



ULTRADMA-100 Series

High Performance 14/12/8-Bit PCI Bus Data Acquisition Boards with 64/32-bit DMA

Models: ADDA14-100DMA Dual 25 Ms/s 14-bit A/D + Dual 14-bit D/A
AD14-100DMA Dual 25 Ms/s 14-bit A/D
ADDA12-100DMA Dual 25 Ms/s 12-bit A/D + Dual 14-bit D/A
AD12-100DMA Dual 25 Ms/s 12-bit A/D
AD8-100DMA Dual 50-Ms/s and Single 100 MHz 8-bit A/D
AD8S-100DMA Single 100 MHz 8-bit A/D, Ultra-low Spur
AD8S-100DMA-FPDP Single 100 MHz 8-bit A/D, Ultra-low Spur w/FPDP TM
AD8CH12B-100DMA 8-Channel 8/12 bit A/D

Product Specification

January 5, 2007

Covers Boards With Firmware rev 5/7/05, 5/16/05 and 8/4/06
For Solaris 8,9,10™, RedHat™ Enterprise Linux™ WS4 64bit, RedHat™ Linux™
9 (32-bit), Windows 2000™ and Windows XP™ (32-bit)

Ultraview Corporation

34 Canyon View, Orinda, CA 94563

(925) 253-2960

Fax (925) 253-4894

e-mail : support@ultraviewcorp.com

URL: www.ultraviewcorp.com

copyright c 2002, 2003, 2004, 2005, 2006 Ultraview Corporation

TABLE OF CONTENTS

1. WARRANTY.....	4
2. MODEL DESCRIPTIONS.....	5
2.1 MODEL ADDA14-100DMA	5
2.2 MODEL AD14-100DMA	5
2.3 MODEL ADDA12-100DMA	5
2.4 MODEL AD12-100DMA	6
2.5 MODEL AD8-100DMA, AD8S-100DMA AND AD8S-100DMA-FPDP.....	6
2.6 MODEL AD8CH12B-100DMA.....	6
3. SPECIFICATIONS	7
3.1 A/D CONVERTERS.....	7
3.2 D/A CONVERTERS (MODELS ADDA14, ADDA12 AND DA14-100DMA).....	8
3.3 FAST VECTORED TTL INPUTS (MODELS ADDA14, ADDA12, AD14 AND AD12).....	8
3.4 FAST VECTORED TTL OUTPUTS (ADDA14, ADDA12, AD14, AD12-100DMA).....	8
3.5 LOW SPEED SOFTWARE-PROGRAMMED TTL OUTPUTS (MODELS ADDA14, AD14, ADDA12, AD12 AND AD8CH12B-100DMA).....	8
3.6 FRONT PANEL DATA PORT (FPDP) INTERFACE (AD8S-100DMA-FPDP ONLY).....	9
3.7 GENERAL.....	9
3.8 PHYSICAL.....	9
4. HARDWARE ARCHITECTURE	11
4.1 ANALOG INPUTS	11
4.2 ANALOG OUTPUTS (MODELS ADDA14, ADDA12 AND DA14-100DMA ONLY).....	11
4.3 I/O CONNECTOR (MODELS ADDA14, AD14, ADDA12, AD12 AND DA14-100DMA ONLY).....	11
4.3.1 <i>Trigger Input Line</i>	13
4.3.2 <i>Event Input Line (ADDA14, ADDA12, AD14 and AD12 only)</i>	13
4.3.3 <i>TTL_In[7,5,4,1,0]</i>	13
4.3.4 <i>TTL_Out[1..0] (Shared with a D/A channel on boards with D/As)</i>	13
4.3.5 <i>External Clock (optional)</i>	14
4.3.6 <i>Programmed TTL outputs (OSSTB, OSCLK, OSDAT)</i>	14
4.4 FRONT-END MEZZANINE BOARD INTERFACE CONNECTIONS.....	15
4.5 LED INDICATORS.....	15
4.5.1 <i>Run LED</i>	15
4.5.2 <i>RDY (Ready) LED</i>	15
4.5.3 <i>DMA and D64 (64-Bit DMA Transfer) LED</i>	15
5. LOW-LEVEL SOFTWARE INTERFACE.....	16
5.1 PCI CONFIGURATION HEADER.....	16
5.2 ULTRADMA CONTROL REGISTER.....	17
5.2.1 <i>Software_Run bit (write only)</i>	17
5.2.2 <i>Buffer_Wrap bit (write only)</i>	18
5.2.3 <i>Interrupt_Enable bit (write only)</i>	18
5.2.4 <i>Double_Speed bit (write only)</i>	18
5.2.5 <i>D/A Mode bit (write only)</i>	19
5.2.6 <i>Internal Clock Mode bit (write only)</i>	19
5.2.7 <i>OSSTB bit (write only)</i>	19
5.2.8 <i>OSCLK bit (write only)</i>	20
5.2.9 <i>OSDAT bit (write only)</i>	20
5.2.10 <i>TimeStamp_Test bit (write only)</i>	20
5.2.11 <i>Sample Interval bits (write only) for model AD8-100DMA only</i>	21
5.2.12 <i>Sample Interval bits (write only) for model AD8S-100DMA and AD8S-100DMA-FPDP only</i>	21
5.2.13 <i>Sample Interval bits (write only) for ADDA14-100DMA, AD14-100DMA, ADDA12-100DMA</i>	

and AD12-100DMA.....	22
5.2.14 Sample Interval bits (write-only) for AD8CH12B-100DMA in 8-bit mode.....	23
5.2.15 Sample Interval bits for AD8CH12B-100DMA in 12-bit mode.....	23
5.2.16 DMA Block Size bits (write only).....	24
5.2.17 Board Interrupt after A/D or D/A block completed (read only status bit).....	24
5.2.18 Board Interrupt after DMA block completed (read only status bit).....	24
5.2.19 Board Stopped (read only status bit).....	24
5.2.20 DMA in progress (read only status bit).....	25
5.2.21 No Clock or Slow Clock (read only status bit).....	25
5.3 DMA LOW STARTING ADDRESS REGISTER AND READ/WRITE CONTROL.....	25
5.4 DMA HIGH STARTING ADDRESS REGISTER (FOR EXTENDED ADDRESSING ONLY).....	25
5.5 DATA REPRESENTATION IN HOST SYSTEM MEMORY DURING D/A TRANSFERS (MODELS ADDA14-100DMA, ADDA12-100DMA AND DA14-100DMA ONLY)	26
5.6 DATA REPRESENTATION IN HOST SYSTEM MEMORY DURING A/D TRANSFERS (MODELS ADDA14-100DMA AND AD14-100DMA ONLY)	27
5.7 DATA REPRESENTATION IN HOST SYSTEM MEMORY DURING A/D TRANSFERS (MODELS ADDA12-100DMA AND AD12-100DMA ONLY)	27
5.8 DATA REPRESENTATION IN HOST SYSTEM MEMORY DURING A/D TRANSFERS (MODEL AD8-100DMA, AD8S-100DMA AND AD8S-100DMA-FPDP ONLY)	28
5.9 DATA REPRESENTATION IN HOST MEMORY DURING A/D TRANSFERS (MODEL AD8CH12B-100DMA)	29
5.10 TIME STAMP FUNCTION (ADDA14/AD14 AND ADDA12/AD12-100DMA ONLY).....	29
6. HARDWARE INSTALLATION AND SETUP.....	30
7. SOFTWARE INSTALLATION AND SETUP.....	31
7.1 SOFTWARE INSTALLATION FOR WINDOWS XP™ OR WINDOWS 2000™	31
7.2 SOFTWARE INSTALLATION FOR SOLARIS 8,9 OR 10 (SPARC PLATFORM EDITION™)	31
7.3 SOFTWARE INSTALLATION UNDER RED HAT™ ENTERPRISE LINUX WS 4.0 (64-BIT).....	33
7.4 SOFTWARE INSTALLATION UNDER RED HAT™ LINUX 9.0 (32-BIT ONLY).....	34
7.5 RUNNING THE EXAMPLE PROGRAMS UNDER SOLARIS 8, 9 OR 10™.....	35
7.5.1 digosc (digital oscilloscope).....	35
7.5.2 fast_acquire_dma_data (acquire data to disk file).....	35
7.5.3 acquire_dma_data (Acquire data to disk file of any length).....	36
7.5.4 synth (dual sine wave synthesizer).....	37
7.5.5 da_from_disk (D/A playback of file).....	37
7.5.6 fast_dma_da_from_disk (play back of file at high speed).....	37
7.6 RUNNING THE EXAMPLE PROGRAMS UNDER RED HAT™ LINUX WS 4.0 64-BIT.....	39
7.6.1 acquire_data2.c (acquire data to RAM, then store to disk file).....	39
7.6.2 ad_mt.c (Acquire data to disk file of any length).....	40
7.6.3 ad_mt_wrap.c (Acquire data to disk, with pre-trigger recording capability).....	41
7.6.4 disk_test.c (test disk system throughput, using dummy data).....	42
7.6.5 da.c (D/A playback of file).....	42
7.6.6 digosc_file.c (for AD14-100DMA and ADDA14-100DMA only).....	42
7.7 RUNNING EXAMPLE PROGRAMS UNDER WINDOWS XP™ OR WINDOWS 2000™.....	44
7.7.1 acquire.exe.....	44
7.7.2 spitout.exe.....	44
7.7.3 inout.exe.....	44
7.7.4 Altering the Windows XP/2000 example programs above.....	45
8. APPENDIX A FPDP INTERFACE (AD8S-100DMA-FPDP ONLY).....	46
9. APPENDIX B. FRONT-END MEZZANINE BOARD INTERFACE.....	48
9.1 FRONT-END MEZZANINE INTERFACE PINOUT.....	48
10. APPENDIX C. USER LIBRARY COMMANDS FOR LINUX64 & SOLARIS.....	50
10.1 LINUX64 LIBRARY COMMANDS.....	50
10.2 SOLARIS LIBRARY COMMANDS.....	53

1. Warranty

Ultraview Corporation hardware, software and firmware products are warranted against defects in materials and workmanship for a period of two (2) years from the date of shipment of the product. During the warranty period, Ultraview Corporation shall, at its option, either repair or replace hardware, software or firmware products that prove to be defective. This limited warranty does not cover damage caused by misuse or abuse by customer, and specifically excludes damage caused by the application of excessive voltages to the inputs and/or outputs of data acquisition boards. The limited warranty additionally excludes damage caused by overheating due to installation of the product in systems that do not have direct forced airflow over the PCI bus slots.

While Ultraview Corporation hardware, software and firmware products are designed to function in a reliable manner, Ultraview Corporation does not warrant that the operation of the hardware, software or firmware will be uninterrupted or error free. Ultraview products are not intended for use as critical components in life support systems, aircraft, military systems or other systems whose failure to perform can reasonably be expected to cause significant injury to humans. Ultraview expressly disclaims liability for loss of profits and other consequential damages caused by the failure of any product, and recommends that customer purchase spare units for applications in which the failure of any product would cause interruption of work or loss of profits, such as industrial, shipboard or military equipment.

THIS LIMITED WARRANTY IS IN LIEU OF ALL OTHER WARRANTIES EXPRESSED OR IMPLIED. THE WARRANTIES PROVIDED HEREIN ARE BUYER'S SOLE REMEDIES. IN NO EVENT SHALL ULTRAVIEW CORPORATION BE LIABLE FOR DIRECT, SPECIAL, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES SUFFERED OR INCURRED AS A RESULT OF THE USE OF, OR INABILITY TO USE THESE PRODUCTS. THIS LIMITATION OF LIABILITY REMAINS IN FORCE EVEN IF ULTRAVIEW CORPORATION IS INFORMED OF THE POSSIBILITY OF SUCH DAMAGES.

Some states do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation and exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights that vary from state to state.

2. Model Descriptions

The ULTRADMA series of data acquisition and control boards are complete high-speed A/D and D/A systems on a single PCI bus card. Designed for low jitter operation in scientific, medical and industrial applications, these boards function in PCI bus systems using supplied drivers for either Solaris 8, 9 and 10 (Sparc Platform Edition™), Linux (Red Hat™ Enterprise Linux 4.0-64bit, Red Hat™ 9.0 32bit), Windows 2000™ or XP™. For highest throughput, ULTRADMA boards should be installed in Sun systems with 64-bit PCI slots or server systems based on AMD ATHLON64 or OPTERON processors with 64-bit PCI slots and 64-bit Linux or 32-bit Windows XP. Machines running 64-bit Linux or Windows XP (32-bit) usually achieve full speed operation on A/D mode, but may be limited to approximately 1/3 this speed when running the board in D/A mode.

As the ULTRADMA series of data acquisition boards relies on DMA to transfer large amounts of data to system RAM, for best results only one ULTRADMA board should be installed per system, to ensure adequate DMA throughput to support the highest sampling rates.

In addition to their A/D and D/A converters, all ULTRADMA boards except Model AD8S-100DMA, AD8CH12B-100DMA have TTL inputs (five on the AD12 and ADDA12-100DMA, and one on models ADDA14 and AD14-100DMA) and which can acquire digital data concurrent with the A/D samples, allowing the ULTRADMAS to monitor control signals while analog data is being acquired, and two vectored TTL outputs that can be used in lieu of one D/A channel. Also these boards have three simple programmable TTL outputs.

2.1 MODEL ADDA14-100DMA

Model ADDA14-100DMA has two simultaneously sampling 14-bit A/D converters, two 14-bit D/A converters, one vectored TTL input and two vectored TTL outputs, and is able to transfer data directly into the computer system's memory at 100 MB/s in systems with 64-bit PCI slots or 80 MB/s in 32-bit slots. Two A/D channels may be simultaneously sampled at up to 25 MS/s each, or two D/A channels may be simultaneously converted at up to 3-15 MS/s, depending on system bus throughput. Sampling is controlled by an automatic timebase that is software settable in increments of 20ns. Conversion intervals of 20ns (25Ms/s on two simultaneous channels or 50Ms/s on 1 channel), 40ns, 60ns,, up to 20.46 μ s/sample, are software selectable. Alternatively, sampling may be controlled by an external clock.

2.2 MODEL AD14-100DMA

Model AD14-100DMA is similar to the ADDA14-100DMA, but does not have D/A converters. It includes dual, simultaneously sampled 14-bit A/D converters that can each sample at up to 25 Megasamples per second into the host system's RAM.

2.3 MODEL ADDA12-100DMA

Model ADDA12-100DMA has two simultaneously sampling 12-bit A/D converters, two 14-bit D/A converters, five vectored TTL inputs and two vectored TTL outputs, and is able to transfer data directly into the computer system's memory at 100 MB/s in systems with 64-bit PCI slots or 80 MB/s in 32-bit slots. Two A/D channels may be simultaneously sampled at up to 25 MS/s each, or two D/A channels may be simultaneously converted at up to 3-15 MS/s, depending on system bus throughput. Sampling is controlled by an automatic timebase that is software settable in increments of 20ns. Conversion intervals of 20ns (25Ms/s on two simultaneous channels or 50Ms/s on 1 channel), 40ns, 60ns,, up to 20.46 μ s/sample, are software selectable. Alternatively, sampling may be controlled by an external clock.

2.4 MODEL AD12-100DMA

Model AD12-100DMA is similar to the ADDA12-100DMA, but does not have D/A converters. It includes dual, simultaneously sampled 12-bit A/D converters that can each sample at up to 25 Megasamples per second (aggregate rate of 50Ms/s) into the host system's RAM.

2.5 MODEL AD8-100DMA, AD8S-100DMA and AD8S-100DMA-FPDP

Model AD8-100DMA is similar to the AD12-100DMA, but is a dual eight-bit A/D model that can acquire up to 50 Ms/s each (100Ms/s on a single channel) into host system RAM. The AD8-100DMA accepts either an internal or external clock, but does not have TTL I/O, nor an EVENT input. Sampling is controlled by a high-speed timebase that is software settable in increments of *20ns*. Hence conversion intervals of *20ns* (*50Ms/s on two simultaneous channels or 100Ms/s on 1 A/D channel*), *40ns*, *60ns*, *80ns*, ..., up to *20.46 μs/sample*, are software selectable. The model AD8S-100DMA is similar to AD8-100DMA, but has only a single channel with up to 100Ms/s operation, and has virtually no 50 MHz spur even when sampling at 100MS/s. The AD8S-100DMA-FPDP is an AD8S-100DMA with an added FPDP™ (Transmitter) Interface.

2.6 MODEL AD8CH12B-100DMA

Model AD8CH12B-100DMA can simultaneously acquire eight channels of 12-bit data at up to 6.25 MS/s per channel, or eight channels of 8-bit data at up to 12.5 MS/s per channel.

3. Specifications

3.1 A/D Converters

Number of Input Channels: 2 simultaneously sampled, or single channel
(8 simult. channels for AD8CH12B-100DMA)

A/D converter resolution:

ADDA14-100DMA and AD14-100DMA	14 Bits
AD12-100DMA and AD8CH12B-100DMA	12 Bits
AD8-100DMA	8 Bits

Signal-to-noise Ratio for ADDA14-100DMA and AD14-100DMA:

Dual Channel mode (2x14b@25Ms/s):	76 dB min
Single Channel mode (14b @50Ms/s):	61 dB min

Signal-to-noise Ratio for ADDA12-100DMA and AD12-100DMA:

Dual Channel mode (2x12b@25Ms/s):	68 dB min
Single Channel mode (12b @50Ms/s):	58 dB min

Signal-to-noise Ratio for AD8-100DMA and AD8S-100DMA:

Dual Channel mode (2x8b@25Ms/s):	42 dB min (AD8-100DMA only)
Single Channel mode (8b @50Ms/s):	38 dB for AD8-100DMA, 45 dB for AD8S-100DMA and AD8S-100DMA-FPDP)

(Note: Single channel (2x speed) mode on AD8-100DMA has identical SNR to dual channel mode, except for spur at $F_s/2$. AD8S-100DMA has no such spur)

Signal-to-noise Ratio (AD8CH12B-100DMA only):

12-Bit mode:	66 dB min
8-Bit mode:	45 dB min

Analog input range:	-350mV to +350mV
Input impedance:	50 ohms 10pF
Input connectors:	
All except AD8CH12B-100DMA:	Two SMA connectors.
AD8CH12B-100DMA only:	2M cable assy with 8 SMA male conn.

