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 *DynamicSignals / Signatec PDA500-1M* at our website: *Click HERE*



PDA500

Operator's Manual

Signatec, Incorporated

Corona, California, USA

Copyright (c) 1999

All Rights Reserved

Revised March 26, 2001

TABLE OF CONTENTS

1 BEFORE USING THE PDA500!! 1

1.1 Package Contents 1
1.2 Unpacking and Handling 1
1.3 Checking for Damage 1
1.4 Warranty 1
1.5 System Requirements 1
1.6 Software 2
1.7 Physical Layout 2
1.8 DIP Switch Settings 4
1.8.1 Setting the Input Signal Coupling 4
1.8.2 Low Pass Filter Selection 4
1.9 Jumper Settings 4
1.9.1 Soldered Jumpers 4
1.9.2 Jumper Bars 5
1.9.2.1 External Trigger Selection 6
1.9.2.2 Synchronized Output Trigger 6
1.9.2.3 External Clock Selection 6
1.10 Board Addresses 7

2 FUNCTIONAL DESCRIPTION 8

2.1 Overview 8
2.2 Accessing the Data Registers 10
2.3 Analog Input Circuitry 10
2.3.1 Input Coupling 10
2.3.2 Analog Signal Conditioning 10
2.3.3 Analog Bandwidth 11
2.3.4 Calibration 11
2.3.5 DC Offset Voltage Adjustment 11
2.4 ADC and Demultiplexer FIFOs 11
2.5 Memory Addressing 12
2.5.1 Memory Mapped Data Transfers 12
2.5.2 DMA Data Transfers 12
2.6 Clock Source 13
2.6.1 Clock Divider 13
2.7 Operating Modes 13
2.7.1 Off Mode 14
2.7.2 PCI Acquisition 14
2.7.3 RAM Acquisition Mode 14
2.7.4 SAB Acquisition Mode 15
2.7.5 PCI Transfer Mode 15
2.7.6 SAB Transfer Mode 15
2.7.7 Write RAM Mode 15
2.7.8 Standby Mode 16
2.8 Trigger Modes and Methods 16
2.8.1 Trigger Modes 16
2.8.1.1 Single Shot Mode 16
2.8.1.2 Segmented Mode 16
2.8.2 Trigger Methods 16
2.8.2.1 Pre-trigger 16
2.8.2.2 Delayed Trigger 17
2.8.3 Trigger Source 17
2.8.4 Trigger Slope 17
2.8.5 Trigger Level 17
2.8.6 Trigger Hysteresis 17
2.8.7 Software Trigger 18

2.9 Memory Size Selections 18
2.9.1 Starting Address 18
2.9.2 Ending Address 18
2.9.3 Segment Size 18
2.9.4 Pre-trigger Samples 19
2.10 PCI Interface 19
2.10.1 Plug and Play Features 19
2.10.1.1 I/O Addresses 20
2.10.1.2 Memory Space 20
2.10.2 DMA Transfers 20
2.11 Signatec Auxiliary Bus Interface 21
2.11.1 PDA500 SAB Data Transfer 21
2.11.1.1 Block Transfer 22
2.11.1.2 Packetized Transfer 22
2.11.2 PDA500 SAB Control 22
2.12 Thermal Management and Power Down Features 22
2.12.1 Off Mode 23
2.12.2 Automatic Power Control 23
2.12.3 Power Down Override 23
2.12.4 Thermal Shutdown 23
2.13 Master-Slave Operation 23
2.14 PDA500 Verses DA500A Hardware 24
2.14.1 DA500A Hardware Removed 24
2.14.1.1 250MHz Clock Source 24
2.14.1.2 Interleaved Data Mode 24
2.14.2 DA500A Hardware Changed 24
2.14.2.1 Signal Coupling 24
2.14.2.2 Pre-trigger Acquisition 24
2.14.2.3 Power Modes 25
2.14.2.4 Signatec Auxiliary Bus (SAB) Version 3 25
2.14.3 PDA500 New Features 25
2.14.3.1 Analog Front End Protection 25
2.14.3.2 Delayed Trigger 25
2.14.3.3 Clock and Trigger Output 26

3 PDA500 SOFTWARE 27

3.1 Introduction 27
3.1.1 Software for DOS 27
3.1.2 Software for Windows 27
3.2 Installing Software 28
3.2.1 DOS Installation 28
3.2.2 Windows Installation 28
3.2.3 Windows 95 Installation 29
3.2.4 Windows 2000 Installation 30
3.3 Global Variables 32
3.3.1 Addressing Multiple PDA500 Boards 32
3.3.1.1 Addressing in DOS 32
3.3.1.2 Addressing in Windows 32
3.3.2 PDA500 Control Registers 33
3.3.2.1 PDA500 Control Registers in DOS 33
3.3.2.2 PDA500 Control Registers in Windows 33
3.3.3 Miscellaneous Variables 34
3.3.3.1 Mode Variables 34
3.3.3.2 Parameter Variables 34
3.3.3.3 Memory Control Variables 34
3.3.3.4 Software Register Contents 37
3.3.4 Data Acquisition Techniques 37
3.3.4.1 Data Acquisition for Multiple PDA500 Boards 37
3.4 Programming PDA500 Verses DA500A 38
3.4.1 DOS 38
3.4.2 Windows 38

3.5 PDA500 C LIBRARY FUNCTIONS 39
3.5.1 acqcomp500 41
3.5.2 acqPCIdata500 42
3.5.3 acqPCIdatabuffers500 43
3.5.4 acqRAMdata500 45
3.5.5 acqRAMdataDMA500 46
3.5.6 acqSABdata500 47
3.5.7 allocatedmabuffer500 48
3.5.8 allocatedmabuffers500 49
3.5.9 attnset500 50
3.5.10 autoaddrst500 51
3.5.11 brd2disk500 52
3.5.12 brd2pc500 53
3.5.13 brd2pcdma500 54
3.5.14 brdname500 57
3.5.15 brdnum500 57
3.5.16 cancelACFint500 58
3.5.17 chanset500 60
3.5.18 clkdivbyfive500 61
3.5.19 clkdivide500 62
3.5.20 clkfreq500 64
3.5.21 clock500 65
3.5.22 closedevice500 67
3.5.23 disk2brd500 68
3.5.24 disk2pc500 69
3.5.25 dmaint500 70
3.5.26 enableACFint500 70
3.5.27 enabledma500 71
3.5.28 enablepcint500 71
3.5.29 fftstats500 72
3.5.30 fifoflag500 73
3.5.31 findbrd500 74
3.5.32 freeallbuffers500 75
3.5.33 freedmabuffer500 75
3.5.34 freedmabuffers500 76
3.5.35 hyst500 76
3.5.36 Iobase500 77
3.5.37 membase500 78
3.5.38 mode500 79
3.5.39 nulloff500 80
3.5.40 offset500 81
3.5.41 opendevice500 82
3.5.42 presamples500 83
3.5.43 pwdownor500 84
3.5.44 pwrdy500 85
3.5.45 readEPROM500 86
3.5.46 readreg500 87
3.5.47 readset500 87
3.5.48 realfft500 89
3.5.49 realfftmag500 91
3.5.50 SABbrdnum500 93
3.5.51 SABclk500 94
3.5.52 SAB1compatible500 95
3.5.53 SABconf500 96
3.5.54 SABinit500 97
3.5.55 segmentdetect500 99
3.5.56 segmentsize500 99
3.5.57 sendEOI 101

3.5.58 setendadd500 101
3.5.59 setstartadd500 103
3.5.60 setupbrd500 104
3.5.61 setuppciint500 104
3.5.62 setupregisters500 105
3.5.63 temperature500 106
3.5.64 trig500 107
3.5.65 trigdelay500 108
3.5.66 triglevel500 109
3.5.67 trigmode500 110
3.5.68 trigslope500 111
3.5.69 trigsoft500 112
3.5.70 trigsource500 112
3.5.71 uninstallpciint500 114
3.5.72 voltrange500 114
3.5.73 waitforACFint500 115
3.5.74 writeEPROM500 116
3.5.75 writeram500 116
3.5.76 writereg500 118
3.5.77 writeregs500 119
3.5.78 writeset500 120

4 APPENDIX A - PDA500 SPECIFICATIONS AND ORDERING INFORMATION 122

5 APPENDIX B - HARDWARE REGISTERS 123

6 APPENDIX C - HARDWARE REGISTERS 125

7 APPENDIX D - MASTER-SLAVE CONNECTIONS 126

8 APPENDIX E - SIGNATEC AUXILIARY BUS CONNECTIONS 127

9 APPENDIX F - C LIBRARY TABLE 128

IMPORTANT NOTICE
ON
HARDWARE COMPATIBILITY

The PDA500 is a PCI Local Bus compliant device. As such the PDA500 contains the configuration space register organization as defined by the PCI Local Bus Specification. Among the functions of the configuration registers is the storage of unique identification values for the PDA500 as well as storage of base address size requirements for PDA500 operation.

The host computer that the PDA500 is installed in is responsible for reading and writing to/from the PCI configuration registers to enable proper operation. This functionality is referred to as 'Plug and Play' (PnP). As such, the host computer PnP BIOS must be capable of automatically identifying a PCI compliant device, determining the system resources required by the device, and assigning the necessary resources to the device. Failure of the host computer to execute any of these operations will prohibit the use of the PDA500 in such a system.

It has been determined that systems that implement PnP BIOS, and contain only fully compliant PnP boards and drivers, operate properly. However, systems that do not have a PnP BIOS installed, or contain hardware or software drivers which are not PnP compatible, may not successfully execute PnP initialization. This can render the PDA500 inoperable. It is beyond the ability of Signatec hardware or software to force a non-PnP system to operate a PDA500 board.

1 Before Using the PDA500!!

1.1 Package Contents

The PDA500 package shipped to you will normally contain (as a minimum) the following:

a. PDA500 circuit board assembly.

b. A binder containing the following:

1. PDA500 Operator's Manual
2. PDA500 DOS Disk
3. PDA500 Windows Disk
4. Calibration Table Disk

c. Two BNC to SMA coaxial cables.

1.2 Unpacking and Handling

The PDA500, and any other electronic circuit board assembly included in its shipment, is shipped in an anti-static bag. These circuit boards are extremely sensitive to static electricity which can damage sensitive electronic parts. The human body can build up a damaging amount of electrical charge especially in dry weather and in carpeted rooms. To avoid damaging the boards, rid yourself of any charge buildup by touching some large metal object which ideally is at earth potential. Also, only touch the circuit boards along the edges, carefully avoiding contact with the electronic circuitry.

1.3 Checking for Damage

Carefully inspect the circuit board assemblies for any sign of physical damage during shipment. Any such damage must be reported within 15 days from the date of actual shipment in order to be covered by warranty. To report such damage, call Signatec, explain the nature of the damage, and request a RMA number for returning the merchandise.

1.4 Warranty

All Signatec products carry a full 3-year warranty. During the warranty period, Signatec will repair or replace any defective product at no cost to the customer. This warranty does not cover physical damage not reported within 15 days of the time of shipment by Signatec. Call Signatec for a RMA number before returning any merchandise.

1.5 System Requirements

The PDA500 requires the following minimum hardware configuration:

For DOS,

IBM AT personal computer or compatible with at least one open PCI slot and Plug and Play BIOS.
MSDOS (or PCDOS) version 2.0 or higher.

For Windows 95,

386/486/Pentium or higher CPU with at least 8MB RAM with at least one open PCI slot and Plug and Play BIOS.
Windows 95/98/98 SE.

For Windows NT,

386/486/Pentium or higher CPU with at least 16MB RAM with at least one open PCI slot and Plug and Play BIOS.
Windows NT 3.51/4.0.

For Windows 2000,

Pentium 133 or higher CPU with at least 64MB RAM with at least one open PCI slot and Plug and Play BIOS.
Windows 2000.

1.6 Software

The PDA500 is supplied with a set of software functions written in the C programming language that may be used in programs to simplify the job of creating custom applications. The source code is provided for all the functions so that they may be used with all C language compilers and may be modified to suit the user's needs. The functions used to create DOS applications have been compiled and tested with Borland's C++ 5.02. The Windows 95/NT/2000 functions have been tested and compiled using Microsoft's Visual C++ 6.0. Hereafter when Windows is mentioned it is assumed to mean Windows 95/NT/2000 unless otherwise noted. Windows 3.x is not supported. It is possible that some modification may be needed for other vendor's compilers. All names of functions in this manual are in *italics*. For more information on a function, please refer to the section on that specific function under section 3.5 (PDA500 C LIBRARY FUNCTIONS).

1.7 Physical Layout

The PDA500 is shown in Figure 1. Signals are input via the SMA connectors on the bracket as shown. The upper two connectors are for the channel 1 and channel 2 analog input signals, the third connector is the external clock connection, and the bottom connector is the external trigger connector.

At the top of the board, near the left-hand side, is the master-slave connector that is used to connect a master board to up to three slave boards. A slave board derives its clock and trigger signals from the master board so that data on the boards is completely synchronous (or aligned), even for large amounts of memory.

Along the right portion of the board are two 100 pin connectors J7 and J8. These connectors are used for the Signatec Auxiliary Bus (SAB3) interface that is capable of transferring data at a maximum rate of 500 megabytes per second. See section 2.11 for details of the SAB3 interface.

The PDA500 has a card edge connectors on the bottom which is compatible with 32 bit 5 volt Peripheral Component Interconnect (PCI) bus.

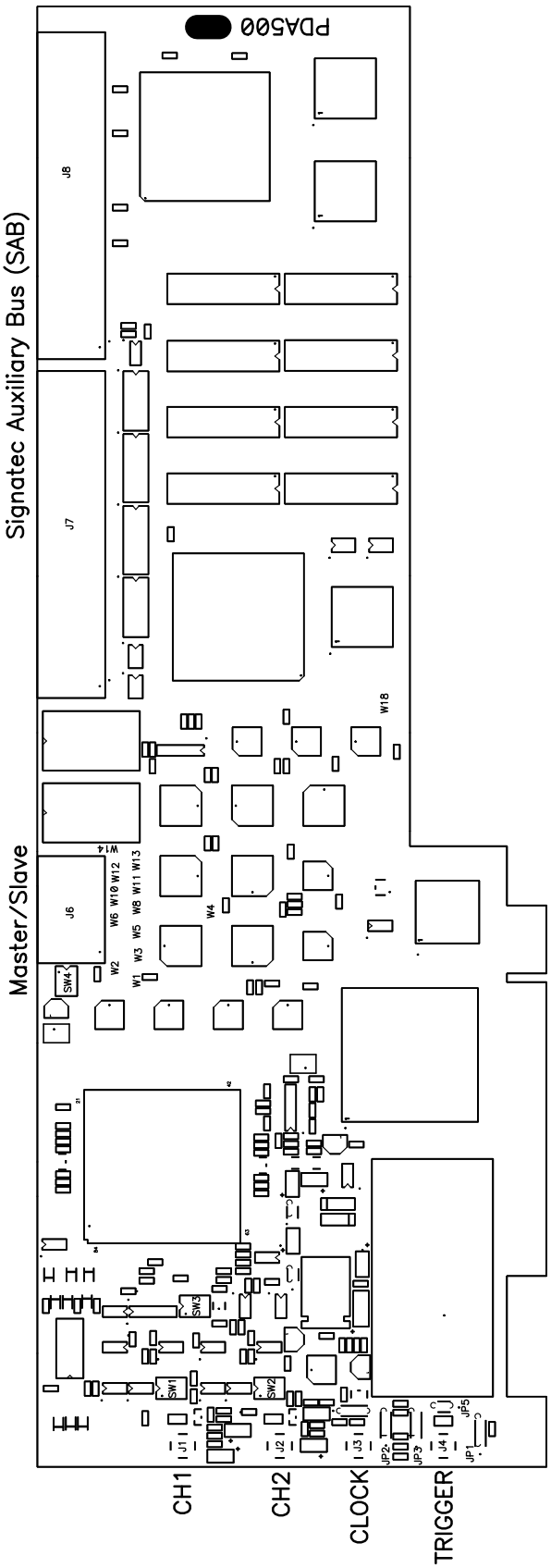

PDA500
Signatec Auxiliary Bus (SAB)
Master/Slave
CH1
CH2
CLOCK
TRIGGER

Figure 1

1.8 DIP Switch Settings

Four DIP switches are included on the board, three of which are likely to be changed by the user. These are designated on the board as SW1, SW2, and SW3. One additional DIP switch, SW4, located on the PDA500 board is used for clock phase matching of the Analog to Digital Converter (ADC) clock on the board. This switch is factory set on each individual board and generally will not be changed by the user.

1.8.1 Setting the Input Signal Coupling

The DIP switch SW1 selects the coupling of the input analog signals of Channel 1 as AC or DC coupling with an impedance of 50 ohms or 1 Meg ohms. DIP switch SW2 sets the selection for Channel 2. For each input channel, placing switch position number 4 in the "on" position selects DC coupling while placing switch position number 3 in the "on" position selects AC coupling. For maximum flexibility the PDA500 can be configured with an input impedance of 50-ohms or 1Meg-ohms. Placing switch position number 2 in the "on" position sets the input impedance to 50-ohms, while placing switch position number 1 in the "on" position sets the input impedance to 1 Meg-ohm (contact factory for availability). For all 4 switch positions on SW1 & SW2, moving the individual switch position to the right places the switch in the "on" mode, and moving the switch position to the left places the switch in the "off" mode. For both switches (SW1 & SW2), switch positions number 1 and number 2 should never be turned on at the same time. Likewise, positions 3 and 4 should never be turned on at the same time. For more information regarding input signal coupling see section 2.3 entitled Analog Input Circuitry. The selections for SW1 and SW2 are shown in the following table.

SW1	**Input Signal Coupling Channel 1**
Position 4: Ch1	On = DC Coupling (Default)
Position 3: Ch1	On = AC Coupling
Position 2: Ch1	On = 50 Ohm Impedance (Default)
Position 1: Ch1	On = 1 Meg Ohm Impedance (Contact Factory for Availability)
SW2	**Input Signal Coupling Channel 2**
Position 4: Ch2	On = DC Coupling (Default)
Position 3: Ch2	On = AC Coupling
Position 2: Ch2	On = 50 Ohm Impedance (Default)
Position 1: Ch2	On = 1 Meg Ohm Impedance (Contact Factory for Availability)
Switch Modes	Right = ON/ Left = OFF
Positions 1 and 2 should not be on at the same time.	
Positions 3 and 4 should not be on at the same time.	

1.8.2 Low Pass Filter Selection

The DIP switch SW3 sets the analog bandwidth of the PDA500 boards. SW3 can be set for an AC coupled bandwidth of 40 kHz - 500MHz maximum or for 100MHz maximum bandwidth using an inline low pass filter. For a 500 MHz maximum bandwidth, set SW3 position 1 to ON and positions 2, 3, and 4 to OFF. For a 100MHz maximum analog bandwidth, set SW3 positions 1 and 4 to OFF and 2 and 3 to ON.

1.9 Jumper Settings

There are two types of jumpers on the PDA500, fixed soldered jumpers and movable jumper bars. These jumpers are used to configure the board for different operations.

1.9.1 Soldered Jumpers

Soldered jumpers are set at the factory and will not need to be changed. These jumpers are designated on the board with the letter 'W' such as W1, W2, etc. The functions of the jumpers are given in the following table. The

term Master Board can refer to either a board which is a Master and has at least one Slave board connected to it or a single board with no Slave boards connected. A Slave board is a board that is connected to another PDA500 which acts as the Master.

Designator	Function/Connection	Default Position
W1	Connect Pins 1 and 2: Master Board	Connect Pins 1 and 2 (Bottom)
W2	Connect Pins 1 and 2: Master Board	Connect Pins 1 and 2 (Bottom)
W3	Connect Pins 1 and 2: Master Board	Connect Pins 1 and 2 (Bottom)
W4	Connect Pins 1 and 2: Master Board Slave (All) : Not Connected	Connect Pins 1 and 2 (Right)
W5	Connect Pins 1 and 2: Slave Board #3 Connect Pins 2 and 3: Master Board	Connect Pins 2 and 3 (Bottom)
W6	Connect Pins 1 and 2: Slave Board #2 Connect Pins 2 and 3: Master Board	Connect Pins 2 and 3 (Bottom)
W7	Not Used	Not Used
W8	Connect Pins 1 and 2: Slave Board #1 Connect Pins 2 and 3: Master Board	Connect Pins 2 and 3 (Bottom)
W9	Connect Pins 1 and 2: Slave Board (All) Connect Pins 2 and 3: Master Board	Connect Pins 2 and 3 (Right)
W10	Connected: Master Board Not Connected: Slave Board (All)	Connected
W11	Connect Pins 1 and 2: Slave Board #3 Connect Pins 2 and 3: Master Board	Connect Pins 2 and 3 (Bottom)
W12	Connect Pins 1 and 2: Slave Board #2 Connect Pins 2 and 3: Master Board	Connect Pins 2 and 3 (Bottom)
W13	Connect Pins 1 and 2: Slave Board #1 Connect Pins 2 and 3: Master Board	Connect Pins 2 and 3 (Bottom)
W14	Connected: Master Board Not Connected: Slave Board	Connected
W15	Production Use Only	Connect Pins 1 and 2 (Bottom)
W16	Production Use Only	Not Connected
W17	Production Use Only	Not Connected
W18	Connected: Master Board Not Connected: Slave Board	Connected
W19	Production Use Only	Not Connected
W20	Production Use Only	Connect Pins 1 and 2 (Top)
W21	Connected: Non-SAB Version Not Connected: SAB Version	SAB Boards Connected Non-SAB Not Connected
W22	Production Use Only	Not Connected

1.9.2 Jumper Bars

Jumper bars are designated by the prefix 'JP' such as JP1. Jumper bars connect two pins together at each location. The functions of the jumper bars are given in the following sections.

1.9.2.1 External Trigger Selection

The jumpers JP2, JP3 and JP5 select the configuration of the external trigger connector. The external trigger connector can be configured with AC or DC coupling with an input impedance of 50 ohms or 1K ohms. The 1K ohm input impedance also divides the input by a factor of 10. With the default connection of pins 1 and 2, for JP2, JP3, and JP5, the external trigger is DC coupled with an impedance of 50 ohms. The jumper settings to select the various options are shown below. For information regarding a trigger output see section 1.9.2.2.

Jumper	External Trigger Selection
JP5: AC or DC Coupling	Connected = DC Coupling (Default) Not Connected = AC Coupling
JP3 and JP2: Impedance Selection	Connect pins 1 and 2 = 50 Ohm impedance (Default) Connect pins 2 and 3 = 1K Ohm impedance

1.9.2.2 Synchronized Output Trigger

A synchronized output trigger is available that can be used to virtually eliminate all "jitter" between an external trigger signal and the ADC clock. Under normal triggering from an asynchronous external source there will be a maximum of 2 clock cycles of "apparent jitter". This consists of 1 clock cycle of true, randon jitter plus one clock cycle of uncertainty due to the fact that that the internal mechanization of the PDA500 actually utilizes two interleaved, free running ADCs wherin ADC A must always capture the very first sample.

Supplying an external clock to the PDA500 along with an external trigger that is synchronized to that clock will eliminate the random jitter but will leave the one clock cycle of clock uncertainty. To eliminate the clock uncertainty it is necessary to input an external trigger to the PDA500 and use the synchronized output trigger as the trigger source for external devices. The Output trigger uses the "clock" connector on the board.

The Synchronized Output Trigger is an ECL trigger that should be terminated with a 100 ohm resistor to -2 volts. The active edge is always positive going.

ECL Trigger Out	SW6: Position 1 = Off, 2 = On, 3 = Off, 4 = Off; (Rev B Only) JP4: 2 & 3 ; JP1: 2 & 3; (Rev A Only)

1.9.2.3 External Clock Selection

Jumpers JP1 and JP4 select the configuration of the external clock connector: J3. The external clock connector can operate as a TTL clock input, a TTL clock output or an ECL trigger output. When configured as a clock input, J3 has 50 ohm input impedance and an input threshold of approximately 100mV. Alternatively, J3 can also be configured as a TTL Clock out or ECL Trigger Out. When configured as a TTL Clock Out, J3 will output the digitizer clock (500 or 200 MHz) divided by 8. The jumper setting to select the various options is shown below.

Configuration	Jumper Connections
Clock Out for PDA500 RevA (*Digitizer Clock Divided by 2*)	JP4: 2 & 3 50-ohms JP1: 1 & 2 (requires hardware modification)
Clock Out for PDA500 RevB	SW5: (DIP Switch) Divide by 8: Position 1 = On, 2 = Off, 3 = Off, 4 = Off Divide by 16: Position 1 = Off, 2 = On, 3 = Off, 4 = Off Divide by 32: Position 1 = Off, 2 = Off, 3 = On, 4 = Off Divide by 64: Position 1 = Off, 2 = Off, 3 = Off, 4 = On SW6: Position 1 = Off, 2 = Off, 3 = Off, 4 = On
External Input Clock	JP4: 1 & 2 (default) JP1: Don't Care

1.10 Board Addresses

The PDA500 board requires two blocks of 64 sequential bytes of I/O address space, one block is used for communication to the PDA500 PCI controller, the other block is for the PDA500 control/status registers. The board requires 8k bytes of sequential memory address for data transfer to and from the on board signal memory. The PDA500 is fully PCI Plug and Play compatible. Therefore, the systems Plug and Play BIOS will automatically configure the PDA500 into available address space.

2 Functional Description

2.1 Overview

The PDA500 is a circuit board assembly designed and tested per the Peripheral Component Interconnect (PCI) Local Bus Specification Revision 2.1. This powerful waveform capture board is designed with the capability of capturing very large amounts of signal information at high digitization rates. The maximum digitization rate is 500 Megasamples per second with 8 bits of resolution. The PDA500 has an analog bandwidth of 500 Megahertz. The PDA500 has a minimum of 1 megasamples of on-board signal memory and may be ordered with a signal memory size of 4 megasamples.

The PDA500 implements a Signatec Auxiliary Bus (SAB) Version 3 that has a data width of 64 bits and is capable of sustained data transfer rates up to 500 Megabytes per second. This interface is used for transferring data acquired by the PDA500 to other boards such as DSP boards, high-speed disk systems, or large memory boards. Data can also be acquired directly to the SAB completely bypassing the on board RAM to allow acquisition lengths limited only by the storage limits of other devices connected to the SAB. In addition, the PDA500 can be completely controlled by a "Bus Controller" device connected to the SAB. See section 2.11 for more information regarding SAB version 3 data transfer and control.

Acquiring data directly to the SAB with a PMEM500 memory board connected allows the PDA500 to capture up to 512 megasamples of signal data at a rate of 500 MHz. If multiple PMEM500 boards are used, even longer data records can be acquired. The SAB supports up to 4 PMEM500 boards, which allows for capturing up to 2 gigasamples of data. Data acquired directly to the SAB can also be stored and processed on any one of the Signatec lines of DSP boards such as the PMP8. The PMP8 is capable of controlling the PDA500 over the SAB interface without the need for host computer intervention. This allows for maximum data acquisition/processing rates in signal averaging and time critical applications. For more information on SAB Control see Section 2.11.2.

The PDA500 has two input signal channels, an external trigger connector, and an external clock connector. Figure 2 is a simplified block diagram for the board and will be referenced many times in the following descriptions of various board functions.

"**APPENDIX B - Hardware Registers**" at the back of this manual is a listing of the eight 32 bit registers that control and monitor the operation of the PDA500. The write registers are used to control the operation while the read registers perform the monitoring function.

All write registers are set to a logical '0' on power up.

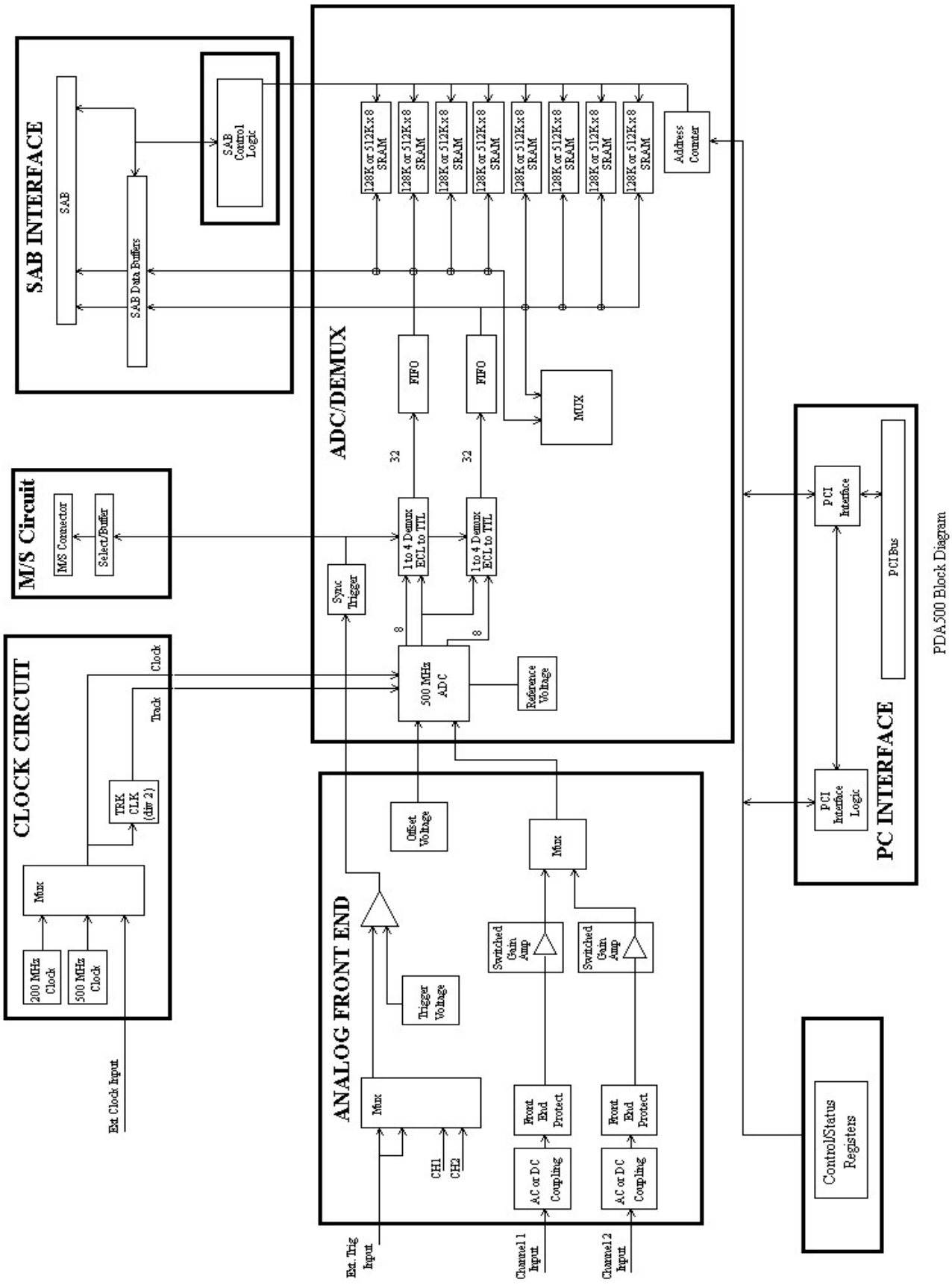

SAB INTERFACE
SAB
SAB Data Buffers
SAB Control Logic
ADC/DEMUX
128K or 512Kx8 SRAM
Address Counter
FIFO
MUX
32
1 to 4 Demux ECL to TTL
Sync Trigger
8
300 MHz ADC
Reference Voltage
M/S Circuit
M/S Connector
Select/Buffer
CLOCK CIRCUIT
Mux
200 MHz Clock
500 MHz Clock
TRK CLK (div 2)
Clock
Trak
Ext Clock Input
ANALOG FRONT END
Offset Voltage
Trigger Voltage
Switched Gain Amp
Front End Protect
A.C or D.C Coupling
CH1
CH2
Ext. Trig Input
Channel 1 Input
Channel 2 Input
PC INTERFACE
PCI Interface
PCI Interface Logic
PCI Bus
Control/Status Registers
PDA500 Block Diagram

Figure 2

2.2 Accessing the Data Registers

The PDA500 supports 32 bit accesses for both memory accesses and I/O accesses. Each register is 32 bits (4 bytes) therefore each register requires 4 bytes of address space (see Appendix B). Register 0 is located at the Base I/O address as set in the PCI configuration register (See section on *Iobase500* function). The other registers are offset from the Base address by the register number times 4. For example register 1 is at address Base+4, register 2 is at address Base+8, etc.

