ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

**Artisan Technology Group**
(217) 352-9330 | sales@artisantg.com | artisantg.com

**Full-service, independent repair center**
with experienced engineers and technicians on staff.

**We buy your excess, underutilized, and idle equipment**
along with credit for buybacks and trade-ins.

**Custom engineering**
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

**Expert team | Trust guarantee | 100% satisfaction**

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Acromag / Xembedded / Xycom AP730* at our website: *Click HERE*

# AcroPack Windows Driver Software
## AP730 Driver Function Reference

**ACROMAG INCORPORATED**
**30765 South Wixom Road**
**Wixom, MI 48393-2417 U.S.A.**

**Tel: (248) 295-0310**
**Fax: (248) 624-9234**

**Copyright 2020, Acromag, Inc., Printed in the USA.**
**Data and specifications are subject to change without notice.**

The information in this document is subject to change without notice. Acromag, Inc., makes no warranty of any kind with regard to this material and accompanying software, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Further, Acromag, Inc., assumes no responsibility for any errors that may appear in this document and accompanying software and makes no commitment to update, or keep current, the information contained in this document. No part of this document may be copied or reproduced in any form, without the prior written consent of Acromag, Inc.

Copyright 2020, Acromag, Inc.

All trademarks are the property of their respective owners.

# Contents

**Contents** ..... 3
**Introduction** ..... 8
**Message Signaled Interrupts** ..... 8
**General Calling Sequence** ..... 9
**Function Reference** ..... 10
**General Functions** ..... 11
**AP730_Open** ..... 11
**AP730_OpenEx** ..... 12
**AP730_Close** ..... 13
**AP730_ReadLocation** ..... 14
**AP730_Reset** ..... 15
**Interrupt Related Functions** ..... 16
**AP730_ReadIntStatus** ..... 16
**AP730_ReadIntPending** ..... 18
**AP730_ReadIntEnable** ..... 19
**AP730_WriteIntEnable** ..... 20
**AP730_WriteIntAck** ..... 22
**AP730_SetIntEnable** ..... 23
**AP730_ClearIntEnable** ..... 24
**AP730_CtrDioReadIntStatus** ..... 25
**AP730_CtrDioWriteIntStatus** ..... 26
**AP730_SetUserCallback** ..... 27
**AP730_StartIsrSynch** ..... 30
**AP730_EndIsrSynch** ..... 31
**ADC Functions** ..... 32
**Calling Sequence Examples** ..... 32
**Continuous Conversions, Polled, No DMA** ..... 32
**Continuous Conversions, Polled, Auto DMA** ..... 32
**Continuous Conversions, Interrupt Driven, No DMA** ..... 33
**Continuous Conversions, Interrupt Driven, Auto DMA** ..... 33
**Control Register Configuration Functions** ..... 34
**AP730_AdcGetConversionMode** ..... 34
**AP730_AdcSetConversionMode** ..... 35
**AP730_AdcGetExtTrigger** ..... 36
**AP730_AdcSetExtTrigger** ..... 37
**AP730_AdcGetAutoDmaMode** ..... 38
**AP730_AdcSetAutoDmaMode** ..... 39
**AP730_AdcReadControlReg** ..... 40
**Additional Configuration Functions** ..... 41
**AP730_AdcReset** ..... 41
**AP730_AdcGetInputRange** ..... 42
**AP730_AdcSetInputRange** ..... 43
**AP730_AdcGetConversionTimer** ..... 44
**AP730_AdcSetConversionTimer** ..... 45
**AP730_AdcGetThreshold** ..... 46
**AP730_AdcSetThreshold** ..... 47
**FIFO Status Functions** ..... 48
**AP730_AdcFifoIsEmpty** ..... 48
**AP730_AdcFifoIsFull** ..... 49
**AP730_AdcFifoThresholdReached** ..... 50
**Ring Buffer Functions** ..... 51
**AP730_AdcEnableBuffers** ..... 52
**AP730_AdcDisableBuffers** ..... 53
**AP730_AdcSelectBuffers** ..... 54
**AP730_AdcGetBufferCount** ..... 55
**AP730_AdcGetRawBufferValue** ..... 56

AP730_AdcGetVoltBufferValue ..... 57
AP730_AdcClearBuffer ..... 58
AP730_AdcClearBuffers ..... 59
Calibration functions ..... 60
AP730_AdcResetCalCorrections ..... 60
AP730_AdcGetCalCorrections ..... 61
AP730_AdcSetCalCorrections ..... 62
AP730_AdcGenerateCalCount ..... 63
AP730_AdcCalcCalCorrections ..... 65
AP730_AdcReadCalCorrectionsFromFlash ..... 67
Data acquisition functions ..... 68
AP730_AdcStartConvert ..... 68
AP730_AdcStopConvert ..... 69
AP730_AdcReadFifoRaw ..... 70
AP730_AdcReadFifoVolts ..... 71
AP730_AdcCopyFifoToRingBuffers ..... 72
AP730_AdcCopyDmaBufferToRingBuffers ..... 73
Data conversion functions ..... 74
AP730_AdcRawToVolts ..... 74
Counter/Timer Functions ..... 75
Configuration Functions ..... 75
AP730_CtrReset ..... 76
AP730_CtrGetMode ..... 77
AP730_CtrSetMode ..... 78
AP730_CtrGetOutputPolarity ..... 79
AP730_CtrSetOutputPolarity ..... 80
AP730_CtrGetInAPolarity ..... 81
AP730_CtrSetInAPolarity ..... 82
AP730_CtrGetInBPolarity ..... 83
AP730_CtrSetInBPolarity ..... 84
AP730_CtrGetInCPolarity ..... 85
AP730_CtrSetInCPolarity ..... 86
AP730_CtrGetClockSource ..... 87
AP730_CtrSetClockSource ..... 88
AP730_CtrGetIntCondition ..... 89
AP730_CtrSetIntCondition ..... 90
AP730_CtrGetIntEnable ..... 91
AP730_CtrSetIntEnable ..... 92
AP730_CtrConfigureControl ..... 93
AP730_CtrGetConstantA ..... 94
AP730_CtrSetConstantA ..... 95
AP730_CtrGetConstantB ..... 96
AP730_CtrSetConstantB ..... 97
AP730_CtrConfigureConstants ..... 98
AP730_CtrReadControlReg ..... 99
AP730_CtrWriteControlReg ..... 100
AP730_CtrReadIntEnable ..... 101
AP730_CtrWriteIntEnable ..... 102
AP730_CtrReadConstantA ..... 103
AP730_CtrWriteConstantA ..... 104
AP730_CtrReadConstantB ..... 105
AP730_CtrWriteConstantB ..... 106
AP730_CtrReadCounter ..... 107
Counter/Timer start and stop Functions ..... 108
AP730_CtrStart ..... 108
AP730_CtrStop ..... 109
Toggle Counter Constant Functions ..... 110
AP730_CtrReadToggleConstant ..... 110
AP730_CtrWriteToggleConstant ..... 111
Counter/Timer status Functions ..... 112

AP730_CtrReadIntInfoReg ..... 112
DAC Functions ..... 113
Calling Sequence Examples ..... 113
Direct Access Mode ..... 113
Continuous or Single Burst Modes ..... 113
FIFO Mode, Interrupt Driven, No DMA ..... 113
FIFO Mode, Interrupt Driven, Using DMA ..... 114
Global Configuration Functions ..... 115
AP730_DacReset ..... 115
AP730_DacSetTrigOutputEnable ..... 116
AP730_DacGetTrigOutputEnable ..... 117
AP730_DacSetTimerDivider ..... 118
AP730_DacGetTimerDivider ..... 119
Channel Configuration Functions ..... 120
AP730_DacChannelReset ..... 120
AP730_DacSetOperatingMode ..... 121
AP730_DacGetOperatingMode ..... 122
AP730_DacSetMemoryAddresses ..... 123
AP730_DacGetMemoryAddresses ..... 124
AP730_DacSetTriggerSource ..... 125
AP730_DacGetTriggerSource ..... 126
AP730_DacSetOutputRange ..... 127
AP730_DacGetOutputRange ..... 128
AP730_DacSetPowerUpVoltage ..... 129
AP730_DacGetPowerUpVoltage ..... 130
AP730_DacSetThermalShutdown ..... 131
AP730_DacGetThermalShutdown ..... 132
AP730_DacSetDataFormat ..... 133
AP730_DacGetDataFormat ..... 134
AP730_DacSetOverRange ..... 135
AP730_DacGetOverRange ..... 136
AP730_DacSetClearVoltage ..... 137
AP730_DacGetClearVoltage ..... 138
AP730_DacWriteChanControl ..... 139
AP730_DacGetChanControl ..... 140
Channel Status Functions ..... 141
AP730_DacGetChannelStatus ..... 141
AP730_DacClearChannelStatus ..... 142
Data Output Functions ..... 143
AP730_DacSoftwareTrigger ..... 143
AP730_DacSetWaveformOutputEnable ..... 144
AP730_DacGetWaveformOutputEnable ..... 145
Direct Access Functions ..... 146
AP730_DacSetVout ..... 146
AP730_DacWriteInput ..... 147
AP730_DacUpdateDAC ..... 148
Continuous and Single Burst Waveform Functions ..... 149
AP730_DacWriteWaveBufferCodes ..... 149
AP730_DacWriteWaveBufferVolts ..... 150
FIFO Buffer Functions ..... 151
AP730_DacAttachFifoBuffer ..... 151
AP730_DacDetachFifoBuffer ..... 152
AP730_DacSetBufferEnd ..... 153
AP730_DacGetBufferEnd ..... 154
AP730_DacSetBufferNext ..... 155
AP730_DacGetBufferNext ..... 156
FIFO Waveform Functions ..... 157
AP730_DacSetFIFOLoading ..... 157
AP730_DacGetFIFOLoading ..... 158
AP730_DacLoadFifo ..... 159

AP730_DacWriteFifo16 ..... 161
AP730_DacWriteFifo32 ..... 162
Data Conversion Functions ..... 163
AP730_DacVoltsToCodes ..... 163
AP730_DacVoltsToCodesEx ..... 164
Calibration functions ..... 165
AP730_DacResetCalCorrections ..... 165
AP730_DacGetCalCorrections ..... 166
AP730_DacSetCalCorrections ..... 167
AP730_DacCalcCalCorrections ..... 168
AP730_DacReadCalCorrectionsFromFlash ..... 170
Digital I/O Functions ..... 171
I/O Functions ..... 171
AP730_DioWriteIoReg ..... 171
AP730_DioReadIoReg ..... 172
AP730_DioWriteChannel ..... 173
AP730_DioReadChannel ..... 174
Status Functions ..... 175
AP730_DioReadIntStatus ..... 175
AP730_DioWriteIntStatus ..... 176
Configuration Functions ..... 177
AP730_DioReset ..... 177
AP730_DioReadDirection ..... 178
AP730_DioWriteDirection ..... 179
AP730_DioReadIntEnable ..... 180
AP730_DioWriteIntEnable ..... 181
AP730_DioReadIntType ..... 182
AP730_DioWriteIntType ..... 183
AP730_DioReadIntPolarity ..... 184
AP730_DioWriteIntPolarity ..... 185
AP730_DioGetChannelDebounce ..... 186
AP730_DioSetChannelDebounce ..... 187
DMA Functions ..... 188
CDMA Functions ..... 188
AP730_CdmaReadControl ..... 188
AP730_CdmaReset ..... 189
AP730_CdmaReadStatus ..... 190
AP730_CdmaReadIntStatus ..... 191
AP730_CdmaClearIntStatus ..... 192
AP730_CdmaIsIdle ..... 193
AP730_AdcCdmaSetup ..... 194
AP730_DacCdmaSetup ..... 195
AP730_CdmaStartTransfer ..... 196
DMA Buffer Functions ..... 197
AP730_DmaGetBuffMapAddress ..... 198
AP730_DmaGetBufferSize ..... 199
AP730_DmaReadBufferByte ..... 200
AP730_DmaWriteBufferByte ..... 201
AP730_DmaReadBufferWord ..... 202
AP730_DmaWriteBufferWord ..... 203
AP730_DmaReadBufferDword ..... 204
AP730_DmaWriteBufferDword ..... 205
AP730_DmaReadBufferQword ..... 206
AP730_DmaWriteBufferQword ..... 207
AP730_DmaBufferSet ..... 208
AP730_DmaBufferCopy ..... 209
Miscellaneous Functions ..... 210
AP730_WriteCalDataToFlash ..... 210
AP730_ReadPCIeReg ..... 211
AP730_ReadFirmwareRev ..... 212

**AP730_GetBaseAddress** ..... **213**
**AP730_GetDeviceCount** ..... **214**
**AP730_ListDevices** ..... **215**
**AP730_SystemMonitor** ..... **217**
**AP730_MicrosecondDelay** ..... **218**

# Introduction

The AP730 dynamic link library provides an application programming interface (API) to the AP730 Multifunction I/O AcroPack board.

There are 32-bit and 64-bit versions of the DLL. The 64-bit DLL can only be used with 64-bit versions of Windows. The 32-bit DLL can be used with both 32-bit and 64-bit versions of Windows.

**Notes:**

- **A 32-bit application must use the 32-bit DLL and a 64-bit application must use the 64-bit DLL. 64-bit versions of Windows run 32-bit applications within the WoW64 subsystem.**

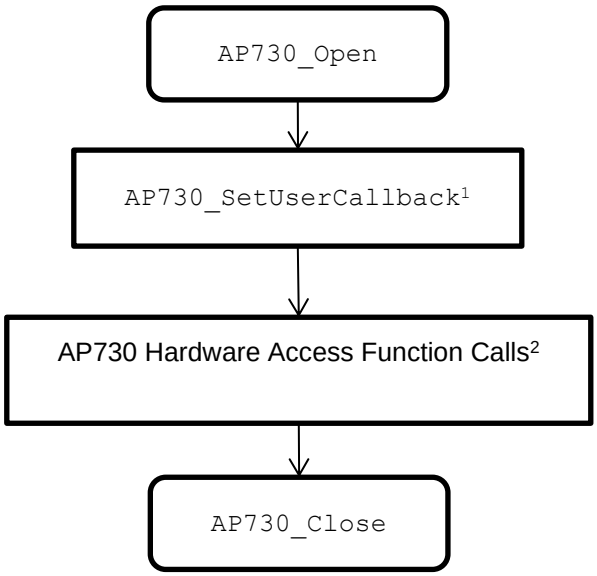
This document describes the functions provided by the AP730 Dynamic link library. It is intended as a supplement to the **AcroPack Windows Driver User's Manual**. Readers should consult that document first for information on the installation and initial configuration of the software. Except where noted otherwise, the syntax and usage of the 32-bit and 64-bit DLLs is identical.

## *Message Signaled Interrupts*

When the software and AP730 boards are installed in a system running Windows Vista or later, the software will attempt to use Message Signaled Interrupts (MSI) by default. Note, however, that if MSI capable system hardware or sufficient system resources are not present, Windows will fallback to using legacy line-based interrupts.

## *General Calling Sequence*

The following is a typical calling sequence for the AP730 functions




**Notes:**

(1) Running the board with interrupts enabled is not required. However, if you do so, you will generally want to attach a callback function.

(2) The ADC and DAC sections include several detailed Calling Sequence examples.

# Function Reference

This section describes the functions provided by the AP730 DLL for controlling the AP730 board.

Each AP730 function has the following form:

status = AP730_FunctionName( arg1, arg2, … argn )

Every function returns a 32-bit status value. This value is set to 0 when a function completes successfully or to a negative error code if a problem occurred. The Status Codes section of the **AcroPack Windows Driver User's Manual** summarizes the status values that may be returned from DLL functions.

For many of the functions, a parameter will directly correspond to a specific register, bit or group of bits present on the hardware. Although descriptions of all parameters are provided, in some cases it may be helpful to consult the **Programming Information** section of the hardware **User's Manual** for additional information about a particular setting.

Each function description includes the syntax for the C programming language.

**Digital Channel Usage Note:**

Seven of the AP730's digital channels may also be used for ADC triggering, DAC triggering or counter/timer actions. To use these actions the corresponding digital I/O port must be configured for the appropriate data direction. The *AP730_DioWriteDirection* function can be used to set the data direction.

If counter control input signals are used or an ADC or DAC trigger signal is enabled for input, then the Direction Control Register must also be set as input for channels 0 to 7.

If ADC or DAC trigger signal is enabled for output, then the data Direction Control Register must also be set as output for channels 8 to 15.

# General Functions

## AP730_Open

Opens a connection to the specified AP730 board

| Syntax | |
|---|---|
| C: | int _stdcall AP730_Open (int BoardNum,<br>int* Handle); |

**Parameters**

*BoardNum*

Specifies the desired AP730 board to open. The *BoardNum* uses zero-based numbering. A single installed board is opened with a *BoardNum* of 0. A second board of the same type (same device ID) would use a *BoardNum* of 1 and so on.

*Handle*

Receives the value used to reference this board.

**Return Value**

ERR_OK, ERR_INSUF_RESOURCES, ERR_CONNECT, ERR_MAPMEM, ERR_BAD_PARAM, ERR_BOARD_IN_USE, ERR_OUTOFHANDLES, ERR_TIMEOUT or ERR_CALIB

**Comments**

This function performs the following actions:

- A connection is opened to the specified AP730 board and the board is locked for exclusive use by the current application.
- The AXI Base Address Translation Configuration Registers are written with the physical address of the DMA buffer
- *AP730_Reset* is called to put the board into a known state
- The internal ADC and DAC calibration variables are initialized with the corresponding values stored in flash. The flash sector containing the calibration values should also contain an ASCII string (“AP730”) as an indicator that the calibration data is present. If this string is not found the function returns *ERR_CALIB*. This is not a fatal error and the board can still be used, however, no corrections will be made to the analog input or output. Functions are provided for generating calibration coefficients and writing them to flash memory.

The *AP730_GetDeviceCount* function can be used to help determine the range of valid values for *BoardNum*.

**See Also**

AP730_Close
AP730_GetDeviceCount

## AP730_OpenEx

Opens a connection to, and initializes the AP730 at the specified location.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_OpenEx (WORD Slot,`<br>`char Site,`<br>`int* Handle);` |

**Parameters**

*Slot*

The slot location in a system of the AP carrier card (0 – 31)

*Site*

The location on the carrier of the AP730 (A – H)

*Handle*

Receives the value used to reference this board.

**Return Value**

ERR_OK, ERR_INSUF_RESOURCES, ERR_CONNECT, ERR_MAPMEM, ERR_BAD_PARAM, ERR_BOARD_IN_USE, ERR_OUTOFHANDLES, ERR_INVALID_CTRLHNDL, ERR_TIMEOUT or ERR_CALIB

**Comments**

This function is an alternative to *AP730_Open* that simplifies opening a board installed at a known location.

Internally, this function calls *AP730_ListDevices* to query the system for the location information of all the installed AP730s. It then searches the list for the specified Slot and Site values and, if found, passes the corresponding Board Number to *AP730_Open*.

Note that if the specified board is already open, *AP730_ListDevices* can't inspect its Location register. In this case, *AP730_OpenEx* will return *ERR_CONNECT* rather than *ERR_BOARD_IN_USE*.

**See Also**

*AP730_Open*
*AP730_ListDevices*

## AP730_Close

Closes the connection to the specified AP730 board and releases the lock on the hardware.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_Close (int Handle);` |

**Parameters**
*Handle*
The handle to the board

**Return Value**
ERR_OK or ERR_INVALID_HNDL

**Comments**
This function disables waveform output and analog input conversions, disables interrupts, closes the connection to the specified AP730 board and frees the resources allocated to it. This function should be called for each open board prior to terminating the application.

**See Also**
AP730_Open

## AP730_ReadLocation

Reads the specified AP730 module's Location In System Register.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_ReadLocation (int Handle,`<br>`WORD* Slot,`<br>`char* Site);` |

**Parameters**

*Handle*

The handle to the board received from *AP730_Open*.

*Slot*

Receives number indicating the system slot where the carrier card is located

*Site*

Receives the carrier card site ('A' to 'H') where the AP module is located

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The *Slot* number identifies the slot location in a system of the AP carrier card. The Carrier may use backplane signals as in a VPX system or a carrier DIP switch to uniquely identify the system location of the carrier. Thirty two unique system slots can be identified by the Slot location bits (0 to 31). See the system backplane or motherboard documentation for the physical location of the carrier in the system corresponding to a *Slot* number.

## AP730_Reset

Resets the board into a known state

| Syntax | |
|---|---|
| C: | `int _stdcall AP730_Reset (int Handle);` |

**Parameters**
*Handle*
The handle to the board

**Return Value**
ERR_OK, ERR_INVALID_HNDL, ERR_TIMEOUT or ERR_CALIB

**Comments**
This function performs the following actions:

Interrupt Controller
- The Interrupt Enable Register is cleared.
- Any existing interrupt requests are cleared
- The Master IRQ Enable and Hardware Interrupt Enable bits in the Master Enable Register are set

AXI Quad SPI
- Resets the core by writing to the Quad SPI Software Reset Register
- Writes 0xE6 to the Quad SPI Control Register

AXI CDMA
- Performs a soft reset by writing to the CDMA control register. The function will return ERR_TIMEOUT if the reset does not complete within 100 microseconds

DAC
- *AP730_DacReset* is called with *ResetType* set to 1.

ADC
- *AP730_AdcReset* is called. Note that this function reads offset and gain correction values from flash memory and writes them to the ADC Offset and Gain registers. The flash sector containing the coefficient values should also contain an ASCII string ("AP730") as an indicator that the calibration data is present. If this string is not found the function returns *ERR_CALIB*.

Digital
- *AP730_DioReset* is called.

Counter/Timer
- *AP730_CtrReset* is called.

# *Interrupt Related Functions*

## AP730_ReadIntStatus

Reads the interrupt status from the Interrupt Controller's Interrupt Status Register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_ReadIntStatus (int Handle, DWORD* IntStatus);` |

**Parameters**

*Handle*

The handle to the board

*IntStatus*

Receives the status bits from the Interrupt Status register

| Bit(s) | Function | |
|---|---|---|
| 0-3 | When set indicates DAC channel 0 to 3 requires service. | |
| | 0 | No service required |
| | 1 | In single burst mode indicates burst complete. In FIFO mode, indicates FIFO is half full |
| 4 | When set indicates an AXI CDMA interrupt | |
| | 0 | Interrupt inactive |
| | 1 | Interrupt active |
| 5 | When set indicates that the ADC Data FIFO contains a number of samples equal to or greater than the value stored in the FIFO Full Threshold Register | |
| | 0 | Interrupt inactive |
| | 1 | Interrupt active |
| 6 | When set indicates a Counter interrupt | |
| | 0 | No service required |
| | 1 | Read the Counter Interrupt Information register to determine the type of interrupt |
| 7 | When set indicates a Digital interrupt | |
| | 0 | No service required |
| | 1 | Read the Global Interrupt Status register to identify the interrupting channel(s) |

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The contents of this register indicate the presence or absence of an active interrupt for each of the active interrupting sources. Each bit in this register that is set to a '1' indicates an active interrupt signal on the corresponding interrupt input. Bits that are '0' are not active. The bits in the Interrupt Status register are independent of the interrupt enable bits in the Interrupt Enable register. An Interrupt, even if not enabled, can still show up as active in the Interrupt Status register.

