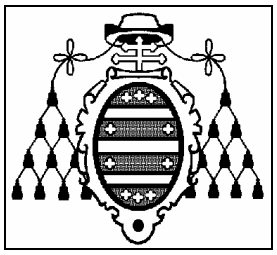
UNIVERSIDAD DE OVIEDO



ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA INFORMÁTICA Y TELEMÁTICA DE GIJÓN

DOCUMENTO Nº 3

MANUALES DE USUARIO

CONJUNTO DE DRIVERS PARA TARJETAS DE ADQUISICIÓN DE SEÑALES BAJO LINUX

Adolfo Antonio Fernández Trabanco

JUNIO 2005

Índice

Listado de Figuras

Listado de Tablas

Listado de Ejemplos

1.- Introducción

1.1.- Identificación del proyecto

Título: Conjunto de drivers para tarjetas de adquisición de señales bajo Linux

Directores: Víctor Manuel González Suárez
José Antonio Cancelas Caso

Autor: Adolfo Antonio Fernández Trabanco

Fecha: Junio 2005

1.2.- Visión general del documento

En el presente documento se incluyen los manuales de usuario de las librerías desarrolladas. Son manuales de referencia para el programador, entendiendo como programador al usuario que utilizará las librerías desarrolladas.

Se incluyen en el documento tres manuales:

- **FieldPoint, manual de usuario** (epígrafe 2)
  Manual de usuario de la librería desarrollada para el sistema FieldPoint de la empresa National Instruments.

- **NuDAM, manual de usuario** (epígrafe 3)
  Manual de usuario de la librería desarrollada para el sistema NuDAM de la empresa ADLink Technology.

- **LabJack U12, manual de usuario** (epígrafe 4)
  Manual de usuario de la librería desarrollada para la tarjeta de adquisición de datos LabJack U12 de la empresa LabJack.

En cada uno de ellos se muestra el proceso de instalación, conceptos sobre la utilización de la librería y una descripción detallada de ésta. En esta descripción, se muestran las particularidades de las dos interfaces de programación (en lenguaje C y en lenguaje Ada), así como la descripción de cada una de las funciones que componen la librería.

2.- FieldPoint, manual de usuario

2.1.- Instalación

Dado que los módulos FieldPoint se comunican a través del puerto serie, es necesario verificar que el fichero asociado al puerto que se va a utilizar en la comunicación, tiene permisos de lectura y escritura para todos los usuarios.

En los sistemas Linux, los periféricos se encuentran representados como simples ficheros del sistema de archivos. Cada uno de los puertos serie tiene asociado uno o más ficheros. En la Tabla 2.1 se muestra la asociación más habitual entre los puertos serie y los ficheros del sistema de archivos.

Puerto	Fichero
COM1	/dev/ttyS0
COM2	/dev/ttyS1
COM3	/dev/ttyS2
COM4	/dev/ttyS3

Tabla 2.1 Asociación de puertos serie a ficheros de dispositivo

Para permitir que cualquier usuario pueda leer y escribir en el segundo puerto serie, se cambiarán los permisos del fichero asociado, ejecutando como usuario root el siguiente comando:

```
chmod 666 /dev/ttyS1
```

Para utilizar la librería, no es necesario realizar ningún proceso de instalación adicional. Es suficiente con copiar en el directorio de trabajo los ficheros adecuados:

- Si se va a utilizar la versión en Ada, se deben copiar los ficheros *ComPort.h*, *ComPort.c*, *DriverC_NI.h*, *DriverC_NI.c*, *driverada_ni.ads* y *driverada_ni.adb*.
- Si se va a utilizar la versión en C, se deben copiar los ficheros *ComPort.h*, *ComPort.c*, *DriverC_NI.h* y *DriverC_NI.c*.

En el epígrafe 2.2.3 se explica el proceso a seguir para que el usuario cree sus propios programas de control y las instrucciones de compilación necesarias para crear el ejecutable.

En el CD que acompaña a la documentación, se encuentran los ficheros fuente que forman el driver, así como las aplicaciones de prueba.

Para utilizar las aplicaciones de prueba proporcionadas, se debe copiar la carpeta adecuada del CD al directorio de trabajo del usuario. Para crear los ejecutables, ejecutar el comando `make`.

2.2.- Utilización

2.2.1.- Configuración del módulo de red

La configuración del módulo de red se hace mediante unos pequeños interruptores alojados en el frontal del módulo (Figura 2.1) que permiten fijar la dirección física y la velocidad de comunicación del puerto serie. En las tablas 2.2 y 2.3 se muestran las configuraciones posibles para fijar la dirección física y la velocidad del puerto respectivamente.

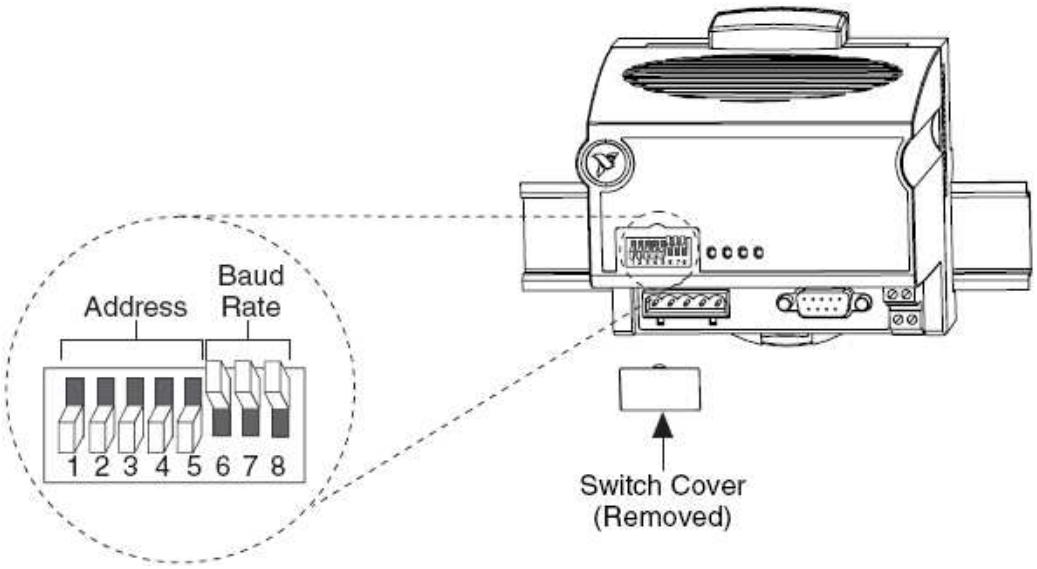
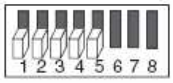
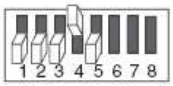
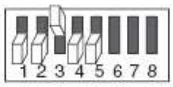
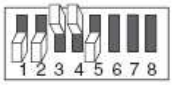
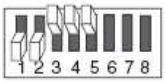
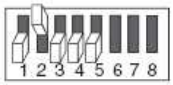
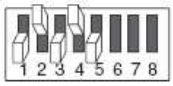


Figura 2.1 Detalle de los switchs de configuración del FP-1000

Switch Positions 1–5	Network Module Address (Decimal)
	0
	10
	20
	30
	40
	50
	60
	70
	80
	90
	100
	110
	120

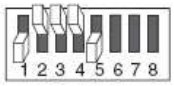
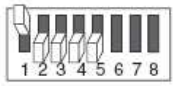
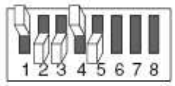
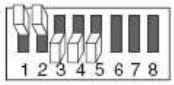
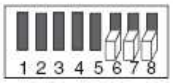
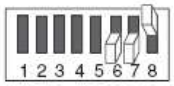
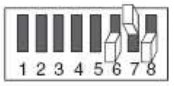
Switch Positions 1–5	Network Module Address (Decimal)
	130
	140
	150
	160
	170
	180
	190
	200
	210
	220
	230
	240
Other Settings	Not Allowed

Tabla 2.2 Switch de configuración de la dirección física

Switch Positions 6–8	Network Module Baud Rate
	300
	1200
	2400
	9600

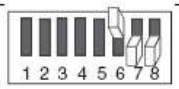
Switch Positions 6–8	Network Module Baud Rate
	19200
	38400
	57600
	115200

Tabla 2.3 Switch de configuración de la velocidad del puerto serie

2.2.2.- El bus local

Como se ha visto anteriormente, el módulo de red se puede configurar con distintas direcciones físicas. Para utilizar las funciones proporcionadas, el usuario no necesita conocer en ningún momento las direcciones físicas de los módulos conectados al bus. El argumento *modulo* que aparece en las funciones hace referencia a la posición del módulo en el **bus local**. Para conocer la dirección de un módulo en el bus local se puede proceder de dos formas:

- Observar la disposición física de los módulos.
- Ejecutar el programa de prueba ***list_all*** proporcionado con las fuentes (ver epígrafe 2.2.4.1).

Para explicar el primer caso vamos a mostrar un ejemplo. Supongamos que la disposición física de los módulos es la mostrada en la Tabla 2.4.

	Módulo FP-1000	Módulo FP-DI-301	Módulo FP-RLY-420	Módulo FP-RLY-420
Dirección física	base	base + 1	base + 2	base + 3
Posición en el bus		0	1	2

Tabla 2.4 Ejemplo de asignación de direcciones en el bus local

Como se puede ver, la posición en el bus empieza a contar a partir de cero y sin contar el módulo de red. El primer módulo a continuación del módulo de red tendrá la dirección cero, el siguiente la uno y así sucesivamente.

La detección de los módulos que forman el bus se hace de forma dinámica. Esto permite disponer de distintas configuraciones hardware, con uno, dos o más módulos de E/S conectados al bus.

2.2.3.- Creación de un programa

Los pasos a seguir para obtener el ejecutable del programa realizado, varían según el lenguaje de programación utilizado. Para simplificar el proceso de compilación de los programas, se recomienda al usuario que confeccione su propio fichero *Makefile*. Junto a las fuentes, se proporcionan ficheros *Makefile* que pueden servir de ejemplo.

Para crear un programa en lenguaje C que utilice las funciones de la librería, es necesario seguir los pasos indicados a continuación:

- Incluir en el código del programa la cabecera "*DriverC_NI.h*".

```
#include <DriverC_NI.h>
```

- Realizar el proceso de compilación para generar el fichero ejecutable. Se deben realizar los siguientes pasos:
  - Compilar los ficheros fuente que componen el driver:

```
g++ -I. -c ComPort.c
g++ -I. -c DriverC_NI.c
```

  - Compilar el programa del usuario (supongamos que el fichero se llama ejemplo.cc):

```
g++ -I. -c ejemplo.cc
```

  - Enlazar todos los ficheros objeto para obtener el ejecutable final:

```
g++ -g ComPort.o DriverC_NI.o ejemplo.o -o ejemplo
```

Si se desea crear un programa en lenguaje Ada que utilice las funciones de la librería, es necesario seguir los siguientes pasos:

- Incluir en el código del programa el paquete "*DriverAda_NI*".

```
with DriverAda_NI; use DriverAda_NI;
```

- Realizar el proceso de compilación para generar el fichero ejecutable. Se deben realizar los siguientes pasos:
  - Compilar los ficheros fuente que componen el driver:

```
gcc -c -I. ComPort.c
gcc -c -I. DriverC_NI.c
```

  - Compilar el programa del usuario (supongamos que el fichero se llama ejemplo.adb):

```
gnatmake -c ejemplo.adb
```

  - Enlazar todos los ficheros objeto para obtener el ejecutable final:

```
gnatbind -x ejemplo
gnatlink ejemplo ComPort.o DriverC_NI.o
```

2.2.4.- Aplicaciones de prueba

Junto con las fuentes, se proporcionan un conjunto de programas genéricos que permiten al usuario interaccionar con el proceso e iniciarse en el uso de las distintas funciones. Son programas en modo consola, carentes de interfaz gráfica, y se dispone de dos versiones para cada uno de ellos: una implementada en lenguaje C y otra implementada en lenguaje Ada.

2.2.4.1.- list_all

Esta aplicación muestra por pantalla el número de módulos que forman el bus local y las características de cada uno de ellos: dirección en el bus local, dirección física, tipo de módulo, nombre, identificador, número de canales, etc.

Para ejecutar la aplicación es necesario indicarle tres parámetros en la orden de ejecución:

- El puerto serie a utilizar:
  - Indicar un 1 si se va utilizar el puerto serie COM1.
  - Indicar un 2 si se va utilizar el puerto serie COM2.
- La velocidad de comunicación del puerto serie.
- La dirección física del módulo de red.

La velocidad de comunicación y la dirección física han de coincidir con la configuración del módulo de red (ver epígrafe 2.2.1).

Si se omite alguno de estos parámetros, la aplicación informa al usuario de esta situación y le indica los parámetros que ha de introducir (Ejemplo 2.1). En el Ejemplo 2.2 se puede ver una ejecución correcta del programa.

```
[txolfo@fms-200-6 ada]$ ./list_all

Uso: ./list_all <puerto> <velocidad> <dir.base>

<puerto> Para el puerto serie COM1 indicar un  1
         Para el puerto serie COM2 indicar un  2
<velocidad> Indicar la velocidad del puerto.
<dir.base> Dirección base del módulo de cabecera

[txolfo@fms-200-6 ada]$
```

Ejemplo 2.1 Ejecución de *list_all* incorrecta

```
[txolfo@fms-200-6 ada]$ ./list_all 2 115200 0

Posición en el bus:  0
  Tipo de dispositivo: Entradas Digitales
  Dirección física del módulo: 1
  Nombre del módulo: FP-DI-301
  Número del módulo: 261
  Número de líneas: 16

Posición en el bus:  1
  Tipo de dispositivo: Salidas Digitales
  Dirección física del módulo: 2
  Nombre del módulo: FP-RLY-420
  Número del módulo: 264
  Número de líneas: 8

Posición en el bus:  2
  Tipo de dispositivo: Salidas Digitales
  Dirección física del módulo: 3
  Nombre del módulo: FP-RLY-420
  Número del módulo: 264
  Número de líneas: 8

[txolfo@fms-200-6 ada]$
```

Ejemplo 2.2 Ejecución de *list_all* correcta

2.2.4.2.- entradas

Esta aplicación muestra por pantalla el estado de los canales digitales de entrada del módulo de entradas digitales FP-DI-301 que le indiquemos.