Maximum Continuous A/D Sampling Rate into host system RAM (host system dependent):
Typical systems with 64-bit PCI bus (e.g. Sun Ultra60, 80, E220, 250, 420 or E450, or Dell Precision 670 workstation under RedHat Linux EL 4.0 64-bit or Windows XP):

AD12,AD14, ADDA12 and ADDA14:	25 Million dual-12/14-bit samples/sec
AD8-100DMA:	50 Million Dual Chan. 8-bit or 100 Million single-channel 8-bit samples
AD8CH12B-100DMA:	6.25 Million simult. 12-bit samples or 12.5 Million 8-bit samples on all 8 channels

Typical Linux or Windows XP machine with 32-bit PCI bus:

AD12,AD14, ADDA12 or ADDA14:	16.6 Million dual-14 or 12-bit samples/sec
AD8-100DMA:	36 Million Dual Chan. 8-bit or 72 Million single-channel 8-bit samples
AD8CH12B-100DMA:	5 Million simultaneous 12-bit samples or 10 Million 8-bit samples on all 8 channels

Continuous storage to disk: 4 to 50 (2x25) MSamples/sec. (16-100 MB/s), depending on disk system throughput. Sampling to disk at >50MB/sec usually requires system with hardware RAID controller and 4 drives.

Sample-to-sample period: 10 ns (100 MHz – AD8-100DMA only) and 20 ns (50 MHz) to 20.46µs in 20 ns increments (160 ns to 4080 ns in 12-bit mode or 80 ns to 2040 ns in 8-bit mode on AD8CH12B-100DMA)

3.2 D/A Converters (Models ADDA14, ADDA12 and DA14-100DMA)

Number of Output Channels: Two, simultaneously updated

D/A resolution: 14 Bits

Signal-to-noise Ratio: 78 dB min. at all update rates

Maximum Continuous Conversion Rate from host system RAM (host system dependent):
 In typical SUN systems with 64-bit PCI bus: 15 Million dual-14-bit samples/sec
 In Dell Precision670 with 64-bit PCI bus: 2 Million dual-14-bit samples/sec
 In Dell PowerEdge™ 1650 with 64-bit PCIbus: 6.25 Million dual 14-bit samples/sec

Continuous playback from disk: 1 to 15 Megasamples/sec. (4-60MB/sec), depending on disk system throughput

Output voltage range: -2V to +2V into open circuit
 -1V to +1V nominal into 50 ohms.

3.3 Fast Vektored TTL Inputs (Models ADDA14, ADDA12, AD14 and AD12)

Number of TTL Input lines:
 ADDA12-100DMA and AD12-100DMA: 5, simultaneously sampled at A/D sample rate
 ADDA14-100DMA and AD14-100DMA: 1, Sampled at A/D sample rate

Input threshold: Standard TTL ($V_{il} < 0.8V$, $V_{ih} > 2.0V$)

3.4 Fast Vektored TTL Outputs (ADDA14, ADDA12, AD14, AD12-100DMA)

Number of TTL Output lines: 2, simultaneously sampled at D/A sample rate, If used, allows only one D/A channel to operate

3.5 Low Speed Software-Programmed TTL Outputs (Models ADDA14, AD14, ADDA12, AD12 and AD8CH12B-100DMA)

Number of TTL Output lines: 3, CPU-writeable output bits, updated independently of A/D, D/A or other TTL lines. (only 1 bit on AD8CH12B-100DMA)

3.6 Front Panel Data Port (FPDP) Interface (AD8S-100DMA-FPDP only)

FPDP Spec level compliance:	VITA 17-199x Rev 1.7, November 24, 1998 Does not support SUSPEND* operation.
Class of FPDP connection:	FPDP/TM (Transmitter Master), with standard Non-Inverted pin-out.
Data Framing	Single Frame Data, Externally triggered.

3.7 General

Operating Temperature Range:	0 to +55 Degrees Celsius
Storage Temperature Range:	-25 to +85 Degrees Celsius

Power Requirements

AD8, AD8S, AD14 and AD12-100DMA:	+5V +/-5% at 2.6A Maximum
ADDA14, ADDA12, AD8CH12B:	+5V +/-5% at 3.3A Maximum

3.8 Physical

ULTRADMA boards are half size 64-bit PCI bus boards, which will operate in either 64-bit or 32-bit PCI systems with either 5V or 3.3V signaling environment. They will function at 33MHz in either 33MHz or 66MHz systems, but if installed in a 66MHz bus, will cause the slot's bus to revert to 33MHz operation. For this reason, the ULTRADMA **should be plugged into 33MHz/5V slots, and not 66MHz or 3.3V slots, unless 5V/33MHz slots are unavailable.** Also, to ensure maximum sampling rates under heavy system loading, only one board should be installed per system, for best results. However, **if installing two ULTRADMA boards in a single system, the first should be installed in a 33MHz slot, and the second in a 66MHz slot,** as 33MHz and 66MHz slots are generally separate PCI buses in the system, allowing higher total DMA throughput from the two ULTRADMA boards. The figure below shows the locations of the analog SMA and digital I/O connectors, and LED indicators.

To avoid overheating, all ULTRADMA boards **must be installed either in well-cooled workstation or server chassis, preferably with 64-bit slots, or alternatively installed in an industrial chassis PC or well-cooled server chassis.** Installation in a standard desktop PC chassis without fans at the front end of the PCI card cage will cause the ULTRADMA to overheat, and such resulting damage from overheating will not be covered by the warranty.

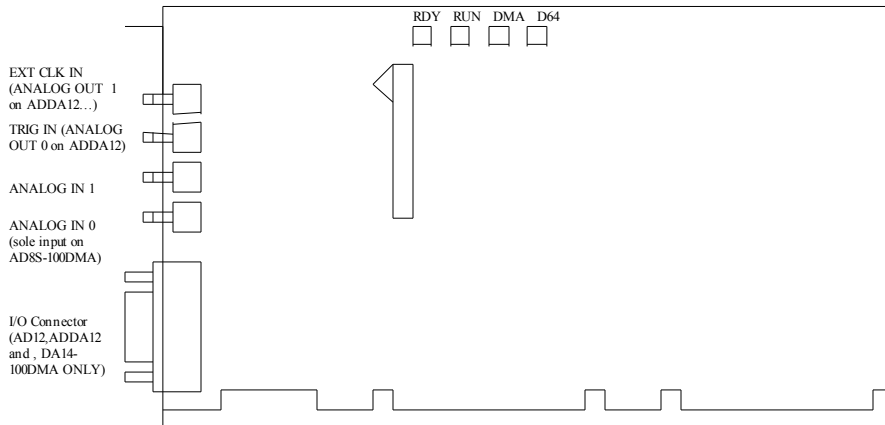


Figure 1. Board layout for ADDA14-100DMA, AD14-100DMA, ADDA12-100DMA, AD12-100DMA, DA14-100DMA, AD8-100DMA and AD8S-100DMA

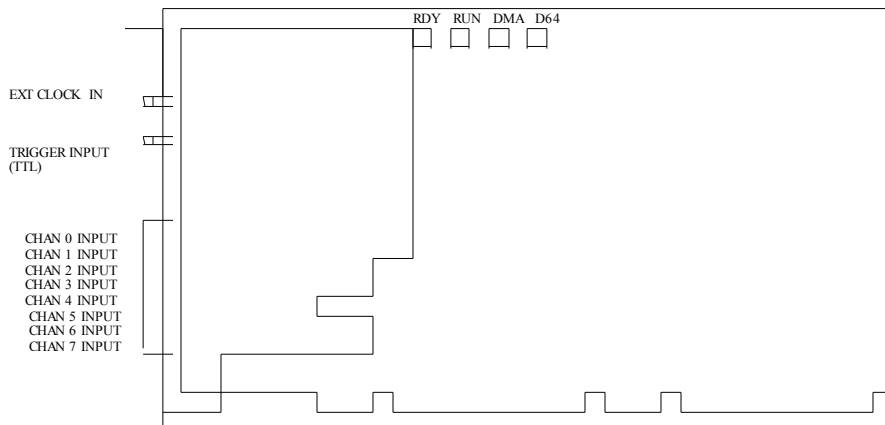


Figure 2. Board layout for AD8CH12B-100DMA

4. Hardware Architecture

ULTRADMA series boards are comprised of a digital section and an analog section. The digital section includes a large elasticity buffer memory (1M x 32 bits) for A/D input data and D/A output data, and five high speed programmable logic devices which implement the bus interface and the fast DMA data transfer engine and PCI master interface with burst cycle capability, and time base.

The analog section contains input amplifiers and A/D converter(s) (models ADDA14-100DMA, AD14-100DMA, ADDA12-100DMA, AD12-100DMA, AD8-100DMA, AD8S-100DMA, AD8S-100DMA-FPDP and AD8CH12B-100DMA) and D/A converter(s) (models ADDA14-100DMA, ADDA12-100DMA and DA14-100DMA).

4.1 Analog Inputs

The two Analog Inputs on models ADDA14-100DMA, AD14-100DMA, ADDA12-100DMA, AD12-100DMA and AD8-100DMA (or single input on AD8S-100DMA, or 8 inputs on AD8CH12B-100DMA) have SMA coaxial jacks and can accept analog data with a voltage range from -350 to +350 millivolts into 50Ω. The data at the two inputs is sampled continuously at up to 25 Ms/s (50Ms/s on each channel or 100Ms/s on a single channel for AD8-100DMA) and stored at the sample storage rate specified using the ULTRADMA Control Register. Furthermore, on models ADDA14 or AD14-100DMA and ADDA12 or AD12-100DMA, sample storage may be started and stopped using the EVENT input. Because the A/D converters sample continuously, the data will be free of transients that occur in boards in which the A/D converter is started and stopped.

On models AD8CH12B-100DMA, a connector assembly consisting of eight SMA male connectors, each on a 2-meter 50-ohm cable terminated in an 8x2 block connector is plugged into the 2x8 input header on the board. These eight A/D channels accept analog data with a voltage range of -350 to +350 millivolts into 50 ohms, and are all concurrently sampled continuously at up to 6.25 MS/s in 12-bit mode or 12.5 MS/s on each channel in 8-bit mode.

4.2 Analog Outputs (Models ADDA14, ADDA12 and DA14-100DMA only)

The two Analog outputs have SMA connectors and provide analog data with a voltage range of -2 to + 2V (-1 to +1V into 50 ohms). The output data rate is the same as the sample storage rate.

4.3 I/O connector (Models ADDA14, AD14, ADDA12, AD12 and DA14-100DMA only)

ULTRADMA boards use a micro-D 26-pin connector to connect optional control signals for the conversion process, as well as TTL I/O lines. An I/O cable and connector is included with all models, which brings in the TTL inputs and outputs and the TRIGGER and EVENT inputs. The pin-out for the 26-pin connector is shown in the table below. The function for each signal is outlined in the following sections. This connector is present but unused in AD8-100DMA boards, and is absent on models AD8S-100DMA and AD8CH12B-100DMA.

Note: EXTREME CARE MUST BE TAKEN TO ENSURE THAT NO VOLTAGES GREATER THAN +/-5V ARE EVER CONNECTED TO ANY ANALOG INPUT, AND NO VOLTAGE OUTSIDE THE 0 TO +5V RANGE IS EVER APPLIED TO ANY OTHER LINE.

Pin	Signal Name	Pin	Signal Name
1	EXT CLK (PECL)	2	/EXT CLK (PECL)
3	OSDAT OUT	4	Digital GND
5	OSCLK OUT	6	Digital GND
7	EVENT IN	8	Digital GND
9		10	Digital GND
11		12	TTL_Out[1]*
13	TTL_Out[0]*	14	TRIGGER IN
15	Digital GND	16	TTL_In[3]
17	OSSTB OUT	18	Digital GND
19	TTL_In[5]**	20	TTL_In[4]**
21		22	Digital GND
23		24	TTL_In[1]**
25	TTL_In[0]**	26	Digital GND

Figure 4.1. Pinout for I/O cable for models ADDA14, AD14, ADDA12, AD12 or DA14-100DMA. *Note that TTL Out bits [1,0] are the two MS bits of D/A channel 1, so the analog output on channel 1 will be affected if these are used as TTL outputs, and conversely, these bits will reflect the MS bits of D/A data when D/A channel 1 is used as an analog channel. **TTL inputs 0, 1, 4 and 5 are not present on AD14 or ADDA14-100DMA boards.

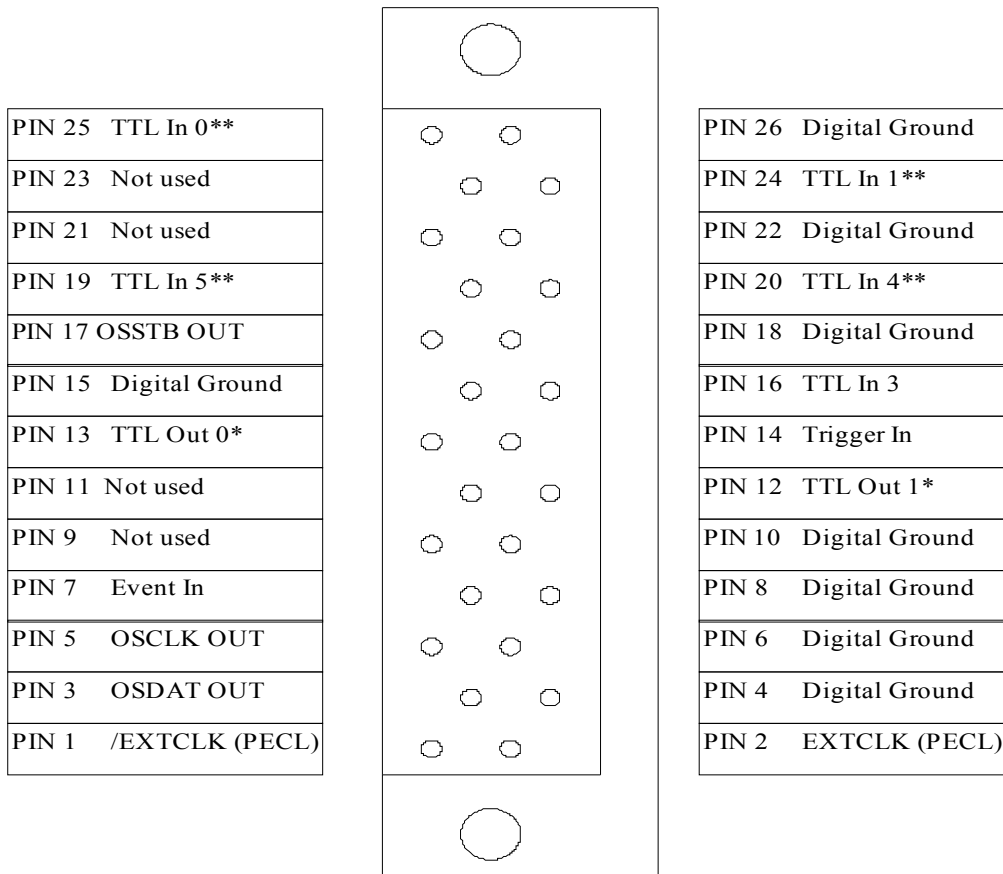


Figure 4.2 Bottom view of mating I/O connector for 26-conductor I/O cable.

4.3.1 Trigger Input Line

The TTL-compatible **Trigger** and **Event** signals permit data acquisition to be controlled by external devices. Upon assertion of **Trigger**, the ULTRADMA will begin acquiring A/D samples, or outputting D/A samples. (See also **Software_Run** bit in ULTRADMA Control Register.)

If left unconnected, the **Trigger** input is automatically held in the asserted state by an on-board pull-up resistor, and sampling will start as soon as the **Software_Run** bit is set in the software.

On Models AD8-100DMA, AD8S-100DMA, AD8S-100DMA-FPDP, AD14-100DMA, AD12-100DMA, and AD8CH12B-100DMA, the Trigger input signal is instead inputted via SMA connector 2 (second connector from the top). On Models **ADDA14-100DMA** and **ADDA12-100DMA**, the trigger may also be input via **SMA connector 2**, if the DA1/TRIG jumper is jumpered for “**TRIG**”. However, in this case, D/A converter DA1 cannot be used.

The trigger line must be driven by a fast-rising TTL signal, such as that from a 74FCT244 driver, or equivalent. To minimize sample uncertainty when using an external clock, the rising edge of trigger must not occur between 2ns before the rising edge of the external clock and 3ns after the rising edge of the external clock.

4.3.2 Event Input Line (ADDA14, ADDA12, AD14 and AD12 only)

The **Event** input allows a condition external to the UVPCI board to start and stop storing data. Once A/D sampling has begun (using the **Trigger** signal and the **Software_Run** bit) it will continue at the specified rate while the **Event** signal is driven high. If **Event** is driven low at any time during the acquisition process, sampling will stop and then a time-stamp will be written into the next memory location. Storage of data will resume when **Event** is brought high again.

If left unconnected (the most common case), the **Event** input is held in the asserted state by an onboard pull-up resistor, and the board will continuously sample (after **Trigger** has been asserted) without interruptions and without writing timestamps. Models AD8-100DMA, AD8S-100DMA, AD8S-100DMA-FPDP, and AD8CH12B-100DMA do not have an Event input, nor timestamp.

4.3.3 TTL_In[7,5,4,1,0]

In addition to the Analog Inputs, all boards, except AD8-100DMA, AD8S-100DMA and AD8CH12B-100DMA have vectored TTL inputs. TTL3's value is stored in Bit 15 and TTL1 and 0's values are stored in bits 29 and 28, respectively of the 32-bit data word, on models ADDA12 and AD12-100DMA. In this same data word, A/D channel 0 uses bits 11-0 and A/D channel 1 uses bits 27-16. In ADDA12 or AD12-100DMA boards with firmware revision 3/3/02 or later, TTL inputs 5 and 4 are also present, and are stored at bit locations 13 and 12 respectively. **In ADDA14 or AD14-100DMA boards, only TTL3 is available as an input, as TTL inputs 0,1,4 and 5 are unavailable** due to their use as the most significant A/D bits.

4.3.4 TTL_Out[1..0] (Shared with a D/A channel on boards with D/As)

In addition to their Analog Inputs, all boards except AD8-100DMA, AD8S-100DMA and AD8CH12B-100DMA have two vectored TTL outputs whose values are specified in bits 29 and 28 of the 32-bit data word. Note that TTL Out bits [1,0] are the two MS bits of D/A channel 1 (on boards such as ADDA12-100DMA that have D/A converters) so the analog output on channel 1

will be affected by the use of these bits as TTL outputs and, conversely, these bits will reflect the MS bits of D/A data when D/A channel 1 is used as an analog channel. For models AD14 or AD12-100DMA that do not contain D/A converters, these bits may be freely used as outputs, and will reflect the value of bits 29 and 28 of any file of 32-bit words that are output using programs such as fast_dma_da_from_disk (Solaris), da (Linux64) or spitout (Windows 2000 or XP).