**See Also**

`AP730_ReadIntPending`
`AP730_CtrDioReadIntStatus`

AP730_CtrReadIntInfoReg

## AP730_ReadIntPending

Reads the Interrupt Controller's Interrupt Pending Register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_ReadIntPending (int Handle, DWORD* IntPending);` |

**Parameters**

*Handle*

The handle to the board

*IntPending*

Receives the value read from the Interrupt Pending register

| Bit(s) | Function | |
|---|---|---|
| 0-3 | When set indicates DAC channel 0 to 3 interrupt is pending | |
| | 0 | No interrupt pending |
| | 1 | In single burst mode indicates burst complete. In FIFO mode, indicates FIFO is half full |
| 4 | When set indicates an AXI CDMA interrupt is pending | |
| | 0 | No interrupt pending |
| | 1 | Interrupt pending |
| 5 | When set indicates that the ADC Data FIFO threshold met or exceeded interrupt is pending | |
| | 0 | No interrupt pending |
| | 1 | Interrupt pending |
| 6 | When set indicates a Counter interrupt is pending | |
| | 0 | No interrupt pending |
| | 1 | Interrupt pending. Read the Counter Interrupt Information register to determine the type of interrupt |
| 7 | When set indicates a Digital interrupt is pending | |
| | 0 | No interrupt pending |
| | 1 | Interrupt pending. Read the Global Interrupt Status register to identify the interrupting channel(s) |

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The contents of this register indicate the presence or absence of active interrupt signals that are also enabled. Each bit in this register is the logical AND of the bits in the Interrupt Status register and the Interrupt Enable register.

**See Also**

`AP730_ReadIntStatus`
`AP730_SetIntEnable`

## AP730_ReadIntEnable

Reads the Interrupt Controller's Interrupt Enable Register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_ReadIntEnable (int Handle, DWORD* IntEnable);` |

**Parameters**

*Handle*

The handle to the board

*IntEnable*

Receives the value read from the Interrupt Enable register

| Bit(s) | Function | |
|---|---|---|
| 0-3 | When set indicates DAC channel 0 to 3 is enabled | |
| | 0 | Interrupt disabled |
| | 1 | Interrupt enabled |
| 4 | When set indicates an AXI CDMA interrupt is enabled | |
| | 0 | Interrupt disabled |
| | 1 | Interrupt enabled |
| 5 | When set indicates that the ADC Data FIFO threshold met or exceeded interrupt is enabled | |
| | 0 | Interrupt disabled |
| | 1 | Interrupt enabled |
| 6 | When set indicates a Counter interrupt is enabled | |
| | 0 | Interrupt disabled |
| | 1 | Interrupt enabled |
| 7 | When set indicates a Digital interrupt is enabled | |
| | 0 | Interrupt disabled |
| | 1 | Interrupt enabled |

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

A '1' bit in this register enables the corresponding Interrupt Status bit to cause assertion of the interrupt output. An Interrupt Enable bit set to '0' does not inhibit an interrupt condition from being captured. It will still show up in the Interrupt Status register even when not enabled here. To show up in the Interrupt Pending register it needs to be enabled here. Writing a '0' to a bit disables, or masks, the generation of interrupt output for the corresponding interrupt input signal. Note, however, that disabling an interrupt input is not the same as clearing it. Disabling an active interrupt prevents that interrupt from reaching the IRQ output. When it is re-enabled, the interrupt immediately generates a request on the IRQ output. An interrupt must be cleared by writing to the Interrupt Acknowledge Register. Reading this Interrupt Enable register indicates which interrupt inputs are enabled; where a '1' indicates the input is enabled and a '0' indicates the input is disabled.

**See Also**

`AP730_WriteIntEnable`
`AP730_SetIntEnable`
`AP730_ClearIntEnable`
`AP730_WriteIntAck`

## AP730_WriteIntEnable

Writes a value to the Interrupt Controller's Interrupt Enable Register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_WriteIntEnable (int Handle, DWORD IntEnable);` |

**Parameters**

*Handle*

The handle to the board

*IntEnable*

The interrupt sources to enable

| Bit(s) | Function | |
|---|---|---|
| 0-3 | When set indicates DAC channel 0 to 3 is enabled | |
| | 0 | Interrupt disabled |
| | 1 | Interrupt enabled |
| 4 | When set indicates an AXI CDMA interrupt is enabled | |
| | 0 | Interrupt disabled |
| | 1 | Interrupt enabled |
| 5 | When set indicates that the ADC Data FIFO threshold met or exceeded interrupt is enabled | |
| | 0 | Interrupt disabled |
| | 1 | Interrupt enabled |
| 6 | When set indicates a Counter interrupt is enabled | |
| | 0 | Interrupt disabled |
| | 1 | Interrupt enabled |
| 7 | When set indicates a Digital interrupt is enabled | |
| | 0 | Interrupt disabled |
| | 1 | Interrupt enabled |

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

A '1' bit in the Interrupt Enable Register enables the corresponding Interrupt Status bit to cause assertion of the interrupt output. An Interrupt Enable bit set to '0' does not inhibit an interrupt condition from being captured. It will still show up in the Interrupt Status register even when not enabled here. To show up in the Interrupt Pending register it needs to be enabled here. Writing a '0' to a bit disables, or masks, the generation of interrupt output for the corresponding interrupt input signal. Note, however, that disabling an interrupt input is not the same as clearing it. Disabling an active interrupt prevents that interrupt from reaching the IRQ output. When it is re-enabled, the interrupt immediately generates a request on the IRQ output. An interrupt must be cleared by writing to the Interrupt Acknowledge Register. Reading this Interrupt Enable register indicates which interrupt inputs are enabled; where a '1' indicates the input is enabled and a '0' indicates the input is disabled.

**See Also**

`AP730_ReadIntStatus`
`AP730_ReadIntPending`
`AP730_ReadIntEnable`
`AP730_SetIntEnable`

AP730_ClearIntEnable
AP730_WriteIntAck

## AP730_WriteIntAck

Writes a value to the Interrupt Controller's Interrupt Acknowledge Register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_WriteIntAck (int Handle, DWORD IntAck);` |

**Parameters**

*Handle*

The handle to the board

*IntAck*

The value written to the Interrupt Acknowledge register

| Bit(s) | Function |
|---|---|
| 3-0 | Clear DAC channel 0 to 3 interrupt request |
| 4 | Clear AXI CDMA interrupt request |
| 5 | Clear ADC Data FIFO threshold met or exceeded interrupt request |
| 6 | Clear Counter interrupt request |
| 7 | Clear Digital interrupt request |

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

The Interrupt Acknowledge register is a write-only location that clears the interrupt request associated with selected interrupt inputs. Note that writing '1' to a bit in Interrupt Acknowledge register clears the corresponding bit in Interrupt Status register, and clears the same bit in Interrupt Acknowledge register.

Writing a '1' to a bit location in the Interrupt Acknowledge register will clear the interrupt request that was generated by the corresponding interrupt input. An interrupt input that is active and masked by writing a '0' to the corresponding bit in the Interrupt Enable register will remain active until cleared by acknowledging it. Unmasking an active interrupt causes an interrupt request output to be generated (if the Master Interrupt Enable bit-0 in the Master Enable register is set). Writing 0s has no effect as does writing a '1' to a bit that does not correspond to an active input or for which an interrupt input does not exist.

**See Also**

`AP730_ReadIntStatus`
`AP730_ReadIntPending`
`AP730_ReadIntEnable`
`AP730_WriteIntEnable`
`AP730_SetIntEnable`
`AP730_ClearIntEnable`

## AP730_SetIntEnable

Writes a value to the Interrupt Controller's Set Interrupt Enable Register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_SetIntEnable (int Handle, DWORD IntEnable);` |

**Parameters**

*Handle*

The handle to the board

*IntEnable*

The interrupt source to enable

| Bit(s) | Function |
|---|---|
| 3-0 | Set DAC channel 0 to 3 interrupt enable |
| 4 | Set AXI CDMA interrupt enable |
| 5 | Set ADC Data FIFO threshold met or exceeded interrupt enable |
| 6 | Set Counter Interrupt enable |
| 7 | Set Digital Interrupt enable |

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

The Set Interrupt Enable register is a location used to set Interrupt Enable register bits in a single atomic operation, rather than using a read / modify / write sequence. Writing a '1' to a bit location in Set Interrupt Enable register will set the corresponding bit in the Interrupt Enable register. Writing 0's does nothing, as does writing a '1' to a bit location that corresponds to a non-existing interrupt input.

**See Also**

`AP730_ReadIntEnable`
`AP730_WriteIntEnable`
`AP730_ClearIntEnable`

## AP730_ClearIntEnable

Writes a value to the Interrupt Controller's Clear Interrupt Enable Register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_ClearIntEnable (int Handle, DWORD IntClear);` |

**Parameters**

*Handle*

The handle to the board

*IntClear*

The value written to the Clear Interrupt Enable register

| Bit(s) | Function |
|---|---|
| 3-0 | Clear DAC channel 0 to 3 interrupt enable |
| 4 | Clear AXI CDMA interrupt enable |
| 5 | Clear ADC Data FIFO threshold met or exceeded interrupt enable |
| 6 | Clear Counter Interrupt enable |
| 7 | Clear Digital Interrupt enable |

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

The Clear Interrupt Enable register is a location used to clear Interrupt Enable register bits in a single atomic operation, rather than using a read / modify / write sequence. Writing a '1' to a bit location in Clear Interrupt Enable register will clear the corresponding bit in the Interrupt Enable register. Writing 0's does nothing, as does wiring a '1' to a bit location that corresponds to a non-existing interrupt input.

**See Also**

`AP730_ReadIntEnable`
`AP730_WriteIntEnable`
`AP730_SetIntEnable`

## AP730_CtrDioReadIntStatus

Reads the interrupt status from the Global Interrupt Status Register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrDioReadIntStatus (int Handle, DWORD* IntStatus);` |

**Parameters**

*Handle*

The handle to the board

*IntStatus*

Receives the status bits from the Global Interrupt Status register

| Bit(s) | Function | |
|---|---|---|
| 0-15 | When set indicates digital channel 0 to 15 interrupt is pending | |
| | 0 | No interrupt pending |
| | 1 | Interrupt pending |
| 16-23 | Not used | |
| 24 | When set indicates a Counter/Timer interrupt is pending | |
| | 0 | No interrupt pending |
| | 1 | Interrupt pending |
| 25-31 | Not used | |

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The Global Interrupt Status Register reflects the status of each of the Digital interrupting channels, and Counter Timer interrupts. A function that does not have interrupts enabled will never set its interrupt status flag.

**See Also**

`AP730_CtrDioWriteIntStatus`

## AP730_CtrDioWriteIntStatus

Writes a value to the Global Interrupt Status Register to release pending interrupts

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrDioWriteIntStatus (int Handle, DWORD* Clear);` |

**Parameters**

*Handle*

The handle to the board

*Clear*

The value written to the Global Interrupt Status register

| Bit(s) | Function |
|---|---|
| 0-15 | Clear digital channel 0 to 15 interrupt request |
| 16-23 | Not used |
| 24 | Clear Counter/Timer interrupt request |
| 25-31 | Not used |

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The Global Interrupt Status Register reflects the status of each of the Digital interrupting channels, and Counter Timer interrupts. Writing a logic “1” to a bit releases the pending interrupt. However, if the condition which caused the interrupt to occur remains, the interrupt will be generated again (unless disabled via its Interrupt Enable Register). Writing “0” to a bit location has no effect; that is, a pending interrupt will remain pending.

**See Also**

`AP730_CtrDioReadIntStatus`
`AP730_CtrWriteIntEnable`
`AP730_DioWriteIntEnable`

## AP730_SetUserCallback

Sets the user defined function that will be called from the DLL's internal interrupt service routine.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_SetUserCallback (int Handle, AP730_USER_HANDLER Fcn, BOOL Replace);` |

**Parameters**

*Handle*

The handle to the board

*Fcn*

(C/C++) Pointer to callback function

*Replace*

Set to TRUE to use the callback function in place of the DLL's internal interrupt service routine.

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

This function allows you to designate the user callback the DLL will invoke when an interrupt is generated by the specified board. The *Replace* parameter indicates whether the callback function should override the DLL's internal interrupt service routine or be called after the internal ISR has processed the interrupt.

The callback function must have the following signature:

```
C: void _stdcall YourIsrName (
       int Handle,
       DWORD Pending,
       DWORD CtrIntInfo,
       DWORD DioIntStatus,
       DWORD DmaIntEvent,
       WORD AdcNewData,
       WORD AdcMissedData);
```

**Callback Parameters**

*Handle*

The handle to the board that interrupted.

*Pending*

Value of the Interrupt Pending register before the kernel driver disabled the interrupting sources

*CtrInfInfo*

If a counter interrupt was pending, this was the value of the Counter Interrupt Information Register before the interrupt was cleared

*DioIntStatus*

If a digital I/O interrupt was pending, this was the value of the digital channel bits in the Global Interrupt Status register before the interrupt was cleared

*DmaIntEvent*

If CDMA interrupt was pending, this mask indicates the CDMA interrupt events

*AdcNewData*
If an ADC threshold interrupt was pending, this bit mask of ADC channels indicates which channels had data copied to their copied to the ring buffers.

*AdcMissedData*
If an ADC threshold interrupt was pending, this bit mask of ADC channels indicates the channels which did not have data copied to their ring buffers because their buffers were full.

When *AP730_SetUserCallback* is called with *Replace* is set to FALSE, the following sequence of events takes place when an interrupt occurs:

1. The kernel level driver reads the Interrupt Controller's Interrupt Pending register and writes the value read to the Clear Interrupt Enable Register. The driver signals the AP730 DLL and passes it the value read from the Interrupt Pending register.
2. A dedicated thread within the DLL invokes the interrupt service routine (ISR).
3. The ISR processes the interrupt as follows:
   **CDMA Interrupts**
   - A "DMA Event" mask value is created identifying interrupts that are both active in the CDMA Status register and enabled in the CDMA Control register. This value is written to the CDMA Status register.
   - If an "Interrupt on Complete" interrupt was pending and ADC Auto DMA mode is enabled, the *AP730_AdcCopyDmaBufferToRingBuffers* function is called to copy values from the DMA buffer to the currently selected ring buffers. The value of the FIFO Full Threshold register determines the number of values copied. The values the function assigned to its *New* and *Missed* data arguments are saved.

   **ADC Data FIFO threshold met or exceeded interrupt**
   - If ADC Auto DMA mode is NOT enabled, the *AP730_AdcCopyFifoToRingBuffers* function is called to copy values from the FIFO to the currently selected ring buffers. The value of the FIFO Full Threshold register determines the number of values copied. The values the function assigned to its *New* and *Missed* data arguments are saved.

   **DAC Channel interrupts**
   - For each channel with a FIFO-half-full interrupt pending, *AP730_DacLoadFifo* is called.
   - For each channel with a burst-complete interrupt pending, *AP730_DacClearChannelStatus* is called.

   **Counter interrupts**
   - The contents of the Counter Interrupt Information Register are read and saved

   **Digital interrupts**
   - The Digital Channel Interrupt Pending bits in the Global Interrupt Status Register are read and saved
- The value read from the Global Interrupt Status Register is written back to clear pending counter/time and digital channel interrupts.
- The value read from the Interrupt Controller's Interrupt Pending register is written to the Interrupt Acknowledge register.
- Typically, the value read from the Interrupt Controller's Interrupt Pending register is then written to the Set Interrupt Enable register to re-enable the interrupt(s). Note, however, that if a serious error is encountered while servicing an interrupt source, interrupts will not be re-enabled for the source. For example, if no buffer was attached to receive analog input data, the ADC threshold interrupt would be left disabled

4. The ISR then calls the user function passing the handle of the board that caused the interrupt and the saved callback function arguments.
5. The callback function performs some application specific action.

For flexibility, *Replace* may be set to TRUE to allow the user callback to override the functionality of the internal ISR. In this case the following sequence of events takes place when an interrupt occurs:

1. The kernel level driver reads the Interrupt Controller's Interrupt Pending register and writes the value read to the Clear Interrupt Enable register. The driver signals the AP730 DLL and passes it the value read from the Interrupt Pending register.
2. The DLL calls the user function passing the handle of the board that caused the interrupt and the value read from the Interrupt Pending register. The other callback function arguments are passed as zeroes.
3. The callback function performs some application specific action. After servicing the interrupt(s) it should call *AP730_WriteIntAck* to clear the interrupt requests and *AP730_SetIntEnable* to restore the Interrupt Enable bits disabled by the driver.

The provided demonstration program (with source code) includes example callback functions that override and do not override the DLL's internal ISR . See the *ISR_Basic* and *ISR_Override* functions in **AP730Demo.c**.

Setting a callback function is optional. To detach a previously set callback function, call *AP730_SetCallback* with *Fcn* set to *NULL* (C/C++). In this case, the *Replace* parameter is ignored and the DLL's internal ISR will once again process all interrupts.

## AP730_StartIsrSynch

Used to synchronize application threads with the ISR thread and to synchronize multiple threads within an application with each other.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_StartIsrSynch (int Handle);` |

**Parameters**

*Handle*

The handle to the board

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

See the Synchronization topic in the **AcroPack Windows Driver User's Manual** for additional information on the use of this function.

**See Also**

AP730_EndIsrSynch

## AP730_EndIsrSynch

Ends synchronized execution of ISR and application threads.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_EndIsrSynch (int Handle);` |

**Parameters**
*Handle*
The handle to the board

**Return Value**
ERR_OK or ERR_INVALID_HNDL

**Comments**
See the Synchronization topic in the **AcroPack Windows Driver User's Manual** for additional information on the use of this function.

**See Also**
AP730_StartIsrSynch

# *ADC Functions*

## CALLING SEQUENCE EXAMPLES

The following are some typical calling sequences for the AP730 ADC functions. These sequences would be performed after opening the board with *AP730_Open* and before closing the board with *AP730_Close*. The examples assume that all registers are starting from their default/reset state.

### Continuous Conversions, Polled, No DMA

| Function | Comment |
|---|---|
| AP730_AdcSetConversionMode | Set Conversion mode to Continuous |
| AP730_AdcSetInputRange | Enable some channels and set their input ranges |
| AP730_AdcSetConversionTimer | Adjust conversion time (if necessary) |
| AP730_AdcSetThreshold | Adjust threshold (if necessary) |
| AP730_AdcStartConvert | Trigger conversions |
| AP730_AdcFifoThresholdReached | Call this function until it indicates the threshold has been reached |
| AP730_AdcCopyFifoToRingBuffers | Read the data out of the FIFO and into the ring buffers |

### Continuous Conversions, Polled, Auto DMA

| Function | Comment |
|---|---|
| AP730_AdcCdmaSetup | Configure the DMA transfer. Set the IntMask argument to zero. |
| AP730_AdcSetConversionMode | Set Conversion mode to Continuous |
| AP730_AdcSetAutoDmaMode | Enable automatic DMA transfer |
| AP730_AdcSetInputRange | Enable some channels and set their input ranges |
| AP730_AdcSetConversionTimer | Adjust conversion time (if necessary) |
| AP730_AdcSetThreshold | Adjust threshold (if necessary) |
| AP730_AdcStartConvert | Trigger conversions |
| AP730_CdmaReadIntStatus | Call this function until bit-12 indicates the transfer has completed |
| AP730_AdcCopyDmaBufferToRingBuffers | Read the data out of the DMA buffer and into the ring buffers |

## Continuous Conversions, Interrupt Driven, No DMA

| Function | Comment |
|---|---|
| AP730_AdcSetConversionMode | Set Conversion mode to Continuous |
| AP730_AdcSetInputRange | Enable some channels and set their input ranges |
| AP730_AdcSetConversionTimer | Adjust conversion time (if necessary) |
| AP730_AdcSetThreshold | Adjust threshold (if necessary) |
| AP730_SetUserCallback | Attach an interrupt callback function. The callback will be invoked after the DLL's internal interrupt handler has read the data out of the FIFO. |
| AP730_SetIntEnable | Enable ADC Data FIFO threshold met or exceeded interrupt (bit 9) |
| AP730_AdcStartConvert | Trigger conversions |
| **Within callback function** | |
| | The callback is invoked after the DLL's internal interrupt handler has read the data out of the FIFO and into the ring buffers. Process the data as required by your application. |

## Continuous Conversions, Interrupt Driven, Auto DMA

| Function | Comment |
|---|---|
| AP730_AdcCdmaSetup | Configure the DMA transfer. Set bit-12 (interrupt-on-complete) in the IntMask argument. |
| AP730_AdcSetConversionMode | Set Conversion mode to Continuous |
| AP730_AdcSetAutoDmaMode | Enable automatic DMA transfer |
| AP730_AdcSetInputRange | Enable some channels and set their input ranges |
| AP730_AdcSetConversionTimer | Adjust conversion time (if necessary) |
| AP730_AdcSetThreshold | Adjust threshold (if necessary) |
| AP730_SetUserCallback | Attach an interrupt callback function. The callback will be invoked after the DLL's internal interrupt handler has read the data out of the DMA buffer. |
| AP730_SetIntEnable | Enable CDMA interrupt (bit 8) |
| AP730_AdcStartConvert | Trigger conversions |
| **Within callback function** | |
| | The callback is invoked after the DLL's internal interrupt handler has read the data out of the DMA buffer and into the ring buffers. Process the data as required by your application. |

## CONTROL REGISTER CONFIGURATION FUNCTIONS

### AP730_AdcGetConversionMode

Reads the conversion mode setting from the ADC Control Register.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcGetConversionMode (int Handle, DWORD* ConvMode);` |

**Parameters**

*Handle*

The handle to the board

*ConvMode*

Receives the conversion mode setting value (0 - 2)

0: Disabled
1: Single Conversion Mode
2: Continuous Conversion Mode

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**See Also**

AP730_AdcSetConversionMode

## AP730_AdcSetConversionMode

Sets the conversion mode bits in the ADC Control Register.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcSetConversionMode (int Handle, DWORD ConvMode);` |

**Parameters**

*Handle*

The handle to the board

*ScanMode*

The new conversion mode setting (0 - 2)

0: Disabled
1: Single Conversion Mode
2: Continuous Conversion Mode

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**See Also**

`AP730_AdcGetConversionMode`

## AP730_AdcGetExtTrigger

Reads the external trigger setting from the ADC Control Register.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcGetExtTrigger (int Handle, DWORD* Trigger);` |

**Parameters**

*Handle*

The handle to the board

*Trigger*

Receives the external trigger setting (0 - 2)

0: External Trigger Disable
1: External Trigger Set as Input
2: External Trigger Set as Output

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**See Also**

`AP730_AdcSetExtTrigger`

## AP730_AdcSetExtTrigger

Sets the external trigger bits in the ADC Control Register.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcSetExtTrigger (int Handle, DWORD Trigger);` |

**Parameters**

*Handle*

The handle to the board

*Trigger*

The new data external trigger setting (0 - 2)

0: External Trigger Disable
1: External Trigger Set as Input
2: External Trigger Set as Output

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**See Also**

`AP730_AdcGetExtTrigger`

## AP730_AdcGetAutoDmaMode

Reads the Auto DMA Mode setting from the ADC Control Register.

| Syntax | |
|---|---|
| C: | `int _stdcall AP730_AdcGetAutoDmaMode (int Handle, BOOL* AutoDma);` |

**Parameters**

*Handle*

The handle to the board

*AutoDma*

Receives the Auto DMA Mode setting

TRUE: DMA transfers are initiated automatically in hardware when the FIFO threshold has been met or exceeded

FALSE: Hardware initiated DMA transfers disabled

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

CDMA transfers should be configured using the *AP730_AdcCdmaSetup* function prior to enabling Auto DMA Mode.

