ADS Windows CE Digital I/O Driver

Specification Version 1.2

ADS Document 110025-10056

Introduction

The purpose of the Digital I/O (DIO) driver is to encapsulate all available digital I/O lines in a single stream driver interface. Each I/O line is represented as one bit in a 32-bit DIO register. This register can be read and written to using the standard **ReadFile** and **WriteFile** functions. I/O Control codes provide additional functionality such as setting the line direction (i.e. input or output) and masking individual lines.

Theory of Operation

This section describes how the ADS DIO driver interacts with system hardware. Details about the settings and APIs are listed in the following sections, ***Using the Driver*** and ***API Reference***.

Reading Line States

The current state of any DIO line can be read using the **ReadFile** function. **ReadFile** reads the states of all 32 DIO lines and associates each with a bit in the DIO register. The state of the register is stored in the 32-bit location (4 bytes) pointed to by *lpBuffer*. If a line is masked (see below), its state will always be read as zero.

Writing Line States

The current state of any DIO line can be set using the **WriteFile** function. **WriteFile** maps the 4 byte location pointed to by *lpBuffer* to the 32-bit DIO register and writes the bit values to their corresponding lines. If a line is masked, its state will be unaffected by a write.

Line Settings

DIO line direction settings are managed through I/O control functions. These functions determine which lines can be inputs, which can be outputs, and get and set the current direction of each line. The direction of a masked line is unaffected by these functions.

Line Masking

Individual DIO lines can be masked by setting them in the DIO mask. Masked lines will be unaffected by all operations in the DIO driver. The DIO mask can be accessed through the **DIO_IOCTL_GET_DRIVER_MASK** and **DIO_IOCTL_SET_DRIVER_MASK** I/O controls. At boot the driver mask is 0x00 (all lines unmasked).

Interrupts

Individual DIO lines may be enabled to generate interrupts. The availability of this feature depends on the platform. The **DIO_IOCTL_CAPABLE_INT** I/O control can be used to find out which DIO lines are interrupt capable. When a DIO line is enabled as an interrupt, it is associated with a trigger mode which determines what condition will cause the interrupt to occur. Also, each interrupt capable DIO line has a unique event, which can be configured in the registry. When a DIO interrupt is triggered, the driver pulses the corresponding event.

System Wake

Some interrupt lines on some systems can provide the ability to wake the system from sleep. Availability can be checked using the **DIO_IOCTL_CAPABLE_WAKEUP_INT** I/O control. DIO lines that use this functionality should be enabled with the appropriate wake-up trigger mode.

Using the Driver

Driver Name

The Digital I/O driver is referenced with the filename **`DIO1:`**.

Files

The following files are useful for developers of DIO applications:

DIO.dll	*Digital IO Driver Library*
DIOapp.h	*Header file for DIO driver constants*
ADSerror.h	*Header file for ADS error codes*

Supported Registry Keys

The following registry values can be used to modify the Digital I/O driver behavior. All are values under the `[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\DIO]` registry key. These settings can be modified by changing the ADSLOAD.REG file or by changing the registry and persisting it using either ADS tools or hive-based images.

`[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\DIO]`
`"DIOEvent[n]"="DIOInt[n]"`

