UNICO - Un solo disparo.
  DISPARO_MULTIPLE - Generador de pulso.

4.2.4.- FUNCIONES DLLs.

Funciones de dispositivo:

- `PCL_InicializarDriver(HANDLE &hndFile, UCHAR tipoTarjeta)`

  Descripción: Abre el dispositivo para su posterior manejo por la DLL.

  Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	OUT	HANDLE	Descriptor del dispositivo para su posterior manejo por la DLL.
tipoTarjeta	IN	UCHAR	Tipo de tarjeta a controlar. Ver constantes DLLs.

  Valor devuelto:

  EXITO si todo correcto.
  ERROR_APERTURA_DRIVER si no se puede abrir el dispositivo.

- `PCL_CerrarDriver(HANDLE hndFile)`

  Descripción: Cierra el dispositivo después de su manejo por la DLL.

  Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.

Valor devuelto:

EXITO si todo correcto.
ERROR_CIERRE_DRIVER si no se puede cerrar el dispositivo.

- `PCL_LeerBytePuerto(HANDLE hndFile, ByteDato &dato)`

Descripción: Lee un byte de la dirección de memoria Base + puerto, donde el valor Base la dirección inicial de la tarjeta.

Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.
dato	OUT	ByteDato	dato.puerto es el puerto de memoria a leer y dato.dato es el valor leído desde memoria.

Valor devuelto:

EXITO si todo correcto
ERROR_LECTURA_BYTE si no se ha podido leer el dato de memoria.

- `PCL_EscribirBytePuerto(HANDLE hndFile, ByteDato dato)`

Descripción: Escribe un byte en la dirección de memoria Base + puerto, donde el valor Base es la dirección inicial de la tarjeta.

Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.
dato	IN	ByteDato	dato.numero es el puerto de memoria a escribir y dato.dato el valor a escribir en memoria.

Valor devuelto:

EXITO si todo correcto
ERROR_ESCRITURA_BYTE si no se ha podido escribir el dato de memoria.

Entradas Digitales:

- `UCHAR PCL_LeerEntradaDigital(HANDLE hndFile, BitDato &dato)`

  Descripción: Lee el valor de la entrada digital número n, indicada por dato.numero. El valor leído es devuelto en dato.dato.

  Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.
Dato	IN/OUT	BitDato	dato.numero es la entrada digital a leer y dato.dato es el valor leído de la entrada.

  Valor devuelto:

  EXITO si todo correcto
  ERROR_DI_FUERA_RANGO si no existe la entrada digital.
  ERROR_LECTURA_DIGITAL si no se ha podido leer el bit especificado.

- `UCHAR PCL_LeerPuertoDigital(HANDLE hndFile, ByteDato &dato)`

  Descripción: Lee el valor del puerto digital número n (ocho entradas digitales), indicado por dato.puerto. El valor leído es devuelto en dato.dato.

  Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.
dato	IN/OUT	ByteDato	dato.puerto es el puerto digital a leer y dato.dato es el valor leído.

  Valor devuelto:

  EXITO si todo correcto.
  ERROR_PUERTO_FUERA_RANGO si no existe el puerto.
  ERROR_LECTURA_DIGITAL si no se ha podido leer la entrada.

Salidas Digitales:

- `UCHAR PCL_EscribirSalidaDigital(HANDLE hndFile, BitDato dato)`

  Descripción: Establece el valor de la salida digital n, indicada por dato.numero, al valor indicado por dato.dato.

Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.
dato	IN	BitDato	dato.numero es la salida digital a modificar con el valor dato.dato.

Valor devuelto:

EXITO si todo correcto
ERROR_DO_FUERA_RANGO si no existe la salida digital.
ERROR_ESCRITURA_DIGITAL si no se ha podido escribir el bit.

- `UCHAR PCL_EscribirPuertoDigital(HANDLE hndFile, ByteDato dato)`

Descripción: Escribe en el puerto digital número n (ocho salidas digitales), indicado por dato.puerto, el valor indicado por dato.dato.

Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.
dato	IN	ByteDato	dato.dato es el valor a escribir en el puerto de memoria indicador por dato.puerto.

Valor devuelto:

EXITO si todo correcto
ERROR_PUERTO_FUERA_RANGO si no existe el puerto.
ERROR_ESCRITURA_DIGITAL si no se ha podido escribir la salida.

Entradas Analógicas:

- `UCHAR PCL_SetEntradaAnalogica(HANDLE hndFile, UCHAR entrada)`

Descripción: Selecciona el canal de entrada analógico para la conversión analógico/digital.

Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.
Entrada	IN	UCHAR	Canal de entrada analógica que se selecciona para la conversión. Ver constantes DLLs.

Valor devuelto:

EXITO si todo correcto
ERROR_AI_FUERA_RANGO si no existe el canal analógico.
ERROR_ENTRADA_ANALOGICA si no se ha podido seleccionar el canal.

- `UCHAR PCL_SetGananciaAD(HANDLE hndFile, UCHAR ganancia)`

Descripción: Establece la ganancia o amplificación en la conversión analógico/digital. Esta función solo es valida para la tarjeta PCL-711B.

Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.
ganancia	IN	UCAHR	Ganancia o amplificación en la conversión. Ver constantes DLLs.

Valor devuelto:

EXITO si todo correcto
ERROR_FUNCION_NO_VALIDA si la tarjeta no soporta amplificación.
ERROR_GANANCIA_FUERA_RANGO si la ganancia no es valida.
ERROR_GANANCIA si no se ha podido establecer la amplificación.

- `UCHAR PCL_SetModoControl(HANDLE hndFile, UCHAR modo)`

Descripción: Establece el modo de control de la conversión analógico/digital. Es decir, modo de disparo de la conversión, y como se lee el dato.

Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.
modo	IN	UCAHR	Modo de control de la tarjeta. Ver constantes DLLs

Valor devuelto:

EXITO si todo correcto
ERROR_MODO_NO_VALIDO si el modo de control no es valido.
ERROR_MODO_CONTROL si no se ha podido establecer el modo de control.

- `UCHAR PCL_ComenzarConversion(HANDLE hndFile)`

  Descripción: Inicia la conversión analógico/digital cuando el modo de disparo de la tarjeta es disparo software.

  Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.

  Valor devuelto:

  EXITO si todo correcto.
  ERROR_COMIENZO_CONVERSION si no se ha podido iniciar la conversión.
  ERROR_DISPARO_SOFTWARE el modo de control no permite disparo software.

- `UCHAR PCL_LeerDatoAD(HANDLE hndFile, ULONG &datoConvertido)`

  Descripción: Lee el dato convertido. Es decir, intenta leer el dato: si el dato esta listo, se devuelve el valor convertido, si no esta listo se produce un error.

  Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.
datoConvertido	OUT	ULONG	Valor del dato convertido.

  Valor devuelto:

  EXITO si todo correcto.
  ERROR_DATO_NO_LISTO si el dato convertido no esta listo.

- `UCHAR PCL_ProgramaContadorC1C2(HANDLE hndFile, ULONG c1, ULONG c2, UCHAR modo)`

  Descripción: Programa el contador interno de la tarjeta con los valores c1 y c2. Tiene dos modos de funcionamiento: un solo disparo, o disparos continuados. El tiempo del contador interno viene dado por la formula siguiente:

  $$\text{Frecuencia} = (2\ \text{MHz}) / (C1 \times C2) \qquad (1)$$

  Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.
c1	IN	ULONG	Valor para C1 en la formula (1)
c2	IN	ULONG	Valor para C2 en la formula (1)
modo	IN	UCAHR	Modo del contador. Ver constantes DLLs.