**See Also**

AP730_AdcSetAutoDmaMode
AP730_AdcCdmaSetup

## AP730_AdcSetAutoDmaMode

Sets the Auto DMA Mode setting from the ADC Control Register.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcSetAutoDmaMode (int Handle, BOOL AutoDma);` |

**Parameters**

*Handle*

The handle to the board

*AutoDma*

The new Auto DMA Mode setting

TRUE: DMA transfers are initiated automatically in hardware when the FIFO threshold has been met or exceeded

FALSE: Hardware initiated DMA transfers disabled

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

CDMA transfers should be configured using the *AP730_AdcCdmaSetup* function prior to enabling Auto DMA Mode.

**See Also**

AP730_AdcGetAutoDmaMode
AP730_AdcCdmaSetup

## AP730_AdcReadControlReg

Reads the value of ADC Control Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcReadControlReg (int Handle, WORD* Data);` |

**Parameters**

*Handle*

The handle to the board

*Data*

Receives the 16-bit value of the register.

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function may be used to check the board configuration.

## ADDITIONAL CONFIGURATION FUNCTIONS

### AP730_AdcReset

Resets the ADC logic into a default state

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcReset (int Handle);` |

**Parameters**
*Handle*
The handle to the board

**Return Value**
ERR_OK, ERR_BAD_PARAM, ERR_CALIB or ERR_INVALID_HNDL

**Comments**
This function performs the following actions:

- Performs a software reset by setting bit 15 in the ADC Control Register.
- ADC Data FIFO threshold interrupts are disabled in the Interrupt Enable register
- The ADC Conversion Timer Register is set to 0x1869.
- The offset and gain correction values are read from flash memory and written to the ADC Offset and Gain registers. The flash sector containing the coefficient values should also contain an ASCII string (“AP730”) as an indicator that the calibration data is present. If this string is not found the function returns ERR_CALIB. In this case, no corrections will be made to the analog input. Functions are provided for generating calibration coefficients and writing them to flash memory.

## AP730_AdcGetInputRange

Reads the channel's Range/Enable setting from the ADC Range Select Channel Enable Register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcGetInputRange (int Handle, DWORD Channel, DWORD* Range);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

The channel number (0 – 7)

*Range*

Receives the range/enable setting

| Value | Range/Enable |
|---|---|
| 0 | Disabled |
| 1 | 0 Vdc to 5.12 Vdc |
| 2 | -5.0 Vdc to 5.0 Vdc |
| 3 | -5.12 Vdc to 5.12 Vdc |
| 4 | 0 Vdc to 10 Vdc |
| 5 | 0 Vdc to 10.24 Vdc |
| 6 | -10 Vdc to 10 Vdc |
| 7 | -10.24 Vdc to 10.24 Vdc |

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**See Also**

`AP730_AdcSetInputRange`

## AP730_AdcSetInputRange

Writes the channel's Range/Enable setting to the ADC Range Select Channel Enable Register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcSetInputRange (int Handle,`<br>`DWORD Channel,`<br>`DWORD Range);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

The channel number (0 – 7)

*Range*

The new the range/enable setting

| Value | Range/Enable |
|---|---|
| 0 | Disabled |
| 1 | 0 Vdc to 5.12 Vdc |
| 2 | -5.0 Vdc to 5.0 Vdc |
| 3 | -5.12 Vdc to 5.12 Vdc |
| 4 | 0 Vdc to 10 Vdc |
| 5 | 0 Vdc to 10.24 Vdc |
| 6 | -10 Vdc to 10 Vdc |
| 7 | -10.24 Vdc to 10.24 Vdc |

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**See Also**

`AP730_AdcGetInputRange`

## AP730_AdcGetConversionTimer

Reads the ADC Conversion Timer Register.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcGetConversionTimer (int Handle, DWORD* Count);` |

**Parameters**

*Handle*

The handle to the board

*Count*

Receives the 32-bit value from the register.

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The Channel Conversion Timer is used to control the frequency at which the conversion cycle is repeated for all enabled channels. The time period between trigger pulses is described by the following equation:

(Conversion Timer Value + 1) ÷ 62,500,000Hz = T (in seconds)

**See Also**

`AP730_AdcSetConversionTimer`

## AP730_AdcSetConversionTimer

Writes the ADC Conversion Timer Register.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcSetConversionTimer (int Handle, DWORD Count);` |

**Parameters**

*Handle*

The handle to the board

*Count*

The new conversion timer value (78 to 4,294,967,295)

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The Channel Conversion Timer is used to control the frequency at which the conversion cycle is repeated for all enabled channels. The time period between trigger pulses is described by the following equation:

(Conversion Timer Value + 1) ÷ 62,500,000Hz = T (in seconds)

**See Also**

`AP730_AdcSetConversionTimer`

## AP730_AdcGetThreshold

Reads the ADC FIFO Full Interrupt Threshold Register.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcGetThreshold (int Handle, WORD* Threshold);` |

**Parameters**

*Handle*

The handle to the board

*Threshold*

Receives the value from the register

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The ADC FIFO Full Interrupt Threshold register is a 10-bit register that is used to control when an interrupt will be generated and/or DMA transfer will occur. The events, enabled in the ADC Control Register will occur when the FIFO contains the number of samples equal to the FIFO Full Interrupt Threshold value.

**See Also**

`AP730_AdcSetThreshold`

## AP730_AdcSetThreshold

Writes a value to the ADC FIFO Full Interrupt Threshold Register.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcSetThreshold (int Handle, WORD Threshold);` |

**Parameters**

*Handle*

The handle to the board

*Threshold*

The new FIFO Full Interrupt Threshold Register value (5 – 1023)

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The FIFO Full Interrupt Threshold register is a 10-bit register that is used to control when an interrupt will be generated and/or DMA transfer will occur. The events, enabled in the ADC Control Register will occur when the FIFO contains the number of samples equal to the FIFO Full Interrupt Threshold value.

**See Also**

`AP730_AdcGetThreshold`

## FIFO STATUS FUNCTIONS

### AP730_AdcFifoIsEmpty

Reads the state of the FIFO Empty Status bit in the ADC Control register

| Syntax | |
|---|---|
| C: | int _stdcall AP730_AdcFifoIsEmpty (int Handle,<br>BOOL* IsEmpty); |

**Parameters**

*Handle*

The handle to the board

*IsEmpty*

Receives the status bit state:
FALSE: FIFO is not empty
TRUE: FIFO is empty

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**See Also**

AP730_AdcFifoIsFull
AP730_AdcFifoThresholdReached

## AP730_AdcFifoIsFull

Reads the state of the FIFO Full Status bit in the ADC Control register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcFifoIsFull (int Handle,`<br>`BOOL* IsFull);` |

**Parameters**

*Handle*

The handle to the board

*IsFull*

Receives the status bit state:

FALSE: FIFO is not full

TRUE: FIFO is full

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**See Also**

AP730_AdcFifoIsEmpty
AP730_AdcFifoThresholdReached

## AP730_AdcFifoThresholdReached

Reads the state of the FIFO Threshold Status bit in the ADC Control register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcFifoThresholdReached (int Handle, BOOL* Reached);` |

**Parameters**

*Handle*

The handle to the board

*Reached*

Receives the status bit state:

FALSE: The FIFO has less samples than defined by the Threshold Register

TRUE: The FIFO samples are greater than or equal to the Threshold

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**See Also**

AP730_AdcFifoIsFull
AP730_AdcFifoIsEmpty

## RING BUFFER FUNCTIONS

The DLL implements two sets of internal ring buffers. The raw data ring buffer set consists of one raw data ring buffer per channel. Likewise, the volt data ring buffer set consists of one volt data ring buffer per channel. Neither set is enabled by default. A ring buffer set is enabled by calling *AP730_AdcEnableBuffers* and specifying the desired buffer type.

Once a buffer set has been enabled, the *AP730_AdcCopyFifoToRingBuffers* function can be used to read data out of the Sample FIFO and copy it to the ring buffers using the channel tags. When using Auto DMA Mode, the *AP730_AdcCopyDmaBufferToRingBuffers* can be used to transfer data from the DMA buffer to the ring buffers. When using interrupts, the DLL's internal interrupt service routine (ISR) calls the appropriate function based on whether Auto DMA Mode is enabled.

If only the raw data ring buffer set has been enabled, 16-bit raw data is read from the hardware or DMA buffer and then copied to the ring buffers. If only the volt data ring buffer set has been enabled, values converted to voltages and then copied to the buffers.

If both raw and voltage ring buffer sets have been enabled, the software defaults to using the buffer set that was enabled first. The *AP730_AdcSelectBuffers* function can be used to alter this selection.

Each ring buffer supports single producer/single consumer access. In other words, you should not attempt to read from a ring buffer using more than one thread.

Using the ring buffers is not required. You can choose to not enable them and read the sample data using *AP730_AdcReadFifoRaw* or *AP730_AdcReadFifoVolts* instead. When using interrupts, you would need to define a callback function that overrides the DLL's internal ISR. The AP730 Demo project source code illustrates acquiring data with and without using the ring buffers.

## AP730_AdcEnableBuffers

Allocates and configures a ring buffer set of the specified type

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcEnableBuffers (int Handle,`<br>`DWORD BufferType,`<br>`DWORD Capacity);` |

**Parameters**

*Handle*

The handle to the board

*BufferType*

0: Raw ring buffers

1: Voltage ring buffers

*Capacity*

The maximum number of values each channel buffer can hold. This must be a power of 2.

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_INSUF_RESOURCES or ERR_INVALID_HNDL

**Comments**

This function allocates 8 ring buffers (1 per channel) of the specified type and capacity. For performance reasons, the capacity must be a power of 2.

Each ring buffer supports single producer/single consumer access. In other words, you should not attempt to read from a ring buffer using more than one thread.

This function can be used to add or change the buffer set used to receive data. To disable a buffer set and free its resources, call *AP730_AdcDisableBuffers.* Buffer sets are also disabled automatically when *AP730_Close* is called.

**See Also**

AP730_AdcDisableBuffers
AP730_AdcSelectBuffers

## AP730_AdcDisableBuffers

Disables the specified ring buffer set

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcDisableBuffers (int Handle, DWORD BufferType);` |

**Parameters**
*Handle*
The handle to the board
*BufferType*
0: Raw ring buffers
1: Voltage ring buffers

**Return Value**
ERR_OK, ERR_BAD_PARAM, or ERR_INVALID_HNDL

**Comments**
This function disables and deallocates memory for the specified ring buffer set.

All buffer sets are also disabled automatically when *AP730_Close* is called.

**See Also**
AP730_AdcEnableBuffers
AP730_AdcSelectBuffers

## AP730_AdcSelectBuffers

Selects the ring buffer set used to receive data

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcSelectBuffers (int Handle, DWORD BufferType);` |

**Parameters**

*Handle*

The handle to the board

*BufferType*

0: Raw data buffer set
1: Voltage data buffer set

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_INVALID_HNDL or ERR_BUFFER

**Comments**

This function can be used to select which buffer set will be used by the interrupt service routine and the *AP730_AdcCopyFifoToRingBuffers* and *AP730_AdcCopyDmaBufferToRingBuffers* functions. The function will return ERR_BUFFER if the specified buffer set is not enabled.

**See Also**

AP730_AdcEnableBuffers
AP730_AdcDisableBuffers

## AP730_AdcGetBufferCount

Gets the number of values in the specified ring buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcGetBufferCount (int Handle,`<br>`DWORD BufferType,`<br>`DWORD Channel,`<br>`DWORD* Count);` |

**Parameters**

*Handle*

The handle to the board

*BufferType*

0: Raw data buffer set

1: Voltage data buffer set

*Channel*

The channel number (0 – 7)

*Count*

Receives the number of values in the ring buffer

**Return Value**

ERR_OK, ERR_BAD_PARAM, or ERR_INVALID_HNDL

**See Also**

`AP730_AdcEnableBuffers`

## AP730_AdcGetRawBufferValue

Gets a raw value from the specified channel's ring buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcGetRawBufferValue (int Handle, DWORD Channel, WORD* Value);` |

**Parameters**

*Handle*
The handle to the board

*Channel*
The channel number (0 – 7)

*Value*
Receives the value read from the buffer

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_BUFFER, ERR_EMPTY or ERR_INVALID_HNDL

**Comments**

Data is read from the ring buffer in the same order it was added. In other words, the function returns the oldest sample in the buffer.

The function will return *ERR_BUFFER* if the raw ring buffer set is not enabled. *ERR_EMPTY* is returned if the channel's raw ring buffer contains no data.

**See Also**

AP730_AdcEnableBuffers
AP730_AdcGetVoltBufferValue

## AP730_AdcGetVoltBufferValue

Gets a voltage value from the specified channel's ring buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcGetVoltBufferValue (int Handle, DWORD Channel, double* Value);` |

**Parameters**

*Handle*
The handle to the board

*Channel*
The channel number (0 – 7)

*Value*
Receives the value read from the buffer

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_BUFFER, ERR_EMPTY or ERR_INVALID_HNDL

**Comments**

Data is read from the ring buffer in the same order it was added. In other words, the function returns the oldest sample in the buffer.

The function will return *ERR_BUFFER* if the volt ring buffer set is not enabled. *ERR_EMPTY* is returned if the channel's volt ring buffer contains no data.

**See Also**

AP730_AdcEnableBuffers
AP730_AdcGetRawBufferValue

## AP730_AdcClearBuffer

Clears the specified ring buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcClearBuffer (int Handle,`<br>`DWORD BufferType,`<br>`DWORD Channel);` |

**Parameters**

*Handle*

The handle to the board

*BufferType*

0: Raw ring buffers

1: Voltage ring buffers

*Channel*

The channel number (0 – 7)

**Return Value**

ERR_OK, ERR_BAD_PARAM, or ERR_INVALID_HNDL

**See Also**

AP730_AdcClearBuffers

## AP730_AdcClearBuffers

Clears the specified ring buffer set

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcClearBuffers (int Handle,`<br>`DWORD BufferType);` |

**Parameters**

*Handle*

The handle to the board

*BufferType*

0: Raw ring buffers
1: Voltage ring buffers

**Return Value**

ERR_OK, ERR_BAD_PARAM, or ERR_INVALID_HNDL

**See Also**

`AP730_AdcClearBuffer`

## CALIBRATION FUNCTIONS

The AP730 board is delivered with factory calibration correction constants stored in on-board flash memory. During the call to *AP730_Open* the software reads these gain and offset correction values from flash and into internal variables. When a channel's range is selected, the corresponding correction values are written to the channel's ADC Offset and Gain registers.

The following functions can be used to view and update the calibration constants.

### AP730_AdcResetCalCorrections

Resets the offset and gain corrections for the specified ADC circuit.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcResetCalCorrections (int Handle, DWORD Channel);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

The channel number (0 – 7)

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function resets the internal calibration correction variables for the specified channel. Offset corrections are set to 0 and gain corrections are set to 10000 hex. The variables for all ranges are reset. The ADC Offset and Gain registers for the channel are reset as well. After calling this function, channel values written to the FIFO will be uncorrected.

**See Also**

`AP730_AdcCalcCalCorrections`

## AP730_AdcGetCalCorrections

Gets the calibration corrections for the specified ADC channel and range

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcGetCalCorrections (int Handle,`<br>`DWORD Channel,`<br>`DWORD Range,`<br>`WORD* OffsetCorr,`<br>`DWORD* GainCorr);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

The channel number (0 – 7)

*Range*

The range setting

| Value | Range/Enable |
|---|---|
| 1 | 0 Vdc to 5.12 Vdc |
| 2 | -5.0 Vdc to 5.0 Vdc |
| 3 | 5.12 Vdc to 5.12 Vdc |
| 4 | 0 Vdc to 10 Vdc |
| 5 | 0 Vdc to 10.24 Vdc |
| 6 | -10 Vdc to 10 Vdc |
| 7 | -10.24 Vdc to 10.24 Vdc |

*OffsetCorr*

Receives the offset correction

*GainCorr*

Receives the gain correction

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**See Also**

AP730_AdcCalcCalCorrections

## AP730_AdcSetCalCorrections

Sets the calibration corrections for the ADC channel and range to the specified values

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcSetCalCorrections (int Handle,`<br>`DWORD AdcNum,`<br>`DWORD Range,`<br>`WORD OffsetCorr,`<br>`DWORD GainCorr);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

The channel number (0 – 7)

*Range*

The range setting

| Value | Range/Enable |
|---|---|
| 1 | 0 Vdc to 5.12 Vdc |
| 2 | -5.0 Vdc to 5.0 Vdc |
| 3 | 5.12 Vdc to 5.12 Vdc |
| 4 | 0 Vdc to 10 Vdc |
| 5 | 0 Vdc to 10.24 Vdc |
| 6 | -10 Vdc to 10 Vdc |
| 7 | -10.24 Vdc to 10.24 Vdc |

*OffsetCorr*

The new offset correction

*GainCorr*

The new gain correction (0 to 3FFFF)

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function sets the internal calibration correction variables the specified ADC channel and range. The ADC Offset and Gain registers are also updated if the channel is currently using the specified range.

**See Also**

`AP730_AdcCalcCalCorrections`

## AP730_AdcGenerateCalCount

Generates an average count for a calibration voltage

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcGenerateCalCount (int Handle,`<br>`DWORD Channel,`<br>`DWORD Range,`<br>`DWORD NumSamples,`<br>`double* AvgCountCal);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

The channel number (0 – 7)

*Range*

The range setting

| Value | Range/Enable |
|---|---|
| 1 | 0 Vdc to 5.12 Vdc |
| 2 | -5.0 Vdc to 5.0 Vdc |
| 3 | 5.12 Vdc to 5.12 Vdc |
| 4 | 0 Vdc to 10 Vdc |
| 5 | 0 Vdc to 10.24 Vdc |
| 6 | -10 Vdc to 10 Vdc |
| 7 | -10.24 Vdc to 10.24 Vdc |

*NumSamples*

The number of samples to average (1 – 2052)

*AvgCountCal*

Receives the average count value

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function is used to generate the average *CountCALLO* and *CountCALHI* values described in the **ADC Use of Calibration Data** section of the **AP730 User's Manual**.

To use the function, connect a precisely measured calibration voltage (*VoltsCALLO* or *VoltsCALHI*) to the channel and call the function.

Recommended voltages:

| Range | *VoltsCALLO* | *VoltsCALHI* |
|---|---|---|
| 0 Vdc to 5.12 Vdc | 0.1V | +4.88V |
| -5.0 Vdc to 5.0 Vdc | -4.88V | +4.88V |
| 5.12 Vdc to 5.12 Vdc | -4.88V | +4.88V |
| 0 Vdc to 10 Vdc | 0.1V | +9.88V |
| 0 Vdc to 10.24 Vdc | 0.1V | +9.88V |
| -10 Vdc to 10 Vdc | -9.88V | +9.88V |
| -10.24 Vdc to 10.24 Vdc | -9.88V | +9.88V |

After generating the high and low counts, pass the counts and their corresponding voltages to *AP730_AdcCalcCalCorrections* to complete the calibration of the channel.

**See Also**
AP730_AdcCalcCalCorrections

## AP730_AdcCalcCalCorrections

Generates offset and gain correction values and saves them to the specified channel's internal calibration correction variables. ADC Offset and Gain registers are updated if necessary.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcCalcCalCorrections (int Handle,`<br>`DWORD Channel,`<br>`DWORD Range`<br>`double AvgCountCALLO,`<br>`double VoltsCalLo,`<br>`double AvgCountCALHI,`<br>`double VoltsCalHi);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

The channel number (0 – 7)

*Range*

The range setting

| Value | Range/Enable |
|---|---|
| 1 | 0 Vdc to 5.12 Vdc |
| 2 | -5.0 Vdc to 5.0 Vdc |
| 3 | 5.12 Vdc to 5.12 Vdc |
| 4 | 0 Vdc to 10 Vdc |
| 5 | 0 Vdc to 10.24 Vdc |
| 6 | -10 Vdc to 10 Vdc |
| 7 | -10.24 Vdc to 10.24 Vdc |

*AvgCountCALLO*

Average CAL LO count for the channel

*VoltsCalLo*

The measured voltage used for CAL LO

*AvgCountCALHI*

Average CAL HI count for the channel

*VoltsCalHi*

The measured voltage used for CAL HI

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function generates offset and gain correction values as described in the **ADC Use of Calibration Data** section of the **AP730 User's Manual**. Use *AP730_AdcGenerateCalCount* to generate the *AvgCountCALLO* and *AvgCountCALHI* values passed to this function.

The function will return ERR_BAD_PARAM if the passed counts and volts yield offset or gain corrections that are out of the ranges supported by the ADC Offset and Gain registers.

After calculating the offset and gain corrections the function passes the values to *AP730_AdcSetCalCorrections* to apply them. To save the changes to flash call *AP730_WriteCalDataToFlash*.

