DYNAMIC ENGINEERING
150 DuBois St., Suite C Santa Cruz, CA 95060
(831) 457-8891 **Fax** (831) 457-4793
http://www.dyneng.com
sales@dyneng.com
Est. 1988

ccPMC Parallel TTL BA18 Base & Channel

Driver Documentation

Win32 Driver Model

Manual Revision A
Corresponding Hardware: Revision A
10-2008-1301
Corresponding Firmware:
BA18: Design 2 Revision 3

BA18Base & BA18Chan
WDM Device Drivers for the
PMC Parallel TTL BA18
Parallel TTL Interface w/ COS
8 ADC with Data Reduction

Dynamic Engineering
150 DuBois St., Suite C
Santa Cruz, CA 95060
(831) 457-8891
FAX: (831) 457-4793

Manual Revision A Revised July 15th, 2009

Table of Contents

Introduction

The BA18Base and BA18Chan drivers are Win32 driver model (WDM) device drivers for the ccPMC-Parallel-TTL BA18 from Dynamic Engineering.

The BA18 driver package has three parts. The driver is installed into the Windows® OS, the test executable and the User Application "Userap" exectutable.

The driver and test are delivered as installed or executable items to be used directly or indirectly by the user. The Userap code is delivered in source form [C] and is for the purpose of providing a reference to using the driver.

The "test" executable allows the user to use the driver in script form from a DOS window. Each driver call can be accessed, parameters set and returned. Normally not need or used by the integrator, but a very handy tool in certain circumstances. The test executable has a "help" menu to explain the calls, parameters and returned information.

UserAp is a stand-alone code set with a simple and powerful menu plus a series of "tests" that can be run on the installed hardware. Each of the tests execute calls to the driver, pass parameters and structures, and get results back. With the sequence of calls demonstrated, the functions of the hardware are utilized for loop-back testing. The software is used for manufacturing test at Dynamic Engineering. For example most Dynamic Engineering PCI based designs support DMA. DMA is demonstrated with the memory based loop-back tests. The tests can be ported and modified to fit your requirements.

The test software can be ported to your application to provide a running start. It is recommended to port the switch and status tests to your application to get started. The tests are simple and will quickly demonstrate the end-to-end operation of your application making calls to the driver and interacting with the hardware.

The menu allows the user to add tests, to run sequences of tests, to run until a failure occurs and stop or to continue, to program a set number of loops to execute and more. The user can add tests to the provided test suite to try out application ideas before committing to your system configuration. In many cases the test configuration will allow faster debugging in a more controlled environment before integrating with the rest of the system.

The hardware has features common to the board level and features that are set apart in "channels". The channels have the same offsets within the channel, and the same status and control bit locations allowing for symmetrical software in the calling routines. The driver supports the channels with a variable passed in to identify which channel is being accessed. The hardware manual defines the pinouts for each channel and the bitmaps and detailed configurations for each channel. The driver handles all aspects of

interacting with the channels and base features.

We strive to make a useable product, and while we can guarantee operation we can't foresee all concepts for client implementation. If you have suggestions for extended features, special calls for particular set-ups or whatever please share them with us, [engineering@dyneng.com] and we will consider and in many cases add them.

ccPMC Parallel TTL BA18 utilizes a Xilinx Spartan3 FPGA to implement the PCI interface, FIFO's and protocol control and status for 32 IO plus 8 ADC channels each with DMA. Each IO can be programmed to be an output or an input at any time. Each IO can have rising edge, falling edge or COS processing enabled. In addition the BA18 version has eight ADC channels each with independent DMA, clock selection, data reduction and other SW controlled options. The driver supports programming a programmable PLL. Channel A and Channel B of the PLL are currently used for controlling the COS and ADC frequencies. Please refer to the HW manual for the specifics.

When the PMC Parallel TTL BA18 board is recognized by the PCI bus configuration utility it will start the PmcParTtlBA18Base driver which will create a device object for each board, initialize the hardware, create child devices for the eight ADC channels and request loading of the PmcParTtlBA18Chan driver. The PmcParTtlBA18Chan driver will create a device object for each of the ADC channels and perform initialization on each channel. IO Control calls (IOCTLs) are used to configure the board and read status. Read and Write calls are used to move blocks of data in and out of the device.