Valor devuelto:

EXITO si todo correcto
ERROR_PROGRAMA_CONTADOR si no se ha podido programar el contador.

- `UCHAR PCL_ProgramaContadorTiempo(HANDLE hndFile, ULONG mseg, UCHAR modo)`

Descripción: Programa el contador interno de la tarjeta con el tiempo indicado por mseg. Tiene dos modos de funcionamiento, igual que la función anterior.

Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.
mseg	IN	ULONG	Tiempo en mseg para el temporizador interno de la tarjeta,
modo	IN	UCHAR	Modo del contador. Ver constantes DLLs.

Valor devuelto:

EXITO si todo correcto
ERROR_PROGRAMA_CONTADOR si no se ha podido programar el contador.

Salidas Analógicas:

- `UCHAR PCL_ConversionDA(HANDLE hndFile, UCHAR canal, USHORT datoDigital)`

Descripción: Convierte de digital a analógico el dato de entrada datoDigital. El valor analógico se toma por el canal indicado.

Parámetros:

Nombre	Dirección	Tipo	Descripción
hndFile	IN	HANDLE	Descriptor del dispositivo utilizado por la DLL.
canal	IN	UCHAR	Canal de salida por el que se toma el valor convertido.
datoDigital	IN	USHORT	El dato digital a convertir.

Valor devuelto:

EXITO si todo correcto.
ERROR_AO_FUERA_RANGO si no existe el canal de salida.
ERROR_DA_FUERA_RANGO si el dato a convertir no es valido.
ERROR_CONVERSION_DA si no se ha podido convertir datoDigital.

4.2.5.- USANDO LAS DLLs.

Como hemos dicho anteriormente, las DLLs pueden ser utilizadas tanto en Microsoft Visual C++, como en C++ Builder. En el disco de instalación del driver, se han incluido las dos DLLs junto con sus respectivas librerías, así como el archivo PCLDLL.h. Este archivo importa las funciones, variables y estructuras utilizadas por las DLLs, de manera que incluyendo este archivo a nuestro proyecto podemos utilizar las funciones de la DLLs sin necesidad de importarlas explícitamente. Seguidamente se detallarán los pasos a seguir para utilizar las DLLs en ambos lenguajes, así como de unos fundamentos de programación.

DLL para Microsoft Visual C++.

Para utilizar las funciones de la DLL para Microsoft Visual C++, debes usar la DLLVisual, siguiendo los siguientes pasos.

1. Click File | New del menú principal para crear un proyecto nuevo.

2. Define el tipo de proyecto, define la plataforma como “Win32” y selecciona un directorio para el proyecto.

3. Una vez creado el proyecto, copia la DLL con su correspondiente archivo de librería (DLLVisual.dll y DLLVisual.lib) y el archivo PCLDLL.h al directorio de trabajo.

4. Incluye el archivo PCLDLL.h al proyecto. Para ello, selecciona Project | Add to Project | Files… del menú principal.

5. Incluye el archivo de librería DLLVisual.lib al proyecto. Para ello, selecciona Project | Settings… del menú principal.

6. Selecciona la página Link, e inserta DLLVisual.lib en Object/Library modules. Click en el botón de OK.

DLL para C++ Builder

Para utilizar las funciones de la DLL para C++ Builder, debes usar la DLLBorland, siguiendo los siguientes pasos.

1. Click File | New Application para crear una nueva aplicación.

2. Una vez creado el proyecto, copia la DLL con su correspondiente archivo de librería (DLLBorland.dll y DLLBorland.lib) y el archivo PCLDLL.h al directorio de trabajo.

3. Incluye el archivo PCLDLL.h al proyecto. Para ello, selecciona Project | Add to Project del menú principal.
4. Incluye el archivo de librería DLLVisual.lib al proyecto. Para ello, selecciona Project | Add to Project del menú principal, y selecciona el archivo DLLBorland.lib.

Fundamentos de programación

En la Figura 4 podemos observar la forma normal de operar con la DLL. La función PCL_InicializarDriver inicializa el dispositivo driver para su posterior manejo por la DLL. Por contra, la función PCL_CerrarDriver cierra el dispositivo previamente abierto. Device Handle es el descriptor de dispositivo driver devuelto por PCL_InicializarDriver, que será utilizado por las demás funciones de la DLL. En el bloque de instrucciones, se incluye el código para el control de la tarjeta, es decir, las funciones de la DLL para el manejo de la tarjeta.

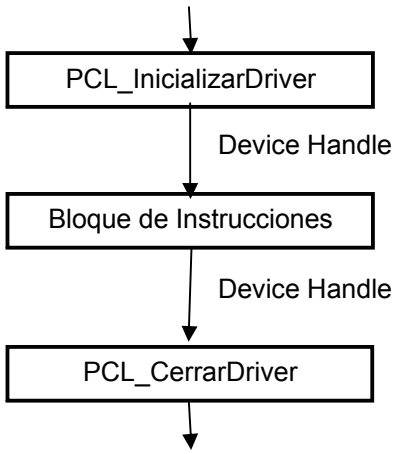


Figura 4: Uso de la DLLs.

En el Anexo A se incluye el código fuente de algunos ejemplos de programas que siguen este esquema, donde se puede observar la lectura/escritura digital y analógica. También se adjunta los proyectos en Microsoft Visual C++, donde se han implementados estos ejemplos.

4.4.- PROGRAMA CONTROL

4.4.1.- MENÚ INICIAL

Una vez ejecutamos la aplicación entramos en el menú principal, desde donde podremos acceder a las diferentes funcionalidades del programa, a saber: entradas digitales, salidas digitales, entradas analógicas y salidas analógicas. Se puede seleccionar la opción deseada tanto desde el menú clásico “Acciones” como desde los botones de acceso directo, de un modo más rápido.

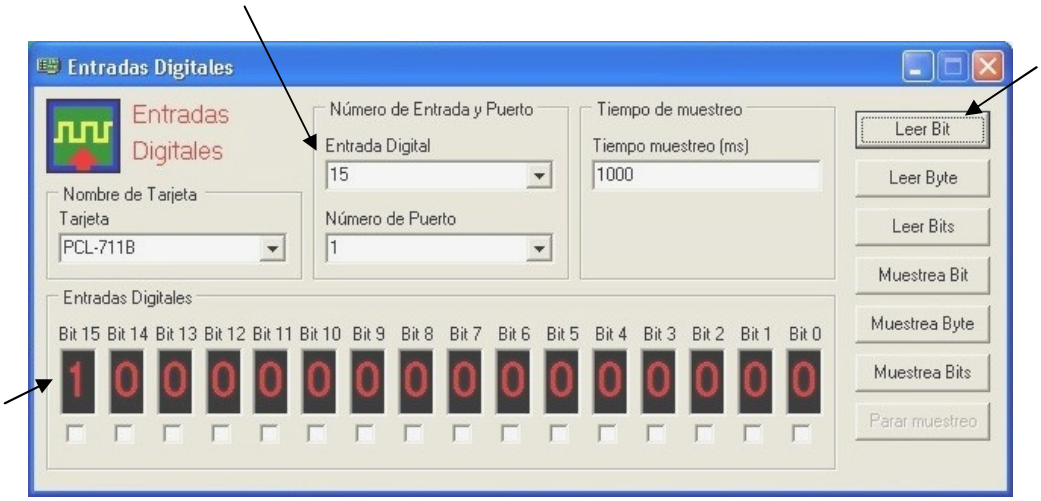
4.4.2.- ENTRADAS DIGITALES

Lo primero que tendremos la posibilidad de elegir es la tarjeta que vamos a usar: PCL-711B ó PCL-812, siendo la primera la que aparecerá por defecto. Esta opción la vamos a tener en cada uno de los subprogramas que componen la aplicación. En esta guía enseñamos el funcionamiento para la tarjeta PCL-711B, ya que para el modelo PCL-812 es análogo.