Default Value: DIOInt[n]
Supported Values: n ranges from 0 to the maximum number of interrupt-supported lines
Description: There will be one key for each interrupt-capable bit in the driver, starting with DIOEvent0. These keys provide the ability to override the string used for the event name the driver will use for signaling when an interrupt has occurred. By default these values will be "DIOInt[n]"
For example, if a platform has a single interrupt-capable bit, it will have the following registry entry:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\DIO]
    "DIOEvent0"="DIOInt0"
```

When an interrupt occurs on the line, the driver will pulse of set an event (see UseSetEvent key documentation below for more information) created with the name "DIOInt0".

`[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\DIO]`
`"UseSetEvent"=dword:0`

Default Value: 0x00
Supported Values: 0x00 or 0x01
Description: Determines whether the driver uses SetEvent or PulseEvent to signal an interrupt has been detected (see Driver Events section below).

Driver Events

By default, when a DIO interrupt is triggered, the driver sets the event associated with that line with a call to `PulseEvent`. The driver can be configured to use `SetEvent` instead of `PulseEvent` through the registry key outlined in the previous section. For each interrupt capable line available on a system, there will be a registry key called DIOEventX, where X is the DIO bit position of the line it corresponds to. The value of each key is the name of the event that will be pulsed when that interrupt is triggered. These event names are DIOInt0, DIOInt1, etc. by default, but can be modified by changing the name in the registry.

API Reference

Windows CE applications access the ADS Digital I/O driver using the standard Windows stream interface functions (**`CreateFile`**, **`ReadFile`**, **`WriteFile`** and **`DeviceIoControl`**).

`HANDLE CreateFile( lpFileName, dwDesiredAccess, dwShareMode, lpSecurityAttributes, dwCreationDispostion, dwFlagsAndAttributes, hTemplateFile )`

Opens the DIO driver and returns a handle to access it with. Returns NULL if an error occurred. **`GetLastError`** can be used to retrieve a specific error code. ADS error codes are listed in the file ADSerror.h and described in the "Error Codes" section of this document.

Example:

```
HANDLE hDioPort;
hDioPort = CreateFile( _T("DIO1:"),
                       GENERIC_WRITE | GENERIC_READ,
                       0,
                       NULL,
                       OPEN_EXISTING,
                       FILE_ATTRIBUTE_NORMAL,
                       NULL);
```

`BOOL ReadFile( hFile, lpBuffer, nNumberOfBytesToRead, lpNumberOfBytesRead, lpOverlapped )`

Reads the state of the DIO register into the location pointed to by lpBuffer. nNumberOfBytesToRead must be DIO_SIZE. Returns TRUE if successful or FALSE if there is an error. If FALSE is returned, **GetLastError** can be used to retrieve the error code. ADS error codes are listed in the file ADSerror.h and described in the "Error Codes" section of this document.. Unsupported or masked bit states are considered undefined and the bit state read may or may not reflect the actual state.

Example:

```
DWORD dwState;
DWORD dwBytesRead;

if(!ReadFile(hDioPort, &dwState, DIO_SIZE, &dwBytesRead, NULL))
{
    RETAILMSG(TRUE, (_T("ReadFile : GetLastError() returned %i"), GetLastError()));
}
```

`BOOL WriteFile( hFile, lpBuffer, nNumberOfBytesToWrite, lpNumberOfBytesWritten, lpOverlapped )`

Writes the four bytes of data (DIO_SIZE) pointed to by lpBuffer to the DIO register. nNumberOfBytesToWrite must be DIO_SIZE. Returns TRUE if successful or FALSE if there is an error. If FALSE is returned, **GetLastError** can be used to retrieve the error code. ADS error codes are listed in the file ADSerror.h. Unsupported or masked bits are ignored.

Example:

```
DWORD dwState = 0x00FF;
DWORD dwBytesWritten;
...
if(!WriteFile(hDioPort, &dwState, DIO_SIZE, &dwBytesWritten, NULL))
{
    RETAILMSG(TRUE, (_T("WriteFile : GetLastError() returned %i"), GetLastError()));
}
```

`BOOL DeviceIoControl( hDevice, dwIoControlCode, lpInBuffer, nInBufferSize, lpOutBuffer, nOutBufferSize, lpBytesReturned, lpOverlapped )`

Provides an additional set of DIO functions. These functions are listed in the next section. Input parameters are passed to these functions through lpInBuffer, and outputs are returned in lpOutBuffer. Returns TRUE if successful or FALSE if there is an error. If FALSE is returned, **GetLastError** can be used to retrieve the error code. ADS error codes are listed in the file ADSerror.h.

Example:

```
DWORD DIODir = 0;
DWORD NumberOfBytesRead = 0;
DWORD NumberOfBytesWritten = 0;

// Get the current DIO direction.
if(!DeviceIoControl( hDIO, DIO_IOCTL_GET_DIRECTION, NULL, 0, &DIODir, DIO_SIZE, \
                     &NumberOfBytesRead, NULL ))
{
    RETAILMSG(TRUE, (_T("DeviceIoControl : GetLastError() returned %i"), \
                          GettLastError()));
    return FALSE;
}
```

`BOOL CloseHandle( hObject )`

Closes the DIO driver referenced by *`hObject`*. Returns TRUE if successful or FALSE if there is an error.

Example:

```
CloseHandle(hCanPort);
```

`DWORD Seek( hOpenContext, Amount, Type )`

Calls to the **Seek** function have no effect and always return **DIO_ERROR_NOT_IMPLEMENTED**.

I/O Controls

The I/O control codes listed below provide access to additional functionality in the ADS Digital I/O driver. Usage of **`DeviceIoControl`** is described in the previous section.

DIO_IOCTL_INPUT (0x00): Returns a four-byte bit mapping of the DIO register with bits corresponding to input lines set to 1 and all others set to 0. The bit mapping is returned in *lpOutBuffer*.

DIO_IOCTL_OUTPUT (0x01): Returns a four-byte bit mapping of the DIO register with bits corresponding to output lines set to 1 and all others set to 0. The bit mapping is returned in *lpOutBuffer*.

DIO_IOCTL_HIGH (0x02): Returns a four-byte bit mapping of the DIO register with bits corresponding to logic high lines set to 1 and all others set to 0. The bit mapping is returned *in lpOutBuffer*.

DIO_IOCTL_LOW (0x03): Returns a four-byte bit mapping of the DIO register with bits corresponding to logic low lines set to 1 and all others set to 0. The bit mapping is returned in *lpOutBuffer*.

DIO_IOCTL_CAPABLE_INPUTS (0x100): Returns a four-byte bit mapping of the DIO register with bits corresponding to lines capable of being inputs set to 1 and all other lines set to 0. The bit mapping is returned in *lpOutBuffer*.

DIO_IOCTL_CAPABLE_OUTPUTS (0x101): Returns a four-byte bit mapping of the DIO register with bits corresponding to lines capable of being outputs set to 1 and all others set to 0. The bit mapping is returned in *lpOutBuffer*.

DIO_IOCTL_GET_DRIVER_MASK (0x102): Returns a four-byte bit mapping of the DIO register with bits corresponding to lines that are masked set to 1 and all others set to 0. The bit mapping is returned in *lpOutBuffer*.

DIO_IOCTL_SET_DRIVER_MASK (0x103): Reads a DIO bit mapping provided in *lpInBuffer*. Lines that are set to 1 in the mapping are set as masked lines and lines set to 0 in the mapping are set as unmasked lines.