Note

This documentation will provide information about all calls made to the drivers, and how the drivers interact with the device for each of these calls. For more detailed information on the hardware implementation, refer to the ccPMC Parallel TTL BA18 user manual (also referred to as the hardware manual).

Driver Installation

There are several files provided in each driver package. These files include driver: BA18Base.sys, PMCBA18.inf, DDBA18Base.h, BA18BaseGUID.h, BA18Chan.sys, DDBA18Chan.h, BA18ChanGUID.h. Driver Test: BA18Test.exe, Userap: User Application source files.

BA18BaseGUID.h and BA18ChanGUID.h are C header files that define the device interface identifiers for the drivers. DDBA18Base.h and DDBA18Chan.h files are C header files that define the Application Program Interface (API) to the drivers. These files are required at compile time by any application that wishes to interface with the drivers, but they are not needed for driver installation.

BA18Test.exe is a sample Win32 console applications that makes calls into the BA18Base/BA18Chan drivers to test each driver call without actually writing any application code. They are not required during driver installation either.

To run BA18Test, open a command prompt console window and type ***BA18Test -d0 -?*** to display a list of commands (the PmcParTtlBA18Test.exe file must be in the directory that the window is referencing). The commands are all of the form ***BA18Test -dn -im*** where ***n*** and ***m*** are the device number and PmcParTtlBA18Base driver ioctl number respectively or ***BA18Test -cn -im*** where ***n*** and ***m*** are the channel number (0-1) and PmcParTtlBA18Chan driver ioctl number respectively.

This test application is intended to test the proper functioning of each driver call, **not** for normal operation. Many integration efforts will never need the debugger capability that the test menu represents. The test capability will allow the designer to access the card without any other software in the way to make sure that the system can “see” the card and to do basic card manipulations.

Windows 2000 Installation

Copy PmcBA18.inf, BA18Base.sys and BA18Chan.sys to a floppy disk, or CD if preferred. In some cases the files can be accessed over a network or from local HDD. Substitute the network address for the floppy instructions to proceed with an over the network installation.

With the hardware installed, power-on the PCI host computer and wait for the ***Found New Hardware Wizard*** dialogue window to appear.
_ Select ***Next***.
_ Select ***Search for a suitable driver for my device.***
_ Select ***Next***.
_ Insert the disk prepared above in the desired drive.
_ Select the appropriate drive e.g. ***Floppy disk drives***.
_ Select ***Next***.
_ The wizard should find the PmcBA18.inf file.
_ Select ***Next***.
_ Select ***Finish*** to close the ***Found New Hardware Wizard***.
The system should now see the channels and reopen the ***New Hardware Wizard.*** Repeat this for each channel as necessary.

Windows XP Installation

Copy PmcBA18.inf, BA18Base.sys and BA18Chan.sys to a floppy disk, or CD if preferred. In some cases the files can be accessed over a network or from local HDD. Substitute the network address for the floppy instructions to proceed with an over the network installation.

With the hardware installed, power-on the PCI host computer and wait for the ***Found New Hardware Wizard*** dialogue window to appear.
_ Insert the disk prepared above in the desired drive.
_ Select ***No when asked to connect to Windows Update***.
_ Select ***Next***.
_ Select ***Install the software automatically.***
_ Select ***Next***.
_ Select ***Finish*** to close the ***Found New Hardware Wizard***.
The system should now see the channels and reopen the ***New Hardware Wizard.*** Proceed as above for each channel as necessary.

Driver Startup

Once the drivers have been installed they will start automatically when the system recognizes the hardware.

Handles can be opened to a specific board by using the CreateFile() function call and passing in the device names obtained from the system.

The interfaces to the devices are identified using globally unique identifiers (GUIDs), which are defined in BA18BaseGUID.h and BA18ChanGUID.h.

The User Application software contains a file called “main.c”. Main has the initialization needed to get the handles to the base and channel assets of the installed ccPMC Parallel TTL BA18 device.

The main file provided is designed to work with our test menu and includes user interaction steps to allow the user to select which board is being tested in a multiple board environment. The integrator can hardcode for single board systems or use an automatic loop to operate in multiple board systems without using user interaction. For multiple user systems it is suggested that the board number is associated with a switch setting so the calls can be associated with a particular board from a physical point of view.

IO Controls

The drivers use IO Control calls (IOCTLs) to configure the device. IOCTLs refer to a single Device Object, which controls a single board or I/O channel. IOCTLs are called using the Win32 function DeviceIoControl() (see below), and passing in the handle to the device opened with CreateFile() (see above). IOCTLs generally have input parameters, output parameters, or both. Often a custom structure is used.

```
BOOL DeviceIoControl(
  HANDLE        hDevice,          // Handle opened with
CreateFile()
  DWORD         dwIoControlCode, // Control code defined in API
header file
  LPVOID        lpInBuffer,       // Pointer to input parameter
  DWORD         nInBufferSize,    // Size of input parameter
  LPVOID        lpOutBuffer,      // Pointer to output parameter
  DWORD         nOutBufferSize,   // Size of output parameter
  LPDWORD       lpBytesReturned, // Pointer to return length
parameter
  LPOVERLAPPED lpOverlapped,     // Optional pointer to
overlapped structure
);                                 // used for asynchronous I/O
```

The IOCTLs defined for the BA18Base driver are described below:

IOCTL_BA18_BASE_GET_INFO

Function: Return the Instance Number, Switch value, PLL device ID, Xilinx rev and Current Driver Version

Input: None

Output: BA18_BASE_DRIVER_DEVICE_INFO : Structure

Notes: SwitchValue is the configuration of the on-board dip-switch that has been set by the User (see the board silk screen for bit position and polarity).

The PllDeviceId is the device address of the PLL device. This value, which is set at the factory[Cypress], is usually 0x69 but may also be 0x6A.

The DriverVersion can be used for configuration control.

The XilinxVersion reflects the revision of the FLASH installed and can also be used for configuration control.

The InstanceNumber is the number of the device – when more than one card is used in the same system multiple “instances” are created. The instance number coupled with the Switch information can be used to deterministically select the correct device in the

user application code. When only one device per system this is not an issue. See DDBA18Base.h for the definition of BA18_BASE_DRIVER_DEVICE_INFO

IOCTL_BA18_BASE_LOAD_PLL_DATA
Function: Loads the internal registers of the PLL.
Input: BA18_BASE_PLL_DATA structure
Output: None
Notes: After the PLL has been configured, the register array data is analyzed to determine the programmed frequencies, and the IO clock A-D initial divisor fields in the base control register are automatically updated.

IOCTL_BA18_BASE_READ_PLL_DATA
Function: Returns the contents of the PLL's internal registers
Input: None
Output: BA18_BASE_PLL_DATA structure
Notes: The register data is output in the BA18_BASE_PLL_DATA structure in an array of 40 bytes.

The user reference files supplied include utilities to convert the Cypress tool supplied files into a programmable format. Please refer to XLATE.c and PLL_IF.c

IOCTL_BA18_BASE_SET_DIRU
Function: Write to Direction Register Upper IO 63-32
Input: ULONG
Output: none
Notes: 0 = Rx, 1 = Tx for each bit.

IOCTL_BA18_BASE_GET_DIRU
Function: Read From Direction Register Upper IO 63-32
Input: none
Output: ULONG
Notes: 0 = Rx, 1 = Tx for each bit.

IOCTL_BA18_BASE_SET_DATU
Function: Write to Data Register Lower bits 63-32
Input: ULONG
Output: none
Notes: Bits written to register will go to IO if Parallel Enable bit is set and IO type is set to registered [DRL and DATEN IOCTLs]

IOCTL_BA18_BASE_GET_DATU
Function: Read from Data IO Lower bits 63-32
Input: none
Output: ULONG
Notes: IO lines are read-back not register value – may or may not match register

IOCTL_BA18_BASE_GET_DATUREG
Function: Read from Data Register Upper bits 63-32
Input: none
Output: ULONG
Notes: SET DATU Register data read-back.

IOCTL_BA18_BASE_SET_COSCLK
Function: Write to COS clock register
Input: short on ULONG boundary
Output: none
Notes: The bit defines are in DDBA18Base.h. Please note that the COS clock can be driven to an IO pin to verify frequency with a scope. Please see the HW manual for more information on the usage of the bits. Please note that the HW manual calls this register COS Control. The name for the SW has been brought forward from the BA16 design which was the precursor to the BA18.

IOCTL_BA18_BASE_GET_COSCLK
Function: Read from COS clock register
Input: none
Output: short on ULONG boundary
Notes: reading provides the current values in the COS Clock definition register with no side affects – can read at any time without affecting the clock.

IOCTL_BA18_BASE_SET_RISUREG
Function: Write to COS Rising Upper Register bit enables 63-32
Input: ULONG
Output: none
Notes: Select which bits are tested for rising edge activity.

IOCTL_BA18_BASE_GET_RISUREG
Function: Read from COS Rising Upper Register bit enables 63-32
Input: none
Output: ULONG
Notes: no side affects from reading

IOCTL_BA18_BASE_SET_FALLUREG
Function: Write to COS Falling Upper Register bit enables 63-32
Input: ULONG
Output: none
Notes: Select which bits are tested for Falling edge activity.

IOCTL_BA18_BASE_GET_FALLUREG
Function: Read from COS Falling Upper Register bit enables 63-32
Input: none
Output: ULONG
Notes: no side affects from reading

IOCTL_BA18_BASE_SET_INTRISUREG
Function: Write to COS Interrupt Rising Upper Register Interrupt Enables 63-32
Input: ULONG
Output: none
Notes: Enable the interrupt corresponding to the rising COS status for each bit. Not setting the interrupt will allow polled operation using the status.

IOCTL_BA18_BASE_GET_INTRISUREG
Function: Read from COS Interrupt Rising Upper Register Interrupt Enables 63-32
Input: none
Output: ULONG
Notes: no side affects from reading

IOCTL_BA18_BASE_SET_INTFALLUREG
Function: Write to COS Interrupt Falling Upper Register Interrupt Enables 63-32
Input: ULONG
Output: none
Notes: Enable the interrupt corresponding to the Falling COS status for each bit. Not setting the interrupt will allow polled operation using the status.

IOCTL_BA18_BASE_GET_INTFALLUREG
Function: Read from COS Interrupt Falling Upper Register Interrupt Enables 63-32
Input: none
Output: ULONG
Notes: no side affects from reading

IOCTL_BA18_BASE_CLR_INTRISUSTAT
Function: Write to COS Interrupt Rising Status Upper Register Interrupt Status
Input: ULONG
Output: none
Notes: Writing to the Interrupt Status register will clear the interrupts on a bit by bit basis.

IOCTL_BA18_BASE_GET_INTRISUSTAT
Function: Read from COS Interrupt Rising Status Upper Register Interrupt Status
Input: none
Output: ULONG
Notes: Read the status register to see which bits have been set indicating that a rising event has occurred for a programmed bit. It is recommended that the Interrupt status is cleared each time the enabled bits are changed. Writing back the data read will clear only the bits that the SW has registered as interrupts and will prevent missing interrupt events.

IOCTL_BA18_BASE_CLR_INTFALLUSTAT
Function: Write to COS Interrupt Falling Status Upper Register Interrupt Status
Input: ULONG
Output: none
Notes: Writing to the Interrupt Status register will clear the interrupts on a bit by bit basis.

IOCTL_BA18_BASE_GET_INTFALLUSTAT
Function: Read from COS Interrupt Falling Status Upper Register Interrupt Status
Input: none
Output: ULONG
Notes: Read the status register to see which bits have been set indicating that a falling event has occurred for a programmed bit. It is recommended that the Interrupt status is cleared each time the enabled bits are changed. Writing back the data read will clear only the bits that the SW has registered as interrupts and will prevent missing interrupt events.

IOCTL_BA18_BASE_SET_BASEREG
Function: Write to Base Control Register - general access to base control register of card, use with bit definitions
Input: ULONG
Output: none
Notes: Use for general purpose – bit mapped access to the base control register.

IOCTL_BA18_BASE_GET_BASEREG
Function: Read from Base Control Register - general access from base control register

of card, use with bit definitions
Input: none
Output: ULONG
Notes: Use for general purpose – bit mapped access to the base control register.

IOCTL_BA18_BASE_GET_STATUS
Function: Read from Status Register
Input: none
Output: ULONG
Notes: Use for general purpose – bit mapped access from the register. See DDBA1Base.h for bit map information. See the HW manual for exact definitions of bits.

IOCTL_BA18_BASE_SET_MASTEREN
Function: read base register then set master interrupt enable bit - read modify write to set master interrupt enable
Input: none
Output: updated Base Register contents
Notes: The Master Interrupt enable is needed to allow interrupts from some sources to be asserted. Please refer to the HW manual for details.

IOCTL_BA18_BASE_CLR_MASTEREN
Function: read base register then clear master interrupt enable bit - read mod write to clear master interrupt enable
Input: none
Output: updated Base Register contents
Notes: The Master Interrupt enable is needed to allow interrupts from some sources to Be asserted onto the bus. The Clr function can be used to disable some board level interrupt sources.

IOCTL_BA18_BASE_SET_TIMESTAMP
Function: Set Pre-load register with initial TimeStamp value
Input: ULONG
Output:
Notes: The TimeStamp is a 32 bit counter. Values less than 0 can be written to allow for other system delays. Normally the count is started from 0x00.

IOCTL_BA18_BASE_GET_TIMESTAMP
Function: Read back the pre-load value
Input: none
Output: ULONG
Notes:

IOCTL_BA18_BASE_GET_TIMESTAMPCNT
Function: Read back the current TimeStamp count
Input: none
Output: ULONG
Notes: The count can be read in real time and used to see if the counter is progressing, and can be used as a timer if desired.

IOCTL_BA18_BASE_SET_COS_STATUS
Function: Write to the COS Status register
Input: ULONG
Output:
Notes: Use to clear the sticky bits within the COS Status register.

IOCTL_BA18_BASE_GET_COS_STATUS
Function: Read back the current COS Status
Input: none
Output: ULONG
Notes: no side affects, Sticky bits will require explict clearing.

IOCTL_BA18_BASE_SET_COS_FIFO_PAF
Function: Set the Programmable Almost Full register
Input: ULONG
Output:
Notes: The PAF value is used to compare against the COS FIFO data count. When the COS FIFO has the same or more data than programmed the PAF interrupt can be generated to alert the user software. The Almost Full value can also be polled via the status register.

IOCTL_BA18_BASE_GET_COS_FIFO_PAF
Function: Read back the PAF register
Input: none
Output: ULONG
Notes:

IOCTL_BA18_BASE_GET_COS_FIFO_WORD_CNT
Function: Read the FIFO data stored count
Input: none
Output: ULONG
Notes: no side affects from reading the count

IOCTL_BA18_BASE_GET_COS_FIFO_DATA
Function: Read the FIFO data
Input: none
Output: ULONG
Notes: Once read Data is no longer in FIFO

The IOCTLs defined for the PmcParTtlBA18Chan driver are described below:

IOCTL_BA18_CHAN_GET_INFO
Function: Return the Instance Number and Current Driver Version
Input: None
Output: BA18_CHAN_DRIVER_DEVICE_INFO structure
Notes: See the definition of BA18_CHAN_DRIVER_DEVICE_INFO in the DDBA18Chan.h header file.

IOCTL_BA18_CHAN_GET_STATUS
Function: Return the value of the status register and clear latched bits
Input: None
Output: Status register value(ULONG)
Notes: Latched interrupt status bits are cleared by read – [call writes back and clears bits]. Defines available in DDBA18Chan.h Detailed definitions are available in the HW manual.

IOCTL_BA18_CHAN_GET_FIFO_COUNTS
Function: Returns the number of data words in ADC FIFO.
Input: None
Output: ULONG
Notes: Returns the FIFO data count plus the DMA pipeline. Use this count when doing discrete reads to know how many samples can be moved, or to do a final DMA transfer to empty the memory once the process is stopped.

IOCTL_BA18_CHAN_RESET_FIFOS
Function: Resets one or both FIFOs for the referenced channel.
Input: ULONG [any value]
Output: None
Notes: Resets FIFO.

IOCTL_BA18_CHAN_REGISTER_EVENT
Function: Registers an event to be signaled when an interrupt occurs.
Input: Handle to the Event object
Output: None
Notes: The caller creates an event with CreateEvent() and supplies the handle returned from that call as the input to this IOCTL. The driver then obtains a system pointer to the event and signals the event when a user interrupt is serviced. The user interrupt service routine waits on this event, allowing it to respond to the interrupt. The DMA interrupts do not cause the event to be signaled.

IOCTL_BA18_CHAN_ENABLE_INTERRUPT

Function: Enables the channel Master Interrupt.
Input: None
Output: None
Notes: This command must be run to allow the board to respond to user interrupts. The master interrupt enable is disabled in the driver interrupt service routine when a user interrupt is serviced. Therefore this command must be run after each interrupt occurs to re-enable it.

IOCTL_BA18_CHAN_DISABLE_INTERRUPT

Function: Disables the channel Master Interrupt.
Input: None
Output: None
Notes: This call is used when user interrupt processing is no longer desired.

IOCTL_BA18_CHAN_FORCE_INTERRUPT

Function: Causes a system interrupt to occur.
Input: None
Output: None
Notes: Causes an interrupt to be asserted on the PCI bus as long as the channel master interrupt is enabled. This IOCTL is used for development, to test interrupt processing.

IOCTL_BA18_CHAN_GET_ISR_STATUS

Function: Returns the interrupt status read in the ISR from the last user interrupt.
Input: None
Output: Interrupt status value (unsigned long integer)
Notes: Returns the interrupt status that was read in the interrupt service routine of the last interrupt caused by one of the enabled channel interrupts. The interrupts that deal with the DMA transfers do not affect this value. Masked version of channel status. This version should be read when dealing with interrupts due to the sticky bit status.

IOCTL_BA18_CHAN_SWR_RX_FIFO

Function: Returns a 32-bit data word from the ADC FIFO.
Input: None
Output: FIFO word (unsigned long integer)
Notes: Used to make single-word accesses to the ADC FIFO instead of using DMA. Please note that values read are no longer available in the FIFO.

IOCTL_BA18_CHAN_SET_CONT
Function: write to Channel Control register using structure
Input: BA18_CHAN_CONT
Output: None
Notes: See DDBA18Chan.h for structure. See below for quick reference.

IOCTL_BA18_CHAN_GET_CONT
Function: Read from Channel Control register using structure
Input: None
Output: BA18_CHAN_CONT
Notes: See DDBA18Chan.h for structure. See below for quick reference.

```
MIntEn;       // Master Interrupt Enable
RdDmaEn;      // Read DMA Interrupt Enable
AdcFifoRst    // set to reset FIFO
OutUrgent     // set to raise priority of this channel in DMA processing
IntForce      // Force Int function for this channel, use for Interrupt testing
```

IOCTL_BA18_CHAN_SET_FIFO_LEVELS
Function: Sets the ADC FIFO almost full level for the channel.
Input: ULONG
Output: None
Notes: The FIFO counts are compared to this level to determine the value of the STAT_RX_FF_AFL status bits, and if enabled the AFL interrupt associated with the status.

IOCTL_BA18_CHAN_GET_FIFO_LEVELS
Function: Returns the ADC almost full level for the channel.
Input: None
Output: ULONG
Notes:

IOCTL_BA18_CHAN_SET_ADC_CONT
Function: write to Channel ADC Control register using structure
Input: BA18_CHAN_ADC_CONT
Output: None
Notes: See DDBA18Chan.h for structure. See below for quick reference.

IOCTL_BA18_CHAN_GET_ADC_CONT
Function: Read from Channel ADC Control register using structure
Input: None
Output: BA18_CHAN_ADC_CONT
Notes: See DDBA18Chan.h for structure. See below for quick reference.

```
AdcClkSel;          // Set for COS Clock, Clr for PLLB
AdcFifoDataSel;     // Set for Alternate Data processing, clear for double packed data
AdcLocalEn;         // Set to enable data collection.  Set Clock and method first
AdcFifoIntEn;       // set to enable interrupts from the programmed AFL level
AdcFifoLatchIntEn;  // set to enable interrupt from the programmed latched AFL level
AdcOverFlowIntEn;   // set to enable interrupt from the FIFO OverFlow condition
AdcClockDiv;        // Count to use for 2(n+1) divisor in channel, need to select  local
                    PLLB.  8 bit value
```

IOCTL_BA18_CHAN_SET_ADC_STATUS
Function: write to Channel ADC Status register to clear sticky bits
Input: ULONG
Output: None
Notes: Some of the status bits are held – for transitory signals that might not be detected when read otherwise. These bits are cleared with an explicit write back to the status register with the same bit(s) set.

IOCTL_BA18_CHAN_SET_ADC_TOLERANCE
Function: write to Channel ADC Tolerance register using structure
Input: BA18_CHAN_ADC_TOLERANCE
Output: None
Notes: See DDBA18Chan.h for structure. See below for quick reference.

IOCTL_BA18_CHAN_GET_ADC_TOLERANCE
Function: Read from Channel ADC Tolerance register using structure
Input: None
Output: BA18_CHAN_ADC_TOLERANCE
Notes: See DDBA18Chan.h for structure. See below for quick reference.

```
AdcToleranceLow;   // 12 bits, subtracted from current reference value for lower bounds
test
AdcToleranceHigh;  // 12 bits, added to current reference value for upper bounds test
```

Write

DMA data is written to the referenced I/O channel device using the write command. Writes are executed using the Win32 function WriteFile() and passing in the handle to the I/O channel device opened with CreateFile(), a pointer to a pre-allocated buffer containing the data to be written, an unsigned long integer that represents the size of that buffer in bytes, a pointer to an unsigned long integer to contain the number of bytes actually written, and a pointer to an optional Overlapped structure for performing asynchronous IO.

Read

DMA data is read from the referenced I/O channel device using the read command. Reads are executed using the Win32 function ReadFile() and passing in the handle to the I/O channel device opened with CreateFile(), a pointer to a pre-allocated buffer that will contain the data read, an unsigned long integer that represents the size of that buffer in bytes, a pointer to an unsigned long integer to contain the number of bytes actually read, and a pointer to an optional Overlapped structure for performing asynchronous IO.

Warranty and Repair

Dynamic Engineering warrants this product to be free from defects under normal use and service and in its original, unmodified condition, for a period of one year from the time of purchase. If the product is found to be defective within the terms of this warranty, Dynamic Engineering's sole responsibility shall be to repair, or at Dynamic Engineering's sole option to replace, the defective product.

Dynamic Engineering's warranty of and liability for defective products is limited to that set forth herein. Dynamic Engineering disclaims and excludes all other product warranties and product liability, expressed or implied, including but not limited to any implied warranties of merchandisability or fitness for a particular purpose or use, liability for negligence in manufacture or shipment of product, liability for injury to persons or property, or for any incidental or consequential damages.

Dynamic Engineering's products are not authorized for use as critical components in life support devices or systems without the express written approval of the president of Dynamic Engineering.

Service Policy

Before returning a product for repair, verify as well as possible that the driver is at fault. The driver has gone through extensive testing and in most cases it will be “cockpit error” rather than an error with the driver. When you are sure or at least willing to pay to have someone help then call the Customer Service Department and arrange to speak with an engineer. We will work with you to determine the cause of the issue. If the issue is one of a defective driver we will correct the problem and provide an updated module(s) to you [no cost]. If the issue is of the customer’s making [anything that is not the driver] the engineering time will be invoiced to the customer. Pre-approval may be required in some cases depending on the customer’s invoicing policy.

Out of Warranty Repairs

Out of warranty support will be billed. The current minimum repair charge is $125. An open PO will be required.

For Service Contact:

Customer Service Department
Dynamic Engineering
150 DuBois Street, Suite C
Santa Cruz, CA 95060
831-457-8891
831-457-4793 Fax

support@dyneng.com