4.3.5 External Clock (optional)

External clocking is usually not necessary, as the internal clock is more stable and has lower jitter than an external clock. Nevertheless, if an external clock is desired instead of the internal clock, the software must be set up to select external clock mode. On models **AD14-100DMA, AD12-100DMA, AD8-100DMA, AD8S-100DMA, AD8S-100DMA-FPDP and AD8CH12B-100DMA** an external clock may be connected to SMA connector 1 (the top SMA connector). and the DA0/XCLK jumper must be jumpered to the XCLK position. On these models, the external clock must be a **continuous** square or sine wave signal from **5 to 103MHz (65-103MHz for AD8S, and AD8CH12B-100DMA)**, with an amplitude of 200mV to 800mV p-p.

On models **ADDA14-100DMA, ADDA12-100DMA and DA14-100DMA**, an external clock may also be connected to SMA connector 1. However, in this case, as the DA0/XCLK jumper must be jumpered to the XCLK position, D/A converter DA0 may not be used. Alternatively, on these models the lines labeled EXT CLK and /EXT CLK may be used to input an external clock, in differential PECL signal format, using a good differential PECL signal, such as that produced by an MC100E-series component with differential PECL (+3.2/+4.2V) outputs. **Negative ECL voltages (-0.8/-1.8Volts) must NEVER be used, as they may damage the board.**

The external clock frequency must always be between 5 MHz and 103 MHz (70-103MHz for AD8S and AD8CH12B-100DMA). Slower sample rates may be achieved by setting the sample interval to a larger number. For example, on an AD14-100DMA, AD12-100DMA or AD8-100DMA, a sample interval of 0x02 and an external clock frequency of 80MHz will result in a sampling rate of 20 million dual samples per second. On model AD8CH12B-100DMA, however, the sample rate per channel is four times slower per external clock cycle, so that in 12-bit or 14-bit mode, a sample interval of 0x02 and an external clock frequency of 80 MHz will result in a sampling rate of 5 MS/s.

In 8-bit mode on an AD8CH12B-100DMA, a sample interval of 0x01 and an external clock frequency of 100 MHz will result in a sampling rate of 12.5 MS/s. If a sample rate of 10 MS/s is instead needed, the external clock input frequency must be changed to 80 MHz.

In general it is better not to use an external clock, but instead use the trigger input to selectively sample data. Also, it is recommended that applications first be tried using the internal clock, and only when all else is working should the external clock be substituted. Before starting the board sampling, if the user program selects the external clock mode in the user software, the **user program must then check the “No Clock or Slow Clock” bit in the control register, as indicated in the example programs, to be sure that the external clock has been correctly supplied to the board. If no external clock is indicated, by the “No Clock or Slow Clock” bit being read as a “1”, then the board must not be started, or it will hang. Instead, the user program should switch back to the internal clock and exit.**

4.3.6 Programmed TTL outputs (OSSTB, OSCLK, OSDAT)

Models AD14-100DMA, ADDA14-100DMA, AD12-100DMA and ADDA12-100DMA boards are equipped with three general-purpose TTL output pins on the I/O connector for controlling external devices. These data bits simply convey to the three lines OSSTB, OSCLK, and OSDAT the data

that was last written to the control register at bits 18, 17 and 16, respectively. These TTL outputs are completely independent of all other TTL, A/D or D/A operation. As one example, the OSSTB signal can be used as a Frame Sync Bit, OSDAT can be used as a Serial Data Bit, and OSCLK can be used as a Serial Clock to control external devices.

4.4 Front-End Mezzanine Board Interface Connections

Certain ULTRADMA models are equipped with an optional mezzanine connector to allow special purpose I/O devices to be used, such as RF demodulators or other A/D front ends. A 2x30 pin connector (see appendix A) carries 32 data bits and control and power signals for mezzanine connections. One such mezzanine is the 8-channel A/D boardlet used in model AD8CH12B-100DMA, and another mezzanine is the 1-channel 100MSPS FP/AD100 boardlet used in models AD8S-100DMA and AD8SFPDP-100DMA.

4.5 LED Indicators

The four LEDs on the top edge of the ULTRADMA board are useful during system integration for monitoring the ULTRADMA board status. The functions of the LEDs are outlined below.

4.5.1 Run LED

The RUN LED is used to indicate an actual A/D or D/A sample is occurring. For configurations that use an external trigger (supplied via the I/O connector), this LED can be used to indicate a successful trigger. The LED will appear very dim at all but the fastest sample rates.

4.5.2 RDY (Ready) LED

The RDY LED is illuminated when the ULTRADMA board has a clock supplied to it (either external or internal), and has been told via the user program to begin storing A/D conversions.

4.5.3 DMA and D64 (64-Bit DMA Transfer) LED

The DMA LED is illuminated when the ULTRADMA board is currently performing 32 or 64-bit DMA transfers. The D64 LED will be additionally illuminated during 64-bit transfers, which should be the only type of DMA transfers that occur when the board is installed in a 64-bit PCI slot.

5. Low-level Software Interface

The ULTRADMA board is easy to communicate with. In most cases, this section may be skipped, as the drivers supplied with the board automatically handle all communication with the board registers. **The best way to develop your own custom software is simply to modify the appropriate sample programs included with the board, and then recompile.** However, the following section gives an overview of how the driver calls actually control the board.

The software interface consists of a PCI type-00 Configuration Header and three board specific registers - a Control Register and LOW DMA Address Register. Additionally, there is DMA High Start Address Register, only for use in applications in which dual address cycles are required. Each of these groups of registers is outlined in the following sections.

Accesses to CONTROL and DMA START ADDRESS registers must be made as 32-bit transfers.

5.1 PCI Configuration Header

ULTRADMA series boards support a PCI Configuration Header, whose map is shown below.

Double Word Address	byte 3	byte 2	byte 1	byte 0
00 H	Device ID		Vendor ID	
04 H	Status		Command	
08 H	Class Code			Revision ID
0C H		Header Type	Latency Timer	
10 H	Base Address for ULTRADMA data memory			
14 H				
18 H				
1C H				
20 H				
24 H				
28 H				
2C H				
30 H				
34 H				
38 H				
3C H	Max Lat (=01 H)	Min Gnt (=01 H)	Interrupt Pin	Interrupt Line

The board control register is mapped at **configuration space address 80H**, as well as in **memory space at Base Address + 1FFFFC**. The DMA Low Start Address register is mapped at **configuration space address 8CH**, and in **memory space at Base Address + 1FFFF4**. Accessing either place will read or write these register. The optional DMA High Start Address register (which, if used must be written BEFORE the DMA Low Start Address reg.) is at configuration space address 84H and also in memory space at Base Address + 1FFFF0.

Double Word Address	byte 3	byte 2	byte 1	byte 0
80 H	ULTRADMA Control Register			
84 H	DMA High Start Address Register (for dual address cycles only)			
8C H	DMA Low Start Address Register..... WRT			

5.2 ULTRADMA Control Register

The ULTRADMA Control Register is used to configure the board and to start and stop the data acquisition process. The table below shows the usage of the ULTRADMA Control Register, and these bits' functions are outlined in the sections that follow. The first table shows the function of the Control Register during a **write**. During a **read**, **these bits will not be read back, but instead, three of the bit positions may be read**, as shown in the second table. The control register is never directly written or read by user programs, but is modified by calls to the driver, which are each summarized in the discussion of the respective bit.

Bit	Function
30	DMA Blocksize 2
29	DMA Blocksize 1
28	DMA Blocksize 0
27	Software_Run
26	Buffer_Wrap
24	/Interrupt_Enable
21	Double_Speed
20	D/A MODE
19	Internal Clock mode
18	OSSTB
17	OSCLK
16	OSDAT
15	TimeStamp_Test
9..0	Sample_Interval [9..0]

Function of Control Register bits during **write**.

Bit	Function
31	No Clock or Slow Clock
30	Board Interrupting (DMA completion)
29	Board Interrupting (A/D completion)
28	Board Stopped
27	DMA in progress

Function of Control Register bits during **read**.

5.2.1 Software_Run bit (write only)

Software_Run is used to start and stop data acquisition. If no connection is made to the TRIGGER or EVENT inputs, data acquisition begins when Software_Run is set to 1, and ends when it is set to 0. If the TRIGGER signal is used, Software_Run must be set to 1 before TRIGGER can start data acquisition. Acquisition will always stop when Software_Run is set to 0.

The Software_Run bit is set to 1, turning on the board, by using the **Device.RunBoard(TRUE)** driver call under Windows 2000/XP or **the uvpci_set_go()** call under the Linux64 or Solaris OS.

The Software_Run bit is set to 0, stopping the board, by using the **Device.RunBoard(FALSE)** driver call under Windows 2000/XP or the **ultrad_stop()** (or indirectly by the **uvpci_stop_at_n_blocks()**) calls under the Linux64 or Solaris operating system.

5.2.2 Buffer_Wrap bit (write only)

Buffer_Wrap is used to specify if the RAM buffer will be filled with A/D conversion samples a single time (storage stops when memory is filled) or if it will be treated as a cyclic buffer and filled continuously, wrapping around to the start of the buffer after the end is reached.

If **Buffer_Wrap** is set to 0, data acquisition will end when the RAM buffer has been filled once.

When **Buffer_Wrap** is set to 1, A/D data will be stored to RAM continuously. When the buffer is filled, the next A/D sample will be stored in the first RAM data location and the buffer will be overwritten with new data, and the buffer will thus operate as a cyclic elasticity buffer.

The **Buffer_Wrap** bit is automatically set to 1, enabling the wraparound mode, using the **Device.SetBufferWrap(TRUE)** driver call under Windows XP or 2000 or the **uvpci_set_wrap (board_fd)** call under the Linux64 or Solaris operating system.

The **Buffer_Wrap** bit is automatically set to 0, disabling the wraparound mode, using the **Device.SetBufferWrap(FALSE)** driver call under Windows XP or 2000 or the **uvpci_unset_wrap (board_fd)** call under the Linux64 or Solaris operating system.

5.2.3 Interrupt_Enable bit (write only)

The **Interrupt_Enable** bit is used to determine if the ULTRADMA board will issue A/D interrupts as the host system RAM is filled. The ULTRADMA will always use PCI bus interrupt line INTA#. Once acquisition begins and A/D samples are written to the RAM buffer, the ULTRADMA can issue an interrupt as each block of the RAM buffer is filled. When **Interrupt_Enable** is set to 0, interrupts will be generated. When set to 1, interrupts will not be generated.

The **Interrupt_Enable** bit is set to 0, **enabling** interrupts, by the **Device.DisableInterrupts(FALSE)** call under Windows XP or 2000 or by using **uvpci_set_int(board_fd)** under Linux64 or Solaris.

The **Interrupt_Enable** bit is set to 1, **disabling** interrupts, using the **Device.DisableInterrupts(TRUE)** call under Windows XP/2000 or by using **uvpci_unset_int(board_fd)** under Linux64 or Solaris.

5.2.4 Double_Speed bit (write only)

The **Double_Speed** bit is used to allow the AD12/ADDA12-100DMA or AD14/ADDA14-100DMA boards to perform data acquisition at 50 MHz on a single channel (100MHz on a single channel on model AD8-100), but with significantly lower signal-to-noise ratio.

When **Double_Speed** is set to 0, the two A/D converters will perform data conversion simultaneously, **on two separate channels**. This is the normal mode of operation, and the maximum sample rate in this mode is 25 Ms/s on the two A/D channels simultaneously (50 Ms/s on AD8-100DMA). Model AD8S-100DMA is automatically always in 100 MSPS mode.

When **Double_Speed** is set to 1, the two A/D converters will perform sequential data conversions on a **single** channel. This mode is used to perform A/D conversions at 50 MHz (100Ms/s on models AD8-100DMA and AD8S-100DMA), assuming SAMPINT is set to 0. In this mode, only Analog Input 0 is sampled, and Analog Input 1 is ignored. In model AD8S-100DMA, a single converter accomplishes this same 1-channel functionality at all times, without the small spur that might occur at 50MHz in doublespeed mode on a 2-converter AD8-100DMA. **Doublespeed must always be set to 1 when using the AD8S-100DMA.**

The **Double_Speed** bit is set to zero by default, disabling double speed mode, or by using the **set_ultrad_board_register(ultrad_board_handle, ULTRAD_TWO_X,FALSE)** driver call under Windows 2000/XP, or the **uvpci_unset_double_speed(board_fd)** call under Linux64 or Solaris.

The **Double_Speed** bit is set to one, enabling double speed mode, by using the **set_ultrad_board_register(ultrad_board_handle, ULTRAD_TWO_X,TRUE)** driver call under Windows 2000/XP or the **uvpci_set_double_speed(board_fd)** call under Linux64 or Solaris.

IMPORTANT: When altering the doublespeed bit, add a sleep(1) statement or other delay, to allow at least 1mS for an on-board relay to settle before starting the board running.

5.2.5 D/A Mode bit (write only)

D/A Mode is used to select D/A operation instead of A/D operation. When set to 0, the ULTRADMA performs A/D sampling. When set to 1, the ULTRADMA performs D/A conversions.

The D/A Mode bit is set to 1, enabling D/A operation, using the driver call **Device.SetADDA (TRUE)** driver call under Windows XP or 2000 or the **uvpci_set_adda_mode(board_fd)** call under the Solaris operating system or the **uvpci_set_da_mode(board_fd)** call under Linux64.

The D/A Mode bit is set to 0, selecting A/D (and not D/A) operation, using the **Device.SetADDA (FALSE)** driver call under Windows XP or 2000 or the **uvpci_unset_adda_mode(board_fd)** call under the Solaris operating system, or the **uvpci_unset_adda_mode(board_fd)** call under Linux64.

5.2.6 Internal Clock Mode bit (write only)

Internal Clock Mode is used to cause the board to sample using its internal time base, instead of an external clock. When set to 1 (the default mode), conversions use the internal time-base.

When set to 0, the ULTRADMA will instead use an external clock. **NEVER set this bit to 0 (external clock) unless you are sure a CONTINUOUS clock, with correct signal levels, is connected to the external clock input(s).** Otherwise the system may hang.

The Internal Clock Mode bit is automatically set to 1, enabling internal clock operation, using the **setup_ultrad_io(INTERNAL_CLOCK,..)** driver call under Windows 2000 or the **uvpci_unset_ext_clock(board_fd)** call under the Linux64 or Solaris operating system.

The Internal Clock Mode bit is set to 0, selecting external clock operation, using the **setup_ultrad_io(ULTRAD_EXTERNAL_CLOCK,..)** driver call under Windows 2000 or the **uvpci_set_ext_clock(board_fd)** call under the Linux64 or Solaris operating system.

5.2.7 OSSTB bit (write only)

The OSSTB, OSDAT and OSCLK bits may be used together as a TTL serial communication path for control of devices connected to the ULTRADMA board. These outputs are provided on the 26-pin connector on models ADDA14, ADDA14, AD12, ADDA12 and AD8-100DMA only.

OSSTB is a registered control bit connected to the OSSTB pin on the I/O connector. It can be

used as a Frame Sync bit for control of devices connected to the ULTRADMA board or, alternatively as a general write-only TTL output bit. **On the model AD8CH12B-100DMA, this bit is used to select 8-bit mode (if set to 0) or higher resolution (12-bit/14-bit) mode (if set to 1).**

The OSSTB bit is automatically set to 1, driving the OSSTB output to a TTL logic 1, using the **set_ultrad_board_register(ultrad_board_handle,ULTRAD_RSSTB,TRUE)** driver call under Windows 2000 or the **uvpci_set_serial(...)** call under Linux64 or Solaris operating systems.

The OSSTB bit is automatically set to 0, driving the OSSTB output to a TTL logic 0, using the **set_ultrad_board_register(ultrad_board_handle, ULTRAD_RSSTB,FALSE)** driver call under Windows 2000 or the **uvpci_set_serial(...)** call under Linux64 or Solaris operating system.

5.2.8 OSCLK bit (write only)

OSCLK is a registered control bit connected to the OSCLK pin on the I/O connector, on models AD8, ADDA14, AD14, AD12 and ADDA12-100DMA only. It can be used as a Data Clock for control of devices connected to the ULTRADMA, or as a general write-only TTL output. **On model AD8CH12B-100DMA, this bit selects 8-channel mode (if set to 0) or 4-channel mode (if set to 1). On model AD8S-100DMA-FPDP, this bit is also output on FPDP line PIO2.**

The OSCLK bit is automatically set to 1, driving the OSCLK output to a TTL logic 1, using the **set_ultrad_board_register(ultrad_board_handle,ULTRAD_RSCLK,TRUE)** driver call under Windows 2000 or the **uvpci_set_serial(...)** call under Linux64 or Solaris operating system.

The OSCLK bit is automatically set to 0, driving the OSCLK output to a TTL logic 0, using the **set_ultrad_board_register(ultrad_board_handle, ULTRAD_RSCLK,FALSE)** driver call under Windows 2000 or the **uvpci_set_serial(...)** call under Linux64 or Solaris operating system.

5.2.9 OSDAT bit (write only)

OSDAT is a registered control bit connected to the OSDAT pin on the I/O connector, on models AD8, ADDA14, AD14, AD12 and ADDA12-100DMA. It can be used as a Serial Data bit for control of devices connected to the ULTRADMA board, or alternatively can be used as a general TTL output bit. **On model AD8SFPDP-100DMA, this bit is additionally output on FPDP line PIO1.**

The OSDAT bit is automatically set to 1, driving the OSDAT output to a TTL logic 1, using the **set_ultrad_board_register(ultrad_board_handle,ULTRAD_RSDAT,TRUE)** driver call under Windows 2000 or the **uvpci_set_serial(...)** call under Linux64 or Solaris operating systems.

The OSDAT bit is automatically set to 0, driving the OSDAT output to a TTL logic 0, using the **set_ultrad_board_register(ultrad_board_handle, ULTRAD_RSDAT,FALSE)** driver call under Windows 2000 or the **uvpci_set_serial(...)** call under Linux64 or Solaris operating systems

5.2.10 TimeStamp_Test bit (write only)

This bit is a Built-In-Self-Test for factory use during testing. Users should not set this bit to 1.

5.2.11 Sample Interval bits (write only) for model AD8-100DMA only

The **Sample_Interval** word controls the A/D sampling rate. The value written to **Sample_Interval [9..0]** specifies the number of 20ns periods between samples. A partial table of **Sample_Interval** values and conversion periods and frequencies is below. Note that sampling rates slower than 48.876 kHz may be realized by setting the board to a slow rate and skipping samples when reading the A/D data from RAM. For example, a 10KS/s sample rate may be attained by setting the board for 200 KS/s sampling (Sample_interval = 250), and incrementing the pointer by 20 each time an A/D sample is read from RAM.

Sample_Interval[7..0] (decimal)	Conversion Period	Conversion Frequency
1 (with Double_Speed)	10.0 ns	100.0000 MHz
1	20.0 ns	50.0000 MHz
2	40.0 ns	25.0000 MHz
3	60.0 ns	16.6666 MHz
...
1022	20.44 us	48.924 kHz
1023*	20.46 us	48.876 kHz

To determine the sampling period more generally, use the following equation.

$$\text{Frequency} = 1 / (\text{Sample_Interval}) \times 20 \text{ ns}$$

The Sample interval is set using the **Device.SetConversionPeriod(SampleInterval)** driver call under Windows XP or 2000 or the **uvpci_set_sampint(...)** call under the Linux64 or Solaris OS.

5.2.12 Sample Interval bits (write only) for model AD8S-100DMA and AD8S-100DMA-FPDP only

The **Sample_Interval** word determines the A/D conversion rate. The AD8S-100DMA has a fixed sampling rate of 100 MSPS, or 1 x the external clock frequency if an external clock is instead used. **The sample interval must be set to 1 and doublespeed must be set active.**

Sample_Interval[7..0] (decimal)	Conversion Period	Conversion Frequency
1 (with Double_Speed)	10.0 ns	100.0000 MHz
2-1023*	DO NOT USE	DO NOT USE

*The sample interval is settable from 2 to 1023 for ADDA14-100DMA, AD14-100DMA, ADDA12-100DMA, AD12-100DMA and AD8-100DMA boards of firmware revision 7/17/00 or later.

5.2.13 Sample Interval bits (write only) for ADDA14-100DMA, AD14-100DMA, ADDA12-100DMA and AD12-100DMA

The **Sample_Interval** word specifies the A/D and D/A conversion rate. The value written specifies the number of 20ns periods between samples. A partial table of **Sample_Interval** values and conversion periods and frequencies is below. Note that sampling rates slower than 48.876 kHz may be realized by setting the board to a slow sampling rate, and then skipping alternate samples when reading the A/D data from RAM (and duplicating the same sample several times in RAM when writing data for output via the D/As). For example, an 8KS/s sample rate may be attained by setting the board for 200 KS/s sampling (**Sample_Interval** = 250), and incrementing the pointer by 25 each time an A/D sample is read from RAM.

Sample_Interval[7..0] (decimal)	Conversion Period	Conversion Frequency
1	DO NOT USE	DO NOT USE
2 (with Double_Speed)	20.0 ns	50.0000 MHz
2	40.0 ns	25.0000 MHz
3	60.0 ns	16.6666 MHz
4	80.0 ns	12.5000 MHz
...
1022	20.44 us	48.924 kHz
1023*	20.46 us	48.876 kHz

To determine the sampling period more generally, use the following equation.

$$\text{Frequency} = 1 / (\text{Sample_Interval}) \times 20 \text{ ns}$$

The Sample interval is set using the **Device.SetConversionPeriod(SampleInterval)** driver call under Windows XP or 2000 or the **uvpci_set_sampint(board_fd)** call under the Solaris operating system or the **uvpci_set_sample_interval(board_fd)** call under Linux64.

*The sample interval is settable from 2 to 1023 for ADDA12/14 and AD12/14-100DMA and AD8-100DMA boards of firmware revision 7/17/00 or later, running software release 2.1 (2/8/01) or later. For earlier firmware or software revisions, the maximum sample interval is 255, not 1023.

5.2.14 Sample Interval bits (write-only) for AD8CH12B-100DMA in 8-bit mode

The value written to **Sample_Interval[7..0]** specifies the number of 80ns periods between samples. A partial table of **Sample_Interval** values and conversion periods and frequencies is below. Sampling rates below 49.019 KS/s may be realized by setting the board to a slow rate and then skipping alternate samples when reading the A/D data from RAM. For example, a 10KS/s sample rate may be attained by setting the board for 50 KS/s sampling (**Sample_Interval** = 250), and incrementing the pointer by 5 each time an A/D sample is read from RAM.

Sample_Interval[7..0] (decimal)	Conversion Period	Conversion Frequency
1	80.0 ns	12.5000 MHz
2	160.0 ns	6.2500 MHz
3	240.0 ns	4.26666 MHz
...
254	20.320 us	49.212 kHz
255	20.400 us	49.020 kHz

To determine the sampling period more generally, use the following equation.

$$\text{Frequency} = 1 / (\text{Sample_Interval}) \times 80 \text{ ns}$$

The Sample interval is set using the **Device.SetConversionPeriod(SampleInterval)** driver call under Windows XP or 2000 or the **uvpci_set_sampint(...)** call under the Linux64 or Solaris OS.

5.2.15 Sample Interval bits for AD8CH12B-100DMA in 12-bit mode

The value written to **Sample_Interval[7..0]** specifies the number of 160ns periods between samples. A partial table of **Sample_Interval** values and conversion periods and frequencies is below. Sampling rates slower than 49.020 KS/s may be realized by setting the board to one of the lowest sampling rates, and then skipping alternate samples when reading the A/D data from RAM. For example, a 10KS/s rate may be attained by setting the board for 50 KS/s sampling, and incrementing the pointer by 5 each time a group of 8 channel samples is read from RAM.

Sample_Interval[7..0] (decimal)	Conversion Period	Conversion Frequency
1	DO NOT USE	DO NOT USE
2	160.0 ns	6.25 MHz
3	240.0 ns	4.16666 MHz
...
254	20.320 us	49.213 kHz
255	20.400 us	49.020 kHz

To determine the sampling period more generally, use the following equation.

$$\text{Frequency} = 1 / (\text{Sample_Interval}) \times 80 \text{ ns}$$

The Sample interval is set using the **Device.SetConversionPeriod(SampleInterval)** driver call under Windows XP or 2000 or the **uvpci_set_sampint(...)** call under the Linux64 or Solaris OS.

5.2.16 DMA Block Size bits (write only)

DMABS[2..0] specify the length of the DMA burst that will be started upon a write to the DMA Starting Address Register. Burst lengths of 4KB to 128KB and 512KB are supported, to facilitate operation in host systems with Scatter/Gather requirements, and varying MMU page sizes. At the end of each such burst, the ULTRADMA issues an interrupt on PCI signal INTA#. The following table shows all values of **DMABS[2..0]** and the corresponding **DMA Burst Size**.

DMABS[2,1,0]	DMA Burst Length (Bytes)
0,0,0	NO DMA
0,0,1	4,096
0,1,0	8,192
0,1,1	16,384
1,0,0	32,768
1,0,1	65,536
1,1,0	131,072
1,1,1	528,244

5.2.17 Board Interrupt after A/D or D/A block completed (read only status bit)

This bit (bit 29), when 1, indicates that the ULTRADMA has issued an interrupt on PCI bus line INTA# that is pending, signifying that the ULTRADMA has transferred 512Kbytes of A/D or D/A conversion data to/from its 4 MB on-board RAM. This bit, and the associated interrupt, are automatically cleared by any write to the ULTRADMA Control Register. If you want to clear the A/D interrupt without disturbing any ongoing operation, write the same data to the ULTRADMA Control Register that had last been written to it. Bit 29 should be checked every time the interrupt service routine for this driver is entered, to be sure that it was the ULTRADMA that actually issued the interrupt presently being serviced, and not an interrupt from another board, nor the alternate (DMA done) interrupt from the ULTRADMA board. Note that this interrupt does not in any way signify that data has been transferred from the on-board RAM to host system RAM. Completion of such DMA bursts is signified by the DMA completion interrupt described directly below

5.2.18 Board Interrupt after DMA block completed (read only status bit)

This bit (bit 30), when 1, indicates that the ULTRADMA has issued an interrupt on PCI bus line INTA# that is pending, signifying that it has transferred 4K to 512K (depending on the value written earlier to bits **DMABS[2,1,0]** of the Control Register) to/from host RAM. This bit may be cleared by writing binary xxxx...xx10 (any word in which data bit D1=1) to the board's **DMA Low Starting Address Register**. Bit 30 should be checked every time the driver's interrupt service routine is entered, to determine if it was the ULTRADMA DMA-block-completed flag that actually issued the interrupt presently being serviced, and not some other board, nor the ULTRADMA's alternate (A/D) block completion. Note that bit 30 does not signify that data has been transferred between the ULTRADMA's A/D and D/A converters and its local 4MB RAM. Completion of such converter bursts is signified by the A/D / D/A block completion interrupt (bit 29) described earlier.

5.2.19 Board Stopped (read only status bit)

This bit (bit 28), indicates whether the ULTRADMA has stopped acquiring or outputting data to/from its on-board dual-ported RAM, in the case where the Wrap bit is not set, and the board

has finished an acquiring an entire board full of data. This bit is a 1 in this case, and is a 0 in all other cases, including when the board is stopped due to the Run bit being set to 0. Its function is superfluous, as the driver will alternately know that the board has finished acquiring a board full of data by the fact that eight interrupts have been received from the board. Note that this bit does not in any way signify that data has been transferred from the on-board RAM to host system memory. Completion of such DMA bursts is signified by the DMA completion interrupt described directly below. Therefore, the driver must not shut down the board merely because the Board Stopped bit is 1, but must also wait for all DMA bursts to be completed, to ensure that data has made the full journey between the converters and the host system memory.

5.2.20 DMA in progress (read only status bit)

This bit (bit 27), if 1 indicates that the ULTRADMA is still performing a DMA burst to or from system RAM. Its function is superfluous, as the driver should know that the board has finished a DMA transfer by the fact that a DMA interrupt has been received from the board and the driver has not asked for a new DMA burst to begin. It should be used as a "sanity check" only.

5.2.21 No Clock or Slow Clock (read only status bit)

This bit (bit 31), if 1 indicates that the ULTRADMA is not receiving a clock, or is receiving one that is too slow for correct operation. This bit should be checked by any user program before starting the board, **if the external clock mode has been selected**. To use this bit, first select external clock (`uvpci_set_ext_clock(board_fd)`), and then wait a few milliseconds and read this bit, as indicated in the Solaris example program `fast_acquire_dma_data.c`. If the "No Clock or Slow Clock" condition is indicated by this bit being high, **do not start the board**, or the program will hang. Instead switch back to the internal clock and exit the program.

5.3 DMA Low Starting Address Register and Read/Write control

The ULTRADMA Low Starting Address Register is used to start a DMA burst, in which the ULTRADMA board directly transfers samples into host system RAM. This register is never directly written nor read by user programs, but is modified by calls to the driver. Bits 31 through 2 specify DMA address bits 31 through 2 (Bit 1 is always zero for longword addresses). Bit 0 is also always zero for longword addresses, and is therefore available for specifying whether the DMA transfer is a DMA Write (Bit 0 = 1) or a DMA Read (Bit 0 = 0). For A/D operation, the driver would specify a DMA Write (writing A/D data to host system RAM), and for D/A operation, the driver would specify a DMA Read (reading data from host system RAM and outputting it via the D/As). For example, to tell the board to write a 16KB burst of data starting at PCI address 0x8456e728, we would write the number 0x8456e729 (which is 0x8456e729 | 1). The 16K burst length would have had to have been specified earlier by writing 011 to bits DMABS[2..0].

When the ULTRADMA board has completed writing the burst of data to host RAM, it will issue an interrupt on PCI line INTA#. The fact that this interrupt was issued by the ULTRADMA, and not another board, can be determined by reading bit 30 of the control register, as discussed above.

5.4 DMA High Starting Address Register (For extended addressing only)

The ULTRADMA High Starting Address Register specifies the upper 11 bits in optional 64-bit addressing mode. This register, if used, must be written just before the DMA Low Starting Address Register (described above) is written. Bits 31-0 specify DMA address bits 63-32, although only the first 11 bits are actually used internally, giving the board an address range of 2^{43} bytes (8 TB). This register must not be written to for standard 32-bit addressing mode.

5.5 Data Representation in Host System Memory During D/A Transfers (Models ADDA14-100DMA, ADDA12-100DMA and DA14-100DMA only)

The ULTRADMA transfers D/A data from host system memory, via DMA. In this way, very long strings of D/A data may be outputted, limited only by the size of the computer system's user-available RAM. The format for data storage in host system RAM is shown below. Each double longword of RAM contains two D/A values, each in **offset binary format** and the associated TTL values, which are simultaneously output to the two D/A converters. **The format below is also identical to the format in which D/A data must be stored on disk in the example programs in all Ultraview-supplied software packages. Note that bits D29 and D28 also are output to the two TTL outputs TTL_Out[1] and TTL_Out[0] respectively, and if these outputs are instead driven as part of D/A data for Channel 1, care must be taken by any TTL equipment, so as to ignore the data on these bits. Similarly, if these bits are to be used as TTL outputs, the D/A data on Channel 1 will reflect this usage, and analog equipment fed by D/A channel 1 must ignore any erroneous excursions on the analog output on D/A channel 1 due to the use of its MS D/A data bits as TTL data.**

Address	D31	D30	D29 - D16: D/A Data for Chan 0	D15, D14	D13 - D0: D/A Data for Chan 1
BA+\$0000000	x	x	Chan 0 Samp 0	x	Chan 1 Samp 0
BA+\$0000004	x	x	Chan 0 Samp 1	x	Chan 1 Samp 1
BA+\$0000008	x	x	Chan 0 Samp 2	x	Chan 1 Samp 2
BA+\$000000C	x	x	Chan 0 Samp 3	x	Chan 1 Samp 3
BA+\$0000010	x	x	Chan 0 Samp 4	x	Chan 1 Samp 4
BA+\$0000014	x	x	Chan 0 Samp 5	x	Chan 1 Samp 5
BA+\$0000018	x	x	Chan 0 Samp 6	x	Chan 1 Samp 6
BA+\$000001C	x	x	Chan 0 Samp 7	x	Chan 1 Samp 7
.	x	x	.		
.	x	x			
End of array					

5.6 Data Representation in Host System Memory During A/D Transfers (Models ADDA14-100DMA and AD14-100DMA only)

The ULTRADMA transfers data directly to host system memory, allowing very long strings of A/D data to be acquired, limited only by the size of user-available host RAM. Data is stored in RAM as shown below. For models AD14 or ADDA14 each RAM longword contains two simultaneously acquired A/D values, in **offset binary format**, and TTL values. **Data is also stored to disk this way in the example programs for models AD14-100DMA and ADDA14-100DMA.**

Address	D31 ,30	D29 - D16: A/D Chan 0 Data	D15	D13 - D0: A/D Chan 1 Data
BA+\$0000000	-	Chan 0 Samp 0	TTL In 3	Chan 1 Samp 0
BA+\$0000004	-	Chan 0 Samp 1	TTL In 3	Chan 1 Samp 1
BA+\$0000008	-	Chan 0 Samp 2	TTL In 3	Chan 1 Samp 2
BA+\$000000C	-	Chan 0 Samp 3	TTL In 3	Chan 1 Samp 3
BA+\$0000010	-	Chan 0 Samp 4	TTL In 3	Chan 1 Samp 4
BA+\$0000014	-	Chan 0 Samp 5	TTL In 3	Chan 1 Samp 5
BA+\$0000018	-	Chan 0 Samp 6	TTL In 3	Chan 1 Samp 6
BA+\$000001C	-	Chan 0 Samp 7	TTL In 3	Chan 1 Samp 7
.
End of array				

NOTE: In 64-bit x86-based Linux systems, the byte ordering is reversed, and user software must reconstruct the samples before using them, by shifting the original 8 MS bits in each 16-bit word right by eight bits, and then adding 256 x the original 8 LS bits to this value.

5.7 Data Representation in Host System Memory During A/D Transfers (Models ADDA12-100DMA and AD12-100DMA only)

The ULTRADMA transfers data directly to host system memory, allowing very long strings of A/D data to be acquired, limited only by the size of user-available host system RAM. Data is stored in host RAM as shown below. For models AD12- or ADDA12-100DMA, each longword of RAM contains two simultaneously acquired A/D values, in **offset binary format**, and TTL values. **The format below is also the way in which data is stored to disk in the example programs for models AD12-100DMA and ADDA12-100DMA.**

Address	D31 ,30	D29,D28	D27 - D16: A/D Chan 0 Data	D15	D13,D12	D11 - D0: A/D Chan 1 Data
BA+\$0000000	-	TTL In 1,0	Chan 0 Samp 0	TTL In 3	TTL In 5,4	Chan 1 Samp 0
BA+\$0000004	-	TTL In 1,0	Chan 0 Samp 1	TTL In 3	TTL In 5,4	Chan 1 Samp 1
BA+\$0000008	-	TTL In 1,0	Chan 0 Samp 2	TTL In 3	TTL In 5,4	Chan 1 Samp 2
BA+\$000000C	-	TTL In 1,0	Chan 0 Samp 3	TTL In 3	TTL In 5,4	Chan 1 Samp 3
BA+\$0000010	-	TTL In 1,0	Chan 0 Samp 4	TTL In 3	TTL In 5,4	Chan 1 Samp 4
BA+\$0000014	-	TTL In 1,0	Chan 0 Samp 5	TTL In 3	TTL In 5,4	Chan 1 Samp 5
BA+\$0000018	-	TTL In 1,0	Chan 0 Samp 6	TTL In 3	TTL In 5,4	Chan 1 Samp 6
BA+\$000001C	-	TTL In 1,0	Chan 0 Samp 7	TTL In 3	TTL In 5,4	Chan 1 Samp 7
.
End of array						

In 64-bit x86-based Linux systems, byte ordering is reversed, and user software must reconstruct the samples before using them, by shifting the original 8 MS bits in each 16-bit word right by eight bits, and then add 256 times the original 8 LS bits to this value.

5.8 Data Representation in Host System Memory During A/D Transfers (Model AD8-100DMA, AD8S-100DMA and AD8S-100DMA-FPDP only)

For models **AD8-100DMA**, each longword contains four 8-bit samples, **each in offset binary format**. In **2-channel mode (AD8-100DMA)**, two pairs of simultaneously acquired A/D values are stored per longword. For example, Chan 0 Samp 0 and Chan 1 Samp 0 are acquired simultaneously. Next, Chan 0 Samp 1 and Chan 1 Samp 1 are acquired simultaneously, etc. **The format below is also identical to the format in which data is stored to disk in the example programs in 2-channel mode:**

Address	D31..24 (Byte 0)	D23..16 (Byte 1)	D15..8 (Byte 2)	D7..0 (Byte 3)
BA+\$0000000	Chan 0 Samp 0	Chan 1 Samp 0	Chan 0 Samp 1	Chan 1 Samp 1
BA+\$0000004	Chan 0 Samp 2	Chan 1 Samp 2	Chan 0 Samp 3	Chan 1 Samp 3
BA+\$0000008	Chan 0 Samp 4	Chan 1 Samp 4	Chan 0 Samp 5	Chan 1 Samp 5
BA+\$000000C	Chan 0 Samp 6	Chan 1 Samp 6	Chan 0 Samp 7	Chan 1 Samp 7
BA+\$0000010	Chan 0 Samp 8	Chan 1 Samp 8	Chan 0 Samp 9	Chan 1 Samp 9
BA+\$0000014	Chan 0 Samp 10	Chan 1 Samp 10	Chan 0 Samp 11	Chan 1 Samp 11
BA+\$0000018	Chan 0 Samp 12	Chan 1 Samp 12	Chan 0 Samp 13	Chan 1 Samp 13
End of array				

For model **AD8S-100DMA or AD8-100DMA**, in **double_speed** (single-channel 2x speed) mode, four sequential A/D values (**each in offset binary format**) are stored in each longword. **The format below is also the way data is stored to disk in the example programs for model AD8-100DMA or AD8S-100DMA in double_speed mode:**

Address	D31..24 (Byte 0)	D2..16 (Byte 1)	D15..8 (Byte 2)	D7..0 (Byte 3)
BA+\$0000000	Chan 0 Samp 0	Chan 0 Samp 1	Chan 0 Samp 2	Chan 0 Samp 3
BA+\$0000004	Chan 0 Samp 4	Chan 0 Samp 5	Chan 0 Samp 6	Chan 0 Samp 7
BA+\$0000008	Chan 0 Samp 8	Chan 0 Samp 9	Chan 0 Samp 10	Chan 0 Samp 11
BA+\$000000C	Chan 0 Samp 12	Chan 0 Samp 13	Chan 0 Samp 14	Chan 0 Samp 15
BA+\$0000010	Chan 0 Samp 16	Chan 0 Samp 17	Chan 0 Samp 18	Chan 0 Samp 19
BA+\$0000014	Chan 0 Samp 20	Chan 0 Samp 21	Chan 0 Samp 22	Chan 0 Samp 23
BA+\$0000018	Chan 0 Samp 24	Chan 0 Samp 25	Chan 0 Samp 26	Chan 0 Samp 27
BA+\$000001C	Chan 0 Samp 28	Chan 0 Samp 29	Chan 0 Samp 30	Chan 0 Samp 31
End of array				

5.9 Data Representation in Host Memory During A/D Transfers (Model AD8CH12B-100DMA)

For the 8-channel AD8CH12B-100DMA, data is stored in host system RAM is shown below, when operating in **12-bit** mode. Every four longwords of RAM contain eight simultaneously acquired A/D values, **each in offset binary format**. For example, Chan0 Samp 0 through Chan 7 Samp 0 are acquired simultaneously. Next, Chan 0 Samp 1 through Chan 7 Samp 1 are acquired simultaneously, etc. **Data is stored to disk in this same format, in the example programs.**

Address	D31-D28	D27 - D16: A/D Data for Chan 0	D15-D12	D11 - D0: A/D Data for Chan 1
BA+\$0000000	-	Chan 1 Samp 0	-	Chan 0 Samp 0
BA+\$0000004		Chan 3 Samp 0		Chan 2 Samp 0
BA+\$0000008		Chan 5 Samp 0		Chan 4 Samp 0
BA+\$000000C		Chan 7 Samp 0		Chan 6 Samp 0
BA+\$0000010		Chan 1 Samp 1		Chan 0 Samp 1
BA+\$0000014		Chan 3 Samp 1		Chan 2 Samp 1
BA+\$0000018		Chan 5 Samp 1		Chan 4 Samp 1
BA+\$000001C		Chan 7 Samp 1		Chan 6 Samp 1
.		.		.
End of array				

For **8-bit** mode on the AD8CH12B-100DMA, every two longwords contain eight simultaneously acquired 8-bit samples, **in offset binary format**. For example, Chan0 Samp 0 through Chan 7 Samp 0 are acquired simultaneously. Next, Chan 0 Samp 1 through Chan 7 Samp 1 are acquired simultaneously, etc. **Data is stored to disk in this same format in the example programs such as ad_mt.c or acquire_data2.c under Linux64, or acquire_dma_data.c under Solaris.**

Address	D31..24 (Byte 0)	D23..16 (Byte 1)	D15..8 (Byte 2)	D7..0 (Byte 3)
BA+\$0000000	Chan 3 Samp 0	Chan 2 Samp 0	Chan 1 Samp 0	Chan 0 Samp 0
BA+\$0000004	Chan 7 Samp 0	Chan 6 Samp 0	Chan 5 Samp 0	Chan 4 Samp 0
BA+\$0000008	Chan 3 Samp 1	Chan 2 Samp 1	Chan 1 Samp 1	Chan 0 Samp 1
BA+\$000000C	Chan 7 Samp 1	Chan 6 Samp 1	Chan 5 Samp 1	Chan 4 Samp 1
BA+\$0000010	Chan 3 Samp 2	Chan 2 Samp 2	Chan 1 Samp 2	Chan 0 Samp 2
BA+\$0000014	Chan 7 Samp 2	Chan 6 Samp 2	Chan 5 Samp 2	Chan 4 Samp 2
BA+\$0000018	Chan 3 Samp 3	Chan 2 Samp 3	Chan 1 Samp 3	Chan 0 Samp 3
BA+\$000001C	Chan 7 Samp 3	Chan 6 Samp 3	Chan 5 Samp 3	Chan 4 Samp 3
.		.		.
End of array				

5.10 Time Stamp Function (ADDA14/AD14 and ADDA12/AD12-100DMA only)

Models ADDA14 and AD14-100DMA and ADDA12 and AD12-100DMA, AD14-100DMA contain a time stamp counter to record the relative time positions of discontinuous A/D data bursts. The 31-bit counter has a 100ns resolution and can record intervals up to 214.7 seconds. The Time Stamp counter starts when acquisition begins and runs continuously, wrapping automatically, until **Event** is de-asserted. When sampling is thus suspended, the 31-bit Timestamp value is written to RAM. Data bit 31 is set to 1, indicating the word is a time stamp value, not A/D data. Timestamping is useful in radar or any application that requires recording disjoint portions of an analog waveform.

6. Hardware Installation and Setup

Before you begin, **be sure your system has at least 1GB of installed RAM**, and preferably 2GB or more. To avoid overheating, the ULTRADMA board **must be installed in a well-cooled workstation or server chassis, preferably with 64-bit slots, or alternatively in an industrial chassis PC**. Installation in a standard desktop PC **without fans at the front end of the card cage will cause the ULTRADMA to overheat**, and resulting damage is not covered by warranty.

1. Use the shutdown command on your system and then turn OFF the power to the system.

BEFORE REMOVING THE COMPUTER SYSTEM COVER OR REMOVING ANY BOARD, BE SURE THAT THE POWER TO THE COMPUTER, AS WELL AS TO ALL PERIPHERAL DEVICES IS OFF. WEAR A STATIC-DISSIPATING WRISTBAND THAT IS GROUNDED TO THE SYSTEM CHASSIS WHILE OPENING OR WORKING ON YOUR SYSTEM.

2. Remove any screws that attach the computer system cover and remove the cover.
3. Remove the filler bracket from the PCI bus slot into which you wish to install your ULTRADMA board. If a mixture of 64 and 32-bit slots or 5V and 3.3V slots are available in the system, **choose a 64-bit 5V slot as your first preference**. If that is not available, install it in a 64-bit 3.3V slot, or as a last resort, in a 32-bit 5V or 3.3V slot. If installing **two ULTRADMA boards in a single system (which may result in reduced throughput in some cases), the first should be installed in a 33MHz slot, and the second should be installed in a 66MHz slot**, as 33MHz and 66MHz slots are generally separate PCI buses in the system, allowing higher total DMA throughput from the two boards. For details, refer to your computer system hardware.
4. Hold the ULTRADMA board by the top of the metal PCI bracket. Then hook the tab on the bottom edge of the ULTRADMA's metal bracket into the corresponding slot in the computer's rear panel. Carefully push the ULTRADMA down so its PCI bus connector mates with the PCI bus connector on the motherboard. Be sure that the ULTRADMA is seated firmly into the motherboard PCI bus connector. Check that no other PCI boards have become unseated when the ULTRADMA was installed, as motherboards may flex slightly when installing PCI boards.
5. Plug coaxial I/O cables for the analog inputs and/or outputs into the appropriate SMA connectors on the ULTRADMA's rear bracket at the rear of the system. Please refer to the diagram on page 8 of this manual. You should not plug in the optional 26-conductor I/O cable unless you will be using the **Trigger** line, **Event** line or any of the TTL I/O lines. Connect the free ends of the analog input cables to the signal sources to be digitized.
6. We recommend that A/D channels 0 and 1 (the 4th and 3rd SMA connectors from the top) initially be connected to a signal generator set for a 300mV peak sine wave at approximately 100 KHz, allowing a quick test of the ULTRADMA's functionality using the demonstration software. If you have a model with D/A converters (e.g. ADDA14-100DMA), connect the output channels (the 1st and 2nd SMA connectors from the top) to an oscilloscope set for 500mV/div., so you can observe D/A operation when running the D/A programs, such as `da_from_disk.c` or `synth.c`.
6. Replace the computer system cover, installing all screws you had removed. Reconnect the power cables to the system and peripherals.
7. Power up and reboot the system. The system will then be ready for software installation.

7. Software Installation and Setup

7.1 Software Installation for WindowsXP™ or Windows2000™

Before installing the software, be sure the ADDA12-100DMA, AD12-100DMA, ADDA14-100DMA or AD8-100DMA board is installed in the system. Then, insert the diskette titled **AD/ADDA14/12/8 DRIVER PKG WITH USER DEMO SOFTWARE - FOR REDHAT™ LINUX™ 4.0 64bit and WINDOWS XP™ 32-BIT**. Create a directory for installing the ULTRADMA software and copy the entire contents of the diskette to this directory as follows:

```
C:> mkdir uvpci
C:> xcopy A: C:\uvpci /s /e /v
```

The following directories should appear in C:\UVPCI: “UltraDMA Package – beta2” and “Ultraview Application Source Package”.

Then, answer the questions in the “Found New Hardware” window, and browse to the “C:\UVPCI\UltraDMA Package – beta2” directory when prompted (the file **UltraDMA.inf** should be displayed in the installation window), and click “OK”. To run the software, and to modify and recompile the example programs, please refer to the documents in the two directories in the release, as well as the section of this manual on running the WindowsXP and 2000 examples.

Windows, Windows 2000 and Windows XP are trademarks of Microsoft Corp™.

7.2 Software Installation for Solaris 8,9 or 10 (SPARC Platform Edition™)

Before installing the software, be sure the board is installed in the system. **Solaris 9 or 10 are the preferred OS versions**. Versions of Solaris 8 prior to the 10/00 release should be avoided.

To avoid data overruns, interruptions in data acquisition, and hanging of user programs, **permanently turn off all power management options, screen savers, etc.** The **system must always remain in full-power mode when running the data acquisition board, and must not be allowed to go into sleep mode**, or even screen saving mode when the board is running.

To begin the software installation insert the diskette titled **AD/ADDA14/12 DRIVER PKG WITH USER-DEMO SOFTWARE-FOR Solaris 8, 9, 10 SPARC**.

Log in as root, and type in the following two lines at the prompt (shown here as #):

```
# volcheck
# pkgadd -a none -d /floppy/floppy0/uvpci
```

You will be shown a list of packages on the diskette (should be Ultrad only) and asked which you wish to install. You can just take the default (press Return) to install the package.

Next, the installation script will ask you which directory should be the base directory for the package. You can choose **/opt/uvpci**, or choose some other place on your system. You should **choose an empty directory for installation**. The pkgadd program may issue a warning about /etc/devlink.tab; ignore the warning, as it is just going to modify the file, not overwrite it. You may also see a question about running programs with superuser permissions – just answer yes to this.

Once all the files have been copied to the base directory, some installation scripts are automatically run, giving a usable binary distribution of the package.

One of the questions you are asked by the post-installation script is whether you wish to recompile the package. If you have the Gnu C compiler (this compiler is free from Gnu and is the recommended compiler for developing software for the Ultrad boards) or the SunPro C compiler, you can recompile. You will be asked where the compiler resides (eg. /opt/gnu/bin/).

Just before returning to the prompt, pkgadd will warn you that you need to reboot your system, since a new driver has been added. Now it is necessary to reboot the system as shown below :

```
# /usr/bin/shutdown -y -i6 -g0
```

When the system reboots, the driver will be installed and operational. If you will only need to use memory buffers that are smaller in size than half of the installed RAM in the system, or you are not using an AD/ADDA14-100DMA, your installation is complete. **If you are using an AD14 or ADDA14-100DMA, you may wish to install a demo example program that correctly displays the 14-bit A/D data in waveform format.** To do this, insert the floppy again, and type:

```
# volcheck
# cd /opt/uvpci/bw_dig_osc      (assuming you installed the uvpci software at /opt/uvpci)
# cp /floppy/floppy0/digosc14/* .  (don't forget the period)
```

With any model board, if you will need a larger memory buffer than half of the size of the installed physical RAM, such as if you want to acquire 3 gigabytes of data continuously at 100 MB/sec, and your system has 4 gigabytes of RAM DIMMS installed), then you will need to temporarily become superuser and **add the following line to the /etc/system** file on your system, and then reboot:

```
set max_page_get=0x7fffffff  (note that this is 0x7 followed by seven f's)
```

Caution must be used once this has been done, **as it will potentially allow you to allocate a buffer so large that no user memory might be available for other system activities, causing the system to hang.** For this reason, it is essential to leave at least 1/2GB of free memory for other operations. If your system has 2GB of memory, do not ask for more than 1.5GB (3000 blocks) when running programs. For example, when running the example program fast_acquire_dma_data, the following is the largest fast_acquire recommended in that system:

```
fast_acquire_dma_data -s2 -b3000 -fsam.dat
```

After the driver has been installed as described above, it is now possible to immediately run the demonstration programs. Go to the directory bw_dig_osc, to run an oscilloscope demo. Connect a signal generator set for a 300-millivolt peak signal at 100 kilohertz to both A/D input channels (the third and fourth SMA connectors from the top). Also, if your ULTRADMA has D/A converters, connect the two D/A outputs (The top two SMA connectors) to an oscilloscope set for 500mV/Div and 10 μ s per division. Each time you click on the "Digitize" button on the screen, the ULTRADMA board should digitize and display on the screen the data from the signal generator.

You may next want to go to the example_programs directory, and run some of the examples, such as **acquire_dma_data**, or **fast_acquire_dma_data**, which store A/D data to disk, or **da_from_disk** or **fast_da_from_disk**, which play back disk data on the optional D/A converters. Another example, **synth**, generates two sine waves, continuously synthesized by the CPU, on the D/A outputs. These examples, which can be invoked with command line arguments that specify sample rate, amount of data to store, etc, are described in the next chapter.

The above example programs are a good starting point for users developing their own code. They can easily be modified and recompiled by modifying the makefile and then typing "make".

7.3 Software Installation under RedHat™ Enterprise Linux WS 4.0 (64-bit)

WARNING – KNOWN BUGS:

- 1) There is a known driver bug in the current Linux64 release (uvpci-1-3.1.x86_64) in which, on every acquisition of data, the first 512KB block of data transferred into host RAM or written to disk is garbage (incorrect data). All subsequent blocks of data are correct. This issue is with the Linux driver and not with the board. At the fastest sampling rate, 0x02, which corresponds to 25MSPS on both channels, the delay between when the start instruction occurs and the board begins acquiring the 2nd (first good) block of data is 5.24us (40ns/sample x 131072 samples).
- 2) On D/A transfers to Intel Xeon/EM64T-based machines such as Dell 670, the maximum D/A sample rate is only approximately 2 MSPS, due to that platform' slow PCI DMA on reads (AMD Opteron or Athlon-64-based systems can achieve approximately 10MSPS on D/A). This problem does not occur A/D transfers, where the full-speed transfer rate of 25MSPS on two concurrent channels can almost always be achieved.

Before installing the software, be sure the board is installed in the system. **RedHat Enterprise Linux WS 4.0 is the preferred OS, although Fedora Core5 may be used if the driver and user programs are recompiled. While other versions of Linux may be usable, Ultraview only supports installations in which RedHat Linux WS4.0 is used.**

To avoid data acquisition overruns or interruptions, permanently **disable all power management options, screen savers, etc.** The system must always remain in full-power mode when running the data acquisition board, and must not be allowed to go into sleep mode. Also the system must have at least 1GB (2-4GB preferred) of installed RAM to realistically acquire gigabytes of data at a time, at full speed, unless a suitably fast RAID system (>120MB/sec) is present. Program `disk_test.c` allows measurement of system disk throughput.

To begin the software installation insert the diskette titled **AD/ADDA12/14 DRIVER PKG WITH USER-DEMO SOFTWARE-FOR RedHat™ ENTERPRISE LINUX 4.0 64 BIT.**

Log in as **root**, copy the file `uvpci-1-3.1.x86_64.rpm` to any directory, and type in the following two lines at the prompt (shown here as #):

```
# rpm -ivh uvpci-1-3.1.x86_64.rpm
```

The installation script will automatically create a directory `/uvpci`, and install the software there. If you have an AD12 or ADDA12-100DMA board, you must then copy the program `digosc_file12.c`, from the floppy, over `digosc_file.c` to correctly display data for these 12-bit boards. Before running the example programs, go to `/uvpci/example_programs`, and recompile them by typing:

```
# make clean
# make all
# make -f make_digosc (optional – for recompiling digosc_file.c. Requires GTK+ library)
```

Follow the directions in the section “Running the Example Programs under Linux64”, to run the example programs. Example programs `ad_mt.c` and `acquire_data2.c` acquire data via the A/D converters, to a file, while the program `da.c` outputs data on boards with D/A converters. Program `digosc_file.c` displays previously acquired data from the file, in waveform format. To uninstall the Linux64 software release for the UVPCI board, as required prior to installing a newer version of UVPCI software, the following command is used:

```
# rpm -e uvpci-1-2
```

7.4 Software Installation under RedHat™ Linux 9.0 (32-bit only)

32-bit Linux versions are no longer supported by Ultraview, but are included for legacy purposes. Before installing the software, be sure the board is installed in the system. **RedHat Linux 9.0 (Build 2.4.20-31.9 or later) is the preferred OS. While other versions of Linux may be usable, Ultraview has only supported installations in which the RedHat Linux 9.0 was used.**

To avoid data overruns, interruptions in data acquisition, and hanging of user programs, **permanently turn off all power management options, screen savers, etc.** The **system must always remain in full-power mode when running the data acquisition board, and must not be allowed to go into sleep mode**, or even screen saving mode when the board is running. **Also the system must have at least 1GB (2-3GB preferred) of installed RAM** to realistically acquire data records in the multi-hundred megabyte size, at full speed.

To begin the software installation insert the diskette titled **AD/ADDA12/14 DRIVER PKG WITH USER-DEMO SOFTWARE-FOR RedHat™ LINUX 9.0 32 BIT and WINDOWS XP & 2K 32-BIT.**

Log in as **root**, copy the file `uvpci-2005-06-22.tar.gz` to the directory (such as `/ultradma`) in which you would like to install the software, and type in the following two lines at the prompt (shown here as #):

```
# gunzip uvpci-2005-06-22.tar.gz
# tar xvf uvpci-2005-06-22.tar
# cd uvpci-2002-06-27
```

Follow the directions in the README file, as well as the section “Running the Example Programs under Linux”, below, for running the example programs. When the driver first installs, you may see some warning messages about unresolved symbols. These messages may be ignored.

If you wish to recompile the programs, you will need access to the kernel headers and configuration used to build the kernel you are going to compile the driver against. If this kernel source is not yet installed on your system, it must be installed. For example, for RedHat 9.0 build 2.4.20-31.9, you should install `kernel-source-2.4.20-31.9.rpm`. Once this package has been installed, do the following (the actual directories and file names may vary if you are using a different version of Linux, and may vary depending on your machine architecture):

```
# cd /usr/src/linux-2.4.20-31.9/configs
# cp configs/kernel-2.4.20-i686.config ../config
```

(NOTE: you may need to use kernel-2.4.20-i386 on certain non-Intel platform machines)

There should now be a file `.config` in your `/usr/src/linux-2.4.20-31.9` directory. You may now go back to `/ultradma/uvpci-2002-06-27` (or the directory in which you installed the ultraview software) and type “make clean” (which will remove the precompiled objects and executables), followed by “make”. This should cause a rebuild of the entire release.

Go to the section in this manual, entitled “Running the Example Programs Under RedHat Linux 9.0” for instructions on acquiring data with the board. This section is very important, as specifying the appropriate buffer size in the command line arguments for the example programs is essential, if maximum throughput is to be obtained.

7.5 Running the Example Programs Under Solaris 8, 9 or 10™

There are two directories with both source and executables for various example programs, which can immediately be run to demonstrate the use of the board, and which form an excellent basis for developing your own custom software for the board. Full source and makefiles are provided, allowing for easy modification and recompilation.

7.5.1 digosc (digital oscilloscope)

The first program to run is called “**digosc**”, and is a crude digital waveform display of the A/D signal on both channels on your board. This program is in the directory `/opt/uvpci/bw_dig_osc`. Go to this directory (`# cd /opt/uvpci/bw_dig_osc`). If your board is an **ADDA12-100DMA** or **AD12-100DMA**, the program **digosc** is the executable, and the programs `digosc_controls_uvpci_stubs.c` and `dig_osc_controls_ui.c` are the source routines. Run the program **digosc** by typing **digosc** at the prompt. If you then get a window with an oscilloscope display, then hit the **digitize** button, which should result in the display of the two waveforms from your signal generator. Adjust your signal generator’s frequency and amplitude to display several cycles of the desired waveform.

If you do not see waveforms, be sure that the system’s environment is correctly set. The lines present in `/opt/uvpci/environment/cshrc` should be contained in your `.cshrc` file. Once these lines are copied into your `.cshrc`, try invoking a new C shell and rerunning **digosc**.

If your board is instead an **AD8-100DMA**, the program **digosc8x2** is the executable, and `digosc8x2.c` must be compiled. “make” will recompile `dig_osc_controls_uvpci_stubs8x2.c` if this file is modified, and the **digosc8x2** executable will be produced.

If your board is instead an **AD8CH12B-100DMA**, the program **digosc8ch** is the executable, and `digosc8ch.c` must be compiled. “make” will recompile `dig_osc_controls_uvpci_stubs8ch.c` if this file is modified, and the **digosc8ch** executable will be produced. From then on, use the software in the same manner as described above for the **ADDA12-100DMA** or **AD12-100DMA**.

Other programs that are very useful are **acquire_dma_data.c**, **fast_acquire_dma_data.c**, **da_from_disk.c**, **fast_dma_da_from_disk.c** and **synth.c**. These programs are all in the directory `/opt/uvpci/example_programs`.

7.5.2 fast_acquire_dma_data (acquire data to disk file)

The program **fast_acquire_dma_data.c** allocates a large buffer in user memory, causes the board to acquire up to hundreds megabytes (or even up to 3.5 Gigabytes under Solaris 8 or later) of data and store it in this memory buffer and then, after the buffer is full, to store it to disk. To do this at full speed (100MB/sec) will require a large amount of physical RAM be installed in the system. For example, if 2GB of data is to be acquired, uninterrupted, at full speed, it is generally necessary to have 4GB of DIMMS installed in the system.

The program may be invoked as in the following example:

```
# fast_acquire_dma_data -s8 -b60 -fhello.dat
```

where..

-s8 in this example instructs the board to have a sample interval of 8 ($8 \times 20\text{ns} = 160\text{ns} = 6.25$ million dual 14-bit samples per second, for AD14-100DMA or ADDA14-100DMA),

-b60 in this example instructs the board to acquire a total of 60 512KB blocks (approximately 30MB, or 7.5 million dual 12/14-bit samples for an AD12/14-100DMA or ADDA12/14-100DMA or 15 million dual 8-bit samples for an AD8-100DMA or AD8S-100DMA),

-fhello.dat specifies that the data is to be stored to a file named "hello.dat"

You may verify that this data has been stored by typing "ls -l hello.dat"(you should see a file hello.dat of length 31457280 bytes.

IMPORTANT: When running fast_acquire_dma_data.c or acquire_dma_data.c, the following limitations apply:

- 1) **Do not specify a number of blocks more than approximately half of the amount of installed RAM in your system (unless you have added the line "set max_page_get=0x7fffffff" in /etc/system), and in any event not more than the amount of installed DIMM RAM minus approximately 0.5GB.**
- 2) **Do not specify a sample interval faster than 2 (25 million dual 12-bit samples/second = 100MB/sec) for an AD14, AD12, ADDA14 or ADDA12-100DMA, or AD12B8CH in 12-bit mode and do not specify a sample interval faster than 1 (50 million dual 8-bit samples/second = 100MB/sec) for an AD8-100DMA, AD8S-100DMA or AD12B8CH board in 8-bit mode.** These speeds assume that your board is installed in a 64-bit 33MHz PCI slot. If your board is installed in a 32-bit slot, these numbers should be increased (sampint ≥ 3 for an AD12-100DMA or ADDA12-100DMA or 2 for an AD8-100DMA). If you try to sample faster than this, you could experience data overruns (periods of skipped data).

7.5.3 acquire_dma_data (Acquire data to disk file of any length)

The program **acquire_dma_data.c** is similar to **fast_acquire_dma_data.c** allocates a large buffer in user memory, causes the board to acquire many megabytes (or even hundreds of megabytes) of data and store it in this memory buffer. However, unlike fast_acquire_dma_data.c, acquire_dma_data.c begins storing data from the buffer to disk as soon as data starts coming into the buffer. Also, it allows buffer wraparound, allowing storage of larger amounts of data than will fit into the memory buffer. However, in cases where the desired file size (size of data file stored to disk) exceeds the size of the buffer, the acquisition speed is greatly limited, as the average acquisition rate must not exceed the rate at which data can be continuously stored on the disk – typically only 4-12MB/sec. For this reason, it is best to specify the largest buffer size that the system will give you (which is a function of the amount of memory installed in the system).

The program may be invoked as in the following example:

```
# acquire_dma_data -s32 -b60 -c700 -fhello.dat
```

where..

-s32 in this example instructs the board to have a sample interval of 32 ($32 \times 20\text{ns} = 640\text{ns} = 1.5625$ million dual 12/14-bit samples per second, for AD12/14-100DMA or ADDA12/14-100DMA),

-b60 in this example instructs the board to use a memory buffer of 60 512KB blocks (approximately 30MB, or 7.5 million dual 12/14-bit samples (each in offset binary format) for an AD12/14-100DMA or ADDA12/14-100DMA or 15 million dual 8-bit samples for an AD8-100DMA),

-c700 in this example instructs the board to store a total of 700 512KB blocks (approximately 350MB) of data to disk. Since the disk needs to be able to keep up with the sustained rate of the A/D acquiring data, sample intervals must be limited to those that do not cause data to be pushed at the disk faster than rates that it can continuously accept.

-fhello.dat specifies that the data is to be stored to a file named "hello.dat"

7.5.4 synth (dual sine wave synthesizer)

The program **synth.c** is an example of a program that uses the CPU to continuously synthesize the data for two sine waves at different, and unrelated frequencies, and store them to a memory buffer, from which the board can read the data and output it via the D/A converters (on model ADDA14, ADDA12 or DA14-100DMA). An example of the use of **synth.c** is as follows:

synth -s30

This program generates two sine waves, the first at slightly over 1 KHz and the second at 2KHz (regardless of the sample interval chosen – shorter sample intervals will merely have more points per waveform). The two waveforms will appear to slide slowly by one another.

In general, on 64-bit SUN systems, such as Ultra 60, Ultra 80 and E250/E450, if **synth** is run with sample intervals less than approximately 15-20, overruns may occur.

7.5.5 da_from_disk (D/A playback of file)

The program **da_from_disk.c** outputs data from a file, via the D/A converters on models ADDA14-100DMA, ADDA12-100DMA and DA14-100DMA. It is used as follows:

da_from_disk -iad.dat -s30

In this example, data stored in the file **ad.dat** is outputted via the dual D/A converters at a rate (1.666 million dual 14-bit samples/second) specified by the sample interval of 30 (30 x 20ns). A sample file that has a sine wave on one channel and a triangle wave on another channel may be generated by using the program **make wave** (**# make_wave -b50** generates a 50MB long file.) Alternatively, one may "play back" a file acquired from A/D data by merely specifying the name of the file that had the A/D data stored in it. Please note that if this is done on an ADDA12-100DMA, the d/a waveforms will have an amplitude only ¼ that of the original A/D data, and will be offset to the bottom one fourth of the -2 to +2v range, since the A/D data is stored as 12-bit data, while the D/A data is in 14-bit format (both boards store data in offset binary format). On an ADDA14-100DMA, this will not occur, and the D/As will have full-scale amplitude, as the D/A and A/D bit formats are identical.

The maximum speed for running **da_from_disk** is limited by the speed that the disk can continuously supply data at, typically a sample interval of 15 or more.

7.5.6 fast_dma_da_from_disk (play back of file at high speed)

The program **fast_dma_da_from_disk** also outputs data from a file, via the D/A converters on models ADDA14-100DMA, ADDA12-100DMA and DA14-100DMA, but allows the user to specify a larger buffer than the default size of 4MB, resulting in potentially much higher speed of output. For example, if a file whose data is to be output is 500 MB long, and one wants to output the data

at a speed that is not limited by the disk speed, one can specify that a 500 MB buffer be used by using an argument of **-b500** as shown below:

```
# fast_dma_da_from_disk -iad.dat -s20 -b500
```

In this example, data stored in the file **ad.dat** is outputted via the dual D/A converters at a rate (2.5 Million dual samples/second) specified by the sample interval of 20 (20 x 20ns). Note that, unlike the program `da_from_disk`, the program `fast_dma_da_from_disk` first loads as much of the disk file as it can fit into the 500 MB memory buffer, and only then starts to output the data using the D/A. As the buffer is emptied via the D/A converters, new data, if any, from the file is transferred to the buffer. Therefore, although very large buffers allow very high-speed operation, they do cause an initial delay in starting the output of data via the D/A converters, by virtue of the fact that the first buffer must be completely filled from disk before any D/A outputting happens.

The maximum speed for running `fast_dma_da_from_disk` is limited by the speed that the board's dma engine and the system's PCI bus can transfer data on DMA reads, which is typically 80MB/sec on 64-bit Sun platforms, such as Ultra 80, Ultra60, E250, E420 and E450 and SunBlade1000/2000. This assumption of course is based on the user specifying a buffer size that will hold most, or all, of the file being outputted. To do this might, in some cases, require a large amount of physical RAM be installed in the system. For example, if 1GB of data is to be outputted at full speed, it is generally necessary to have 2GB of DIMMS installed in the system.

7.6 Running the Example Programs under RedHat™ Linux WS 4.0 64-bit

The directory /uvpci has full source and executables for the driver and various example programs, which can immediately be run to demonstrate the use of the board, and which form an excellent basis for developing your own custom software. Full source is provided for all sample user programs (in subdirectory example_programs), allowing for easy modification and recompilation.

WARNING – KNOWN BUGS:

- 3) **There is presently a known driver bug in the current Linux64 release (uvpci-1-2.x86_64) in which, on every acquisition of data, the very first 512KB block of data transferred into host RAM or written to disk is garbage (incorrect data). All subsequent data blocks are correct.** This issue is with the Linux driver and not with the board. At the fastest sampling rate, 0x02, which corresponds to 25MSPS on both channels, the delay between when the start instruction occurs and the board begins acquiring the second (1st good) block of data is 5.24us (40ns/sample x 131072 samples).
- 4) **On D/A transfers to Intel Xeon/EM64T-based machines such as Dell 670, the maximum D/A sample rate is only approximately 2 MSPS (On AMD Opteron or Athlon-64 systems, the maximum sample rate is can reach 10 MSPS). The problem is due to the slow PCI bus DMA rate on reads on certain machines. This problem does not occur A/D transfers, in which the full-speed rating of 25MSPS x 2 channels is achievable.**

7.6.1 acquire_data2.c (acquire data to RAM, then store to disk file)

The program **acquire_data2.c** allocates a large buffer in host user memory, causes the board to acquire up to hundreds of megabytes (gigabytes in systems with sufficient RAM installed) of data and store it in this memory buffer and then, after the buffer is full, stores it to disk. This program generally allows the fastest sample rate to be used, as it is only dependent on the DMA rate of the PCI bus, as data is stored from the board directly to high-speed system RAM, (and later to disk) rather than being limited to the speed of the slower disk system. To do this at high speed (up to 100MB/sec = 2 x 25 MS/sec of 12 or 14-bit data) requires a large amount of installed physical RAM in the system. For example, if 800MB of data is to be acquired at 100MB/s, uninterrupted, it is generally good to have 2 GB of DIMMS installed in the system. To achieve sample rates of over approximately 25MB/sec, the buffer size, n, must 1) be large enough to hold all of the acquired data, and 2) be small enough to be allocated by the system without it paging. The second condition may require several GB of RAM to be installed in the system. If either of these conditions are not met, the system will page, and acquisition speed will be disk-limited. The program may be invoked as follows:

```
# ./acquire_data2 /dev/uvpci0 1000 data.dat
```

where:

/dev/uvpci0 specifies that the first ULTRADMA board is to be used to acquire data,

1000, in this example. instructs the board to acquire a total of 1000 512KB blocks (approximately 512MB, or 128 million dual 14 or 12-bit samples for an AD14, ADDA14, AD12 or ADDA12-100DMA or 256 million dual 8-bit samples for an AD8-100DMA or AD8S-100DMA),

data.dat specifies that the data is to be stored to a file named “data.dat”

You may verify that this data has been stored by typing “ls -l” (you will see a 512MB file, data.dat).

The default sample interval is 0x02 (100MB/sec, which is equivalent to 25 million dual 12 or 14-bit

samples/second). To modify the sample rate, for example to half this rate (interval of 0x04), edit the line “**uvpci_set_sample_interval(board_fd, 0x02);**” to become “**uvpci_set_sample_interval(board_fd, 0x04);**”, and then recompile the program by typing “**make acquire_data2**”. The default output file name is uvpci.dat, the default number of blocks is 1000 (which equals 512MB of data). Therefore, merely typing “./acquire_data” will cause the first A/D board (/dev/uvpci0) to acquire 1000 blocks, at a sample interval of 2 to RAM and then store the data to file uvpci.dat.

7.6.2 ad_mt.c (Acquire data to disk file of any length)

The program **ad_mt.c** is similar to **acquire_data2.c**, but allocates a smaller, cyclic, buffer in user memory and causes the board to acquire data and begin to store it in this memory buffer. Unlike **acquire_data2.c**, **ad_mt.c** begins storing data from the buffer to disk as soon as data starts coming into the buffer. Also, it allows buffer wraparound, allowing storage of larger amounts of data than will fit into the memory buffer. However, in cases where the desired file size (size of data file stored to disk) exceeds the size of the buffer, the acquisition speed is greatly limited, as the average acquisition rate must not exceed the rate at which data can be continuously stored on the disk – typically only 4-15MB/sec. **To achieve sample rates of over approximately 25MB/sec, at least one of the following conditions must be true:**

- 1) The buffer size, **BUFF_SIZE**, must be large enough to store the entire amount of data acquired, yet small enough to be allocated by the system without paging. The second condition may require several GB of RAM to be installed in the system. If either of these conditions is not met, the system will page, and acquisition speed will be disk-limited.
- 2) The disk system must be a fast RAID, or other system with sufficient continuous, sustained throughput to store data at the specified sample rate, which may be as high as 100MB/sec. To achieve the highest sampling rate, and store a multi-gigabyte file of data, a RAID controller such as, Adaptec U320, and four SCSI-320 drives, are usually needed.

To use this program, type

```
# ./ad_mt /dev/uvpci0 250 1000 data.dat
```

where:

/dev/uvpci0 specifies that the first ULTRADMA board is to be used to acquire data,

250 in this example instructs the board to use a user memory elasticity buffer of 250 blocks to store incoming data before storing it to disk.

1000 in this example instructs the board to acquire a total of 1000 512KB blocks (approximately 512MB, or 128 million dual 14 or 12-bit samples for an AD14, ADDA14, AD12 or ADDA12-100DMA or 256 million dual 8-bit samples for an AD8-100DMA or AD8S-100DMA),

data.dat specifies that the data is to be stored to a file named “data.dat”

You may verify that this data has been stored by typing “ls -l” (you will see a 512MB file, data.dat).

The program may be run with default values by simply typing “./ad_mt”, and has a default sample interval of 0x02 (100MB/sec, which is equivalent to two simultaneous 12 or 14-bit A/D channels acquired at 25MSPS). Also, the default number of blocks acquired is 3000 (approximately 1.5GB).

To modify the default **sample interval**, edit the line “uvpci_set_sample_interval(board_fd, 0x02)” in **ad_mt.c**. For example editing this line to **uvpci_set_sample_interval(board_fd, 0x10);**, and recompiling would cause the board to have a sample interval of 16 (16x20ns = 640ns = 1.5625 million dual 14 or 12-bit samples/second, for AD14, ADDA14, AD12 or ADDA12-100DMA),

To modify the default **number of blocks to be acquired**, edit the line “#define CVT_COUNT 3000”. For example, to acquire 5GB of data (subject to having sufficient disk space to do this), change this line to #define CVT_COUNT 10000” and recompile ad_mt.c. Be sure that the disk system has sufficient continuous throughput to acquire 5GB of data at this rate. To test your system’s approximate average disk throughput, use the program disk_test.c, described below.

To modify the default **file to store data to**, edit the line “#define DEF_FILE_NAME “uvpci.dat”, substituting the name of the file you wish to store the data to, and then recompile ad_mt.c.

7.6.3ad_mt_wrap.c (Acquire data to disk, with pre-trigger recording capability)

The program **ad_mt_wrap.c** is similar to **ad_mt.c**, but allows the continuous recording of data until an externally supplied trigger event, after which time the N blocks prior to the trigger are retained in a file. **As with the program ad_mt.c, to achieve sample rates of over approximately 25MB/sec, at least one of the following conditions must be true:**

- 1) The buffer size, BUFF_SIZE, must be large enough to hold the entire amount of data acquired, and yet small enough to be allocated by the system without paging. The second condition may require several GB of RAM to be installed in the system. If either condition is not met, the system will page, and acquisition speed will be disk-limited.
- 2) The disk system must be a fast RAID, or other system with sufficient continuous, sustained throughput to store data at the specified sample rate, which may be as high as 100MB/sec. To achieve the highest sampling rate, and store a multi-gigabyte file of data, a RAID controller such as, Adaptec U320, and four SCSI-320 drives, are usually needed.

To use this program, type

```
# ./ad_mt_wrap /dev/uvpci0 250 1200 uvpci.dat
```

In this example, the board acquires A/D data into a 250-block elasticity buffer in host RAM and re-writes this RAM data to a 1200 block circular disk buffer as blocks are acquired. When the end of the disk file is reached, it wraps around and begins writing new data at the beginning of the file, over old data. The board stops acquiring data only once a software trigger is invoked. In this program, the trigger is simply the pressing of the keyboard ENTER key. Acquisition then stops, with the program knowing the block in the cyclic disk buffer (the file uvpci.dat) on which acquisition stopped. The program then reconstructs the data from this disk file so that the oldest sample is the first and the newest sample is last, and stores the new data in a file named “new_data.dat, which is then a file of the last 1200 contiguous blocks of data recorded prior to the trigger.

This program’s default sample interval is 0x02 (100MB/sec), equivalent to two concurrent 12/14-bit A/D channels acquired at 25MSPS. The default number of blocks acquired is 1200 (600MB).

To modify the default **sample interval**, edit the line “uvpci_set_sample_interval(board_fd, 0x02)” in ad_mt_wrap.c. For example editing this line to **uvpci_set_sample_interval(board_fd, 0x10);**, and recompiling would cause the board to have a sample interval of 16 (16x20ns = 640ns = 1.5625 million dual 14 or 12-bit samples/second, for AD14, ADDA14, AD12 or ADDA12-100DMA),

To modify the default **number of blocks to store**, edit the line “#define CVT_COUNT 1200”. For example, to acquire 5GB of data (if you have sufficient disk space), change this line to #define CVT_COUNT 10000” and recompile ad_mt_wrap.c. Be sure that the disk system has sufficient throughput to acquire 5GB of data continuously at this rate. To test your system’s rough average disk throughput, use the program disk_test.c described below.

To modify the default intermediate **file to store data to**, edit the line “#define DEF_FILE_NAME “uvpci.dat”, substituting in the name of the new file, and then recompile ad_mt_wrap.c.

To modify the final **file to store the contiguous data to**, edit the line “new_file=open (“new_data.dat”, O_CREAT | O_RDWR | O_TRUNC, 0666);”, substituting the name of the file you wish to store the data to, and then recompile ad_mt_wrap.c.

7.6.4 disk_test.c (test disk system throughput, using dummy data)

disk_test.c synthesizes dummy data and stores this data to large files on disk. To run this program, simply type ./disk_test. The default number of blocks stored is 15000 (7.5GB). To change this value, edit the appropriate line in disk_test.c and recompile. The program, when run, will print out the number of seconds required to store the specified amount of data, as well as the average disk throughput. In general, to ensure that a data acquisition program, such as ad_mt.c will be able to store data at 100MB/sec, the program disk_test.c should indicate a sustained disk throughput (for the same size file) of at least 120MB/sec, and preferably 140MB/sec or more.

7.6.5 da.c (D/A playback of file)

The program **da.c** outputs data from a file, via the D/A converters on models ADDA14, ADDA12 and DA14-100DMA. It is used as follows:

```
# ./da /dev/uvpci0 1000 uvpci.dat
```

In this example, data previously stored in the file **uvpci.dat** is output, via the dual D/A converters, at a rate (1.19 million dual 14-bit samples/second) specified by the sample interval of 0x15 (21 x 20ns). One may “play back” a file acquired from A/D data by merely specifying the name of the file that had the A/D data stored in it. Please note that if this is done on an ADDA12-100DMA, the d/a waveforms will have an amplitude only ¼ that of the original A/D data, and will be offset to the bottom one fourth of the -2 to +2v range, since the A/D data is stored as 12-bit data, while the D/A data is in 14-bit format. On an ADDA14-100DMA, this will not occur, and the D/As will have full-scale amplitude, as the D/A and A/D bit formats are identical. **NOTE: For sample rates above approximately 5 megasamples/sec on D/A operation is it usually necessary to use an AMD Athlon64 or Opteron-based system. An Intel XEON/EM64T system tested (Dell Precision 670 workstation) was unable to support sample intervals under 14 on D/A, while several AMD-based systems were able to support sample intervals of 4 or 5 on D/A. Both platforms, however, support A/D sampling rates of 0x02, the board’s fastest sampling rate.**

To modify the **sample interval**, edit the line “uvpci_set_sample_interval(board_fd, 0x02)” in da.c. For example editing this line to **uvpci_set_sample_interval(board_fd, 0x20)**;, and recompiling would cause the board to have a sample interval of 0x20 (32x40ns = 1280ns = 781,250 dual 14 or 12-bit samples/second, for AD14, ADDA14, AD12 or ADDA12-100DMA),

7.6.6 digosc_file.c (for AD14-100DMA and ADDA14-100DMA only)

digosc_file.c displays, in waveform format, the A/D data in any file containing data acquired using an AD14 or ADDA14-100DMA. If you instead have an AD12 or ADDA12-100DMA, you must overwrite the file digosc_file.c in the example_programs directory, with the file digosc_file12.c that is separately located on the installation floppy. Then, recompile this program by typing **make -f make_digosc**. To display the data in previously recorded file, uvpci.dat, type:

```
./digosc_file -f uvpci.dat
```

ULTRAVIEW

The program will open a window and display a rapidly scrolling waveform that, if not interrupted, scrolls through the entire data recorded in the file. This output can be interrupted any time by clicking on the slider bar. The contents of any desired block may be displayed by typing the block number and then clicking "Update". The scroll bar then allows any portion of the block to be displayed; note that the number displayed on *the scroll bar* indicates the offset in the block of the first sample plotted on the screen. To display a different block, hit the up arrow (or type in the new desired block number) and click the "Update" button. The data shown is not valid until "Update" has been pressed. Then, you may scroll through the new block using the scroll bar.

This example program may be modified and recompiled by typing, "make -f make_digosc". As mentioned in the include readme.txt file, the GNOME software development package, Specifically the GTK+ library files, which provide the graphical interface, must be installed.

7.7 Running Example Programs Under WindowsXP™ or Windows2000™

Refer to the document “ULTRADMA Driver Functional Introduction”, in the UltraDMA Package – Beta 2 directory of the Windows XP/2000 release for further details. The source code for the three application programs, `acquire.c`, `spitout.c` and `inout.c` are in the directory Ultraview Application Source Package/`acquire`, `/spitout`, and `/inout`, respectively. Full source and makefiles are provided in the three respective subdirectories of directory “Ultraview Application Source Package”, and instructions for recompiling are in the file “Compiling the User Applications”, in the top directory, allowing for easy modification and recompilation. The executables for these example programs, which can immediately be run to demonstrate the use of the board, are in the directory “UltraDMA Package – Beta2”. These three applications demonstrate how the driver is used to communicate with the Ultraview A/D board. Two of the applications (`acquire.exe` and `spitout.exe`) use the synchronized method to allow users to transfer large amounts of data to and from the devices and store/retrieve this data from the hard disk. The third application (`inout.exe`) demonstrates the utility of single frame method of IO.

7.7.1 `acquire.exe`

This program acquires the number of blocks specified, at the specified sample rate, and stores the data to the specified file on disk. Usage is as follows:

```
acquire.exe  -n board_num -s int -b blocks -f filename
```

-n The device number (if multiple devices are present), defaults to 1
 -s The sample interval as specified in the users manual
 -b The number of 512KB blocks to be acquired
 -f The name of the file on the hard disk in which the data will be stored

7.7.2 `spitout.exe`

This program plays back data from the specified file, at the specified sample rate. Usage is as follows:

```
spitout.exe  -n board_num -s int -f filename
```

-n The device number (if multiple devices are present), defaults to 1
 -s The sample interval as specified in the users manual
 -f The name of the file on the hard disk in which the data will be stored

- This application requires the `-f` parameter to be supplied. Note that `spitout.exe` does not allow an input specifying the number of blocks, instead the number of blocks is found from the length of the file specified to avoid errors.

7.7.3 `inout.exe`

This program acquires a single block of data, at the specified sample interval, and then plays it back, either once or repeatedly, as shown below:

```
inout.exe    -n board_num -s int -p
```

- n The device number (if multiple devices are present), defaults to 1
 - s The sample interval as specified in the users manual
 - p Supplying this option creates periodic output. Omitting this option causes reproduction of just the single frame acquired.
- Inout.exe does not store data or retrieve data from the hard disk or system RAM. The application causes the acquisition of a single frame of data using the sample interval supplied with the `-s` option (if this option is not supplied a sample interval of 10 is used). The application then reproduces this data using the same sample interval as either just the single frame, or as periodic output, as specified by the `-p` option.

7.7.4 Altering the Windows XP/2000 example programs above

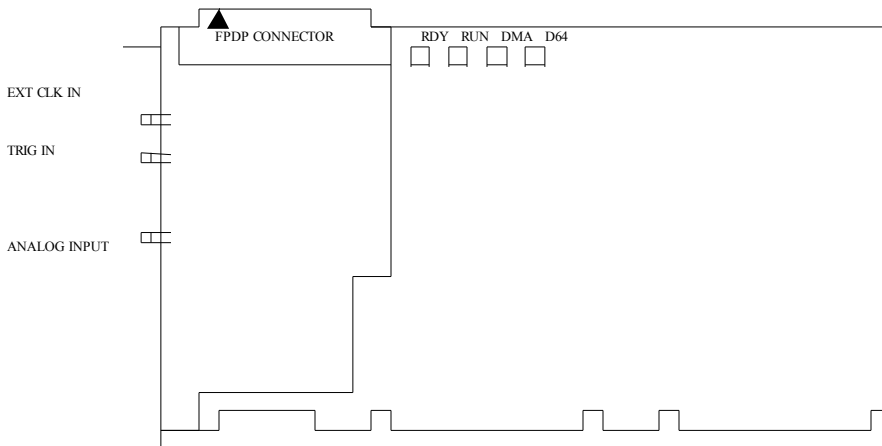
If you desire to use the program source code for the above programs which is distributed as part of the UltraDMA software package, the following points should be considered:

- Due to the method for communication between user applications and the UltraDMA device driver (GUID method) applications built using the program code distributed with the package must be linked to MFC.
- The code that identifies and “attaches” to the UltraDMA devices uses COjects declared in “afxtempl.h”, therefore this file must be included into any file which uses this process for connecting to UltraDMA devices.
- The file “winioctl.h” must be included in any file that uses IOCTLs (IO control codes).
- The driver initializes all bits in the control register to be cleared except the Interrupt_Enable bit, which is set, and bit8 of the sample interval is set to insure an initial non-zero sampling interval.

8. APPENDIX A FPDP Interface (AD8S-100DMA-FPDP only).

Model AD8S-100DMA-FPDP is a model AD8S-100DMA with an added FPDP/TM (Transmitter Master) function that can send data to an FPDP Receiver/Master and optional Receivers. The FPDP interface on the ADSFPDP-100DMA complies with the VITA 17-199x Rev 1.7 November 24, 1988 FPDP specification, with the exception that it does not support the SUSPEND* signal, and therefore will not save up data if a receiver asserts the SUSPEND* signal.

The figure below indicates the orientation of the FPDP connector on the top of the FPDP/AD100 Mezzanine board. Pin 1 is shown by the black triangle at the left end of the FPDP connector. The FPDP connector uses the 5-volt signaling environment and FPDP non-inverted connector pin assignments, as specified in Table 2 of the Rev 1.7 FPDP specification. Both the TTL STROB signal and the PECL PSTROBE and PSTROBE* signals are outputted simultaneously. Termination of PSTROBE and PSTROBE* is by Method 1 (330 ohm pull downs on both outputs), although the AD8S-100DMA-FPDP may optionally be ordered with Method 2 termination.



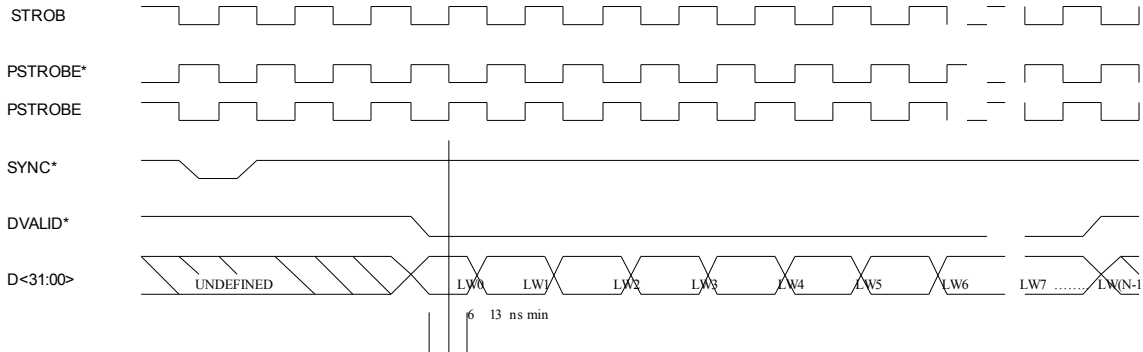
The FPDP interface on the AD8S-100DMA-FPDP implements the Single Frame Data Type, and has the timing shown in the figure on the following page:

The FPDP interface may be used alone or in combination with the board's normal PCI DMA operation. **To run the board so that data is output on both the FPDP interface and via PCI bus DMA, open the board and set up the board parameters** (eg. run `uvpci_set_doublespeed()`, set `SAMPINT = 1`) and then use the command `uvpci_set_go()` to set the board running. Data will begin being acquired and DMA'd to the PCI bus beginning when the TRIGGER input signal is brought high, even momentarily. Data will also begin being outputted on the FPDP interface when the FPDP NRDY* signal is de-asserted (brought high) (and TRIGGER is high).

The FPDP SYNC* signal is asserted within 10 clock cycles after the board begins acquiring data. DVALID* is asserted 3 clocks later, along with valid data, which continues to be outputted until either the board is told, via software, to stop acquiring data or the NRDY* signal is asserted (brought low), whichever occurs first, at which time DVALID* is de-asserted.

To run the board so that data is output on only the FPDP interface (with no PCI bus DMA), first open the board and set up the board parameters (eg. run `uvpci_set_doublespeed()`, set `SAMPINT = 1`), but **do not** run the command `uvpci_set_go()`. Data will begin being outputted via the FPDP interface when the FPDP NRDY* signal is de-asserted (brought high) and TRIGGER is brought high. Note that the board must not be closed until you are finished outputting data via the FPDP interface, to prevent the board parameters from being reset to their defaults. If the board is closed (using the command `uvpci_close()`) while FPDP data is being outputted, the data may

appear approximately correct, but will in fact be wrong, as the doublespeed mode will be cancelled if the board is closed. For this reason, it is essential to 1) Open the board, 2) Set the board parameters 3) De-assert the NRDY* signal to the FPDP interface, and assert TRIGGER to begin outputting FPDP data, and 4) Wait until you have finished outputting the desired amount of FPDP data before you 5) close the board. The sample program `acquire_fpdp_only.c` illustrates this sequence required when it is desired to output 100 megasample/second 8-bit data only via the FPDP port, without having the ULTRADMA board do any DMA to the bus.



The two TTL outputs PIO1 and PIO2 can be used to control various functions on FPDP receivers. To program these two lines to any desired values, write to control register bits OSDAT and OSCLK respectively, using the `uvpci_set_serial(...)` command under the Solaris or Linux64 OS.

Finally, to transfer data via normal DMA, without outputting data via the FPDP port, merely run the board as you normally would (see example programs such as `fast_acquire_dma_data.c`), and disable the FPDP port by merely not deasserting NRDY* (leave NRDY* low). Alternatively, you may even leave NRDY* de-asserted and ignore the FPDP data that is being outputted.

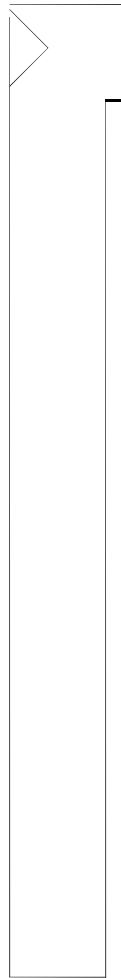
9. APPENDIX B. FRONT-END MEZZANINE BOARD INTERFACE

The ULTRADMA board series is available in two main configurations:

- 1) The standard configuration ULTRADMA boards, which have A/D and/or D/A converters installed on the board, and do not therefore need a mezzanine interface.
- 2) The ULTRADDMA12CAR board, which has a connector P3, which is the Front-end Mezzanine Interface. This interface allows various mezzanine boards to be installed on the ULTRADDMA-12CAR board for implementing custom front-end options or standard options, such as the ULTRADMA parallel TTL I/O interface. These front-end options are in lieu of the standard 12-bit A/D and D/A converters that are normally configured with the ULTRADMA. In some cases OEM customers may design mezzanine boards of their own which will mount on the ULTRADMA-12CAR, and thereby implement a custom function.

9.1 Front-end Mezzanine Interface Pinout

The following is the pinout of connector P3, as viewed from the front (component side) of the ULTRADMA board: Outputs from the ULTRADMA are shown as (O), inputs are shown as (I).



	PIN:
/DA (O)	1
/PBOE (O)	3
/TSRESET	5
/MCLK (O)	7
MCLK (O)	9
DD31 (I/O)	11
DD29 (I/O)	13
DD27 (I/O)	15
DD25 (I/O)	17
-5.5V @ 100mA (O)	19
DD23 (I/O)	21
DD21 (I/O)	23
DD19 (I/O)	25
DD17 (I/O)	27
-5.5V @ 100mA (O)	29
DD15 (I/O)	31
DD13 (I/O)	33
DD11 (I/O)	35
DD09 (I/O)	37
+5.5V @ 250mA (O)	39
DD07 (I/O)	41
DD05(I/O)	43
DD03 (I/O)	45
DD01 (I/O)	47
+5.5V@250mA (O)	49
NOT USED (RESV'd)	51
OSSTB (O)	53
OSCLK (O)	55
NOT USED	57
AD12LE0 (O)	59

PIN
2 GROUND
4 ADCNT (O)
6 GROUND
8 GROUND
10 GROUND
12 DD30 (I/O)
14 DD28 (I/O)
16 DD26 (I/O)
18 DD24 (I/O)
20 GROUND
22 DD22 (I/O)
24 DD20 (I/O)
26 DD18 (I/O)
28 DD16 (I/O)
30 GROUND
32 DD14 (I/O)
34 DD12 (I/O)
36 DD10 (I/O)
38 DD08 (I/O)
40 GROUND
42 DD06 (I/O)
44 DD04 (I/O)
46 DD02 (I/O)
48 DD00 (I/O)
50 TRIGGER (I/O)
52 GROUND
54 OSDAT (O)
56 GROUND
58 BO0 (O)
60 AD12LE1 (O)

10. APPENDIX C. User Library Commands for Linux64 & Solaris

The sample user programs in the Linux64 and Solaris user packages are compiled and linked with a user library. The usages of the common user library calls are illustrated in the example programs; a summary of the user library commands is below. Not all library calls and modes may be used with all types of Ultraview boards (for example, D/A operations shown below are only for boards which contain D/A converters, etc.). Unless otherwise specified, routines return 0 unconditionally. The example programs are an excellent reference for the use of each call.

10.1 Linux64 Library Commands

```
int uvpci_open(char *dev_name)
```

This command opens the Ultraview data acquisition board and the region of host memory in which the board will place its acquired data, via DMA, so that the user program can directly access it.

Parameter "dev_name" should be the name of the device. For the first board in a system, it will be /dev/uvpci/0, for the second board (if any), it will be /dev/uvpci/1, etc.

```
int uvpci_close(int fd)
```

Closes the board pointed to by fd.

```
int uvpci_set_go(int fd)
```

uvpci_set_go() starts the board running (sampling). Before you call this, be sure that all board parameters have been correctly set.

```
int uvpci_read(int fd, void *buffer, size_t size)
```

This command reads size bytes into the system memory pointed to by buffer from the Ultraview board pointed to by fd.

```
int uvpci_write(int fd, void *buffer, size_t size)
```

This command writes size bytes to the Ultraview board pointed to by fd from the system memory pointed to by buffer.

```
int uvpci_get_overruns(int fd)
```

Returns TRUE if an overrun occurred. Overruns will occur when the sample rate is too high for the speed with which the system can process the data.

```
int uvpci_set_da_mode(int fd)
```

Setting DA causes the board to operate in D/A mode instead of A/D mode.

```
int uvpci_set_wrap(int fd)
```

Setting wrap causes the board to sample continuously. When the RAM is filled with incoming samples, new samples wrap around to the beginning of the RAM. In this manner, the RAM acts as a large cyclic buffer,

ULTRAVIEW

with the address pointer moving up to the top of RAM and then wrapping around back to the beginning, until the driver stops the board.

int uvpci_unset_wrap(int fd)

Unsetting wrap causes the board to sample only until the RAM is filled to the top with incoming samples. At this point, the board automatically stops sampling.

int uvpci_set_sample_interval(int fd, int interval)

Sets the AD/DA sampling frequency.

int uvpci_set_double_speed(int fd)
int uvpci_unset_double_speed(int fd)

Set/Unset the double speed mode of operation.

int uvpci_set_ext_clock(int fd)
int uvpci_unset_ext_clock(int fd)

Select/Deselect use of the external clock.

int uvpci_set_ext_trigger(int fd)
int uvpci_unset_ext_trigger(int fd)

Enable/Disable the use of the external trigger.

ulong uvpci_get_block_size(int fd)

Returns the number of DMA blocks of memory on the board.

ulong uvpci_get_n_int_per_board(int fd)

Returns the number of DMA interrupts required to fill the entire board memory.

int uvpci_start_dma_at_block_n(int fd, int n)

Sets the onboard memory address pointer to start the next DMA operation at the beginning of block n.

ulong uvpci_get_board_blocks(int fd)

Returns the number of board blocks synthesized/acquired from the onboard memory.

ulong uvpci_get_dma_blocks(int fd)

Returns the number of DMA blocks transferred to/from system memory.

int uvpci_info(int fd)

uvpci_info() prints to stderr most of the information the driver knows about the board's state. Use sparingly; at high sample rates calling uvpci_info() may interfere with smooth operation of the board possible bus error or even a panic!. Use other library functions such as uvpci_get_overruns() or uvpci_read_ifr() instead.

int uvpci_stop(int fd)

ULTRAVIEW

This stops the board as soon as possible. It DOES NOT WAIT until the end of a block.

```
int uvpci_stop_at_end_of_block(int fd)
```

uvpci_stop_at_end_of_block() causes the board to stop at the end of acquiring the current block of data.

```
int uvpci_stop_at_n_blocks(int fd, int n)
```

uvpci_stop_at_n_blocks() causes the board to stop after acquiring n blocks of data.

```
int uvpci_stop_at_n_blocks_da(int fd, int n)
```

10.2 Solaris Library Commands

```
int uvpci_dma_open(char *dev_name, uint32_t buffer_blocks)
```

This command opens the Ultraview data acquisition board and the region of host memory in which the board will place its acquired data, via DMA, so that the user program can directly access it.

Parameter "dev_name" should be the name of the device. For the first board in a system, it will be /dev/uvpci/0, for the second board (if any), it will be /dev/uvpci/1, etc.

Parameter "buffer_blocks" is the desired size, in 512KB blocks, of the host memory buffer region where the board is to place its data. For example, if the buffer region is to be 128M, then buffer_blocks should be specified as 256.

This routine returns a file descriptor (eg fd or board_fd) if successful, or returns -1 if unsuccessful at opening the board and allocating the requested memory buffer.

```
int uvpci_close(int fd)
```

This command closes the Ultraview data acquisition board and the region of host memory the board had used to place its acquired data. fd is the file descriptor originally returned by uvpci_dma_open.

```
int uvpci_set_ext_clock(int fd)
```

Command uvpci_set_ext_clock forces use of external clock. Before using it, be sure that a continuous external clock, of correct amplitude and frequency is connected to the board's external clock input.

```
int uvpci_unset_ext_clock(int fd)
```

This command forces the board to use its internal clock, and not an external clock. To avoid noise pickup, if this command is use, no external clock should be attached to the board.

```
int uvpci_set_double_speed(int fd)
```

This call turns on the double speed bit, so the a/d converters sample only one a/d channel, at twice the maximum speed. This call must be used with caution, as the signal-to-noise ratio of doublespeed operation is not as good as for two channels operating simultaneously. **It may only be used when the sample interval is set to 1 for model AD8-100DMA or 2 for other models of board.**

```
int uvpci_unset_double_speed(int fd)
```

uvpci_unset_double_speed turns off double speed bit. This routine is normally not used, as the default is not to use double speed mode.

```
uvpci_set_interrupts(int fd) NOT USED IN NORMAL OPERATION
```

uvpci_set_interrupts turns on the interrupts. The use of this call is not necessary, and should normally never be made in the user program.

ULTRAVIEW

The driver automatically sets the board to use interrupts, so the use of this call need never be made.

int uvpci_unset_interrupts(int fd) NOT USED IN NORMAL OPERATION
uvpci_unset_interrupts prevents the board from issuing interrupts. **Users should never use this call**, as the driver will hang waiting for an interrupt that will never come. This forbidden call is described merely to make the user library documentation complete.

int uvpci_rt(int priority)

This function sets a real time priority from 0 to MAXRTPRI. See sample program "acquire_dma_data.c" for an example of setting to MAXRTPRI. If this call is unsuccessful, it returns -1. Normally, the board's large buffer ensures that data overruns will not occur regardless of the priority of the user program, and this call is not needed.

int uvpci_ts(int priority) NOT USED IN NORMAL OPERATION

This function sets a time sharing priority from 0 to maxtspri
Note: this can only be used to set the priority higher, or back to 0, it will not produce priorities lower than 0. Example uvpci_ts (UVPCI_TS_MAX). This call is dangerous, and can prevent other programs from running.

uvpci_set_adda_mode(int fd, int mode)

This routine sets the a/d and d/a (if any) mode of the driver. Mode is one of the following:

- UVPCI_AD_MODE - user wants a/d operation (not D/A)
(incoming a/d samples are put in memory,
a/d pointers are checked for overruns)
- UVPCI_DA_MODE - user wants d/a operation (not A/D)
(d/a samples are outputted from memory,
the d/a pointers are checked for overruns)
- UVPCI_DA_PERIODIC_MODE - user will repeatedly output data
(d/a samples are outputted from memory. NO
pointers are checked for overruns)

int uvpci_set_sampint(int fd, u_long sample_interval)

This routine sets the sample interval (time between samples), in boards in which there is an internal timebase. For models AD12-100DMA, ADDA12-100DMA, AD14-100DMA, and ADDA14-100DMA the parameter "sample_interval" is a number between 2 and 1023. For model AD8-100DMA, sample interval is between 1 and 1023. The relationship between sample_interval and sampling rate is given in the Sample Interval sections of this manual.

int uvpci_set_wrap(int fd)

Setting wrap causes the board to sample continuously (when later told to start). When the RAM buffer is filled with incoming samples, new samples wrap around to the beginning of RAM. In this manner, the RAM acts as a large cyclic buffer, with the address pointer moving up to the top of RAM and then wrapping around back to the beginning, etc, etc, until the board is stopped by the driver.

ULTRAVIEW

int uvpci_unset_wrap(int fd)

Unsetting wrap sets up the board to sample only until the RAM is filled to the top with incoming samples, and then automatically stop sampling.

int uvpci_block_size(int fd)

Returns block size in bytes. A block is the amount of A/D data that is placed into RAM before each interrupt. This routine is not normally needed, as the block size is always 512K bytes for all Ultraview ULTRADMA series boards.

int uvpci_da_reset(int fd) NOT NORMALLY NEEDED - FOR D/A OPERATION ONLY

Does a reset of the buffer info on the board; this also clears the entire D/A space. Returns a -1 if unsuccessful (due, for example, to the board being running when this command is issued).

int uvpci_set_go(int fd)

uvpci_set_go starts the board acquiring data. This command should only be run after setting the above parameters (wrap, sampint, etc).

int uvpci_stop_at_n_blocks(int fd, u_long n)

uvpci_stop_at_n_blocks causes the board to stop after acquiring n blocks of data. This call should be used immediately after running "uvpci_set_go" (see above). See the example program "fast_acquire_dma_data.c" for an example of a program that acquires n blocks, using this call.

int uvpci_stop_at_end_of_block(int fd)

uvpci_stop_at_end_of_block causes the board to stop at the end of acquiring the current block of data. This call is used as an alternate to uvpci_stop_at_n_blocks (see above), and is used after getting all of the pointers (see below). This call is useful if data is to be stopped after an external event, or at a certain time, rather than after a specific number of blocks.

int uvpci_stop(int fd) NOT USED IN NORMAL OPERATION

This harsh command stops the board as soon as possible. It DOES NOT WAIT until the end of a block. If DMA is still running, it returns with value UVPCI_UNDEFINED. Otherwise, it returns with 0. Normally, this command is never used. Either uvpci_stop_at_n_blocks (to tell the driver to automatically stop after n blocks has been acquired) or uvpci_stop_at_end_of_block (to tell the driver to stop after the current block is finished) is all that is ever required to use the board.

int uvpci_get_go(int fd) NOT USED IN NORMAL OPERATION

Returns TRUE if board is running (i.e. uvpci_set_go has been invoked, and the board has not yet been stopped, either directly, or due to the requested number of blocks (from uvpci_stop_at_n_blocks) having been completely transferred. Returns FALSE if board is not running. This call is unnecessary, as board will always be running if it has been

ULTRAVIEW

told to go, and has not been told to stop, and has not stopped due to n blocks expiring (if uvpci_stop_at_n_blocks was invoked).

uint32_t * uvpci_ad_ptr(int fd)

This command, used while the board is doing an A/D transfer, returns a pointer to the beginning of the next a/d block in which newly acquired A/D data resides in host memory. This will result in the block being marked as available to the user (and unavailable to the driver). This routine will wait until a block is available from the driver. The block pointed to must be returned to the driver using uvpci_rel_ad_ptr() after it has been filled! See example program "fast_acquire_dma_data.c" for an example of getting a pointer for each block of A/D data as it is acquired, using this pointer to tell the system where to read the data, and then releasing the pointer after it has been used.

int uvpci_rel_ad_ptr(int fd)

This call releases the last block obtained for the user from the driver with the above-described uvpci_ad_ptr() call. A side effect is that the block is marked as available to the driver, and unavailable to the user. If this command is unsuccessful, it will return with a value of UVPCI_UNDEFINED (see header files).

uint32_t * uvpci_da_ptr(int fd)

This call, used during D/A transfer, returns a pointer to the next d/a block in host memory that the user can write more d/a data into. Be sure to release the block using uvpci_rel_da_ptr(), so the block will be output by the driver when the appropriate interrupt happens. If the board is not running, and the buffer is full, this routine will not return a pointer, but will instead return 0.

int uvpci_rel_da_ptr(int fd)

This call releases the d/a pointer that was supplied earlier by uvpci_da_ptr. This release must be done for each block that has been acquired using uvpci_da_ptr(), immediately after filling the block with d/a data; the driver will not output a block of d/a data that the user has not released. If this command is unsuccessful, it will return with a value of UVPCI_UNDEFINED (see header files).

int uvpci_get_overruns(int fd)

Get the number of overruns that have occurred since the last board reset -- an overrun occurs when the board is going, and writes incoming A/D data into a block that contains older A/D data, which has not yet been processed by the user. Overruns occur when the sample rate is too high for the speed with which the user can process the data. This routine is often unreliable, and users should instead look at the console window, where messages are reliably printed for every overrun.

int uvpci_get_n_blocks_output(int fd) NOT USED IN NORMAL OPERATION

uvpci_get_n_blocks_output() returns the number of D/A data blocks the board has converted and output since the last board reset or close. This call is normally not needed, as the user program can easily just count the number of da_ptrs it has gotten so far, anyway.

ULTRAVIEW

int uvpci_get_dma_running(int fd) NOT USED IN NORMAL OPERATION

Tells user program if DMA transfers from last blocks has not yet been completed. This call is normally not needed, as every time an A/D pointer is asked for, it will not be given to the user program until, DMA for that block (to host memory) has been completed.

int uvpci_set_serial(int fd, u_long rsstb_rsclk_rsdatt)

Sets the RSSTB, RSCLK, RSDAT TTL output lines. For example, if called as uvpci_set_serial(board_fd, 0x0000005), a TTL high will be output on TTL output line RSSTB, a TTL low will be output on RSCLK, and a high will be output on RSDAT. These lines are useful for controlling external apparatus, under software control of the user program.

int uvpci_read_ifr(int fd) NOT USED IN NORMAL OPERATION

This routine reads the control register bits, as defined in section 5.2 of this manual. This information is normally never used in a user program, as the board normally will automatically transfer the needed data as asked, and the user program will know of its acquisition status in the normal course of asking for pointers.

int uvpci_wait(int fd) NOT USED IN NORMAL OPERATION

uvpci_wait() is used to maintain synchronization between the driver and the user program. uvpci_wait() will return immediately if the user program may read (for a/d) or write (for d/a) the next block in sequence; if not, then uvpci_wait() will return after the next board interrupt, at which time a block is available. This routine is superfluous, as merely asking for the next A/D data pointer will automatically cause the driver to wait until the pointer is available, anyway. None of the sample user programs supplied by Ultraview use either uvpci_wait or uvpci_timed_wait.

int uvpci_timed_wait(int fd, uint64_t usec_max_wait) NOT USED IN NORMAL OPERATION

uvpci_timed_wait() same as uvpci_wait() with a maximum wait time in usecs.

int uvpci_info(int fd) NOT USED IN NORMAL OPERATION

uvpci_info() prints to stderr most of the information the driver knows about the board's state. Use sparingly; at high sample rates calling uvpci_info() may interfere with smooth operation of the board (possible bus error or even a panic!). Use other library functions such as uvpci_get_go() or uvpci_read_ifr() instead.

11. APPENDIX D. Known Bugs

There is a known bug in the Linux device driver. Under the Linux operating system the device driver fails to correctly transfer the first block of data, resulting in inaccurate data in the first block acquired, all other blocks contain correct data. The workaround is to disregard the first block of data. This bug does not affect any other operating systems and is an issue with the device driver not the device itself.