The PDA500 software library contains functions for accessing the registers. These functions handle the chore of manipulating the register bit patterns. The programmer may also choose to control the PDA500 operation by direct access to the registers.

2.3 Analog Input Circuitry

The PDA500 has two identical analog input channels referred to as channel 1 and channel 2. Both channels are equipped with a transient voltage protection circuit and independent signal selections. An on-board software controlled analog switch is provided to select which one of the two input channels is digitized.

2.3.1 Input Coupling

The input signal into the PDA500 can be AC or DC coupled into the analog front end. The signal coupling of each channel in independently selected with a DIP switch. See section 1.8.1 for more details.

2.3.2 Analog Signal Conditioning

After input protection and coupling, the PDA500 analog circuit has gain, attenuation, and multiplexer features to allow for a wide range of signal conditions. The full-scale voltage range for the PDA500 is set by a combination of the "Switched Gain Amplifier" and "Programmable Attenuator" stages. The voltage range setting is discussed in terms of attenuation between the input connector and the ADC. This means that the minimum attenuator setting is used for the smallest full-scale voltage range.

The amplifiers have a gain of 12 dB or 0 dB. From an attenuation point of view, these represent an attenuation of 0 dB or 12 dB respectively. The gain setting is controlled by the attenuator bit ATT3. Both input channels must have the same gain setting since there is one attenuator whose input is the multiplexed signals of either channel 1 or channel 2. The default (power up) value of this bit is 0, which sets an attenuation of 0 dB. Setting ATT3 to 1 switches the amplifier to an attenuation of 12 dB.

The output of the channel 1 or channel 2 amplifier is selected by the input signal multiplexer on the PDA500. The register bit CHSEL controls the selected channel. A logical '0' selects channel 1. Function *chanset500* may be used to select the input signal channel.

The output of the multiplexer is passed through the Programmable Attenuator which can be programmed for 0 to 14 dB of attenuation in 2 dB steps. Register bits ATT0 through ATT2 of register 0 set the attenuation. Setting ATT0-ATT2 to all '1's sets the attenuator for 14 dB. The attenuator is used in conjunction with the amplifier gain setting to properly adjust the system attenuation.

To work with the smallest signal levels, or 0 dB of attenuation, the amplifier is set to 0dB attenuation (ATT3=0) and the attenuator will be set for 0 dB (ATT0-ATT2=0). For increasingly larger signals the attenuator setting is increased in 2 dB steps up to 10 dB. This is the cross over point after which the selected input amplifier is switched from the 0 dB to 12 dB of attenuation and the Programmable Attenuator is returned to 0 dB. Thus, to accomplish a 12 dB attenuation, ATT3 will be set to 1 and all the attenuator bits (ATT0-ATT2) will be set to 0. The attenuation may be set up to 26 dB in this manner although the maximum practical attenuation is about 22 dB

in order to attain full scale ADC signal levels with no signal compression. The function *attnset500* may be used to set the attenuation and will handle both the amplifier gain switching and the attenuator setting.

2.3.3 Analog Bandwidth

The output of the attenuator is routed through DIP switch SW3 for selection of the analog signal bandwidth. This allows a 100 MHz, 3 pole low pass filter to be inserted before the ADC for applications where this is desirable. When the filter is not used there is no bandwidth reduction and the analog bandwidth will typically be 500 MHz. See Section 1.8.2 for SW3 DIP switch settings.

2.3.4 Calibration

The PDA500 is shipped with a calibration table which lists the full scale (peak to peak) input voltage for each attenuator setting for both channel 1 and channel 2. The table is supplied as a disk file on a disk entitled "PDA500 Calibration Table". The file name is the serial number of the board with a file extension of .ptb. This file consists of 28 floats, a total of 1Kbytes. Each float represents the full scale input voltage (in millivolts peak-to-peak) for one of the 14 attenuator settings for each channel, with the first float representing 0 dB attenuation for channel 1 and the last float value representing 26 dB attenuation on channel 2.

2.3.5 DC Offset Voltage Adjustment

The dc level of the signal into the ADC can be shifted using the offset voltage DAC. The offset voltage range is approximately equal to the ADC full scale voltage range. The offset shift selected applies to both channel 1 and channel 2 and is selected relative to the full scale voltage making the offset independent of the attenuator setting.

The offset voltage range is inverted relative to the ADC signal input. A mid range offset of 128 produces a signal, which is approximately centered within the ADC signal input range. A maximum offset value of 255 will produce an offset at the minimum ADC input level and an offset value of 0 produces an offset as the maximum ADC value. The 8-bit value for the dc offset DAC is represented by bits OF0-OF7 of register 1, with OF0 being the lsb. Function *offset500* may be used to set the dc offset.

2.4 ADC and Demultiplexer FIFOs

The heart of the PDA500 is the 500 MHz analog to digital converter. This 8-bit converter has superior high frequency performance as indicated by its dynamic accuracy of 7.0 effective bits at a signal frequency of 250 MHz. The full scale input voltage for the ADC is ±250 millivolts.

The digitizer clock input to the ADC has a limited frequency range of 50 to 500 MHz. For this reason lower effective digitization rates are achieved by dividing down at the output of the ADC by skipping data. This is accomplished in the clock divider to the demultiplexer circuit. The ADC also has a divide by 5 input that internally divides down the output clock used to clock the demultiplexer by a factor of 5. The divide by 5 control, DIV5 bit of register 0 set to a logical '1' will activate the divide by 5 function.

The demultiplexer is a data steering and buffering device which takes in 8 data samples of 8 bits each and outputs 64 bits of parallel data. This 8 to 1 expansion of the data width reduces the maximum write rate to signal RAM to 62.5 MHz when the input rate is 500 MHz. The demultiplexer is the key to implementing large memory systems for high-speed digitizers.

The demultiplexer contains First In First Out (FIFO, 8K X 8 => 64K) buffers that add functionality to the PDA500 board in many different ways. The FIFOs not only allow pre-trigger data to be captured, they also allow pre-trigger data to be captured when the PDA500 is used in the segmented acquisition mode. The FIFOs also make it possible for the PDA500 to acquire data directly to the host computer memory via the PCI bus and make possible the advanced packet transfer features of the SAB interface.

Figure 2 shows that the clock division is accomplished with an 8 to 1 multiplexer which selects one of 8 possible frequencies, each of which is one-half of the previous one (see the CLOCK CIRCUIT block of figure 2). For the purposes of this discussion the ADC output clock should be considered to be the same frequency as the ADC input clock and the demultiplexer clock frequency is the ADC clock frequency divided by 1, 2, 4, 8, 16, 32, 64, or 128. The clock divider bits are labeled CK0-CK2 and are bits 13 through 15 of register 0, with CK0 being the lsb. All 0s selects a divider of 1. The effective digitization rate is equal to the ADC clock divided by the clock divider value and also divided by 5 if DIV5 is set.

2.5 Memory Addressing

The PDA500 contains a counter that generates the signal RAM address when data is being transferred to or from the RAM. Eight 8-bit values from the ADC are stored at each RAM write. Therefore each address counter value addresses 8 data samples. On transfers to or from the PDA500 board, further address decodes are done from the PC address to enable the proper 32 bit value to be transferred to the PC.

Prior to a data acquisition the address counter is set to the address at which the first data sample is to be stored. The address counter is set by writing the Start Address bits of register 2. The ability to set the address counter to any value allows acquisition to start at any 8 byte boundary in RAM. Setting the address counter value also determines the location of the beginning of the start of a SAB data transfer. See the *setstartadd500* function regarding the setting of the starting address. The end of an acquisition or an SAB transfer is dictated by the value of the ending address. See the *setendadd500* function for more information.

The PDA500 board is capable of transferring data to and from the on board signal RAM to the PCI bus as both memory mapped data or as Direct Memory Access (DMA) transfers.

2.5.1 Memory Mapped Data Transfers

Memory mapped transfers have the advantage of a simple software implementation. The data can be moved to and from the PDA500 data RAM just like any other memory location in the system. The disadvantage of memory mapped transfers is that the PC will not move the data as burst transfers. Burst transfers maximize the data transfer rates across the PCI bus.

The PDA500 data RAM is mapped into a host system memory space beginning at the address specified by the system 'Plug and Play' BIOS. In DOS this address can be read using the function *membase500*. In Windows this address can be found in the Membase element of the software register structure (see section 3.3.2.2). There are 8 kilobytes of memory space assigned to the PDA500. Since each ADC sample is one byte, the 8-kilobyte space corresponds to 8 kilosamples.

The number of samples mapped into the system memory space is less than the total number of samples on the board. Access to various blocks of 8 kilosamples of signal data is accomplished by using the *setstartadd500* function. This function will set the address value of the on board RAM to the desired 8 kilosample (8 kilobyte) block. For example setting the start address to zero will allow access to the first 8 kilosamples of data from address 0 to address 8191. The start address must be set to 8192 to read the next 8 kilosamples of data, reading system memory addresses of 0 to 8191. Note: The setting of the 'block' is necessary only when transferring memory mapped data. DMA transfers do not require setting of the 'block'.

2.5.2 DMA Data Transfers

DMA transfers can be used to achieve high data transfer rates between the PDA500 board and the PC system memory. DMA transfers are accomplished by using the PDA500 as a PCI Bus Master (see section 2.10.2). The PDA500 is capable of transferring data at the full PCI theoretical bandwidth of 132 Megabytes per second. The actual, sustained data transfer rate in a system will be lower and is dependent primarily on the hardware of the host system.

2.6 Clock Source

The ADC clock is selected from one of 3 sources. There are two internal clocks with frequencies of 500 MHz and 200 MHz, and an external clock input that can be supplied with any frequency from 50 to 500 MHz.

The clock source is selected with the register bits CS0 and CS1 according to the following table:

CS1	CS0	Clock Source
0	0	500 MHz
0	1	500 MHz
1	0	200 MHz
1	1	External

Function *clock500* or *clkfreq500* may be used to set the clock source.

The effective clock rate is also controlled by means of a divide-by-5 ADC control and an ADC output clock divider which divides the digitization rate in factors of 2 in 7 steps from 2 to 128. The divide-by-5 (DIV5) is bit 5 of register 0. Set to a logical 1, it will activate the divide by 5 function. The functions *clock500* or *clkdivbyfive500* may be used to set the divide by 5 operation.

The ADC output clock divider is set by register 0 bits CK0-CK2. When bits CK0-CK2 are set to all 0s the clock divider value is 1. When bits CK0-CK2 are set to all 1s the clock divider value is 128. The functions *clock500* or *clkdivide500* may be used to set the clock divider.

2.6.1 Clock Divider

The clock input to the ADC has a limited frequency range of 50 to 500 MHz. Therefore, in order to obtain lower effective digitizing rates it is necessary to skip ADC data samples. For example, if only every other data sample is used the effect on the digitizing rate is a division by two. The data from the ADC is written directly to a FIFO before it is written to signal RAM. For clock division by two, the FIFO write clock is divided. For clock division greater than 2 (4 - 256) bits 13-15 of register 0 (CK0-CK2) set the clock divider. When bits CK0-CK2 are set to all zeros the clock divider value is 1. When bits CK0-CK2 are set to all 1s the clock divider value is 256. The functions *clock500* or *clkdivide500* in the PDA500 library may be used to set the clock divider.

2.7 Operating Modes

The PDA500 has eight operating modes. The operating mode is selected via the mode bits OM0-OM2 of register 0. The function *mode500* may be used to set the operating mode. A device connected to the SAB such as Signatec's PMP8 Digital Signal Processor board can also control the operating mode of the PDA500. See Section 2.11.2 for more details regarding control of the PDA500 from the SAB.

The PDA500 operating modes are as follows:

Mode No.	Operating Mode	OM2	OM1	OM0	HEX
0	Off	0	0	0	0x0
1	PCI Acquisition	0	0	1	0x1
2	RAM Acquisition	0	1	0	0x2
3	SAB Acquisition	0	1	1	0x3
4	PCI Transfer	1	0	0	0x4
5	SAB Transfer	1	0	1	0x5
6	Write RAM	1	1	0	0x6
7	Standby	1	1	1	0x7

The minimum amount of circuitry necessary to communicate with the PDA500 via the PCI bus is in the Standby Mode to reduce power consumption. In the Standby Mode, all circuitry is powered with the exception of the Acquisition circuits. Power is applied to the acquisition circuits when the board is set to any of the three acquisition modes. Approximately 1.5ms is required after the board is set to an acquisition mode before a trigger event will be accepted. This delay is required to allow the acquisition circuits' power time to stabilize before an acquisition begins. If this delay is a problem for a particular application, the Power-Down Override feature can be used to remove the delay requirement.

2.7.1 Off Mode

In this mode most of the circuitry on the PDA500 is disconnected from its supply voltages so that the board dissipates the lowest possible power. This is the mode the board is in when the computer is first powered up. The programmer should place the board back into the Off-Mode when exiting a software program that operates the PDA500 in order to reduce power consumption. **NOTE: When the PDA500 is placed into Off Mode, all register information on the board will be lost.**

2.7.2 PCI Acquisition

This mode is used to acquire data directly to the PCI bus. The data must be transferred to the system using Direct Memory Access (DMA) transfer. Before the PDA500 can be put into the PCI Acquisition mode, the board must be set-up to perform a DMA transfer. See section 2.10.2 for more details on DMA transfers.

Once the PDA500 is put into the PCI Acquisition mode and a trigger has been received, data from the ADC is written to the on board FIFO. The PDA500 will then initiate a PCI Bus-Master, Burst-transfer until the desired number of samples has been acquired. The maximum acquisition rate for a PCI acquisition is dependent upon the hardware of the host system and other PCI bus traffic. If the acquisition rate is higher than the system can support, the FIFO full flag will be set, indicating some error in the transfer. The FIFO full indicates that the rate of acquisition was greater than the rate of data transfer and, at some point during the transfer, data was lost. There is no way of determining when data was lost during the acquisition, nor is it possible to determine how much data was lost. It is important to note that the data transfer rate to the PCI bus may not be constant due to other PCI bus traffic. Therefore, it is necessary to check the FIFO full flag after each acquisition to determine if an error occurred.

No attempt should be made to read or write to the PDA500 board until the PCI acquisition is complete. The Bus Master disconnect or system interrupt (section 2.10.2) should be used to determine the end of the acquisition, not the Acquisition Complete Flag (ACF).

2.7.3 RAM Acquisition Mode

This mode is the principal data acquisition mode. When the PDA500 is placed in this mode, waveform sampling will commence with the next trigger detected (Triggering is discussed in section 2.8). The first data sample is stored into signal RAM at the address set by the *setstartadd500* function. The PDA500 continues to acquire data

and store it to RAM until the on-board address counter reaches the ending address as set by the *setendadd500* function, at which time the Acquisition Complete Flag (ACF) is set. The ACF may be read by the PC to determine if data acquisition has been completed. The PDA500 can also be set-up to generate a PCI interrupt when data acquisition is completed.

2.7.4 SAB Acquisition Mode

This mode places the digitized data directly onto the 64 bit Signatec Auxiliary Bus (SAB) during data acquisition. To use this mode there must be some device that can capture the data transmitted over the bus, such as a PMEM500 or PMP8 board. The full digitizing rate of 500 MHz can be used in this mode, however, the device connected to the PDA500, which will receive the data, must be SAB Version 3 compatible. If the external device is compatible with an earlier version of the SAB, then the maximum digitizing frequency will be 200MHz. It is the job of the programmer to make sure that the destination device is prepared to accept the data. NOTE: If any clock divider is used, the data will only be sent-out across the lower 32 bits of the SAB.

2.7.5 PCI Transfer Mode

This is the mode used for PC access to the waveform data in the PDA500 RAM. Data can be transferred from the PDA500 to the PC using either conventional memory transfer functions (such as *'memcopy'*) or with Direct Memory Access (DMA) transfers (see section 2.10.2). If the PDA500 is set-up for DMA transfers, the transfer will begin as soon as the board is put into the PCI transfer mode. If the PDA500 is not set-up for DMA transfers, the data can be read from the PDA500 using the memory-mapped RAM technique outlined in section 2.5.1.

2.7.6 SAB Transfer Mode

This mode is used to send acquired data from the RAM on the PDA500 board to the Signatec Auxiliary Bus (SAB). Before entering this mode the address counter on the PDA500 must be set to the desired starting address. Data transfer will start immediately after the board is set to the SAB Transfer Mode. This mode is used to send data to a DSP or memory board, such as Signatec's PMP8 or PMEM500, but other target boards are also possible. When the address counter reaches the end address, the acquisition complete flag (ACF) will be set. See Section 2.11.2 for control of the PDA500 for a SAB Bus Controller connected to the SAB.

The PDA500A supports data transfer widths of 64 and 32 bits. Bus width configuration is set by the WID bit of register 6 as follows:

WID	Data Width
0	64 bits
1	32 bits

The PDA500 is a SAB version 3 compatible device (see section 2.11) capable of data strobe frequencies up to 66.6MHz. To insure backward compatibility with earlier versions of the SAB that have a maximum strobe frequency of either 62.5MHz or 25MHz, the strobe frequency is selectable on the PDA500. Speed Select bits SP0 and SP1 of register 6 select the source of the strobe clock. The table below shows the possible comminations.

SP1	SP0	Source Clock
0	0	25MHz
0	1	62.5MHz
1	0	66MHz
1	1	External Clock

2.7.7 Write RAM Mode

This mode is used to write data to the on-board RAM of the PDA500.

2.7.8 Standby Mode

This mode should be considered as the base mode for the PDA500. In this mode all circuitry, with the exception of the acquisition circuitry, is powered and the board is available for any command.

2.8 Trigger Modes and Methods

2.8.1 Trigger Modes

Two triggering modes and two triggering methods are implemented on the PDA500. Trigger modes determine the manner in which waveform data is stored in memory. Trigger modes should not be confused with operating modes. The most widely used is the single shot mode, however, the segmented mode has features that are useful in many applications. Functions *trig500* or *trigmode500* in the function library may be used to select the trigger mode. Operation in the various modes is described in the following sections.

2.8.1.1 Single Shot Mode

When Single Shot acquisition is selected, a single trigger event will cause the data acquisition to start and waveform information will be stored when the PDA500 board is placed into data acquisition mode. Acquisition to PDA500 RAM puts the first data sample in the address set by the *setstartadd500* function and the RAM is filled sequentially until the end address set by *setendadd500* has been reached. Data acquisition will then terminate as indicated by the Acquisition Complete Flag (ACF) going high. If the data is acquired to the SAB the ACF flag is set when the ending address is reached.

2.8.1.2 Segmented Mode

When Segmented acquisition is selected, each trigger event causes a fixed portion of the active PDA500 memory to be filled until the entire active memory is filled. The amount of memory that is filled for each trigger is called the segment size. To be meaningful, the segment size must always be smaller than the active memory size. Refer to the function *segmentsize500* for details concerning the segment sizes available and how they are selected.

The PDA500 is equipped with a Segment Detect bit in Resister 2. This bit can be monitored by the application to indicate that a segment has been acquired. This is especially useful in conditions where long time periods may exist between segment triggers by providing a means to indicate that at least one segment of data is available. The flag is cleared on read.

2.8.2 Trigger Methods

In addition to the normal method of data acquisition, two other trigger methods are provided on the PDA500. The trigger method determines the relationship of the stored data samples to the trigger event. Using the normal method as described above, data acquisition begins on detection of a trigger event. If one of the other two trigger methods is not enabled then the PDA500 functions in this manner. The other trigger methods are pre-trigger and delayed-trigger.

2.8.2.1 Pre-trigger

When acquiring pre-trigger data, the trigger signal actually performs as a stop trigger rather than as a start trigger. After the PDA500 board is placed into Acquisition mode, the board begins writing ADC samples to the on-board FIFO. A programmed number of pre-trigger samples later set using the function *presamples500*, data begins shifting out of the FIFO but is not written to RAM. This method insures that the FIFO always contains the

specified number of pre-trigger samples. When a trigger event occurs, data storage to the RAM begins with the samples shifting out of the FIFO. These values were shifted into the FIFO before the trigger event occurred, resulting in storage of pre-trigger data. This continues until the end address or end of a segment is reached. The data written to the RAM consists of the defined number of pre-trigger samples immediately preceding the trigger, with the remaining number of samples occurring immediately after the trigger. This mode is useful for seeing "backward in time", i.e. seeing the waveform as it existed before the trigger signal occurred.

The pre-trigger method of acquisition can be used with either Single Shot or Segmented acquisitions.

2.8.2.2 Delayed Trigger

When delayed trigger is enabled with the function *trigdelay500*, data acquisition will begin a selectable number of digitizer clock cycles after a trigger event occurs. This feature allows acquisition of a signal to begin at some predetermined time after a trigger event occurs.

The delayed trigger method of acquisition can be used with either Single Shot or Segmented acquisitions.

2.8.3 Trigger Source

The TRS bit of register 0 selects whether the source of the trigger signal is internal or external. Internal means that the ADC input signal is used as the trigger signal. External selects the signal input from the bottom SMA connector on the PDA500 bracket. Functions *trig500* or *trigsource500* in the library may be used to set the trigger source. The trigger source settings are as follows:

TRS	Trigger Source
0	Internal
1	External

2.8.4 Trigger Slope

The TSL bit of register 0 selects the edge on which to trigger as positive or negative. Setting TSL to a logical 0 selects POSITIVE slope. Functions *trig500* or *trigslope500* in the PDA500 library may be used to set the trigger slope.

2.8.5 Trigger Level

The trigger level defines the amplitude at which a trigger will occur. A valid trigger occurs when the selected trigger signal passes through the defined trigger level in the direction specified by the trigger slope setting. An 8 bit DAC (digital to analog converter) on the PDA500 board creates the analog trigger level. Bits TL0 through TL7 of register 5 define the 8-bit value, associated with the trigger level. For internal triggering the trigger level range of the DAC is approximately equal to the range of the ADC. If an external trigger is used the following table may be used to approximate the trigger level:

Trigger Value (TL0-TL7)	Low Level Trigger (50 ohm)	High Level Trigger (1K ohm)
0	-250mV	-2.5V
128	0	0
255	+250mV	+2.5V

The function *triglevel500* in the library may be used to set the trigger level.

2.8.6 Trigger Hysteresis

The trigger level information shown in the table in Section 2.8.5 has an amount of hysteresis applied to it to reject noise and minimize "jitter" on the detected trigger signal. The amount of hysteresis depends on the setting of the hysteresis control bit: HYS of register 0, which selects either high or low. Setting HYS to a logical 1 selects high, which corresponds to about 27 millivolts. Setting HYS to a logical 0 selects low hysteresis, which corresponds to about 7 millivolts. Using high hysteresis will help to reject noise and is the suggested setting unless low level triggering is required.

2.8.7 Software Trigger

The PDA500 requires a trigger signal to start data acquisition (or to stop data acquisition in pre-trigger mode). That trigger may be an externally generated trigger from the external trigger input, an internal trigger from the ADC input signal, or it may be a software generated trigger signal. The software trigger is functional whether internal or external triggering is selected.

To generate a software trigger a high level is written to the ST bit of register 0 followed by writing a low level to the same bit. Function *trigsoft500* in the PDA500 library may be used to generate the software trigger.

2.9 Memory Size Selections

There are four different settings that control the size of waveforms captured in the PDA500 RAM. These are the starting address, ending address, segment size, and pre-trigger samples.

2.9.1 Starting Address

This setting determines the address at which the first data sample will be stored when acquiring data or transferring data to the PDA500 from the PCI bus. The function *setstartadd500* can be used to set the starting address. The starting address must be set on 8-byte boundaries. The amount of data transferred is determined by the difference between the starting address and the ending address.

When transferring data from the PDA500 to either the PCI bus or the SAB, the starting address determines the RAM address from which the data will begin transferring.

The Address Counter Reset bit of Register 2 can be used to automatically reset the address counter when the PDA500 is set to Standby Mode. In many applications, data is acquired to the PDA500 RAM and then transferred to the PC. The reset feature removes the necessities of setting the address counter to zero at the end of an acquisition and before the data is transferred.

2.9.2 Ending Address

This setting determines the address at which data transfer to or from the PDA500 will stop. The number of samples written to or from the PDA500 is equal to the ending address minus the starting address. When the board reaches the ending address, the Acquisition Complete Flag (ACF) is set. This flag can be read using the software function *acqcomp500* or a PCI interrupt can be generated. The ending address must be set on boundaries of 64 bytes. The ending address can be set using the function *setendadd500*.

2.9.3 Segment Size

This setting is only functional in segmented trigger mode. Segmented mode allows for partitioning the active memory into many portions, each of which requires its own trigger signal to fill the memory segment. The segment size is programmed with the SS0 through SS3 bits of register 0. The value range set with these bits is **0** through **11** which selects a segment size from 0 samples to 64 kilosamples as shown below.

<val>	Segment Size
0	32
1	64
2	128
3	256
4	512
5	1024(1k)
6	2048(2k)
7	4096(4k)
8	8192(8k)
9	16384 (16k)
10	32768 (32k)
11	65536 (64k)

The segment size selected should always be less than the active memory. Data acquisition will stop when the address reaches the ending address value, even if the segment is not complete. The function *segmentsize500* in the library may be used to set the memory segment size. NOTE: 8 extra samples will be acquired in segment 0 if the clock is set to 500MHz.

2.9.4 Pre-trigger Samples

This setting is only functional if the pre-trigger method is used. The number of pre-trigger samples determines the number of data samples prior to the trigger that will be stored. By using the pre-trigger mode with an appropriate amount of pre-trigger samples, waveform information before and after a trigger event may be captured. The number of pre-trigger samples must be less than or equal to the active memory setting.

The TDP0 through TDP15 bits of register 7 set the amount of pre-trigger samples. The function *presamples500* in the PDA500 library may be used to set the number of pre-trigger samples. The values 0 to 4096 can be passed to the PT register. The number of pre-trigger samples is equal to the register value times 8 giving a possible range of pre-trigger samples between 0 and 32768.

2.10 PCI Interface

The PDA500 is a 32 Bit 5 volt PCI board. The PDA500 is compatible with revision 2.1 of the PCI local bus specification and is "Plug and Play" compatible. The control and monitoring registers are accessed from the PCI bus with 32 bit I/O read/writes. The signal data can be read from, or written to the PDA500 with 32 bit memory transfers, or with 32 bit Bus Master Direct Memory Access (DMA) transfers.

2.10.1 Plug and Play Features

The PDA500 contains a non-volatile EEPROM in which has been stored information required by the system to "Plug and Play" configure the PDA500 board for operation. Upon boot-up the system Plug and Play BIOS will poll each PCI slot in order to identify the unique "Vendor Identification" and "Device Identification" numbers of the boards installed in the system. The system then interrogates each board in order to determine what system resources are required. The system assigns the necessary I/O and Memory addresses to each board. The PDA500 requires two address spaces and one memory space.

The Serial Number of each PDA500 is stored in the on-board EEPROM. This allows user software to distinguish between multiple PDA500 boards installed in a system by reading the Serial Number value. All PDA500 library functions that identify board address parameters have the ability to receive the board Serial Number as a function parameter.

Each time the system is powered-up the "Plug and Play" configuration process is carried out. Therefore it is necessary to retrieve the address space used by the board each time a program is started. For more information on Plug and Play operations contact the Signatec web site at www.signatec.com for the application titled *Plug and Play Operation of the PCI Local Bus*.

2.10.1.1 I/O Addresses

The PCI bus controller on the PDA500 requires two blocks of 64 bytes of address space. This space is used to communicate to the controller important information regarding Bus Master Control/Status, DMA transfer address, and interrupt information. The PDA500 status and control registers described in "**APPENDIX B - Hardware Registers**" occupy another 32 bytes of address space.

2.10.1.2 Memory Space

The PDA500 has the data RAM mapped into the PC memory space. This space is used to read or write data RAM values using standard memory manipulation functions. See the section titled Memory Mapped Data Transfers (2.5.1) for more information. The memory space is not used to transfer data when using DMA transfers.

PCI boards used in systems based on Intel architecture processors operating in "Real Mode" (i.e. DOS) must have the memory space mapped within the lower 1 Megabyte of memory space. Systems operating in the "Protected" mode may have the PCI board memory mapped anywhere within 32 bit address space. The default size and area for the PDA500 is 8 kilobytes anywhere in 32-bit address space.

2.10.2 DMA Transfers

Data can be transferred between the PDA500 and the system memory using two methods. The first is common memory read/write functions. This type of transfer has a limited transfer rate and should only be used in applications where speed is not critical. See section 2.5.1 for more information on memory mapped transfers.

The other memory transfer method is accomplished using Direct Memory Access (DMA) transfers. To generate DMA transfers a PCI board must be a Bus Master. This means that the PCI board must take control of the PCI bus and command the DMA transfer to occur. A DMA transfer is then carried out using the system's DMA controller without intervention from the system processor. This is the only method available for 32 bit burst transfers on the PCI bus in a standard PC.

Data can be transferred from the PDA500 data RAM to the PC using the function *brd2pcdma500*. This function will execute a DMA transfer and enable the PDA500 to generate a system interrupt when the transfer is complete. Before this function is called, system memory must be allocated as a destination for the data and an interrupt handler must be created and installed. In the DOS operating system, a destination pointer can be generated by creating an array and using a pointer to the array or by using a system memory allocation function, such as *malloc()* to create a pointer. A simple interrupt handler is provided with the function *dmaint500* and can be installed with the function *setupciint500*.

Under Windows a similar method is used to achieve a DMA transfer. Memory for a DMA transfer must be allocated and an interrupt is generated when the transfer is complete. The function used to allocate a buffer for DMA transfers is *allocatedmabuffer500*. The function will return a virtual address which Win32 applications can use. The function also places the physical address (where the memory will be transferred on the PC) into one of the parameters passed to the function. The physical address must be stored for later use by the *brd2pcdma500* function. The device driver instead of the user-defined functions handles the interrupt generated at the end of the DMA transfer.

The PDA500 becomes the PCI Bus Master and data is transferred from the data RAM in 32 bit wide bursts. The PDA500 is capable of transferring one 32-bit word on each PCI bus clock cycle. This means the PDA500 is capable of data transfers at the theoretical transfer rate of 132 megabytes per second. The actual transfer rate realized in a system will be significantly less than this and is dependent upon the system hardware configuration and data traffic from other devices on the PCI bus.