**See Also**

```
AP730_AdcGenerateCalCount
AP730_AdcSetCalCorrections
AP730_WriteCalDataToFlash
```

## AP730_AdcReadCalCorrectionsFromFlash

Reads the offset and gain correction values for the specified ADC circuit and range from flash memory

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcReadCalCorrectonsFromFlash (int Handle, DWORD Channel, DWORD Range, WORD* OffsetCorr, DWORD* GainCorr);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

The channel number (0 – 7)

*Range*

The range setting

| Value | Range/Enable |
|---|---|
| 1 | 0 Vdc to 5.12 Vdc |
| 2 | -5.0 Vdc to 5.0 Vdc |
| 3 | 5.12 Vdc to 5.12 Vdc |
| 4 | 0 Vdc to 10 Vdc |
| 5 | 0 Vdc to 10.24 Vdc |
| 6 | -10 Vdc to 10 Vdc |
| 7 | -10.24 Vdc to 10.24 Vdc |

*OffsetCorr*

Receives the offset correction

*GainCorr*

Receives the gain correction

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_CALIB or ERR_INVALID_HNDL

**Comments**

The flash sector containing the coefficient values should also contain an ASCII string (“AP730”) as an indicator that the calibration data is present. If this string is not found the function returns ERR_CALIB.

**See Also**

`AP730_AdcCalcCalCorrections`

## DATA ACQUISITION FUNCTIONS

### AP730_AdcStartConvert

Triggers conversions by setting bit 0 of the ADC Start Conversion register to logic one.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcStartConvert (int Handle);` |

**Parameters**
*Handle*
The handle to the board

**Return Value**
ERR_OK or ERR_INVALID_HNDL

**See Also**
AP730_AdcStopConvert

## AP730_AdcStopConvert

Stops the execution of conversions.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcStopConvert (int Handle);` |

**Parameters**
*Handle*
The handle to the board

**Return Value**
ERR_OK or ERR_INVALID_HNDL

**Comments**
This function stops conversions by clearing the conversion mode bits of the ADC Control register. Call *AP730_AdcSetConversionMode* to restore them.

**See Also**
AP730_AdcStartConvert
AP730_AdcSetConversionMode

## AP730_AdcReadFifoRaw

Reads a raw data value from the FIFO

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcReadFifoRaw (int Handle,`<br>`DWORD* Channel,`<br>`WORD* Data);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Receives the channel tag

*Data*

Receives the raw data

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function reads the ADC FIFO Channel Data and Tag register and splits the value into the channel number and data.

The function does not check that the FIFO contains data prior to performing the read.

**See Also**

AP730_AdcFifoIsEmpty
AP730_AdcCopyFifoToRingBuffers

## AP730_AdcReadFifoVolts

Reads a value from the FIFO and converts it to a voltage

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcReadFifoVolts (int Handle, DWORD* Channel, double* Data);` |

**Parameters**

*Handle*
The handle to the board

*Channel*
Receives the channel tag

*Data*
Receives the voltage

**Return Value**

ERR_OK, ERR_BAD_PARAM, or ERR_INVALID_HNDL

**Comments**

This function reads the ADC FIFO Channel Data and Tag register and splits the value into the channel number and voltage.

The function does not check that the FIFO contains data prior to performing the read.

**See Also**

AP730_AdcFifoIsEmpty
AP730_AdcCopyFifoToRingBuffers

## AP730_AdcCopyFifoToRingBuffers

Reads data out of the Sample FIFO and copies it to the currently selected ring buffer set

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcCopyFifoToRingBuffers (int Handle, DWORD Count, WORD* New, WORD* Missed);` |

**Parameters**

*Handle*
The handle to the board

*Count*
The maximum number of samples to copy (0 – 1026)

*New*
Receives new-data bit mask for channels 0-7

*Missed*
Receives discarded bit mask for channels 0-7

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_BUFFER, ERR_FULL or ERR_INVALID_HNDL

**Comments**

If both raw and voltage ring buffer sets have been enabled, the function will copy to the buffer set that was enabled first. The *AP730_AdcSelectBuffers* function can be used to alter this selection.

The function will return *ERR_BUFFER* if no ring buffer sets are enabled.

The function reads from the FIFO until either *Count* samples have been read or the FIFO becomes empty. If channel data is successfully copied to a buffer, the bit corresponding to the channel is set in the corresponding *New* bit mask. If channel data cannot be copied to a buffer because the buffer is full, the bit corresponding to the channel is set in the corresponding *Missed* mask. The function return code is set to *ERR_FULL* if any bits in *Missed* were set.

Note that the DLL's internal interrupt service routine calls this function unless Auto DMA Mode is in use or a callback has been assigned that overrides the ISR.

**See Also**

AP730_AdcEnableBuffers
AP730_AdcSelectBuffers
AP730_SetUserCallback

## AP730_AdcCopyDmaBufferToRingBuffers

Reads data out of the DMA buffer and copies it to the currently selected ring buffer set

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcCopyDmaBufferToRingBuffers (int Handle,`<br>`DWORD BuffOffset,`<br>`DWORD Count,`<br>`WORD* New,`<br>`WORD* Missed);` |

**Parameters**

*Handle*
The handle to the board

*BuffOffset*
The starting byte offset from the beginning of the DMA buffer

*Count*
The number of 32-bit samples to copy

*New*
Receives new-data bit mask for channels 0-7

*Missed*
Receives discarded bit mask for channels 0-7

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_BUFFER, ERR_FULL or ERR_INVALID_HNDL

**Comments**

If both raw and voltage ring buffer sets have been enabled, the function will copy to the buffer set that was enabled first. The *AP730_AdcSelectBuffers* function can be used to alter this selection.

The function will return *ERR_BUFFER* if no ring buffer sets are enabled.

The function will return *ERR_BAD_PARAM* if the values of *BuffOffset* and/or *Count* would cause a read outside of the boundaries of the DMA buffer.

If channel data is successfully copied to a buffer, the bit corresponding to the channel is set in the corresponding *New* bit mask. If channel data cannot be copied to a buffer because the buffer is full, the bit corresponding to the channel is set in the corresponding *Missed* mask. The function return code is set to *ERR_FULL* if any bits in *Missed* were set.

Note that the DLL's internal interrupt service routine calls this function if Auto DMA Mode is in use and a callback has not been assigned that overrides the ISR.

**See Also**

AP730_AdcEnableBuffers
AP730_AdcSelectBuffers
AP730_SetUserCallback

## DATA CONVERSION FUNCTIONS

### AP730_AdcRawToVolts

Converts a raw data value to a voltage.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_AdcRawToVolts (DWORD Range, WORD Raw, double* Volts);` |

**Parameters**

*Range*

The range setting

| Value | Range/Enable |
|---|---|
| 1 | 0 Vdc to 5.12 Vdc |
| 2 | -5.0 Vdc to 5.0 Vdc |
| 3 | 5.12 Vdc to 5.12 Vdc |
| 4 | 0 Vdc to 10 Vdc |
| 5 | 0 Vdc to 10.24 Vdc |
| 6 | -10 Vdc to 10 Vdc |
| 7 | -10.24 Vdc to 10.24 Vdc |

*Raw*

A 16-bit data value

*Volts*

Receives the voltage value

**Return Value**

ERR_OK or ERR_BAD_PARAM

**Comments**

This function can be used to calculate the voltage from a raw data value that was obtained using *AP730_AdcReadFifoRaw* or *AP730_AdcGetRawBufferValue*.

**See Also**

AP730_AdcReadFifoRaw
AP730_AdcGetRawBufferValue

# *Counter/Timer Functions*

## CONFIGURATION FUNCTIONS

AP730 Windows driver software provides two mechanisms for counter/timer configuration.

The first method is to use the *Set*, *Get* and *Configure* functions. The DLL stores an internal set of configuration variables for each counter timer. These variables hold the counter's constant A and B values as well as the counter's configuration values such as the counter's mode and debounce settings.

These variables exist until the board is closed. Variables are modified through calls to the corresponding *Set* functions. Calling *AP730_CtrReset* will also set all variables to zero. Each variable has a *Get* function that returns the current value of that variable.

The *Set* functions for control register values may be called in any order. In other words, a *Set* function does not test if a new value is compatible with the other variables for that counter. This testing occurs when *AP730_CtrConfigureControl* is called. The *AP730_CtrConfigureControl* function builds a 16-bit word from the counter's configuration variables. If the variables do not represent a valid configuration the function returns an error without modifying the hardware. Otherwise, the word is written to the corresponding counter control register. *AP730_CtrConfigureControl* also updates the Interrupt Enable Register.

**Usage Examples:**

Calling *AP730_CtrStop* clears the counter mode bits in the Counter Control Register. These bits can be restored to their previous value by calling *AP730_CtrConfigureControl*.

When using the Watchdog operation, the counter can be reloaded by calling *AP730_CtrConfigureConstants* with the *ConstA* parameter set to True.

To change the value of the Counter Control Register's clock source, call *AP730_CtrSetClockSource* to set the new value then call *AP730_CtrConfigureCtControl*.

The second method is to simply write directly to a counter's control and constant registers using the *AP730_CtrWriteControlReg*, *AP730_CtrWritetConstantA*, and *AP730_CtrWriteConstantB* functions. Please note that the *AP730_CtrWriteControlReg* function does not do any testing to ensure that the value written represents a valid configuration.

Which method you choose to use in your application is a matter of personal preference. It is recommended, however, not to mix the two methods. The *Write* functions do not modify the corresponding configuration variables so subsequent calls to the *Configure* functions may yield unexpected results.

## AP730_CtrReset

Resets counter/timer registers and variables.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrReset (int Handle);` |

**Parameters**
*Handle*
The handle to the board

**Comments**
This function also sets all counter timer Control and Constant configuration registers and variables to zero. Counter interrupts are disabled in the Interrupt Enable and Global Interrupt Enable registers.

**Return Value**
ERR_OK or ERR_INVALID_HNDL

## AP730_CtrGetMode

Gets the current value of the counter's mode configuration variable

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrGetMode (int Handle,`<br>`DWORD* Mode);` |

**Parameters**

*Handle*

The handle to the board

*Mode*

Receives the Counter mode variable value (0 - 7)

0: Disabled
1: Quadrature Position Measurement
2: Pulse Width Modulation
3: Watchdog Function
4: Event Counting or Frequency Measurement
5: Pulse-Width Measurement
6: Period Measurement
7: One-Shot Generation

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

## AP730_CtrSetMode

Sets the value of the counter's mode configuration variable

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrSetMode (int Handle,`<br>`DWORD Mode);` |

**Parameters**

*Handle*

The handle to the board

*Mode*

The new counter timer mode variable value (0 - 7)

0: Disabled
1: Quadrature Position Measurement
2: Pulse Width Modulation
3: Watchdog Function
4: Event Counting or Frequency Measurement
5: Pulse-Width Measurement
6: Period Measurement
7: One-Shot Generation

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

## AP730_CtrGetOutputPolarity

Gets the current value of the counter's output polarity variable

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrGetOutputPolarity (int Handle, BOOL* Polarity);` |

**Parameters**

*Handle*

The handle to the board

*Polarity*

Receives the Counter Output Polarity variable value
TRUE: Active HIGH
FALSE: Active LOW

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

## AP730_CtrSetOutputPolarity

Sets the value of the counter's output polarity variable

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrSetOutputPolarity (int Handle, BOOL Polarity);` |

**Parameters**

*Handle*

The handle to the board

*Polarity*

The new counter timer output polarity variable value
TRUE: Active HIGH
FALSE: Active LOW

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

## AP730_CtrGetInAPolarity

Gets the current value of the counter's input A polarity variable.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrGetInAPolarity (int Handle, DWORD* Polarity);` |

**Parameters**

*Handle*

The handle to the board

*Polarity*

Receives the counter timer input A polarity variable value (0 – 3).

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The function of the Input A variable is dependent on the selected Counter Mode. Consult the Counter Timer section of the **AP730 User's Manual** for detailed information on how this setting is used with your desired mode of operation.

## AP730_CtrSetInAPolarity

Sets the value of the counter's input A polarity variable.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrSetInAPolarity (int Handle, DWORD Polarity);` |

**Parameters**

*Handle*

The handle to the board

*Polarity*

The new counter timer input A polarity variable value (0 – 3).

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The function of the Input A variable is dependent on the selected Counter Mode. Consult the Counter Timer section of the **AP730 User's Manual** for detailed information on how this setting is used with your desired mode of operation.

## AP730_CtrGetInBPolarity

Gets the current value of the counter's input B polarity variable.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrGetInBPolarity (int Handle, DWORD* Polarity);` |

**Parameters**

*Handle*

The handle to the board

*Polarity*

Receives the counter timer input B polarity variable value (0 - 3).

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The function of the Input B variable is dependent on the selected Counter Mode. Consult the Counter Timer section of the **AP730 User's Manual** for detailed information on how this setting is used with your desired mode of operation.

## AP730_CtrSetInBPolarity

Sets the value of the counter's input B polarity variable

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrSetInBPolarity (int Handle, DWORD Polarity);` |

**Parameters**

*Handle*

The handle to the board

*Polarity*

The new counter timer input B polarity variable value (0 – 3).

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The function of the Input B variable is dependent on the selected Counter Mode. Consult the Counter Timer section of the **AP730 User's Manual** for detailed information on how this setting is used with your desired mode of operation.

## AP730_CtrGetInCPolarity

Gets the current value of the counter's input C polarity variable.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrGetInCPolarity (int Handle, DWORD* Polarity);` |

**Parameters**

*Handle*

The handle to the board

*Polarity*

Receives the counter timer input C polarity variable value (0 - 7).

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The function of the Input C variable is dependent on the selected Counter Mode. Consult the Counter Timer section of the **AP730 User's Manual** for detailed information on how this setting is used with your desired mode of operation.

## AP730_CtrSetInCPolarity

Sets the value of the counter's input C polarity variable.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrSetInCPolarity (int Handle, DWORD Polarity);` |

**Parameters**

*Handle*

The handle to the board obtained

*Polarity*

The new counter timer input C polarity variable value (0 – 7).

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The function of the Input C variable is dependent on the selected Counter Mode. Consult the Counter Timer section of the **AP730 User's Manual** for detailed information on how this setting is used with your desired mode of operation.

## AP730_CtrGetClockSource

Gets the current value of the counter's clock source/ special function variable

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrGetClockSource (int Handle, DWORD* Source);` |

**Parameters**

*Handle*

The handle to the board

*Source*

Receives the counter timer's clock source variable value (0 - 7).

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The meaning of this variable is dependent on the current counter timer mode. The variable typically represents the clock source/frequency. However, when the counter mode is 4, the variable specifies the Counter Mode as Event Counting or Frequency Measurement. Consult the Counter Timer section of the **AP730 User's Manual** for detailed information on how this setting is used with your desired mode of operation.

## AP730_CtrSetClockSource

Sets the value of the counter's clock source/ special function variable.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrSetClockSource (int Handle, DWORD Source);` |

Parameters

*Handle*

The handle to the board

*Source*

The new counter timer Source variable value (0 – 7).

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The meaning of this variable is dependent on the current counter timer mode. The variable typically represents the clock source/frequency. However, when the counter mode is 4, the variable specifies the Counter Mode as Event Counting or Frequency Measurement. Consult the Counter Timer section of the **AP730 User's Manual** for detailed information on how this setting is used with your desired mode of operation.

## AP730_CtrGetIntCondition

Gets the current value of the counter's Output and Interrupt Condition Select variable

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrGetIntCondition (int Handle, DWORD* Condition);` |

**Parameters**

*Handle*

The handle to the board

*Condition*

Receives the counter timer output and interrupt condition select variable value

0: Not Output or Interrupt Selected
1: Output and Interrupt on counter equal Constant A Register
2: Output and Interrupt on Index and reload on Index
3: Output and Interrupt on Index but do not reload counter on Index

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The Output and Interrupt Condition Select variable is only used with Quadrature Position Measurement Counter Mode. Consult the **Quadrature Position Measurement** section of the hardware **User's Manual** for detailed information on how this setting is used.

## AP730_CtrSetIntCondition

Sets the value of the counter's Output and Interrupt Condition Select variable

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrSetIntCondition (int Handle, DWORD Condition);` |

**Parameters**

*Handle*

The handle to the board

*Condition*

The new counter timer output and interrupt condition select variable value

0: Not Output or Interrupt Selected
1: Output and Interrupt on counter equal Constant A Register
2: Output and Interrupt on Index and reload on Index
3: Output and Interrupt on Index but do not reload counter on Index

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The Output and Interrupt Condition Select variable is only used with Quadrature Position Measurement Counter Mode. Consult the **Quadrature Position Measurement** section of the hardware **User's Manual** for detailed information on how this setting is used.

## AP730_CtrGetIntEnable

Gets the current value of the counter's interrupt enable variable

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrGetIntEnable (int Handle,`<br>`BOOL* IntEnable);` |

**Parameters**

*Handle*

The handle to the board

*IntEnable*

Receives the Counter interrupt enable variable value
TRUE: Enabled
FALSE: Disabled

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This variable corresponds to the counter's interrupt enable bit in the Global Interrupt Enable Register.

## AP730_CtrSetIntEnable

Sets the value of the counter's interrupt enable variable

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrSetIntEnable (int Handle,`<br>`BOOL IntEnable);` |

**Parameters**

*Handle*

The handle to the board

*IntEnable*

The new counter timer interrupt enable variable value
TRUE: Enabled
FALSE: Disabled

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This variable corresponds to the counter's interrupt enable bit in the Global Interrupt Enable Register.

## AP730_CtrConfigureControl

Configures the counter with the current values of the counter's control variables

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrConfigureControl (int Handle);` |

**Parameters**
*Handle*
The handle to the board

**Return Value**
ERR_OK, ERR_CONFIG_SET or ERR_INVALID_HNDL

**Comments**
This function builds a 16-bit word from the configuration variables and writes it to the counter control register. The Global Interrupt Enable Register is also updated. If the variables do not represent a valid configuration the function returns ERR_CONFIG_SET without modifying the hardware.

Prior to writing a valid control register value, the function disables the counter and clears any interrupts for the counter in the Global Interrupt Status Register and Interrupt Status Register.

## AP730_CtrGetConstantA

Gets the current value of the counter's Constant A variable

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrGetConstantA (int Handle,`<br>`BOOL Reg2,`<br>`DWORD* Constant);` |

**Parameters**

*Handle*

The handle to the board

*Reg2*

Specifies the Constant A register

TRUE: Counter Constant A Register 2

FALSE: Counter Constant A Register 1

*Constant*

Receives the value of the counter constant A variable.

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

## AP730_CtrSetConstantA

Sets the value of the counter's Constant A variable

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrSetConstantA (int Handle,`<br>`BOOL Reg2,`<br>`DWORD Constant);` |

**Parameters**

*Handle*

The handle to the board

*Reg2*

Specifies the Constant A register

TRUE: Counter Constant A Register 2

FALSE: Counter Constant A Register 1

*Constant*

The new 32-bit value of the Constant A variable

**Return Value**

ERR_OK or ERR_INVALID_HNDL

## AP730_CtrGetConstantB

Gets the current value of the counter's Constant B variable

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrGetConstantB (int Handle,`<br>`BOOL Reg2,`<br>`DWORD* Constant);` |

**Parameters**

*Handle*

The handle to the board

*Reg2*

Specifies the Constant B register

TRUE: Counter Constant B Register 2

FALSE: Counter Constant B Register 1

*Constant*

Receives the value of the counter constant B variable

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

## AP730_CtrSetConstantB

Sets the value of the counter's Constant B variable

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrSetConstantB (int Handle,`<br>`BOOL Reg2,`<br>`DWORD Constant);` |

**Parameters**

*Handle*

The handle to the board

*Reg2*

Specifies the Constant B register

TRUE: Counter Constant B Register 2

FALSE: Counter Constant B Register 1

*Constant*

The new 32-bit value of the Constant B variable.

**Return Value**

ERR_OK or ERR_INVALID_HNDL

## AP730_CtrConfigureConstants

Configures the counter with the current value of the counter's constant A variable and/or constant B variable.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrConfigureConstants (int Handle,`<br>`BOOL ConstA,`<br>`BOOL ConstB,`<br>`BOOL Reg2);` |

**Parameters**

*Handle*

The handle to the board

*ConstA*

If TRUE, the constant A variable is written

*ConstB*

If TRUE, the constant B variable is written

*Reg2*

If TRUE, the Counter Constant A Register 2 and/or Counter Constant B Register 2 values are written

FALSE, the Counter Constant A Register 1 and/or Counter Constant B Register 1 values are written

**Return Value**

ERR_OK or ERR_INVALID_HNDL

## AP730_CtrReadControlReg

Reads the value of the Counter Control Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrReadCtControlReg (int Handle,`<br>`WORD* Data);` |

**Parameters**

*Handle*

The handle to the board

*Data*

Receives the 16-bit value of the register

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function may be used to check the counter/timer's configuration.

## AP730_CtrWriteControlReg

Writes a value to the Counter Control Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrWriteControlReg (int Handle, WORD Data);` |

**Parameters**

*Handle*

The handle to the board

*Data*

The 16-bit value to be written to the register

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

The function does not test that the value written represents a valid configuration.

## AP730_CtrReadIntEnable

Returns the state of the Counter Interrupt Enable bit in Global Interrupt Enable register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_CtrReadCtIntEnable (int Handle, BOOL* Enable);` |

**Parameters**

*Handle*

The handle to the board

*Enable*

Receives the state of Counter Interrupt Enable bit

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

For a counter interrupt to reach the interrupt output, it must also be enabled in the Interrupt Enable Register.

**See Also**

`AP730_ReadIntEnable`

## AP730_CtrWriteIntEnable

Sets the state of the Counter Interrupt Enable bit in Global Interrupt Enable register

| Syntax | |
|---|---|
| C: | `int _stdcall AP730_CtrWriteIntEnable (int Handle, BOOL Enable);` |

**Parameters**

*Handle*

The handle to the board

*Enable*

The new state of the Counter Interrupt Enable bit

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

For a counter interrupt to reach the interrupt output, it must also be enabled in the Interrupt Enable Register.