DIO_IOCTL_GET_DIRECTION (0x104): Returns a four-byte bit mapping of the DIO register with bits corresponding to output lines set to 1 and inputs set to 0. The bit mapping is returned in *lpOutBuffer*.

DIO_IOCTL_SET_DIRECTION (0x105): Reads a DIO bit mapping provided in *lpInBuffer*. Lines that are set to 1 in the mapping are set as outputs and those set to zero are inputs. The direction of masked lines is not changed.

DIO_IOCTL_CAPABLE_INT (0x110): Returns a four-byte bit mapping of the DIO register with bits corresponding to interrupt capabilities of the lines. Lines capable of generating interrupts are set to 1 and that are not capable of interrupts are 0. The bit mapping is returned in *lpOutBuffer*.

DIO_IOCTL_CAPABLE_WAKEUP_INT (0x111): Returns a four-byte bit mapping of the DIO register with bits corresponding to wake interrupt capabilities of the lines. Lines capable of waking the system on an interrupt are set to 1 and that are not are 0. The bit mapping is returned in *lpOutBuffer*.

DIO_IOCTL_GET_INT (0x112): Reads the four-byte bit mapping of the DIO register referenced by *lpInBuffer*. The mapping must contain one and only one set bit. The current trigger mode of the DIO interrupt corresponding to the set bit will be returned in the DIO_SIZEd location referenced by *lpOutBuffer*.

DIO_IOCTL_ENABLE_INT (0x113): Reads the DIO_INT structure referenced by *lpInBuffer*. The IntMask member must contain one and only one set bit. If the line corresponding to this bit is interrupt capable, the interrupt will be enabled with the trigger mode specified by the TriggerMode member of the DIO_INT structure. If the trigger mode given is invalid, no interrupt will be enabled and GetLastError will return DIO_ERROR_INVALID_INPUT.

DIO_IOCTL_DISABLE_INT (0x114): Reads a DIO bit mapping referenced by *lpInBuffer*. If a bit is set to 1 and corresponds to a line with an interrupt enabled, the interrupt will be disabled. Setting any other bits will have no effect. Enabled masked interrupt lines will not be disabled.

DIO_IOCTL_SET_INT_THREAD_PRIORITY (0x115): Sets the thread priority for driver's internal threads that monitor for interrupts and marshal them to driver events. The driver's default interrupt thread priority is 240.

IOCTL_GET_DRIVER_VERSION (0xA0): Retrieves the specification version that the current driver adheres to in null-terminated string format (i.e. "1.2\0"). The *lpOutBuffer* parameter to **`DeviceIoControl`** must be a pointer to a wchar_t buffer of sufficient size (10 characters) to accept the string or a failure will occur.

Error Codes