2.11 Signatec Auxiliary Bus Interface

Many Signatec products implement a high-speed bus termed the Signatec Auxiliary Bus or SAB. The SAB was first introduced on Signatec products in 1993. It had a data transfer rate capability of 200 megabytes per second using 64 bit wide bus with a 25 MHz clock. In 1998, SAB version 3 was introduced. Version 3 of the SAB is capable of transfer rates of 500 megabytes per second using a 64 bit bus with a 62.5MHz clock. Version 3 also employs many important system features to maximize the utilization of the bus bandwidth, the most important of which is Packetized Data mode. In this mode, data packets of programmable length are transmitted at a fixed clock rate of 66.67 MHz resulting in a maximum data burst transfer rate of 533 megabytes per second for 64-bit transfers or 266 megabytes per second for 32-bit transfers. SAB version 3 adds the ability to transfer commands via the SAB data bus expanding the bus functionality. A differential strobe signal and support for eight boards are other important additions.

Early products like the DA500A, MEM500, SP20, DAC200 and PDA12 boards were designed with the SAB specification. The PDA500 and future Signatec products incorporate the SAB version 3 interface. SAB version 3 products like the PDA500 board can be connected to SAB products like the MEM500 successfully as long as the SAB maximum clock rate of 25MHz is not exceeded.

The purpose of the SAB is to allow the transfer of data between boards in a PC environment at a much higher data rate than can be achieved using the PCI bus. The PCI bus has a theoretical transfer rate limit of 132 megabytes per second with actual rates being slower. The PDA500 is capable of sustaining 500 megabytes per second transfer rate over the SAB regardless of system speed and is not susceptible to traffic on the PCI bus, which can adversely affect transfer rate. These issues make the SAB attractive for the most demanding applications.

The SAB is located at the top of the PDA500 board on the right side. There can be only one active "master" board on the SAB at a time. The PDA500 is only capable of transmitting data on the SAB and not of receiving data. That is, it can only be a "master" board and not a "slave" board. The terms "master" and "slave" used here should not be confused with the same terms used elsewhere in this manual to refer to the operation of multiple PDA500 boards. A maximum of five boards can be interconnected on the SAB.

A full specification detailing the SAB is available from Signatec upon request. "**Appendix E Signatec Auxiliary Bus Connections**" shows the pin connection for the SAB on the PDA500.

2.11.1 PDA500 SAB Data Transfer

The SAB version 3 supported by the PDA500 provides block transfers and packetized transfers for moving data from the PDA500 to another device on the SAB. Either method can be used to transfer data to the SAB: Data that has been acquired to PDA500 RAM first, or to acquire data directly to the SAB.

When the PDA500 is put into the AcqMode2 the SAB BUSY- line will be pulled low and the STRB32- line will begin to strobe out data immediately after a trigger event has occurred. The strobe frequency will be equal to one-eighth of the digitizer frequency. The data will strobe onto the SAB until the specified ending address is reached (see section 2.9.2). The device that is to be the destination for the acquired data, such as Signatec's PMEM500 or PMP8, must be set-up before the acquisition begins. In the SAB acquisition mode the PDA500 will **not** respond to the WAIT32- or SLOW32- line on the SAB.

2.11.1.1 Block Transfer

Block transfers can be used to transfer a large amount of data from a single SAB device over the SAB. The sending device, termed the SAB Master, will transfer the entire data block specified continuously until complete or the process is interrupted, either by the Bus Controller Owner or by a WAIT signal. When transferring data with this method the PDA500 drives the data lines and the data strobe signal with a well defined timing relationship relative to the data. At the end of each Block Transfer cycle, the PDA500 will automatically turn itself into an Inactive Slave, which releases the SAB for other traffic.

In Block Transfer mode the PDA500 can transfer data at any clock rate up to 62.5 MHz. Boards designed to earlier versions of the SAB have a clock rate limit of 25MHz. In systems using earlier SAB boards, it is the programmer's responsibility to insure that the proper clock rate selection is made before data transfer begins.

2.11.1.2 Packetized Transfer

Packetized Data mode was added starting with version 3 of the SAB specification. In this mode, the bus clock is always 66.66 MHz. This mode was specifically added to maximize the utilization of the SAB bandwidth in systems using multiple SAB devices. Packetized Data mode allows data transfers between multiple sources and destinations to be interleaved onto the bus at the maximum bus speed, independent of the actual data speeds internal to the individual boards. This is accomplished by the addition of FIFOs at the SAB interface of each board. Operation in Packetized Data mode requires a Bus Controller implementing SAB version 3 protocols. An example of such a device is Signatec's PMP8, see the SAB specification for more details.

2.11.2 PDA500 SAB Control

The PDA500 is capable of responding to commands issued over the Signatec Auxiliary Bus (SAB) by an external device, such as Signatec's PMP8 Digital Signal Processor board. The PDA500 must be initially turned on, or set to the Standby mode, from the PCI bus, after which all board operation can be controlled from the SAB. Controlling the operation and transferring data over the SAB maximizes the overall data throughput rate in applications requiring multiple acquisition cycles or in systems with a large amount of PCI traffic.

Commands are issued over the SAB with the use of the SAB interrupt lines (INTR1-INTR5), the SAB communication lines (COM1-COM4), and in some cases the lower 32 data lines. The PDA500 is assigned a specific INTR line to which it will respond. Selecting the interrupt is accomplished by setting the Interrupt Enable (IEN) register, referred to as SAB Board Number. This register enables control of the PDA500 from the SAB when the register is set to a non-zero value using *SABconf500*.

A combination of direct commands, which use only the INTR and COM lines, and indirect command, which also use the data lines, are used for SAB communication. The SAB protocol allows indirect commands to be used to write all PDA500 status and control registers accessed by the PCI bus for maximum flexibility. The programmer must be aware when using SAB commands that access from the PCI bus is still possible and care must be taken to ensure proper operation when using both interfaces to control the PDA500.

2.12 Thermal Management and Power Down Features

The high-speed performance of the PDA500 comes at the price of power dissipation. The analog to digital converter for example consumes 7.5 watts of power and there are a number of ECL devices used in the high-speed sections that also use substantial amounts of power. Power dissipation translates to heat, which is detrimental to long life and reliability as well as reduced battery life in portable systems.

Because of this, the PDA500 has implemented a fan/heat-sink combination for the analog to digital converter and power down features. Proper use of these features by the software operating the board will lead to "cool"

operation and increased board life. The three main thermal control features are Off-Mode, Automatic Power Control, and Power-Down override.

2.12.1 Off Mode

Off-mode is the mode that the PDA500 will assume when the host computer is powered up. In this mode most of the circuitry on the PDA500 is powered down including all information registers. If a user program uses the Off-mode and then returns to one of the "on" modes, all register information must be rewritten to the PDA500. When the PDA500 operating program is terminated, the PDA500A should always be returned to the Off-mode.

2.12.2 Automatic Power Control

All of the "hot" parts on the PDA500 are associated with the data acquisition circuitry. The default configuration of the board is to apply power to this circuitry only when necessary. In other words, when the board is in Standby mode, all circuitry is powered with the exception of the acquisition circuitry. The automatic power control feature causes the power to be applied to the acquisition circuitry when the board is set to one the acquisition modes. The trigger circuit is disabled until the acquisition circuit voltage level becomes stable, allowing for proper acquisition. The stable power condition is indicated by the Power Ready Flag of register 2. After a trigger is received, data is acquired and an acquisition complete is detected. The automatic power control feature then turns the acquisition circuit power off.

2.12.3 Power Down Override

In some cases the short time required for the acquisition power to stabilize before the trigger circuit is enabled can be a limit to the application. In these cases it may be desirable to leave the acquisition circuitry powered on even after an acquisition complete is detected. The Power-Down Override feature is included for such applications. When Power-Down Override is used, the acquisition circuitry is powered down only when the PDA500 is set to Off-mode. The Power-Down Override is enabled using the PDO bit of Register 0. It is suggested that the PDO bit only be set to a 1 when an application cannot tolerate the power up time of about 1.5 milliseconds.

2.12.4 Thermal Shutdown

The ADC on the PDA500 is an extremely expensive part. Because of its high power dissipation it has a heat sink and a fan mounted on it to eliminate the heat from the device. A board may be installed in a PC slot adjacent to the PDA500 component side. However, doing so will limit the efficiency of the ADC cooling fan. In applications where the PDA500 will be in the Acquisition mode for long periods of time, the slot adjacent to the PDA500 component side must be left empty to allow sufficient cooling to stop the ADC from overheating. In the event of failure of the ADC fan, the ADC is subject to overheating and damage. Therefore, a thermistor has been mounted on the heat sink of the ADC and is used to place the PDA500 into the power down condition if the heat sink temperature reaches 65°C. Once this occurs the PDA500 must be set to Off-mode and then to Standby mode to resume acquisition (assuming of course that the temperature is below 65°C).
A temperature bit may be read from Register 2 to indicate that the ADC heat sink temperature is greater than 65°C. A 1 on this bit indicates that the PDA500 is in the thermally activated power down condition and no data acquisition can occur. If this condition occurs repeatedly, the PDA500 must be relocated to another slot in the PC to maximize airflow over the ADC.

2.13 Master-Slave Operation

The PDA500 is designed to connect four PDA500 boards together for master-slave operation. In master-slave connection, the master board drives the clock and trigger signals for the slave boards so that data on the slave boards will align sample for sample with the data on the master board. Using one master and three slaves allows for simultaneous acquisition of four channels at 500MHz.

A 20-conductor ribbon cable is used to interconnect the boards. The master-slave connector is shown pictorially in Figure 1. The electrical connections are specified in "**APPENDIX D - Master-Slave Connections**". Configuring a board for master or slave operation requires setting the board jumpers as specified in section 1.9.1 **Solder Jumpers**.

2.14 PDA500 Verses DA500A Hardware

This section describes the differences between the PDA500 hardware and the previous ISA bus version, the DA500A. This section is written specifically for users familiar with the DA500A who are upgrading to the PDA500. Differences in features and mechanization that may not be apparent from the product data sheet are described here. Since this section is for users of the DA500A, hardware that no longer exists on the PDA500 will be listed first with new hardware listed last. For software specific information regarding the differences between the two products see section 3.4, Programming PDA500 Verses DA500A.

2.14.1 DA500A Hardware Removed

The following items describe the results of hardware implementations which render features or previously available selections on the DA500A as no longer available on the PDA500.

2.14.1.1 250MHz Clock Source

The DA500A had four clock source selections: 500MHz, 250Mhz, 200Mz and external clock. The PDA500 no long has a 250MHz clock source. The same effective clock rate can be achieved by using the 500MHz clock with a divider of 2. If application software selects the clock source number 1 previously used for 250MHz, the 500MHz clock source will be selected.

2.14.1.2 Interleaved Data Mode

The DA500A was equipped with an interleaved data mode that caused the Channel 1/Channel 2 multiplexer to switch between the two inputs to alternately sample two signals. Because of the limited switching speed of available analog switches suitable for this purpose, the interleaved data mode was limited to operating at a clock rate of 50Mhz, or 25MHz sampling per channel. This limitation made the interleaved data mode attractive only for those applications requiring two channels at 25MHz or less. Signatec is now able to offer these customers other less expensive products such as the PDA12A, which can acquire two channels as high as 62.5Mhz making the interleaved data mode obsolete for the PDA500.

2.14.2 DA500A Hardware Changed

The following items describe the results of hardware implementations that change features or selections as they existed on the DA500A as well as presenting the implementation on the PDA500.

2.14.2.1 Signal Coupling

The DA500A is equipped with software controlled switches for selection of AC or DC coupling of the signal and trigger inputs. The PDA500 provides DIP-Switches for coupling selection.

2.14.2.2 Pre-trigger Acquisition

The DA500A uses the on-board data RAM in a circular fashion for pre-trigger data acquisitions. That is, data samples are continuously stored in the data RAM until a trigger event occurs. A trigger causes the software selectable number of post-trigger samples to be stored at which point data acquisition stops. The address counter must be read to determine where data acquisition stopped. The trigger location will be this address minus the number of post-trigger samples. For example, if 4k post-trigger samples are selected, and the address after

acquisition is 72k, the trigger is known to be at address 72k-4k, or approximately 68k. However, since 8 data samples are stored at each RAM address, the location uncertainty of the trigger sample is 1 in 8.

The PDA500 simplifies the pre-trigger process for the user by using FIFOs to store pre-trigger data. With the PDA500 the number of pre-trigger samples is specified rather than the number of post-trigger samples. The on-board FIFO is used to accumulate the pre-trigger data by waiting until the desired number of pre-trigger samples are written to the FIFO before the FIFO is read. When a trigger event occurs, the samples being read from the FIFO begin to be stored to the RAM.

The pre-trigger method used on the PDA500 has the advantage of always storing the first pre-trigger sample at the first RAM location with subsequent pre-trigger samples following. The RAM location of the trigger point is always at the offset defined by the number of pre-trigger samples. That is, if 4k of pre-trigger samples are requested, the trigger location will be at the RAM location 4k+1.

2.14.2.3 Power Modes

The Ready mode used on the DA500 to power the acquisition circuits before acquisition mode no longer exists on the PDA500. The PDA500 uses a feature called automatic power control, which is also available on the DA500A, to accomplish acquisition circuit power-up. The automatic power control allows the board to be set to acquisition mode directly from Standby mode by monitoring the acquisition circuit power. The trigger circuit is locked out until the acquisition power becomes stable. The time required for the power to stabilize is approximately 1.5ms. Just like the DA500A, the PDA500 has a "Power-Down Override" feature that will allow the acquisition power to remain on even in Standby mode for those applications that can not accept the power-on delay. However, this feature should be used with caution, as sufficient air supply to the PDA500 cooling fan must exist to prevent the ADC from overheating.

2.14.2.4 Signatec Auxiliary Bus (SAB) Version 3

The implementation of the Signatec Auxiliary Bus on the PDA500 is far superior to that of the DA500A. The SAB Version 3 used on the PDA500 is capable of transfers at higher rates with more control features and "Packetized" transfer to increase efficiency in systems with multiple SAB devices.

The SAB Version 3 is backward compatible with the SAB implementation on the DA500A. The PDA500 is capable of the same transfer features available on the DA500A.

2.14.3 PDA500 New Features

The PDA500 is equipped with new features not available on the DA500A in addition to the features improvements listed above.

2.14.3.1 Analog Front End Protection

The PDA500 is equipped with a protection circuit at the analog input. The circuit protects against damage to the PDA500 analog circuitry from external sources. The protection circuit adds no measurable noise or distortion to the analog signal being acquired.

2.14.3.2 Delayed Trigger

The PDA500 is capable of delaying the beginning of the data acquisition from the trigger event by a software-selected number of clock cycles. This allows data to be collected from a significant time after the trigger.

2.14.3.3 Clock and Trigger Output

The PDA500 can be configured via DIP-switch settings to provide an output clock or trigger that is synchronous with the acquisition clock or trigger event. These signals can be used to synchronize or trigger a device external to the PDA500 in some system applications.

3 PDA500 Software

3.1 Introduction

The PDA500 is shipped with a disk titled PDA500 that contains 16-bit software modules that are compatible with the DOS operating systems. Another disk titled PDA500 Windows Software holds the 32-bit software compatible with the Windows(95/NT/2000) operating systems.

3.1.1 Software for DOS

The PDA500 disk contains the following directories:

EXAMPLES
INCLUDE
LIBRARY
SOURCE

The EXAMPLES directory contains example source code and executable files to aid the user in creating his/her own programs. They can also be used for initial checkout of the PDA500 board. The program PCIfind.exe is a program which will verify the existence of the 'Plug and Play' BIOS and report the configuration of devices on the PCI bus. If a PDA500 is installed, this program will verify its existence and report the I/O and Memory addresses assigned to the PDA500 by the 'Plug and Play' system. The EXAMPLES directory also contains an executable program called runp500.exe. This program provides a quick way for users to acquire and display data with the PDA500 board.

The INCLUDE directory contains the header file used to create the PDA500 library. The header file named PDA500.h contains all of the function prototypes and global variable definitions.

The LIBRARY directory contains the library file created with Borland C++ Version 5.02.

The SOURCE directory contains source code from which the function library can be created. The function library simplifies the task of creating custom application programs for the PDA500 board by removing from the programmer the necessity of understanding the intricacies of the hardware.

Observing the PDA500 global variables described in Section 3.3 allows the programmer to use both the PDA500 functions and direct register control. In order for software control registers to be set up correctly, the PDA500 function *setupbrd500* must be called before any register manipulation is done or any PDA500 function is called.

3.1.2 Software for Windows

The PDA500 Windows disk contains the following directories:

DeviceD
Example
Include
Library
Source
Win2000

The DeviceD directory contains the lower level device driver files. The files with the extension .vxd are virtual device drivers for use in Windows 95. The files with the extension .sys are kernal mode drivers for use in Windows NT. These low-level drivers are accessed using the functions provided in the PDA500win.DLL located in the Library directory.

The Example directory contains the executable file PdEx.exe. This example application allows the user to operate the PDA500 in the Windows environment. The Source sub-directory contains all of the source code for the example.

The Include directory contains the PDA500win.h header file. This file holds the function prototypes and global variables required by the Dynamic Link Library (DLL) located in the Library directory.

The Library directory contains the library files created with the Microsoft Visual C++ 6.0 compiler.

The Source directory contains the source code and make-files for the DLL.

The Win2000 directory has an .inf file used to install the PDA500 in Windows 2000. See the section on Windows 2000 below for more info.

Proper use of the PDA500 global variables described in Section 3.3 allows the programmer to use the PDA500 functions contained in the DLL. In order for all variables to be setup correctly the functions *setupregisters500* and *opendevice500* must be called before any register manipulation is done or any other function is called.

3.2 Installing Software

3.2.1 DOS Installation

The PDA500 software distribution disk contains a library of functions linkable to Borland's C++ 5.02. These functions operate Signatec's PDA500 board. The disk also contains the source code for the functions. To install the software type a:\setup followed by the name of the directory where you would like to install it. This assumes that the disk is already in the a: drive.

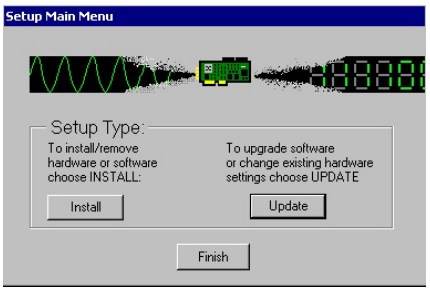
For example to install the software to the PDA500 directory on the C: drive type:

a:\setup c:\PDA500

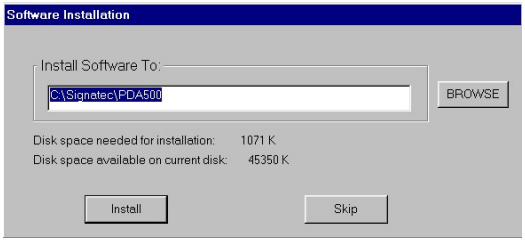
If you need technical assistance or have any questions Signatec can be reached by e-mail at techsupport@signatec.com or by calling Signatec at (909) 734-3001.

3.2.2 Windows Installation

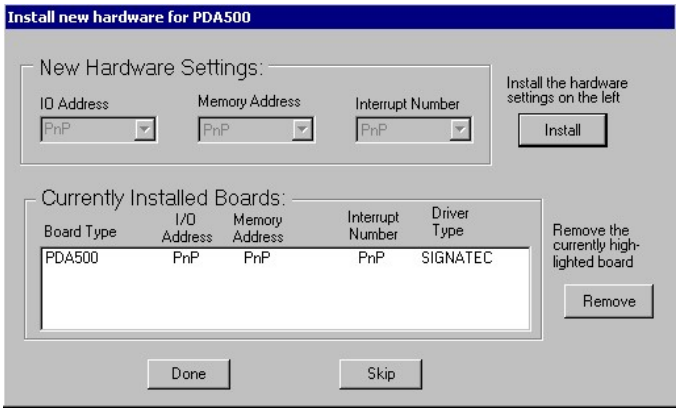
The PDA500 Windows software disk contains a DLL that can be linked to applications created using various tools. The functions contained in the DLL in conjunction with the lower level driver operate Signatec's PDA500 board. The source code for the DLL is included on the disk in the Source directory. To install the software, run the Setup.exe file on the PDA500 Windows software disk. In the initial menu window select the button marked Install.



The next window allows you to choose the directory where the files will be copied. Type in or browse for the directory and select the Install button.



The software is copied to the directory specified. The next window places driver information in the registry. Select the install button on the screen to add this information to the registry. This must be done once for each board.



The I/O Address, Memory Address, and Interrupt Number areas cannot be selected for a PDA500 board because it is Plug and Play configured. As such, the system configures the parameters at start-up, thus, they are not user specified. Select the Done button and the Finish button to exit the setup program. The setup.exe program is not needed to add information to the registry under Windows 95. Instead a .INF file is included so that the Add New Hardware Wizard or Update Device Driver Wizard can be used to install the hardware on the system (See section 3.2.3 for more details). An .INF file is also provided for Windows 2000. Unlike Windows 95, the board must be installed using the Setup program in addition to the Plug and Play installation procedure provided by Windows 2000 (See section 3.2.4). In any case, the computer must be restarted for the new settings to take effect.

If the initial settings ever need to be changed, the setup program can be run again to modify them.

If you need technical assistance or have any questions Signatec can be reached by e-mail at techsupport@signatec.com or by calling Signatec at (909) 734-3001.

3.2.3 Windows 95 Installation

Windows 95 offers Plug and Play support. Therefore the first time that Windows 95 is started with the PDA500 installed, the Update Device Driver Wizard application will prompt for a driver disk or will give a list of options which contains the selection "Driver Disk Provided by Manufacturer." Insert the PDA500 Windows disk and use the browse button to locate the path of the disk. Use the mouse to select the OK button and the Update Device Driver Wizard will show that a PDA500 has been found. Select Next with the mouse button. The Update Device Driver Wizard will then look for the pda500.vxd file. This file is located in the DeviceD directory of the PDA500 Windows disk and the Browse button can be used to find it. Select OK with the mouse. This will install the

information describing the device to the registry. The computer must be restarted for any of the changes to take effect.

If there is a previous version of the device driver installed, it must be replaced with the new one. This can be done by right clicking on the My Computer icon on the desktop and selecting the properties tab. This will open the system properties dialog. Select the Device Manager tab and a listing of all devices recognized by the system is displayed. Open the Signatec item and Double click on the PDA500 item inside. A dialog with information on the device is now displayed. Selecting the Driver tab brings up a dialog with a button labeled Update Driver. Place the floppy with the new version of the driver into the floppy drive and select this button. When asked if you want Windows to look for the device for you, select no and press the next button. Windows will then search the a: drive for an updated driver and when it finds one will go through the same process as the paragraph above.

3.2.4 Windows 2000 Installation

Windows 2000 offers Plug and Play support. Therefore the first time that Windows 2000 is started with the PDA500 inserted; the New Hardware Wizard will appear. Select Next to get past the first screen.



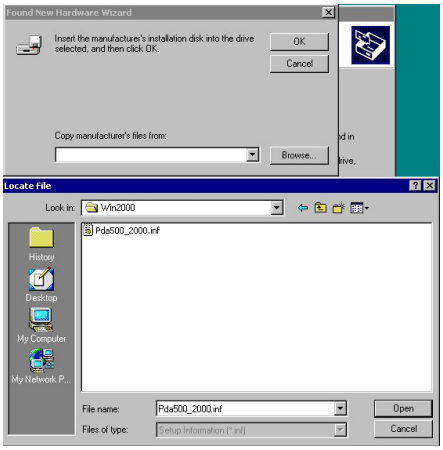
The screen below which then appears allows the user to select whether to search for the driver or display a list of known drivers. Select Search for a suitable driver for my device and select Next.



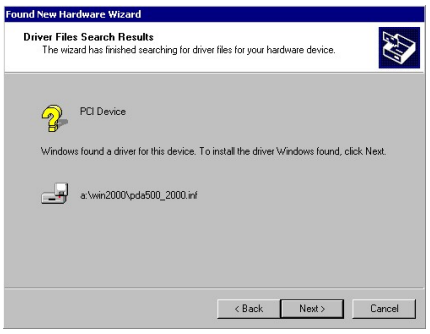
The next screen allows the user to specify a location for the driver. Because Windows 95 and Windows 2000 both use .INF files to find suitable drivers and because Windows 2000 cannot tell the difference between a Windows 95 and Windows 2000 .INF file, the Windows 2000 driver is placed in the Win2000 folder on the Windows software disk. Select the Specify a location item to specify this folder and select Next.



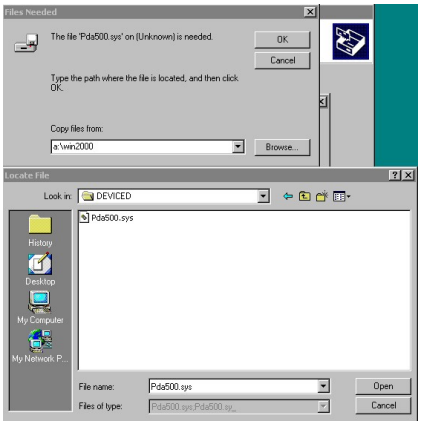
The dialogs below allow the user to browse for the location. Use the two dialogs to browse to the Win2000 folder on the Windows software disk and select Open.



After Windows 2000 searches for suitable drivers, it will find the one in the Win2000 directory. Select Next to install it.



The next screen tells us that a file is required which is not in the current directory. Rather it is in the DeviceD directory on the Windows disk so browse to that directory, select the Pda500.sys file and select Open.



Finally the last screen informs us that the driver has been successfully installed. If you haven't already run the setup program (Section 3.2.2) do so at this time.

3.3 Global Variables

3.3.1 Addressing Multiple PDA500 Boards

The PDA500 library is set up so that up to eight different PDA500 boards can be addressed in a system. Because of this, many of the PDA500 routines require the board to be accessed be identified.

3.3.1.1 Addressing in DOS

The functions that require a board identification parameter use the PDA500 global variables Brd1P500, Brd2P500,…., Brd8P500 to recognize the I/O base address of the board to be accessed. These variables have been defined with the following default values:

Brd1P500=6200h;	Brd5P500=6600h;
Brd2P500=6300h;	Brd6P500=6700h;
Brd3P500=6400h;	Brd7P500=6800h;
Brd4P500=6500h;	Brd8P500=6900h;

The PDA500 is a plug and play compatible board, therefore, the system BIOS will assign the I/O address of the board. Refer to section 2.10.1 for details on detecting the PDA500 I/O address assignment and changing the Brd variable to the necessary value.

The PDA500 routines, which require board number input, require that either the variable Brd1P500, Brd2P500, …, or Brd8P500 be passed in the parameter <brd> or the number 1 through 8. Passing the Brd variable will result in an operation being performed on the specified board. Passing the corresponding number value 1 through 8 will result in no operation being performed but a value being returned which identifies some present state of the board. See the individual routines for more details.

3.3.1.2 Addressing in Windows

Like DOS, the Windows operating system can address up to eight PDA500's. The functions that require a board identification parameter in Windows use a pointer to a structure. The structure holds a handle that is required by windows, to control the board and a pointer to the software register structure. The register structure contains information about the board. See the PDA500 Control Registers in Windows section 3.3.2.2 for more information on the software register structure.

```
typedef struct
{
  HANDLE Brdhandle;
  P500SWR * SWRpointer;
} P500ID, *PP500ID;
```

The information for the structure is filled using the function *opendevice500*. See the section on *opendevice500* for more information.

Many of the PDA500 Windows routines require that a second parameter in addition to the board identification parameter be specified. The parameter <rw>, that is at the end of a function parameter list, is used to indicate whether information should be read-from/written-to the PDA500 or the software registers in memory. If READ is passed, the information is read from the software registers. If WRITE is passed, then the information is written or read from the PDA500 depending upon the details of the particular function.

3.3.2 PDA500 Control Registers

Control and monitoring of the PDA500 is done through eight-32 bit hardware registers on the board. The bit patterns of the seven write only registers and the one read/write register used for control are stored in "software registers". When any of the board functions are changed, the bit pattern of the software register variable is modified and the entire 32-bit value is written to the board's hardware register. The PDA500 library functions free the programmer from the concerns of the control register bit patterns. The programmer can directly manipulate the contents of the control registers by using the global software variables defined in PDA500.h. Because these are the same register variables used by PDA500 functions, the PDA500 library functions can be used in conjunction with the programmer's manipulation. The actual structure of the hardware registers is defined in Appendix B.

3.3.2.1 PDA500 Control Registers in DOS

The PDA500 global register variables have the following form REGn[#].PARM. Where n=Register number, #=(board number-1), and PARM is the parameter identifier. For the software registers to be set-up correctly, the PDA500 function *setupbrd500* must be called before any register manipulation is done or any PDA500 function is called. Section 0 details the format of the register variables.

As long as the software register structure as defined in PDA500.h is used, a programmer may directly manipulate the software registers and use PDA500 functions for this purpose. A programmer may change the structure of the software registers or other global variables, however, if this is done, the PDA500 library functions will no longer operate correctly.

3.3.2.2 PDA500 Control Registers in Windows

In Windows, software register information together with I/O ports, memory base, memory base size, serial number, and IRQ number are combined in a software register structure named P500SWR.

```
typedef struct
{
  ULONG Port;
  ULONG Port2;
  ULONG Membase;
  ULONG Memsize;
  ULONG Serialnum;
  SHORT Brdnum;
  SHORT IRQnum;
  P500REGA Reg0;
  P500REGB Reg1;
  P500REGC Reg2;
  P500REGD Reg3;
  P500REGE Reg4;
  P500REGF Reg5;
  P500REGG Reg6;
  P500REGH Reg7;
  ULONG * Regptr[8];
} P500SWR, *PP500SWR;
```

Users must declare a variable of type P500WR and pass it to *setupregisters500* in order to initialize each field **before** calling any other function. This must be done for each board in the system. See the *setupregisters500* function for more details.

3.3.3 Miscellaneous Variables

3.3.3.1 Mode Variables

The following variables can be used in conjunction with either the PDA5500 library mode500 function or the global register variables to set the PDA500 board modes.

OffMode	PDA500 in Off Mode	(Section 2.7.1)
AcqPCI	PDA500 in Acquisition to PCI Bus	(Section 2.7.2)
Acq1Mode	PDA500 in Acquisition to Memory	(Section 2.7.3)
Acq2Mode	PDA500 in Acquisition to SAB	(Section 2.7.4)
PCIMode	PDA500 in Transfer to PCI	(Section 2.7.5)
AuxMode	PDA500 in Transfer to SAB	(Section 2.7.6)
WramMode	PDA500 in Write RAM Mode	(Section 2.7.7)
StdbyMode	PDA500 in Standby Mode	(Section 2.7.8)

3.3.3.2 Parameter Variables

The following can be used with PDA500 library functions or global register variables to set board parameters.

Pos	Sets Trigger Slope Positive	(Section 2.8.4)
Neg	Sets Trigger Slope Negative	(Section 2.8.4)
Int	Sets Trigger Source Internal	(Section 2.8.3)
Ext	Sets Trigger Source External	(Section 2.8.3)
Bits64	Sets Auxiliary Bus to 64 bit transfers	(Section 2.11)
Bits32	Sets Auxiliary Bus to 32 bit transfers	(Section 2.11)
Single	Single Shot Trigger Mode	(Section 2.8.1.1)
Segment	Segmented Trigger Mode	(Section 2.8.1.2)
READ	Read Value from Software Register (Windows only)	(Section 3.3.1.2)
WRITE	Write Value to Software Register (Windows only)	(Section 3.3.1.2)

3.3.3.3 Memory Control Variables

P500MEMBASE [Brd#-1]	Memory Mapped Address	*membase500*
P500IOBASE [Brd#-1]	Base I/O Address	*Iobase500*

Note: In Windows these variables are contained in the software register structure. See the section titled PDA500 Control Registers in Windows 3.3.2.2 for details.

The following software registers can be used to manipulate the contents of the PDA500 hardware registers.

REG0[Brd#-1].OM	Operating Mode (0-7)
REG0[Brd#-1].PDO	Power Down Override (0-1)
REG0[Brd#-1].CHSEL	Channel Select (0-1)
REG0[Brd#-1].DIV5	Clock Divide by Five (0-1)
REG0[Brd#-1].TMD	Trigger Mode (0-1)
REG0[Brd#-1].TRS	Trigger Source (0-1)
REG0[Brd#-1].HYS	Hysteresis (0-1)
REG0[Brd#-1].TSL	Trigger Slope (0 = ' + ';1 = ' - ')
REG0[Brd#-1].ST	Software Trigger (0=Off 1=Trigger)
REG0[Brd#-1].CS	Clock Source (0 or 1 = 500 MHz; 2 = 200 MHz; 3=Ext)
REG0[Brd#-1].CK	Clock Divider Bits (0-7)
REG0[Brd#-1].SS	Segment Size (0-15)
REG0[Brd#-1].ATT	Attenuation (0-15)
REG0[Brd#-1].DMA	DMA Transfer (0=disable, 1=enable)
REG0[Brd#-1].PIE	ACF Interrupt Enable (0-1)
REG0[Brd#-1].TM	Pre/Delay Trigger Select (0-2)
REG1[Brd#-1].HIZ	High Input Impedance, 1 MΩ (0-1)
REG1[Brd#-1].OF	Input Signal DC Offset (0-255)
REG2[Brd#-1].ACL	Address Counter, Lower 16 bits (0-65535)
REG2[Brd#-1].ACH	Address Counter, Upper 6 bits (0-63)
REG2[Brd#-1].ACRST	Auto Address Reset (0-1)
***Read Only Register Bits**	
REG2[Brd#-1].SDBIT	Segment Detect (0=Segment Not Detected, 1=Segment Detected)*
REG2[Brd#-1].T50	Temperature Flat (0-1)*
REG2[Brd#-1].PRF	Power Ready Flag (0-1)*
REG2[Brd#-1].AIP	Acquisition in Progress Flag (0-1)*
REG2[Brd#-1].RO	Roll Over Flag (0-1)*
REG2[Brd#-1].FF	FIFO Full Flag (0=Not Full, 1=Full)*
REG2[Brd#-1].ACF	Acquisition Complete Flag (0=Not Comp, 1=Acq Comp)*
REG3[Brd#-1].FI	FIFO Setting (0-255)
REG3[Brd#-1].PROG	Program FIFO (0-1)
REG4[Brd#-1].EAL	Ending Address Low (0-65535)
REG4[Brd#-1].EAH	Ending Address High (0-63)
REG4[Brd#-1].FR	Free Run Bit (0-1)
REG5[Brd#-1].TL	Trigger Level (0-255)
REG6[Brd#-1]	(See next section SAB Memory Control Register)
REG7[Brd#-1].TPD	Trigger Time Delay or Pre-Trigger Samples (0-65536)
REG7[Brd#-1].ISP	ISP Programming (0-63)

Note: In the Windows version the software registers are contained in the software register structure. The bit fields in the structure can be accessed just as above without the array index. For example to access the trigger level element in register 5 from a SWR structure named SoftRegs and place it in a variable called TrigLev do the following:

```
SWR SoftRegs;                    //declare the software register variable
SHORT TrigLev;
setupregisters500(&SoftRegs,1);  //setup and initialize the software register structure
TrigLev=SoftRegs.Reg5.TL;
```

See the section titled PDA500 Control Registers in Windows 3.3.2.2 for details.

3.3.3.3.1 SAB Memory Control Register

Register 6 is a special register which allows access to 4 different SAB registers . The first five bits of the register determine which SAB register will be written. The first register consists of most of the general SAB parameters.

REG6[Brd#-1].SABREG	SAB Register Select (0 selects this register)
REG6[Brd#-1].SABMODE	SAB Mode: Inactive slave/Active slave/Master (0-3)
REG6[Brd#-1].RDATAW	Data Width: 1 = 32; 0 = 64
REG6[Brd#-1].SABCLK	SAB Clock: 25/62.5/66/External(0-3)
REG6[Brd#-1].BPK	Block/Packet Mode Transfer (0-1)
REG6[Brd#-1].RAFS	Auto FIFO Status (0-1)
REG6[Brd#-1].ROWNER	Bus Owner/Non-Owner (0-1)
REG6[Brd#-1].RWAITINH	Wait Inhibit (0-1)
REG6[Brd#-1].RQMBS	Quick Memory Board Switch (0-1)
REG6[Brd#-1].INTDETEN	SAB Interrupt Detect Mask (0-1)

The second register contains packet length selection for packet mode use.

REG6[Brd#-1].SABREG	SAB Register Select (1 selects this register)
REG6[Brd#-1].PKTLENGTH	Packet Length Select

The third register contains fifo configuration parameters.

REG6[Brd#-1].SABREG	SAB Register Select (2 selects this register)
REG6[Brd#-1].RFFPROG	FIFO Programming Mode
REG6[Brd#-1].RFFSEL	I/O FIFO Select
REG6[Brd#-1].RFFSI	FIFO Serial Input
REG6[Brd#-1].RFFMRST	FIFO Master Reset
REG6[Brd#-1].RFFPRST	FIFO Partial Reset
REG6[Brd#-1].EOTIMASK	End of Transfer Interrupt Inhibit
REG6[Brd#-1].IDDTEN	Enable Interrupt Driver Transfer Communication

The last register allows for the setting of the board number on the bus and for enabling termination.

REG6[Brd#-1].SABREG	SAB Register Select (0xF selects this register)
REG6[Brd#-1].SABBRDNUM	SAB Board Select
REG6[Brd#-1].RTEMEN	Termination Enable

Each of the SAB software registers must be placed into the register 6 software register before it is written. For example to set the SABCLK in DOS:

```
//Get board number
i = brdnum500(brd);
// Set board to use SAB register 0
P500REG6SAB0[i-1].SABREG = 0x0;
//Set the actual value
P500REG6SAB0[i-1].SABCLK = val;
//Copy the address of the SAB register into the P500REGPTR array
P500REGPTR[i-1][6]=MK_FP(FP_SEG(&P500REG6SAB0[i-1]),FP_OFF(&P500REG6SAB0[i-1]));
//Now write the register
writereg500(brd,6);
```

To do the same in Windows:

```
//Set board to use SAB register 0
brd->SWRpointer->Reg6SAB0.SABREG = 0x0;
//Set the actual value
brd->SWRpointer->Reg6SAB0.SABCLK = src;
//Copy the address of the SAB register into the Regptr array
brd->SWRpointer->Regptr[6] = (ULONG *) &brd->SWRpointer->Reg6SAB0;
//Now write the register
writereg500(brd, 6);
```

3.3.3.4 Software Register Contents

In DOS, the entire bit pattern of a software register can be written with a single statement by using the following:

```
*regptr[(Board Number-1)][Register Number]
```

Where "Board Number-1" has the value of 0 to 7 and "Register Number" has the value 0 to 3 or 6. Caution should be used when doing this as ALL bits values are determined by the value of the constants assigned. This structure is set up by calling the PDA500 function *setupbrd500*.

In Windows the entire bit pattern of a software register can be written with a single statement by using the following:

```
*SWR.Regptr[Register Number]
```

The SWR Variable is a variable of type P500SWR and "Register Number" has the value 0 to 3 or 6. The SWR Variable is declared by the programmer and is setup by calling the PDA500 function *setupregisters500*.

3.3.4 Data Acquisition Techniques

3.3.4.1 Data Acquisition for Multiple PDA500 Boards

The PDA500 function library has three functions for handling data acquisition: *acqRAMdata500*, *acqSABdata500* and *acqPCIdata500*. The only difference between these functions is the destination of the acquired data. The data acquired with *acqRAMdata500* is stored in the PDA500 signal RAM, with *acqSABdata500* data is transferred over the SAB, and with *acqPCIdata500* the data is transferred directly to the PCI bus. These functions have some limiting features:

1. Cannot properly handle simultaneous data acquisition on more than one board.
2. Do not allow for any abnormal acquisition termination except for a key pressed by the operator.

To handle these types of situations users should implement their own acquisition function with the code shown below (without line numbers):

(1) mode500(BrdxP500,Acq1Mode);	/* Board x is now in data acquisition mode */
(2) while(acqcomp500(BrdxP500));	/* Check for board x data acquisition complete */
(3) mode500(BrdxP500,StbyMode);	/* Return board x to standby mode */

For BrdxP500 above substitute Brd1P500, Brd2P500, etc. as appropriate. For multiple boards in a master/slave configuration, the proper mode switching sequence must be used to ensure proper operation. The following explanation assumes that the master is configured as the lowest number board (Brd1P500) and the slave boards are higher board numbers (Brd2P500, Brd3P500, etc.). The following will assume only one slave board.

Both boards are put into the acquisition mode with instruction number 1 as shown in the code above. This must be done from the largest to smallest board number. A loop must be generated for instruction 2 to check for the completion of data acquisition. When all active boards are finished with data acquisition, instruction number 3 should be executed for both boards starting with the largest numbered board and ending with the smallest numbered board. The sequence of slave then master must be observed so that it is guaranteed that the master does not attempt to provide a trigger to the slave board, before the slave is ready to begin acquisition.

3.4 Programming PDA500 Verses DA500A

This section describes the differences between programming for the PDA500 and the previous ISA bus version, the DA500A. This section is specifically intended for programmers who are upgrading applications written for the DA500A to be used with the PDA500. The software implementation of the two boards is very similar. The only significant difference from a software point of view is that the DA500A is accessed from fixed address spaces and the PDA500 is a Plug and Play configured board. For this reason the address spaces of the PDA500 must be retrieved from the system. The method for doing this is simple, yet slightly different for Windows and DOS based applications, so this subject will be discussed separately in following sections. Otherwise, if the software functions provided by Signatec have been used to create the DA500A application the process should be simple. Generally the only modification required is to change function names. For example the DA500A clock setting function *clkfreq5* is changed to *clkfreq500* for the PDA500.

3.4.1 DOS

In the DOS operating system, the DA500A software registers are setup using the function *SetupRegisters*. The board is then accessed using the appropriate Brd# variable that has been set to the DA500A IO address. For example if a DA500A is set at IO address 0x300 (Brd1) then the DA500A can be accessed using that variable.

```
SetupRegisters();       //allocate software registers
mode5(Brd1,7);          //place board in Standby mode
```

For the PDA500 the *SetupRegisters* function is replaced with *setupbrd500* and a call to the function *findbrd500* is added to retrieve Plug and Play parameters from the system. The parameters for a particular PDA500 can be found using the serial number of the board, or the parameters for any board can be found by using the instance number. For example, to find the first PDA500 in the system, use findbrd500(&Brd1P500,0);. Notice also that the Brd1P500 variable is prefixed by an & symbol. This is because the IO address of the PDA500 found will be placed in Brd1P500. This variable can then be used to access the PDA500 found.

```
setupbrd500();                  //Initializes the software registers
findbrd500(&Brd1P500,123456);   //get the IOBase of the PDA500 with serial number 123456
mode500(Brd1P500,7);            //place the PDA500 in Standby mode
```

3.4.2 Windows

In the Windows operating system an application for the DA500A board calls the *SetupRegisters5* function to set up the software registers and calls *OpenDA5ADevice* to open the driver. The DA500A code uses the Brd1 identification variable to identify the board. At the end of the DA500A application the function *CloseDA5ADevice* must be called.

```
SetupRegisters5();          //Setup Software Registers
OpenDA5ADevice(Brd1);       //Open the Driver
mode5(Brd1,7);              //Turn the Board to Standby Mode
CloseDA5ADevice(Brd1);      //Close the Driver
```

The software for the PDA500 must declare two variables before the PDA500 can be accessed. The first is of type P500SWR and contains storage for the software registers and resources used by a particular PDA500. The second is of type P500ID and contains the handle to the device driver and a pointer to the P500SWR variable. The *setupregisters500* function is then called with two parameters: The first is the address of the declared P500SWR variable, the second is either the PDA500 number which is 1 based (1-16) or the serial number of a particular board. This call is equivalent to the *SetupRegisters5* call of the DA500A program to set up the software register.

With the register set up, the application must open the device driver. Calling *opendevice500* with the software register variable (BRD1SWR) and the identification variable (Brd1ID) as parameters is equivalent to the DA500A *OpenDA5ADevice*. The pointer to the Identification variable is used as the identifier in the PDA500 functions. As with the DA500A, the device should be closed at the end of the application with the function *closedevice500*.

Many PDA500 functions require one additional parameter to their DA500A equivalent. The extra parameter is called rw and defines whether the function should access the hardware registers or only access the software register (no board access). In general, if the DA500A function call used 'Brd1' (Brd2, Brd3, etc.) as the first parameter, then the PDA500 function should use WRITE. If the DA500A function call used '1' (2, 3, etc.) then the PDA500 function should use READ).

```
P500SWR Brd1SWR;                    //Declare Software Register Variable
P500ID *Brd1P500, Brd1ID;           //Declare Board Identification Variables

setupregisters500(&Brd1SWR,123456); //Get Board Resources and Setup Software Registers
opendevice500(&Brd1SWR,&Brd1ID);    //Open a Handle to the Device
Brd1P500=&Brd1ID;                   //Get a Pointer to the ID Variable

mode500(Brd1P500,7,WRITE);          //Pass in Pointer and WRITE to PDA500

closedevice500(Brd1P500);           //Close Handle to Device
```

3.5 PDA500 C LIBRARY FUNCTIONS

The PDA500 library functions perform a variety of operations that are grouped by functionally as follows:

Data Acquisition Routines

acqcomp500	*acqPCIdata500*	*acqRAMdata500*	*acqSABdata500*
acqPCIdatabuffers500	*acqRAMdataDMA500*		

FFT Routines

fftstats500	*realfft500*	*realfftmag500*	

File Routines

brd2disk500	*disk2brd500*	*disk2pc500*	*readset500*
writeset500	*voltrange500*		

Hardware Setting Routines

attnset500	*autoaddrst500*	*brdname500*	*brdnum500*
chanset500	*clkdivbyfive500*	*clkdivide500*	*clkfreq500*
clock500	*hyst500*	*mode500*	*nulloff500*
offset500	*presamples500*	*pwdownor500*	*pwrdy500*
readEPROM500	*SABconf500*	*segmentdetect500*	*segmentsize500*
setendadd500	*setstartadd500*	*setupbrd500*	*temperature500*
trig500	*trigdelay500*	*triglevel500*	*trigmode500*
trigslope500	*trigsoft500*	*trigsource500*	*writeEPROM500*
writereg500	*writeregs500*		

Hardware Monitor Routines

fifoflag500	*readreg500*		

Interrupt Routines

cancelACFint500	*enableACFint500*	*enablepcint500*	*sendEOI*
setuppciint500	*uninstallpciint500*	*waitforACFint500*	

Memory Manipulation Routines

brd2pc500	*writeram500*		

PCI Configuration Routines

IObase500	*findbrd500*	*membase500*	

PCI DMA Transfer Routines

allocatedmabuffer500	*allocatedmabuffers500*	*brd2pcdma500*	*dmaint500*
enabledma500	*freeallbuffers500*	*freedmabuffer500*	*freedmabuffers500*

Windows Device Driver Routines

closedevice500	*opendevice500*	*setupregisters500*	

3.5.1 acqcomp500

Form

DOS	Short acqcomp500(unsigned short brd);
Windows	SHORT acqcomp500(P500ID * brd, BYTE rw);

Description

The function *acqcomp500* checks the status of the PDA500 for an acquisition complete flag (ACF) and returns the flag status.

The use of this function is not required for normal, single-board data-acquisition using *acqSABdata500* or *acqRAMdata500*. The function *acqcomp500* is useful when acquiring data on more than one PDA500 board. See section 3.3.4.1.

DOS

The parameter <brd> selects which PDA500 board will be operated. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then the value of the ACF is read from the PDA500 board and returned. If 1 through 8 is passed, the value of the ACF last read is returned. The meaning of the value returned is as follows:

0	Data acquisition not complete.
1	Data acquisition complete.
-1	Error

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

<rw> is the READ/WRITE flag. If READ is passed, the current value of the ACF in the software registers is returned. If WRITE is passed, the current value of the ACF is read from the board and returned.

Return Value

If WRITE is passed in <rw> then the value of the ACF is read from the PDA500 board and returned. If READ is passed in <rw>, the value of the ACF last read is returned. The meaning of the value returned is as follows:

0	Data acquisition not complete.
1	Data acquisition complete.
-1	Invalid <brd> passed.

3.5.2 acqPCIdata500

Form

DOS	long acqPCIdata500(unsigned short brd, long length, unsigned long *dest);
Windows	LONG acqPCIdata500(P500ID *brd, ULONG len, ULONG phyaddr);

Description

The function *acqPCIdata500* acquires data with the current board parameters from the PDA500 directly to the PCI bus. The digitized data is stored in on board First-In-First-Out (FIFO) buffers on the PDA500 to be read from the PCI bus. The maximum acquisition rate in this mode is dependent on the actual, overall, sustained transfer rate of the PCI bus and the speed of the specific computer in which the board is installed. If the rate of acquisition is faster than the rate at which the data is transferred from the board, the FIFO will fill, resulting in lost data samples.

DOS

The parameter <brd> selects which PDA500 board will be operated. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

The parameter <length> specifies the number of samples to acquire directly to the PCI bus.

The parameter <*dest> specifies the destination address in PC memory. The amount of memory space required for the array is equal to four times <length> in bytes.

Return Value

The function *acqPCIdata500* returns the following error codes:

0	Data acquired successfully.
-1	The specified board is not valid.
-2	The PDA500 FIFO became full at some point during the acquisition, indicating a data error.

Related Functions

acqSABdata500, *acqRAMdata500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <len> specifies the number of samples to acquire directly to the PCI bus.

The parameter <phyaddr> is the physical location in memory where the data is to be transferred. This memory location is a physical address returned by *allocatedmabuffer500*. Use the virtual address associated with the physical address to access the memory in user space.

The thread running *acqPCIdata500* cannot be terminated early in Windows 9x. In Windows 9x this function will use all system resources while waiting for a trigger, if this is not desired then use *acqPCIdatabuffers500* instead

which does not do this. This is not the case in Win NT/2000 and can be safely terminated at any time. If the thread is terminated then all dma buffers are automatically freed.

It is possible to acquire data to the PCI bus directly from more than 1 board at the same time in Windows NT/2000 using multiple threads. This is not possible in Windows 9x with this function but can be done using *acqPCIdatabuffers500*.

Return Value

The function *acqPCIdata500* returns the following error codes:

0	Data acquired successfully.
-1	The specified board is not valid.
-2	The PDA500 FIFO became full at some point during the acquisition, indicating a data error.

Related Functions

acqSABdata500, *acqRAMdata500*

3.5.3 acqPCIdatabuffers500

Form

DOS	N/A
Windows	LONG acqPCIdatabuffers500(P500ID *brd, XFER_INFO_STRUCT * xferlist)

Description

The function *acqPCIdata500* acquires data with the current board parameters from the PDA500 directly to the PCI bus. The digitized data is stored in on-board First-In-First-Out (FIFO) buffers on the PDA500 to be read from the PCI bus. The maximum acquisition rate in this mode is dependent on actual, overall, sustained transfer rate of the PCI bus and the speed of the specific computer in which the board is installed. If the rate of acquisition is faster than the rate at which the data is transferred from the board, the FIFO will fill, resulting in lost data samples. This function differs from *acqPCIdata500* in that it allows multiple buffers to be used with the PCI acquisition. This function is required if buffer lengths are greater than 64 Megabytes or if one buffer of the desired size cannot be obtained using the *allocatedmabuffer500* function.

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <xferlist> is a pointer to a linked list containing information about each of the buffers. This list is created by the function *allocatedmabuffers500* and the data can be accessed by using the VirtAddress parameter of each structure in the list. For an example of how to use the function and access the data, see below.

The thread running *acqPCIdatabuffers500* cannot be terminated early in Windows 9x. This is not the case in Win NT/2000 and can be safely terminated at any time. If the thread is terminated then all dma buffers are automatically freed.

It is possible to acquire data to the PCI bus directly using this function from more than 1 board at the same time using multiple threads.

Return Value

The function *acqPCIdatabuffers500* returns the following error codes:

0	Data acquired successfully.
-1	The specified board is not valid.
-2	The PDA500 FIFO became full at some point during the acquisition, indicating a data error.

Related Functions

acqPCIdata500, *allocatedmabuffers500*, *freedmabuffers500*, *acqSABdata500*, *acqRAMdata500*

Example

```
//console example illustrating the setup and use of a multiple pci //acquisition
#include <windows.h>
#include "pda500win.h"
#include <stdio.h>
#include <conio.h>

void main(void)
{
//Pointers to linked list of structures
XFER_INFO_STRUCT * temp, *first;
//Board ID and SWR
P500ID id;
P500SWR swr;
//Misc variables
ULONG len;
ULONG count;

//length of requested buffer in bytes (2meg)
len=1024*1024*2;

//open device
setupregisters500(&swr,1);
opendevice500(&swr,&id);

//set mode and allocate buffer
mode500(&id,StbyMode,WRITE);
printf("Requested 0x%x Bytes\n",len);
first=allocatedmabuffers500(&id,&len);
printf("Allocated 0x%x Total Bytes\n\n",len);
//set temporary pointer
temp=first;

//print out info for all spaces allocated
for(;;)
{
        printf("\nLength: %u",temp->Length);
        printf("\nPhysAddress: 0x%x",temp->PhysAddress);
        printf("\nVirtAddress: 0x%x\n",temp->VirtAddress);
```

```
        if(temp->Next==NULL)
                break;
        temp=temp->Next;
}

//setup board
offset500(&id,128,WRITE);
triglevel500(&id,128,WRITE);
trigmode500(&id,0,WRITE);
trigsource500(&id,0,WRITE);
attnset500(&id,0,WRITE);
clkdivide500(&id,2,WRITE);
clkfreq500(&id,0,WRITE);
//acquire the data to the buffers
acqPCIdatabuffers500(&id, first);
//reset pointer
temp=first;

//wait to print out data
getch();

//print out some data - which illustrates how to access data
for(;;)
{       UCHAR * datapointer;
        datapointer=(UCHAR *)temp->VirtAddress;

        printf("\nStart:");
        for(count=0;count<10;count++)
                printf("%x ",datapointer[count]);
        printf("\nEnd:");
        for(count=((temp->Length)/2)-10;count<((temp->Length)/2);count++)
                printf("%x ",datapointer[count]);
        printf("\n");
        if(temp->Next==NULL)
                break;
        temp=temp->Next;
}

//free the buffers
freedmabuffers500(&id,first);

//close the device and exit
mode500(&id,OffMode,WRITE);
closedevice500(&id);
getch();
}
```

3.5.4 acqRAMdata500

Form

DOS	short acqRAMdata500(unsigned short brd);
Windows	SHORT acqRAMdata500(P500ID * brd);

Description

The function *acqRAMdata500* acquires data with the current board parameters to the PDA500 signal RAM and returns after data has been acquired, a key has been pressed, or on an error.

This function will acquire data on only one board at a time. For acquisition on multiple boards see section 3.3.4.1.

DOS

The parameter <brd> selects which PDA500 board will be operated. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

Return Value

The function *acqRAMdata500* returns the following error codes:

0	Data acquisition successful.
-1	Error condition.
-2	Key Pressed.

Related Functions

acqSABdata500, *acqPCIdata500*

Windows

The Windows version of the function will also return if either mouse button is pressed.

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

Return Value

0	Data acquisition successful.
-1	Invalid <brd> passed.
-2	Key or mouse button pressed.

Related Functions

acqSABdata500

3.5.5 acqRAMdataDMA500

Form

DOS	N/A
Windows	LONG acqRAMdataDMA500 (P500ID *brd, ULONG startsample, ULONG len, ULONG phyaddr)

Description

The function *acqRAMdataDMA500* acquires data with the current board parameters to the PDA500 signal RAM and then transfers the data to the PC memory using a DMA transfer. The function will transfer all the acquired data.

This function eliminates the delay between the data acquisition function call and the DMA transfer function call by doing both operations within one device driver function. NOTE: The function *enableACFint500* should not be called before and after calling this function.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <startsample> specifies the address at which the acquisition to the PDA500 will begin and the first sample to be transferred to the PC.

The parameter <len> is the number of samples to acquire and transfer.

The parameter <phyaddr> is the address of the physical memory where the DMA transfer is to place the data. This address is returned by *allocatedmabuffer500*.

The thread running *acqRAMdataDMA500* cannot be terminated early in Windows 9x. This is not the case in Win NT/2000 and can be safely terminated at any time using *cancelACFint500* or the Win32 function TerminateThread. If the thread is terminated while the dma transfer is occurring, all allocated dma buffers are automatically freed.

It is possible to transfer data through the PCI bus directly from more than 1 board at the same time in Windows NT/2000 using multiple threads. This is not possible in Windows 9x.

Return Value

The function will return a -1 if an error occurs, otherwise the number of bytes transferred.

Related Functions

acqRAMdata500, *brd2pcdma500*

3.5.6 acqSABdata500

Form

DOS	short acqSABdata500(unsigned short brd);
Windows	SHORT acqSABdata500(P500ID * brd);

Description

The function *acqSABdata500* acquires data with the current board parameters to the Signatec Auxiliary Bus and returns after data has been acquired, a key has been pressed, or on an error has occurred.

This function is the same as *acqRAMdata500* with the exception of the acquired data destination. This function will acquire data on only one board at a time. For acquisition on multiple boards see section 3.3.4.1 Data Acquisition Techniques.

DOS

The parameter <brd> selects which PDA500 board will be operated. Parameter <brd> can take on the values Brd1P500 through Brd8P500. The function returns when the active memory is full.

Return Value

The function *acqSABdata500* returns the following error codes:

0	Data acquired successfully.
-1	Error condition.
-2	Acquisition aborted by key press.

Related Functions

acqRAMdata500, *SABconf500*

Example

This example sends 256k of acquired data from board #1 to the SAB.

/* setup external device to receive Auxiliary Bus data */

trig500(Brd1P500,Int,AC,Pos,Single);	/* setup trigger parameters */
clock500(Brd1P500,0,0,0);	/* setup internal clock to 500 MHz */
setstartadd500(Brd1P500,0)	/* set start address to zero */
setendadd500(Brd1P500,16384)	/* setup end address to 16k */
SABinit500(Brd1P500,1,1,0,0,0)	/* setup SAB to be board number 1 and use 64 bit transfers */
acqSABdata500(Brd1P500);	/* acquire data to SAB */

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*. The function returns when the active memory is full.

Return Value

0	Data acquisition successful.
-1	Invalid <brd> passed.
-2	Acquisition aborted by key press or mouse button.

Related Functions

acqRAMdata500, *SABinit500*

3.5.7 allocatedmabuffer500

Form

DOS	N/A
Windows	PULONG allocatedmabuffer500(P500ID * brd, PULONG physaddress, ULONG len);

Description

The function *allocatedmabuffer500* is used to allocate a contiguous space in PC memory for a DMA transfer.

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <physaddress> is a pointer that is filled by the function with the physical address of the memory allocated. This value must be stored for later use by the function *brd2pcdma500* and *freedmabuffer500*.

The parameter <len> specifies the number of bytes to allocate.

Return Value

The return value is a pointer to the virtual address of the DMA buffer. This pointer can be used by Win32 applications to access the memory transferred by the DMA operation. A NULL pointer is returned if the memory space could not be allocated.

Related Functions

freedmabuffer500, *brd2pcdma500*

3.5.8 allocatedmabuffers500

Form

DOS	N/A
Windows	XFER_INFO_STRUCT * allocatedmabuffers500(P500ID * brd, ULONG * totallength)

Description

The function *allocatedmabuffers500* is used to allocate multiple contiguous spaces in PC memory for a direct PCI acquisition transfer.

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <totallength> is a pointer to a value, which indicates in bytes the total length of the buffers needed. The total length that the function could allocate replaces the original value.

Return Value

The return value is a pointer to a linked list of structures which contain information about each of the buffers. This pointer is then passed to the *acqPCIdatabuffers500* function which acquires directly to these buffers. A NULL pointer is returned if the memory space could not be allocated. See the function *acqPCIdatabuffers500* for an example of how to access the data.

Related Functions

freedmabuffers500, *acqPCIdatabuffers500*

3.5.9 attnset500

Form

DOS	short attnset500(unsigned short brd, unsigned short val);
Windows	SHORT attnset500(P500ID * brd, UCHAR val, BYTE rw);

Description

The function *attnset500* allows the user to control the input signal attenuation on the PDA500. The attenuator setting is from 0 to 26 dB in steps of 2. The function automatically handles the switch over between high and low gain input amplifiers and the signal attenuator. A calibration file is provided with each PDA500 board that lists the full-scale voltage input for each attenuator setting for channel 1 and channel 2.

DOS

The parameter <brd> selects whether this function writes to the attenuator register (ATTN) or returns the attenuator value last written to the board. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <val> is written to the attenuator register on the corresponding PDA500 board. When <brd> takes on the value 1 through 8, the value of the voltage-range register is not changed, rather, the last value written to the corresponding PDA500 board is returned.

The parameter <val> can take on values from 0 through 26 in steps of 2 and represents the nominal attenuation setting in dB.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then <val> is returned. If 1 through 8 is passed in <brd>, the last value written to the corresponding PDA500 board voltage-range setting is returned. If an invalid value for <brd> is passed, the function returns -1.

Related Functions

voltrange500

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <val> can take on values from 0 through 26 in steps of 2 and represents the attenuator setting in dB.

<rw> is the READ/WRITE flag. If READ is passed, the current value of the voltage-range setting in the software registers is returned. If WRITE is passed then <val> is written to the board.

Return Value

If <rw> is WRITE then <val> is returned. If <rw> is READ the voltage range is read from the software registers and returned. If <brd> is invalid then -1 is returned.

Related Functions

voltrange500

3.5.10 autoaddrst500

Form

DOS	short autoaddrst500(unsigned short brd, unsigned char val);
Windows	SHORT autoaddrst500(P500ID * brd, UCHAR val, BYTE rw);

Description

The function *autoaddrst500* is used to cause the address counter on the PDA500 board to be automatically reset to zero when the board is put into the Standby Mode. Normally the address counter value is controlled by the function *setstartadd500*. Enabling the automatic reset feature is useful in applications where repeated acquisitions are made to the beginning of the RAM address.

DOS

The parameter <brd> selects whether this function writes to the automatic reset register bit or returns the value last written to the register. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <val> is written. When <brd> takes on the value 1 through 8 the value in the register is not changed, rather, the last value written to the corresponding PDA500 board register is returned.

The parameter <val> can take on values from 0 or 1 to disable or enable automatic address counter reset.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then <val> is returned. If 1 through 8 is passed in <brd>, the last value written to the corresponding PDA500 register is returned. Returns -1 if <brd> is invalid.

Related Functions

setstartadd500, *setendadd500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <val> can take on values from 0 or 1 to disable or enable the automatic reset feature.

<rw> is the READ/WRITE flag. If READ is passed, then the last value written to the ACRST bit of Register 2 returned. If WRITE is passed, then <val> is written to the board and returned.

Return Value

If <rw> is WRITE then <val> is returned. If <rw> is READ, the last value written to the software registers is returned. If <brd> is invalid, then -1 is returned.

Related Functions

setstartadd500, *setendadd500*

3.5.11 brd2disk500

Form

DOS	long brd2disk500(unsigned short brd, unsigned long startsample, unsigned long len, char *filename);
Windows	LONG brd2disk500(P500ID * brd, ULONG startsample, ULONG len, CHAR * filename);

Description

The function *brd2disk500* copies a section of PDA500 signal RAM to a disk file and attaches a 256-byte header containing board setup information and comments.

DOS

The parameter <brd> selects which PDA500 board is the source of the data to be transferred. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

The parameter <startsample> specifies the first sample of PDA500 data to be copied. The parameter <len> specifies the number of data samples to be copied. The <startsample> parameter must be a multiple of 8 and the <len> parameter must be a multiple of 64. The range of values for both parameters is shown in the table below.

Memory Installed	<startsample>	<len>
1 Meg	0 - 1,048,568	0 - 1,048,576
4 Meg	0 - 4,194,296	0 - 4,194,240

The parameter <filename> is the name of the file that will be saved. The file header is described in the function *writeset500*.

Return Value

Returns -1 if <brd> is not valid. Returns -2 if the file <filename> cannot be opened. Otherwise, returns the number of samples that were written.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <startsample> specifies the first sample of PDA500 data to be copied. The parameter <len> specifies the number of data samples to be copied. The <startsample> parameter must be a multiple of 8 and the <len> parameter must be a multiple of 64. The range of values for <startsample> and <len> depend upon the type of PDA500 board being used. The range of values is shown in the following table.

Memory Installed	<startsample>	<len>
1 Meg	0 - 1,048,568	0 – 1,048,576
4 Meg	0 - 4,194,296	0 – 4,194,240

The parameter <filename> is the name of the file that will be saved. The file header is described in the function *writeset500*.

Return Value

Returns -1 if <brd> is not valid. Returns -2 if the file <filename> cannot be opened. Otherwise, returns the number of samples that were written.

3.5.12 brd2pc500

Form

DOS	long brd2pc500(unsigned short brd, unsigned long startsample, unsigned long len, unsigned char *dest);
Windows	LONG brd2pc500(P500ID * brd, ULONG startsample, ULONG len, UCHAR * dest);

Description

The function *brd2pc500* copies a section of PDA500 signal RAM to an array of RAM in the PC's memory using memory-mapped transfers. The function *brd2pcdma500* transfers data using DMA transfers.

DOS

The parameter <brd> selects which PDA500 board is the source of the data to be transferred. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

The parameter <startsample> specifies the first sample of PDA500 data to be copied and must be set to a multiple of 8.

The parameter <len> specifies the number of data samples to be copied and must be a multiple of 64. The range of values for <startsample> and <len> depend upon the type of PDA500 board being used. The range of values is shown in the following table.

Memory Installed	<startsample>	<len>
1 Meg	0 - 1,048,568	0 - 1,048,576
4 Meg	0 - 4,194,296	0 - 4,194,240

The parameter <*dest> specifies the destination address in PC memory. The amount of memory space required for the array is equal to <len> in bytes.

Return Value

The function *brd2pc500* returns the number of samples copied, or -1 on an error.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <startsample> specifies the first sample of PDA500 data to be copied and must be set to a multiple of 8.

The parameter <len> specifies the number of data samples to be copied and must be a multiple of 64. The range of values for <startsample> and <len> depend upon the type of PDA500 board being used. The range of values is shown in the following table.

Memory Installed	<startsample>	<len>
1 Meg	0 - 1,048,568	64 - 1,048,576
4 Meg	0 - 4,194,296	64 - 4,194,240

The parameter <*dest> specifies the destination address in PC memory. The amount of memory space required for the array is equal to <len> in bytes.

Return Value

The function *brd2pc500* returns the number of samples copied, or -1 on an error.

Example

```
//Windows code segment which shows how to initialize, setup, and acquire data:
//declare variables
P500ID * PBrdID, BrdID;                          //declare ID variables
P500SWR BrdSWR;                                  //declare SWR variable
UCHAR * data;                                    //declare data pointer
//structure initialization
setupregisters500(&BrdSWR, 1);                   //initialize SWR structure using first PDA500
opendevice500(&BrdSWR, &BrdID);                  //initialize ID structure
PBrdID=&BrdID;                                   //assign pointer
data=(UCHAR *)malloc(32768*sizeof(UCHAR)); //allocate memory
//board setup
mode500(PBrdID, StbyMode, WRITE);                //set to Standby Mode
setstartadd500(PBrdID, 0, WRITE);                //set start address to 0
setendadd500(PBrdID, 32768, WRITE);              //set end address to 32768
offset500(PBrdID, 128, WRITE);                   //set offset of channel 1
triglevel500(PBrdID, 128, WRITE);                //set trigger level
//acquire and receive data
acqRAMdata500(PBrdID);                           //acquire data
brd2pc500(PBrdID, 0, 32768, data);               //get data from PDA500
//operate on data

//close device
mode500(PBrdID, 0, WRITE);                       //set to Off Mode
closedevice500(PBrdID);                          //release resources
```

3.5.13 brd2pcdma500

Form

DOS	void brd2pcdma500(unsigned short brd, unsigned long startsample, unsigned long len, unsigned long *dest);
Windows	LONG brd2pcdma500(P500ID *brd, ULONG startsample, ULONG len, ULONG phyaddr);

Description

The function *brd2pcdma500* copies a section of PDA500 signal RAM to an array of RAM in the PC's memory using Direct Memory Access (DMA) transfers. This function is similar to the *brd2pc500* function except the transfers are DMA.

DOS

The parameter <brd> selects which PDA500 board is the source of the data to be transferred. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

The parameter <startsample> specifies the first sample of PDA500 data to be copied and must be set as a multiple of 8.

The parameter <len> specifies the number of data samples to be copied and must be set as a multiple of 64. The range of values for <startsample> and <len> depend upon the type of PDA500 board being used. The range of values is shown in the following table.

Memory Installed	<startsample>	<len>
1 Meg	0 - 1,048,568	64 - 1,048,576
4 Meg	0 - 4,194,296	64 - 4,194,240

The parameter <*dest> specifies the destination address in PC memory. The amount of memory space required for the array is equal to <len> in bytes.

Return Value

None

Windows

Before the function *brd2pcdma500* can be used, a contiguous memory space must be allocated using the *allocatedmabuffer500* function.

The parameter <brd> selects which PDA500 board is the source of the data to be transferred. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

The parameter <startsample> specifies the first sample of PDA500 data to be copied and must be a multiple of 8.

The parameter <len> specifies the number of data samples to be copied and must be a multiple of 64.

Memory Installed	<startsample>	<len>
1 Meg	0 - 1,048,568	0 - 1,048,576
4 Meg	0 - 4,194,296	0 - 4,194,240

The parameter <phyaddr> is the physical location in memory where the data is to be transferred. This memory location is a physical address returned by *allocatedmabuffer500*. Use the virtual address associated with the physical address to access the memory in user space.
The thread running *brd2pcdma500* cannot be terminated early in Windows 9x. This is not the case in Win NT/2000 and can be safely terminated at any time. If the thread is terminated then all dma buffers allocated are automatically freed.

It is possible to transfer data through the PCI bus directly from more than 1 board at the same time in Windows NT/2000 using multiple threads. This is not possible in Windows 9x.

Return Value

The function *brd2pcdma500* returns 0 on success, or -1 on an error.

Example

//This console example acquires data to PDA500 ram, does a dma transfer to the pc, and displays some of the data.

```
#include <windows.h>
#include "pda500win.h"
#include <stdio.h>
#include <conio.h>

void main(void)
{
//Board ID and SWR
P500ID id;
P500SWR swr;
UCHAR * data;
ULONG address;
USHORT count;

//open device
setupregisters500(&swr,1);
opendevice500(&swr,&id);

//set mode and allocate buffer
mode500(&id,StbyMode,WRITE);
data=(UCHAR *)allocatedmabuffer500(&id,&address,4096);
if(data==NULL)
{
        closedevice500(&id);
        return;
}

//setup board
offset500(&id,128,WRITE);
triglevel500(&id,128,WRITE);
trigmode500(&id,0,WRITE);
trigsource500(&id,0,WRITE);
attnset500(&id,0,WRITE);
clkdivide500(&id,2,WRITE);
clkfreq500(&id,0,WRITE);
setstartadd500(&id,0,WRITE);
setendadd500(&id,4096,WRITE);
//acquire data
acqRAMdata500(&id);

//transfer data
brd2pcdma500(&id,0,4096,address);
```

```
//print out some data
printf("\nStart:");
for(count=0;count<10;count++)
        printf("%x ",data[count]);
printf("\nEnd:");
for(count=4096-10;count<4096;count++)
        printf("%x ",data[count]);
printf("\n");


//free the buffer
freedmabuffer500(&id,(ULONG *)data,address);

//close the device and exit
mode500(&id,OffMode,WRITE);
closedevice500(&id);
getch();
}
```

3.5.14 brdname500

Form

DOS	short brdname500(unsigned short brd);
Windows	N/A

Description

The function *brdname500* is used to determine what board name (Brd1P500-Brd8P500) is assigned to a particular board number. This function can be used to determine what I/O address is assigned to a board number.

The parameter <brd> is the board number for which the address will be returned. The possible values are 1 through 8.

Return Value

The numerical value of the I/O address associated with the board number passed in <brd> is returned. If brd is out of range -1 is returned.

Related Functions

brdnum500

3.5.15 brdnum500

Form

DOS	short brdnum500(unsigned short brd);
Windows	N/A

Description

The function *brdnum500* is used to determine what board number (1-8) has been assigned to a particular I/O address.

The parameter <brd> determines what address the board number will be returned for. Parameter <brd> can be passed the numerical value of the address or the global constants Brd1P500 through Brd8P500.

Return Value

The function *brdnum500* returns the board number associated with the I/O address passed in <brd>. If the I/O address has not been assigned as a board number the value of -1 will be returned. The possible return values are 1 through 8.

Examples

x=brdnum500(0x6200);	/* x will equal 1 */
x=brdnum500(Brd1P500);	/* x will equal 1 */
x=attnset500(brdnum500(Brd1P500),2);	/* x will equal voltage range setting for board 1 */

Related Functions

brdname500

3.5.16 cancelACFint500

Form

DOS	N/A
Windows	void cancelACFint500(P500ID * brd);

Description

The function *cancelACFint500* cancels the waiting state of the device driver that was initiated by the *waitforACFint500* function or *acqRAMdataDMA500*(NT/2000 only). Without the cancel function, the device driver will stay in a wait state until the PDA500 generates the ACF interrupt. This function is useful in situations where either the user or a unit of time must cancel an acquisition because without this function the call to *waitforACFint500* or *acqRAMdataDMA500*(NT/2000 only) will not allow the program to exit gracefully until an acquisition complete interrupt is generated by the PDA500.

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

In order to use the *cancelACFint500* function, a separate thread must be created. This is because the main thread is waiting for the *waitforACFint500* or *acqRAMdataDMA500*(NT/2000 only) function call to return signifying an ACF interrupt was generated. The *cancelACFint500* function will cause the *waitforACFint500* or *acqRAMdataDMA500*(NT/2000 only) function call to return before an interrupt was generated.

Related Functions

enableACFint500, *waitforACFint500*

Example

The following is an example of how to use the *cancelACFint500* function. This program's main thread sets up the board and controls the board's acquisition process. The second thread waits for a keypress to cancel the waiting.

If a keypress is never detected, then the main thread will continue to wait until an ACF interrupt is generated. If an ACF interrupt is generated, then the program will loop again. The critical sections are used to insure that *enableACFint500* is not called after the *cancelACFint500* function because this would allow the *waitforACFint500* function to wait on the ACF flag again.

```
#include <windows.h>
#include "pda500win.h"
#include "pda500macros.h"  //included for macro WaitForACFInt500
#include <stdio.h>
#include <conio.h>

DWORD WINAPI CancelThread(LPVOID temp);
CRITICAL_SECTION CriticalSection;
BOOL Cancel;

void main()
{
	P500ID id;
	P500SWR swr;
	HANDLE hThread;
	DWORD tid;
	DWORD ReturnedLength;

	setupregisters500(&swr,1);
	opendevice500(&swr,&id);

	//set up board
	mode500(&id,StbyMode,WRITE);
	clkfreq500(&id,0,WRITE);
	chanset500(&id,0,WRITE);
	offset500(&id,128,WRITE);
	triglevel500(&id,150,WRITE);
	attnset500(&id,0,WRITE);
	trigmode500(&id,0,WRITE);
	setstartadd500(&id,0,WRITE);
	setendadd500(&id,1048575,WRITE);

	Cancel=FALSE;
	InitializeCriticalSection(&CriticalSection);
	hThread=CreateThread(NULL,0,CancelThread,&id,0,&tid);	//Start the cancel thread

	for(;;)
	{
		//wait to enter
		EnterCriticalSection(&CriticalSection);
			if(Cancel==TRUE)
				break;
			enableACFint500(&id,1);			//enable interrupt
			mode500(&id,Acq1Mode,WRITE);	//start acquisition
		LeaveCriticalSection(&CriticalSection);
```

```
		//wait for the interrupt (notice the macro version of the WaitForACFInt500 function is called
		//because the Windows 95 version of waitforACFint500 automatically creates another thread
		//which is not desired in /this instance.)

		WaitForACFInt500(id.Brdhandle, &id.SWRpointer->Brdnum, &ReturnedLength);


		if(Cancel==TRUE)
			break;

		mode500(&id,StbyMode,WRITE);	//set to standby mode
		printf("Data Acquired\n");	//data is normally acquired here
		setstartadd500(&id,0,WRITE);	//reset start address (not needed if DMA is done)
	}

	//disable interrupt, close the device and clean up
	enablepcint500(&id,0);
	mode500(&id,OffMode,WRITE);
	closedevice500(&id);
	DeleteCriticalSection(&CriticalSection);

	printf("Program Finished\n");
	getch();
}

DWORD WINAPI CancelThread(LPVOID temp)
{
	P500ID *id=(P500ID *)temp;

	//wait for keyboard hit
	while (!_kbhit());

	//wait to enter
	EnterCriticalSection(&CriticalSection);
		//cancel acf int if waiting
		cancelACFint500(id);
		Cancel=TRUE;
	LeaveCriticalSection(&CriticalSection);

	printf("\nKey Hit\n");
	return 0;
}
```

3.5.17 chanset500

Form

DOS	short chanset500(unsigned short brd, unsigned char val);
Windows	SHORT chanset500(P500ID * brd, UCHAR val, BYTE rw);

Description

The function *chanset500* sets the PDA500 channel that will be digitized.

DOS

The parameter <brd> selects which PDA500 board is addressed. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <val> is written to the channel select bit (CHSEL) of register 0 for the specified PDA500 board. When <brd> takes on the value 1 through 8 the value is not changed, rather, the present setting of CHSEL for the designated PDA500 board is returned.

The parameter <val> can take on the value of 1or 2 which sets the input channel as either channel 1 or 2. A value f other than 2 sets the input channel to channel 1.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then <val> is returned. If 1 through 8 is passed in <brd>, the present value of CHSEL on the PDA500 board specified by <brd> is returned.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <val> can take on the value of 1 or 2 which sets the active input channel to 1 or 2. A value other than 2 sets the input channel to channel 1.

<rw> is the READ/WRITE flag. If READ is passed the current value of the CHSEL bit in software register 0 is returned. If WRITE is passed <val> is written to the CHSEL bit on the board.

Return Value

If WRITE is passed in <rw> then the value of the CHSEL is written to the PDA500 board and returned. If READ is passed in <rw>, the value of the CHSEL in the software registers is returned.

3.5.18 clkdivbyfive500

Form

DOS	Short clkdivbyfive500(unsigned short brd, unsigned char div5);
Windows	SHORT clkdivbyfive500(P500ID * brd, UCHAR div5, BYTE rw);

Description

The function *clkdivbyfive500* allows the user to turn on or off the clock divide by 5.

DOS

Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <div5> is written to the DIV5 register on the appropriate PDA500 board. When <brd> takes on the value 1 through 8, the board value is not changed, rather, the last value written to the board is returned.

The parameter <div5> can take on the values 0 - 1. With 0 disabling the clock divide by five and 1 enabling it.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd>, then <div5> is returned. If 1 through 8 is passed in <brd>, the last value written to the DIV5 register on the corresponding PDA500 is returned. If an error occurs a value of -1 is returned.

Related Functions

clkfreq500, *clock500*, *clkdivide500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function opendevice500.

The parameter <div5> can take on the values 0 - 1. With 0 disabling the clock divide by five and 1 enabling it.

<rw> is the READ/WRITE flag. If READ is passed, the last value of <div5> written to the board is returned. If WRITE is passed, <div5> is written to the board and returned.

Return Value

If WRITE is passed in <rw>, the value of the <div5> is written to DIV5 register on the PDA500 board and returned. If READ is passed in <rw>, the value last written to the DIV5 register is returned from the software registers. If an error occurs -1 is returned.

Related Functions

clkfreq500, *clock500*, *clkdivide500*

3.5.19 clkdivide500

Form

DOS	short clkdivide500(unsigned short brd, unsigned char div);
Windows	SHORT clkdivide500(P500ID * brd, UCHAR div, BYTE rw);

Description

The function *clkdivide500* allows the user to set the digitizer clock frequency divider value to one of 8 possible settings.

DOS

Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, is written to the CD register on the appropriate PDA500 board. When <brd> takes on the value 1 through 8, the board value is not changed, rather, the last value written to the board is returned.

The parameter can take on the values 0 - 7. This results in a clock frequency division of 2 on the internal 500 MHz, 200 MHz, or the external clock.

Parameter	Clock Divider
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

Return Value

If Brd1P500 through Brd8P500 is passed in <brd>, then is returned. If 1 through 8 is passed in <brd>, the last value written to the CD register on the corresponding PDA500 is returned. If an error occurs a value of -1 is returned.

Related Functions

clkfreq500, *clock500*, *clkdivbyfive500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter can take on the values 0 - 7. This results in a clock frequency division of 2 on the internal 500 MHz, 200 MHz or the external clock.

Parameter	Clock Divider
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

<rw> is the READ/WRITE flag. If READ is passed, the last value of written to the board is returned. If WRITE is passed, is written to the board and returned.

Return Value

If WRITE is passed in <rw>, the value of the is written to CD register on the PDA500 board and returned. If READ is passed in <rw>, the value last written to the CD register is returned from the software registers. If an error occurs -1 is returned.

Related Functions

clkfreq500, *clock500*, *clkdivbyfive500*

3.5.20 clkfreq500

Form

DOS	short clkfreq500(unsigned short brd, unsigned char freq);
Windows	SHORT clkfreq500(P500ID * brd, UCHAR freq, BYTE rw);

Description

The function *clkfreq500* allows the user to select which clock source to use as the base for the digitizer clock.

DOS

Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <freq> is written to the CS register on the appropriate PDA500 board. When <brd> takes on the value 1 through 8, the board value is not changed, rather, the last value written to the board is returned.

The parameter <freq> can take on the values 0 - 3. These values result in the selection of the following clock sources:

Parameter <freq>	Base Clock
0	500 MHz (Internal)
1	500 MHz (Internal)
2	200 MHz (Internal)
3	External Clock

Return Value

If Brd1P500 through Brd8P500 is passed in <brd>, then <freq> is returned. If 1 through 8 is passed in <brd>, the last value written to the CS register on the corresponding PDA500A is returned. If an error occurs a value of -1 is returned.

Related Functions

clkdivide500, *clock500*, *clkdivbyfive500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <freq> can take on the values 0 - 3. These values result in the selection of the following clock sources:

Parameter <freq>	Base Clock
0	500 MHz (Internal)
1	500 MHz (Internal)
2	200 MHz (Internal)
3	External Clock

<rw> is the READ/WRITE flag. If READ is passed, the last value of <freq> written to the board is returned. If WRITE is passed, <freq> is written to the board and returned.

Return Value

If WRITE is passed in <rw>, the value of the <freq> is written to CS register on the PDA500 board and returned. If READ is passed in <rw>, the value last written to the CS register is returned from the software registers. If an error occurs -1 is returned.

Related Functions

clkdivide500, *clock500*, *clkdivbyfive500*

3.5.21 clock500

Form

DOS	float clock500(unsigned short brd, unsigned char freq, unsigned char div, unsigned char div5);
Windows	FLOAT clock500(P500ID * brd, UCHAR freq, UCHAR div, UCHAR div5, BYTE rw);

Description

The function *clock500* allows the user to setup all digitizer clock control options with a single function.

DOS

Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, the parameter values are written to the corresponding register on the PDA500 board. When <brd> takes on the value 1 through 8, the board values are not changed, rather, the function returns the value of the effective digitizer frequency in MHz.

The parameter <freq> can take on the values 0 - 3 to select the clock source as shown below.

Parameter <freq>	Base Clock
0	500 MHz (Internal)
1	500 MHz (Internal)
2	200 MHz (Internal)
3	External Clock

The parameter <div5> takes the value 0 or 1 (OFF or ON) to set the clock divide by 5 off or on.

The parameter takes on the value 0 – 7, which sets the clock divider as shown below.

Parameter	Clock Divider
0	1
1	2

2	4
3	8
4	16
5	32
6	64
7	128

The possible clock frequencies for the PDA500 are shown in the table below.

Frequency (MHz)	Clock (MHz)	DIV5	Divider
500.0	500	OFF	1
250.0	500	OFF	2
200.0	200	OFF	1
100.0	500	ON	1
62.5	500	OFF	8
50.0	200	OFF	4
40.0	200	ON	1
31.25	500	OFF	16
25.0	200	OFF	8
20.0	200	ON	2
15.625	500	OFF	32
12.5	200	OFF	16
10.0	200	ON	4
7.8125	500	OFF	64
6.25	500	ON	16
5.0	200	ON	8
3.90625	500	OFF	128
3.125	500	ON	32
2.5	200	ON	16
1.5625	500	ON	64
1.25	200	ON	32
0.78125	500	ON	128
0.625	200	ON	64
0.3125	200	ON	128

Return Value

If Brd1P500 through Brd8P500 is passed in <brd>, then 1 is returned. If 1 through 8 is passed in <brd>, the effective digitizer frequency in MHz is returned. If an error occurs a value of -1 is returned.

Related Functions

clkdivide500, *clkfreq500*, *clkdivbyfive500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <freq> can take on the values 0 - 3. These values result in the selection of the following clock sources:

Parameter <freq>	Base Clock
0	500 MHz (Internal)
1	500 MHz (Internal)
2	200 MHz (Internal)
3	External Clock

The parameter <div5> takes the value 0 or 1 (OFF or ON) to set the clock divide by 5 off or on.
The parameter takes on the value 0 – 7, which sets the clock divider as shown below.

Parameter	Clock Divider
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

<rw> is the READ/WRITE flag. If READ is passed the effective digitizer frequency in MHz is returned. If WRITE is passed, the values are written to the board.

Return Value

If READ is passed in <rw>, the effective digitizer frequency in MHz is returned. If WRITE is passed in <rw>, the values are written to the board, and the effective digitizer frequency in MHz is returned. If an error occurs -1 is returned.

Related Functions

clkdivide500, *clockfreq500*, *clkdivbyfive500*

3.5.22 closedevice500

Form

DOS	N/A
Windows	SHORT closedevice500(P500ID * brd);

Description

The function *closedevice500* is used to release the handle for the PDA500 when it is no longer needed in the program.

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

Return Value

A 0 is returned on success. A -1 indicates an error.

Example

See *brd2pc500*.

Related Functions

setupregisters500, *opendevice500*

3.5.23 disk2brd500

Form

DOS	short disk2brd500(unsigned short brd, char *filename);
Windows	LONG disk2brd500(P500ID * brd, CHAR * filename);

Description

This function transfers data from a disk file previously stored with function *brd2disk500* to PDA500 RAM.

The amount of data loaded is determined by the memory size setting stored in the data file header. Besides transferring data, this function updates the software and hardware registers for the PDA500 board specified by <brd> to the values stored in the file header of the specified file. The function *readset500* is called to perform this task. See function *writeset500* for a description of the file header.

DOS

Parameter <brd> determines the PDA500 board number which is addressed. Allowable entries are Brd1P500 through Brd8P500. Parameter <filename> is the name of a data file. This function will open and close the file.

Return Value

The value returned is -1 if the named file <filename> cannot be opened, is not a valid PDA500 file, or if <brd> is invalid; otherwise, the number of samples of data loaded is returned.

Related Functions

brd2disk500

Example

```
 /* Loads data and register info from file signal500.dat to board 2 */

unsigned val;
val=disk2brd500(Brd2P500,"signal500.dat");
if (val==-1) cprintf("\nError!  Function Failed.");
else if (val==-2) cprintf("\nError!  File not PDA500 Format.");
else if (val==-3) cprintf("\nError!  Could not allocate temp. buffer in PC Memory.");
else cprintf("\n%u kilobytes loaded.",val);
cprintf("\nHit any key to continue..."); getch();
```

Windows

The parameter <brd> selects which PDA500 board will be addressed. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

Parameter <filename> is the name of a data file. This function will open and close the file.

Return Value

The value returned is -1 if the named file <filename> cannot be opened or if <brd> is invalid. If the file is not a valid PDA500 file, -2 is returned. If a temporary buffer could not be allocated in PC memory, a –3 is returned. Otherwise, the number of samples of data loaded is returned.

Related Functions

brd2disk500

3.5.24 disk2pc500

Form

DOS	int disk2pc500(char *filenam, unsigned char *dest, long startsample, unsigned short len);
Windows	LONG disk2pc500(CHAR * filenam, USHORT *dest, LONG startsample, ULONG len);

Description

This function copies signal data from a disk file that was stored with function *brd2disk500* to PC RAM.

Parameter <filenam> is the name of a data file that will be accessed.

Parameter <dest> is a pointer to an array into which the disk data will be transferred. The size of the array must be large enough to hold the data.

Parameter <startsample> is the starting sample in the data file.

Parameter <len> specifies the total number of samples to transfer and can take on any value up to 65536 in DOS or can be the maximum size of the board's memory in Windows. If the actual number of samples available in the file is less than <len> the function will terminate when all available samples have been transferred.

If the requested channel data is not in the disk file (as indicated by the channels stored byte in the file header) an error code is returned. The function *readset500* may be used to inspect the file header prior to calling this function.

Return Value

Negative return values represent errors as follows:

-1	file <filnam> could not be opened
-2	not a valid PDA500 file

Positive return values indicate the actual number of bytes of data that were transferred.

Related Functions

brd2disk500

3.5.25 dmaint500

Form

DOS	void interrupt dmaint500();
Windows	N/A

Description

The function *dmaint500* is a basic shell of an interrupt service routine used in conjunction with DMA transfers. When a DMA transfer is complete or is forced to abort by the system, the PDA500 generates an interrupt. A flag is set in the PDA500 controller register to identify the reason for the interrupt. An interrupt service routine must be installed in the system to identify the cause for the interrupt and take appropriate action. Before executing a DMA transfer, this interrupt handler must be installed using the function *setupciint500*.

3.5.26 enableACFint500

Form

DOS	N/A
Windows	SHORT enableACFint500(P500ID * brd, UCHAR val);

Description

The function *enableACFint500* is used to enable the acquisition complete interrupt. By calling this function before an acquisition, the PDA500 will signal the PC with an interrupt when the acquisition to the PDA500 is complete. This function is used in conjunction with the function *waitforACFint500,* which actually waits for the interrupt.

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <val> determines if the acquisition complete interrupt is to be turned on or off. A value of 0 passed to the function will turn the interrupt off while a value of 1 will turn the interrupt on.

Return Value

If <brd> is not valid -1 is returned. If there is an error writing to the device a -2 is returned. Otherwise, 0 is returned.

Related Functions

waitforACFint500

Examples

See *waitforACFint500*.

3.5.27 enabledma500

Form

DOS	short enabledma500(unsigned short brd, unsigned char val);
Windows	N/A

Description

The function *enabledma500* is used to indicate to the PDA500 board logic that transfers from data RAM will be Direct Memory Access (DMA) transfers.

The parameter <brd> selects which PDA500 board is addressed. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <val> is written to the DMA bit (DMA) of register 0 for the specified PDA500 board. When <brd> takes on the value 1 through 8 the value is not changed rather, the present setting of the DMA for the designated PDA500 board is returned.

The parameter <val> can take on the value of 0 or 1. DMA access is disabled when <val> is set to 0 and enabled when <val> is 1.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd>, then <val> is returned. If 1 through 8 is passed in <brd>, the present value of DMA on the PDA500 board specified by <brd> is returned.

3.5.28 enablepcint500

Form

DOS	short enablepcint500(unsigned short brd, unsigned char val);
Windows	N/A

Description

The function *enablepcint500* is used to enable the acquisition complete interrupt. By calling this function before an acquisition, the PDA500 will signal the PC with an interrupt when the acquisition to the PDA500 is complete. This function is used in conjunction with the function *waitforACFint500,* which actually waits for the interrupt.

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <val> determines if the acquisition complete interrupt is to be turned on or off. A value of 0 passed to the function will turn the interrupt off while a value of 1 will turn the interrupt on.

Return Value

If <brd> is not valid -1 is returned. If there is an error writing to the device a -2 is returned. Otherwise, 0 is returned.

Related Functions

waitforACFint500

Examples

See *waitforACFint500*.

3.5.29 fftstats500

Form

DOS	short fftstats500(unsigned short fftsize, float *magdata, float far *out);
Windows	INT fftstats500(USHORT fftsize, FLOAT * magdata, FLOAT * out);

Description

The function *fftstats500* calculates statistical information for the fft output data from the function *realfftmag500*. The statistics calculated by this function give the best results if the fft function *realfftmag500* is called using the Blackman-Harris window. For the results of this function to be meaningful, a narrow frequency band "signal" must exist in the data.

Parameter <fftsize> is a binary number representing the size of the fft from 256 to 8192 points in factors of two. This value must match the fftsize value that was used in the call to the function *realfftmag500*.

Parameter <magdata> is a pointer to magnitude squared data. This pointer is returned by the function *realfftmag500*.

Parameter <out> is a pointer to an array of 8 floating point numbers into which the output data (statistics) will be written. The statistical data format is as follows:

*out = "signal" amplitude (voltage squared)
*(out+1)= noise amplitude
*(out+2)= second harmonic amplitude
*(out+3)= third harmonic amplitude
*(out+4)= highest spurious signal amplitude
*(out+5)= relative frequency of highest energy signal component
*(out+6)= bin number with highest amplitude
*(out+7)= not used

The signal amplitude is computed from the highest bin and a number of bins on each side of the highest bin. The total bin width is equal to 9 bins for fft sizes less than 1024, 17 bins for fft sizes of 1k or 2k, and equal to 33 bins for fft sizes of 4k or 8k points. Since all amplitudes are in voltage squared units, they are proportional to power.

The frequency value *(out+5) must be multiplied by the clock frequency in megahertz to obtain the true signal frequency.

Return Value

If <fftsize> is not valid, -1 is returned. Otherwise 0 is returned.

Related Functions

realfftmag500

Examples

float *magdata,statdata[8],snr;

acqRAMdata500(Brd1P500,0);	/* acquire data */
mMagdata=realfftmag500(Brd1P500,4,4096,0,lintbl,1);	/* do 4k fft */
Fftstats500(4096,magdata,statdata);	/* get the stats */

```
snr=statdata[0]/statdata[1];
snr=10*log10(snr);
cprintf("\nSignal to noise ratio=%2.2f dB",snr);
cprintf("\nSignal frequency=%2.6f MHz",statdata[5] * clock500(brdnum12(Brd1P500),0,0));
```

3.5.30 fifoflag500

Form

DOS	short fifoflag500(unsigned short brd);
Windows	SHORT fifoflag500(P500ID * brd, BYTE rw);

Description

The function *fifoflag500* checks the status of the PDA500 flag which indicates that the on-board First In First Out (FIFO) data buffer has become full during an acquisition to the PCI bus. During a data acquisition to a destination other than the PCI bus, the hardware is mechanized such that the FIFO cannot become full. During an acquisition to the PCI bus, the FIFO is filled at the PDA500 sample rate and drained at the PCI transfer rate. If the drain rate is slower than the fill rate, the FIFO will become full. If this occurs data samples acquired during the time the FIFO is full will be lost, resulting in a missing time slice in the data record.

This function is only required when acquiring data directly to the PCI bus, and when the data acquisition is completed.

DOS

The parameter <brd> selects which PDA500 board will be operated. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then the value of the FF is read from the PDA500 board and returned. If 1 through 8 is passed, the value of the FF last read is returned. The meaning of the value returned is as follows:

0	FIFO did not fill, data valid.
1	FIFO became full some time during the acquisition. Data invalid.
-1	Error

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

<rw> is the READ/WRITE flag. If READ is passed, the current value of the FF bit in the software registers is returned. If WRITE is passed, the FF bit is read directly from the board.

Return Value

If <rw> is WRITE then the FF is read from the PDA500 board and returned. If <rw> is READ, the value of the FF is read from the software registers and returned. The meaning of the value returned is as follows:

0	FIFO did not fill, data valid.
1	FIFO became full some time during the acquisition. Data invalid.
-1	Invalid <brd> passed

3.5.31 findbrd500

Form

DOS	long findbrd500(unsigned short *brd, long SN);
Windows	N/A (see *setupregisters500* and *opendevice500* functions)

Description

The function *findbrd500* is used to locate and identify a PDA500 that is installed in the system. The PCI boards in the system are interrogated to determine the existence of a PDA500 board using the PDA500 Vendor Identification Number (0x10E8) and Device Identification Number (0x82B2). When a PDA500 is located, the function *findbrd500* sets the global variables "P500BUSNUM[x]" and "P500DEVICENUM[x]" where x is the number 0 through 7 corresponding to the <brd> identifier Brd1P500 through Brd8P500. These two global variables are used to gain access to the PDA500 command and control registers.

The parameter <brd> is used by findbrd12 as the identifier for the board that is found. This parameter can take the value Brd1P500 through Brd8P500. After executing *findbrd500*, the identifier (Brd1P500-Brd8P500) can be used by all other PDA500 software functions to gain access to the PDA500.

The parameter <SN> can be used in two ways. If the value of 0 through 8 is passed, *findbrd500* will find the corresponding PCI device number and return the Serial Number (SN) of that board. For example, if a 0 is passed, *findbrd500* will return the SN of the first PDA500 board it finds, if 1 is passed *findbrd500* returns the SN of the second PDA500, etc. This is useful when either single or multiple PDA500 boards are installed in a system and the SN of a board is not known.

The Serial Number of a board can also be passed in the parameter <SN>. This will insure that the desired board will be identified with a given Brd identifier. For example, if a Master and a Slave PDA500 are installed in a system, the function *findbrd500* can be called with brd=Brd1P500 and SN equal to the serial number of the Master board. This will insure that all PDA500 software functions can use the Brd1P500 as the identifier for the Master PDA500.

Return Value

If a value of 0 to 8 is passed in the parameter <SN>, the function *findbrd500* will return the SN of the corresponding device number. If a PDA500 board SN (written on the lower right had corner of every board) is passed in the parameter <SN>, then a 1 will be returned if the board is located. In any case, when a board is located, the function *findbrd500* initializes global variables required by other PDA500 functions to access the board. A negative number will be returned on an error condition. If no PCI BIOS is found in a system, a value of -1 will be returned. If no SN is specified, and *findbrd500* does not locate any PDA500 board, a value of -2 will be returned. If a SN is specified but not found, a value of -2 will be returned.

3.5.32 freeallbuffers500

Form

DOS	N/A
Windows	SHORT freeallbuffers500(P500ID * brd);

Description

The function *freeallbuffers500* is used to free the contiguous space in PC memory allocated using *allocatedmabuffer500* or *allocatedmabuffers500*. This function will only work in Windows NT/2000 because Windows 9x does not maintain an independent driver list of buffers. Normally the *freedmabuffer500* or *freedmabuffers500* functions should be used instead of this function and is intended to help the debugging process only.

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

Return Value

The return values for the function *freeallbuffers500* is as follows:

0	Failed to free memory.
1	Success.

Related Functions

allocatedmabuffer500, allocatedmabuffers500, freedmabuffer500, freedmabuffers500

3.5.33 freedmabuffer500

Form

DOS	N/A
Windows	SHORT freedmabuffer500(P500ID * brd, PULONG virtaddress, ULONG physaddress);

Description

The function *freedmabuffer500* is used to free the contiguous space in PC memory that was allocated using *allocatedmabuffer500*.

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

Both of the parameters <virtaddress> and <physaddress> are the parameters that were obtained by calling the *allocatedmabuffer500*. Neither parameter can be NULL in order for the memory to be freed properly.

Return Value

-1	Invalid <brd>
-1	Failed to free memory
0	Success

Related Functions

allocatedmabuffer500, *brd2pcdma500*

3.5.34 freedmabuffers500

Form

DOS	N/A
Windows	SHORT freedmabuffers500(P500ID * brd, XFER_INFO_STRUCT * xferlist)

Description

The function *freedmabuffers500* is used to free the contiguous memory spaces in PC memory that were allocated using *allocatedmabuffers500* function.

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <xferlist> is a pointer to a linked list of structures that were allocated using *allocatedmabuffers500*.

Return Value

-1	Invalid <brd>
-1	Failed to free memory
0	Success

Related Functions

allocatedmabuffers500, *acqPCIdatabuffers500*

3.5.35 hyst500

Form

DOS	short hyst500(unsigned short brd, unsigned char val);
Windows	SHORT hyst500(P500ID * brd, UCHAR val, BYTE rw);

Description

The function *hyst500* allows the user to select between two levels of trigger hysteresis. The hysteresis can be selected as either low level: 7 millivolts, or high level: 27 millivolts. The hysteresis voltages are at the input to the ADC.

DOS

The parameter <brd> selects whether this function writes to the HYS register or returns the value last written to the register. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <val> is written to the HYS register on the corresponding PDA500 board. When <brd> takes on the value 1 through 8 the value in the HYS register is not changed, rather, the last value written to the corresponding PDA500 board is returned.

The parameter <val> can take on the values of 0 to 1. Writing a value of 1 will set the hysteresis to the low value of 7 millivolts. Writing a value of 0 will set the hysteresis to the high value of 27 millivolts.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then <val> is returned. If 1 through 8 is passed in <brd>, the last value written to the corresponding PDA500 board HYS register is returned.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <val> can take on the values of 0 to 1. Writing a value of 1 will set the hysteresis to the low value of 7 millivolts. Writing a value of 0 will set the hysteresis to the high value of 27 millivolts.

<rw> is the READ/WRITE flag. If READ is passed, the current value of the HYS bit in the software registers is returned. If WRITE is passed, <val> is written to the HYS bit.

Return Value

If <rw> is WRITE then <val> is returned. If <rw> is READ, the value of the current mode control register is read from the software registers and returned. If -1 is returned then an invalid <brd> was passed.

3.5.36 Iobase500

Form

DOS	short Iobase500(unsigned short *brd, long SN);
Windows	N/A

Description

The function *Iobase500* is used to determine the base I/O address of a PDA500 board installed in a system. Many PCI compatible computers conform to the PCI "plug and play" standard. Therefore, the system may change the I/O address location used by the PDA500 to access the control registers. The function *Iobase500* writes the I/O address, which the system BIOS has assigned to the PDA500 brd variable. This function is automatically called inside *findbrd500*.

Note: In Windows the IObase can be found in the P500SWR structure by reading the Port2 element after the *setupregisters500* function has been called.

The parameter <SN> identifies the individual PDA500 board to be read. The SN is the six-digit serial number of the PDA500 board.

Executing this function writes the base I/O address of the board identified by SN to the global variable identified by the parameter <brd>. This parameter can take the values Brd1P500 through Brd8P500.

Return Value

The following error values are possible:

-1	PCI BIOS not found
-2	PDA500 with specified SN not found
-3	Configuration Register Error

A value greater than 0 returned indicates the I/O address is of the PDA500 board identified by SN.

Examples

The following example determines the base I/O address of PDA500 board serial number 102435 to be identified as Brd1P500.

```
Iobase500(&Brd1P500,102435);
```

The following example prints out the base I/O address of serial number 112436 without assigning a global board value.

```
printf("%d", Iobase500(NULL,102436));
```

3.5.37 membase500

Form

DOS	long membase500(unsigned short brd, long SN);
Windows	N/A

Description

The function *membase500* is used to determine the base memory address of a PDA500 board installed in a system. The PCI system will perform "Plug and Play configuration of the PDA500 on system start-up. Therefore, the memory address location used to map PDA500 RAM could be different each time the system is powered on. The function *membase500* writes the address that the system BIOS has assigned to the PDA500 to the global array P500MEMBASE for the specified board. This function is automatically called by *findbrd500*.

Note: In Windows the Membase can be found in the P500SWR structure by reading the Membase element after the *setupregisters500* function has been called.

The parameter <brd> specifies which board identifier will be associated with the specified SN and base memory address.

The parameter <SN> identifies the individual PDA500 board to be read. The SN is the six-digit serial number of the PDA500 board.

Return Value

If the specified SN is located, the function returns the base memory address of the PDA500. The error conditions are as follows:

-1	PCI BIOS not found in the system.
-2	Configuration register does not contain a memory address.
-3	Configuration register does not contain a memory address.

3.5.38 mode500

Form

DOS	short mode500(unsigned short brd, unsigned char val);
Windows	SHORT mode500(P500ID * brd, UCHAR val, BYTE rw);

Description

The function *mode500* allows the user to select one of eight operating modes for the PDA500 based on the value of the parameter <val>. The following mode names have been defined in PDA500.h and may be supplied to the parameter <val>: OffMode (0), AcqPCI (1), Acq1Mode (2), Acq2Mode (3), PCIMode (4), SABMode (5), WramMode (6), and StbyMode (7). Operation in the various modes is as follows:

OffMode : Removes power from most of the PDA500 circuitry so that the board has very low power dissipation. This is the default mode on power up and the user should return the board to this mode when the application software is exited. When the PDA500 is placed in OffMode, all hardware register information is lost.

AcqPCI : This mode is used to acquire data directly to the PCI bus. After entering this mode, the PDA500 begins sampling the input waveform after the trigger condition is met. In this acquisition mode the samples are written to the on-board FIFO memory. The PC must begin reading data through the PCI bus. If the FIFO is empty, on-board logic will disconnect the PCI burst transfer and signal a re-try to the PCI bus. The PCI bus continues to read the data until the ACF flag is set and an interrupt is sent to the PC. See section 2.7.2 for more information.

Acq1Mode : This is the main data acquisition mode. After entering this mode the PDA500 begins storing the input waveform samples to signal RAM after the next trigger condition is met. In this acquisition mode the samples will be stored in the signal memory on the PDA500 board. The PDA500 board will signal that the acquisition is complete by setting the Memory Full flag.

Acq2Mode : This data acquisition mode places the input waveform samples directly onto the Signatec Auxiliary Bus (SAB) rather than into on board signal RAM. To use this mode there must be a MEM500 memory board or other device for data storage connected to the SAB. Acquisition complete is signaled in the same manner as Acq1Mode.

PCIMode : This mode is used to access the signal RAM data from the PC. Data access is described in section 2.10.

SABMode : This mode allows the user to send acquired data from the PDA500 signal RAM to the Signatec Auxiliary Bus (SAB). Before entering this mode, the address counter on the PDA500 must be set to the desired

starting address (3.5.57). Data transfer across the bus with the specified width (SABconf500 function) will begin upon entering the SABMode.

WramMode : This mode is used to write data to the PDA500 board signal RAM.

StbyMode : This mode powers the acquisition circuitry and control registers on the PDA500. This mode should be considered as the "home-base" mode.

DOS

The parameter <brd> selects whether this function writes to the mode register (OM) on the PDA500 or returns the value last written to the register. It also controls which PDA500 board is written to in multiple-board systems.

Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, the mode number <val> is written to the mode control register on the corresponding PDA500 board. When <brd> takes on the value 1 through 8, the value in the mode control register is not changed, rather, the last value written to this register on the corresponding PDA500 board is returned. For <brd> values of 1 through 8 the value passed in <val> has no effect. **CAUTION**: If the Auxiliary Bus Control (see section on *SABconf500*) is enabled on the PDA500 board, this function will return the last mode set by the PC software which may not be the same mode set by the Auxiliary Bus.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd>, the value passed in <val> is written to the mode control register (OM) and returned. If 1 through 8 is passed in <brd>, the value returned is the last value written to the corresponding PDA500 mode control register. A value of -1 indicates an error.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.
<rw> is the READ/WRITE flag. If READ is passed, the last value written to the mode control register is returned. If WRITE is passed then <val> is written to the mode control register (OM) and returned.

CAUTION: If the Auxiliary Bus Control (see section on *SABconf500*) is enabled on the PDA500 board, this function will return the last mode set by the PC software which may not be the same mode set by the Auxiliary Bus.

Return Value

If <rw> is WRITE then <val> is returned. If <rw> is READ, the value of the current mode control register is read from the software registers and returned. If -1 is returned then an invalid <brd> was passed.

3.5.39 nulloff500

Form

DOS	Short nulloff500(unsigned short brd, unsigned short val);
Windows	SHORT nulloff500(P500ID * brd, USHORT val);

Description

The function *nulloff500* adjusts the offset voltage to the Analog to Digital Converter to result in a value of <val> with a zero-volt input. This is accomplished by adjusting the value of the offset register (OF#) until the offset voltage of the system reaches the desired value <val>.

DOS

The parameter <brd> specifies the PDA500 board number. Valid values are Brd1P500 through Brd8P500.

The parameter <val> has the range of 0 to 255. The following table shows suggested settings for <val>:

Signal Coupling	Signal	<val>
DC	Positive Going	0 (min)
DC	Negative Going	255 (max)
DC	Symmetrical about center	128 (1/2 scale)
AC	Any	128

For this function to work properly the input signal must either be disconnected or a true zero-signal must be input. The latter method is preferred, as this will also null-out any offset voltage in the signal source.

Return Value

Returns the value of the resulting offset. If <brd> is invalid, -1 is returned. If the return value is -2, then the function was unable to null the offset.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <val> has the range of 0 to 255. The following table shows suggested settings for <val>:

Signal Coupling	Signal	<val>
DC	Positive Going	0 (min)
DC	Negative Going	255 (max)
DC	Symmetrical about center	128 (1/2 scale)
AC	Any	128

For this function to work properly the input signal must either be disconnected or a true zero-signal must be input. The latter method is preferred as this will also null out any offset voltage in the signal source.

Return Value

Returns the value of the resulting offset. If <brd> is invalid, -1 is returned. If the return value is -2, then the function was unable to null the offset.

3.5.40 offset500

Form

DOS	short offset500(unsigned short brd, unsigned char val);
Windows	SHORT offset500(P500ID * brd, UCHAR val, BYTE rw);

Description

The function *offset500* writes an 8-bit value to the offset DAC of the specified channel on the PDA500 board. This produces a DC voltage that is summed with the input signal after the scaling amplifiers, before the ADC. This voltage is normally used to cancel any dc offsets that exist in the input amplifier or the external signal source or to shift the input signal voltage range.

DOS

The parameter <brd> selects whether this function writes to the offset register (OF) or returns the value last written to the register. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <val> is written to the offset register on the corresponding PDA500 board. When <brd> takes on the value 1 through 8 the value in the offset register is not changed, rather, the last value written to the corresponding PDA500 board offset register is returned.

The parameter <val> can take on values from 0 to 255. A value of 128 produces little or no offset shift. A value of 0 produces a shift to approximately the negative full scale range of the ADC. A value of 255 produces a shift of approximately the positive full scale voltage of the ADC.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then <val> is returned. If 1 through 8 is passed in <brd>, the last value, that was written to the corresponding PDA500 board offset register, is returned. If <brd> is invalid, -1 is returned.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <val> can take on values from 0 to 255. A value of 128 produces little or no offset shift. A value of 0 produces a shift to approximately the negative full-scale range of the ADC. A value of 255 produces a shift of approximately the positive full-scale voltage of the ADC.

<rw> is the READ/WRITE flag. If READ is passed the last value written to the offset register is returned. If WRITE is passed then <val> is written to the offset register (OF) and returned.

Return Value

If <rw> is WRITE then <val> is returned. If <rw> is READ the value of the current offset register is read from the software registers and returned. If -1 is returned then an invalid <brd> was passed.

3.5.41 opendevice500

Form

DOS	N/A
Windows	SHORT opendevice500(P500SWR * SWR, P500ID * ID);

Description

In order to access a PDA500 board in Windows *opendevice500* must be called. This function acquires a handle to the device based on the information contained in the SWR parameter.

The parameter <SWR> is a pointer to a structure of type P500SWR. The variable passed into this parameter must be setup using the function *setupregisters500* prior to calling opendevice500.

The parameter <ID> is a pointer to the board ID structure. This structure holds a handle to the device, which is initialized after a call to this function. The structure also contains a pointer to the SWR structure containing the software registers and information about the board.

Return Value

On success a 0 is returned. If the function could not obtain a handle to the PDA500 then a -1 is returned.

Example

This example shows how to setup all the parameters needed to run a PDA500.

```
P500SWR Brd1SWR;                    //declare software register structure variable
P500ID Brd1, *PBrd1;                //declare id structure and pointer

setupregisters500(&Brd1SWR,1);      //get info on first PDA500 found
opendevice500(&Brd1SWR,&Brd1);      //get handle
PBrd1=&Brd1;                        //set pointer to struct

//rest of program goes here
mode500(PBrd1, StbyMode, WRITE);    //set mode  (shows how to pass PBrd1 in)

//close device when finished
closedevice500(PBrd1);              //close handle
```

Related Functions

setupregisters500, *closedevice500*

3.5.42 presamples500

Form

DOS	short presamples500(unsigned short brd, unsigned short val);
Windows	SHORT presamples500(P500ID * brd, USHORT val, BYTE rw);

Description

The function *presamples500* enables pre-trigger mode if val is greater than 0 and allows the user to select the number of pre-trigger samples to be acquired in the pre-trigger mode. This is the number of data samples to be stored in signal RAM before the trigger condition is satisfied.

DOS

Parameter <brd> selects whether this function writes to the pre-trigger samples register (TPD) or returns the value last written to the register. It also controls which PDA500 board is written to, in multiple board systems. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500-Brd8P500 <val> is written to the corresponding PDA500 board. When <brd> takes on the values 1-8, the value is not changed, rather, the last value written to the corresponding PDA500 is returned.

The parameter <val> can take the value of 0 - 4095. The number of pre-trigger samples per channel is equal to 8 times <val>.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then <val> is returned. If 1 through 8 is passed in <brd>, the last value written to the corresponding PDA500 board is returned. If an invalid <brd> is passed -1 is returned.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <val> can take the value of 0 - 4095. The number of pre-trigger samples per channel is equal to 8 times <val>.

<rw> is the READ/WRITE flag. If READ is passed the last value written to the board is returned. If WRITE is passed then <val> is written to the board and returned.

Return Value

If <rw> is WRITE then <val> is returned. If <rw> is READ, the value of the current number of pre-samples is read from the software registers and returned. If -1 is returned then an invalid <brd> was passed.

3.5.43 pwdownor500

Form

DOS	short pwdownor500(unsigned short brd, unsigned short val);
Windows	SHORT pwdownor500(P500ID * brd, USHORT val, BYTE rw);

Description

The function *pwdownor500* allows the user to override the PDA500 board's power down feature by setting the hardware register PDO. The PDA500 turns off the power to the data acquisition circuitry when in any mode except one of the three acquisition modes. Overriding this feature means that the PDA500 board acquisition circuits will be powered at all times unless the board is in the Off Mode.

DOS

The parameter <brd> selects whether this function writes to the PDO register or returns the value last written to the register. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <brd> is written to the corresponding PDA500 board. When <brd> takes on the value 1 through 8 the value in the register is not changed, instead, the last value written to the corresponding PDA500 board is returned.

The parameter <val> can take on the value of 0 or 1 to turn power down override off or on. Additionally, the global variables OFF and ON can also be passed for <val>. Note that this parameter is setting the override of the PDA500 board's ability to automatically go to power down mode. That is, when <val> is off, the PDA500 can automatically enter power down mode.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then <val> is returned. If 1 through 8 is passed in <brd>, the last value written to the corresponding PDA500 board is returned.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <val> can take on the value of 0 or 1 to turn power down override off or on. Additionally, the global variables OFF and ON can also be passed for <val>. Note that this parameter is setting the override of the PDA500 board's ability to automatically go to power down mode. That is, when <val> is off, the PDA500 can automatically enter power down mode.

<rw> is the READ/WRITE flag. If READ is passed, the last value written to the board is returned. If WRITE is passed then <val> is written to the board and returned.

Return Value

If <rw> is WRITE then <val> is returned. If <rw> is READ, the current value of the PDO register is read from the software registers and returned. If -1 is returned then an invalid <brd> was passed.

3.5.44 pwrdy500

Form

DOS	short pwrdy500(unsigned short brd);
Windows	SHORT pwrdy500(P500ID * brd, BYTE rw);

Description

The function *pwrdy500* checks the status of the PDA500 PR flag and returns the flag's status. The PR flag indicates that the on-board power monitoring circuits have detected stable power to the acquisition circuits. This indicates that a data acquisition can begin on the next trigger event after the flag occurs.

DOS

The parameter <brd> selects which PDA500A board will be operated. Parameter <Brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then the value of the PR is read from the PDA500 board and returned. If 1 through 8 is passed, the value of the PRACF last read is returned. The meaning of the value returned is as follows:

0 = Acquisition circuit power not ready.
1 = Acquisition circuit power ready.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

<rw> is the READ/WRITE flag. If READ is passed, the last value written to the PR flag of the software register is read and returned. If WRITE is passed then value is read from the board, written to the software register and returned.

Return Value

If <rw> is WRITE then the value of the hardware register on the board is returned. If <rw> is READ, the value last read from the board and stored in the software register is returned. If -1 is returned then an invalid <brd> was passed.

3.5.45 readEPROM500

Form

DOS	short readEPROM500(unsigned short brd, unsigned short parm);
Windows	SHORT readEPROM500(P500ID * brd, ULONG parm);

Description
The function *readeprom500* reads a value from the configuration EEPROM on the PDA500 board specified by <brd>. The main purpose of the configuration EEPROM is to load the PDA500 PCI controller with default values. The configuration EEPROM also stores other useful information such as the board Serial Number and programmable logic version number.

DOS

The parameter <brd> selects which PDA500 board is read. Parameter <brd> can take on the values 1 through 8.

The parameter <parm> specifies the EEPROM address that will be read. The table below shows the assignment of the EEPROM address locations.

Address (Hex)	Parameter
0-7F	PCI Controller Default Values
80-82	Serial Number
83	ISP Logic Version Number
84-FFF	Reserved
1000-1FFF	User Defined

Return Value

Returns the 8-bit value read from the PDA500 EEPROM. If the board can not be located or if a PCI read error occurs a -1 is returned.

Related Functions

findbrd500, *writeEPROM500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <parm> specifies the EEPROM address that will be read.

Return Value

Returns the 8-bit value read from the PDA500 EEPROM. If <brd> is invalid then -1 is returned.

Related Functions

findbrd500, *writeEPROM500*

3.5.46 readreg500

Form

DOS	unsigned long readreg500(unsigned short brd, unsigned short regnum);
Windows	ULONG readreg500(P500ID * brd, USHORT regnum);

Description

The function *readreg500* is used to directly read the contents of the PDA500 hardware "read" register. See appendix B for the bit patterns of the hardware register.

DOS

The parameter <brd> selects which PDA500 board is read. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

The parameter <regnum> selects which register to read and can take on the values 0 through 7.

Return Value

The 32- bit value of PDA500 register 2.

Related Functions

writereg500

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <regnum> selects which register to read and can take on the values 0 through 7.

Return Value

On Error 0 is returned. Otherwise, the 32-bit value of PDA500 register 2.

Related Functions

writereg500

3.5.47 readset500

Form

DOS	short readset500(unsigned short brd, char *filename);
Windows	SHORT readset500(P500ID * brd, CHAR * filename);

Description

The function *readset500* reads the PDA500 board's setup information stored in <filename> and writes the setup to the hardware registers of the PDA500 board specified in <brd>.

The file header is defined as follows:

Byte #	Contents
0-3	Board Type 500=PDA500
4-7	Register 0
8-11	Register 1
12-15	Register 2
16-19	Register 3
20-23	Register 4
24-27	Register 5
28-31	Register 6
32-35	Register 7
36-255	Reserved for User Comments

Comments are character strings up to a maximum of 219 characters, which allow reference information to be attached to a data file.

DOS

Parameter <brd> can take on the values Brd1P500 through Brd8P500.

The data is stored in <filename> with the function *writeset500* and contains 256 unsigned characters.

Return Value

The value returned is -1 if the file cannot be opened, -2 if there are any illegal values. A value of 1 is returned if successful.

Related Functions

writeset500

Example

```
/*      The following example reads the board setup parameters from the file "setup" and writes them to
        board #2
*/

        readset500(2,"c:\setup");
```

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The data is stored in <filename> with the function *writeset500* and contains 256 unsigned characters.

Return Value

The value returned is -1 if the file cannot be opened, -2 if there are any illegal values or a -3 if an improper <brd> value was passed. A value of 1 is returned if successful.

Related Functions

writeset500

3.5.48 realfft500

Form

DOS	COMPLEX *realfft500(unsigned short brd, unsigned short fftwindow, unsigned short fftsize, long startsample, short dealloc);
Windows	COMPLEX * realfft500(P500ID * brd, USHORT fftwindow, USHORT fftsize, LONG startsample, SHORT dealloc, FLOAT *idata);

Description

The function *realfft500* performs a real FFT on data in the PDA500 RAM. The output data is an array of complex values representing the real and imaginary parts of each spectral component. The input data is formatted via an FFT window.

This function allocates memory for various arrays used in the calculations. During execution the peak amount of memory required, in bytes, is equal to 14 times the fft size. At completion, some of the memory is de-allocated. The amount of memory de-allocated depends on the setting of the parameter <dealloc>.

DOS

The parameter <brd> selects which PDA500 board is written to. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

Parameter <fftwindow> selects one of four different fft windows as follows:

1: Rectangular 2: Hanning 3: Hamming 4: Blackman-Harris.

Parameter <fftsize> specifies the size of the FFT and may be any size from 64 points to 8192 points in factors of 2.

Parameter <startsample> is the first sample to be used, measured from the start of the PDA500 RAM.

If parameter <dealloc> is set to 0, minimal de-allocation of the memory used by the function occurs. At completion the amount of memory still allocated is 10 times <fftsize> in bytes. This allows the function to execute faster when future calls are made due to the fact that internal coefficient arrays do not need to be recalculated as long as the fft size is not changed.

Setting <dealloc> to 1 will de-allocate all of the memory used by the function with the exception of that needed for the output data array, which is 4 times <fftsize> in bytes. For example, an 8k fft requires 112 kilobytes of free memory during execution, but at completion frees all memory except the 32 kilobytes required for the output data array consisting of 4k complex values. To de-allocate all memory used by this function it is necessary to call the function with parameter <brd> set equal to 0.

Return Value

On success returns a pointer to the output data array. Returns NULL if insufficient memory is available or brd is invalid.

Related Function

realfftmag500

Example

See *fftstats500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

Parameter <fftwindow> selects one of four different fft windows as follows:

1: Rectangular 2: Hanning 3: Hamming 4: Blackman-Harris.

Parameter <fftsize> specifies the size of the FFT and may be any size from 64 points to 8192 points in factors of 2.

Parameter <startsample> is the first sample to be used measured from the start of the PDA500 RAM.

If parameter <dealloc> is set to 0, minimal de-allocation of the memory used by the function occurs. At completion the amount of memory still allocated is 10 times <fftsize> in bytes. This allows the function to execute faster when future calls are made due to the fact that internal coefficient arrays do not need to be recalculated as long as the fft size is not changed.

Setting <dealloc> to 1 will de-allocate all of the memory used by the function with the exception of that needed for the output data array, which is 4 times <fftsize> in bytes. For example, an 8k fft requires 112 kilobytes of free memory during execution but at completion frees all memory except the 32 kilobytes required for the output data array consisting of 4k complex values. To de-allocate all memory used by this function it is necessary to call the function with parameter <brd> set equal to 0.

Parameter <idata> is a pointer to buffer of floats, which represent data the user is supplying to the function to be used with the fft calculations. **NOTE:** This parameter is optional. If no data is to be supplied by the user, this parameter must be set to NULL. If this parameter is to be used, idata must represent a buffer, which is the same size as the fftsize.

Return Value

On success returns a pointer to the output data array. Returns NULL if insufficient memory is available or brd is invalid.

Related Function

realfftmag500

Example

See *fftstats500*

3.5.49 realfftmag500

Form

DOS	float *realfftmag500(unsigned short brd, unsigned short fftwindow, unsigned short fftsize, long startsample, short dealloc);
Windows	FLOAT * realfftmag500(P500ID * brd, USHORT fftwindow, USHORT fftsize, LONG startsample, SHORT dealloc, FLOAT *idata);

Description

The function *realfftmag500* performs a real FFT on data in the PDA500 RAM. The output data is an array of magnitude (voltage) squared values, which may be easily converted to dB readings. The input data is formatted via an FFT window.

This function allocates memory for various arrays used in the calculations. During execution the peak amount of memory required, in bytes, is equal to 14 times the fft size. At completion some of the memory is de-allocated. The amount of memory de-allocated depends on the setting of the parameter <dealloc>.

DOS

The parameter <brd> selects which PDA500 board is written to. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

Parameter <fftwindow> selects one of four different fft windows as follows:

1: Rectangular 2: Hanning 3: Hamming 4: Blackman-Harris.

Parameter <fftsize> specifies the size of the FFT and may be any size from 64 points to 8192 points in factors of 2.

Parameter <startsample> is the first sample to be used, measured from the start of the PDA500 RAM.

If parameter <dealloc> is set to 0, minimal de-allocation of the memory used by the function occurs. At completion the amount of memory still allocated is 10 times <fftsize> in bytes. This allows the function to execute faster when future calls are made due to the fact that internal coefficient arrays do not need to be recalculated as long as the fft size is not changed.

Setting <dealloc> to 1 will de-allocate all of the memory used by the function with the exception of that needed for the output data array plus an additional equal amount. In this case the total amount of memory still allocated is equal to 4 times <fftsize> in bytes. For example, an 8k fft requires 112 kilobytes of free memory during execution but at completion frees all memory except the 16 kilobytes required for the output data array consisting of 4k floating point values, plus an additional 16 kilobytes which can not be de-allocated. To de-allocate all memory used by this function it is necessary to call the function with parameter <brd> set equal to 0.

Return Value

On success returns a pointer to the output data array. Returns NULL if insufficient memory is available or brd is invalid.

Related Functions

realfft500, *fftstats500*

Example

See *fftstats500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

Parameter <fftwindow> selects one of four different fft windows as follows:

1: Rectangular 2: Hanning 3: Hamming 4: Blackman-Harris.

Parameter <fftsize> specifies the size of the FFT and may be any size from 64 points to 8192 points in factors of 2.

Parameter <startsample> is the first sample to be used, measured from the start of the PDA500 RAM.

If parameter <dealloc> is set to 0, minimal de-allocation of the memory used by the function occurs. At completion the amount of memory still allocated is 10 times <fftsize> in bytes. This allows the function to execute faster when future calls are made due to the fact that internal coefficient arrays do not need to be recalculated as long as the fft size is not changed.

Setting <dealloc> to 1 will de-allocate all of the memory used by the function with the exception of that needed for the output data array plus an additional equal amount. In this case the total amount of memory still allocated is equal to 4 times <fftsize> in bytes. For example, an 8k fft requires 112 kilobytes of free memory during execution but at completion frees all memory except the 16 kilobytes required for the output data array consisting of 4k floating point values, plus an additional 16 kilobytes which can not be de-allocated. To de-allocate all memory used by this function it is necessary to call the function with parameter <brd> set equal to 0.

Parameter <idata> is a pointer to buffer of floats, which represent data the user is supplying to the function to be used with the fft calculations. NOTE: This parameter is optional. If no data is to be supplied by the user, this parameter must be set to NULL. If this parameter is to be used, idata must represent a buffer, which is the same size as the fftsize.

Return Value

On success returns a pointer to the output data array. Returns NULL if insufficient memory is available or brd is invalid.

Related Functions

realfft500, *fftstats500*

Example

See *fftstats500*

3.5.50 SABbrdnum500

Form

DOS	short SABbrdnum500(unsigned short brd, unsigned char brdnum)
Windows 95/NT	SHORT SABbrdnum500(P500ID * brd, UCHAR brdnum, BYTE rw)

Description

The function *SABbrdnum500* allows the user to select the value of the Auxiliary Interrupt. This defines which SAB interrupt will be used by the PDA500 board for control and signaling on the SAB.

SABbrdnum500 enables the PDA500 data acquisition and transfer to be controlled by another device, such as Signatec's PMP8 Digital Signal Processor board via the SAB. The SAB can control the PDA500 with the use of the COMM line and the INTR line corresponding to the assigned SAB Number. The PDA500 will signal the SAB device that an acquisition is complete by issuing and interrupt on the INTR line corresponding to the board number. This function must be called with a board number other than 0 before all other SAB functions are called.

The *SABinit500* function can be used instead of this function which allows the user to set all SAB parameters with one function call.

DOS

The parameter <brd> selects whether this function writes to the AIE register or returns the value last written to the register. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <val> is written to the AIE register on the corresponding PDA500 board. When <brd> takes on the value 1 through 8, the value in the AIE register is not changed, instead, the last value written to the corresponding PDA500 board register is returned.

The parameter <val> can take on values from 0 to 5. A value of 0 will disable all SAB control. A non-zero value determines which SAB INTR line will cause a mode change on the PDA500.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then <val> is returned. If 1 through 8 is passed in <brd>, the last value written to the corresponding PDA500 board AIE register is returned. If an error occurs a value of -1 is returned.

Related Functions

SABinit500

Windows 95/NT

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <val> can take on values from 0 to 5. A value of 0 will disable all SAB control. A non-zero value determines which SAB INTR line will cause a mode change on the PDA500.

<rw> is the READ/WRITE flag. If READ is passed then the last value written to the board is returned. If WRITE is passed then <val> is written to the board and returned.

Return Value

If <rw> is WRITE then <val> is returned. If <rw> is READ, the last value written is read from the software registers and returned. If <brd> is invalid then -1 is returned.

Related Functions

SABinit500

3.5.51 SABclk500

Form

DOS	short SABclk500(unsigned short brd, unsigned char val)
Windows 95/NT	SHORT SABclk500(P500ID *brd, UCHAR src, BYTE rw)

Description

The function *SABclk500* is used to set the source clock frequency data transfers across the Auxiliary Bus when in SAB Transfer Mode (SABMode) or SAB Acquisition Mode (Acq2Mode). The parameter <src> can take on the values 0 through 3 yielding a source clock shown below.

<src>	Source Clock
0	25 MHz
1	62.5 MHz
2	66.66MHz
3	External (Acq. Clock)

For maximum SAB Version 3 transfer rates the clock source of 66.6 MHz should be selected. For SAB compatible transfers, the maximum clock rate is 25 MHz.

The *SABinit500* function can be used instead of this function which allows the user to set all SAB parameters with one function call.

DOS

Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <src> is written to the appropriate PDA500 board. When <brd> takes on the value 1 through 8, the board value is not changed, rather, the last value written to the board is returned.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd>, then <src> is returned. If 1 through 8 is passed in <brd>, the last value written to the corresponding PDA500 is returned. If an error occurs a value of -1 is returned.

Related Functions

SABinit500

Windows 95/NT

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

<rw> is the READ/WRITE flag. If READ is passed then the last value written to the board is returned. If WRITE is passed then <src> is written to the board and <src> is returned.

Return Value

If <rw> is WRITE then <src> is returned. If <rw> is READ, the last value written to the board is read from the software registers and returned. If <brd> is invalid then -1 is returned.

Related Functions

SABinit500

3.5.52 SAB1compatible500

Form

DOS	short SAB1compatible500(unsigned short brd, unsigned char val)
Windows 95/NT	SHORT SAB1compatible500(P500ID * brd, UCHAR val)

Description

The function *SAB1compatible* is used to configure the board to be compatible with the SAB1 specification. The parameter <val> can take on the values 0 or 1. Respectively, these values result in switching the board between SAB1 and SAB3 compatibility. This function should be called after the *SABbrdnum500* function and before any other SAB functions.

This function is considered obsolete and is only included here for backward compatibility. Use the *SABinit500* function instead to set all SAB parameters.

DOS

Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <val> is written to the appropriate PDA500 board. When <brd> takes on the value 1 through 8, the board value is not changed, rather, the last value written to the board is returned.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd>, then <val> is returned. If 1 through 8 is passed in <brd>, the last value written to the corresponding PDA500 is returned. If an error occurs, a value of -1 is returned.

Windows 95/NT

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

Return Value

If the parameter <brd> is invalid, a –1 will be returned. Otherwise, the parameter <val> is returned.

3.5.53 SABconf500

Form

DOS	short SABconf500(unsigned short brd,unsigned char val)
Windows 95/NT	SHORT SABconf500(P500ID * brd, UCHAR val, BYTE rw)

Description

The function *SABconf500* is used to set the data transfer mode across the Auxiliary Bus. The parameter <val> can take on the values 0 or 1. Respectively, these values result in 64 and 32 bit data transfers. Global variables Bits64 and Bits32 can also be passed in <val>.

The *SABinit500* function can be used instead of this function which allows the user to set all SAB parameters with one function call.

DOS

Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <val> is written to the appropriate PDA500 board. When <brd> takes on the value 1 through 8, the board value is not changed, rather, the last value written to the board is returned.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd>, then <val> is returned. If 1 through 8 is passed in <brd>, the last value written to the corresponding PDA500 is returned. If an error occurs a value of -1 is returned.

Related Functions

SABinit500

Windows 95/NT

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

<rw> is the READ/WRITE flag. If READ is passed then the last value written to the board is returned. If WRITE is passed then <val> is written to the board and returned.

Return Value

If <rw> is WRITE then <val> is returned. If <rw> is READ the last value written to the board is read from the software registers and returned. If <brd> is invalid then -1 is returned.

Related Functions

SABinit500

3.5.54 SABinit500

Form

DOS	short SABinit500(unsigned short brd, unsigned char brdnum, unsigned char clock, unsigned char width, unsigned char control, unsigned char membrdnum);
Windows	SHORT SABinit500(P500ID * brd, UCHAR brdnum, UCHAR clock, UCHAR width, UCHAR control, UCHAR membrdnum);

Description

The function *SABinit500* is used to set all the parameters for data transfer modes across the Auxiliary Bus at one time. After calling this function, the *mode500* function can be called at any time using the SABmode or Acq2Mode parameters to actually start the SAB transfer. This function replaces the functions *SABbrdnum500*, *SABclk500*, *SAB1compatible500*, and *SABconf500* and should be used instead of these functions.

DOS

Parameter <brd> can take on the values Brd1P500 through Brd8P500.

<brdnum> enables the PDA500 data acquisition and transfer to be controlled by another device, such as Signatec's PMP8 Digital Signal Processor board via the SAB. The SAB can control the PDA500 with the use of the COMM line and the INTR line corresponding to the assigned SAB Number. The PDA500 will signal the SAB device that an acquisition is complete by issuing and interrupt on the INTR line corresponding to the board number. This function must be called with a board number other than 0 before starting any transfer to the SAB bus or doing any SAB control.

<clock> is used to set the source clock frequency data transfers across the Auxiliary Bus when in SAB Transfer Mode (SABMode) or SAB Acquisition Mode (Acq2Mode). This parameter can take on the values 0 through 3 yielding a source clock shown below.

<src>	Source Clock
0	25 MHz
1	62.5 MHz
2	66.66MHz
3	External (Acq. Clock)

For maximum SAB Version 3 transfer rates the clock source of 66.6 MHz should be selected. For SAB compatible transfers, the maximum clock rate is 25 MHz.

<width> can take on the values 0 or 1. Respectively, these values result in 64 and 32 bit data transfers. Global variables Bits64 and Bits32 can also be passed in <width>.

<control> if 0 is passed then no other device can control the PDA500 over the SAB bus. If 1 is passed then another device (such as a PMP8) can control the PDA500 over the SAB.

<membrdnum> should normally be set to 0 unless the PDA500 is being used with MEM500 boards. If so then <membrdnum> should be set to the MEM500 1-4 which the data should be sent to.

Return Value

If all parameters passed are valid than a 1 is returned. Otherwise a 0 is returned and the registers are not written.

Related Functions

mode500, acqSABdata500

Example

See *acqSABdata500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

<brdnum> enables the PDA500 data acquisition and transfer to be controlled by another device, such as Signatec's PMP8 Digital Signal Processor board via the SAB. The SAB can control the PDA500 with the use of the COMM line and the INTR line corresponding to the assigned SAB Number. The PDA500 will signal the SAB device that an acquisition is complete by issuing and interrupt on the INTR line corresponding to the board number. This function must be called with a board number other than 0 before starting any transfer to the SAB bus or doing any SAB control.

<clock> is used to set the source clock frequency data transfers across the Auxiliary Bus when in SAB Transfer Mode (SABMode) or SAB Acquisition Mode (Acq2Mode). This parameter can take on the values 0 through 3 yielding a source clock shown below.

<src>	Source Clock
0	25 MHz
1	62.5 MHz
2	66.66MHz
3	External (Acq. Clock)

For maximum SAB Version 3 transfer rates the clock source of 66.6 MHz should be selected. For SAB compatible transfers, the maximum clock rate is 25 MHz.

<width> can take on the values 0 or 1. Respectively, these values result in 64 and 32 bit data transfers. Global variables Bits64 and Bits32 can also be passed in <width>.

<control> if 0 is passed then no other device can control the PDA500 over the SAB bus. If 1 is passed then another device (such as a PMP8) can control the PDA500 over the SAB.

<membrdnum> should normally be set to 0 unless the PDA500 is being used with MEM500 boards. If so then <membrdnum> should be set to the MEM500 1-4 which the data should be sent to.

Return Value

If all parameters passed are valid than a 1 is returned. Otherwise a 0 is returned and the registers are not written.

Related Functions

mode500, acqSABdata500

Example

See *acqSABdata500*

3.5.55 segmentdetect500

Form

DOS	short segmentdetect500(unsigned short brd);
Windows	SHORT segmentdetect500(P500ID * brd, BYTE rw);

Description

The function *segmentdetect500* checks the status of the PDA500's segment complete flag when the PDA500 is acquiring data in the segmented acquisition mode. The flag is set after the 1st segment of data has been acquired. It will remain set until all remaining data segments have been acquired.

DOS

The parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500 or 1 through 8, the value of the (SD) segment-detect flags: a read only register is returned.

Return Value

If Brd1P500 through Brd8P500 or 1 through 8 is passed in <brd>, value of the segment-detect flag (SD register) is returned. A value of 0 indicates that the 1st acquisition segment has not completed. A value of 1 is returned if the 1st acquisition segment has completed.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

<rw> is the READ/WRITE flag. If READ is passed then the last value read from the board's segment detect flag will be returned from the software register. If WRITE is passed then the segment detect flag (SD) will be read from the board and its value returned.

Return Value

If <rw> is WRITE then the value from the board is returned. If <rw> is READ the last value read from the board is returned from the software register. If <brd> is invalid then -1 is returned.

3.5.56 segmentsize500

Form

DOS	short segmentsize500(unsigned short brd, unsigned char val);
Windows	SHORT segmentsize500(P500ID * brd, UCHAR val, BYTE rw);

Description

The function *segmentsize500* selects the amount of signal memory that will be assigned to each data segment when the board is in Segmented Mode. When the PDA500 board is set to the data acquisition mode and a trigger occurs, the amount of data specified by <val> will be acquired. Acquisition will then stop until another trigger is

received, at which point another segment of <val> size will be acquired. This series of events will continue until the PDA500 reaches the end address count.

DOS

The parameter <brd> selects whether this function writes to the segment size register (SS) or returns the value last written to the register. It also controls which PDA500 board is written to in multiple board systems. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8.

When <brd> takes on the value Brd1P500 through Brd8P500, <val> is written to the corresponding PDA500 board. When <brd> takes on the value 1 through 8, the value in the active memory register is not changed but the last value written to the corresponding PDA500 board's register is returned.

The <val> parameter is an unsigned character, which can take on the values 0-11. The following table shows the size of the data segments for each value.

<val>	Segment Size
0	32
1	64
2	128
3	256
4	512
5	1024
6	2048
7	4096
8	8192
9	16384
10	32768
11	65536

To be meaningful, the segment size should always be set smaller than the active memory size.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd>, the segment size <val> is returned. If 1 through 8 is passed in <brd>, the last value written to the segment size register (SS) of the corresponding PDA500 board is returned. If an invalid value for <brd> is passed, the function will return -1.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <val> can take on the values 0-11. The following table shows the size of the data segments for each value.

val>	Segment Size
0	32
1	64
2	128
3	256
4	512
5	1024
6	2048
7	4096
8	8192
9	16384
10	32768
11	65536

To be meaningful, the segment size should always be set smaller than the active memory size.

<rw> is the READ/WRITE flag. If READ is passed then the last value written to the segment size (SS) register is returned. If WRITE is passed then <val> is written to the segment size register and returned.

Return Value

If <rw> is WRITE then <val> is returned. If <rw> is READ, the last value written to the board is read from the software registers and returned. If <brd> is invalid then -1 is returned.

3.5.57 sendEOI

Form

DOS	void sendEOI(short irqnum);
Windows	N/A

Description

The function *sendEOI* is used to signal the PC that the interrupt handler is complete and control can be returned to the task that was interrupted. This function MUST be called at the end of the user created interrupt handler routine. If this line is omitted, the PC will lock-up when a hardware interrupt occurs.

The parameter <irqnum> is the hardware interrupt number.

Return Value

None

Related Functions

dmaint500, *setuppciint500*

3.5.58 setendadd500

Form

DOS	long setendadd500(unsigned brd short , long endadd);
Windows	LONG setendadd500(P500ID * brd, LONG endadd, BYTE rw);

Description

This function sets the ending transfer address for data transfer to or from the PDA500. In the acquisition mode, *setendadd500* sets the address at which data transfer to RAM will stop: determining the end of data acquisition. When using the SAB transfer mode, *setendadd500* sets the address counter value at which data transfer to the SAB will stop. The function *setendadd500* along with *setstartadd500* determine how much data is acquired or transferred.

DOS

The parameter <brd> selects whether this function writes to the ending address register or returns the present value in the register. It also controls which PDA500 board is written to in multiple board systems. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <endadd> is written to the ending address register on the associated PDA500 board. When <brd> takes on the value 1 through 8, the hardware register is not changed. Instead, the last ending address value written to the corresponding PDA500 software register is returned.

The parameter <endadd> defines the end of a transfer and must be set on a 64-byte boundary. It can take a value of 0 through 1048576 on a PDA500-1Meg board and a value of 0 through 4194240 on a PDA500-4Meg board. The value passed is equal to the number of samples acquired. If the parameter <endadd> is greater than 4194304 the acquisition/transfer will not stop until the board is removed from the acquisition or transfer mode.

Return Value

If 1 through 8 is passed in <brd> then the last ending address value written to the board is returned. If Brd1P500 through Brd8P500 is passed in <brd>, the ending transfer address is written to the corresponding PDA500 board's hardware register and returned.

Related Functions

setstartadd500, *acqcomp500*

Examples

```
setstartadd500(Brd1P500, 0);
setendadd500(Brd1P500, 16384);          /* next transfer will be 16k */
```

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <endadd> defines the end of a transfer and must be set on 64-byte boundaries and can take a value of 0 through 1048576 on a PDA500-1Meg board and a value of 0 through 4194240 on a PDA500-4Meg board. The value passed is equal to the number of samples acquired. If the parameter <endadd> is greater than 4194304, the acquisition/transfer will not stop until the board is removed from the acquisition or transfer mode.
<rw> is the READ/WRITE flag. If READ is passed then the current ending transfer address in the software registers is returned. If WRITE is passed then <endadd> is written to the board and returned.

Return Value

If <rw> is WRITE then <endaddress> is returned. If <rw> is READ, the last value written to the board is read from the software registers and returned. If <brd> is invalid then -1 is returned.

Related Functions

setstartadd500, *acqcomp500*

3.5.59 setstartadd500

Form

DOS	long setstartadd500(unsigned short brd, unsigned long count);
Windows	LONG setstartadd500(P500ID * brd, ULONG count, BYTE rw);

Description

The function *setstartadd500* sets the address counter to the value specified. All of the counter bits are set by this function. This determines the address at which a data acquisition to the PDA500, or a SAB transfer from the PDA500 will begin. The starting address used by most applications is zero. This means that data acquisitions and transfers will start at the beginning of the data RAM. If a value of zero is to be used, the function *setstartadd500* need be called only once. After the initial transfer, the PDA500 address counter will automatically reset to zero.

DOS

The parameter <brd> selects whether the starting address PDA500 board will be written to, or if the current starting address will be read from the board. The parameter <brd> also determines which PDA500 board will be operated on in a multiple-board system. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. If the value is Brd1P500 through Brd8P500, the value in <count> will be written to the PDA500 board. If the value is 1 through 8, the address counter on the PDA500 board will be read and the value returned.

Parameter <count> is the address to be set. Acceptable values are 0 to 1048568 for a PDA500-1Meg and 0 to 4194296 for a PDA500-4Meg. The starting address must be a multiple of 8.

Return Value

If <brd> is passed a value 1 through 8, the PDA500 board address counter value is returned. If <brd> is passed a value of Brd1P500 through Brd8P500, the value of <count> is returned. If an invalid value for <brd> is passed, -1 is returned.

Related Functions

autoaddrst500

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

Parameter <count> is the address to be set. Acceptable values are 0 to 1048568 for a PDA500-1Meg and 0 to 4194296 for a PDA500-4Meg. The starting address must be a multiple of 8.

<rw> is the READ/WRITE flag. If READ is passed then the current starting transfer address in the software registers is returned. If WRITE is passed then <count> is written to the board and returned.

Return Value

If <rw> is WRITE then <count> is returned. If <rw> is READ, the last value written to the board is returned.
If <brd> is invalid then -1 is returned.

Related Functions

autoaddrst500

3.5.60 setupbrd500

Form

DOS	void setupbrd500(void);
Windows	N/A (See *setupregisters500*)

Description

The function *setupbrd500* performs assignments of memory register pointers to the "software register" variables, which correspond to the PDA500 hardware control and monitoring registers. It is the software registers that allow software routines to manipulate the bit patterns of the registers and keep track of the write-only register values.

This function MUST be executed as the first line of an applications program before any PDA500 function is executed.

Return Value

none

3.5.61 setuppciint500

Form

DOS	short setuppciint500(unsigned short brd, void interrupt (*intfunct)());
Windows	N/A

Description

The function *setuppciint500* installs an interrupt vector to the PC interrupt vector table. This can be used to cause the PC to respond to a PCI interrupt generated by a PDA500 hardware interrupt.

The parameter <brd> determines the PDA500 board for which the interrupt will be installed. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

The parameter <intfunct> is the name of the user-created interrupt handler: the function that will be executed when a hardware interrupt is generated. The function <intfunct> must be created as "void interrupt intfunct()" and must call the function *sendEOI*(irqnum) as the last line to properly terminate the interrupt and return to the task that was interrupted.

Return Value

A value of 0 is returned if the function properly installs the interrupt. If an error occurs the value -1 will be returned.

Related Functions

dmaint500, *sendEOI*, *uninstallpciint500*

3.5.62 setupregisters500

Form

DOS	N/A (see *setupbrd500* function)
Windows	LONG setupregisters500(P500SWR * SWR, ULONG num);

Description

The function *setupregisters500* performs assignments of memory register pointers to the "software register" variables, which correspond to the PDA500 hardware control and monitoring registers. It is the software registers that allow software routines to manipulate the bit patterns of the registers and keep track of the write-only register values. This function must be executed once for each PDA500 to be controlled in the system.

The parameter <SWR> is a pointer to the software register structure of type P500SWR. The user declares this variable for each PDA500 in the system.

The parameter <num> can be either the number of the board in the system from 1 to 8, or the serial number of the a specific board. If it is the number of the board, this setupregisters500 looks for the corresponding board in the system. For example, if a 1 is entered, the P500SWR structure passed is filled with the information about the first board found in the system. If the serial number is entered, the PDA500 boards in the system are searched until the board with the serial number specified is found. The P500SWR structure passed is then filled with the information about the specified board.

This function MUST be executed first before any other PDA500 function is executed.

Return Value

If a serial number is passed in <num> then a 1 is returned on success. A -1 indicates an error and a -2 indicates the specified serial number was not found.

If a board number is passed in <num> then the serial number of the board found is returned on success. A -1 indicates an error.

Example

This example shows how to setup the register structures of two PDA500 boards in a system.

```
P500SWR PBrd1, PBrd2;                  //declare two variables of type P500SWR
LONG Serialnum1, Serialnum2;
Serialnum1=setupregisters500(&PBrd1,1); //find first board in the system
setupregisters500(&PBrd2,12345);       //find board with serial number 12345
Serialnum2 = PBrd2.Serialnum;          //extract serial number from struct
```

Related Functions

opendevice500

3.5.63 temperature500

Form

DOS	short temperature500(unsigned short brd);
Windows	SHORT temperature500(P500ID * brd, BYTE rw);

Description

The function *temperature500* reads the temperature bit from the PDA500 board's hardware register. This bit is set when the temperature of the ADC on the board exceeds 65 degrees C. If this condition is reached, the PDA500 will automatically turn the acquisition power off to prevent damage to the ADC.

Because the ADC is equipped with a fanned-heatsink, the 65-degree set point should never be reached. Reaching 65 degrees is likely caused by one of two conditions.

1. The fan on the ADC has failed and must be replaced.
2. The board is being operated in an exceptionally warm environment, AND the acquisition power remains on for a long period (minutes), AND the airflow to the fan is restricted. In this case the PC slot in front of the ADC fan should be left open to allow proper airflow to the ADC heatsink.

DOS

The parameter <brd> selects which PDA500 board temperature will be returned. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then the value of the temperature bit is read from the PDA500 board and returned. If 1 through 8 is passed, the value of the temperature bit is read from the software register and returned. The meaning of the value returned is as follows:

0	Temperature less than 65 degrees C
1	Temperature greater than 65 degrees C
-1	Error

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

<rw> is the READ/WRITE flag. If READ is passed, the current value of the temperature bit in the software registers is returned. If WRITE is passed, the current value of the temperature bit is read from the board and returned.

Return Value

If WRITE is passed in <rw> then the value of the temperature bit is read from the PDA500 board and returned. If READ is passed in <rw>, the value of the temperature bit is read from the software registers and returned. If <brd> is invalid then -1 is returned.

The meaning of the value returned is as follows:

0	Temperature less than 65 degrees C
1	Temperature greater than 65 degrees C
-1	Error

3.5.64 trig500

Form

DOS	short trig500(unsigned short brd, unsigned char source, unsigned char slope, unsigned char mode, unsigned char level);
Windows	SHORT trig500(P500ID * brd, UCHAR source, UCHAR slope, UCHAR mode, UCHAR level);

Description

The function *trig500* allows the user to set all the PDA500 board trigger parameters with a single function call. The trigger parameters, which are set with *trig500,* are Source, Slope and Mode.

DOS

The parameter <brd> selects which PDA500 board the trigger parameters will be written to. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

The parameter values that can be passed are as follows:

Parameter	Value	Function
Source	0 1	Sets trigger source to Internal Sets trigger source to External
Slope	0, Pos 1, Neg	Sets trigger slope to Positive Sets trigger slope to Negative
Mode	0, Single 1, Segment	Sets trigger mode to Single Shot Sets trigger mode to Segmented
Level	0-255	

Return Value

This function returns a 0 on success, -1 if <brd> is invalid and -2 if any other parameter is out of range.

Related Functions

trigmode500, *trigslope500*, *trigsource500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter values that can be passed are as follows:

Parameter	Value	Function
Source	0 1	Sets trigger source to Internal Sets trigger source to External
Slope	0, Pos 1, Neg	Sets trigger slope to Positive Sets trigger slope to Negative
Mode	0, Single 1, Segment	Sets trigger mode to Single Shot Sets trigger mode to Segmented
Level	0-255	

Return Value

This function returns a 0 on success, -1 if <brd> is invalid and -2 if any other parameter is out of range.

Related Functions

trigmode500, *trigslope500*, *trigsource500*

3.5.65 trigdelay500

Form

DOS	short trigdelay500(unsigned short brd, unsigned short delay);
Windows	SHORT trigdelay500(P500ID * brd, ULONG delay, BYTE rw);

Description

The function *trigdelay500* writes a 16-bit delay value to the trigger delay register (TPD) on the PDA500 board. When using delayed trigger, the acquisition of data will be delayed from the trigger event. The length of the delay is the number of digitizer clock cycles specified by the parameter <delay>.

DOS

The parameter <brd> selects whether this function writes to the trigger delay register (TPD) or returns the value last written to the register. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <delay> is written to the trigger delay register on the corresponding PDA500 board. When <brd> takes on the value 1 through 8 the value in the register is not changed, instead, the last value written to the corresponding PDA500 board register is returned.

The parameter <delay> specifies the number of effective digitizer clock cycles times 8 after the trigger event to begin acquisition. This value must be in the range of 0 - 65535 which results in a range of values from 0 - 524280.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then <delay> is returned. If 1 through 8 is passed in <brd>, the last value written to the corresponding PDA500 register is returned. Returns -1 if <brd> is invalid, or delay is not within the specified range.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <delay> specifies the number of effective digitizer clock cycles times 8 after the trigger event to begin acquisition. This value must be in the range of 0 - 65535 which results in a range of values from 0 - 524280.

<rw> is the READ/WRITE flag. If READ is passed then the last value written to the board is returned. If WRITE is passed then <delay> is written to the board and returned.

Return Value

If <rw> is WRITE then <delay> is returned. If <rw> is READ, the last value written to the board is read from the software registers and returned. -1 is returned if <brd> is invalid or if <delay> is not within the specified range.

3.5.66 triglevel500

Form

DOS	short triglevel500(unsigned short brd, unsigned short val);
Windows	SHORT triglevel500(P500ID * brd, USHORT val, BYTE rw);

Description

The function *triglevel500* writes an 8-bit value to the trigger level DAC on the PDA500 board. The trigger level sets the actual voltage at which a trigger signal is detected. The selected trigger signal (see *trigsource500*) passing through the trigger level in the direction of the specified trigger slope (see *trigslope500*) constitutes a valid trigger.

DOS

The parameter <brd> selects whether this function writes to the trigger level register (TL) or returns the value last written to the register. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <val> is written to the trigger register on the corresponding PDA500 board. When <brd> takes on the value 1 through 8 the value in the register is not changed, instead, the last value written to the corresponding PDA500 board register is returned.

The parameter <val> can take on values from 0 to 255. The actual trigger level produced depends upon the trigger source. See section 2.8.5 for the actual trigger levels produced by different values for <val>.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then <val> is returned. If 1 through 8 is passed in <brd>, the last value written to the corresponding PDA500 register is returned. Returns -1 if <brd> is invalid.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <val> can take on values from 0 to 255. The actual trigger level produced depends upon the trigger source. See section 2.8.5 for the actual trigger levels produced by different values for <val>.

<rw> is the READ/WRITE flag. If READ is passed then the last value written to the trigger level register (TL) is returned. If WRITE is passed then <val> is written to the TL and returned.

Return Value

If <rw> is WRITE then <val> is returned. If <rw> is READ, the last value written to the TL is read from the software registers and returned. If <brd> is invalid then -1 is returned.

3.5.67 trigmode500

Form

DOS	short trigmode500(unsigned short brd, unsigned char Mode);
Windows	SHORT trigmode500(P500ID * brd, UCHAR Mode, BYTE rw);

Description

The function *trigmode500* allows the user to select one of two trigger modes for the PDA500 board. The two modes are:

Single Shot (Single): Data is acquired to the signal memory until the end address is reached. (The function *setendadd500* is used to set this address.)

Segmented Mode (Segment): For each trigger condition, data is acquired to a fixed portion of the signal memory. The PDA500 continues to store data segments to signal memory with each successive trigger until the end address is reached. (The function *segmentsize500* is used to set the size of the memory segments.)

DOS

The parameter <brd> selects whether this function writes to the TM register or returns the last value written to the register. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <Mode> is written to the corresponding PDA500 board. When <brd> takes on the value 1 through 8, the value in the register is not changed, instead, the last value written to the corresponding PDA500 board is returned.

The parameter <Mode> can take on the value of 0 or 1 to set the trigger mode to Single-Shot or Segmented. Additionally, the global variables Single and Segment can also be passed for <Mode>.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then <Mode> is returned. If 1 through 8 is passed in <brd>, a value of 0 is returned if the mode was set to Single Shot, 1 is returned if the mode was set to Segmented, and 2 is returned if the mode was set to Pre-trigger. Returns -1 if <brd> is invalid.

Related Functions

trig500, *trigslope500*, *trigsource500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <Mode> can take on the value of 0 or 1 to set the trigger mode to Single-Shot or Segmented. Additionally, the global variables Single and Segment can also be passed for <Mode>.

<rw> is the READ/WRITE flag. If READ is passed then the last value written to the trigger mode register (TMD) is returned. If WRITE is passed then <val> is written to the TMD and returned.

Return Value

If <rw> is WRITE then <Mode> is returned. If <rw> is READ, the last value written to the TMD is read from the software registers and returned. If <brd> is invalid then -1 is returned. The value returned is as follows:

0	Single Shot
1	Segmented

3.5.68 trigslope500

Form

DOS	short trigslope500(unsigned short brd, unsigned char Slope);
Windows	SHORT trigslope500(P500ID * brd, UCHAR Slope, BYTE rw);

Description

The function *trigslope500* allows the user to set the PDA500 board trigger slope to positive or negative by writing a 0 or a 1 to the TSL register.

DOS

The parameter <brd> selects whether this function writes to the TSL register or returns the value last written to the register. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <Slope> is written to the corresponding PDA500 board. When <brd> takes on the value 1 through 8 the value in the register is not changed, instead, the last value written to the corresponding PDA500 board is returned.

The parameter <Slope> can take on the value of 0 or 1 to set the coupling to positive or negative. Additionally, the global variables Pos and Neg can also be passed for <Slope>.

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then <Slope> is returned. If 1 through 8 is passed in <brd>, a value of 0 is returned if the slope was set to Positive and a value of 1 is returned if the slope was set to Negative. Returns -1 if <brd> is invalid.

Related Functions

trig500, *triglevel500*, *trigmode500*, *trigsoft500*, *trigsource500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <Slope> can take on the value of 0 or 1 to set the coupling to positive or negative. Additionally, the global variables Pos and Neg can also be passed for <Slope>.

<rw> is the READ/WRITE flag. If READ is passed then the last value written to the trigger slope register (TSL) is returned. If WRITE is passed then <Slope> is written to the TSL and returned.

Return Value

If <rw> is WRITE then <Slope> is returned. If <rw> is READ, the last value written to the TSL register is read from the software registers and returned. A 0 is returned for Positive and a 1 is returned for Negative. If <brd> is invalid then -1 is returned.

Related Functions

trig500, *triglevel500*, *trigmode500*, *trigsoft500*, *trigsource500*

3.5.69 trigsoft500

Form

DOS	void trigsoft500(unsigned short brd);
Windows	void trigsoft500(P500ID * brd);

Description

The function *trigsoft500* allows the user to issue a software trigger to the PDA500 board. This is accomplished by setting the ST register to 1 and back to 0.

DOS

The parameter <brd> determines which PDA500 board will receive the software trigger. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

Return Value

If <brd> is invalid, -1 is returned; otherwise 0 is returned.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

Return Value

If <brd> is invalid, -1 is returned; otherwise 0 is returned.

3.5.70 trigsource500

Form

DOS	short trigsource500(unsigned short brd, unsigned char Source);
Windows	SHORT trigsource500(P500ID * brd, UCHAR Source, BYTE rw);

Description

The function *trigsource500* allows the user to set the PDA500 board trigger source to one of three choices by writing to the trigger source bits (TRS0, TRS1) of register 0. If the trigger source is set to one of the internal settings, the signal to the ADC is used for the trigger. If the trigger source is set to one of the external settings, the signal from the external trigger input connector is used for the trigger source.

DOS

The parameter <brd> selects whether this function writes to the TRS register or returns the value last written to the register. Parameter <brd> can take on the values Brd1P500 through Brd8P500 or 1 through 8. When <brd> takes on the value Brd1P500 through Brd8P500, <Source> is written to the corresponding PDA500 board. When <brd> takes on the value 1 through 8, the value in the register is not changed. Instead, the last value written to the corresponding PDA500 board is returned.

The parameter <Source> can take on the value of 0 or 1 to select one of the following trigger sources:

<Source>	Selection
0	Internal
1	External

Return Value

If Brd1P500 through Brd8P500 is passed in <brd> then <Source> is returned. If 1 through 8 is passed in <brd>, the present value of TRS is returned. If <brd> is invalid, -1 is returned.

Related Functions

trig500, *trigmode500*, *trigslope500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <Source> can take on the value of 0 or 1 to select one of the following trigger sources:

<Source>	Selection
0	Internal
1	External

<rw> is the READ/WRITE flag. If READ is passed then the last value written to the trigger source register (TRS) is returned from the software registers. If WRITE is passed then <Source> is written to the board and returned.

Return Value

If <rw> is WRITE then <Source> is returned. If <rw> is READ, the last value written to the TRS register is read from the software registers and returned. If <brd> is invalid then -1 is returned.

3.5.71 uninstallpciint500

Form

DOS	int uninstallpciint500(unsigned short brd);
Windows	N/A

Description

The function *uninstallpciint500* removes a PDA500 interrupt service routine from the system. In doing so, if the interrupt was previously in use, the old interrupt handler is reinstalled. This function should be called before program execution is terminated if an interrupt handler was installed for the PDA500.

The parameter <brd> determines which PDA500 board will have the interrupt handler uninstalled. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

Return Value

The function *uninstallpciint500* returns a 0 if the installation of the interrupt handler was successful uninstalled. A value of -1 is returned if an error occurs.

3.5.72 voltrange500

Form

DOS	float voltrange500(char *filename, unsigned short chan, unsigned char attn);
Windows	FLOAT voltrange500(char *filename, USHORT chan, USHORT attn);

Description

The function *voltrange500* is used to provide a calibrated value for the full-scale input signal voltage range of a PDA500 board for all attenuation settings. A calibration file specifying the full-scale voltage for each attenuator setting is provided for each PDA500A board.

The parameter <filename> is the name of the calibration file provided with the PDA500 board including drive and path. The calibration file name for a board is the serial number of the board with the file extension "ptb". Serial numbers are typically 6 digits in length.
The parameter <chan> must have the value 1 or 2 which specifies whether channel 1 or channel 2 data is desired.

The parameter <attn> is the attenuator setting for which the full-scale voltage value is desired and can take on the values 0 to 26 in steps of 2.

Return Value

If successful, the full-scale input voltage range for the specified gain and channel setting is returned. The voltage contains the units of volts peak to peak. A return value of 0 indicates either an invalid channel or the attenuation value is out of range, a –1 indicates the file could not be found, and a –2 indicates an invalid file.

Related Functions

attnset500

3.5.73 waitforACFint500

Form

DOS	N/A
Windows	void waitforACFint500(P500ID * brd);

Description

The function *waitforACFint500* waits for an interrupt from the PDA500. Before calling this function, a call to the function *enableACFint500* must be made with a 1 passed into the parameter <val>. (Calling *enableACFint500* enables the interrupt.) The interrupt MUST be enabled before the waitforACFint500 function is called.

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

Return Value

None

Related Functions

enableACFint500

Examples

```
//console example which shows how to use enableACFint500 and waitforACFint500
#include<windows.h>
#include"pda500win.h"
#include <stdio.h>

void main()
{
        P500ID id;
        P500SWR swr;

        setupregisters500(&swr,1);
        opendevice500(&swr,&id);
        //set up board
        mode500(&id,StbyMode,WRITE);
        offset500(&id,128,WRITE);
        triglevel500(&id,150,WRITE);
        attnset500(&id,0,WRITE);
        trigmode500(&id,0,WRITE);
        setstartadd500(&id,0,WRITE);
        setendadd500(&id,1048575,WRITE);
        //enable interrupt
        enableACFint500(&id,1);
        //start acquisition
        mode500(&id,Acq1Mode,WRITE);
        //wait for the interrupt
        waitforACFint500(&id);
        mode500(&id,StbyMode,WRITE);
```

```
        //data is normally retrieved here


        //disable interrupt
        enableACFint500(&id,0);
        mode500(&id,0,WRITE);
        closedevice500(&id);
}
```

3.5.74 writeEPROM500

Form

DOS	short writeEPROM500(unsigned short brd, unsigned short parm, short val);
Windows	SHORT writeEPROM500(P500ID * brd, ULONG parm, SHORT val);

Description

The function *writeEPROM500* is used to write a value to the configuration EEPROM on the PDA500. The main purpose of the EEPROM is to load the PCI controller with the PDA500 default values. Other parameters such as the board Serial Number are stored in the EEPROM. There are also user definable EEPROM addresses, which can be used to store values. See the function *readEPROM500* for a description of the EEPROM address locations.

DOS

The parameter <brd> selects which PDA500 board is written to. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

The parameter <parm> specifies the EEPROM address that will be written to.

The parameter <val> is the value that will be written to the specified configuration EEPROM address.

Return Value

Returns <val> when successful. If the specified board is not found, or if a PCI error occurs, -1 is returned.

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <parm> specifies the EEPROM address that will be written to.

The parameter <val> is the value that will be written to the specified configuration EEPROM address.

Return Value

Returns <val> when successful. If the <brd> parameter is invalid -1 is returned.

3.5.75 writeram500

Form

DOS	Long writeram500(unsigned short brd, unsigned long saddr, unsigned long len, unsigned char * source);
Windows	LONG writeram500(P500ID * brd, ULONG saddr, ULONG len, UCHAR * source);

Description

The function *writeram500* allows the user to fill a section of PDA500 signal RAM with a buffer allocated on the PC.

DOS

The parameter <brd> selects which PDA500 board is written to. Allowable entries are Brd1P500 through Brd8P500.

The starting address parameter <saddr> and the ending address parameter <len> can take on values from 0 to 4194296.

The <source> parameter points to the buffer that is to be used to fill the RAM. It must be at least as long as <len>.

Return Value

If <brd> is invalid, -1 is returned; otherwise len is returned.

Related Functions

brd2pc500

Example

```
//This example will write a series of ramps to the PDA500
#include <stdio.h>
#include "pda500.h"

void main(void)
{
unsigned char writebuffer[256];
int count;
unsigned long writecount;

//setup and find first board
setupbrd500();
findbrd500(&Brd1P500,0L);

//put in Standby Mode
mode500(Brd1P500,StbyMode);

//fill buffer
for(count=0;count<256;count++)
{
    writebuffer[count]=count;
```

```
}

//write ram to board
printf("Writing Ram\n");

//write 64k of ramps
for(writecount=0;writecount<65536L;writecount+=256)
{
    writeram500(Brd1P500,writecount,256,writebuffer);
    printf(".");
}

printf("\nFinished Writing to RAM\n");

//turn the board off
mode500(Brd1P500,OffMode);
}
```

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The starting address parameter <saddr> can take on values from 0 to 4194296 and is used to set the starting address for the write. The starting address + the length can not exceed 4194296.

The <source> parameter points to the buffer that is to be used to fill the RAM. It must be at least as long as <len>.

Return Value

If <brd> is invalid, -1 is returned; otherwise len is returned.

Related Functions

brd2pc500

3.5.76 writereg500

Form

DOS	Short writereg500(unsigned short brd, unsigned short regnum);
Windows	SHORT writeregs500(P500ID * brd, USHORT regnum);

Description

The function *writereg500* is used to directly write a 32-bit value to a PDA500 hardware register. The value written is taken from the corresponding software register. See "**APPENDIX B - Hardware Registers**" for the bit patterns of the hardware registers.

DOS

The parameter <brd> selects which PDA500 board is written to. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

The parameter <regnum> specifies the register to be written and can take on the values 0 - 7 or 4 - 7.

Return Value

Returns 1 if successful. Returns -1 if <brd> is invalid.

Related Functions

readreg500, *setupbrd500*, *writeregs500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

The parameter <regnum> specifies the register to be written and can take on the values 0 - 7 or 4 - 7.

Return Value

Returns 1 if successful, 0 if write failed, or -1 if <brd> is invalid.

Related Functions

readreg500, *setupbrd500*, *writeregs500*

3.5.77 writeregs500

Form

DOS	Short writeregs500(unsigned short brd);
Windows	SHORT writeregs500(P500ID * brd);

Description

The function *writeregs500* is used to directly write to hardware registers 0 - 7 the values stored in the corresponding software registers. The software registers are discussed in section 3.3.3.3.1 of this manual.

DOS

The parameter <brd> selects which PDA500 board is written to. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

Return Value

Returns 1 if successful. Returns -1 if <brd> is invalid.

Related Functions

readreg500, *setupbrd500*, *writereg500*

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

Return Value

Returns a 1 if successful. Returns -1 if <brd> is invalid.

Related Functions

readreg500, *setupbrd500*, *writereg500*

3.5.78 writeset500

Form

DOS	short writeset500(unsigned short brd, char *filename);
Windows	SHORT writeset500(P500ID * brd, CHAR * filename);

Description

This function writes a 256-byte file header to a data file that contains information about the data that is saved.

The file header is defined as follows:

Byte #	Contents
0-3	Board Type 500=PDA500
4-7	Register 0
8-11	Register 1
12-15	Register 2
16-19	Register 3
20-23	Register 4
24-27	Register 5
28-31	Register 6
32-35	Register 7
36-255	Reserved for User Comments

Register information is taken from the global variable P500REGPTR and is of the form
P500REGPTR[board#-1][Register#] where board# equates to brdnum(brd).

Comments are character strings up to a maximum of 219 characters, which allow reference information to be attached to a data file.

DOS

The parameter <brd> selects which PDA500 board the parameters are taken from. Parameter <brd> can take on the values Brd1P500 through Brd8P500.

Parameter <filename> is the name of a file, which will be created. The 256- byte header is written to the file and the file is closed.

Return Value

If an error is detected, -1 is returned; otherwise 1 is returned.

Related Functions

readset500

Windows

The parameter <brd> selects which PDA500 board will be operated. This variable of type P500ID is declared by the user and is initialized using the function *opendevice500*.

Parameter <filename> is the name of a file, which will be created. The 256-byte header is written to the file and the file is closed.

Return Value

If an error is detected, -1 is returned; otherwise 1 is returned.

Related Functions

readset500

4 APPENDIX A - PDA500 Specifications and Ordering Information

Specifications

Input Signals
Channel 1 Analog Input
Channel 2 Analog Input
External Trigger (in/out) [1]
External Clock (in/out) [1]
External Connectors : 4 SMA

Analog Inputs
full scale voltage : 200 mV p-p to 3.0 V p-p[2]
impedance : 50 ohms or 1Meg
bandwidth : 100 MHz or 500 MHz[1]
equivalent noise : 0.5 lsb RMS (typical)
coupling : AC or DC[1]

External Trigger[3]
Impedance[5] : 50 ohms / 1k ohms[1]
trigger level[5] : ±250 mV @ 50 ohms, ±2.5 V @ 1k ohm
adjustment method[5] : via 8 bit DAC[2]
bandwidth[5] : 350 MHz @ 50 ohms, 10 MHz @ 1k ohms
coupling [5] : AC or DC[1]
output trigger[6] : ECL synchronous with start & end of acquisition

External Clock[4]
signal type[5] : ECL logic or sine/square wave input
impedance[5] : 50 ohms to -2 V or ground1
frequency[5] : 50 MHz to 500 MHz[7]
amplitude[5] : 100 mV p-p to 2.0 V p-p
output clock[6] : ECL clock one-eighth digitizer frequency

Attenuator
range : 0 to 26 dB
steps : 14, 2 dB per step
control : digital, 4 bits[2]

DC Offset Voltage
8 bit DAC, ±270 mV at ADC input

Digitizer
voltage range : ±270 millivolts full scale
resolution : 8 bits
linearity, integral : ±0.5 lsb max.
linearity, differential : ±0.75 lsb max.
aperture width : 270 ps typical
aperture jitter : 2 ps typical
internal clock rates : 500 MHz to 781.25 kHz in factors of 2,
200 MHz to 312.5 kHz in factors of 2,
divide by 5 of all above frequencies
internal clock accuracy: +/- .01%

Trigger Modes
single shot : single start trigger fills active memory
segmented : start trigger for each memory segment

Trigger Methods
Pre-trigger : samples prior to trigger are stored
0 to 4k samples in steps of 8
Delayed Trigger : delay from trigger to data storage
0 to 65k digitizer clocks in steps of 8

Memory
Size : 1 or 4 megabytes
Start Address Setting[8] : 0 to memory size in steps of 8
End Address Setting[8] : 0 to memory size in steps of 64
segment re-arm time[9] : 150 nanoseconds
Addressing : via Memory Mapping or DMA
Memory Address (PC): Plug and Play selected

I/O Addressing
PCI Controller Address: 64 bytes, Plug and Play selected
Control/Status Registers: 32 bytes, Plug and Play selected

Signatec Auxiliary Bus
data transfer modes : 64 bit burst, 32 bit burst
data transfer rates : 500 MB/s max in burst mode
data direction : output only

Power Down Features
Off Mode : Board Deactivated; Power usage less than 350 mW
Power Down : Deactivation of acquisition circuits reduces power by 2/3
Thermal Shut-down : ADC temp. greater than 65?C deactivates power

Power Requirements
+12 : 1.5 Amps max. (normal operation)
190 mA max. (power down and off mode)
-12 : 10 mA max. (all modes)
+5 : 2.5 Amps max. (acquisition), 4.5 Amps w/MX5-8
0.7 Amps max. (power down), 1.3 Amps w/MX5-8
120 mA max. (off mode)

Absolute Maximum Ratings
Analog Inputs : ±5 volts
Trigger Input : ±5 volts
Clock Input : 5 volts peak to peak
Ambient Temperature : 0 to 50?C

Ordering Information

PDA500 Board

1 Meg sample memory	Part number: PDA500-1M
4 Meg sample memory	Part number: PDA500-4M

SAB Cables
Refer to the "SAB Cable Assembly Ordering Guide" to select and order the appropriate cable assemblies.

Master-Slave Cables
The PDA500 is configured to operate as a Master or a Slave via solder jumpers on the board, which can be set by Signatec. In order to operate in a Master/Slave configuration a 20-pin ribbon cable is required to connect multiple boards. This cable is ordered using the basic part number PDA500MS-X where X is the total number of boards connected together. Master/Slave boards must occupy adjacent slots. The maximum number of boards to be connected is one master and three slaves. The part number for this connector is: PDA500MS-4.

Documentation & Accessories
The PDA500 is supplied with a comprehensive operator's manual, which thoroughly describes the operation of both the hardware and the software. Also supplied are two four-foot coaxial cables with BNC to SMA connectors. Extra cables may be purchased from Signatec. Supplied software disks contain a DOS function library for Borland C, Windows 95/NT function libraries for Borland and Microsoft Visual C/C++, example programs, and all source code to libraries and examples.

Customer Support
Customer Support and Software Updates can be obtained from the Signatec Web Page: www.signatec.com, via email to technical support at techsupport@signatec.com, or by calling Signatec at (909) 734-3001.

Product Warranty
All Signatec products carry a full 3-year warranty. During the warranty period, Signatec will repair or replace any defective product at no cost to the customer. This warranty does not cover physical damage not reported within 15 days of the time of shipment to Signatec.

Notes:
1. Selected via dip-switch.
2. Software selectable.
3. External Trigger connector is selected as trigger input or an output trigger, synchronized with the start of acquisition using a dip-switch
4. External Clock connector is selected as clock input or an output clock, which is synchronous to the digitizer clock using a dip-switch.
5. Applies to input only.
6. Applies to output only.
7. Internal divider is operational.
8. Active memory of acquisition is end address minus start address.
9. In segmented mode, time from the end of a segment until a trigger will be accepted to begin another segment acquisition.

Signatec reserves the right to make changes in this specification at any time without notice. The information furnished herein is believed to be accurate, however no responsibility is assumed for its use.

5 APPENDIX B - Hardware Registers

Hardware Register Mnemonics

Register 0

Name	Function	Hardware Reference	Software Function
OM0 - OM2	Operation Mode	2.7	*mode500*
PDO	Power Down Override	2.12.3	*pwdownor500*
CHSEL	Channel Select	2.3.2	*chanset500*
DIV5	Divide by 5	2.6	*clkdivbyfive500*
TMD	Trigger Mode	2.8	*trigmode500*
TRS	Trigger Source	2.8.3	*trigsource500*
HYS	Hysterisis	2.8.6	*hyst500*
TSL	Trigger Slope	2.8.4	*trigslope500*
ST	Software Trigger	2.8.7	*trigsoft500*
CS0 - CS1	Clock Source	2.6	*clkfreq500*
CK0 - CK2	Clock Divider	2.6.1	*clkdivide500*
SS	Segment Size	2.9.3	*segmentsize500*
ATT0-ATT3	Attenuation	2.3.2	*attnset500*
DMA PIE	DMA Enable PCI Enable	2.7	*enabledma500* *mode500*
TM0 - TM1	Trigger Method		*trigdelay500*
HIZ	High Impedance		

Register 1

Name	Function	Hardware Reference	Software Function
OF0 - OF7	DC Offset	2.3.5	*offset500*

Register 2

Name	Function	Hardware Reference	Software Function
AC	Address Counter	2.5	*setstartadd500*
ACRST	Address Counter Reset	2.9.1	*autoaddrst500*
SD	Segment Detect	2.8.1.2	*segmentdetect500*
PRF	Power Ready Flag	2.12.2	*pwrdy500*
FF	FIFO Full	2.7.2	*fifoflag500*
ACF	Acquisition Complete Flag	2.7.3	*acqcomp500*

Register 4

Name	Function	Hardware Reference	Software Function
EC	Ending Address	2.9.2	*setendadd500*
FR	Free Run	2.9.2	*setendadd500*

Register 5

Name	Function	Hardware Reference	Software Function
TL	Trigger Level	2.8.5	*triglevel500*

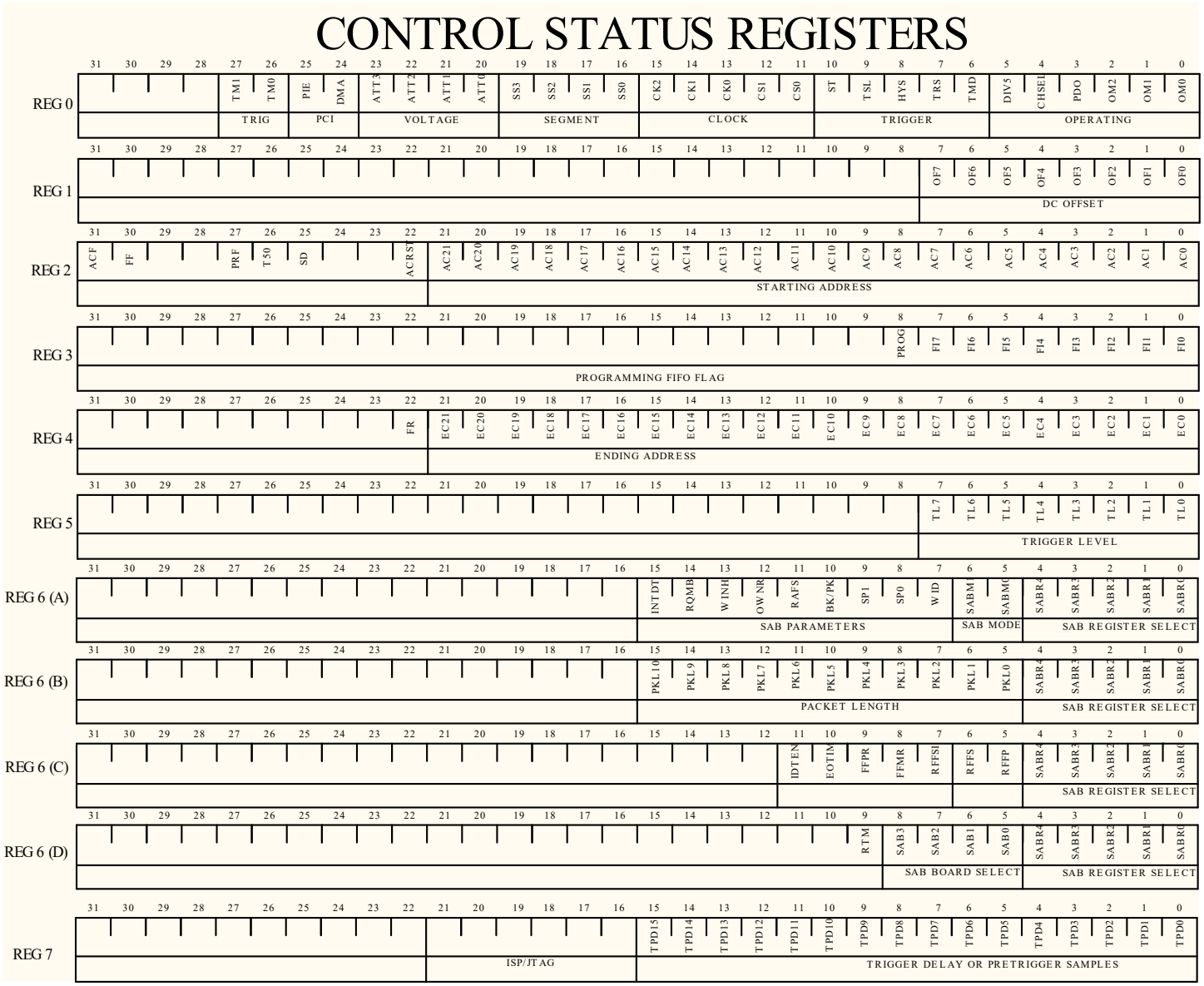
Register 6

Name	Function	Hardware Reference	Software Function
SABREG	SAB Register Select		
SABMODE	SAB Mode: Inactive Slave/Active Slave /Master		*acqSABdata500, mode500*
RDATAW	Data Width: 32 or 64		*SABinit500, SABconf500*
SABCLK	SAB Clock: 25/62.5/66/External		*SABclk500, SABconf500*
BPK	Block/Packet Mode Transfer	2.11.1.1	
RAFS	Auto FIFO Status		
ROWNER	Bus Owner/Non-owner Operation		
RWAITINH	Wait Inhibit		
RQMBS	Quick Memory Board Switch		*SAB1init500 (membrdnum)*
INTDT	SAB Interrupt Detect Mask		
PKTLENGTH	Packet Length		
RFFPROG	FIFO Programming Mode		
RFFSEL	I/O FIFO Select		
RFFSI	FIFO Serial Input		
RFFMRST	FIFO Master Reset		
RFFPRST	FIFO Partial Reset		
EOTIMAST	End of Transfer Interrupt Inhibit		
IDDTEN	Enable Interrupt Driver Transfer Communication		*SABinit500 (control)*
SABBRDNUM	SAB Board Select		*SABinit500, SABbrdnum500*
RTEMEN	Termination Enable		

Register 7

Name	Function	Hardware Reference	Software Function
TPD0 - TPD15	Trigger Delay or Pretrigger Samples	2.8.2.1 2.8.2.2	*trigdelay500*
ISP0 – ISP6	In System Programming		

6 APPENDIX C - Hardware Registers



7 APPENDIX D - Master-Slave Connections

Connector J6

Pin No.	Connection
1	SCLK1
2	GND
3	TRK1
4	GND
5	RES1
6	GND
7	SCLK2
8	GND
9	TRK2
10	GND
11	RES2
12	GND
13	SCLK3
14	GND
15	TRK3
16	GND
17	RES3
18	GND
19	GND
20	TRIG/TRIGOUT
21	MODE
22	SCLK0
23	SDI
24	ISPEN-
25	+5V
26	SDO

Notes:

1. SCLKx- and SCLKx+ are the differential ADC clock signals for slave boards number x.
2. TRIGOUT is a trigger enable signal used by all slave boards.
3. Pins 21 through 26 are used for in-circuit programmable logic on the board.

8 APPENDIX E - Signatec Auxiliary Bus Connections

All odd numbered pins on J1 and J2 are grounded.

Connector J1		Connector J2	
Pin No.	**Connection**	**Pin No.**	**Connection**
100	DB32	100	DB0
98	DB33	98	DB1
96	DB34	96	DB2
94	DB35	94	DB3
92	DB36	92	DB4
90	DB37	90	DB5
88	DB38	88	DB6
86	DB39	86	DB7
84	no connection	84	REQUEST-A LOW
82	no connection	82	WAIT-32 LOW
80	no connection	80	BUSY LOW
78	no connection	78	REPLY-A LOW
76	DB40	76	DB8
74	DB41	74	DB9
72	DB42	72	DB10
70	DB43	70	DB11
68	DB44	68	DB12
66	DB45	66	DB13
64	DB46	64	DB14
62	DB47	62	DB15
60	DB48	60	DB16
58	DB49	58	DB17
56	DB50	56	DB18
54	DB51	54	DB19
52	DB52	52	DB20
50	DB53	50	DB21
48	DB54	48	DB22
46	DB55	46	DB23
44	no connection	44	STROBE +32
42	no connection	42	STROBE-32 LOW
40	no connection	40	SLOW-32 LOW
38	no connection	38	No connection
36	DB56	36	DB24
34	DB57	34	DB25
32	DB58	32	DB26
30	DB59	30	DB27
28	DB60	28	DB28
26	DB61	26	DB29
24	DB62	24	DB30
22	DB63	22	DB31
20	no connection	20	MSEL0
18	no connection	18	MSEL1/COM2
16	no connection	16	COM3
14	no connection	14	COM4
12	no connection	12	no connection
10	no connection	10	INTR1
8	no connection	8	INTR2
6	no connection	6	INTR3
4	no connection	4	INTR4
2	no connection	2	INTR5

9 APPENDIX F - C Library Table

Function	DOS Borland 5.02	Win NT/95 32-bit MSVC++ 6.0
acqcomp500	Yes	Yes
acqPCIdata500	Yes	Yes
acqPCIdatabuffers500	No	Yes
acqRAMdata500	Yes	Yes
acqRAMdataDMA500	No	Yes
acqSABdata500	Yes	Yes
allocatedmabuffer500	No	Yes
allocatedmabuffers500	No	Yes
attnset500	Yes	Yes
autoaddrst500	Yes	Yes
brd2disk500	Yes	Yes
brd2pc500	Yes	Yes
brd2pcdma500	Yes	Yes
brdname500	Yes	No
brdnum500	Yes	No
cancelACFint500	No	Yes
chanset500	Yes	Yes
clkdivbyfive500	Yes	Yes
clkdivide500	Yes	Yes
clkfreq500	Yes	Yes
clock500	Yes	Yes
closedevice500	No	Yes
disk2brd500	Yes	Yes
disk2pc500	Yes	Yes
dmaint500	Yes	No
enableACFint500	No	Yes
enabledma500	Yes	No
fftstats500	Yes	Yes
fifoflag500	Yes	Yes
findbrd500	Yes	No
freeallbuffers500	No	Yes
freedmabuffer500	No	Yes
freedmabuffers500	No	Yes
hyst500	Yes	Yes
IObase500	Yes	No
membase500	Yes	No
mode500	Yes	Yes
nulloff500	Yes	Yes
offset500	Yes	Yes
opendevice500	No	Yes
presamples500	Yes	Yes
pwdownor500	Yes	Yes
pwrdy500	Yes	Yes
readEPROM500	Yes	Yes

Function	DOS Borland 5.02	Win NT/95 32-bit MSVC++ 6.0
readreg500	Yes	Yes
readset500	Yes	Yes
realfft500	Yes	Yes
realfftmag500	Yes	Yes
SABbrdnum500	Yes	Yes
SABclk500	Yes	Yes
SABcompatible500	Yes	Yes
SABconf500	Yes	Yes
SABinit500	Yes	Yes
segmentsize500	Yes	Yes
sendEOI	Yes	No
setendadd500	Yes	Yes
setstartadd500	Yes	Yes
setuppciint500	Yes	No
setupbrd500	Yes	No
setupregisters500	No	Yes
temperature500	Yes	Yes
trig500	Yes	Yes
trigdelay500	Yes	Yes
trigdelayen500	Yes	Yes
triglevel500	Yes	Yes
trigmode500	Yes	Yes
trigslope500	Yes	Yes
trigsoft500	Yes	Yes
trigsource500	Yes	Yes
uninstallpciint500	Yes	No
voltrange500	Yes	Yes
waitforACFint500	No	Yes
writeEPROM500	Yes	Yes
writeram500	Yes	Yes
writereg500	Yes	Yes
writeregs500	Yes	Yes
writeset500	Yes	Yes

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