**See Also**

`AP730_WriteIntEnable`

## AP730_CtrReadConstantA

Reads the value of the Constant A Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrReadConstantA (int Handle,`<br>`BOOL Reg2,`<br>`DWORD* Constant);` |

**Parameters**

*Handle*

The handle to the board

*Reg2*

Specifies the Constant A register
TRUE: Counter Constant A Register 2
FALSE: Counter Constant A Register 1

*Constant*

Receives the 32-bit value of the Constant A register

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

## AP730_CtrWriteConstantA

Writes a value to the Constant A Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrWriteConstantA (int Handle,`<br>`BOOL Reg2,`<br>`DWORD Constant);` |

**Parameters**

*Handle*

The handle to the board

*Reg2*

Specifies the Constant A register

TRUE: Counter Constant A Register 2

FALSE: Counter Constant A Register 1

*Constant*

The 32-bit value to be written to the register

**Return Value**

ERR_OK or ERR_INVALID_HNDL

## AP730_CtrReadConstantB

Reads the value of the Constant B Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrReadConstantB (int Handle,`<br>`BOOL Reg2,`<br>`DWORD* Constant);` |

**Parameters**

*Handle*

The handle to the board

*Reg2*

Specifies the Constant A register
TRUE: Counter Constant B Register 2
FALSE: Counter Constant B Register 1

*Constant*

Receives the 32-bit value of the Constant B register

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

## AP730_CtrWriteConstantB

Writes a value to the Constant B Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrWriteConstantB (int Handle,`<br>`BOOL Reg2,`<br>`DWORD Constant);` |

**Parameters**

*Handle*

The handle to the board

*Reg2*

Specifies the Constant B register

TRUE: Counter Constant B Register 2

FALSE: Counter Constant B Register 1

*Constant*

The 32-bit value to be written to the register

**Return Value**

ERR_OK or ERR_INVALID_HNDL

## AP730_CtrReadCounter

Reads the value of Counter Read Back Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrReadCounter (int Handle,`<br>`DWORD* CounterVal);` |

**Parameters**

*Handle*

The handle to the board.

*CounterVal*

Receives the value of the register

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function writes a “1” to bit 8 in the Counter Trigger, Stop, Load Read Back, and Toggle Register and then it reads the current value held in the counter from the Counter Read Back Register.

## COUNTER/TIMER START AND STOP FUNCTIONS

### AP730_CtrStart

Performs a software trigger

| Syntax | |
|---|---|
| C: | `int _stdcall AP730_CtrStart (int Handle);` |

**Parameters**
*Handle*
The handle to the board

**Return Value**
ERR_OK or ERR_INVALID_HNDL

## AP730_CtrStop

Stops the specified counter.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrStop (int Handle);` |

**Parameters**
*Handle*
The handle to the board

**Return Value**
ERR_OK or ERR_INVALID_HNDL

**Comments**
This function writes to bit-16 in the Counter Trigger, Stop, Load Read Back, and Toggle register. Note that this action also clears bits 0, 1 and 2 in the Counter Control Register.

## TOGGLE COUNTER CONSTANT FUNCTIONS

### AP730_CtrReadToggleConstant

Reads the counter's Toggle Counter Constant setting

| Syntax | |
|---|---|
| C: | `int _stdcall AP730_CtrReadToggleConstant (int Handle, BOOL* Reg2);` |

**Parameters**

*Handle*

The handle to the board

*Reg2*

Receives the channel's Toggle Counter Constant setting

TRUE: Using Counter Constant A Register 2 and Counter Constant B Register 2

FALSE: Using Counter Constant A Register 1 and Counter Constant B Register 1

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

## AP730_CtrWriteToggleConstant

Writes the counter's Toggle Counter Constant setting.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CtrWriteToggleConstant (int Handle, BOOL Reg2);` |

**Parameters**

*Handle*

The handle to the board

*Reg2*

The new Toggle Counter Constant setting for the channel
TRUE: Use Counter Constant A Register 2 and Counter Constant B Register 2
FALSE: Use Counter Constant A Register 1 and Counter Constant B Register 1

**Return Value**

ERR_OK or ERR_INVALID_HNDL

## COUNTER/TIMER STATUS FUNCTIONS

### AP730_CtrReadIntInfoReg

Reads the counter's Interrupt Information Register

| Syntax | |
|---|---|
| C: | `int _stdcall AP730_CtrReadIntInfoReg (int Handle, WORD* Value);` |

**Parameters**

*Handle*

The handle to the board

*Value*

Receives the 16-bit value read from the register

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The Interrupt Information Register provides information on what type of interrupt has been generated.

| BIT | FUNCTION |
|---|---|
| 0 | Counter equal to value in Counter Constant A Register (available in Event Counting and Quadrature Position Measurement) |
| 1 | Read Back Register has been loaded by hardware (available in Event Counting and Quadrature Position Measurement) |
| 2 | Index Pulse (available in Quadrature Position Measurement) |
| 3 | High to Low or Low to High Transitions (available in Pulse Width Modulation or One-Shot Pulse Mode) |
| 4 | End of Measurement (available in Frequency Measurement, Input Period Measurement, or Input Pulse Width Measurement) |
| 5 | Time Out (available in Watchdog Timer) |
| 6-31 | Not Used |

**See Also**

`AP730_ReadIntStatus`
`AP730_CtrDioReadIntStatus`

# *DAC Functions*

## CALLING SEQUENCE EXAMPLES

The following are some typical calling sequences for the AP730 DAC functions. These sequences would be performed after opening the board with *AP730_Open* and before closing the board with *AP730_Close*.

### Direct Access Mode

| Function | Comment |
|---|---|
| AP730_DacSetOperatingMode | Set Operating Mode to Direct |
| AP730_DacSetOutputRange | Set the desired output range |
| AP730_DacSetTriggerSource | Set Trigger Source to Software |
| AP730_DacSetVout | Writes value to channel's input register and optionally updates the DAC output voltage |
| AP730_DacUpdateDAC or AP730_DacSoftwareTrigger | Updates the DAC output voltage. This is only necessary if it was not done by AP730_DacSetVout. |

### Continuous or Single Burst Modes

| Function | Comment |
|---|---|
| AP730_DacSetOperatingMode | Set Operating Mode to Continuous or Single Burst |
| AP730_DacSetOutputRange | Set the desired output range |
| AP730_DacSetTriggerSource | Set Trigger Source to Timer |
| AP730_DacSetMemoryAddresses | Sets the portion of Sample Memory used by the channel |
| AP730_DacWriteWaveBufferVolts | Takes a buffer of voltage values, converts them to corrected codes and writes them to the channel's memory |
| AP730_DacSetWaveformOutputEnable | Enable waveform output |

### FIFO Mode, Interrupt Driven, No DMA

Assume we have two data buffers. As the software is feeding the data from one buffer to the FIFO, the other buffer is being prepared. After the data in the first buffer is consumed the software switches to the other buffer and the process repeats.

| Function | Comment |
|---|---|
| AP730_DacSetOperatingMode | Set Operating Mode to FIFO |
| AP730_DacSetOutputRange | Set the desired output range |
| AP730_DacSetTriggerSource | Set Trigger Source to Timer |
| AP730_DacSetMemoryAddresses | Sets the portion of Sample Memory used by the channel |
| AP730_DacVoltsToCodes | Takes a buffer of voltage values and converts them to corrected codes |
| AP730_DacAttachFifoBuffer | Attach code buffer that will be used to load the FIFO |
| AP730_DacSetFifoLoading | Since we will be responding to FIFO-half-full interrupts, set Count to half the FIFO size and Poll to FALSE. Wrap is set to FALSE. |

| | |
|---|---|
| AP730_DacLoadFifo | Half fills the FIFO |
| AP730_SetUserCallback | Set Replace to FALSE so that the callback will execute after the DLL's internal ISR copies data to the FIFO |
| AP730_SetIntEnable | Enable interrupts for the channel |
| **Within callback function** | |
| AP730_DacGetBufferNext | If BufferNext is 0, it indicates all the data in the attached buffer has been copied to the FIFO |
| AP730_DacAttachFifoBuffer | Attach buffer with new data |

## FIFO Mode, Interrupt Driven, Using DMA

Assume the DMA buffer is partitioned into multiple segments. Each segment contains enough data to half-fill the FIFO. When a FIFO-half-full interrupt occurs the interrupt service routine uses DMA to copy one segment to the FIFO. The software rotates through the segments as interrupts occur. Note that this approach requires implementing a callback function that is used instead of the DLL's internal ISR.

| **Function** | **Comment** |
|---|---|
| AP730_DacSetOperatingMode | Set Operating Mode to FIFO |
| AP730_DacSetOutputRange | Set the desired output range |
| AP730_DacSetTriggerSource | Set Trigger Source to Timer |
| AP730_DacSetMemoryAddresses | Sets the portion of Sample Memory used by the channel |
| AP730_DacVoltsToCodes | Takes a buffer of voltage values and converts them to corrected codes which are written to the DMA buffer |
| AP730_SetUserCallback | Set Replace to TRUE so that the callback will execute in place of the DLL's internal ISR. |
| AP730_SetIntEnable | Enable interrupts for the channel |
| **Within callback function** | |
| LoadDacFifoUsingDma | Executes the DMA transfer and waits for it to complete. See the AP730DacDemo.c file for an example implementation of this function. |
| AP730_WriteIntAck | Acknowledge interrupts |
| AP730_SetIntEnable | Re-enable interrupts |

## GLOBAL CONFIGURATION FUNCTIONS

### AP730_DacReset

Resets all DAC channels in the specified way

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DacReset (int Handle,<br>DWORD ResetType); |

**Parameters**

*Handle*

The handle to the board

*ResetType*

The type of reset to perform (1 – 4)

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_INVALID_HNDL or ERR_TIMEOUT

**Comments**

When *ResetType* equals 2, bit-1 is set in the DAC Control Register. This returns all DACs to their default power-on status where the output is clamped to ground and the output buffer is powered down. Each channel control register is then set to 0x208. This sets the output range to -10V to +10V, and sets the Power-up and Clear voltage selections to midscale.

When *ResetType* equals 3, bit-2 is set. This sets each DAC register to zero-scale, midscale, or full scale (as specified by the channel's Clear Voltage setting).

When *ResetType* equals 4, bit-4 is set. This resets the registers in the AD5721 interface. The DAC control register is also cleared

*ResetType* 1 is a "full DAC reset." It performs *ResetTypes* 2 and 4 and also disables DAC interrupts.

**See Also**

AP730_DacSetClearVoltage
AP730_DacChannelReset

## AP730_DacSetTrigOutputEnable

Sets the trigger direction

| Syntax | |
|---|---|
| C: | `int _stdcall AP730_DacSetTrigOutputEnable (int Handle, BOOL Enable);` |

**Parameters**

*Handle*

The handle to the board

*Direction*

The new trigger output enable setting
TRUE: Outputs trigger to external devices
FALSE: Output trigger disabled

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

This function sets or clears bit-3 in the Control register.

**See Also**

`AP730_DacGetTrigOutputEnable`

## AP730_DacGetTrigOutputEnable

Gets the trigger output enable setting

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetTrigOutputEnable (int Handle, BOOL* Enable);` |

**Parameters**

*Handle*

The handle to the board

*Direction*

Receives the trigger output enable setting
TRUE: Outputs trigger to external devices
FALSE: Output trigger disabled

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function reads bit-3 in the Control register.

**See Also**

`AP730_DacSetTrigOutputEnable`

## AP730_DacSetTimerDivider

Sets the timer divider value

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DacSetTimerDivider (int Handle,<br>DWORD TimerDivider); |

**Parameters**

*Handle*

The handle to the board.

*TimerDivider*

The new timer divider value (136 – FFFFFFFF hex)

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function writes to the Timer Divider register. This 32-bit register controls the period of the internal trigger. The timer is clocked at a rate of 31.25 MHz. The period of the internal trigger is calculated by multiplying the Timer Divider register contents by 32 nano-seconds. The minimum recommended value is 136 hex. With this value the frequency of trigger is 100 KHz, which is the maximum frequency the DACs can operate at and still settle to within 1 LSB accuracy.

**See Also**

AP730_DacGetTimerDivider

## AP730_DacGetTimerDivider

Gets the timer divider value

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DacGetTimerDivider (int Handle,<br>DWORD* TimerDivider); |

**Parameters**

*Handle*

The handle to the board

*TimerDivider*

Receives the timer divider value

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function reads from the Timer Divider register. This 32-bit register controls the period of the internal trigger. The timer is clocked at a rate of 31.25 MHz. The period of the internal trigger is calculated by multiplying the Timer Divider register contents by 32 nano-seconds. The minimum recommended value is 136 hex. With this value the frequency of trigger is 100 KHz, which is the maximum frequency the DACs can operate at and still settle to within 1 LSB accuracy.

**See Also**

AP730_DacSetTimerDivider

## CHANNEL CONFIGURATION FUNCTIONS

### AP730_DacChannelReset

Performs a data reset on the specified channel

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacChannelReset (int Handle,`<br>`DWORD Channel,`<br>`DWORD ResetType);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*ResetType*

The type of reset to perform (1 – 4)

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

When *ResetType* equals 2, the function performs a clear channel by setting bit 4 in the channel's control register. This clears the Start and End Address registers and resets channels internal read/write pointers.

When *ResetType* equals 3, the function performs a DAC reset by writing 0xF to the channel's Direct Access register. This returns the DAC to its default power-on status where the output is clamped to ground and the output buffer is powered down. The channel's Direct Access register is then set to 0x208. This sets the output range to -10V to +10V, and sets the Power-up and Clear voltage selections to midscale.

When *ResetType* equals 4, the function performs a data reset by writing 7 to the channel's Direct Access register. This sets each DAC register to zero-scale, midscale, or full scale (as specified by the channel's Power-up Voltage setting).

*ResetType* 1 is a full channel reset. It performs *ResetTypes* 2 and 3, the channel's control register is cleared (direct mode, software triggered), any set Underflow or Burst Complete status bits are cleared, and channel interrupts are disabled.

**See Also**

AP730_DacReset

## AP730_DacSetOperatingMode

Sets the specified channel's operating mode

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacSetOperatingMode (int Handle,`<br>`DWORD Channel,`<br>`DWORD OpMode);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*OpMode*

The new operating mode

| Value | Mode |
|---|---|
| 0 | Direct |
| 1 | Continuous |
| 2 | FIFO |
| 3 | Single Burst |

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function sets bits 0 - 1 in the Channel's Control Register.

**See Also**

`AP730_DacGetOperatingMode`

## AP730_DacGetOperatingMode

Gets the specified channel's operating mode

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetOperatingMode (int Handle,`<br>`DWORD Channel,`<br>`DWORD* OpMode);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*OpMode*

Receives the operating mode

| Value | Mode |
|---|---|
| 0 | Direct |
| 1 | Continuous |
| 2 | FIFO |
| 3 | Single Burst |

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function reads bits 0 - 1 in the Channel's Control Register.

**See Also**

`AP730_DacSetOperatingMode`

## AP730_DacSetMemoryAddresses

Sets the specified channel's Sample Memory address range

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacSetMemoryAddresses (int Handle, DWORD Channel, WORD StartAddr, WORD EndAddr);` |

**Parameters**

*Handle*
The handle to the board

*Channel*
Specifies the channel (0 – 3)

*StartAddr*
The Sample Memory start address (0 – FFFF)

*EndAddr*
The Sample Memory end address (*StartAddr* – FFFF)

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function writes to the channel's Start and End Address registers. For channels configured in Single Burst mode, Continuous mode, or FIFO mode, these registers contain the start and end addresses in sample memory of the waveform to be written to the D/A converter. Addressing is relative to the first location in sample memory.

**See Also**

`AP730_DacGetMemoryAddresses`

## AP730_DacGetMemoryAddresses

Gets the specified channel's Sample Memory address range

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DacGetMemoryAddresses (int Handle,<br>DWORD Channel,<br>WORD* StartAddr,<br>WORD* EndAddr); |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*StartAddr*

Receives the Sample Memory start address

*EndAddr*

Receives the Sample Memory end address

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function reads from the channel's Start and End Address registers. For channels configured in Single Burst mode, Continuous mode, or FIFO mode, these registers contain the start and end addresses in sample memory of the waveform to be written to the D/A converter. Addressing is relative to the first location in sample memory.

**See Also**

AP730_DacSetMemoryAddresses

## AP730_DacSetTriggerSource

Sets the specified channel's trigger source

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DacSetTriggerSource (int Handle,<br>DWORD Channel,<br>DWORD TriggerSrc); |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*TriggerSrc*

The new trigger source

| Value | Source |
|---|---|
| 0 | Software |
| 1 | Timer |
| 2 | External |

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function sets bits 2-3 in the Channel's Control Register.

**See Also**

AP730_DacGetTriggerSource

## AP730_DacGetTriggerSource

Gets the specified channel's trigger source

| Syntax | |
|---|---|
| C: | `int _stdcall AP730_DacGetTriggerSource (int Handle,`<br>`DWORD Channel,`<br>`DWORD* TriggerSrc);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*TriggerSrc*

Receives the trigger source

| Value | Source |
|---|---|
| 0 | Software |
| 1 | Timer |
| 2 | External |

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function reads bits 2-3 in the Channel's Control Register.

**See Also**

`AP730_DacSetTriggerSource`

## AP730_DacSetOutputRange

Sets the specified channel's output range

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacSetOutputRange (int Handle,`<br>`DWORD Channel,`<br>`DWORD OutputRange);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*OutputRange*

The new output range value

| Value | Range |
|---|---|
| 0 | -10 to 10 |
| 1 | 0 to 10 |
| 2 | -5 to 5 |
| 3 | 0 to 5 |
| 4 | -2.5 to 7.5 |
| 5 | -3 to 3 |

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function sets bits 0 - 2 of the channel's Direct Access Register. *AP730_Open* initializes the output range for all channels to ± 10 volts.

**See Also**

AP730_DacGetOutputRange
AP730_Open

## AP730_DacGetOutputRange

Gets the specified channel's output range setting

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetOutputRange (int Handle,`<br>`DWORD Channel,`<br>`DWORD* OutputRange);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*OutputRange*

Receives the output range configuration value

| Value | Range |
|---|---|
| 0 | -10 to 10 |
| 1 | 0 to 10 |
| 2 | -5 to 5 |
| 3 | 0 to 5 |
| 4 | -2.5 to 7.5 |
| 5 | -3 to 3 |

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function reads bits 0 - 2 of the channel's Direct Access Register.

**See Also**

`AP730_DacSetOutputRange`

## AP730_DacSetPowerUpVoltage

Sets the specified channel's power-up voltage

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacSetPowerUpVoltage (int Handle,`<br>`DWORD Channel,`<br>`DWORD PowerUpVoltage);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*PowerUpVoltage*

The new power-up voltage value

| Value | Voltage |
|---|---|
| 0 | Zero scale |
| 1 | Midscale |
| 2 | Full scale |
| 3 | Full scale |

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function sets bits 3 and 4 of the channel's Direct Access Register.

## AP730_DacGetPowerUpVoltage

Gets the specified channel's power-up voltage setting

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetPowerUpVoltage (int Handle, DWORD Channel, DWORD* PowerUpVoltage);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*PowerUpVoltage*

Receives the power-up voltage value

| Value | Voltage |
|---|---|
| 0 | Zero scale |
| 1 | Midscale |
| 2 | Full scale |
| 3 | Full scale |

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function reads bits 3 and 4 of the channel's Direct Access Register.

## AP730_DacSetThermalShutdown

Enables/disables the DAC channel's thermal shutdown alert

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacSetThermalShutdown (int Handle,`<br>`DWORD Channel,`<br>`BOOL PowerDown);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*PowerDown*

The new thermal shutdown setting
TRUE: Power down if die temperature > 150°C
FALSE: Do not power down if die temperature > 150°C

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function sets bit 6 of the channel's Direct Access Register.

## AP730_DacGetThermalShutdown

Gets the DAC channel's thermal shutdown alert setting

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetThermalShutdown (int Handle,`<br>`DWORD Channel,`<br>`BOOL* PowerDown);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*PowerDown*

Receives the thermal shutdown setting
TRUE: Power down if die temperature > 150°C
FALSE: Do not power down if die temperature > 150°C

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function reads bit 6 of the channel's Direct Access Register.

## AP730_DacSetDataFormat

Sets the specified channel's data format

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacSetDataFormat (int Handle,`<br>`DWORD Channel,`<br>`DWORD Format);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*Format*

The new data format

0: Straight binary

1: Two's complement

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function sets bit 7 of the channel's Direct Access Register.

When a channel is configured with a unipolar output range, this setting is ignored and anything written to the DAC is treated as straight binary.

## AP730_DacGetDataFormat

Gets the specified channel's data format

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetDataFormat (int Handle,`<br>`DWORD Channel,`<br>`DWORD* Format);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

Format

Receives the data format
0: Straight binary
1: Two's complement

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function reads bit 7 of the channel's Direct Access Register.

When a channel is configured with a unipolar output range, this setting is ignored and anything written to the DAC is treated as straight binary.

## AP730_DacSetOverRange

Enables/disables the DAC channel's 5% over-range setting

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacSetOverRange (int Handle,`<br>`DWORD Channel,`<br>`BOOL OverRange);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*OverRange*

The new 5% over-range setting
TRUE: 5% over-range enable
FALSE: 5% over-range disable

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function sets bit 8 of the channel's Direct Access Register.

## AP730_DacGetOverRange

Gets the DAC channel's 5% over-range setting

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetOverRange (int Handle,`<br>`DWORD Channel,`<br>`BOOL* OverRange);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*OverRange*

Receives the 5% over-range setting
TRUE: 5% over-range enable
FALSE: 5% over-range disable

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function reads bit 8 of the channel's Direct Access Register.

## AP730_DacSetClearVoltage

Sets the specified channel's clear voltage

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacSetClearVoltage (int Handle,`<br>`DWORD Channel,`<br>`DWORD ClearVoltage);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*ClearVoltage*

The new clear voltage value

| Value | Voltage |
|---|---|
| 0 | Zero scale |
| 1 | Midscale |
| 2 | Full scale |
| 3 | Full scale |

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function sets bits 9 and 10 of the channel's Direct Access Register.

## AP730_DacGetClearVoltage

Gets the specified channel's clear voltage setting

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetClearVoltage (int Handle, DWORD Channel, DWORD* ClearVoltage);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*ClearVoltage*

Receives the clear voltage value

| Value | Voltage |
|---|---|
| 0 | Zero scale |
| 1 | Midscale |
| 2 | Full scale |
| 3 | Full scale |

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function reads bits 9 and 10 of the channel's Direct Access Register.

## AP730_DacWriteChanControl

Writes a value to the specified channel's Control register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacWriteChanControl (int Handle,`<br>`DWORD Channel,`<br>`WORD Value);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*Value*

The new channel Control register value

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function is a low-level alternative to the "Set" DAC channel configuration functions. The register bits are described in detail in the **AP730 User's Manual**.

**See Also**

`AP730_DacReadChanControl`

## AP730_DacGetChanControl

Gets the value of the specified channel's Control register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetChanControl (int Handle,`<br>`DWORD Channel,`<br>`WORD* Value);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*Value*

Receives the channel's Control register value

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function is a low-level alternative to the "Get" DAC channel configuration and status functions. The register bits are described in detail in the **AP730 User's Manual**.