If a driver function call fails, calling GetLastError may return one of the following error codes defined in *ADSError.h*:

The following DIO error codes are defined in *ADSError.h*:

DIO_UNSPECIFIED_ERROR

A DIO error occurred but its cause was not determined.

DIO_NO_ERROR

No error occurred.

DIO_ERROR_FILE_NOT_OPEN

The DIO port is not open.

DIO_ERROR_NO_READ_PERMISSION

You don't have read permission to the DIO port.

DIO_ERROR_NO_WRITE_PERMISSION

You don't have write permission to the DIO port.

DIO_ERROR_OUTPUT_BUFFER_TOO_SMALL

A larger sized output buffer is required.

DIO_ERROR_INPUT_BUFFER_WRONG_SIZE

The size of the input buffer is not the expected size.

DIO_ERROR_NOT_IMPLEMENTED

The requested function is not implemented.

GPIO Bit to Signal Maps

GPIO Bit to Signal Map : VGX		
Bit	**Signal Name**	**Location**
0x00	GC_GPIO0	J8.1
0x01	GC_GPIO 1	J8.3
0x02	GC_GPIO 2	J8.5
0x03	GC_GPIO 3	J8.7
0x04	GC_GPIO 4	J8.9
0x05	GC_GPIO 5	J8.10
0x06	GC_GPIO 6	J8.8
0x07	GC_GPIO 7	J8.6
0x08	GC_GPIO 8	J8.4
0x09	GC_GPIO 9	J8.2
0x0A	GC_GPIO 10	J8.16
0x0B	CPLDIO0	J8.18
0x0C	CPLDIO1	J8.20
0x0D	CPLDIO2	J8.22
0x0E	CPLDIO3	J14.32
0x0F	CPLDIO4	J14.34
0x10	CPLDIO5	J14.36
0x11	CPLDIO6	J14.38
0x12	LEDOUT1	LED - Amber
0x13	LEDOUT0	LED - Green
0x14	Reserved	-
0x15	Reserved	-
0x16	Reserved	-
0x17	Reserved	-
0x18	Reserved	-
0x19	Reserved	-
0x1A	Reserved	-
0x1B	Reserved	-
0x1C	Reserved	-
0x1D	Reserved	-
0x1E	Reserved	-
0x1F	Reserved	-

GPIO Bit to Signal Maps (Continued)

GPIO Bit to Signal Map : GCX		
Bit	**Signal Name**	**Location**
0x00	UCB_IO0	J2.1
0x01	UCB_IO1	J2.3
0x02	UCB_IO2	J2.5
0x03	UCB_IO3	J2.7
0x04	UCB_IO4	J2.9
0x05	UCB_IO5	J2.10
0x06	UCB_IO6	J2.8
0x07	UCB_IO7	J2.6
0x08	UCB_IO8	J2.4
0x09	UCB_IO9	J2.2
0x0A	GPIO6	J7.38
0x0B	GPIO8	J7.36
0x0C	GPIO9	J7.34
0x0D	GPIO12	J7.32
0x0E	NSSP_SCLK	J2.25
0x0F	NSSP_SFRM	J2.23
0x10	NSSP_TDX	J2.21
0x11	NSSP_RXD	J2.19
0x12	LEDOUT1	LED - Amber
0x13	LEDOUT0	LED - Green
0x14	Reserved	-
0x15	Reserved	-
0x16	Reserved	-
0x17	Reserved	-
0x18	Reserved	-
0x19	Reserved	-
0x1A	Reserved	-
0x1B	Reserved	-
0x1C	Reserved	-
0x1D	Reserved	-
0x1E	Reserved	-
0x1F	Reserved	-

GPIO Bit to Signal Maps (Continued)

GPIO Bit to Signal Map : BitsyX		
Bit	**Signal Name**	**Location**
0x00	EIO0	J10.8
0x01	EIO1	J10.6
0x02	EIO2	J10.4
0x03	EIO3	J10.2
0x04	EIO4	J10.5
0x05	EIO5	J3.12
0x06	EIO6	J3.14
0x07	EIO7	J3.3
0x08	EIO8	J3.2
0x09	EIO9	J3.1
0x0A	LEDOUT0	LED - Green
0x0B	Reserved	-
0x0C	Reserved	-
0x0D	Reserved	-
0x0E	Reserved	-
0x0F	Reserved	-
0x10	Reserved	-
0x11	Reserved	-
0x12	Reserved	-
0x13	Reserved	-
0x14	Reserved	-
0x15	Reserved	-
0x16	Reserved	-
0x17	Reserved	-
0x18	Reserved	-
0x19	Reserved	-
0x1A	Reserved	-
0x1B	Reserved	-
0x1C	Reserved	-
0x1D	Reserved	-
0x1E	Reserved	-
0x1F	Reserved	-

GPIO Bit to Signal Maps (Continued)

GPIO Bit to Signal Map : AGX		
Bit	**Signal Name**	**Location**
0x00	EPSON_GPIO0	J8 PIN 1
0x01	EPSON_GPIO1	J8 PIN 3
0x02	EPSON_GPIO2	J8 PIN 5
0x03	EPSON_GPIO3	J8 PIN 7
0x04	EPSON_GPIO4	J8 PIN 9
0x05	EPSON_GPIO5	J8 PIN 2
0x06	EPSON_GPIO6	J8 PIN 4
0x07	EPSON_GPIO7	J8 PIN 6
0x08	EPSON_GPIO8	J8 PIN 8
0x09	EPSON_GPIO9	J8 PIN 10
0x0A	EPSON_GPIO10	J8 PIN 16
0x0B	CPLD_GPIO0	J8 PIN 18
0x0C	CPLD_GPIO1	J8 PIN 20
0x0D	CPLD_GPIO2	J8 PIN 22
0x0E	PXA_GPIO19	LED YELLOW
0x0F	PXA_GPIO20	LED GREEN
0x10	PXA_GPIO21	LED RED
0x11	Reserved	-
0x12	Reserved	-
0x13	Reserved	-
0x14	Reserved	-
0x15	Reserved	-
0x16	Reserved	-
0x17	Reserved	-
0x18	Reserved	-
0x19	Reserved	-
0x1A	Reserved	-
0x1B	Reserved	-
0x1C	Reserved	-
0x1D	Reserved	-
0x1E	Reserved	-
0x1F	Reserved	-

GPIO Bit to Signal Maps (Continued)

GPIO Bit to Signal Map : BitsyXb		
Bit	**Signal Name**	**Location**
0x00	UCBIO0	J10.8
0x01	UCBIO1	J10.6
0x02	UCBIO2	J10.4
0x03	UCBIO3	J10.2
0x04	UCBIO4	J10.5
0x05	UCBIO5	J3.12
0x06	UCBIO6	J3.14
0x07	UCBIO7	J3.3
0x08	UCBIO8	J3.2
0x09	UCBIO9	J3.1
0x0A	LEDOUT0	LED - Green
0x0B	Reserved	-
0x0C	Reserved	-
0x0D	Reserved	-
0x0E	Reserved	-
0x0F	Reserved	-
0x10	Reserved	-
0x11	Reserved	-
0x12	Reserved	-
0x13	Reserved	-
0x14	Reserved	-
0x15	Reserved	-
0x16	Reserved	-
0x17	Reserved	-
0x18	Reserved	-
0x19	Reserved	-
0x1A	Reserved	-
0x1B	Reserved	-
0x1C	Reserved	-
0x1D	Reserved	-
0x1E	Reserved	-
0x1F	Reserved	-

GPIO Bit to Signal Maps (Continued)

GPIO Bit to Signal Map : Sphere		
Bit	**Signal Name**	**Location**
0x00	PORTA-Bit1	J4.19
0x01	PORTA-Bit6	J4.20
0x02	PORTB-Bit10	J4.22
0x03	PORTB-Bit11	J4.23
0x04	PORTB-Bit12	J4.24
0x05	PORTB-Bit13	J4.25
0x06	PORTB-Bit14	J4.26
0x07	PORTE-Bit0	Red LED
0x08	PORTE-Bit1	Green LED
0x09	PORTC-Bit0	J11.1
0x0A	PORTC-Bit1	J11.2
0x0B	PORTC-Bit2	J11.3
0x0C	PORTC-Bit3	J11.4
0x0D	PORTC-Bit4	J11.5
0x0E	PORTC-Bit5	J11.6
0x0F	PORTC-Bit6	J11.7
0x10	PORTC-Bit7	J11.8
0x11	PORTD-Bit0	J11.9
0x12	PORTD-Bit1	J11.10
0x13	PORTD-Bit2	J11.11
0x14	PORTD-Bit3	J11.12
0x15	PORTD-Bit4	J11.13
0x16	PORTD-Bit5	J11.14
0x17	PORTD-Bit6	J11.15
0x18	PORTD-Bit7	J11.16
0x19	Reserved	-
0x1A	Reserved	-
0x1B	Reserved	-
0x1C	Reserved	-
0x1D	Reserved	-
0x1E	Reserved	-
0x1F	Reserved	-

To disable keypad scanning and use the pins as GPIOs' change the registry setting in adsload.reg.

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\DIO1
"EnableAltGPIO "=dword:0            ; "1" enable scanning, "0" disable scanning
```

DIO Application Header File: DIOApp.h

```
#ifndef DIO_APP_H
#define DIO_APP_H

// IOCTLs
#define DIO_IOCTL_INPUT                     0x00
#define DIO_IOCTL_OUTPUT                    0x01
#define DIO_IOCTL_HIGH                      0x02
#define DIO_IOCTL_LOW                       0x03

#define DIO_IOCTL_CAPABLE_INPUTS            0x100
#define DIO_IOCTL_CAPABLE_OUTPUTS           0x101
#define DIO_IOCTL_GET_DRIVER_MASK           0x102
#define DIO_IOCTL_SET_DRIVER_MASK           0x103
#define DIO_IOCTL_GET_DIRECTION             0x104
#define DIO_IOCTL_SET_DIRECTION             0x105

#define DIO_IOCTL_CAPABLE_INT               0x110
#define DIO_IOCTL_CAPABLE_WAKEUP_INT        0x111
#define DIO_IOCTL_GET_INT                   0x112
#define DIO_IOCTL_ENABLE_INT                0x113
#define DIO_IOCTL_DISABLE_INT               0x114
#define DIO_IOCTL_SET_INT_THREAD_PRIORITY   0x115


typedef struct DIO_INT_T
{
       DWORD IntMask;
       DWORD TriggerMode;
} DIO_INT;

#define DIO_TRIGGER_RISING                  0x01
#define DIO_TRIGGER_FALLING                 0x02
#define DIO_TRIGGER_WAKEUP_RISING           0x04
#define DIO_TRIGGER_WAKEUP_FALLING          0x08
#define DIO_TRIGGER_DISABLED                0x00

// DIO register size in bits
#define DIO_NUM_BITS                        32

// DIO register size in bytes
#define DIO_SIZE                            4

// DIO register definitions
#define DIO_0                               (1 << 0)
#define DIO_1                               (1 << 1)
#define DIO_2                               (1 << 2)
#define DIO_3                               (1 << 3)
#define DIO_4                               (1 << 4)
#define DIO_5                               (1 << 5)
#define DIO_6                               (1 << 6)
#define DIO_7                               (1 << 7)
#define DIO_8                               (1 << 8)
#define DIO_9                               (1 << 9)
#define DIO_10                              (1 << 10)
#define DIO_11                              (1 << 11)
#define DIO_12                              (1 << 12)
#define DIO_13                              (1 << 13)
#define DIO_14                              (1 << 14)
#define DIO_15                              (1 << 15)
#define DIO_16                              (1 << 16)
#define DIO_17                              (1 << 17)
#define DIO_18                              (1 << 18)
#define DIO_19                              (1 << 19)
#define DIO_20                              (1 << 20)
#define DIO_21                              (1 << 21)
```

```
#define DIO_22                          (1 << 22)
#define DIO_23                          (1 << 23)
#define DIO_24                          (1 << 24)
#define DIO_25                          (1 << 25)
#define DIO_26                          (1 << 26)
#define DIO_27                          (1 << 27)
#define DIO_28                          (1 << 28)
#define DIO_29                          (1 << 29)
#define DIO_30                          (1 << 30)
#define DIO_31                          (1 << 31)

#define DIO_LINE_UNUSED                 0

#endif // DIO_APP_H
```

Document History

The following list summarizes the changes made between releases of this document.

REV	DESCRIPTION	BY
0	• First version of document	8/1/04 ct
0	• Initial release.	10/1/04 jc
1	• Updated format to match current document template • Released a PRELIMINARY	1/12/05 ct
2	• Added missing text and IOCTLS for interrupts and wake, fixed typos	1/17/05 ct
3	• Added BitsyX Bit to Signal Map	2/9/05 ct
4	• Added LED signals to BitsyX, GCX and VGX	3/14/05 jc
5	• Added SetEvent/PulseEvent differentiation • Added DIO_IOCTL_SET_INT_THREAD_PRIORITY	1/31/06 ct
6	• Added BitsyXb signal mapping	8/24/06 jc
	• Added Sphere signal mappings	9/5/06 be
	• Update formatting for release	1/8/07 ak

Specification History

The following list summarizes the changes made between versions of the specification.

REV	DESCRIPTION	BY
1.0	Initial release.	1/12/05 ct
1.1	Added the UseSetEvent registry key and support for it in the driver	1/3/06 ct
1.2	Added DIO_IOCTL_SET_INT_THREAD_PRIORITY support	1/31/06