Aquí vamos a poder leer entradas digitales de diferentes maneras: leer un solo bit, leer varios bits seleccionados o leer un byte completo. Podremos elegir si queremos hacerlo una única vez o realizando un muestreo cada período determinado por nosotros (en milisegundos).

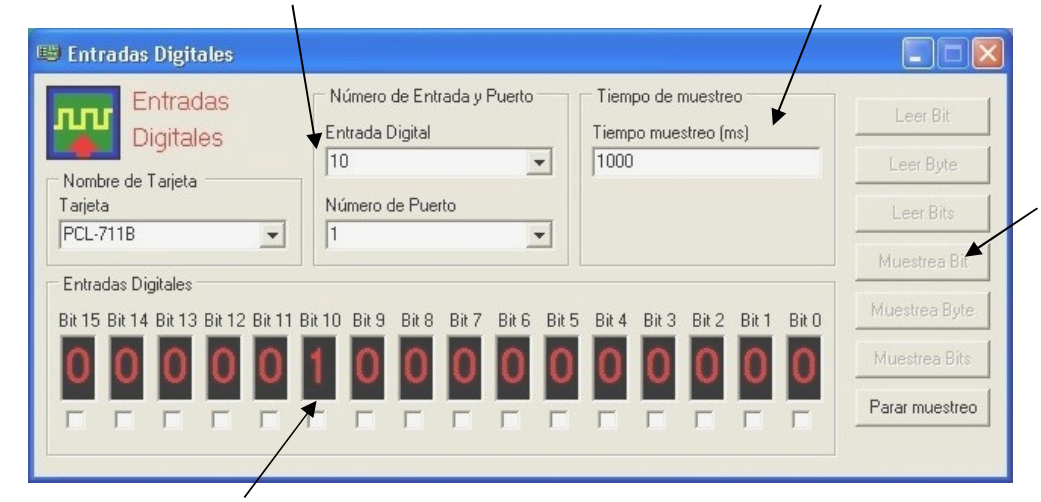
LEER BIT:

Lee el bit seleccionado en “Entrada Digital” y actualiza su valor en el cuadro “Entradas Digitales” en el bit correspondiente. En el caso de la figura se ha leído el bit 15 y se puede ver como se ha actualizado su valor.



MUESTREAR BIT:

Realiza el muestreo del bit elegido en “Entrada Digital” cada período seleccionado en “Tiempo muestreo” y va actualizando el valor en “Entradas Digitales” en el bit correspondiente. Cuando deseamos detener el muestreo pulsamos “Parar muestreo”.

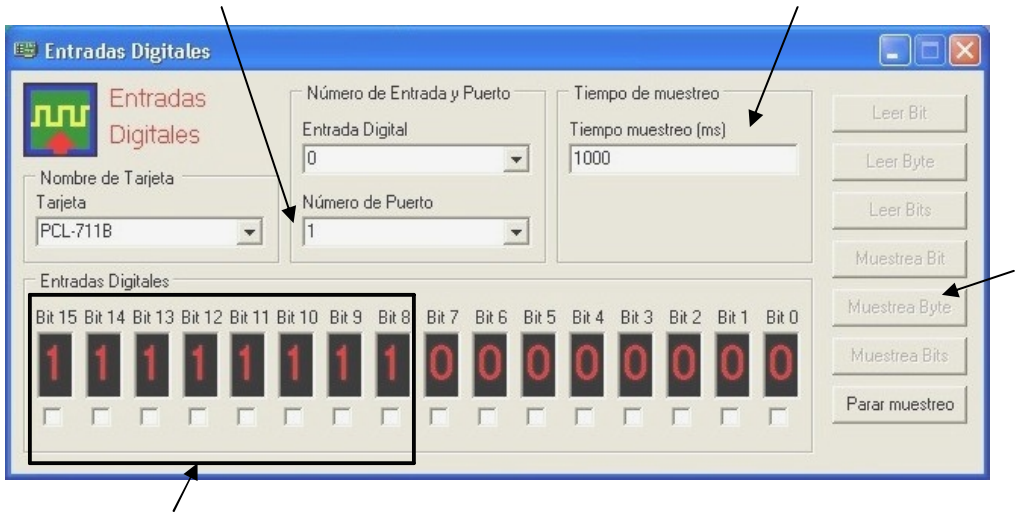


LEER BYTE:

Lee el byte seleccionado en “Número de Puerto” y actualiza su valor en el cuadro “Entradas Digitales” en los bits correspondientes.

MUESTREAR BYTE:

Realiza el muestreo del byte elegido en “Entrada Digital” cada período seleccionado en “Tiempo muestreo” y va actualizando el valor en “Entradas Digitales” en los bits correspondientes. Cuando deseamos detener el muestreo pulsamos “Parar muestreo”.

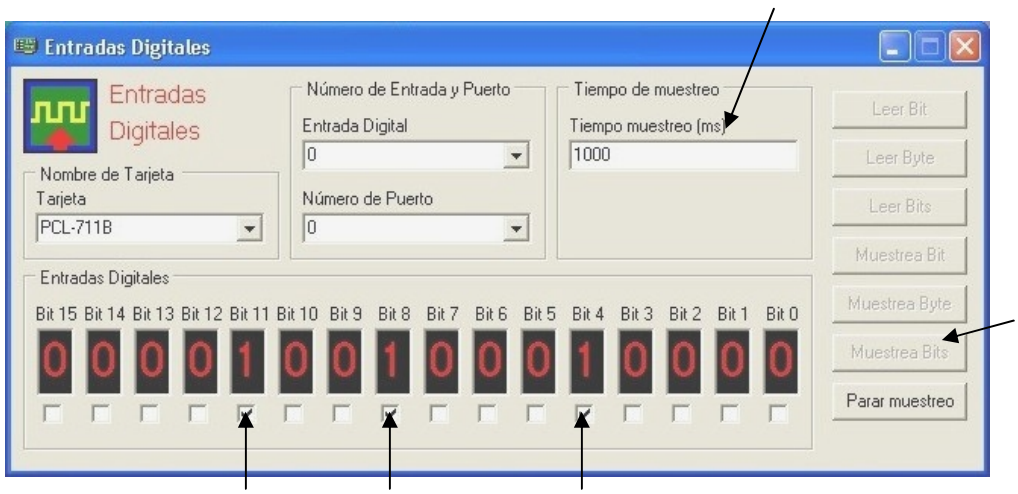


LEER BITS:

Lee los bits elegidos en los cuadros de selección en “Entradas Digitales” y actualiza su valor en ese mismo cuadro en los bits correspondientes.

MUESTREAR BITS:

Realiza el muestreo de los bits elegidos en los cuadros de selección de “Entradas Digital” cada período seleccionado en “Tiempo muestreo” y va actualizando el valor en “Entradas Digitales” en los bits correspondientes. Cuando deseamos detener el muestreo pulsamos “Parar muestreo”.