**See Also**

`AP730_DacWriteChanControl`

## CHANNEL STATUS FUNCTIONS

### AP730_DacGetChannelStatus

Reads the specified channel's Status register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetChannelStatus (int Handle,`<br>`DWORD Channel,`<br>`DWORD Status);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*Status*

Receives the value of the channel's Status register

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The register bits are described in detail in the **AP730 User's Manual**.

## AP730_DacClearChannelStatus

Clears the specified status bits in the channel's Status register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacClearChannelStatus (int Handle,`<br>`DWORD Channel,`<br>`BOOL Underflow,`<br>`BOOL BurstComplete);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*Underflow*

Set to TRUE to clear a set Underflow bit

*BurstComplete*

Set to TRUE to clear a set Burst Complete bit

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function writes bits 3 and/or 4 of the channel's Status Register.

## DATA OUTPUT FUNCTIONS

### AP730_DacSoftwareTrigger

Writes the Software Trigger register

| Syntax | |
|---|---|
| C: | `int _stdcall AP730_DacSoftwareTrigger (int Handle);` |

**Parameters**
*Handle*
The handle to the board

**Return Value**
ERR_OK or ERR_INVALID_HNDL

**Comments**
This function triggers the DAC outputs for all channels configured to respond to a software trigger.

**See Also**
`AP730_DacSetTriggerSource`

## AP730_DacSetWaveformOutputEnable

Starts or stops waveform output

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacSetWaveformOutputEnable (int Handle, BOOL Enable);` |

**Parameters**

*Handle*

The handle to the board

The new state of the Waveform Output Enable bit

TRUE: Enabled

FALSE: Disabled

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

This function sets or clears bit-0 in the Control Register

**See Also**

`AP730_DacGetWaveformOutputEnable`

## AP730_DacGetWaveformOutputEnable

Gets the waveform output enable setting

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetWaveformOutputEnable (int Handle, BOOL* Enable);` |

**Parameters**
*Handle*
The handle to the board
Receives the state of the Waveform Output Enable bit
TRUE: Enabled
FALSE: Disabled

**Return Value**
ERR_OK or ERR_INVALID_HNDL

**Comments**
This function reads bit-0 in the Control Register

**See Also**
`AP730_DacSetWaveformOutputEnable`

## DIRECT ACCESS FUNCTIONS

### AP730_DacSetVout

Converts the voltage argument to a corrected code and writes the code to the specified channel's Direct Access Register.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacSetVout (int Handle,`<br>`DWORD Channel,`<br>`BOOL UpdateDAC,`<br>`double Volts);` |

**Parameters**

*Handle*
The handle to the board

*Channel*
Specifies the channel (0 – 3)

*UpdateDAC*
Set to TRUE to update the DAC immediately

*Volts*
The desired DAC channel voltage value

**Return Value**
ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**
The channel's configured range determines what voltage values the function will accept. For example, if the range is currently set to ±5 volts, the function will return ERR_BAD_PARAM if an attempt is made to set the voltage to 7 volts. The function will also clip a voltage value at the upper limit of the range to an ideal value (e.g., if the range is currently set to ±5 volts, a 5 volt argument will be clipped to the ideal value, 4.999847).

This function converts the voltage argument to a corrected code using the channel's output range setting, and its offset and gain calibration coefficient variables.

If the *UpdateDAC* parameter set to FALSE, the corrected count is written to the Input Shift register but not the DAC register. The data will not show up at the output until *AP730_DacUpdateDAC* is called or a trigger occurs.

If the *UpdateDAC* parameter set to TRUE, the corrected count is written to the Input Shift register and the DAC register is updated automatically.

**See Also**
AP730_DacUpdateDAC

## AP730_DacWriteInput

Writes a 16-bit value directly to the specified channel's Direct Access Register.

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DacWriteInput (int Handle,<br>DWORD Channel,<br>BOOL UpdateDAC,<br>WORD Value); |

**Parameters**

*Handle*
The handle to the board

*Channel*
Specifies the channel (0 – 3)

*UpdateDAC*
Set to TRUE to update the DAC immediately

*Value*
The new DAC channel value

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

This function is provided as a low-level alternative to *AP730_DacSetVout*. The function writes a 16-bit value "as is" to the specified channel. The value is not adjusted using the board's calibration coefficients.

When a channel is configured for unipolar output ranges, *Value* should be in straight binary format. For bipolar ranges, *Value* should be in the format corresponding to the channel's configured data format.

If the *UpdateDAC* parameter set to FALSE, the corrected count is written to the Input Shift register but not the DAC register. The data will not show up at the output until *AP730_DacUpdateDAC* is called or a trigger occurs.

If the *UpdateDAC* parameter set to TRUE, the corrected count is written to the Input Shift register and the DAC register is updated automatically.

**See Also**

AP730_DacSetDataFormat
AP730_DacUpdateDAC

## AP730_DacUpdateDAC

Updates the specified DAC channel

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacUpdateDAC (int Handle, DWORD Channel);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_TIMEOUT or ERR_INVALID_HNDL

**Comments**

After calling *AP730_DacSetVout* or *AP730_DacWriteInput* with *UpdateDAC set to* FALSE, call this function to transfer the channel's digital data from the Input Shift register to the DAC register. This will cause the data to show up at the output.

Alternatively, a trigger can be used to update multiple channels at once.

**See Also**

AP730_DacSoftwareTrigger
AP730_DacSetVout
AP730_DacWriteInput

## CONTINUOUS AND SINGLE BURST WAVEFORM FUNCTIONS

### AP730_DacWriteWaveBufferCodes

Copies data from a code buffer to a channel's sample memory

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacWriteWaveBufferCodes (int Handle,`<br>`DWORD Channel,`<br>`DWORD Offset,`<br>`WORD* Samples,`<br>`DWORD SampleCount);` |

**Parameters**

*Handle*
The handle to the board

*Channel*
Specifies the channel (0 – 3)

*Offset*
Offset relative to channel's Start Address

*Samples*
Address of buffer containing 16-bit data codes

*SampleCount*
The number of values to copy

**Return Value**
ERR_OK, ERR_BAD_PARAM, ERR_INVALID_HNDL

**Comments**
This function populates a channel's sample memory using the data in the specified buffer. This function should only be used with Continuous and Burst Single operating modes. The function will return *ERR_BAD_PARAM* if the specified *Offset* or *SampleCount* would result in data being copied beyond the channel's configured End Address.

**See Also**
AP730_DacWriteWaveBufferVolts

## AP730_DacWriteWaveBufferVolts

Converts voltage data to codes and copies them to a channel's sample memory

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacWriteWaveBufferVolts (int Handle, DWORD Channel, DWORD Offset, double* Samples, DWORD SampleCount);` |

**Parameters**

*Handle*
The handle to the board

*Channel*
Specifies the channel (0 – 3)

*Offset*
Offset relative to channel's Start Address

*Samples*
Address of buffer containing voltage values

*SampleCount*
The number of values to copy

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_INVALID_HNDL

**Comments**

This function converts voltage data to codes using the channel's calibration data and then passes the code data to *AP730_DacWriteWaveBufferCodes*.

This function should only be used with Continuous and Burst Single operating modes.

The function will return *ERR_BAD_PARAM* if the specified *Offset* or *SampleCount* would result in data being copied beyond the channel's configured End Address.

If the function encounters a voltage value that is out of range it will return ERR_BAD_PARAM without processing the remaining values. The channel's configured range determines what voltage values the function will accept. For example, if the range is currently set to ±5 volts, the function will return ERR_BAD_PARAM if an attempt is made to set the voltage to 7 volts. The function will also clip a voltage value at the upper limit of the range to an ideal value (e.g., if the range is currently set to ±5 volts, a 5 volt argument will be clipped to the ideal value, 4.999847).

**See Also**

AP730_DacWriteWaveBufferCodes

## FIFO BUFFER FUNCTIONS

### AP730_DacAttachFifoBuffer

Assigns a data buffer to be used as a source of data for loading a channel's FIFO

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacAttachFifoBuffer (int Handle,`<br>`DWORD Channel,`<br>`WORD* Buffer,`<br>`DWORD Size);` |

**Parameters**

*Handle*
The handle to the board

*Channel*
Specifies the channel (0 – 3)

*Buffer*
The data sample array.

*Size*
The size of the array (1 or greater)

**Return Value**
ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**
This function can be used to add or change the buffer used as a sample data source for loading a channel's FIFO. Only one data buffer may be associated with a channel at a time. To detach the currently assigned data buffer, call *AP730_DacDetachFifoBuffer* or call *AP730_DacAttachFifoBuffer* and specify a new buffer.

Attaching a buffer automatically sets the *BufferNext* index to 0 and the *BufferEnd* index to *Size* - 1

**Example**

**C:**
```
WORD samples[512];          // data buffer
status = AP730_DacAttachFifoBuffer (iHandle, 1, samples, 512);
```

**See Also**
AP730_DacDetachFifoBuffer
AP730_DacSetBufferNext
AP730_DacSetBufferEnd

## AP730_DacDetachFifoBuffer

Detaches the buffer used to supply data samples to a channel's FIFO

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacDetachFifoBuffer (int Handle, DWORD Channel);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

In addition to detaching the data sample buffer this function also resets the channel's "Load FIFO" settings (*Count* = 0, *Wrap* = FALSE, *Poll* = FALSE) and sets the data buffer's *BufferNext* and *BufferEnd* variables to 0.

In the event of a FIFO-half-full interrupt, the default interrupt service routine will disable the channel's Enable bit in the Interrupt Enable register if no buffer is available to reload the FIFO.

**See Also**

AP730_DacAttachFifoBuffer
AP730_DacLoadFifo

## AP730_DacSetBufferEnd

Sets the *BufferEnd* value for the specified channel's data buffer

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacSetBufferEnd (int Handle,`<br>`DWORD Channel,`<br>`DWORD EndIndex);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*EndIndex*

The new *BufferEnd* array index value (0 or greater)

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_BUFFER or ERR_INVALID_HNDL

**Comments**

The *AP730_DacLoadFifo* function is used to write values from an attached data buffer to a channel's FIFO. (FIFO Loading may be configured so that *AP730_DacLoadFifo* is called automatically when an interrupt occurs.) The *BufferEnd* setting specifies the end limit of the data set *AP730_DacLoadFifo* uses when loading the FIFO. In other word, only values stored at array addresses 0 to *BufferEnd* are ever written to the FIFO.

This function returns ERR_BUFFER if no buffer is currently associated with the specified channel. ERR_BAD_PARAM will be returned if *EndIndex* is greater than or equal to the *Size* specified when the buffer was attached.

Care should be taken not to set *BufferEnd* to a value less than the current value of *BufferNext*

**See Also**

AP730_DacGetBufferEnd
AP730_DacSetBufferNext
AP730_DacAttachFifoBuffer
AP730_DacLoadFifo

## AP730_DacGetBufferEnd

Gets the *BufferEnd* value for the specified channel's data buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetBufferEnd (int Handle,`<br>`DWORD Channel,`<br>`DWORD* EndIndex);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*EndIndex*

Receives the channel buffer's End Index value (0 or greater)

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The *AP730_DacLoadFifo* function is used to write values from an attached data buffer to a channel's FIFO. (FIFO Loading may be configured so that *AP730_DacLoadFifo* is called automatically when an interrupt occurs.) The *BufferEnd* setting specifies the end limit of the data set *AP730_DacLoadFifo* uses when loading the FIFO. In other word, only values stored at array addresses 0 to *BufferEnd* are ever written to the FIFO.

If no data buffer is attached the function will return an *EndIndex* value of 0.

**See Also**

AP730_DacSetBufferEnd
AP730_DacLoadFifo

## AP730_DacSetBufferNext

Sets the *BufferNext* value for the specified channel's data buffer

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacSetBufferNext (int Handle,`<br>`DWORD Channel,`<br>`DWORD NextIndex);` |

**Parameters**

*Handle*
The handle to the board

*Channel*
Specifies the channel (0 – 3)

*NextIndex*
The channel new *BufferNext* value (0 to *BufferEnd*)

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_BUFFER or ERR_INVALID_HNDL

**Comments**

The *AP730_DacLoadFifo* function is used to write values from an attached data buffer to a channel's FIFO. (FIFO Loading may be configured so that *AP730_DacLoadFifo* is called automatically when an interrupt occurs.) The set of array data *AP730_DacLoadFifo* uses when loading the FIFO is defined by array addresses 0 to *BufferEnd*.

If *AP730_DacLoadFifo* is called and the complete data is not written to the channel's FIFO, *BufferNext* will be set to one past the last value written. The value at this address will be the first value written to the FIFO the next time *AP730_DacLoadFifo* is called. If *AP730_DacLoadFifo* does write the last value of a data set to the FIFO, *BufferNext* wraps back to 0. Set *BufferNext* to 0 when you want to ensure that FIFO loading starts from the front of the array.

This function returns ERR_BUFFER if no buffer is currently associated with the specified channel. ERR_BAD_PARAM will be returned if *NextIndex* is greater than or equal to the *Size* specified when the buffer was attached.

Care should be taken not to set *BufferNext* to a value greater than the current value of *BufferEnd*.

**See Also**

AP730_DacGetBufferNext
AP730_DacAttachFifoBuffer
AP730_DacLoadFifo

## AP730_DacGetBufferNext

Gets the *BufferNext* value for the specified channel's data buffer

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DacGetBufferNext (int Handle,<br>DWORD Channel,<br>DWORD* NextIndex); |

**Parameters**

*Handle*
The handle to the board

*Channel*
Specifies the channel (0 – 3)

*NextIndex*
Receives the value of the channel's *BufferNext* variable

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The *AP730_DacLoadFifo* function is used to write values from an attached data buffer to a channel's FIFO. (FIFO Loading may be configured so that *AP730_DacLoadFifo* is called automatically when an interrupt occurs.) The set of array data *AP730_DacLoadFifo* uses when loading the FIFO is defined by array addresses 0 to *BufferEnd*.

If *AP730_DacLoadFifo* is called and the complete data is not written to the channel's FIFO, *BufferNext* will be set to one past the last value written. The value at this address will be the first value written to the FIFO the next time *AP730_DacLoadFifo* is called. If *AP730_DacLoadFifo* does write the last value of a data set to the FIFO, *BufferNext* wraps back to 0

If no data buffer is attached the function will return a *NextIndex* value of 0.

**See Also**

AP730_DacSetBufferNext
AP730_DacLoadFifo

## FIFO WAVEFORM FUNCTIONS

### AP730_DacSetFIFOLoading

Sets the specified channel's FIFO loading settings

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DacSetFIFOLoading (int Handle,<br>DWORD Channel,<br>DWORD Count,<br>BOOL Wrap,<br>BOOL Poll); |

**Parameters**

*Handle*
The handle to the board

*Channel*
Specifies the channel (0 – 3)

*Count*
The maximum number of values *AP730_DacLoadFifo* will write to the FIFO

*Wrap*
Allow wrapping back to start of buffer

*Poll*
Poll the FIFO-is-full status bit before each write

**Return Value**
ERR_OK, ERR_BAD_PARAM, ERR_BUFFER or ERR_INVALID_HNDL

**Comments**
The FIFO loading settings specify the action taken when the *AP730_DacLoadFifo* function is called. This function will return ERR_BUFFER if *Count* is greater than 0 and no buffer is attached.

See the description of *AP730_DacLoadFifo* for further information.

**See Also**
AP730_DacGetFIFOLoading
AP730_DacLoadFifo

## AP730_DacGetFIFOLoading

Gets the specified channel's FIFO loading setting

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetFIFOLoading (int Handle,`<br>`DWORD Channel`<br>`DWORD* Count,`<br>`BOOL* Wrap,`<br>`BOOL* Poll);` |

**Parameters**

*Handle*
The handle to the board

*Channel*
Specifies the channel (0 – 3)

*Count*
Receives the maximum number of values *AP730_DacLoadFifo* will write to the FIFO

*Wrap*
Receives the wrap setting

*Poll*
Receives the poll setting

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

The FIFO loading setting specifies the action taken when the *AP730_DacLoadFifo* function is called. See the description of *AP730_DacLoadFifo* for further information.

**See Also**

AP730_DacSetFIFOLoading
AP730_DacLoadFifo

## AP730_DacLoadFifo

Copies sample data from the channel's attached buffer to the channel's FIFO

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacLoadFifo (int Handle,`<br>`DWORD Channel,`<br>`DWORD NumWritten);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*NumWritten*

Receives the number of values written to the channel's FIFO

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_BUFFER or ERR_INVALID_HNDL

**Comments**

This function reads data samples from a sample array previously assigned with *AP730_DacAttachBuffer* and copies them to the specified channel's FIFO register. The function will only copy values with array addresses between 0 and *BufferEnd*. The first value copied from the array will be at the address specified by *BufferNext*.

The copying scheme *AP730_DacLoadFifo* uses depends on the *Count*, *Wrap* and *Poll* settings previously passed to *AP730_DacSetFIFOLoading*.

The *Count* setting specifies the maximum number samples the function will attempt to copy to the channel's FIFO. If this value exceeds the current size of the channel's FIFO, *Count* will be reduced to the FIFO size.

When *Poll* is set to TRUE, the FIFO-is-full status bit is read prior to each write. The function returns early if the bit is set.

Each time data is written to the FIFO, *BufferNext* is set to the address one past that of the last value copied. If *BufferNext* exceeds *BufferEnd*, *BufferNext* is set back to 0.

When *Wrap* is set to TRUE and *BufferNext* wraps back to 0, the function will continue to copy data until *Count* is reached or the FIFO becomes full (if polling). When *Wrap* is set to FALSE and *BufferNext* wraps back to 0, the function will return even if *Count* hasn't been reached yet.

**Example:**

```
// Assume the channel's FIFO size is 256 and waveform output
// is not enabled

// There is an unknown amount of data in the FIFO and we
// want to top it off with the data in buffer1.
dwChannel = 1;
AP730_DacAttachFifoBuffer (iHandle, dwChannel, pwBuffer1, 2000);

// Fill the FIFO using polling
fWrap = FALSE;
fPoll = TRUE;
AP730_DacSetFifoLoading (iHandle, dwChannel, 256, fWrap, fPoll);
AP730_DacLoadFifo (iHandle, dwChannel, &dwNumWritten);

// Now set the loading our interrupt service routine will use.
// When a FIFO-half-full interrupt occurs, we can safely write
// FIFO-size/2 samples without needing to poll
fWrap = FALSE;
fPoll = FALSE;
AP730_DacSetFifoLoading (iHandle, dwChannel, 128, fWrap, fPoll);

-----------------------------------------------------------------

// Within our interrupt callback function (that executes after the
// internal ISR has copied data to the FIFO) we can switch to a new
// buffer after determining that the first buffer has been consumed
AP730_DacGetBufferNext (iHandle, dwChannel, &dwNextIndex);

// Since the NextIndex has wrapped back to 0 all the data in Buffer1
// has been written to the FIFO
if (dwNextIndex == 0)
{
   // previous FIFO Loading settings are used for new buffer
   AP730_DacAttachFifoBuffer (iHandle, dwChannel, pwBuffer2,
   2000);
}
```

When a channel FIFO threshold interrupt is generated the *AP730_DacLoadFifo* function is automatically called for the channel by the default interrupt service routine. If a data buffer is attached, the function will reload the channel's FIFO in the fashion described above. If specialized loading of the FIFO is desired a callback function that overrides the default interrupt service routine can be written. See the AP730Demo project for an override callback that uses DMA to write data to the FIFOs.

**See Also**

AP730_DacSetFIFOLoading
AP730_DacSetBufferNext
AP730_DacSetBufferEnd

## AP730_DacWriteFifo16

Writes a 16-bit value to the specified channel's FIFO register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacWriteFifo16 (int Handle,`<br>`DWORD Channel,`<br>`WORD Data);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*Data*

The 16-bit value to be written to the FIFO register

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_INVALID_HNDL

**Comments**

This function provides an alternative to using sample buffers and the *AP730_DacLoadFifo* function to write values to the channel FIFOs.

The function does not test if the channel FIFO is full prior to writing to the FIFO Port. The *AP730_DacGetChannelStatus* function can be used for this purpose.

*Data* should be in the format the channel is configured to use. The *AP730_DacVoltsToCodes* and *AP730_DacVoltsToCodesEx* functions can be used to generate a sample codes in the appropriate format.

**See Also**

AP730_DacGetChannelStatus
AP730_DacSetDataFormat
AP730_DacVoltsToCodes
AP730_DacLoadBuffer
AP730_DacWriteFifo32

## AP730_DacWriteFifo32

Adds two 16-bit value to the specified channel's FIFO using a single register write

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacWriteFifo32 (int Handle,`<br>`DWORD Channel,`<br>`DWORD Data);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*Data*

The 32-bit value to be written to the FIFO register

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_INVALID_HNDL

**Comments**

This function provides an alternative to using sample buffers and the *AP730_DacLoadFifo* function to write values to the channel FIFOs.

The function does not test that there is room in the FIFO before to writing to the FIFO register. The *AP730_DacGetChannelStatus* function can be used for this purpose.

*Data* should consist of two 16-bit values in the format the channel is configured to use. The *AP730_DacVoltsToCodes* and *AP730_DacVoltsToCodesEx* functions can be used to generate a sample codes in the appropriate format.

**See Also**

AP730_DacGetChannelStatus
AP730_DacSetDataFormat
AP730_DacVoltsToCodes
AP730_DacLoadBuffer
AP730_DacWriteFifo16

## DATA CONVERSION FUNCTIONS

### AP730_DacVoltsToCodes

Converts an array of voltage values to corrected codes and copies them to the passed buffer

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DacVoltsToCodes (int Handle,<br>DWORD Channel,<br>double* VoltArray,<br>WORD* DataArray,<br>DWORD SampleCount); |

**Parameters**

*Handle*
The handle to the board

*Channel*
Specifies the channel (0 – 3)

*VoltArray*
Array of output voltages

*DataArray*
Array that receives the FIFO sample data

*SampleCount*
The number of samples to process (1 up to the size of the smallest passed array)

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function can be used to setup a sample buffer that will use to load a channel's FIFO. This buffer can be one that will be attached for use with *AP730_DacLoadFifo* or it can be a DMA buffer.

The function uses a channel's output range, data format, and calibration coefficients to convert a voltage at a given index in the *VoltArray* into a corrected data code. This data code is then copied to the corresponding index of the *DataArray*.

The function processes *SampleCount* number of values starting at index 0. If the function encounters a *VoltArray* value that is out of range it will return ERR_BAD_PARAM without processing the remaining values. The channel's output range determines what voltage values the function will accept. For example, if the range is currently set to ± 5 volts, the function will return ERR_BAD_PARAM if an attempt is made to set the voltage to 7 volts. The function will also clip a voltage value at the upper limit of the range to an ideal value (e.g., if the range is currently set to ± 5 volts, a 5 volt argument will be clipped to the ideal value, 4.999847).