Para ejecutar la aplicación es necesario indicarle tres parámetros en la orden de ejecución:

- El puerto serie a utilizar:
  - Indicar un 1 si se va utilizar el puerto serie COM1.
  - Indicar un 2 si se va utilizar el puerto serie COM2.
- La velocidad de comunicación del puerto serie.
- La dirección física del módulo de red.

La velocidad de comunicación y la dirección física han de coincidir con la configuración del módulo de red (epígrafe 2.2.1).

Al ejecutar la aplicación, además de mostrar el número de módulos que se han detectado en el bus local, se pide que indiquemos la posición en el bus del módulo de entradas sobre el que deseamos ejecutar la prueba. Si en este paso se indica una posición en el bus local que no existe o que no se corresponde con un módulo de entradas digitales, se informará al usuario de esta circunstancia y se le pedirá que introduzca un valor válido (Ejemplo 2.3). Tras indicar una posición válida, se muestra por pantalla el estado de las entradas (Ejemplo 2.4).

```
[txolfo@fms-200-6 ada]$ ./entradas 2 115200 0

NI_Inicializar:
 0 CORRECTO

Se han encontrado 3 módulos
Indique la posición en el bus del módulo de entradas sobre el que actuar: 1

El número de módulo indicado es INCORRECTO
Indique la posición en el bus del módulo de entradas sobre el que actuar: 9

El número de módulo indicado es INCORRECTO
Indique la posición en el bus del módulo de entradas sobre el que actuar:
```

Ejemplo 2.3 Ejecución de *entradas*

```
[txolfo@fms-200-6 ada]$ ./entradas 2 115200 0

NI_Inicializar:
 0 CORRECTO

Se han encontrado 3 módulos
Indique la posición en el bus del módulo de entradas sobre el que actuar: 0

NI_LeeModulo_DI:
 16 CORRECTO
Estado de las entradas en decimal: 2132
Estado de las entradas en hexadecimal: 16#854#

NI_LeeLinea_DI:
 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
  0  0  0  0  1  0 0 0 0 1 0 1 0 1 0 0

Volver a mostrar las entradas?? (s/n) n

NI_Liberar:
 0 CORRECTO

[txolfo@fms-200-6 ada]$
```

Ejemplo 2.4 Ejecución completa de *entradas*

2.2.4.3.- salidas

Esta aplicación permite interaccionar con el proceso mediante la activación de los distintos canales de salida de un módulo FP-RLY-420.

Para ejecutar la aplicación es necesario indicarle tres parámetros en la orden de ejecución:

- El puerto serie a utilizar:
  - Indicar un 1 si se va utilizar el puerto serie COM1.
  - Indicar un 2 si se va utilizar el puerto serie COM2.
- La velocidad de comunicación del puerto serie.
- La dirección física del módulo de red.

La velocidad de comunicación y la dirección física han de coincidir con la configuración del módulo de red (epígrafe 2.2.1).

Al ejecutar la aplicación, además de mostrar el número de módulos que se han detectado en el bus local, se pide que indiquemos la posición en el bus del módulo de salidas sobre el que deseamos ejecutar la prueba. Si en este paso se indica una posición en el bus local que no existe o que no se corresponde con un módulo de salidas digitales, se informará al usuario de esta circunstancia y se le pedirá que introduzca un valor válido (Ejemplo 2.5).

```
[txolfo@fms-200-6 ada]$ ./salidas 2 115200 0

NI_Inicializar:
 0 CORRECTO

Se han encontrado 3 módulos
Indique la posición en el bus del módulo de salidas sobre el que actuar: 0

El número de módulo indicado es INCORRECTO
Indique la posición en el bus del módulo de salidas sobre el que actuar: 5

El número de módulo indicado es INCORRECTO
Indique la posición en el bus del módulo de salidas sobre el que actuar:
```

Ejemplo 2.5 Ejecución de *salidas*

Tras indicar la posición, se solicita al usuario el canal de salida sobre el que actuar y el valor de éste. Si en este paso se indica un número de línea erróneo pueden suceder dos cosas, dependiendo de la versión utilizada:

- En la versión en C, se producirá un error y se indicará esta circunstancia por pantalla (Ejemplo 2.6).
- En la versión en Ada, se producirá una excepción en tiempo de ejecución.

En el Ejemplo 2.7 se muestra una ejecución completa de la aplicación.

```
[txolfo@fms-200-6 c]$ ./salidas 2 115200 0

NI_Inicializar
0 0

Se han encontrado 3 módulos
Indique la posición en el bus del módulo de salidas sobre el que actuar: 1

NI_EscribeLinea_DI
Interacción con el módulo de salidas...

Posición del módulo de salidas en el bus: 1
Linea: 0
Estado (1->Activar, 0->Desactivar) 1
Continuar?? (s/n) s

Posición del módulo de salidas en el bus: 1
Linea: 9
Estado (1->Activar, 0->Desactivar) 0
-1 -500 (NI_EscribeLinea_DO) Número de línea erróneo (9)
Continuar?? (s/n)
```

Ejemplo 2.6 Ejecución de *salidas* (versión en C)

```
[txolfo@fms-200-6 ada]$ ./salidas 2 115200 0

NI_Inicializar:
 0 CORRECTO

Se han encontrado 3 módulos
Indique la posición en el bus del módulo de salidas sobre el que actuar: 2

NI_EscribeLinea_DO:
Interacción con los módulos de salidas...

Posición del módulo de salidas en el bus:  2
Linea: 1
Estado (1->Activar, 0->Desactivar) 1

Continuar?? (s/n) n

NI_EscribeModulo_DO:
 8 CORRECTO

NI_Liberar:
 0 CORRECTO

[txolfo@fms-200-6 ada]$
```

Ejemplo 2.7 Ejecución completa de *salidas*

2.3.- Descripción de la librería

En este epígrafe se pretende dar la información necesaria al usuario para que pueda comenzar a realizar sus propios programas.

Existen dos interfaces distintas (una en lenguaje C y otra en lenguaje Ada) de manera que el usuario es libre de elegir para implementar su programa de control la que más se adapte a sus necesidades. Dado que ambas versiones difieren ligeramente (en cuanto a tipos definidos y variables globales utilizadas) se invita al usuario a leer con detenimiento las particularidades de cada una de ellas.

2.3.1.- Interfaz en C

2.3.1.1.- Tipos definidos en C

En el archivo de cabecera "*DriverC_NI.h*" (ver epígrafe 2.4) se definen los tipos de datos que son utilizados por la librería. Se recomienda al usuario utilizarlos en la implementación de sus programas. Los tipos definidos son:

- **U8** Precisión: 8 bits. Rango: 0-255
- **U16** Precisión: 16 bits. Rango: 0-65535

Estos tipos se utilizan en las funciones *NI_EscribeModulo_DO* y *NI_LeeModulo_DI* respectivamente.

Un ejemplo de utilización del tipo U8 (para la función *NI_EscribeModulo_DO*) se muestra a continuación. Supongamos que deseamos poner a uno los canales cero y cuatro del módulo de salidas FP-RLY-420. Tendríamos la siguiente situación:

bit	7	6	5	4	3	2	1	0
estado	0	0	0	1	0	0	0	1
máscara	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Para llamar a la función *NI_EscribeModulo_DO* se codificaría en un dato de tipo U8 la información mostrada, lo que equivale a mandar el valor U8 = $2^0 + 2^4 = 1 + 16 = 17$.

2.3.1.2.- Variables globales en C

En el archivo de cabecera "*DriverC_NI.h*" (ver epígrafe 2.4) se definen una serie de variables globales utilizadas por la librería y de utilidad para el usuario:

- **COM1** y **COM2**: Indican el puerto serie a utilizar en la comunicación. Estas variables se utilizan para realizar la llamada a la función *NI_Inicializar* (p.ej. `NI_Inicializar (COM2, 115200, 0)`)

- **_Digital_IN** y **_Digital_OUT**: Indican el tipo de módulo: entradas digitales y salidas digitales respectivamente. Estas variables se utilizan en la función *Get_Tipo*.

```
tipo = Get_Tipo (i);
switch (tipo){
case _Digital_IN:
  cout << "\tTipo de dispositivo: Entradas Digitales" << endl;
  break;
case _Digital_OUT:
  cout << "\tTipo de dispositivo: Salidas Digitales" << endl;
  break;
}
```

- **Variables de error**: Proporcionan información sobre el estado en que finalizó la última operación. En el epígrafe 2.3.1.3 se puede obtener una descripción de cada una de ellas. Siempre que se produzca un error, se recomienda utilizar las funciones *ObtenerError* y *ObtenerErrorString* para obtener el código de error y una descripción de éste respectivamente.

2.3.1.3.- Variables de error en C

En la Tabla 2.5 se muestran las variables de error utilizadas, el valor que toman cada una de ellas y la descripción.

Error	Valor	Descripción
CORRECTO	0	En la última operación no se produjo ningún error
ERROR_COM	-101	Error en la comunicación con el puerto serie
FD_INVALIDO	-102	Descriptor del puerto inválido
TIMEOUT	-103	Timeout al leer del puerto. El dispositivo no ha respondido en el tiempo esperado
RESPUESTA_ERRONEA	-200	El dispositivo ha devuelto una respuesta considerada errónea
ERROR_MEM	-300	No hay memoria dinámica reservada. Se puede producir por dos causas: • Si no se ha llamado a la función de inicialización antes de llamar a cualquier otra función • Si ha fallado la reserva dinámica de memoria durante la inicialización
NO_MODULO	-400	El número de módulo al que se quiere acceder no existe en el bus local
NO_DINPUT	-401	El módulo no es un módulo de entradas digitales
NO_DOUTPUT	-402	El módulo no es un módulo de salidas digitales
NO_LINEA	-500	La línea a la que se quiere acceder no existe

Tabla 2.5 Variables de error en C

Si se produce el error RESPUESTA_ERRONEA, mediante la función *ObtenerErrorString* se puede obtener la respuesta que devolvió el dispositivo. En el epígrafe 2.5 se ofrece la descripción de las respuestas de error que pueden devolver los dispositivos FieldPoint.

2.3.2.- Interfaz en Ada

2.3.2.1.- Tipos definidos en Ada

En el archivo de cabecera "*driverada_ni.ads*" (ver epígrafe 2.4) se definen los tipos de datos que son utilizados por la librería. Se recomienda al usuario utilizarlos en la implementación de sus programas. Vamos a diferenciar entre los tipos básicos y los tipos enumerados.

Todos los tipos básicos son subconjuntos del tipo estándar *Integer* limitados en el rango. En aquellas funciones donde se utilicen estos tipos como parámetros, se debe respetar este rango pues, en caso contrario, se producirá una excepción en tiempo de ejecución. Los tipos básicos definidos son:

- **U1** Rango: 0-1
- **U8** Rango: 0-255
- **U16** Rango: 0-65535
- **Num_Linea_Entrada** Rango: 0-15
- **Num_Linea_Salida** Rango: 0-7

Vemos que en este caso se acotan el número de los canales de entrada y salida (mediante los tipos Num_Linea_Entrada y Num_Linea_Salida respectivamente), de manera que desaparece el error NO_LINEA que había en la implementación en C. Ahora, si se pasa un número de línea erróneo a una función, se producirá una excepción en tiempo de ejecución.

Vamos a mostrar un ejemplo de utilización del tipo U8 mediante la función *NI_EscribeModulo_DO*. Supongamos que deseamos poner a uno los canales cero y cuatro del módulo de salidas FP-RLY-420. Tendríamos la siguiente situación:

bit	7	6	5	4	3	2	1	0
estado	0	0	0	1	0	0	0	1
máscara	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Para llamar a la función *NI_EscribeModulo_DO* se codificaría en un dato de tipo U8 la información mostrada, lo que equivale a mandar el valor $U8 = 2^0 + 2^4 = 1 + 16 = 17$.

Mediante el uso de tipos enumerados se declaran las variables necesarias para acceder al puerto serie, definir el tipo de módulo y definir las variables de error:

- **Puerto_Serie** (valores COM1 y COM2): Indican el puerto serie a utilizar en la comunicación. Estas variables se utilizan para realizar la llamada a la función *NI_Inicializar* (p.ej. `NI_Inicializar (COM2, 115200, 0)`)

- **Device_Type** (valores DIGITAL_IN, DIGITAL_OUT y ERROR): Indican el tipo de módulo (entradas digitales, salidas digitales. Este enumerado se utiliza en la función *Get_Tipo.*

```
Tipo := Get_Tipo (I);
case Tipo is
   when Digital_IN =>
      Put ("  Tipo de dispositivo: Entradas Digitales");
      New_Line;
   when Digital_OUT =>
      Put ("  Tipo de dispositivo: Salidas Digitales");
      New_Line;
   when others =>
      Put ("  Error al obtener el tipo");
      New_Line;
end case;
```

- **Var_Error**: Indican las posibles variables de error. Este enumerado se emplea en la función *ObtenerError*. En el epígrafe 2.3.2.2 se puede obtener una descripción de cada uno de los posibles valores que puede tomar una variable del tipo *Var_Error*. Siempre que se produzca un error se recomienda utilizar la función *ObtenerErrorString* para obtener una descripción de éste.

2.3.2.2.- Variables de error en Ada

En la Tabla 2.6 se muestran los valores que puede tomar el enumerado *Var_Error* y la descripción de cada uno de ellos.

Error	Descripción
CORRECTO	En la última operación no se produjo ningún error
ERROR_COM	Error en la comunicación con el puerto serie
FD_INVALIDO	Descriptor del puerto inválido
TIMEOUT	Timeout al leer del puerto. El dispositivo no ha respondido en el tiempo esperado
RESPUESTA_ERRONEA	El dispositivo ha devuelto una respuesta considerada errónea
ERROR_MEM	No hay memoria dinámica reservada. Se puede producir por dos causas: • Si no se ha llamado a la función de inicialización antes de llamar a cualquier otra función • Si ha fallado la reserva dinámica de memoria durante la inicialización
NO_MODULO	El número de módulo al que se quiere acceder no existe en el bus local
NO_DINPUT	El módulo no es un módulo de entradas digitales
NO_DOUTPUT	El módulo no es un módulo de salidas digitales

Tabla 2.6 Variables de error en Ada

Si se produce el error RESPUESTA_ERRONEA, se puede obtener la respuesta que ha devuelto el dispositivo a través de la función *ObtenerErrorString*. En el epígrafe 2.5 se ofrece la descripción de las respuestas de error que pueden devolver los dispositivos FieldPoint.

2.3.3.- Descripción de las funciones

El argumento *modulo*, utilizado en varias funciones, hace referencia a la posición en el bus local del módulo al que se desea acceder.

Si la ejecución de una función da lugar a un error, se devuelve un valor negativo y se actualizan las variables internas de error. Se recomienda utilizar las funciones *ObtenerError* y *ObtenerErrorString* para obtener el código de error y una descripción de éste respectivamente.

2.3.3.1.- NI_Inicializar

◇ **Descripción**

Esta función sirve para inicializar el bus de comunicaciones de los módulos FieldPoint. Se debe invocar esta función antes de llamar a cualquier otra.

◇ **Sintaxis**

C/C++

```
int NI_Inicializar (int puerto,
                    int velocidad,
                    int dir_modulo);
```

Ada

```
function NI_Inicializar (Puerto: Puerto_Serie;
                         Velocidad: Integer;
                         Dir_Modulo: Integer)
return Integer;
```

◇ **Argumentos**

Puerto: Puerto serie del PC que se va ha emplear para la comunicación. Se deben utilizar las variables COM1 y COM2 definidas.

Velocidad: Velocidad del puerto serie a la que se va a llevar a cabo la comunicación. Este valor ha de coincidir con el valor fijado en el módulo de red. Los valores permitidos son: 300, 1200, 2400, 9600, 19200, 28400, 57600, 115200. Si se indica un valor distinto a estos se producirá un error.

Dir_Modulo: Dirección física que se ha asignado al módulo de red.

◇ **Valor retornado**

0 si todo fue correcto.

-1 en caso de error.

2.3.3.2.- NI_Liberar

◇ **Descripción**

Esta función libera el bus de comunicaciones y los recursos comprometidos durante la inicialización.

◇ **Sintaxis**

C/C++

```
int NI_Liberar (void);
```

Ada

```
function NI_Liberar return Integer;
```

◇ **Valor retornado**

0 si todo fue correcto.

-1 en caso de error.

2.3.3.3.- NI_EscribeLinea_DO

◇ **Descripción**

Esta función actualiza el valor de un canal de salida de un módulo de salidas digitales FP-RLY-420.

◇ **Sintaxis**

C/C++

```
int NI_EscribeLinea_DO (const int modulo,
                        const int linea,
                        const int valor);
```

Ada

```
function NI_EscribeLinea_DO (Modulo: Integer;
                             Linea: Num_Linea_Salida;
                             Valor: U1) return Integer;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

Linea: Número del canal de salida que se desea actualizar (entre 0 y 7).

Valor: Valor de la línea a actualizar (1: activar; 0: desactivar).

◇ **Valor retornado**

0 si todo fue correcto.

-1 en caso de error.

2.3.3.4.- NI_EscribeModulo_DO

◇ **Descripción**

Esta función actualiza los ocho canales del módulo de salidas digitales FP-RLY-420.

◇ **Sintaxis**

C/C++

```
int NI_EscribeModulo_DO (const int modulo,
                         const U8 valor);
```

Ada

```
function NI_EscribeModulo_DO (Modulo: Integer;
                              Valor: U8)
return Integer;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

Valor: Entero donde estén codificados los valores de los ocho canales del módulo.

◇ **Valor retornado**

El número de canales del módulo si todo fue correcto.

-1 en caso de error.

2.3.3.5.- NI_LeeLinea_DI

◇ **Descripción**

Esta función lee el estado de un canal de entrada de un módulo de entradas digitales FP-DI-301.

◇ **Sintaxis**

C/C++

```
int NI_LeeLinea_DI (const int modulo,
                    const int linea);
```

Ada

```
function NI_LeeLinea_DI (Modulo: Integer;
                         Linea: Num_Linea_Entrada)
return Integer;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

Linea: Número del canal de entrada que se desea leer (entre 0 y 15).

◇ **Valor retornado**

El estado del canal de entrada (1: activo; 0: inactivo) si todo fue correcto.

Un número negativo si hubo algún error.

2.3.3.6.- NI_LeeModulo_DI

◇ **Descripción**

Esta función lee el estado de los dieciséis canales de entrada del módulo de entradas digitales FP-DI-301. El estado de las entradas se devuelve codificado en un entero que se ha de pasar a la función como argumento.

◇ **Sintaxis**

C/C++

```
int NI_LeeModulo_DI (int modulo,
                     U16 * valor);
```

Ada

```
procedure NI_LeeModulo_DI (Modulo: in Integer;
                           Valor: in out U16;
                           Retorno: out Integer);
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

Valor: Entero donde se codificará el estado de los dieciséis canales de entrada del módulo.

◇ **Valor retornado**

El número de líneas del módulo si todo fue correcto.

-1 en caso de error.

La implementación de esta función en Ada se hace a través de un procedimiento. El valor de retorno se devuelve a través de la variable *Retorno.*

2.3.3.7.- ObtenerError

◇ **Descripción**

Esta función sirve para obtener el estado en que finalizó la última operación que se ha realizado.

◇ **Sintaxis**

C/C++

```
int ObtenerError (void);
```

Ada

```
function ObtenerError return Var_Error;
```

◇ **Valor retornado**

C/C++

Un entero que identifica la situación en que finalizó la última operación. Los posibles valores están definidos en las variables de error.

Ada

Una variable de tipo *Var_Error* (valor enumerado donde se definen los errores).

2.3.3.8.- ObtenerErrorString

◇ **Descripción**

Esta función sirve para obtener una descripción del estado en que finalizó la última operación realizada. Si en la última llamada a una función se ha producido un error, mediante una llamada a esta función se obtendrá una descripción de éste.

◇ **Sintaxis**

C/C++

```
char * ObtenerErrorString (void);
```

Ada

```
function ObtenerErrorString return string;
```

◇ **Valor retornado**

Una cadena con la descripción del estado en que finalizó la última operación realizada. Si en la última operación no se produjo ningún error, se retornará una cadena vacía.

2.3.3.9.- Num_Lineas

◇ **Descripción**

Esta función proporciona el número de canales que forman un módulo.

◇ **Sintaxis**

C/C++

```
int Num_Lineas (int modulo);
```

Ada

```
function Num_Lineas (Modulo: Integer) return Integer;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

◇ **Valor retornado**

El número de canales del módulo si todo fue correcto.

-1 en caso de error.

2.3.3.10.- Num_Modulos

◇ **Descripción**

Esta función proporciona el número de módulos que se han detectado en el bus local.

◇ **Sintaxis**

C/C++

```
int Num_Modulos (void);
```

Ada

```
function Num_Modulos return Integer;
```

◇ **Valor retornado**

El número de módulos que forman el bus local si todo fue correcto.

-1 en caso de error.

2.3.3.11.- Get_Tipo

◇ **Descripción**

Esta función retorna información sobre el tipo del módulo (entradas digitales, salidas digitales,…).

◇ **Sintaxis**

C/C++

```
int Get_Tipo (int modulo);
```

Ada

```
function Get_Tipo (Modulo: Integer) return Device_Type;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

◇ **Valor retornado**

C/C++

El número que identifica el tipo de módulo (variables *_Digital_IN* y *_Digital_OUT*).

-1 en caso de error.

Ada

Una variable de tipo *Device_Type* (valor enumerado donde están definidos los tipos de dispositivos). En caso de error el valor del enumerado retornado será *ERROR.*

2.3.3.12.- Get_Nombre

◇ **Descripción**

Esta función permite obtener el nombre de un módulo.

◇ **Sintaxis**

C/C++

```
char * Get_Nombre (int modulo);
```

Ada

```
function Get_Nombre (Modulo: Integer) return String;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

◇ **Valor retornado**

Una cadena con el nombre del módulo si todo fue correcto.

Una cadena con la descripción del error sucedido en caso de error.

2.3.3.13.- Get_Id

◇ **Descripción**

Esta función permite obtener el identificador de un módulo. Este identificador es único para cada tipo de módulo y se corresponde con el valor asignado por el fabricante.

◇ **Sintaxis**

C/C++

```
int Get_Id (int modulo);
```

Ada

```
function Get_Id (Modulo: Integer) return Integer;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

◇ **Valor retornado**

El número del fabricante que identifica el módulo si todo fue correcto.

-1 en caso de error.

2.3.3.14.- Get_Direccion

◇ **Descripción**

Esta función permite obtener la dirección física que tiene asignada un módulo.

◇ **Sintaxis**

C/C++

```
int Get_Direccion (int modulo);
```

Ada

```
function Get_Direccion (Modulo: Integer) return Integer;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

◇ **Valor retornado**

La dirección física que tiene asignada el módulo si todo fue correcto.

-1 en caso de error.

2.4.- Anexo A: Archivos de cabecera

DriverC_NI.h

```
// Autor: Adolfo Antonio Fernández Trabanco
// Junio 2005
//
// Archivo: DriverC_NI.h
// Este archivo contiene la especificación de las funciones que forman
// el driver para los módulos FieldPoint de la empresa National Instruments
//
// Módulos involucrados: FP-1000, FP-DI-301 Y FP-RLY-420


#ifndef DriverC_NI_H
#define DriverC_NI_H


//Definición de tipos
//Para valores de 8 bits (rango de 0 a 255)
typedef unsigned char  U8;
//Para valores de 16 bits (rango de 0 a 65535)
typedef unsigned short U16;


//Variables para identificar el puerto serie utilizado
#define COM1   1
#define COM2   2


//VARIABLES para DeviceType
#define _Digital_IN    1
#define _Digital_OUT   2


//VARIABLES DE ERROR
#define   CORRECTO             0
//error en la comunicación con el puerto serie
#define   ERROR_COM         -101
#define   TIMEOUT           -103 //timeout al leer del puerto
#define   FD_INVALIDO       -102 //descriptor del puerto inválido
#define   RESPUESTA_ERRONEA -200 //respuesta errónea del dispositivo
#define   ERROR_MEM         -300 //error al asignar memoria dinámica
```

```
//el numero de modulo al que se quiere acceder no existe en el bus
#define   NO_MODULO         -400
//el módulo no es un módulo de entradas digitales
#define   NO_DINPUT         -401
//el módulo no es un módulo de salidas digitales
#define   NO_DOUTPUT        -402
//la línea a la que se quiere acceder no existe
#define   NO_LINEA          -500


int NI_Inicializar (int puerto, int velocidad, int dir_modulo);

int NI_Liberar (void);

int NI_EscribeLinea_DO (const int modulo, const int linea,
                        const int valor);

int NI_EscribeModulo_DO (const int modulo, const U8 valor);

int NI_LeeLinea_DI (const int modulo, const int linea);

int NI_LeeModulo_DI (int modulo, U16 * valor);

int ObtenerError (void);

char * ObtenerErrorString (void);

int Num_Lineas (int modulo);

int Num_Modulos (void);

int Get_Tipo (int modulo);

char * Get_Nombre (int modulo);

int Get_Id (int modulo);

int Get_Direccion (int modulo);

#endif
```

driverada_ni.ads

```
-- Autor: Adolfo Antonio Fernández Trabanco
-- Junio 2005
--
-- Archivo: driverada_ni.ads
-- Este archivo contiene la especificación en lenguaje Ada de las funciones
que forman
-- el driver para los módulos FieldPoint de la empresa National Instruments
--
-- Módulos involucrados: FP-1000, FP-DI-301 Y FP-RLY-420


package DriverAda_NI is


--Definición de tipos
subtype Num_Linea_Entrada is Integer range 0 .. 15;
subtype Num_Linea_Salida is Integer range 0 .. 7;

subtype U1 is Integer range 0 .. 1;
subtype U8 is Integer range 0 .. (2**8)-1;
subtype U16 is Integer range 0 .. (2**16)-1;

type Puerto_Serie is (COM1, COM2);
for Puerto_Serie'Size use Integer'Size;
for Puerto_Serie use (COM1 => 1,
                      COM2 => 2);

type Device_Type is (DIGITAL_IN, DIGITAL_OUT, ERROR);
for Device_Type'Size use Integer'Size;
for Device_Type use (ERROR => -1,
                     DIGITAL_IN  => 1,
                     DIGITAL_OUT => 2);
```

```
--Variables de error
type Var_Error is (CORRECTO,
--error en la comunicación con el puerto serie
                   ERROR_COM,
                   TIMEOUT,     --timeout al leer del puerto
                   FD_INVALIDO, --descriptor del puerto inválido
                   RESPUESTA_ERRONEA, --respuesta errónea del dispositivo
                   ERROR_MEM,   --error al asignar memoria dinámica
--el numero de modulo al que se quiere acceder no existe en el bus
                   NO_MODULO,
--el módulo no es un módulo de entradas digitales
                   NO_DINPUT,
--el módulo no es un módulo de salidas digitales
                   NO_DOUTPUT);

for Var_Error'Size use Integer'Size;


function NI_Inicializar (Puerto: Puerto_Serie; Velocidad: Integer;
                         Dir_Modulo: Integer) return Integer;

function NI_Liberar return Integer;

function NI_EscribeLinea_DO (Modulo: Integer; Linea: Num_Linea_Salida;
Valor: U1) return Integer;

function NI_EscribeModulo_DO (Modulo: Integer; Valor: U8) return Integer;

function NI_LeeLinea_DI (Modulo: Integer; Linea: Num_Linea_Entrada)
return Integer;

procedure NI_LeeModulo_DI (Modulo: in Integer; Valor: in out U16;
                           Retorno: out Integer);

function ObtenerError return Var_Error;

function ObtenerErrorString return string;

function Num_Lineas (Modulo: Integer) return Integer;

function Num_Modulos return Integer;
```

```
function Get_Tipo (Modulo: Integer) return Device_Type;

function Get_Nombre (Modulo: Integer) return String;

function Get_Id (Modulo: Integer) return Integer;

function Get_Direccion (Modulo: Integer) return Integer;

end DriverAda_NI;
```

2.5.- Anexo B: FieldPoint Responses

En este anexo se proporciona la descripción de los errores que pueden devolver los dispositivos. Esta información se ha obtenido directamente del capítulo 2 "*FieldPoint Responses*" del manual de usuario "*FP-1000/1001 Programmer Reference Manual*".

A FieldPoint module returns an error response when an erroneous condition is detected during the reception or execution of a command. FieldPoint modules return only the Standard errors (N00 through N07) in response to all standard commands, which enables FieldPoint modules to work with host software that is written for the Optomux protocol. In response to the extended commands, FieldPoint modules return either standard or extended errors, depending on which is most appropriate. The error response to a FieldPoint command (standard or extended) has the following form:
N[error number][cr] where [error number] is two ASCII-hex characters.

Error Number (Hex)	Error Tag	Description
00	E_PUCLR_EXP	**Power Up Clear** expected. The command was ignored. A command other than **Power Up Clear** was attempted after power-up or power failure. Once the error is received by the host, it is unnecessary to send the **Power Up Clear** command, because the next command is executed normally. If this error message is received, the FieldPoint network module has gone through its power-up sequence and has reset all characteristics to defaults. If SnapShot is enabled, the module is configured in accordance with the stored SnapShot information.
01	E_INVALID_CMD	Undefined command. The command character was not a legal command character, or the addressed module does not support this command. The command was ignored.
02	E_BAD_CHECKSUM	Checksum error. The checksum received in the command did not match the calculated checksum of the characters in the command. The command was ignored.
03	E_INBUF_OVRFLO	Input buffer overrun. The received command was too long. The command was ignored.
04	E_ILLEGAL_CHAR	Non-printable ASCII character received. Only characters from ASCII value 33 to 127 are permitted within commands. The command was ignored.
05	E_INSUFF_CHARS	Data field error. An insufficient or incorrect number of characters were received for the specified command.
06	E_WATCHDOG_TMO	Communications link network watchdog timed out. The command was ignored.
07	E_INV_LIMS_GOT	Specified limits invalid for the command. This includes notification that an invalid digit (hex or decimal) was received.

Tabla 2.7 Standard Error Responses

Error Number (Hex)	Error Tag	Description
80	E_ILLEGAL_DIGIT	One or more characters sent in the command could not be correctly converted to a digit (hex or decimal).
81	E_BAD_ADDRESS	The command is valid, but the addressed module does not support the command received.
82	E_INBUF_FRMERR	The FieldPoint network module detected a serial framing error in the command. The command was ignored.
83	E_NO_MODULE	The addressed module does not exist.
84	E_INV_CHNL	One or more channels specified in the command either do not exist or do not support the operation specified. The command was ignored.
85	E_INV_RANGE	One or more ranges specified in the command either do not exist or do not support the setting specified. The command was ignored.
86	E_INV_ATTR	One or more attributes specified in the command either do not exist or do not support the setting specified. The command was ignored.
88	E_HOTSWAP	The module has been hot-swapped since it was last sent a command. This response is sent only if the network module is enabled to report hot-swaps, and if the hot-swap occurred after hot-swap reporting mode was enabled. The command was ignored. This error number can be sent in response to a standard command if you have enabled hot-swap reporting for the bank.
89	E_ADDR_NOT_SAME	The module addressed by the Resend Last Response is not the same as the module addressed by the previous command.
8A	E_NO_RESEND_BUF	The response to the last command is unavailable
8B	E_HW_FAILURE	An irrecoverable fault has occurred.
8C	E_UNKNOWN	An unidentifiable error condition has occurred.

Tabla 2.8 Extended Error Responses

3.- NuDAM, manual de usuario

3.1.- Instalación

Dado que los módulos NuDAM se comunican a través del puerto serie, es necesario verificar que el fichero asociado al puerto que se va a utilizar en la comunicación, tiene permisos de lectura y escritura para todos los usuarios.

En los sistemas Linux, los periféricos se encuentran representados como simples ficheros del sistema de archivos. Cada uno de los puertos serie tiene asociado uno o más ficheros. En la Tabla 3.1 se muestra la asociación más habitual entre los puertos serie y los ficheros del sistema de archivos.

Puerto	Fichero
COM1	/dev/ttyS0
COM2	/dev/ttyS1
COM3	/dev/ttyS2
COM4	/dev/ttyS3

Tabla 3.1 Asociación de puertos serie a ficheros de dispositivo

Para permitir que cualquier usuario pueda leer y escribir en el segundo puerto serie, se cambiarán los permisos del fichero asociado, ejecutando como usuario root el siguiente comando:

```
chmod 666 /dev/ttyS1
```

Para utilizar la librería, no es necesario realizar ningún proceso de instalación adicional. Es suficiente con copiar en el directorio de trabajo los ficheros adecuados:

- Si se va a utilizar la versión en Ada, se deben copiar los ficheros *ComPort.h*, *ComPort.c*, *DriverC_ND.h*, *DriverC_ND.c*, *driverada_nd.ads* y *driverada_nd.adb*.
- Si se va a utilizar la versión en C, se deben copiar los ficheros *ComPort.h*, *ComPort.c*, *DriverC_ND.h* y *DriverC_ND.c*.

En el epígrafe 3.2.4, se explica el proceso a seguir para que el usuario cree sus propios programas de control, así como las instrucciones de compilación necesarias para crear el ejecutable.

En el CD que acompaña a la documentación, se encuentran los ficheros fuente que forman el driver, así como las aplicaciones de prueba.

Para utilizar las aplicaciones de prueba proporcionadas, se debe copiar la carpeta adecuada del CD al directorio de trabajo del usuario. Una vez copiada la carpeta, ejecutar el comando `make` para crear los ejecutables.

3.2.- Utilización

3.2.1.- Configuración de los módulos

La configuración de los distintos módulos que forman el sistema NuDAM se hace mediante software. Los parámetros que se pueden configurar son: la dirección física, la velocidad de comunicación y la paridad.

En una red formada por módulos NuDAM no puede haber dos módulos con la misma dirección física. Si se desea añadir un nuevo módulo al bus, se debe asegurar que la configuración del módulo es la correcta. Para llevar a cabo la configuración es necesario, previamente, poner el módulo en *"Default State"*. La forma de situar al módulo en este estado consiste, normalmente, en unir la salida marcada como DEFAULT* a tierra (GND). Se recomienda obtener del manual del módulo la forma exacta de proceder para poner el módulo en este estado.

El fabricante, para llevar a cabo la tarea de configuración, proporciona una herramienta software de administración para los módulos NuDAM, *"NuDAM Administration Utility for Windows"*. Este programa está disponible en los CDs que se proporcionan junto con los módulos y se debe instalar en un sistema operativo Windows.

3.2.2.- El bus local

Cada uno de los módulos NuDAM dispone de una dirección física que lo identifica dentro del bus de comunicaciones. Estas direcciones ni siquiera tienen porque ser correlativas, de tal forma que podríamos tener dos módulos, uno con dirección cinco y otro con dirección veinticinco. Para utilizar las funciones proporcionadas, el usuario no necesita conocer en ningún momento las direcciones físicas de los módulos conectados al bus. El argumento *modulo* que aparece en las funciones hace referencia a la dirección del módulo en el **bus local**. Para conocer la dirección en el bus local de un módulo, se debe ejecutar el programa de prueba ***list_all*** proporcionado con las fuentes (ver epígrafe 3.2.5.1).

La detección de los módulos que forman el bus se hace durante la fase de inicialización. En esta fase se buscan módulos en el bus entre las direcciones cero y *limite*, siendo *limite* un valor indicado por el usuario. Se comprueba que exista el módulo con *dirección x* en el bus y, en caso afirmativo, se le asigna una posición en un *bus local*, que será el valor empleado en las funciones para acceder a los módulos. Estas posiciones son correlativas y se asignan, empezando por cero, según se van encontrando módulos. De esta forma podríamos tener conectados al bus tres módulos con direcciones físicas cinco, ciento cinco y noventa y durante la fase de inicialización se les asignaría las direcciones cero, dos y uno en el bus local respectivamente (Tabla 3.2).

	Módulo ND-6053	Módulo ND-6058	Módulo ND-6053
Dirección física	5	105	90
Posición en el bus	0	2	1

Tabla 3.2 Ejemplo de asignación de direcciones en el bus local

3.2.3.- Problemas conocidos

Durante la fase de desarrollo se produjeron ciertos inconvenientes producidos por un comportamiento extraño por parte de los módulos.

Tras proporcionar la tensión de alimentación a los distintos módulos que forman el bus, el sistema está dispuesto para funcionar. Cuando el sistema se mantiene alimentado durante mucho tiempo, hay ocasiones en que los módulos se apagan automáticamente y, transcurrido un tiempo, vuelven a estar operativos. Aún no se conocen las causas que producen este comportamiento, pues sucede en circunstancias totalmente diferentes.

3.2.4.- Creación de un programa

Los pasos a seguir para obtener el ejecutable del programa realizado varían según el lenguaje de programación utilizado. Para simplificar el proceso de compilación de los programas, se recomienda al usuario que confeccione su propio fichero *Makefile*. Junto a las fuentes, se proporcionan ficheros *Makefile* que pueden servir de ejemplo.

Para crear un programa en lenguaje C que utilice las funciones de la librería, es necesario seguir los pasos indicados a continuación:

- Incluir en el código del programa la cabecera "*DriverC_ND.h*".

```
#include <DriverC_ND.h>
```

- Realizar el proceso de compilación para generar el fichero ejecutable. Se deben realizar los siguientes pasos:
  - Compilar los ficheros fuente que componen el driver:

```
g++ -I. -c ComPort.c
g++ -I. -c DriverC_ND.c
```

- Compilar el programa del usuario (supongamos que el fichero se llama ejemplo.cc):

```
g++ -I. -c ejemplo.cc
```

- Enlazar todos los ficheros objeto para obtener el ejecutable final:

```
g++ -g ComPort.o DriverC_ND.o ejemplo.o -o ejemplo
```

Para crear un programa en lenguaje Ada que utilice las funciones de la librería, es necesario seguir los siguientes pasos:

- Incluir en el código del programa el paquete "*DriverAda_ND*".

```
with DriverAda_ND; use DriverAda_ND;
```

- Realizar el proceso de compilación para generar el fichero ejecutable. Se deben realizar los siguientes pasos:
  - Compilar los ficheros fuente que componen el driver:

```
gcc -c -I. ComPort.c
gcc -c -I. DriverC_ND.c
```

  - Compilar el programa del usuario (supongamos que el fichero se llama ejemplo.adb):

```
gnatmake -c ejemplo.adb
```

  - Enlazar todos los ficheros objeto para obtener el ejecutable final:

```
gnatbind -x ejemplo
gnatlink ejemplo ComPort.o DriverC_ND.o
```

3.2.5.- Aplicaciones de prueba

Junto con las fuentes, se proporcionan un conjunto de programas genéricos que permiten al usuario interaccionar con el proceso e iniciarse en el uso de las distintas funciones. Son programas en modo consola, carentes de interfaz gráfica, y se dispone de dos versiones para cada uno de ellos: una implementada en lenguaje C y otra implementada en lenguaje Ada.

3.2.5.1.- list_all

Esta aplicación muestra por pantalla el número de módulos que forman el bus local y las características de cada uno de ellos: dirección en el bus local, dirección física, tipo de módulo, nombre, identificador, número de canales, etc.

Para ejecutar la aplicación es necesario indicarle tres parámetros en la orden de ejecución:

- El puerto serie a utilizar:
  Indicar un 1 si se va utilizar el puerto serie COM1.
  Indicar un 2 si se va utilizar el puerto serie COM2.
- La velocidad de comunicación del puerto serie: Este valor ha de coincidir con la velocidad configurada en los módulos.
- La dirección límite: Indica hasta que dirección física se buscarán módulos en el bus.

Si se omite alguno de estos parámetros, la aplicación informa al usuario de esta situación y le indica los parámetros que ha de introducir (Ejemplo 3.1). En el Ejemplo 3.2 se puede ver una ejecución correcta del programa.

```
[txolfo@localhost ada]$ ./list_all

Uso: ./list_all <puerto> <velocidad> <limite>

<puerto> Para el puerto serie COM1 indicar un  1
         Para el puerto serie COM2 indicar un  2
<velocidad> Indicar la velocidad del puerto
<limite> Indicar la dirección máxima hasta la cual se desean buscar módulos
en el bus

[txolfo@localhost ada]$
```

Ejemplo 3.1 Ejecución de *list_all* incorrecta

```
[txolfo@localhost ada]$ ./list_all 2 9600 5

A continuación se va a realizar una lectura del bus entre las direcciones 0
y 5
Esta operación puede tardar varios minutos dependiendo de la cantidad de
módulos a buscar
Pulse intro para continuar...

Posición en el bus:  0
  Tipo de dispositivo: Entradas Digitales
  Dirección física del módulo: 1
  Nombre del módulo: ND-6053
  Número del módulo: 6053
  Número de líneas: 16

Posición en el bus:  1
  Tipo de dispositivo: Salidas Digitales
  Dirección física del módulo: 2
  Nombre del módulo: ND-6058
  Número del módulo: 6058
  Número de líneas: 24

[txolfo@localhost ada]$
```

Ejemplo 3.2 Ejecución de *list_all* correcta

3.2.5.2.- entradas

Esta aplicación muestra por pantalla el estado de los canales digitales de entrada del módulo de entradas digitales ND-6053 que le indiquemos.

Para ejecutar la aplicación es necesario indicarle tres parámetros en la orden de ejecución:

- El puerto serie a utilizar:
  Indicar un 1 si se va utilizar el puerto serie COM1.
  Indicar un 2 si se va utilizar el puerto serie COM2.
- La velocidad de comunicación del puerto serie: Este valor ha de coincidir con la velocidad configurada en los módulos.
- La dirección límite: Indica hasta qué dirección física se buscarán módulos en el bus.

Al ejecutar la aplicación, nos informa del número de módulos que se han detectado en el bus local y nos pide que indiquemos la posición en el bus del módulo de entradas sobre el que deseamos ejecutar la prueba. Si en este paso se indica una posición en el bus local que no existe o que no se corresponde con un módulo de entradas digitales, se informará al usuario de esta circunstancia y se le pedirá que introduzca un valor válido (Ejemplo 3.3). Tras indicar una posición correcta, se muestra por pantalla el estado de las entradas (Ejemplo 3.4).

```
[txolfo@localhost ada]$ ./entradas 2 9600 5

ND_Inicializar:
 0 CORRECTO

Se han encontrado 2 módulos en el rango 0 .. 5
Indique la posición en el bus del módulo de entradas sobre el que actuar: 1

El número de módulo indicado es INCORRECTO
Indique la posición en el bus del módulo de entradas sobre el que actuar: 9

El número de módulo indicado es INCORRECTO
Indique la posición en el bus del módulo de entradas sobre el que actuar:
```

Ejemplo 3.3 Ejecución de *entradas*

```
[txolfo@localhost ada]$ ./entradas 2 9600 5

ND_Inicializar:
 0 CORRECTO

Se han encontrado 2 módulos en el rango 0 .. 5
Indique la posición en el bus del módulo de entradas sobre el que actuar: 0


ND_LeeModulo_DI:
 16 CORRECTO
Estado de las entradas en decimal: 37952
Estado de las entradas en hexadecimal: 16#9440#

ND_LeeLinea_DI:
 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
  1  0  0  1  0  1 0 0 0 1 0 0 0 0 0 0

Volver a mostrar las entradas?? (s/n) n

ND_Liberar:
 0 CORRECTO

[txolfo@localhost ada]$
```

Ejemplo 3.4 Ejecución completa de *entradas*

3.2.5.3.- salidas

Esta aplicación permite interaccionar con el proceso mediante la activación de los distintos canales de salida de la tarjeta DIN-24R conectada al módulo ND-6058.

Para ejecutar la aplicación es necesario indicarle tres parámetros en la orden de ejecución:

- El puerto serie a utilizar:
  Indicar un 1 si se va utilizar el puerto serie COM1.
  Indicar un 2 si se va utilizar el puerto serie COM2.
- La velocidad de comunicación del puerto serie: Este valor ha de coincidir con la velocidad configurada en los módulos.
- La dirección límite: Indica hasta que dirección física se buscarán módulos en el bus.

Al ejecutar la aplicación, nos informa del número de módulos que se han detectado en el bus local y nos pide que indiquemos la posición en el bus del módulo de salidas sobre el que deseamos ejecutar la prueba. Si en este paso se indica una posición en el bus local que no existe o que no se corresponde con un módulo de salidas digitales, se informará al usuario de esta circunstancia y se le pedirá que introduzca un valor válido (Ejemplo 3.5).

```
[txolfo@localhost ada]$ ./salidas 2 9600 5

ND_Inicializar:
 0 CORRECTO

Se han encontrado 2 módulos en el rango 0 .. 5
Indique la posición en el bus del módulo de entradas sobre el que actuar: 0

El número de módulo indicado es INCORRECTO
Indique la posición en el bus del módulo de salidas sobre el que actuar: 9

El número de módulo indicado es INCORRECTO
Indique la posición en el bus del módulo de salidas sobre el que actuar:
```

Ejemplo 3.5 Ejecución de *salidas*

Tras indicar la posición, se solicita al usuario el puerto donde se encuentra el canal sobre el que actuar, el canal dentro de ese puerto y el valor ha asignar al canal. Si en este paso se indica un puerto erróneo o un número de canal incorrecto pueden suceder dos cosas, dependiendo de la versión utilizada:

- En la versión en C, se producirá un error y se indicará esta circunstancia por pantalla (Ejemplo 3.6).
- En la versión en Ada, se producirá una excepción en tiempo de ejecución.

En el Ejemplo 3.7 se muestra una ejecución completa de la aplicación.

```
[txolfo@localhost c]$ ./salidas 2 9600 5

ND_Inicializar
0 0

Se han encontrado 2 módulos en el rango 0 .. 5
Indique la posición en el bus del módulo de salidas sobre el que actuar: 1

ND_EscribeLinea_DI
Interacción con el modulo de salidas...

Posición del módulo de salidas en el bus: 1
Puerto donde se encuentra la línea (A, B, C): d
Linea: 1
Estado (1->Activar, 0->Desactivar) 0
-1 -600 (ND_EscribeLinea_DO) Número de puerto para la tarjeta erróneo (-1)
Continuar?? (s/n) s

Posición del módulo de salidas en el bus: 1
Puerto donde se encuentra la línea (A, B, C): a
Linea: 9
Estado (1->Activar, 0->Desactivar) 1
-1 -500 (ND_EscribeLinea_DO) Número de línea erróneo (9)
Continuar?? (s/n)
```

Ejemplo 3.6 Ejecución de *salidas* (versión en C)

```
[txolfo@localhost ada]$ ./salidas 2 9600 5

ND_Inicializar:
 0 CORRECTO

Se han encontrado 2 módulos en el rango 0 .. 5
Indique la posición en el bus del módulo de salidas sobre el que actuar: 1

ND_EscribeLinea_DO:
Interacción con los módulos de salidas...

Posición del modulo de salidas en el bus:  1
Puerto (A, B, C): a
Linea: 0
Estado (1->Activar, 0->Desactivar) 1

Continuar?? (s/n) n

ND_EscribeModulo_DO:
 24 CORRECTO

ND_Liberar:
 0 CORRECTO

[txolfo@localhost ada]$
```

Ejemplo 3.7 Ejecución completa de *salidas*

3.3.- Descripción de la librería

En este epígrafe se pretende dar la información necesaria al usuario para que pueda comenzar a realizar sus propios programas.

Existen dos interfaces distintas (una en lenguaje C y otra en lenguaje Ada) de manera que el usuario es libre de elegir para implementar su programa de control la que más se adapte a sus necesidades. Dado que ambas versiones difieren ligeramente (en cuanto a tipos definidos, variables globales utilizadas) se invita al usuario a leer con detenimiento las particularidades de cada una de ellas.

3.3.1.- Interfaz en C

3.3.1.1.- Tipos definidos en C

En el archivo de cabecera "*DriverC_ND.h*" (ver epígrafe 3.4) se definen los tipos de datos que son utilizados por la librería. Se recomienda al usuario utilizarlos en la implementación de sus programas. Los tipos definidos son:

- **U8** Precisión: 8 bits. Rango: 0-255
- **U16** Precisión: 16 bits. Rango: 0-65535

Estos tipos se utilizan en las funciones *ND_EscribePuerto_DO, ND_EscribeModulo_DO* y *ND_LeeModulo_DI*.

Un ejemplo de utilización del tipo U8 se muestra a continuación: supongamos que deseamos poner a uno los canales cero y cuatro del puerto A de la tarjeta DIN-24R conectada al módulo de salidas digitales ND-6058. Tendríamos la siguiente situación:

bit	7	6	5	4	3	2	1	0
estado	0	0	0	1	0	0	0	1
máscara	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Para llamar a la función *ND_EscribePuerto_DO* se codificaría en un dato de tipo U8 la información mostrada, lo que equivale a mandar el valor U8 = $2^0 + 2^4 = 1 + 16 = 17$.

3.3.1.2.- Variables globales en C

En el archivo de cabecera "*DriverC_ND.h*" (ver epígrafe 3.4) se definen una serie de variables globales utilizadas por la librería y de utilidad para el usuario:

- **COM1** y **COM2**: Indican el puerto serie a utilizar en la comunicación. Estas variables se utilizan para realizar la llamada a la función *ND_Inicializar* (p.ej. `ND_Inicializar (COM2, 9600, 5)`)

- **PUERTO_A**, **PUERTO_B** y **PUERTO_C**: Indican el puerto de la tarjeta DIN-24R sobre el que actuar. Se deben utilizar en las funciones *ND_EscribeLinea_DO* y *ND_EscribePuerto_DO* para indicar el puerto que se desea actualizar (p.ej. `ND_EscribePuerto_DO (modulo, PUERTO_A, 0)`).

- **_Digital_IN** y **_Digital_OUT**: Indican el tipo de módulo: entradas digitales y salidas digitales respectivamente. Estas variables se utilizan en la función *Get_Tipo*.

```
tipo = Get_Tipo (i);
switch (tipo){
case _Digital_IN:
  cout << "\tTipo de dispositivo: Entradas Digitales" << endl;
  break;
case _Digital_OUT:
  cout << "\tTipo de dispositivo: Salidas Digitales" << endl;
  break;
}
```

- **Variables de error**: Proporcionan información sobre el estado en que finalizó la última operación. En el epígrafe 3.3.1.3 se puede obtener una descripción de cada una de ellas. Siempre que se produzca un error, se recomienda utilizar las funciones *ObtenerError* y *ObtenerErrorString* para obtener el código de error y una descripción de éste respectivamente.

3.3.1.3.- Variables de error en C

En la Tabla 3.3 se muestran las variables de error utilizadas, el valor que toman cada una de ellas y la descripción.

Error	Valor	Descripción
CORRECTO	0	En la última operación no se produjo ningún error
ERROR_COM	-101	Error en la comunicación con el puerto serie
FD_INVALIDO	-102	Descriptor del puerto inválido
TIMEOUT	-103	Timeout al leer del puerto. El dispositivo no ha respondido en el tiempo esperado
RESPUESTA_ERRONEA	-200	El dispositivo ha devuelto una respuesta considerada errónea
ERROR_MEM	-300	No hay memoria dinámica reservada. Se puede producir por dos causas: • Si no se ha llamado a la función de inicialización antes de llamar a cualquier otra función • Si ha fallado la reserva dinámica de memoria durante la inicialización
NO_MODULO	-400	El número de módulo al que se quiere acceder no existe en el bus local
NO_DINPUT	-401	El módulo no es un módulo de entradas digitales
NO_DOUTPUT	-402	El módulo no es un módulo de salidas digitales
NO_LINEA	-500	La línea a la que se quiere acceder no existe
NO_PORT	-600	El puerto especificado no es un puerto válido para la tarjeta DIN-24R
BUS_VACIO	-700	No se ha encontrado ningún módulo en el bus

Tabla 3.3 Variables de error en C

Si se produce el error RESPUESTA_ERRONEA, mediante la función *ObtenerErrorString* se puede obtener la respuesta que devolvió el dispositivo.

3.3.2.- Interfaz en Ada

3.3.2.1.- Tipos definidos en Ada

En el archivo de cabecera "*driverada_nd.ads*" (ver epígrafe 3.4) se definen los tipos de datos que son utilizados por la librería. Se recomienda al usuario utilizarlos en la implementación de sus programas. Vamos a diferenciar entre los tipos básicos y los tipos enumerados.

Todos los tipos básicos son subconjuntos del tipo estándar *Integer* limitados en el rango. En aquellas funciones donde se utilicen estos tipos como parámetros, se debe respetar este rango pues, en caso contrario, se producirá una excepción en tiempo de ejecución. Los tipos básicos definidos son:

- **U1** Rango: 0-1
- **U8** Rango: 0-255
- **U16** Rango: 0-65535
- **Num_Linea_Entrada** Rango: 0-15
- **Num_Linea_Salida** Rango: 0-7
- **Num_Modulo** Rango: 0-255

Vemos que en este caso se acotan el número de los canales de entrada y salida (mediante los tipos Num_Linea_Entrada y Num_Linea_Salida respectivamente), de manera que desaparece el error NO_LINEA que había en la implementación en C. Ahora si se pasa un número de línea erróneo a una función, se producirá una excepción en tiempo de ejecución.

También se limita el número de módulos a los que se puede acceder. Esta limitación la impone el fabricante, pues sólo permite direccionar 256 módulos. Al igual que en el caso anterior, si se realiza una llamada a una función que tenga como argumento un tipo *Num_Modulo* con un valor fuera del rango, se producirá una excepción en tiempo de ejecución.

Vamos a mostrar un ejemplo de utilización del tipo U8 mediante la función *ND_EscribePuerto_DO*. Supongamos que deseamos poner a uno los canales cero

y cuatro del puerto A de la tarjeta DIN-24R conectada al módulo de salidas digitales ND-6058. Tendríamos la siguiente situación:

bit	7	6	5	4	3	2	1	0
estado	0	0	0	1	0	0	0	1
máscara	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Para llamar a la función *ND_EscribePuerto_DO* se codificaría en un dato de tipo U8 la información mostrada, lo que equivale a mandar el valor U8 = $2^0 + 2^4 = 1 + 16 = 17$.

Mediante el uso de tipos enumerados se declaran las variables necesarias para acceder al puerto serie, definir los puertos de la tarjeta DIN-24R, definir el tipo de módulo y definir las variables de error:

- **Puerto_Serie** (valores COM1 y COM2): Indican el puerto serie a utilizar en la comunicación. Estas variables se utilizan para realizar la llamada a la función *ND_Inicializar* (p.ej. `ND_Inicializar (COM2, 9600, 5)`)

- **Puerto_Tarjeta** (PUERTO_A, PUERTO_B y PUERTO_C): Indican el puerto de la tarjeta DIN-24R sobre el que actuar. Se deben utilizar en las funciones *ND_EscribeLinea_DO* y *ND_EscribePuerto_DO* para indicar el puerto que se desea actualizar (por ejemplo `ND_EscribePuerto_DO (modulo, PUERTO_A, 0)`).

- **Device_Type** (valores DIGITAL_IN, DIGITAL_OUT y ERROR): Indican el tipo de módulo (entradas digitales, salidas digitales). Este enumerado se utiliza en la función *Get_Tipo*.

```
Tipo := Get_Tipo (I);
case Tipo is
   when Digital_IN =>
      Put ("  Tipo de dispositivo: Entradas Digitales");
      New_Line;
   when Digital_OUT =>
      Put ("  Tipo de dispositivo: Salidas Digitales");
      New_Line;
```

```
      when others =>
         Put ("  Error al obtener el tipo");
         New_Line;
   end case;
```

- **Var_Error**: Indican las posibles variables de error. Este enumerado se emplea en la función *ObtenerError*. En el epígrafe 3.3.2.2 se puede obtener una descripción de cada uno de los posibles valores que puede tomar una variable del tipo *Var_Error*. Siempre que se produzca un error se recomienda utilizar la función *ObtenerErrorString* para obtener una descripción de éste.

3.3.2.2.- Variables de error en Ada

En la Tabla 3.4 se muestran los valores que puede tomar el enumerado *Var_Error* y la descripción de cada uno de ellos.

Error	Descripción
CORRECTO	En la última operación no se produjo ningún error
ERROR_COM	Error en la comunicación con el puerto serie
FD_INVALIDO	Descriptor del puerto inválido
TIMEOUT	Timeout al leer del puerto. El dispositivo no ha respondido en el tiempo esperado
RESPUESTA_ERRONEA	El dispositivo ha devuelto una respuesta considerada errónea
ERROR_MEM	Error al asignar memoria dinámica. Si no se ha llamado a la función de inicialización antes de llamar a cualquier otra función se obtendrá este error.
NO_MODULO	El número de módulo al que se quiere acceder no existe en el bus
NO_DINPUT	El módulo no es un módulo de entradas digitales
NO_DOUTPUT	El módulo no es un módulo de salidas digitales
BUS_VACIO	No se ha encontrado ningún módulo en el bus

Tabla 3.4 Variables de error en Ada

Si se produce el error RESPUESTA_ERRONEA, mediante la función *ObtenerErrorString* se puede obtener la respuesta que devolvió el dispositivo.

3.3.3.- Descripción de las funciones

El argumento *modulo*, utilizado en varias funciones, hace referencia a la posición en el bus local del módulo al que se desea acceder.

Si la ejecución de una función da lugar a un error, se devuelve un valor negativo y se actualizan las variables internas de error. Se recomienda utilizar las funciones *ObtenerError* y *ObtenerErrorString* para obtener el código de error y una descripción de éste respectivamente.

3.3.3.1.- ND_Inicializar

◇ **Descripción**

Esta función sirve para inicializar el bus de comunicaciones de los módulos NuDAM. Se debe invocar esta función antes de llamar a cualquier otra.

◇ **Sintaxis**

C/C++

```
int ND_Inicializar (int puerto,
                    int velocidad,
                    int limite);
```

Ada

```
function ND_Inicializar (Puerto: Puerto_Serie;
                         Velocidad: Integer;
                         Limite: Num_Modulo)
return Integer;
```

◇ **Argumentos**

Puerto: Puerto serie del PC que se va ha emplear para la comunicación. Se deben utilizar las variables COM1 y COM2 definidas.

Velocidad: Velocidad del puerto serie a la que se va a llevar a cabo la comunicación. Este valor ha de coincidir con el valor fijado en los módulos. Los valores permitidos son: 300, 1200, 2400, 9600, 19200, 28400, 57600, 115200. Si se indica un valor distinto a estos se producirá un error.

Limite: Dirección física hasta la que se buscarán módulos en el bus.

◇ **Valor retornado**

0 si todo fue correcto.

-1 en caso de error.

3.3.3.2.- ND_Liberar

◇ **Descripción**

Esta función libera el bus de comunicaciones y los recursos comprometidos durante la inicialización.

◇ **Sintaxis**

C/C++

```
int ND_Liberar (void);
```

Ada

```
function ND_Liberar return Integer;
```

◇ **Valor retornado**

0 si todo fue correcto.

-1 en caso de error.

3.3.3.3.- ND_EscribeLinea_DO

◇ **Descripción**

Esta función actualiza el valor de una línea de salida de uno de los puertos de la tarjeta DIN-24R, conectada al módulo ND-6058.

◇ **Sintaxis**

C/C++

```
int ND_EscribeLinea_DO (const int modulo,
                        const int puerto,
                        const int linea,
                        const int valor);
```

Ada

```
function ND_EscribeLinea_DO (Modulo: Num_Modulo;
                             Puerto: Puerto_Tarjeta;
                             Linea: Num_Linea_Salida;
                             Valor: U1)
return Integer;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

Puerto: Puerto de la tarjeta donde se encuentra la línea a actualizar (utilizar las variables *PUERTO_A*, *PUERTO_B* o *PUERTO_C*).

Linea: Número de línea del puerto que se desea actualizar (entre 0 y 7).

Valor: Valor de la línea a actualizar (1: activar; 0: desactivar).

◇ **Valor retornado**

0 si todo fue correcto.

-1 en caso de error.

3.3.3.4.- ND_EscribePuerto_DO

◇ **Descripción**

Esta función actualiza los ocho canales de uno de los puertos de la tarjeta DIN-24R, conectada al módulo ND-6058.

◇ **Sintaxis**

C/C++

```
int ND_EscribePuerto_DO (const int modulo,
                         const int puerto,
                         const U8 valor);
```

Ada

```
function ND_EscribePuerto_DO (Modulo: Num_Modulo;
                              Puerto: Puerto_Tarjeta;
                              Valor: U8)
return Integer;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

Puerto: Puerto de la tarjeta que se desea actualizar (utilizar las variables *PUERTO_A*, *PUERTO_B* o *PUERTO_C*).

Valor: Entero donde estén codificados los valores de los ocho canales del puerto.

◇ **Valor retornado**

El número de canales del puerto si todo fue correcto.

-1 en caso de error.

3.3.3.5.- ND_EscribeModulo_DO

◇ **Descripción**

Esta función actualiza los tres puertos de la tarjeta DIN-24R, conectada al módulo ND-6058.

◇ **Sintaxis**

C/C++

```
int ND_EscribeModulo_DO (const int modulo,
                         const U8 valorA,
                         const U8 valorB,
                         const U8 valorC);
```

Ada

```
function ND_EscribeModulo_DO (Modulo: Num_Modulo;
                              ValorA: U8;
                              ValorB: U8;
                              ValorC: U8)
return Integer;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

ValorA: Entero donde estén codificados los valores de los ocho canales del puerto A de la tarjeta DIN-24R.

ValorB: Entero donde estén codificados los valores de las ocho canales del puerto B de la tarjeta DIN-24R.

ValorC: Entero donde estén codificados los valores de las ocho canales del puerto C de la tarjeta DIN-24R.

◇ **Valor retornado**

El número de líneas del módulo si todo fue correcto.

-1 en caso de error.

3.3.3.6.- ND_LeeLinea_DI

◇ **Descripción**

Esta función lee el estado de un canal de entrada de un módulo de entradas digitales ND-6053.

◇ **Sintaxis**

C/C++

```
int ND_LeeLinea_DI (const int modulo,
                    const int linea);
```

Ada

```
function ND_LeeLinea_DI (Modulo: Num_Modulo;
                         Linea: Num_Linea_Entrada)
return Integer;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

Linea: Número del canal de entrada que se desea leer (entre 0 y 15).

◇ **Valor retornado**

El estado del canal de entrada (1: activo; 0: inactivo) si todo fue correcto.

Un número negativo si hubo algún error.

3.3.3.7.- ND_LeeModulo_DI

◇ **Descripción**

Esta función lee el estado de los dieciséis canales de entrada del módulo de entradas digitales ND-6053. El estado de las entradas se devuelve codificado en un entero que se ha de pasar a la función como argumento.

◇ **Sintaxis**

C/C++

```
int ND_LeeModulo_DI (const int modulo,
                     U16 * valor);
```

Ada

```
procedure ND_LeeModulo_DI (Modulo: in Num_Modulo;
                           Valor: in out U16;
                           Retorno: out Integer);
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

Valor: Entero donde se codificará el estado de los dieciséis canales de entrada del módulo.

◇ **Valor retornado**

El número de líneas del módulo si todo fue correcto.

-1 en caso de error.

La implementación de esta función en Ada se hace a través de un procedimiento. El valor de retorno se devuelve a través de la variable *Retorno.*

3.3.3.8.- ObtenerError

◇ **Descripción**

Esta función sirve para obtener el estado en que finalizó la última operación que se ha realizado.

◇ **Sintaxis**

C/C++

```
int ObtenerError (void);
```

Ada

```
function ObtenerError return Var_Error;
```

◇ **Valor retornado**

C/C++

Un entero que identifica la situación en que finalizó la última operación. Los posibles valores están definidos en las variables de error.

Ada

Una variable de tipo *Var_Error* (valor enumerado donde se definen los errores).

3.3.3.9.- ObtenerErrorString

◇ **Descripción**

Esta función sirve para obtener una descripción del estado en que finalizó la última operación realizada. Si en la última llamada a una función se ha producido un error, mediante una llamada a esta función se obtendrá una descripción de éste.

◇ **Sintaxis**

C/C++

```
char * ObtenerErrorString (void);
```

Ada

```
function ObtenerErrorString return string;
```

◇ **Valor retornado**

Una cadena con la descripción del estado en que finalizó la última operación realizada. Si en la última operación no se produjo ningún error, se retornará una cadena vacía.

3.3.3.10.- Num_Lineas

◇ **Descripción**

Esta función proporciona el número de canales que forman el módulo.

◇ **Sintaxis**

C/C++

```
int Num_Lineas (int modulo);
```

Ada

```
function Num_Lineas (Modulo: Integer) return Integer;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

◇ **Valor retornado**

El número de canales del módulo si todo fue correcto.

-1 en caso de error.

3.3.3.11.- Num_Modulos

◇ **Descripción**

Esta función proporciona el número de módulos que se han detectado en el bus local.

◇ **Sintaxis**

C/C++

```
int Num_Modulos (void);
```

Ada

```
function Num_Modulos return Integer;
```

◇ **Valor retornado**

El número de módulos que forman el bus local si todo fue correcto.

-1 en caso de error.

3.3.3.12.- Get_Tipo

◇ **Descripción**

Esta función retorna información sobre el tipo del módulo (entradas digitales, salidas digitales,…).

◇ **Sintaxis**

C/C++

```
int Get_Tipo (int modulo);
```

Ada

```
function Get_Tipo (Modulo: Integer) return Device_Type;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

◇ **Valor retornado**

C/C++

El número que identifica el tipo de módulo (variables *_Digital_IN* y *_Digital_OUT*).

-1 en caso de error.

Ada

Una variable de tipo *Device_Type* (valor enumerado donde están definidos los tipos de dispositivos). En caso de error el valor del enumerado retornado será *ERROR.*

3.3.3.13.- Get_Nombre

◇ **Descripción**

Esta función permite obtener el nombre de un módulo.

◇ **Sintaxis**

C/C++

```
char * Get_Nombre (int modulo);
```

Ada

```
function Get_Nombre (Modulo: Integer) return String;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

◇ **Valor retornado**

Una cadena con el nombre del módulo si todo fue correcto.

Una cadena con la descripción del error sucedido en caso de error.

3.3.3.14.- Get_Id

◇ **Descripción**

Esta función permite obtener el identificador de un módulo. Este identificador es único para cada tipo de módulo y se corresponde con el valor asignado por el fabricante.

◇ **Sintaxis**

C/C++

```
int Get_Id (int modulo);
```

Ada

```
function Get_Id (Modulo: Integer) return Integer;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local.

◇ **Valor retornado**

El número del fabricante que identifica el módulo si todo fue correcto.

-1 en caso de error.

3.3.3.15.- Get_Direccion

◇ **Descripción**

Esta función permite obtener la dirección física que tiene asignada un módulo.

◇ **Sintaxis**

C/C++

```
int Get_Direccion (int modulo);
```

Ada

```
function Get_Direccion (Modulo: Integer) return Integer;
```

◇ **Argumentos**

Modulo: Posición del módulo en el bus local

◇ **Valor retornado**

La dirección física que tiene asignada el módulo si todo fue correcto.

-1 en caso de error

3.4.- Anexo A: Archivos de cabecera

DriverC_ND.h

```
// Autor: Adolfo Antonio Fernández Trabanco
// Junio 2005
//
// Archivo: DriverC_ND.h
// Este archivo contiene la especificación de las funciones que forman
// el driver para los módulos NuDAM de la empresa AdLink
//
// Módulos involucrados: ND-6053 y ND-6058

#ifndef DriverC_ND_H
#define DriverC_ND_H

//Definición de tipos
//Para valores de 8 bits (rango de 0 a 255)
typedef unsigned char  U8;
//Para valores de 16 bits (rango de 0 a 65535)
typedef unsigned short U16;

//Variables para identificar el puerto serie utilizado
#define COM1   1
#define COM2   2

//Variables para identificar el puerto en la tarjeta DIN-24R/24
#define PUERTO_A  0
#define PUERTO_B  1
#define PUERTO_C  2

//VARIABLES para DeviceType
#define _Digital_IN    1
#define _Digital_OUT   2

//VARIABLES DE ERROR
#define   CORRECTO             0
//error en la comunicación con el puerto serie
#define   ERROR_COM         -101
#define   TIMEOUT           -103 //timeout al leer del puerto
#define   FD_INVALIDO       -102 //descriptor del puerto inválido
```

```
#define   RESPUESTA_ERRONEA -200 //respuesta errónea del dispositivo
#define   ERROR_MEM         -300 //error al asignar memoria dinámica
//el numero de modulo al que se quiere acceder no existe en el bus
#define   NO_MODULO         -400
//el módulo no es un módulo de entradas digitales
#define   NO_DINPUT         -401
//el módulo no es un módulo de salidas digitales
#define   NO_DOUTPUT        -402
//la línea a la que se quiere acceder no existe
#define   NO_LINEA          -500
//el puerto de la tarjeta especificado no es válido
#define   NO_PORT           -600
//no se ha encontrado ningún módulo en el bus
#define   BUS_VACIO         -700

//Para todas las funciones:
//en caso de error utilizar las funciones ObtenerError, ObtenerErrorString
//para obtener el número de error y una descripción de este


int ND_Inicializar (int puerto, int velocidad, int limite);

int ND_Liberar (void);

int ND_EscribeLinea_DO (const int modulo, const int puerto,
                        const int linea, const int valor);

int ND_EscribePuerto_DO (const int modulo, const int puerto,
                         const U8 valor);

int ND_EscribeModulo_DO (const int modulo, const U8 valorA,
                         const U8 valorB, const U8 valorC);

int ND_LeeLinea_DI (const int modulo, const int linea);

int ND_LeeModulo_DI (const int modulo, U16 * valor);

int ObtenerError (void);

char * ObtenerErrorString (void);
```

```
int Num_Lineas (int modulo);

int Num_Modulos (void);

int Get_Tipo (int modulo);

char * Get_Nombre (int modulo);

int Get_Id (int modulo);

int Get_Direccion (int modulo);

#endif
```

driverada_nd.ads

```
-- Autor: Adolfo Antonio Fernández Trabanco
-- Junio 2005
--
-- Archivo: driverada_nd.ads
-- Este archivo contiene la especificación en lenguaje Ada de las funciones
-- que forman el driver para los módulos NuDAM de la empresa AdLink
--
-- Módulos involucrados: ND-6053 y ND-6058

package DriverAda_ND is

--Definición de tipos
subtype Num_Linea_Entrada is Integer range 0 .. 15;
subtype Num_Linea_Salida is Integer range 0 .. 7;

subtype U1 is Integer range 0 .. 1;
subtype U8 is Integer range 0 .. (2**8)-1;
subtype U16 is Integer range 0 .. (2**16)-1;

subtype Num_Modulo is Integer range 0 .. 255;

type Puerto_Serie is (COM1, COM2);
for Puerto_Serie'Size use Integer'Size;
for Puerto_Serie use (COM1 => 1,
                      COM2 => 2);

type Puerto_Tarjeta is (PUERTO_A, PUERTO_B, PUERTO_C);
for Puerto_Tarjeta'Size use Integer'Size;
for Puerto_Tarjeta use (PUERTO_A => 0,
                        PUERTO_B => 1,
                        PUERTO_C => 2);

type Device_Type is (DIGITAL_IN, DIGITAL_OUT, ERROR);
for Device_Type'Size use Integer'Size;
for Device_Type use (ERROR => -1,
                     DIGITAL_IN  => 1,
                     DIGITAL_OUT => 2);
```

```
--Variables de error
type Var_Error is (CORRECTO,
--error en la comunicación con el puerto serie
                   ERROR_COM,
                   TIMEOUT,     --timeout al leer del puerto
                   FD_INVALIDO, --descriptor del puerto inválido
                   RESPUESTA_ERRONEA, --respuesta errónea del dispositivo
                   ERROR_MEM,   --error al asignar memoria dinámica
--el numero de modulo al que se quiere acceder no existe en el bus
                   NO_MODULO,
--el módulo no es un módulo de entradas digitales
                   NO_DINPUT,
--el módulo no es un módulo de salidas digitales
                   NO_DOUTPUT,
--no se ha encontrado ningún módulo en el bus
                   BUS_VACIO);

for Var_Error'Size use Integer'Size;


function ND_Inicializar (Puerto: Puerto_Serie; Velocidad: Integer;
                         Limite: Num_Modulo) return Integer;

function ND_Liberar return Integer;

function ND_EscribeLinea_DO (Modulo: Num_Modulo; Puerto: Puerto_Tarjeta;
                             Linea: Num_Linea_Salida; Valor: U1)
return Integer;

function ND_EscribePuerto_DO (Modulo: Num_Modulo; Puerto: Puerto_Tarjeta;
                              Valor: U8) return Integer;

function ND_EscribeModulo_DO (Modulo: Num_Modulo; ValorA: U8; ValorB: U8;
                              ValorC: U8) return Integer;

function ND_LeeLinea_DI (Modulo: Num_Modulo; Linea: Num_Linea_Entrada)
return Integer;

procedure ND_LeeModulo_DI (Modulo: in Num_Modulo; Valor: in out U16;
                           Retorno: out Integer);
```

```
function ObtenerError return Var_Error;

function ObtenerErrorString return string;

function Num_Lineas (Modulo: Num_Modulo) return Integer;

function Num_Modulos return Integer;

function Get_Tipo (Modulo: Num_Modulo) return Device_Type;

function Get_Nombre (Modulo: Num_Modulo) return String;

function Get_Id (Modulo: Num_Modulo) return Integer;

function Get_Direccion (Modulo: Num_Modulo) return Integer;

end DriverAda_ND;
```

4.- LabJack U12, manual de usuario

4.1.- Instalación

En el CD que acompaña a la documentación, se encuentra el driver del fabricante y los ficheros fuente que forman las interfaces de programación, así como las aplicaciones de prueba.

4.1.1.- Instalación del driver del fabricante

Se puede ver en el documento "*INSTALL*", que acompaña a las fuentes del driver del fabricante, el documento original que explica como llevar a cabo la instalación. En las siguientes líneas se mostrará un resumen de los pasos a seguir.

En su página Web el fabricante recomienda compilar un núcleo desde cero y no quedarse con el que proporcione la distribución de nuestro sistema operativo Linux. Además, en el documento *"INSTALL",* se advierte de que el driver puede no funcionar correctamente si el módulo "hid" está cargado en el sistema. Para evitar este problema proporcionan un par de soluciones:

- Primera: descargar el módulo hid cada vez que se vayan a usar las tarjetas LabJack ejecutando `modprobe –r hid` .
- Segunda: modificar un fichero fuente del núcleo de forma que las tarjetas de LabJack queden registradas, con el inconveniente de que sería necesario recompilar el núcleo para que los cambios tuvieran efecto.

La opción elegida es la segunda, pues, ya que el fabricante recomienda instalar un nuevo núcleo desde cero, se aprovechará el proceso para realizar las modificaciones necesarias que permitan un funcionamiento correcto de las tarjetas LabJack U12.

Las modificaciones a realizar son las siguientes:

- Se debe modificar el fichero `hid-core.c` ubicado en el directorio *drivers/usb/input* (ruta relativa al directorio donde se ubiquen las fuentes del núcleo).

Insertaremos en la estructura `hid_blacklist` la siguiente línea: {0xcd5, 0x0001, HID_QUIRK_IGNORE}
(LabJack USB vendor, 0x0cd5, and product, 0x0001)

- Se debe recompilar el núcleo para que los cambios tengan efecto.

Una vez que tenemos el nuevo núcleo instalado podemos proceder a compilar e instalar el driver de la tarjeta U12 de LabJack. El proceso descrito a continuación es para la versión del núcleo 2.6 y las rutas que aparecen son relativas al directorio donde se ubiquen las fuentes del driver:

- **Creación del driver**

  Situarse en el directorio *driver/linux-2.6* y ejecutar como usuario root el comando `make`. Se creará el módulo *labjack.ko.*

  Para cargar el módulo ejecutaremos como usuario root el comando:
  `# insmod labjack.ko`

  Se mostrará un mensaje por la consola. Si se está trabajando en una consola en modo X no se mostrará ningún mensaje. Para verlo, ejecutar el comando `dmesg` .

  Podemos comprobar que el módulo ha sido cargando correctamente consultando la lista de módulos cargados en el sistema. Para ello, ejecutaremos como usuario root el comando `lsmod` .

  Deberemos crear el fichero */dev/usb/labjack0* . Es posible que se cree automáticamente, pero de todas formas se recomienda crearlo a mano, sobre todo si disponemos de dos dispositivos LabJack.

  Para ello, ejecutamos como usuario root los siguientes comandos:

  `# mkdir /dev/usb` (si ya existe esta carpeta no sería necesario hacer esto)

  ```
  # mknod --mode=a=rw /dev/usb/labjack0 c 180 240
  ```

  Si disponemos de dos dispositivos LabJack, deberemos crear una entrada para cada uno de ellos. Para el segundo dispositivo, el comando sería:

  ```
  # mknod --mode=a=rw /dev/usb/labjack1 c 180 241
  ```

  (NOTA: sólo es necesario realizar este proceso una vez)

- **Creación de la librería**

  Situarse en el directorio *liblabjack* y ejecutar como usuario root el comando `make`

  Se creará el fichero *liblabjack.so* que deberemos copiar en el directorio */usr/lib* .

4.1.2.- Instalación de la interfaz

Para utilizar la librería, no es necesario realizar ningún proceso de instalación adicional. Es suficiente con copiar en el directorio de trabajo los ficheros adecuados:

- Si se va a utilizar la versión en Ada, se deben copiar los ficheros *ljackul.h*, *DriverC_LJ.h*, *DriverC_LJ.c*, *driverada_lj.ads* y *driverada_lj.adb*. También se deben copiar las librerías *liblabjack.so* y *libm.so* ubicadas en */usr/lib*.
- Si se va a utilizar la versión en C, se deben copiar los ficheros *ljackul.h*, *DriverC_LJ.h* y *DriverC_LJ.c*.

En el epígrafe 4.2.1, se explica el proceso a seguir para que el usuario cree sus propios programas de control, así como las instrucciones de compilación necesarias para crear el ejecutable.

Para utilizar las aplicaciones de prueba proporcionadas, se debe copiar la carpeta adecuada del CD al directorio de trabajo del usuario. Una vez copiada la carpeta, ejecutar el comando `make` para crear los ejecutables.

4.2.- Utilización

4.2.1.- Creación de un programa

Los pasos a seguir para obtener el ejecutable del programa realizado varían según el lenguaje de programación utilizado. Para simplificar el proceso de compilación de los programas, se recomienda al usuario que confeccione su propio fichero *Makefile*. Junto a las fuentes, se proporcionan ficheros *Makefile* que pueden servir de ejemplo.

Para crear un programa en lenguaje C que utilice las funciones de la librería, es necesario seguir los pasos indicados a continuación:

- Incluir en el código del programa la cabecera "*DriverC_LJ.h*".

```
#include <DriverC_LJ.h>
```

- Realizar el proceso de compilación para generar el fichero ejecutable. Se deben realizar los siguientes pasos:
  - Compilar los ficheros fuente que componen el driver:

    ```
    g++ -I. -c DriverC_LJ.c
    ```

  - Compilar el programa del usuario (supongamos que el fichero se llama ejemplo.cc):

    ```
    g++ -I. -c ejemplo.cc
    ```

  - Enlazar todos los ficheros objeto y librerías para obtener el ejecutable final:

    ```
    g++ ejemplo.o DriverC_LJ.o -o ejemplo -I. -llabjack -lm
    ```

Para crear un programa en lenguaje Ada que utilice las funciones de la librería, es necesario seguir los siguientes pasos:

- Incluir en el código del programa el paquete "*DriverAda_LJ*".

```
with DriverAda_LJ; use DriverAda_LJ;
```

- Realizar el proceso de compilación para generar el fichero ejecutable. Se deben realizar los siguientes pasos:
  - Compilar los ficheros fuente que componen el driver:

    ```
    gcc -c -I. DriverC_LJ.c
    ```

  - Compilar el programa del usuario (supongamos que el fichero se llama ejemplo.adb):

    ```
    gnatmake -c ejemplo.adb
    ```

  - Enlazar todos los ficheros objeto para obtener el ejecutable final:

    ```
    gnatbind -x ejemplo
    gnatlink ejemplo DriverC_LJ.o liblabjack.so libm.so
    ```

4.2.2.- Problemas conocidos

Durante la fase de desarrollo se produjeron ciertos inconvenientes que se documentan a continuación.

La primera vez que se enchufa la tarjeta LabJack U12 al puerto USB del PC, ésta no es reconocida por el sistema operativo. Sin embargo, si la tarjeta se desenchufa y se vuelve a enchufar es reconocida perfectamente. No se ha podido averiguar si se trata de un problema del driver proporcionado por el fabricante o del propio sistema operativo.

4.2.3.- Aplicaciones de prueba

Junto con las fuentes, se proporcionan un conjunto de programas genéricos que permiten al usuario interaccionar con el proceso e iniciarse en el uso de las distintas funciones. Son programas en modo consola, carentes de interfaz gráfica.

4.2.3.1.- list_all

Esta aplicación es proporcionada por el fabricante y utiliza las funciones propias de su librería. Realiza una detección de los módulos conectados al bus USB del PC y muestra por pantalla el número de módulos encontrados e información de cada uno de ellos. Los datos a los que el usuario debe prestar especial atención son el número de serie y el Local ID que se asigna a cada módulo encontrado. Estos datos se utilizan en las funciones para identificar el módulo sobre el que se desea actuar. En el Ejemplo 4.1 se puede ver una ejecución correcta del programa.

```
[txolfo@localhost c]$ ./list_all

Found 2 Labjacks!

Info:

productID, serialNum, localID, powerList, calMatrix

1, 100021047, 0, 9999,
 (6 14 10 15 4 11 16 6 6 11 9 16 5 12 16 3 -2 -2 -2 7 )

1, 100020586, 1, 9999,
 (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 )

[txolfo@localhost c]$
```

Ejemplo 4.1 Ejecución de *list_all*

4.2.3.2.- digital_in

Esta aplicación muestra por pantalla el estado de los canales digitales de entrada de la tarjeta RB16 conectada a la tarjeta LabJack U12 que le indiquemos.

Para ejecutar la aplicación, es necesario indicarle en la orden de ejecución el identificador del modulo sobre el que deseamos ejecutar la prueba. Este identificador puede ser el número de serie del módulo o el Local ID asignado durante la inicialización. Si todo ha sido correcto, se muestra por pantalla el estado de las entradas (Ejemplo 4.2).

```
[txolfo@localhost ada]$ ./digital_in 100021047

LJ_Inicializar:
 0  0

LJ_LeeModulo_DI:
 16  0
Estado de las entradas en decimal: 21268
Estado de las entradas en hexadecimal: 16#5314#

LJ_LeeLinea_DI:
 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
  0  1  0  1  0  0 1 1 0 0 0 1 0 1 0 0

Volver a mostrar las entradas?? (s/n) n

[txolfo@localhost ada]
```

Ejemplo 4.2 Ejecución completa de *digital_in*

4.2.3.3.- digital_out

Esta aplicación permite interaccionar con el proceso mediante la activación de los distintos canales de salida de la tarjeta RB16 conectada al módulo LabJack U12 que le indiquemos.

Para ejecutar la aplicación es necesario indicarle en la orden de ejecución el identificador del modulo sobre el que deseamos ejecutar la prueba. Este identificador puede ser el número de serie del módulo o el Local ID asignado durante la inicialización.

Al ejecutar la aplicación, se solicita al usuario el canal de salida sobre el que actuar y el valor de éste. Si en este paso se indica un número de línea erróneo pueden suceder dos cosas, dependiendo de la versión utilizada:

- En la versión en C, se producirá un error y se indicará esta circunstancia por pantalla (Ejemplo 4.3).
- En la versión en Ada, se producirá una excepción en tiempo de ejecución.

```
[txolfo@localhost c]$ ./digital_out 100020586

LJ_Inicializar
0 0

LJ_EscribeLinea_DI
Interacción con el modulo de salidas...

Identificador del módulo: 1
Linea: 0
Estado (1->Activar, 0->Desactivar) 1
Continuar?? (s/n) s

Identificador del módulo: 1
Linea: 16
Estado (1->Activar, 0->Desactivar) 0
-1 40 Illegal input
Continuar?? (s/n) n

LJ_EscribeModulo_DO
16 0

[txolfo@localhost c]$
```

Ejemplo 4.3 Ejecución de *digital_out* (versión en C)

4.3.- Descripción de la librería

En este epígrafe se pretende dar la información necesaria al usuario para que pueda comenzar a realizar sus propios programas.

Existen dos interfaces distintas (una en lenguaje C y otra en lenguaje Ada) de manera que el usuario es libre de elegir para implementar su programa de control la que más se adapte a sus necesidades. Dado que ambas versiones difieren ligeramente (en cuanto a tipos definidos y variables globales utilizadas) se invita al usuario a leer con detenimiento las particularidades de cada una de ellas.

4.3.1.- Interfaz en C

4.3.1.1.- Tipos definidos en C

En el archivo de cabecera "*DriverC_LJ.h*" (ver epígrafe 4.4) se definen los tipos de datos que son utilizados por la librería. Se recomienda al usuario utilizarlos en la implementación de sus programas. Los tipos definidos son:

- **U16** Precisión: 16 bits. Rango: 0-65535

Este tipo se utiliza en las funciones *LJ_EscribeModulo_DO* y *LJ_LeeModulo_DI*

4.3.1.2.- Variables globales en C

En el archivo de cabecera "*DriverC_LJ.h*" (ver epígrafe 4.4) se definen una serie de variables globales utilizadas por la librería y de utilidad para el usuario:

INPUT y **OUTPUT**: Indican la configuración de la tarjeta RB16, como entradas y salidas respectivamente. Se deben utilizar para realizar la llamada a la función *LJ_Inicializar* (p.ej. `LJ_Inicializar (100020586, INPUT)`)

4.3.1.3.- Variables de error en C

Si la ejecución de una función da lugar a un error, se devuelve un valor negativo y se actualizan las variables internas de error. Para obtener los valores de estas variables, se debe invocar a las funciones *ObtenerError* y *ObtenerErrorString*. El driver proporcionado por el fabricante define sus propios códigos de error. En el epígrafe 4.5 se pueden consultar todos los códigos de error que puede devolver la función *ObtenerError*.

4.3.2.- Interfaz en Ada

4.3.2.1.- Tipos definidos en Ada

En el archivo de cabecera "*driverada_lj.ads*" (ver epígrafe 4.4) se definen los tipos de datos que son utilizados por la librería. Se recomienda al usuario utilizarlos en la implementación de sus programas. Vamos a diferenciar entre los tipos básicos y los tipos enumerados.

En aquellas funciones donde se utilicen los tipos básicos como parámetros, se debe respetar el rango de estos, pues, en caso contrario, se lanzará una excepción en tiempo de ejecución. Los tipos básicos definidos son:

- **U1** Rango: 0-1
- **U16** Rango: 0-65535
- **Num_Linea** Rango: 0-15
- **Num_Linea_Salida_Analog** Rango: 0-1
- **Valor_Salida_Analog** Rango: 0.0-5.0

Mediante el uso de tipos enumerados se declaran las variables necesarias para definir la configuración de la tarjeta RB16:

- **Device_Type** (valores INPUT y OUTPUT): indican la configuración de la tarjeta RB16, como entradas y salidas respectivamente. Se deben utilizar para realizar la llamada a la función *LJ_Inicializar* (por ejemplo, `LJ_Inicializar (100020586, INPUT)`)

4.3.2.2.- Variables de error en Ada

Si la ejecución de una función da lugar a un error, se devuelve un valor negativo y se actualizan las variables internas de error. Para obtener los valores de estas variables, se debe invocar a las funciones *ObtenerError* y *ObtenerErrorString*. El driver proporcionado por el fabricante define sus propios códigos de error. En el epígrafe 4.5 se pueden consultar todos los códigos de error que puede devolver la función *ObtenerError*.

4.3.3.- Descripción de las funciones

Se puede ver que varias funciones necesitan el argumento *modulo*. Este argumento hace referencia al número de serie del módulo o al Local ID. El número de serie, se puede consultar en la pegatina situada en la parte posterior de las tarjetas.

Si la ejecución de una función da lugar a un error, se devuelve un valor negativo. Se recomienda utilizar las funciones *ObtenerError* y *ObtenerErrorString* para obtener el código de error y una descripción de este respectivamente.

4.3.3.1.- LJ_Inicializar

◇ **Descripción**

Esta función sirve para inicializar el módulo y configurarlo para manejar las entradas o salidas del proceso. Se debe invocar esta función antes de llamar a cualquier otra.

◇ **Sintaxis**

C/C++

```
int LJ_Inicializar (int modulo,
                    int tipo);
```

Ada

```
function LJ_Inicializar (Modulo: Integer;
                         Tipo: Device_Type)
return Integer;
```

◇ **Argumentos**

Modulo: Número de serie o Local ID.

Tipo: Configuración que se desea otorgar a la tarjeta RB16 conectada al módulo, como entradas o salidas. Utilizar las variables *INPUT* y *OUTPUT.*

◇ **Valor retornado**

0 si todo fue correcto.

-1 en caso de error durante la inicialización.

Sólo para la versión en C/C++: -2 si la variable *tipo* no se corresponde con ninguna de las macros definidas.

4.3.3.2.- LJ_EscribeLinea_DO

◇ **Descripción**

Esta función actualiza el valor de un canal de salida de la tarjeta RB16 conectada al módulo LabJack U12 correspondiente.

◇ **Sintaxis**

C/C++

```
int LJ_EscribeLinea_DO (int modulo,
                        int linea,
                        int valor);
```

Ada

```
function LJ_EscribeLinea_DO (Modulo: Integer;
                             Linea: Num_Linea;
                             Valor: U1)
return Integer;
```

◇ **Argumentos**

Modulo: Número de serie del módulo o Local ID.

Linea: Número del canal de salida que se desea actualizar (entre 0 y 15).

Valor: Valor de la línea a actualizar (1: activar; 0: desactivar).

◇ **Valor retornado**

0 si todo fue correcto.

-1 en caso de error.

4.3.3.3.- LJ_EscribeModulo_DO

◇ **Descripción**

Esta función actualiza los dieciséis canales de salida de la tarjeta RB16 conectada al módulo LabJack U12 correspondiente.

◇ **Sintaxis**

C/C++

```
int LJ_EscribeModulo_DO (int modulo,
                         U16 valor);
```

Ada

```
function LJ_EscribeModulo_DO (Modulo: Integer;
                              Valor: U16)
return Integer;
```

◇ **Argumentos**

Modulo: Número de serie o Local ID del módulo.

Valor: Entero donde estén codificados los valores de los dieciséis canales de la tarjeta RB16.

◇ **Valor retornado**

El número de canales de la tarjeta RB16 si todo fue correcto.

-1 en caso de error.

4.3.3.4.- LJ_LeeLinea_DI

◇ **Descripción**

Esta función lee el estado de un canal de entrada de la tarjeta RB16 conectada al módulo LabJack U12 correspondiente.

◇ **Sintaxis**

C/C++

```
int LJ_LeeLinea_DI (int modulo,
                    int linea);
```

Ada

```
function LJ_LeeLinea_DI (Modulo: Integer;
                         Linea: Num_Linea)
return Integer;
```

◇ **Argumentos**

Modulo: Número de serie o Local ID del módulo.

Linea: Número del canal de entrada que se desea leer (entre 0 y 15).

◇ **Valor retornado**

El estado del canal de entrada (1: activo; 0: inactivo) si todo fue correcto.

Un número negativo si hubo algún error.

4.3.3.5.- LJ_LeeModulo_DI

◇ **Descripción**

Esta función lee el estado de los dieciséis canales de entrada de la tarjeta RB16 conectada al módulo LabJack U12 correspondiente. El estado de las entradas se devuelve codificado en un entero que se ha de pasar a la función como argumento.

◇ **Sintaxis**

C/C++

```
int LJ_LeeModulo_DI (int modulo,
                     U16 * valor);
```

Ada

```
procedure LJ_LeeModulo_DI (Modulo: in Integer;
                           Valor: in out U16;
                           Retorno: out Integer);
```

◇ **Argumentos**

Modulo: Número de serie del módulo o Local ID.

Valor: Entero donde se codificará el estado de los dieciséis canales de la tarjeta RB16.

◇ **Valor retornado**

0 si todo fue correcto.

-1 en caso de error.

La implementación de esta función en Ada se hace a través de un procedimiento. El valor de retorno se devuelve a través de la variable *Retorno.*

4.3.3.6.- LJ_EscribeLinea_AO

◇ **Descripción**

Esta función actualiza el valor de uno de los dos canales de salida analógicos de los que dispone el módulo LabJack U12 (AO0-AO1).

◇ **Sintaxis**

C/C++

```
int LJ_LeeModulo_DI (int modulo,
                     U16 * valor);
```

Ada

```
function LJ_EscribeLinea_AO (Modulo: Integer;
                             Linea: Num_Linea_Salida_Analog;
                             Valor: Valor_Salida_Analog)
return Integer;
```

◇ **Argumentos**

Modulo: Número de serie o Local ID del módulo.

Linea: Número del canal de salida analógico que se desea actualizar (0 ó 1).

Valor: Valor de la línea a actualizar (entre 0.0 y 5.0).

◇ **Valor retornado**

0 si todo fue correcto.

-1 en caso de error.

4.3.3.7.- ObtenerError

◇ **Descripción**

Esta función sirve para obtener el estado en que finalizó la última operación que se ha realizado. En el epígrafe 4.5 se pueden consultar todos los códigos de error que puede devolver la función *ObtenerError*.

◇ **Sintaxis**

C/C++

```
int ObtenerError (void);
```

Ada

```
function ObtenerError return Integer;
```

◇ **Valor retornado**

Un entero que identifica la situación en que finalizó la última operación.

4.3.3.8.- ObtenerErrorString

◇ **Descripción**

Esta función sirve para obtener una descripción del estado en que finalizó la última operación realizada. Si en la última llamada a una función se ha producido un error, mediante una llamada a esta función se obtendrá una descripción de éste.

◇ **Sintaxis**

C/C++

```
char * ObtenerErrorString (void);
```

Ada

```
function ObtenerErrorString return string;
```

◇ **Valor retornado**

Una cadena con la descripción del estado en que finalizó la última operación realizada. Si en la última operación no se produjo ningún error, se retornará una cadena vacía.

4.4.- Anexo A: Archivos de cabecera

DriverC_LJ.h

```
// Autor: Adolfo Antonio Fernández Trabanco
// Junio 2005
//
// Archivo: DriverC_LJ.h
// Este archivo contiene la especificación de las funciones que forman
// el binding con el driver de los módulos U12 de la empresa LabJack

#ifndef DriverC_LJ_H
#define DriverC_LJ_H

//Definición de tipos
//Para valores de 16 bits (rango de 0 a 65535)
typedef unsigned short U16;

#define INPUT   1
#define OUTPUT  2


int LJ_Inicializar (int modulo, int tipo);

int LJ_EscribeLinea_DO (int modulo, int linea, int valor);

int LJ_EscribeModulo_DO (int modulo, U16 valor);

int LJ_LeeLinea_DI (int modulo, int linea);

int LJ_LeeModulo_DI (int modulo, U16 * valor);

int LJ_EscribeLinea_AO (int modulo, int linea, float valor);

int ObtenerError (void);

char * ObtenerErrorString (void);

#endif
```

driverada_lj.ads

```
-- Autor: Adolfo Antonio Fernández Trabanco
-- Junio 2005
--
-- Archivo: driverada_lj.ads
-- Este archivo contiene la especificación de las funciones que forman
-- el binding con el driver de los módulos U12 de la empresa LabJack

package DriverAda_LJ is

--Definición de tipos
subtype Num_Linea is Integer range 0 .. 15;
subtype Num_Linea_Salida_Analog is Integer range 0..1;

subtype Valor_Salida_Analog is Float range 0.0 .. 5.0;

subtype U1 is Integer range 0 .. 1;
subtype U16 is Integer range 0 .. (2**16)-1;

type Device_Type is (INPUT, OUTPUT);
for Device_Type'Size use Integer'Size;
for Device_Type use (INPUT  => 1,
                     OUTPUT => 2);


function LJ_Inicializar (Modulo: Integer; Tipo: Device_Type)
return Integer;

function LJ_EscribeLinea_DO (Modulo: Integer; Linea: Num_Linea; Valor: U1)
return Integer;

function LJ_EscribeModulo_DO (Modulo: Integer; Valor: U16) return Integer;

function LJ_LeeLinea_DI (Modulo: Integer; Linea: Num_Linea) return Integer;

procedure LJ_LeeModulo_DI (Modulo: in Integer; Valor: in out U16;
                           Retorno: out Integer);
```

```
function LJ_EscribeLinea_AO (Modulo: Integer;
                             Linea: Num_Linea_Salida_Analog;
                             Valor: Valor_Salida_Analog) return Integer;

function ObtenerError return Integer;

function ObtenerErrorString return string;

end DriverAda_LJ;
```

4.5.- Anexo B: Description of errorcodes

En este anexo se proporciona la descripción de los errores que puede devolver la tarjeta LabJack U12. Esta información se ha obtenido directamente del epígrafe *4.40 "Description of errorcodes"* del manual de usuario "*LabJack U12 Users Guide*".

0 – No error.
1 – Unknown error.
2 – No LabJacks found.
3 – LabJack n not found.
4 – Set USB buffer error.
5 – Open handle error.
6 – Close handle error.
7 – Invalid ID.
8 – Invalid array size or value.
9 – Invalid power index.
10 – FCDD size too big.
11 – HVC size too big.
12 – Read error.
13 – Read timeout error.
14 – Write error.
15 – Turbo error.
16 – Illegal channel index.
17 – Illegal gain index.
18 – Illegal AI command.
19 – Illegal AO command.
20 – Bits out of range.
21 – Illegal number of channels.
22 – Illegal scan rate.
23 – Illegal number of samples.
24 – AI response error.
25 – LabJack RAM checksum error.
26 – AI sequence error.
27 – Maximum number of streams.
28 – AI stream start error.
29 – PC buffer overflow.
30 – LabJack buffer overflow.
31 – Stream read timeout.
32 – Illegal number of scans.
33 – No stream was found.
40 – Illegal input.
41 – Echo error.
42 – Data echo error.
43 – Response error.
44 – Asynch read timeout error.
45 – Asynch read start bit error.
46 – Asynch read framing error.
47 – Asynch DIO config error.
48 – Caps error.
49 – Caps error.
50 – Caps error.
51 – HID number caps error.
52 – HID get attributes warning.
57 – Wrong firmware version error.
58 – DIO config error.
64 – Could not claim all LabJacks.
65 – Error releasing all LabJacks.
66 – Could not claim LabJack.
67 – Error releasing LabJack.
68 – Claimed abandoned LabJack.
69 – Local ID –1 thread stopped.
70 – Stop thread timeout.

71 – Thread termination failed.

72 – Feature handle creation error.

73 – Create mutex error.

80 – Synchronous CS state or direction error.

81 – Synchronous SCK direction error.

82 – Synchronous MISO direction error.

83 – Synchronous MOSI direction error.

89 – SHT1X CRC error.

90 – SHT1X measurement ready error.

91 – SHT1X ack error.

92 – SHT1X serial reset error.