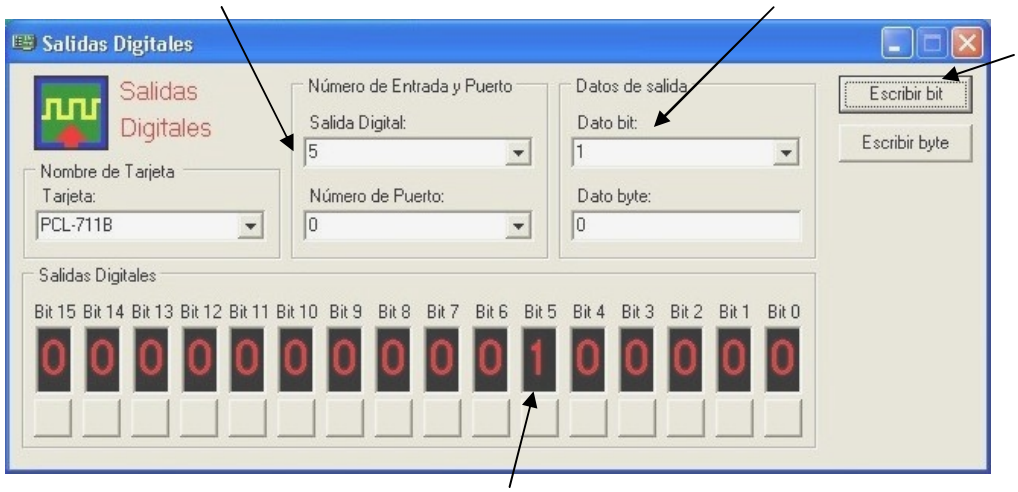


4.4.3.- SALIDAS DIGITALES

En esta parte podremos escribir salidas digitales de varios modos: escribiendo un solo bit, escribiendo varios bits o escribiendo un byte completo.

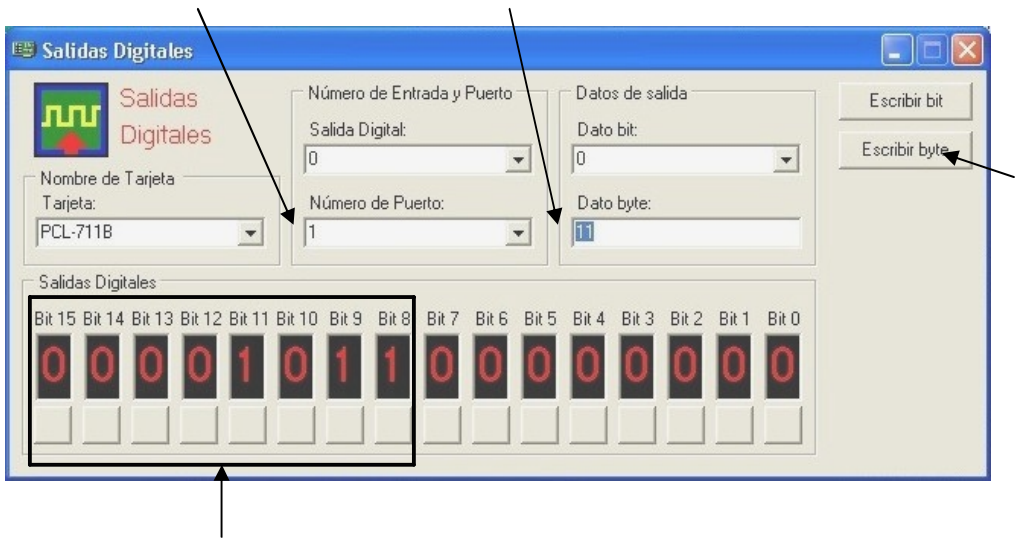
ESCRIBIR BIT:

Escribe el bit seleccionado en “Salida Digital” con el valor elegido en “Dato bit” y pone el nuevo valor en el cuadro “Salidas Digitales” en el bit correspondiente. En el caso de la figura se ha escrito el bit 5 al valor 1.



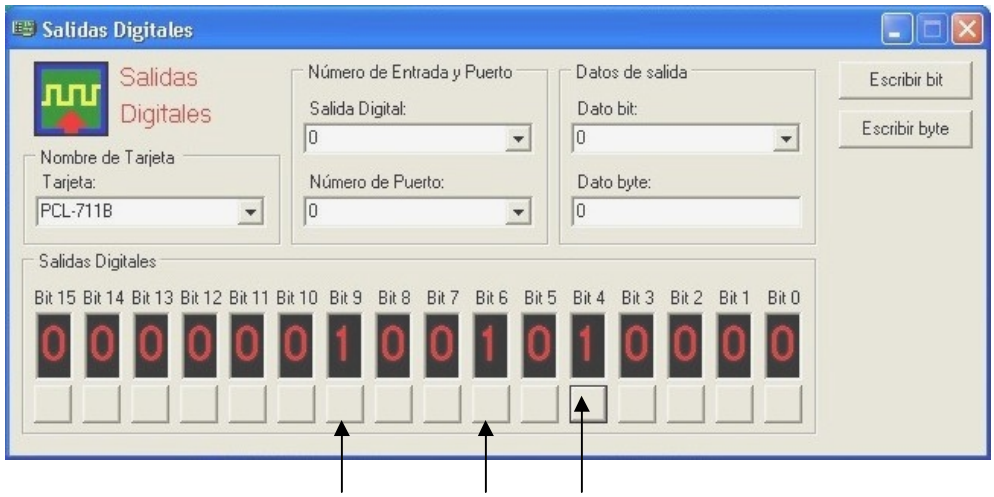
ESCRIBIR BYTE:

Escribe el byte seleccionado en “Número de Puerto” con el valor elegido en “Dato byte” y pone el nuevo valor en el cuadro “Salidas Digitales” en el byte correspondiente. En el caso de la figura se ha escrito el 11 en el byte más significativo.



ESCRIBIR BITS:

Escribe los bits seleccionados pulsando el botón correspondiente a cada bit en “Salidas Digitales”, cambiando así su valor.



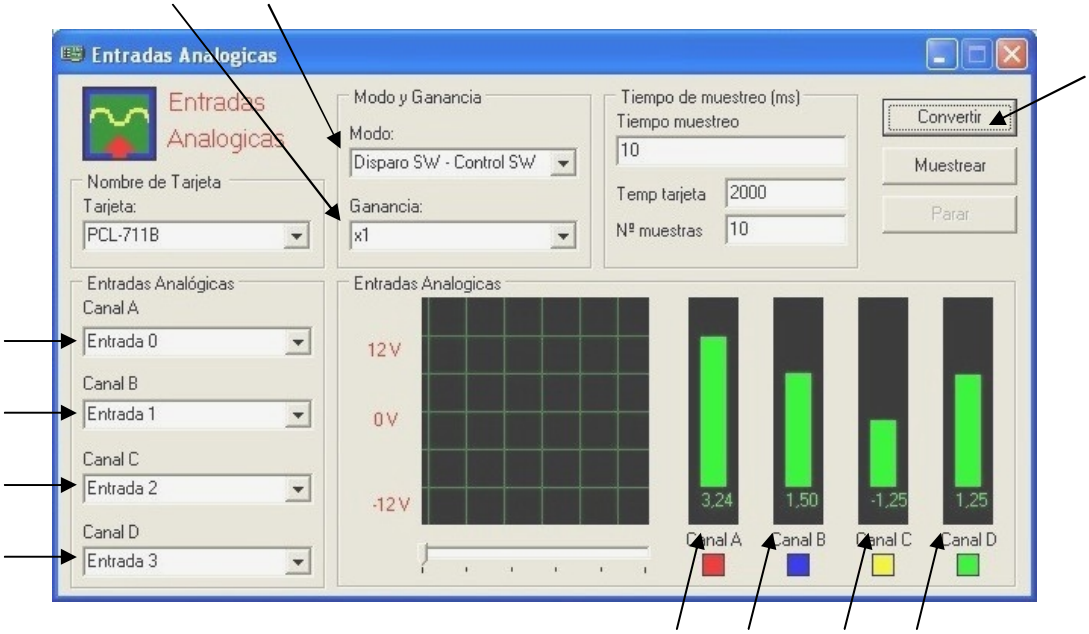
4.4.4.- ENTRADAS ANALÓGICAS

En este subprograma se podrá realizar conversiones A/D para la tarjeta PCL-711B mediante tres modos de disparo: SW, EX, TEMP y mediante dos modos de control: SW e INT. En el caso del modelo de tarjeta PCL-812 tendremos tres modos diferentes de disparo: SW, TEMP y EXT. Podremos elegir si queremos hacerlo una única vez o realizando un muestreo cada período determinado por nosotros (en milisegundos). A continuación mostramos el funcionamiento para los diferentes tipos de disparo en modo de control SW para la tarjeta PCL-711B, en modo INT funciona de modo semejante. El caso de la tarjeta PCL-812 es semejante.

DISPARO SW, CONTROL SW (1 MUESTRA):

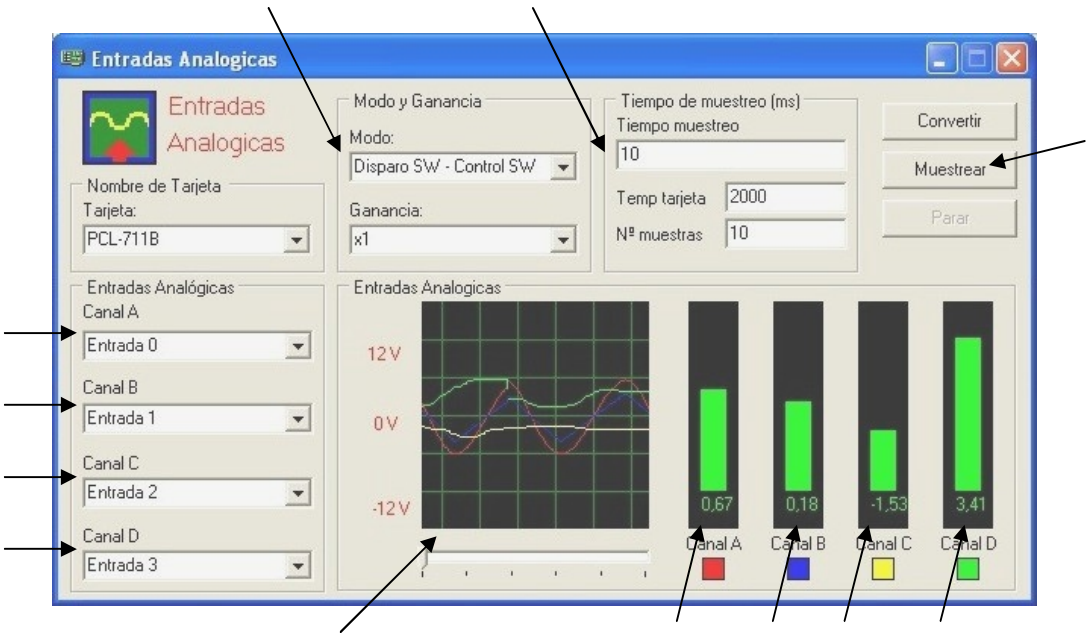
Realiza la conversión A/D en disparo modo SW y control modo SW de una sola muestra. Se seleccionan las entradas que se desean convertir en cada uno de los canales disponibles: A, B, C y D, y vemos su valor en las barras que representa a cada canal. Por defecto, en cualquier modo, está seleccionada la “Entrada 0” en el “Canal A” para que exista al menos una entrada a convertir.

Se puede elegir la ganancia entre los valores permitidos: x1, x2, x4, x8, x16. El valor predeterminado de la ganancia es x1. Esto lo podremos hacer en cada modo para la tarjeta PCL-711B, pero no está permitido en el modelo PCL-812.



DISPARO SW, CONTROL SW (VARIAS MUESTRAS):

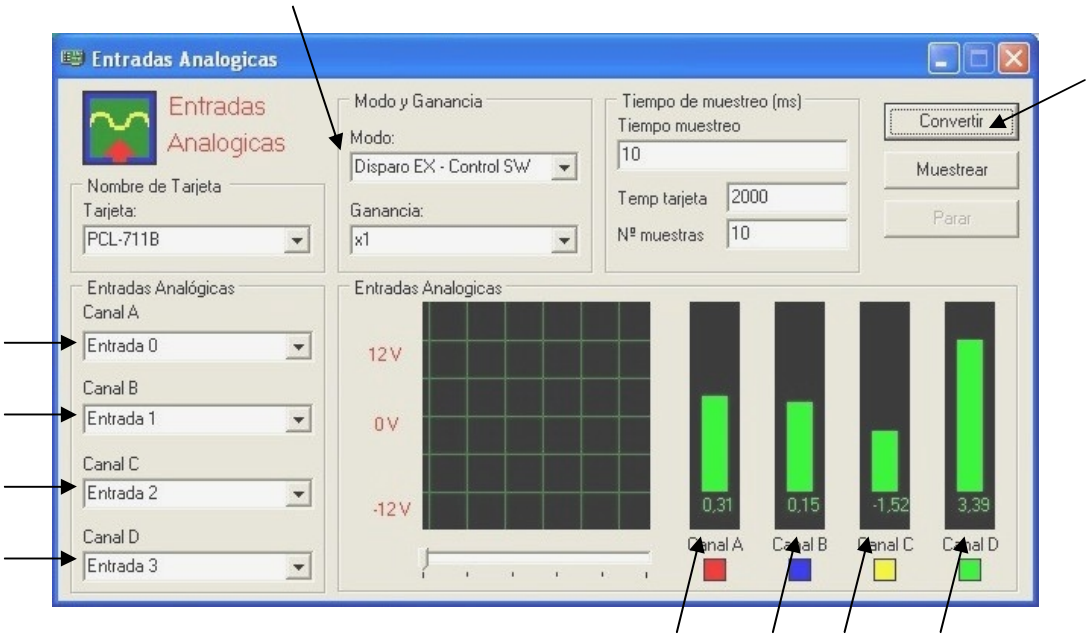
Realiza la conversión A/D en disparo modo SW y control modo SW de varias muestras, realizando una conversión cada período de tiempo seleccionado en “Tiempo muestreo”. Se seleccionan las entradas que se desean convertir en cada uno de los canales disponibles: A, B, C y D, y vemos su valor en las barras que representa a cada canal.



Ese valor se va dibujando en la gráfica para cada muestra y vamos viendo el progreso del valor de las entradas. En este modo, para evitar confusiones, se deshabilita la etiqueta “Temp tarjeta”.

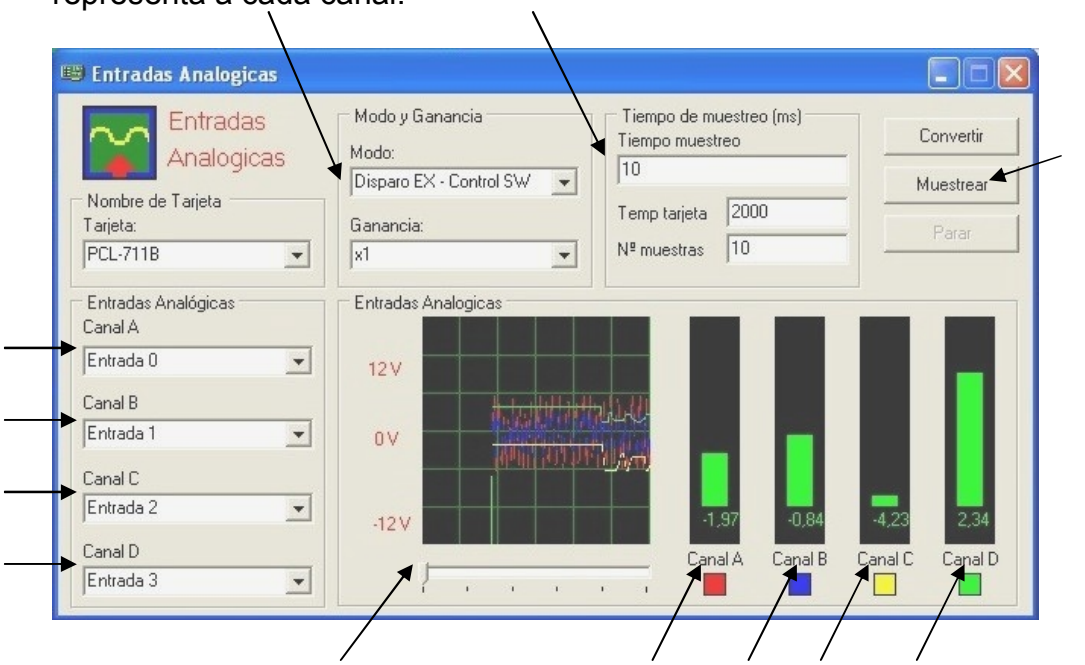
DISPARO EX, CONTROL SW (1 MUESTRA):

Realiza la conversión A/D en disparo modo EX y control modo SW de una sola muestra. Se seleccionan las entradas que se desean convertir en cada uno de los canales disponibles: A, B, C y D, y vemos su valor en las barras que representa a cada canal, tras recibir el disparo externo.



DISPARO EX, CONTROL SW (VARIAS MUESTRAS):

Realiza la conversión A/D en disparo modo EX y control modo SW de varias muestras, realizando una conversión cada período de tiempo seleccionado en “Tiempo muestreo” siempre que se produzca el disparo externo. Se seleccionan las entradas que se desean convertir en cada uno de los canales disponibles: A, B, C y D, y vemos su valor en las barras que representa a cada canal.

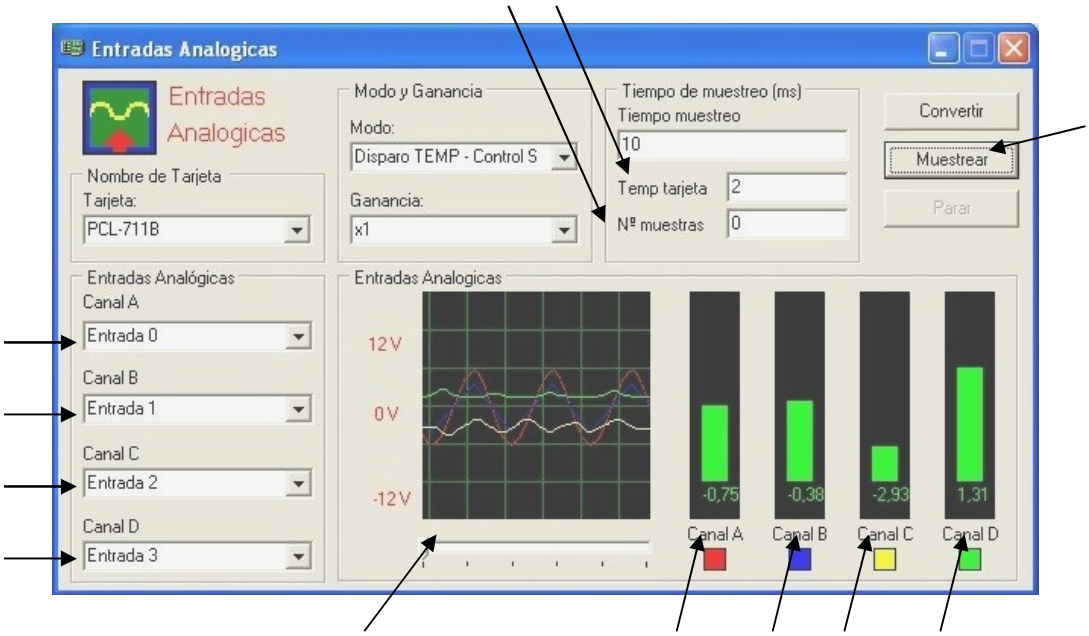


Ese valor se va dibujando en la gráfica para cada muestra y vamos viendo el progreso del valor de las entradas.

En este modo, para evitar confusiones, se deshabilita la etiqueta “Temp tarjeta”.

DISPARO TEMP, CONTROL SW:

Realiza la conversión A/D en disparo modo TEMP y control modo SW de varias muestras que seleccionamos en “Nº muestras”, realizando una conversión cada período de tiempo seleccionado en “Temp tarjeta”. Se seleccionan las entradas que se desean convertir en cada uno de los canales disponibles: A, B, C y D, y vemos su valor en las barras que representa a cada canal.



Ese valor se dibuja en la gráfica para cada muestra y vamos viendo el progreso del valor de las entradas.

Si en lugar de pulsar el botón muestrear se pulsa el botón convertir se realizará una única muestra tras el tiempo seleccionado en “Temp tarjeta”.

En este modo, para evitar confusiones, se deshabilita la etiqueta “Tiempo muestreo”.

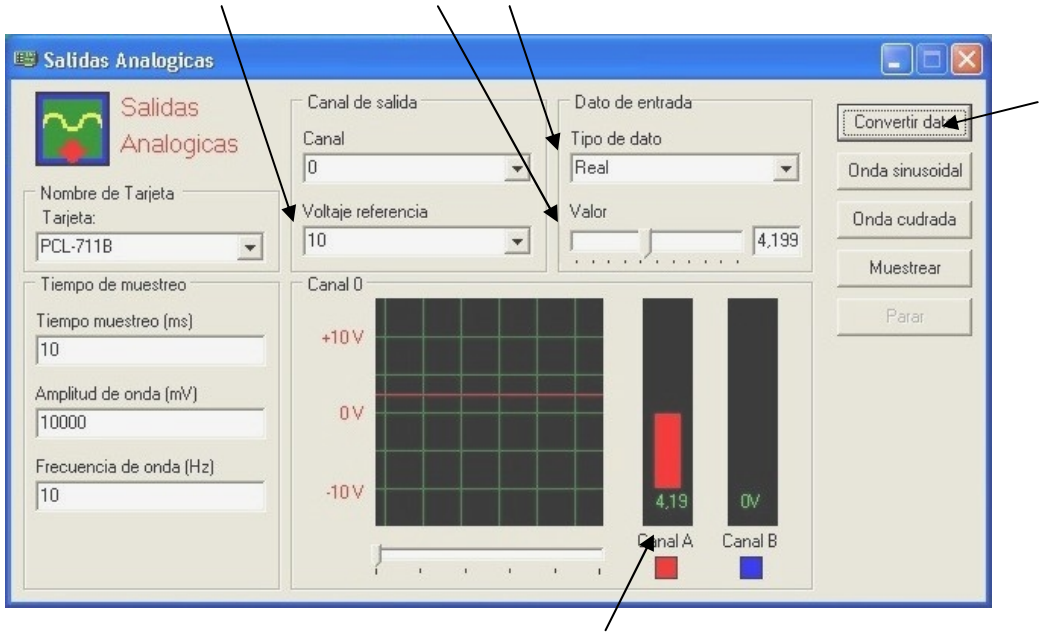
4.4.5.- SALIDAS ANALÓGICAS

En este subprograma se podrá realizar conversiones D/A, tanto para la tarjeta PCL-711B como para la PCL-812. La única diferencia está en que el segundo modelo tiene dos canales de salida a diferencia del primer modelo que sólo tiene uno. Podremos elegir si queremos hacerlo una única vez o realizando varias muestras. También será posible la generación de ondas sinusoidales y cuadradas, eligiendo su amplitud y frecuencia. Mostramos el

funcionamiento para la tarjeta PCL-711B. Para el modelo PCL-812 será análogo.

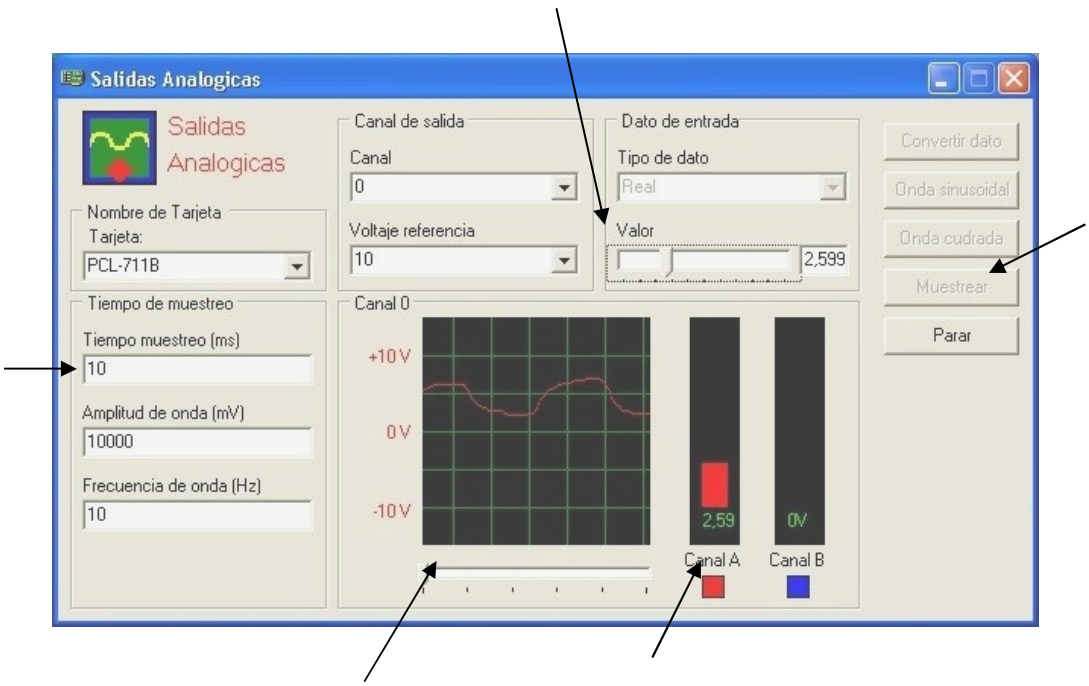
CONVERTIR DATO:

Realiza la conversión D/A del dato cuyo valor hemos seleccionado en "Valor" por medio de una barra de desplazamiento.



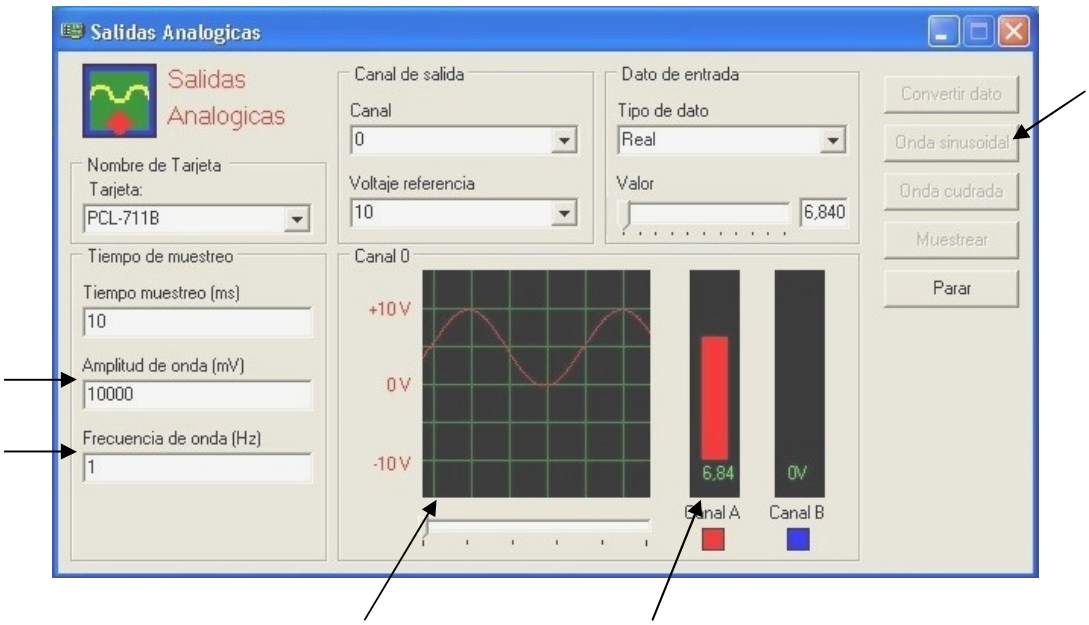
MUESTREAR DATOS:

Realiza la conversión D/A de los datos cuyo valor vamos seleccionando en "Valor" por medio de una barra de desplazamiento. La conversión se realiza cada período indicado en "Tiempo muestra", elegido por el usuario. El valor se va visualizando tanto en la barra del canal correspondiente como en la gráfica.



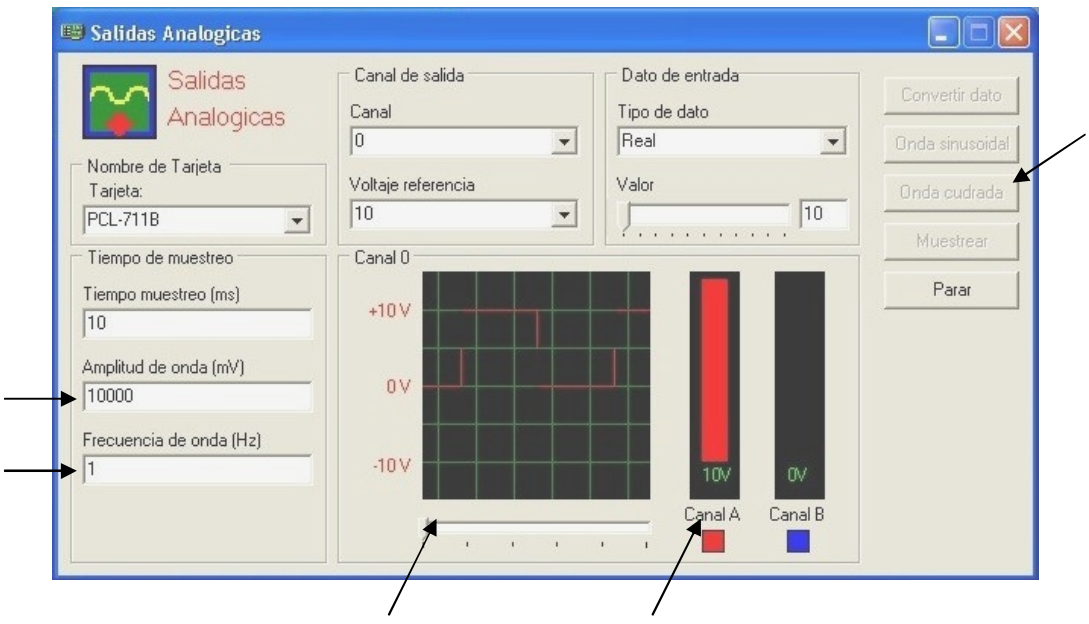
GENERAR ONDA SINUSOIDAL:

Genera una onda sinusoidal con un valor de amplitud que se puede seleccionar en “Amplitud de onda”, y con una frecuencia que también se puede elegir en “Frecuencia de onda”. En la gráfica se dibuja la onda sinusoidal y en la barra del canal correspondiente aparecen los valores en cada momento.