The function can also be used with single variables rather than arrays. In this case set the *SampleCount* parameter to 1. After calling the function, the *DataArray* variable can then be passed to *AP730_DacWriteFifo16*.

**See Also**

AP730_DacAttachFifoBuffer
AP730_DacLoadFifo
AP730_DacVoltsToCodesEx

## AP730_DacVoltsToCodesEx

Converts an array of voltage values to corrected codes and copies them to the passed buffer

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DacVoltsToCodesEx (int Handle,<br>DWORD Channel,<br>double* VoltArray,<br>WORD* DataArray,<br>DWORD SampleCount,<br>DWORD Range,<br>DWORD Format); |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*VoltArray*

Array of output voltages

*DataArray*

Array that receives the FIFO sample data

*SampleCount*

The number of samples to process (1 up to the size of the smallest passed array)

*Range*

The output range value

| Value | Range |
|---|---|
| 0 | -10 to 10 |
| 1 | 0 to 10 |
| 2 | -5 to 5 |
| 3 | 0 to 5 |
| 4 | -2.5 to 7.5 |
| 5 | -3 to 3 |

*Format*

The data format

0: Straight binary

1: Two's complement

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function is an alternative to *AP730_DacVoltsToCodes*. It uses the specified output range and data format values to convert the data rather the range and format the channel is currently configured with.

**See Also**

AP730_DacVoltsToCodes

## CALIBRATION FUNCTIONS

Calibration data is provided in the form of calibration coefficients, so the user can adjust and improve the accuracy of the analog output voltage over the uncalibrated state. Each channel's unique offset and gain calibration coefficients are written to the AP730's Flash memory at the factory. The flash sector containing the coefficient values is also written with an ASCII string ("AP730") to indicate that the calibration data is present.

During the call to *AP730_Open* the software checks for the presence of the ASCII string, and if found, it reads the calibration coefficients from flash into a set of internal coefficient variables. Several functions use these variables in calculations to improve the accuracy of the output. If the ASCII string is missing, the variables are set to zero. In this case, the module can still be used but no corrections would be made to the output.

This section describes the functions that are provided for viewing and generating calibration coefficients.

### AP730_DacResetCalCorrections

Resets the offset and gain corrections for the specified channel.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacResetCalCorrections (int Handle, DWORD Channel);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function resets the internal calibration correction variables for the specified channel (all ranges). After calling this function, channel output will be uncorrected.

**See Also**

AP730_DacCalcCalCorrections

## AP730_DacGetCalCorrections

Gets the values of the calibration correction variables for the specified DAC channel and range

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DacGetCalCorrections (int Handle,`<br>`DWORD Channel,`<br>`DWORD Range,`<br>`short* OffsetCorr,`<br>`short* GainCorr);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*Range*

The output range value

| Value | Range |
|---|---|
| 0 | -10 to 10 |
| 1 | 0 to 10 |
| 2 | -5 to 5 |
| 3 | 0 to 5 |
| 4 | -2.5 to 7.5 |
| 5 | -3 to 3 |

*OffsetCorr*

Receives the offset correction

*GainCorr*

Receives the gain correction

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

## AP730_DacSetCalCorrections

Sets the calibration corrections for a DAC channel and range to the specified values

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacSetCalCorrections (int Handle,`<br>`DWORD Channel,`<br>`DWORD Range,`<br>`short OffsetCorr,`<br>`short GainCorr);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*Range*

The output range value

| Value | Range |
|---|---|
| 0 | -10 to 10 |
| 1 | 0 to 10 |
| 2 | -5 to 5 |
| 3 | 0 to 5 |
| 4 | -2.5 to 7.5 |
| 5 | -3 to 3 |

*OffsetCorr*

The new offset correction

*GainCorr*

The new gain correction

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function is a low-level alternative to the *AP730_DacCalcCalCorrections* function. The function can be used to change the value of the specified channel's offset and gain correction variables in memory. To save updated values to flash, call *AP730_WriteCalDataToFlash*.

**See Also**

AP730_DacCalcCalCorrections
AP730_WriteCalDataToFlash

## AP730_DacCalcCalCorrections

Calculates and updates gain and offset correction variables for the specified channel and range based on user measurements

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacCalcCalCorrections (int Handle, DWORD Channel, DWORD Range, double MeasuredV1, double MeasuredV2);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*Range*

Specifies the output range

| Value | Range |
|---|---|
| 0 | -10 to 10 |
| 1 | 0 to 10 |
| 2 | -5 to 5 |
| 3 | 0 to 5 |
| 4 | -2.5 to 7.5 |
| 5 | -3 to 3 |

*MeasuredV1*

Measured voltage for Code1

*MeasuredV2*

Measured voltage for Code2

**Return Value**

ERR_OK, ERR_BAD_PARAM, or ERR_INVALID_HNDL

**Comments**

This function can be used to calculate and update the gain and offset correction variables for a specific channel and range. This function only updates the variables. To save the new values to flash, call *AP730_WriteCalDataToFlash*.

The procedure for updating calibration coefficients is as follows:

1. Call *AP730_DacSetOutputRange* to configure the range for the channel.
2. Call *AP730_DacSetDataFormat* to set the data format to straight binary.
3. Call *AP730_DacWriteInput* to write Code1 (0x28F) to the Input Shift register. Set the *UpdateDAC* parameter to TRUE so that the DAC register will be updated automatically.
4. Measure the output using a voltage meter. This value is *MeasuredV1*.
5. Call *AP730_DacWriteInput* to write Code2 (0xFD70) to the Input Shift register. Set the *UpdateDAC* parameter to TRUE so that the DAC register will be updated automatically.

6. Measure the output using a voltage meter. This value is *MeasuredV2*.
7. Call *AP730_DacCalcCalCorrections* and pass in the measured values.
8. Repeat steps 1 – 7 for the other channels and ranges.
9. To save the new values to flash call *AP730_WriteCalDataToFlash*

**See Also**

AP730_DacSetOutputRange
AP730_DacSetDataFormat
AP730_DacWriteInput
AP730_WriteCalDataToFlash

## AP730_DacReadCalCorrectionsFromFlash

Reads the offset and gain correction values for the specified DAC channel and range from flash memory

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DacReadCalCorrectionsFromFlash (int Handle, DWORD Channel, DWORD Range, short* OffsetCorr, short* GainCorr);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*Range*

Specifies the output range

| Value | Range |
|---|---|
| 0 | -10 to 10 |
| 1 | 0 to 10 |
| 2 | -5 to 5 |
| 3 | 0 to 5 |
| 4 | -2.5 to 7.5 |
| 5 | -3 to 3 |

*OffsetCorr*

Receives the offset correction

*GainCorr*

Receives the gain correction

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_CALIB or ERR_INVALID_HNDL

**Comments**

The flash sector containing the coefficient values should also contain an ASCII string (“AP730”) as an indicator that the calibration data is present. If this string is not found the function returns ERR_CALIB.

**See Also**

`AP730_DacCalcCalCorrections`

# *Digital I/O Functions*

## I/O FUNCTIONS

### AP730_DioWriteIoReg

Writes a 16-bit value to the Digital Input/Output Register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DioWriteIoReg (int Handle,`<br>`WORD Data);` |

**Parameters**

*Handle*

The handle to the board

*Data*

The 16-bit value to be written (0 – FFFF)

**Return Value**

ERR_OK or ERR_INVALID_HNDL

## AP730_DioReadIoReg

Reads a 16-bit value from the Digital Input/Output Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DioReadIoReg (int Handle,`<br>`  WORD* Data);` |

**Parameters**

*Handle*

The handle to the board

*Data*

Receives the value of the I/O register (0 – FFFF)

**Return Value**

ERR_OK, ERR_INVALID_HNDL or ERR_BAD_PARAM

## AP730_DioWriteChannel

Writes a value to the specified digital I/O channel

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DioWriteChannel (int Handle,`<br>`DWORD Channel,`<br>`DWORD Data);` |

**Parameters**

*Handle*

The handle to the board

*Channel*

The target I/O channel (0 - 15)

*Data*

The value to be written (0 or 1)

**Return Value**

ERR_OK, ERR_INVALID_HNDL or ERR_BAD_PARAM

## AP730_DioReadChannel

Reads the specified digital I/O channel

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DioReadChannel (int Handle,`<br>`DWORD Channel,`<br>`DWORD* Data);` |

**Parameters**

*Handle*

The handle to the board

*Point*

The target I/O channel (0 to 15).

*Data*

Receives the value of the I/O channel (0 or 1)

**Return Value**

ERR_OK, ERR_INVALID_HNDL or ERR_BAD_PARAM

## STATUS FUNCTIONS

### AP730_DioReadIntStatus

Reads the Digital Channel Interrupt Pending bits from the Global Interrupt Status register

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DioReadIntStatus (int Handle,<br>WORD* Status); |

**Parameters**

*Handle*

The handle to the board

*Status*

Receives the 16-bit status value (0-FFFF)

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function reads the Digital Channel Interrupt Pending bits from the Global Interrupt Status register. Bit 0 corresponds to the digital channel 0, while bit 15 corresponds to digital channel 15. A “1” bit indicates a pending interrupt for the corresponding channel. A channel that does not have interrupts enabled will never set its pending flag. A channel’s interrupt can be released by writing a “1” to its bit position in the Global Interrupt Status Register.

Note interrupts will not be reported to the system unless bit-11 has been set in the Interrupt Enable Register.

**See Also**

AP730_DioWriteIntEnable
AP730_WriteIntEnable
AP730_DioWriteIntStatus

## AP730_DioWriteIntStatus

Writes the specified value to the Digital Channel Interrupt Clear bits in the Global Interrupt Status Clear register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DioWriteIntStatus (int Handle, WORD Clear);` |

**Parameters**

*Handle*

The handle to the board

*Clear*

The 16-bit value to be written (0-FFFF)

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

This function writes to to the Digital Channel Interrupt Clear bits in the Global Interrupt Status Clear register. A channel's interrupt can be released by writing a "1" to its bit position in the Global Interrupt Status Register. Bit 0 corresponds to the digital channel 0, while bit 15 corresponds to digital channel 15.

Note interrupts will not be reported to the system unless bit-11 has been set in the Interrupt Enable Register.

**See Also**

`AP730_DioReadIntStatus`

## CONFIGURATION FUNCTIONS

### AP730_DioReset

Resets counter/timer registers and variables.

| Syntax | |
|---|---|
| C: | `int _stdcall AP730_DioReset (int Handle);` |

**Parameters**
*Handle*
The handle to the board

**Comments**
This function also sets all the digital I/O and configuration registers to zero. Digital interrupts are disabled in the Interrupt Enable and Global Interrupt Enable registers.

**Return Value**
ERR_OK or ERR_INVALID_HNDL

## AP730_DioReadDirection

Reads the data direction for the specified channel group from the Digital Direction Control Register.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DioReadDirection (int Handle,`<br>`DWORD ChanGroup,`<br>`DWORD* Direction);` |

**Parameters**

*Handle*

The handle to the board

*ChanGroup*

The channel group
0: Channels 0 – 7
1: Channels 8 - 15

*Direction*

Receives the direction setting
0: Input
1: Output

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function is used to read the data direction (input or output) of the digital channels.

Seven of the AP730's digital channels may also be used for ADC triggering, DAC triggering or counter/timer actions. To use these actions the corresponding digital I/O port must be configured for the appropriate data direction. The *AP730_DioWriteDirection* function can be used to set the data direction.

If counter control input signals are used or an ADC or DAC trigger signal is enabled for input, then the Direction Control Register must also be set as input for channels 0 to 7.

If ADC or DAC trigger signal is enabled for output, then the data Direction Control Register must also be set as output for channels 8 to 15.

**See Also**

AP730_DioWriteDirection

.

## AP730_DioWriteDirection

Sets the data direction for the specified channel group in the Digital Direction Control Register.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DioWriteDirection (int Handle,`<br>`DWORD ChanGroup,`<br>`DWORD Direction);` |

**Parameters**

*Handle*

The handle to the board

*ChanGroup*

The channel group
0: Channels 0 – 7
1: Channels 8 - 15

*Direction*

The new direction setting
0: Input
1: Output

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function is used to set the data direction (input or output) of the digital channels.

Seven of the AP730's digital channels may also be used for ADC triggering, DAC triggering or counter/timer actions. To use these actions the corresponding digital I/O port must be configured for the appropriate data direction. The *AP730_DioWriteDirection* function can be used to set the data direction.

If counter control input signals are used or an ADC or DAC trigger signal is enabled for input, then the Direction Control Register must also be set as input for channels 0 to 7.

If ADC or DAC trigger signal is enabled for output, then the data Direction Control Register must also be set as output for channels 8 to 15.

**See Also**

AP730_DioReadDirection

## AP730_DioReadIntEnable

Reads the Digital Channel Interrupt Enable bits from the Global Interrupt Enable register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DioReadIntEnable (int Handle, WORD* Enable);` |

**Parameters**

*Handle*

The handle to the board

*Enable*

Receives the 16-bit interrupt enable value (0-FFFF)

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function reads the Digital Channel Interrupt Enable bits from the Global Interrupt Enable register. Bit 0 corresponds to the digital channel 0, while bit 15 corresponds to digital channel 15. A “0” bit will prevent the corresponding channel from generating an external interrupt. A “1” bit will allow the corresponding channel to generate an interrupt.

Note interrupts will not be reported to the system unless bit-11 has been set in the Interrupt Enable Register.

**See Also**

`AP730_WriteIntEnable`

## AP730_DioWriteIntEnable

Writes a value to the Digital Channel Interrupt Enable bits in the Global Interrupt Enable register

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DioWriteIntEnable (int Handle, WORD Enable);` |

**Parameters**

*Handle*

The handle to the board

*Enable*

The new 16-bit interrupt enable value (0-FFFF)

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

This function writes the Digital Channel Interrupt Enable bits in the Global Interrupt Enable register. Bit 0 corresponds to the digital channel 0, while bit 15 corresponds to digital channel 15. A “0” bit will prevent the corresponding channel from generating an external interrupt. A “1” bit will allow the corresponding channel to generate an interrupt.

Note interrupts will not be reported to the system unless bit-11 has been set in the Interrupt Enable Register.

**See Also**

`AP730_WriteIntEnable`

## AP730_DioReadIntType

Reads the Interrupt Type Configuration Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DioReadIntType (int Handle,`<br>`WORD* Type);` |

**Parameters**

*Handle*

The handle to the board

*Type*

Receives the 16-bit value of the Interrupt Type Register

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function reads the Interrupt Type Configuration register. Bit 0 corresponds to the digital channel 0, while bit 15 corresponds to digital channel 15. A “0” bit selects interrupt on level. An interrupt will be generated when the input channel level specified by the Interrupt Polarity Register occurs (i.e. Low or High level transition interrupt). A “1” bit means the interrupt will occur when a Change-Of-State (COS) occurs at the corresponding input channel (i.e. any state transition, low to high or high to low).

## AP730_DioWriteIntType

Writes a value to the Interrupt Type Configuration Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DioWriteIntType (int Handle,`<br>`WORD Type);` |

**Parameters**

*Handle*

The handle to the board

*Type*

The 32-bit value to be written to the register

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

This function writes to the Interrupt Type Configuration register. Bit 0 corresponds to the digital channel 0, while bit 15 corresponds to digital channel 15. A “0” bit selects interrupt on level. An interrupt will be generated when the input channel level specified by the Interrupt Polarity Register occurs (i.e. Low or High level transition interrupt). A “1” bit means the interrupt will occur when a Change-Of-State (COS) occurs at the corresponding input channel (i.e. any state transition, low to high or high to low).

## AP730_DioReadIntPolarity

Reads the Interrupt Polarity Register.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DioReadIntPolarity (int Handle, WORD* Polarity);` |

**Parameters**

*Handle*

The handle to the board

*Data*

Receives the 16-bit value of the Interrupt Polarity Register

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function reads the Interrupt Type Configuration register. Bit 0 corresponds to the digital channel 0, while bit 15 corresponds to digital channel 15. The Interrupt Polarity Register determines the level that will cause a channel interrupt to occur for each of the channels enabled for level interrupts. A “0” bit specifies that an interrupt will occur when the corresponding input channel is low (i.e. a “0” in the digital input channel data register). A “1” bit means that an interrupt will occur when the input channel is high (i.e. a “1” in the digital input channel data register).

## AP730_DioWriteIntPolarity

Writes a value to the Interrupt Polarity Register.

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DioWriteIntPolarity (int Handle, WORD Polarity);` |

**Parameters**

*Handle*

The handle to the board

*Polarity*

The 16-bit value to be written to the register

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

This function writes to the Interrupt Polarity register. Bit 0 corresponds to the digital channel 0, while bit 15 corresponds to digital channel 15. The Interrupt Polarity Register determines the level that will cause a channel interrupt to occur for each of the channels enabled for level interrupts. A “0” bit specifies that an interrupt will occur when the corresponding input channel is low (i.e. a “0” in the digital input channel data register). A “1” bit means that an interrupt will occur when the input channel is high (i.e. a “1” in the digital input channel data register).

## AP730_DioGetChannelDebounce

Reads the Debounce Duration setting for the specified channel

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DioGetChannelDebounce (int Handle, DWORD Chan, DWORD* Debounce);` |

**Parameters**

*Handle*

The handle to the board

*Chan*

The target I/O channel number (0 - 15).

*Data*

Receives the debounce duration setting.

| Data | Function |
|---|---|
| 0 | Debounce Disabled |
| 1 | 4μs |
| 2 | 64μs |
| 3 | 1ms |
| 4 | 8ms |

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function reads from the Debounce Enable and Debounce Duration Select registers.

## AP730_DioSetChannelDebounce

Writes the Debounce Duration setting for the specified channel

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DioSetChannelDebounce (int Handle,<br>DWORD Chan,<br>DWORD Debounce); |

**Parameters**

*Handle*

The handle to the board

*Chan*

The target I/O channel number (0 - 15).

*Data*

The debounce duration setting.

| Data | Function |
|---|---|
| 0 | Debounce Disabled |
| 1 | 4μs |
| 2 | 64μs |
| 3 | 1ms |
| 4 | 8ms |

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function writes to the Debounce Enable and Debounce Duration Select registers.

# *DMA Functions*

## CDMA FUNCTIONS

### AP730_CdmaReadControl

Reads the value of the CDMA Control Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CdmaReadControl (int Handle, DWORD* Data);` |

**Parameters**

*Handle*

The handle to the board

*Data*

Receives the 32-bit value of the register.

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function may be used to check the CDMA configuration.

## AP730_CdmaReset

Performs a soft reset on the AXI CDMA core

| Syntax | |
|---|---|
| C: | `int _stdcall AP730_CdmaReset (int Handle);` |

**Parameters**

*Handle*

The handle to the board

**Return Value**

ERR_OK, ERR_INVALID_HNDL, or ERR_TIMEOUT

**Comments**

This function performs a soft reset by writing to the CDMA control register. The function will return ERR_TIMEOUT if the reset does not complete within 100 microseconds

## AP730_CdmaReadStatus

Reads the value of the CDMA Status Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CdmaReadStatus (int Handle, DWORD* Data);` |

**Parameters**

*Handle*

The handle to the board

*Data*

Receives the 32-bit value of the register.

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

## AP730_CdmaReadIntStatus

Reads the value of the CDMA Interrupt Status Bits from the CDMA Status Register

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_CdmaReadIntStatus (int Handle, DWORD* Status);` |

**Parameters**

*Handle*

The handle to the board

*Status*

Receives the 32-bit value.

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

**Comments**

This function isolates interrupt status bits used with Simple DMA Mode.

- Bit 12: Interrupt on Complete
- Bit 14: Interrupt on Error

## AP730_CdmaClearIntStatus

Writes the CDMA Status Register to clear active interrupt bits

| Syntax | |
|---|---|
| C: | `int _stdcall AP730_CdmaClearIntStatus (int Handle, DWORD Data);` |

**Parameters**

*Handle*

The handle to the board

*Data*

The value to write.

**Return Value**

ERR_OK or ERR_INVALID_HNDL

**Comments**

The following interrupt status bits are used with Simple DMA Mode.

- Bit 12: Interrupt on Complete
- Bit 14: Interrupt on Error

## AP730_CdmaIsIdle

Determines if AXI CDMA is idle

| Syntax | |
|---|---|
| C: | int _stdcall AP730_CdmaIsIdle (int Handle,<br>BOOL* Idle); |

**Parameters**

*Handle*

The handle to the board

*Idle*

Receives the state of the CDMA Idle bit in the CDMA Status register

**Return Value**

ERR_OK, ERR_BAD_PARAM or ERR_INVALID_HNDL

## AP730_AdcCdmaSetup

Sets up a simple DMA transfer between the FIFO Channel Data and Tag Register and the DMA buffer

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_AdcCdmaSetup (int Handle,`<br>`DWORD BuffOffset,`<br>`DWORD IntMask);` |

**Parameters**

*Handle*

The handle to the board

*BuffOffset*

The byte offset relative to the start of the DMA buffer

*IntMask*

Interrupt enable mask written to CDMA Control register

- Bit 12: Interrupt on Complete
- Bit 14: Interrupt on Error

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_INVALID_HNDL, ERR_DMA_MAP, or ERR_BUSY

**Comments**

This function should be used to setup the DMA transfer that will be automatically kicked off when using ADC Auto DMA Mode. In Auto DMA Mode a DMA transfer will move samples from the ADC FIFO to the DMA buffer starting at *BuffOffset*. The number of samples moved will be equal to the value in the ADC FIFO Full Interrupt Threshold register.

The *AP730_CdmaStartTransfer* function can also be used to start a DMA transfer.

The *IntMask* parameter can be used to enable interrupts in the AXI CDMA core. In order for the system to see these interrupts AXI CDMA interrupts must also be enabled in the Interrupt Controller. See the *AP730_SetUserCallback* function description for more detail about using Auto DMA Mode with interrupts.

The function will return *ERR_BUSY* if the CDMA core is not idle and cannot be configured. The function will return *ERR_DMA_MAP* is a DMA buffer is not mapped.

The AP730 has a single AXI CDMA core. Simultaneous use of DMA by ADC and DAC functions is not supported.

**See Also**

AP730_AdcSetAutoDmaMode
AP730_CdmaStartTransfer
AP730_WriteIntEnable
AP730_SetUserCallback

## AP730_DacCdmaSetup

Sets up a simple DMA transfer from the DMA buffer to a DAC channel's FIFO register

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DacCdmaSetup (int Handle,<br>DWORD Channel,<br>DWORD BuffOffset,<br>DWORD IntMask); |

**Parameters**

*Handle*

The handle to the board

*Channel*

Specifies the channel (0 – 3)

*BuffOffset*

The byte offset relative to the start of the DMA buffer

*IntMask*

Interrupt enable mask written to CDMA Control register

- Bit 12: Interrupt on Complete
- Bit 14: Interrupt on Error

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_INVALID_HNDL, ERR_DMA_MAP, or ERR_BUSY

**Comments**

This function can be used to setup the DMA transfer that copy data from the DMA buffer to a DAC channel's FIFO register. The DMA transfer will copy samples from DMA buffer starting at *BuffOffset.*

The *AP730_CdmaStartTransfer* function is used to start a DMA transfer.

