Sensoray Instruction Manual for 421.DLL

July 22, 1999

1. Introduction

1.1 Scope

This document describes the installation and use of the Sensoray Windows 95/NT Dynamic Link Library (DLL) for Sensoray Model 421 multi-function I/O boards. Refer to the Model 421 Instruction Manual for information concerning board capabilities and functional behavior.

1.2 Description

The Model 421 DLL enables you to interface up to sixteen Sensoray Model 421 boards to a Windows application program of your own design. Application programs may be developed using most any popular Windows development tool, including Visual C++, Visual Basic, Delphi, Watcom, etc., so long as the tool is able to statically or dynamically link to a standard DLL.

2. Installation

2.1 Software Components

To ensure proper functioning of the DLL, the following software components must be installed on the target system in the specified directories:

`421.DLL`	Windows SYSTEM directory.	
`WINDRVR.SYS`	Windows SYSTEM32\DRIVERS directory.	Required by WinNT only.
`WINDRVR.VXD`	Windows SYSTEM\VMM32 directory.	Required by Win95/98 only.

2.2 Installation Procedure

1. Install all required software components as described in Section 2.1.
2. Execute this command line (excluding quotes): "`WDREG.EXE INSTALL`"
3. Shut down Windows and remove system power.
4. Install the Model 421 board(s) into the backplane.
5. Apply power and restart your system.

3. Usage Fundamentals

3.1 Board Handles

Each board is assigned a handle, which may simply be regarded as a "board number," with a numerical value in the range 0 to 15. All DLL functions include the board handle as a parameter so that DLL function calls can be directed to a specific board. Throughout this document, the parameter `hBD` is used to designate a board handle.

Board handles, which may range in value from 0 to 15, are assigned by the application program.

3.2 Required Function Calls

Some DLL functions are used universally in all applications, while other DLL functions may or may not be used depending on the board resources in use and other considerations. All application programs must, as a minimum, perform the following sequence of steps for each Model 421 board:

1. Call `SetBaseAdrs()` to declare the board and its base address.
2. Call `FaultFlagsGet()` to verify that the board is properly initialized, fault-free and registered with the DLL.
3. Call `BoardReset()` to initialize the board to a known state.

4. DLL Functions

4.1 Board Configuration and Miscellaneous Functions

4.1.1 SetBaseAdrs()

Function Prototype: `void SetBaseAdrs( long hBD, long baseadrs )`

Registers a Model 421 board with the DLL and declares the base I/O address of the board. Because this function opens a communication connection to a board, this must be the first function called for each Model 421 board in your system.

You may specify a board number (hBD) value in the range 0 through 15. To prevent conflicts, do not use a board number that has already been used for another Model 421 board. In general, it is good practice to use board number 0 for the first board, number 1 for the second board, and so on.

Example:

```
// Declare board number 0, which resides at base I/O address 03A0 hex, to the DLL.
SetBaseAdrs( 0, 0x03A0 );
```

4.1.2 FaultFlagsGet()

Function Prototype: `long FaultFlagsGet( long hBD );`

Returns a set of bit flags that indicate specific fault conditions. Listed below are the mask values for each fault bit and a description of the fault that is indicated when the flag is asserted true.

`0x00000008` **Driver failed to open.**
This is usually caused by a missing or unregistered software component. Make sure all required software components are properly installed and registered (see *Installation* instructions above).

`0x00000010` **Failed to register the board when `SetBaseAdrs()` was called.**
This is usually caused by incorrect specification of the base I/O address or a hardware conflict with some other device in the system.

`FaultFlagsGet()` may be called at any time after `SetBaseAdrs()`.

Example:

```
// Fetch the fault flags from board number 0.
long FaultFlags0 = FaultFlagsGet( 0 );
```

4.1.3 StatusGet()

Function Prototype: `long StatusGet( long hBD );`

Returns the contents of a board's status register. Refer to the Model 421 Instruction Manual for detailed descriptions of the status register bits.

Example:

```
// Fetch a copy of the status register on board number 2.
long StatusImage0 = StatusGet( 2 );
```

4.1.4 BoardReset()

Function Prototype: `void BoardReset( long hBD );`

This function invokes a soft reset on the specified board, forcing the board to its default power-up condition. The board Fault indicator will momentarily light when this function is called.

Example:

```
// Reset board number 5.
BoardReset( 5 );
```

4.2 Counter Functions

Some encoder functions specify a channel number parameter, ChNum, which designates the counter channel to be affected by the function. The Model 421 has three counter channels. Counter channel numbers range in value from 0 to 2.

4.2.1 CounterModeSet()

Function Prototype: `void CounterModeSet( long hBD, long Mode );`

Sets the operating mode for all three counter channels. Refer to the Model 421 Instruction Manual for descriptions of the counter operating modes.

Example:

```
// Program all counter channels on board number 2 to operate in mode 5.
CounterModeSet( 2, 5 );
```

4.2.2 CounterGet()

Function Prototype: `long CounterGet( long hBD, long ChNum );`

Returns the contents of the specified counter channel. The returned data value falls between 0 and 65535, inclusive.

Example:

```
// Fetch the current contents of counter channel 2 on board number 0.
long CounterImage2 = CounterGet( 0, 2 );
```

4.2.3 CounterReset()

Function Prototype: `void CounterReset( long hBD, long ChNum );`

Forces the contents of the specified counter channel to zero.

Example:

```
// Reset all three counter channels on board number 0.
for ( char i = 0; i < 3; CounterReset( 0, i++ ) );
```

4.3 Digital I/O Functions

Digital I/O, which is synonomous with “Relay I/O” in this document, may be addressed either as individual channels (single I/O bits) or as groups of eight channels. Maximum performance is attained by addressing groups of channels, as this is the native addressing mode used by Model 421 hardware. If performance optimization is not an issue, you may opt to address channels individually, especially if your program logic requires manipulation of discrete relay channels. This will greatly reduce the efforts you must make to perform bit masking and state image maintenance. You may use both types of addressing in your application as long as you properly handle interactions between the two access styles.

Model 421 has a total of 48 relay channels, corresponding to channel numbers 0 through 47. There are a total of six eight-channel groups, corresponding to group numbers 0 through 5. The relationship between groups and channels are shown in the following table:

Group	Channels	Channel-to-Bit Mapping							
		7	6	5	4	3	2	1	0
0	0 to 7	7	6	5	4	3	2	1	0
1	8 to 15	15	14	13	12	11	10	9	8
2	16 to 23	23	22	21	20	19	18	17	16
3	24 to 31	31	30	29	28	27	26	25	24
4	32 to 39	39	38	37	36	35	34	33	32
5	40 to 47	47	46	45	44	43	42	41	40

For example, take the case of channel group 2. Any accesses to this group will be in the form of a byte whose most significant bit corresponds to channel 23 and least significant bit corresponds to channel 16.

4.3.1 RelayChanSet()

Function Prototype: `void RelayChanSet( long hBD, long ChNum, long Value );`

Modifies the output state of a single relay channel. The `ChNum` parameter specifies which relay channel is to be modified. The `Value` parameter specifies the desired output state; a zero value causes the output to go to the inactive state, while any non-zero value causes the output to go to the active state.

Example:

```
#define RELAY_DEACTIVATE  0
#define RELAY_ACTIVATE    ~RELAY_DEACTIVATE
// Set relay channel 4 on board 0 to the active state.
RelayChanSet( 0, 4, RELAY_ACTIVATE );
```

4.3.2 RelayChanGet()

Function Prototype: `long RelayChanGet( long hBD, long ChNum );`

Returns the input state of a single relay channel. The returned value represents the physical state present at the channel circuit, regardless of whether the signal is driven by the channel output driver or an external signal source. The returned value will be zero if the channel is inactive, or non-zero if the channel is driven active.

Example:

```
#define RELAY_INACTIVE  0
#define RELAY_ACTIVE    !RELAY_INACTIVE
// Fetch the input state of relay channel 4 on board number 0.
boolean IsRelayActive4 = ( RelayChanGet( 0, 4 ) == RELAY_ACTIVE );
```

4.3.3 RelayChanOutputGet()

Function Prototype: `long RelayChanOutputGet( long hBD, long ChNum );`

Returns the output state of a single relay channel. The returned value is the last value that was written to the channel as a result of calling `RelayChanSet()`, `RelayGroupSet()` or `BoardReset()`. Note that the returned value may not be the same value that would be returned by `RelayChanGet()` if the addressed channel has a wired-or connection.

Example:

```
#define RELAY_INACTIVE  0
#define RELAY_ACTIVE    !RELAY_INACTIVE
// Fetch the output state of relay channel 4 on board number 0.
boolean IsRelayOutputActive4 = ( RelayChanOutputGet( 0, 4 ) == RELAY_ACTIVE );
```

4.3.4 RelayGroupSet()

Function Prototype: `void RelayGroupSet( long hBD, long Group, long Value );`

Modifies the output states of all eight channels in the specified relay group. The `Group` parameter specifies which relay group is to be modified. The `Value` parameter specifies the desired output states; a zero in a relay's bit position causes the output to go to the inactive state, while a one causes the output to go to the active state.

Example:

```
// Set relay channels 7-4 active and 3-0 inactive (group 0) on board 0.
RelayGroupSet( 0, 0, 0xF0 );
```

4.3.5 RelayGroupGet()

Function Prototype: `long RelayGroupGet( long hBD, long Group );`

Returns the input states of all eight channels in the specified relay group. The returned value represents the physical states present at the channel circuits, regardless of whether the signals are driven by the output drivers or external signal sources. The returned bit values will be zero if the corresponding channel input state is inactive, or one if the state is active.

Example:

```
// Fetch the input states of relay channels 7-0 (group 0) on board number 0.
long RelayGroup0 = RelayGroupGet( 0, 0 );
```

4.3.6 RelayGroupOutputGet()

Function Prototype: `long RelayGroupOutputGet( long hBD, long Group );`

Returns the output states of all eight channels in the specified relay group. The returned value is the last value that was written to the channels in the group as a result of calling `RelayChanSet()`, `RelayGroupSet()` or `BoardReset()`. Note that the returned value may not be the same value that would be returned by `RelayGroupGet()` if any channel in the addressed group has a wired-or connection.

Example:

```
// Fetch the output states of relay channels 7-0 (group 0) on board number 0.
long RelayOutputGroup0 = RelayGroupOutputGet( 0, 0 );
```

4.4 Analog Input Functions

The Model 421 board has eight analog input channels. Analog input channel numbers range from 0 through 7.

4.4.1 DigitizerChanSet()

Function Prototype: `void DigitizerChanSet( long hBD, long ChNum );`

Selects the next analog output channel to be digitized. The analog input multiplexer on the Model 421 board is switched so that the specified channel number, in the range 0 through 7, is connected to the A/D converter.

Example:

```
// Select analog input channel 3, board number 0, for digitization.
DigitizerChanSet( 0, 3 );
```

4.4.2 Digitize()

Function Prototype: `long Digitize( long hBD );`

Digitizes an analog input channel. Before invoking this function, call `DigitizerChanSet()` to select the channel to be digitized and wait a sufficient time for the signal to settle.

Example:

```
// Digitize analog input channel 3 on board number 0.
DigitizerChanSet( 0, 3 );
DelayTenMicroseconds();    // You must create this function.
long DigitizedData = Digitize( 0 );
```

4.5 Analog Output Functions

The Model 421 has four analog output channels. Analog output channel numbers range from 0 through 3.

4.5.1 DacsEnableSet()

Function Prototype: `void DacsEnableSet( long hBD, long Enable );`

Enables or disables all four analog output channels. Set `Enable` to zero to disable outputs, or to any non-zero value to enable outputs.

Example:

```
// Enable all analog outputs on board number 0.
#define DACS_DISABLE  0
#define DACS_ENABLE  ~DACS_DISABLE
DacsEnableSet( 0, DACS_ENABLE );
```

4.5.2 DacOutputSet()

Function Prototype: `void DacOutputSet( long hBD, long ChNum, long Value );`

Modifies the output voltage of the specified analog output channel. The `ChNum` argument specifies the analog output channel to be modified. The `Value` argument specifies the binary value to be written to the analog output channel.

Example:

```
// Set analog output channel 2 of board 0 to 6.53 volts out.
#define VOLTS_SCALAR  409.5
DacOutputSet( 0, 2, 6.53 * VOLTS_SCALAR );
```

4.6 Watchdog Functions

4.6.1 WatchdogEnableSet()

Function Prototype: `void WatchdogEnableSet( long hBD, long Enable );`

Enables or disables the watchdog timer.

Example:

```
// Enable the watchdog timer on board number 2.
#define WATCHDOG_DISABLE   0
#define WATCHDOG_ENABLE    ~WATCHDOG_DISABLE
WatchdogEnableSet( 2, WATCHDOG_ENABLE );
```

4.7 Watchdog Functions

4.7.1 WatchdogRefresh()

Function Prototype: `void WatchdogRefresh( long hBD );`

Resets the watchdog timer.

Example:

```
// Reset the watchdog timer on board number 2.
WatchdogRefresh( 2 );
```