GENERAR ONDA CUADRADA:

Genera una onda cuadrada con un valor de amplitud que se puede seleccionar en “Amplitud de onda”, y con una frecuencia que también se puede elegir en “Frecuencia de onda”. En la gráfica se dibuja la onda cuadrada correspondiente y en la barra del canal aparecen los valores en cada momento.



5. CONCLUSIONES

En resumen, el proyecto ha consistido básicamente en el desarrollo de un driver para el control de tarjetas de adquisición de datos (PCL-711B y PCL-812), de una DLL y de una palicación para el control de la tarjeta.

Una vez diseñado el driver, el usuario final puede programar y controlar la tarjeta mediante el uso de este driver. A pesa de todo, el usuario final debía tener un amplio conocimiento sobre la funcionalidad de la tarjeta, registros utilizados y el modo de operar con ellos. Para solucionar este inconveniente, se propuso el desarrollo de una librería de enlace dinámico (DLL) para el control de la tarjeta. En esta DLL, se definen una serie de funciones que abarcan la funcionalidad básica de las tarjetas, permitiendo al usuario la programación de las tarjetas con un breve conocimiento sobre la funcionalidad de las mismas.

Finalmente, se propuso el desarrollo de una aplicación para el control interactivo de las tarjetas. Mediante esta aplicación, el usuario puede utilizar las funcionalidades de las tarjetas interactivamente, es decir, sin la necesidad de realizar programación alguna.

Como resultado de este proceso, hemos adquirido el conocimiento necesario para el control del mundo externo a traves del PC. Es decir, ahora estamos en disposición de controlar el mundo externo (plantas de riego, plantas de control…) por medio de señales analógicas y digitales a traves de nuestro PC. Esta conexión se realiza a traves de las tarjetas de adquisición de datos, que por medio del driver proporcionado y de la DLL permite una facil y calra programación de las tarjetas.

6. BIBLIOGRAFÍA

A. Solomon, David, E. Russinovich, Mark, “Inside Microsoft Windows 2000 - Third Edition”, Microsoft Programming Series.

Walter Oney, “Programming the Microsoft Driver Model”, Microsoft Programming Series, Redmond, Washington, 1999.

Manual básico para la creación de drivers para Windows bajo la plataforma DDK (Driver Development Kit)
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/gstart/hh/gstart/z_gstart_hdr_5pwn.asp

Plataforma SDK (Software Development Kit) para Windows 2000, http://www.microsoft.com/whdc/ddk/

7. ANEXOS

ANEXO A: PROGRAMAS EJEMPLOS QUE USAN LAS DLLs.

- **Entradas/Salidas digitales:** en este programa se lee una entrada digital (entrada digital número 3) y se escribe una entrada digital (salida digital número 1 a “1”).

```
include "windows.h"
include "PCLDLL.h"
include <stdio.h>

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

HANDLE hndDriver; // Descriptor del dispositivo driver
BitDato datoBit;

printf("Uso de la DLL para entradas/salidas digitales.\n");
printf("=============================================\n");

// Apertura del driver.
if (PCL_InicializarDriver(hndDriver, PCL711B) != EXITO) {
      // La apertura del driver ha fallado.
      printf("El valor del dispositivo es: %d\n", hndDriver);
      return -1;
}

// Lectura de una entrada digital.
datoBit.numero = 3; // Entrada número 3.
if (PCL_LeerEntradaDigital(hndDriver, datoBit) != EXITO)
      // La lectura de bit ha fallado.
      printf("No se ha podido leer la entrada digital\n");
else
      printf("La entrada digital %d toma el valor %d\n",
              datoBit.numero, datoBit.dato);

// Escritura de una salida digital
datoBit.numero = 1;
datoBit.dato = 1; // Salida número 1 toma el valor 1.
if (PCL_EscribirSalidaDigital(hndDriver, datoBit) != EXITO)
      printf("No se ha podido escribir la salida digital\n");
else
      printf("La entrada digital %d toma el valor %d\n",
              datoBit.numero, datoBit.dato);

// Cierre del driver.
PCL_CerrarDriver(hndDriver);
return 0;
}
```

- **Entradas/Salidas analógicas:** en este programa se realiza una conversión analógica/digital (entrada analógica), y el dato convertido se vuelve a convertir mediante una conversión digital/analógica. Este proceso se repite un total de 25 veces, con un tiempo de 2ms entre conversión y conversión. El canal de entrada es el canal 0, con una ganancia de x1, y el canal de salida es el canal 0. La tarjeta ha sido programada en disparo por temporizador con control software, por lo que se produce una espera activa hasta que el dato convertido este listo. El valor de los datos digitales son relativos, es decir, definen el valor de la

señal según su valor dentro del rango 0..0xFFF y del voltaje de referencia.

```
#include "windows.h"
#include "PCLDLL.h"
#include <stdio.h>

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

HANDLE hndDriver; // Descriptor de fichero
ULONG dato;
int cont = 25;

printf("Uso de la DLL para entradas analógicas.\n");
printf("=====================================\n");
// Apertura del driver.
if (PCL_InicializarDriver(hndDriver, PCL711B) != EXITO) {
      // La apertura del driver ha fallado.
      printf("El valor del dispositivo es: %d\n", hndDriver);
      return -1;
}
// Selección de ganancia.
if (PCL_SetGananciaAD(hndDriver, PCL_GANANCIA_X1) != EXITO)
      // No se ha podido establecer la ganancia.
      printf("No se ha podido establecer la ganancia\n");
// Selección del canal de entrada.
if (PCL_SetEntradaAnalogica(hndDriver, PCL_AI_0) != EXITO)
      // No se ha podido seleccionar el canal de entrada.
      printf("No se ha podido seleccionar el canal de
            entrada\n");
//Selección del modo de control.
if (PCL_SetModoControl(hndDriver, PCL_DISPARO_TEMP_MODO_SW) !=
    EXITO)
      // No se ha podido establecer el modo de control.
      printf("No se ha podido establecer el modo de control\n");
// Programación del contador.
if (PCL_ProgramaContadorTiempo(hndDriver, 2, DISPARO_MULTIPLE)
!=
     EXITO)
      // No se ha podido programar el contador.
      printf("No se ha podido programar el contador\n");
// Lectura del dato convertido.
while (cont >= 0) {
      while (PCL_LeerDatoAD(hndDriver, dato) ==
            ERROR_DATO_NO_LISTO) {
            // Espera activa
      }
      printf("El dato leido es: %d\n", dato); // Dato relativo.
      // Salida analógica.
      if (PCL_ConversionDA(hndDriver, 0, dato) != EXITO)
            // No se ha podido convertir el dato.
            printf("No se ha podido convertir el dato\n");
      cont--;
}
// Cierre del driver.
PCL_CerrarDriver(hndDriver);
return 0;
}
```