The *IntMask* parameter can be used to enable interrupts in the AXI CDMA core. In order for the system to see these interrupts AXI CDMA interrupts must also be enabled in the Interrupt Controller.

The function will return *ERR_BUSY* if the CDMA core is not idle and cannot be configured. The function will return *ERR_DMA_MAP* is a DMA buffer is not mapped.

See the *LoadDacFifoUsingDma* function in the AP730Demo project for example code that performs a DMA transfer.

The AP730 has a single AXI CDMA core. Simultaneous use of DMA by ADC and DAC functions is not supported.

**See Also**

AP730_CdmaStartTransfer

## AP730_CdmaStartTransfer

Starts a DMA transfer of the specified number of bytes

| Syntax | |
|---|---|
| C: | int _stdcall AP730_CdmaStartTransfer (int Handle,<br>DWORD NumBytes); |

**Parameters**

*Handle*

The handle to the board

*NumBytes*

The number of bytes to transfer

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_INVALID_HNDL, ERR_DMA_MAP, or ERR_BUSY

**Comments**

This function can be used to start a DMA transfer that was previously configured using *AP730_AdcCdmaSetup* or *AP730_DacCdmaSetup*.

The function will return *ERR_BUSY* if the CDMA core is not idle and the transfer cannot be started.
The function will return *ERR_DMA_MAP* is a DMA buffer is not mapped.

The AP730 has a single AXI CDMA core. Simultaneous use of DMA for analog input and analog output is not supported.

**See Also**

AP730_AdcCdmaSetup
AP730_DacCdmaSetup

## DMA BUFFER FUNCTIONS

AP730 Windows driver software supports the use of one contiguous DMA buffer. This buffer can be logically partitioned in to smaller buffers that are used for different purposes. See the AP730Demo source code for examples.

The size of the buffer is specified by a registry value that is created when the board is installed. When the driver is loaded it allocates the DMA kernel buffer. The kernel buffer is mapped into user space during the call to *AP730_Open*.

To minimize updates to the AXI Base Address Translation Configuration register, the least significant 19 bits of the physical address of the DMA buffer should be zeroes. In most cases the address of the buffer obtained from the system will not meet this requirement. To work around this issue, the size of the buffer should be set to 512K greater than the size required for the DMA transfers. This allows the software to locate a smaller, properly aligned DMA buffer within the memory allocated.

The buffer size is set to 0xC8000 bytes by default which yields at least 288K of buffer usable for DMA.

**Example DMA buffer address and size values (32-bit system):**

| | |
|---|---|
| Buffer size setting from registry | 0xC8000 bytes (288KB + 512KB) |
| Physical address from system | 0x237e58000 |
| Aligned physical address | 0x237e80000 |
| Usable buffer size | 0xC8000 – (0x237e80000 – 0x237e58000) = 0xA0000 (640KB) (returned by *AP730_DmaGetBufferSize*) |

The DMA buffer size used by boards of the same type (same device ID) can be changed by editing a registry value and rebooting the system.

1. Launch the registry editor
   Click Start
   a. Type ‘regedit’ in the Search box
   b. Press Enter

2. Navigate to the board’s Parameters key

   HKLM\System\CurrentControlSet\Services\Acromag\Parameters\VEN_16D5&DEV_7073

3. Right-click on the **DmaBufferSize** value and select Modify

   Edit this value to increase or decrease the size of the buffer. For example, to specify a DMA buffer of size of 1024KB (at least 512KB usable for DMA), change the value to 100000 (hexadecimal) and click OK.

4. Exit the registry editor and reboot the system

## AP730_DmaGetBuffMapAddress

Gets the base address of the user mode mapping of the DMA buffer

C and C++ programmers can cast the returned value to a pointer and access buffer memory using normal pointer mechanisms

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_DmaGetBuffMapAddress (int Handle, UINT64* BuffAddress);` |

**Parameters**

*Handle*

The handle to the board

*BaseAddress*

Receives the virtual address of the DMA buffer

**Return Value**

ERR_OK, ERR_INVALID_HNDL, ERR_BAD_PARAM or ERR_DMA_MAP

**Comments**

This function will return ERR_DMA_MAP if a buffer is not mapped. The user mode mapped address returned by the function corresponds to the 512KB aligned physical address.

**Example**

```
/* Read second byte of buffer */

UINT64 u64BuffAddr;
volatile BYTE* pbBuffer;
BYTE bBuffVal;

if (AP730_DmaGetBuffMapAddress(Handle, &u64BuffAddr) == ERR_OK)
{
   pbBuffer = (BYTE*)u64BuffAddr;
   bBuffVal = *(pbBuffer + 1);
}
```

## AP730_DmaGetBufferSize

Gets the size of the mapped DMA buffer in bytes

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DmaGetBufferSize (int Handle, DWORD* BuffSize);` |

**Parameters**

*Handle*

The handle to the board

*BuffSize*

Receives the size in bytes of the mapped DMA buffer

**Return Value**

ERR_OK, ERR_INVALID_HNDL, ERR_BAD_PARAM or ERR_DMA_MAP

**Comments**

The function will return ERR_DMA_MAP if a buffer is not mapped. This size of buffer is dependent on the amount of correction required to align the buffer on a 512KB boundary. The size may vary with system restarts but will not fall below the configured minimum value. See the **DMA Buffer Functions** section for details.

## AP730_DmaReadBufferByte

Reads a byte from the mapped DMA buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DmaReadBufferByte (int Handle,`<br>`DWORD Offset,`<br>`BYTE* Data);` |

**Parameters**

*Handle*

The handle to the board

*Offset*

The offset relative to the start of the DMA buffer (0 to (*DmaBufferSize* – 1))

*Data*

Receives the 8-bit value read

**Return Value**

ERR_OK, ERR_INVALID_HNDL, ERR_BAD_PARAM or ERR_DMA_MAP

**Comments**

The function will return ERR_DMA_MAP if a buffer is not mapped.

C and C++ developers may also call *AP730_DmaGetBuffMapAddress* and access the buffer through a pointer.

**See Also**

AP730_DmaGetBufferSize
AP730_DmaGetBuffMapAddress

## AP730_DmaWriteBufferByte

Writes a byte to the specified offset of the mapped DMA buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DmaWriteBufferByte (int Handle,`<br>`DWORD Offset,`<br>`BYTE Data);` |

**Parameters**

*Handle*

The handle to the board

*Offset*

The offset relative to the start of the DMA buffer (0 to (*DmaBufferSize* – 1))

*Data*

The 8-bit value to be written

**Return Value**

ERR_OK, ERR_INVALID_HNDL, ERR_BAD_PARAM or ERR_DMA_MAP

**Comments**

The function will return ERR_DMA_MAP if a buffer is not mapped.

C and C++ developers may also call *AP730_DmaGetBuffMapAddress* and access the buffer through a pointer.

**See Also**

AP730_DmaGetBufferSize
AP730_DmaGetBuffMapAddress

## AP730_DmaReadBufferWord

Reads a word from the mapped DMA buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DmaReadBufferWord (int Handle,`<br>`DWORD Offset,`<br>`WORD* Data);` |

**Parameters**

*Handle*

The handle to the board

*Offset*

The offset relative to the start of the DMA buffer (0 to (*DmaBufferSize* – 2))

*Data*

Receives the 16-bit value read.

**Return Value**

ERR_OK, ERR_INVALID_HNDL, ERR_BAD_PARAM or ERR_DMA_MAP

**Comments**

The function will return ERR_DMA_MAP if a buffer is not mapped.

C and C++ developers may also call *AP730_DmaGetBuffMapAddress* and access the buffer through a pointer.

**See Also**

AP730_DmaGetBufferSize
AP730_DmaGetBuffMapAddress

## AP730_DmaWriteBufferWord

Writes a word to the specified offset of the mapped DMA buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DmaWriteBufferWord (int Handle,`<br>`DWORD Offset,`<br>`WORD Data);` |

**Parameters**

*Handle*

The handle to the board

*Offset*

The offset relative to the start of the DMA buffer (0 to (*DmaBufferSize* – 2))

*Data*

The 16-bit value to be written

**Return Value**

ERR_OK, ERR_INVALID_HNDL, ERR_BAD_PARAM or ERR_DMA_MAP

**Comments**

The function will return ERR_DMA_MAP if a buffer is not mapped.

C and C++ developers may also call *AP730_DmaGetBuffMapAddress* and access the buffer through a pointer.

**See Also**

AP730_DmaGetBufferSize
AP730_DmaGetBuffMapAddress

## AP730_DmaReadBufferDword

Reads a double word from the mapped DMA buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DmaReadBufferDword (int Handle,`<br>`DWORD Offset,`<br>`DWORD* Data);` |

**Parameters**

*Handle*

The handle to the board

*Offset*

The offset relative to the start of the DMA buffer (0 to (*DmaBufferSize* – 4))

*Data*

Receives the 32-bit value read.

**Return Value**

ERR_OK, ERR_INVALID_HNDL, ERR_BAD_PARAM or ERR_DMA_MAP

**Comments**

The function will return ERR_DMA_MAP if a buffer is not mapped.

C and C++ developers may also call *AP730_DmaGetBuffMapAddress* and access the buffer through a pointer.

**See Also**

AP730_DmaGetBufferSize
AP730_DmaGetBuffMapAddress

## AP730_DmaWriteBufferDword

Writes a double word to the specified offset of the mapped DMA buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DmaWriteBufferDword (int Handle,`<br>`DWORD Offset,`<br>`DWORD Data);` |

**Parameters**

*Handle*

The handle to the board

*Offset*

The offset relative to the start of the DMA buffer (0 to (*DmaBufferSize* – 4))

*Data*

The 32-bit value to be written

**Return Value**

ERR_OK, ERR_INVALID_HNDL, ERR_BAD_PARAM or ERR_DMA_MAP

**Comments**

The function will return ERR_DMA_MAP if a buffer is not mapped.

C and C++ developers may also call *AP730_DmaGetBuffMapAddress* and access the buffer through a pointer.

**See Also**

AP730_DmaGetBufferSize
AP730_DmaGetBuffMapAddress

## AP730_DmaReadBufferQword

Reads a quad word (64-bit) from the mapped DMA buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DmaReadBufferQword (int Handle,`<br>`DWORD Offset,`<br>`UINT64* Data);` |

**Parameters**

*Handle*

The handle to the board

*Offset*

The offset relative to the start of the DMA buffer (0 to (*DmaBufferSize* – 8))

*Data*

Receives the 64-bit value read.

**Return Value**

ERR_OK, ERR_INVALID_HNDL, ERR_BAD_PARAM or ERR_DMA_MAP

**Comments**

The function will return ERR_DMA_MAP if a buffer is not mapped.

C and C++ developers may also call *AP730_DmaGetBuffMapAddress* and access the buffer through a pointer.

**See Also**

AP730_DmaGetBufferSize
AP730_DmaGetBuffMapAddress

## AP730_DmaWriteBufferQword

Writes a quad word (64-bit) to the specified offset of the mapped DMA buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DmaWriteBufferQword (int Handle,`<br>`DWORD Offset,`<br>`UINT64 Data);` |

**Parameters**

*Handle*

The handle to the board

*Offset*

The offset relative to the start of the DMA buffer (0 to (*DmaBufferSize* – 8))

*Data*

The 64-bit value to be written

**Return Value**

ERR_OK, ERR_INVALID_HNDL, ERR_BAD_PARAM or ERR_DMA_MAP

**Comments**

The function will return ERR_DMA_MAP if a buffer is not mapped.

C and C++ developers may also call *AP730_DmaGetBuffMapAddress* and access the buffer through a pointer.

**See Also**

AP730_DmaGetBufferSize
AP730_DmaGetBuffMapAddress

## AP730_DmaBufferSet

Sets a block of the DMA buffer to a specified value

| Syntax | |
|---|---|
| C: | int _stdcall AP730_DmaBufferSet (int Handle,<br>DWORD Offset,<br>DWORD Count,<br>BYTE Value); |

**Parameters**

*Handle*

The handle to the board

*Offset*

The offset relative to the start of the DMA buffer (0 to (*DmaBufferSize* – 1))

*Count*

The number of bytes to write (0 to (*DmaBufferSize* – *Offset*))

*Value*

The byte value to set

**Return Value**

ERR_OK, ERR_INVALID_HNDL, ERR_DMA_MAP or ERR_BAD_PARAM

**Comments**

The function will return ERR_DMA_MAP if a buffer is not mapped. The function will return ERR_BAD_PARAM if the specified *Offset* or *Count* would write beyond the limits of the buffer in memory. The DMA buffer size is determined by an entry in the Windows registry. The *AP730_DmaGetBufferSize* function can be used to obtain the buffer size in your program.

**See Also**

AP730_DmaGetBufferSize

## AP730_DmaBufferCopy

Copies a block of data to or from the DMA buffer

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_DmaBufferCopy (int Handle,`<br>`BYTE* DataArray,`<br>`DWORD DataOffset,`<br>`DWORD DmaBuffOffset,`<br>`DWORD Count,`<br>`int Direction);` |

**Parameters**

*Handle*
The handle to the board

*DataArray*
The external array that data will be copied to or from

*DataOffset*
The byte offset relative to the start of the external array

*DmaBuffOffset*
The offset relative to the start of the DMA buffer (0 to (*DmaBufferSize* – 1))

*Count*
The number of bytes to copy (0 to (*DmaBufferSize* – *DmaBuffOffset*))

*Direction*
0: Copy from external buffer to DMA buffer
1: Copy from DMA buffer to external buffer

**Return Value**
ERR_OK, ERR_INVALID_HNDL, ERR_DMA_MAP or ERR_BAD_PARAM

**Comments**
The function will return ERR_DMA_MAP if a buffer is not mapped. The function will return ERR_BAD_PARAM if the specified *DmaBuffOffset* or *Count* would access beyond the limits of the DMA buffer in memory. The DMA buffer size is determined by an entry in the Windows registry. The *AP730_DmaGetBufferSize* function can be used to obtain the buffer size in your program.

An alternative to using this function is to call *AP730_DmaGetBuffMapAddress* to get a pointer to the DMA buffer and then use memcpy to copy the data.

# *Miscellaneous Functions*

## AP730_WriteCalDataToFlash

Writes the current values of all ADC and DAC Offset and Gain corrections variables to flash

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_WriteCalDataToFlash (int Handle);` |

**Parameters**

*Handle*

The handle to the board

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_CALIB, ERR_INVALID_HNDL ERR_READBACK, or ERR_TIMEOUT

**Comments**

The *AP730_AdcCalcCalCorrections* and *AP730_AdcSetCalCorrections* functions can be used to update the ADC correction variable values. The *AP730_DacCalcCalCorrections* and *AP730_DacSetCalCorrections* functions can be used to update the DAC correction variable values. The *AP730_WriteCalDataToFlash* function can then be used to save the new values to flash.

After writing the values to flash the function reads them back to verify they were written properly.

**See Also**

AP730_AdcCalcCalCorrections
AP730_AdcSetCalCorrections
AP730_DacCalcCalCorrections
AP730_DacSetCalCorrections

## AP730_ReadPCIeReg

Reads a PCI Express Configuration Register value

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_ReadPCIeReg (int Handle,`<br>`DWORD Offset,`<br>`DWORD* Data);` |

**Parameters**

*Handle*

The handle to the board

*Offset*

Offset into PCIe Configuration space (0 – 0xFC)

*Data*

Variable to hold the value read from the register

**Return Value**

ERR_OK, ERR_BAD_PARAM, ERR_CONFIG_READ or ERR_INVALID_HNDL

## AP730_ReadFirmwareRev

Reads the board's firmware revision

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_ReadFirmwareRev (int Handle, char* Revision);` |

**Parameters**

*Handle*

The handle to the board

*Revision*

Receives the revision of the MCS firmware file used to configure the board

**Return Value**

ERR_OK, ERR_BAD_PARAM, or ERR_INVALID_HNDL

## AP730_GetBaseAddress

Gets the base address of the user mode mapping of the specified AP730 memory space

C and C++ programmers can cast the returned value to a byte pointer and access memory using normal pointer mechanisms. This method can be used to write additional functions that complement those provided through the DLL.

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_GetBaseAddress (int Handle, UINT64* BaseAddress, DWORD* Length);` |

**Parameters**

*Handle*

The handle to the board.

*BaseAddress*

Receives the 32 or 64 bit base address of the memory space

*Length*

Receives the length of the board's memory space in bytes

**Return Value**

ERR_OK, ERR_INVALID_HNDL or ERR_BAD_PARAM

**Example**

```
/* Read first 16-bit value from DAC Sample Memory */

UINT64 base_address;
volatile BYTE* pbase_addr;
DWORD length;
WORD value;

if (AP730_GetBaseAddress(Handle, &base_address, &length) == 0)
{
   pbase_addr = (BYTE*)base_address;
   value = *(PWORD)(pbase_addr + 0x60000);
}
```

## AP730_GetDeviceCount

Counts the properly installed and enabled AP730 boards in the system

| Syntax | |
|---|---|
| **C:** | `int _stdcall AP730_GetDeviceCount (int* Count);` |

**Parameters**

*Count*

Receives the count of boards present

**Return Value**

ERR_OK, ERR_BAD_PARAM, or ERR_INVALID_CTRLHNDL

**Comments**

This function only counts boards that have been properly installed and are currently enabled. Boards that are physically present but have not had a driver installed, or boards that were installed but were later uninstalled or disabled using Device Manager, are not counted.

The function will return ERR_INVALID_CTRLHNDL if the kernel driver (acrmgpci.sys) is not loaded. To prevent this error at least one Acromag board (any board supported by PCISW-API-WIN, APSW-API-WIN or IPSW-API-WIN) must be properly installed. However, *Count* is still set to 0 when this error occurs to indicate no installed AP730 boards with the specified ID are present.

When boards are present, subtracting 1 from *Count* will typically yield the maximum board number that can be passed to *AP730_Open*. Exceptions to this rule can occur when boards are uninstalled or disabled in Device manager without rebooting. (If two boards are present and “Board 0” is disabled, “Board 1” remains “Board 1” until the system is rebooted.)

**See Also**

AP730_Open

## AP730_ListDevices

Retrieves Board Numbers and Location information for active AP730 boards in the system

| Syntax | |
|---|---|
| C: | int _stdcall AP730_ListDevices (struct AP730Info devInfo[],<br>size_t len,<br>int* Count); |

**Parameters**

*devInfo*

Array of AP730nfo structures

Each struct has the following members

*int iBoardNum*

The identifier that would be passed to *AP730_Open* to connect to a board

*WORD wSlot*

Number indicating the system slot where the carrier card is located

*char cSite*

The carrier card site ('A' to 'H') where the AP module is located

*len*

The size of the array

*Count*

Receives the number of array elements populated

**Return Value**

ERR_OK, ERR_BAD_PARAM, or ERR_INVALID_CTRLHNDL

**Comments**

This function populates an array of structs with Board Numbers and Location information for the AP730 boards in a system. Only boards that have been properly installed and are currently enabled are included. Boards that are physically present but have not had a driver installed, or boards that were installed but were later uninstalled or disabled using Device Manager, are not included.

Location information is read from the board's module's "Location In System" register. The *Slot* number identifies the slot location in a system of the AP carrier card. The Carrier may use backplane signals as in a VPX system or a carrier DIP switch to uniquely identify the system location of the carrier. Thirty two unique system slots can be identified by the Slot location bits (3 to 7). See the system backplane or motherboard documentation for the physical location of the carrier in the system corresponding to a *Slot* number.

If a board currently is open, the function will not be able to read its location register. In this case Slot is set to 0, and Site it set to 'X'.

The function will return *ERR_INVALID_CTRLHNDL* if the kernel driver (acrmgpci.sys) is not loaded. To prevent this error at least one Acromag board (any board supported by APSW-API-WIN or newer versions of PCISW-API-WIN or IPSW-API-WIN) must be properly installed. *Count* is still set to 0 when this error occurs to indicate no installed AP730 boards are present.

**Example**

```
/* List active AP730's */
int count;
int status;
struct AP730Info devInfo[10];
status = AP730_ListDevices (devInfo,
                            sizeof(devInfo)/sizeof(struct AP730Info),
                            &count);
if (status == 0 && count > 0)
{
   printf ("\nBoard No.   Slot   Site");
   printf ("\n-----------------------\n");
   for (int i = 0; i < count; i++)
   {
       if (devInfo[i].cSite != 'X')
       {
          printf ("%3d         %02X       %C\n",
          devInfo[i].iBoardNum, devInfo[i].wSlot, devInfo[i].cSite);
       }
       else
       {
          printf ("%3d         ??      ??  (open in another app)\n",
          devInfo[i].iBoardNum);
       }
   }
}
```

**See Also**

AP730_OpenEx
AP730_ReadLocation

## AP730_SystemMonitor

Reads temperature or voltage data from the System Monitor

| **Syntax** | |
|---|---|
| **C:** | `int _stdcall AP730_SystemMonitor (int Handle,`<br>`DWORD Address,`<br>`double* Value);` |

**Parameters**

*Handle*

The handle to the board

*Address*

System Monitor address – specifies value to read

| Address (hex) | Status Register |
|---|---|
| 00 | Temperature |
| 01 | Vccint |
| 02 | Vccaux |
| 20 | Maximum Temperature |
| 21 | Maximum Vccint |
| 22 | Maximum Vccaux |
| 24 | Minimum Temperature |
| 25 | Minimum Vccint |
| 26 | Minimum Vccaux |

*Value*

Receives a temperature value in degrees Celsius or a voltage value in Volts

**Return Value**

ERR_OK, ERR_INVALID_HNDL, or ERR_BAD_PARAM

## AP730_MicrosecondDelay

Delays execution for the specified duration

| Syntax | |
|---|---|
| C: | `int _stdcall AP730_MicrosecondDelay (double MicroSeconds);` |

**Parameters**
*Microseconds*
The delay time in microseconds

**Return Value**
ERR_OK

**Comments**
This function poll's the system's performance counter in a tight loop until the specified number of microseconds has elapsed. Short delay times are recommended since the polling routine is CPU intensive.

**Artisan Technology Group** is an independent supplier of quality pre-owned equipment

## Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

## We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

## Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP