cPCI-7300

PCI-7300A Rev.B

40MB Ultra-High Speed 32 I/O

(C/C++ & DLL Library)

Manual Rev 2.01: September 3, 1999

Trademarks

NuDAQ®, NuIPC®, cPCI-7300, PCI-7300A is registered trademarks of ADLink Technology Inc., IBM PC is a registered trademark of International Business Machines Corporation. Intel is a registered trademark of Intel Corporation. Other product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

Contents

How to Use This Manual

This manual is designed to help you use the cPCI-7300 and PCI-7300A Rev.B. The manual describes how to modify various settings on the PCI-7300A card to meet your requirements. It is divided into five chapters:

- Chapter 1, "Introduction", gives an overview of the product features, applications, and specifications.
- Chapter 2, "Installation", describes how to install the PCI-7300A. The layout of PCI-7300A is shown, and the installation procedures, pin assignment of connectors, and timer pacer generation are specified.
- Chapter 3, "Register Structure & Format", describes the low-level register structure and format of the PCI-7300A.
- Chapter 4, "Operation Theorem", describes how to use the operations of digital input and output on the PCI-7300A.
- Chapter 5, "C/C++ & DLL Library", describes the high level C and DLL library functions. It will help you to programming in DOS, Win 3.11, Win-95 and Win-NT environments.
- Appendix A, "8254 Programmable Interval Timer“, describes the detailed structure and register format of 8254 timer/counter chip.

1

Introduction

The PCI-7300A is PCI form factor ultra-high speed digital I/O card, it consists of 32 digital input or output channel. High performance designs and the state-of-the-art technology make this card to be ideal for high speed digital input and output applications.

The PCI-7300A performs high-speed data transfers using bus mastering DMA and scatter/gather via 32-bit PCI bus architecture. The maximum data transfer rates can be up to 40MB per second. It is very suitable for interface between high speed peripherals and your computer system.

The PCI-7300A is configured as two ports, PORTA and PORTB, each port controls 16 digital I/O lines. The I/O can configure as either input or output, and 8-bit or 16-bit. According to outside device environment, users can configure PCI-7300A to meet all high speed digital I/O data transfer.

There are many different digital I/O operation modes are supported:

1. Internal Clock: the digital input and output operations are paced by internal clock and transferred by bus mastering DMA.

2. External Clock: the digital input operation is paced by external strobe signal (DIREQ) and transferred by bus mastering DMA.

3. Handshaking: through REQ signal and ACK signal, the digital I/O data can have simple handshaking data transfer.

4. Pattern Generation: You can output a digital pattern repeatedly at a predetermined rate. The transfer rate is controlled by internal timer.

Software Supporting :

There are several software options help you get your applications running quickly and easily.

1. Linking with data acquisition software packages, such as :

. LabVIEW, DASYLab, etc

2. Custom Program:

For the customer who are writing their own programs, the PCI-7300A is supported by a comprehensive set of drivers and programming tools. These software supports are available in multiple platforms.

. MS-DOS Borland C/C++ programming library.

. DLL: Dynamic Linking Library for Win-95/98
(The above libraries are shipped with the board.)

. PCIS-DASK: Dynamic Linking Library for Win-NT/98 (optional included in software package)

. DAQBench/NT: ActiveX Controls Class library

1.1 Applications

- Interface to high-speed peripherals
- High-speed data transfers from other computers
- Automated test equipment(ATE)
- Electronic and logic testing
- Interface to external high-speed A/D and D/A converter
- Digital pattern generator
- Waveform and pulse generation
- Parallel digital communication

1.2 Features

The PCI-7300A Ultra-High Speed DIO card provides the following advanced features:

- 32 digital input/output channels

- Extra 4-bit TTL digital input and output channels
- Transfer up to 40M Bytes per second
- SCSI active terminator for high speed and long distance data transfer
- 32-bit PCI bus
- Plug and Play
- On-board internal clock generator
- Internal timer/external clock controls input sampling rate
- Internal timer control digital output rate
- ACK and REQ for handshaking
- On-board 64KB FIFO
- 100-pin SCSI style connector
- TRIG signal controls start of data acquisition/pattern generation

1.3 Specifications

♦ **Digital I/O (DIO)**

- **Channel:** 32 TTL compatible inputs and outputs
- **Device:** IDT 74FCT373
- **Input Voltage:**
  Low: Min. 0V; Max. 0.8V
  High: Min. +2.0V
- **Input Load:**
  Terminator OFF:
  Low: +0.5V @ ±1μA
  High: +2.7V @ ±1μA max.
  Terminator ON:
  Low: +0.5V @ 22.4mA
  High: +2.7V @ ±1μA max.
- **Output Voltage:**
  Low: Min. 0V; Max. 0.5V
  High: Min. +2.7V
- **Driving Capacity:**
  Low: Max. +0.5V at 48mA (Sink)
  High: Min. 2.4V at -8 mA (Source)
- **Hysteresis:** 500mV
- **Max. Transfer rate:**
  Max sustained transfer rate: 40M Bytes/s

Max burst transfer rate: 132M Bytes/s (PCI burst rate)

♦ **Transfer Characteristic**
- **Mode:** Bus Mastering DMA with Scatter/Gather
- **Data Transfers:** 8/16/32-bit input or output (programmable)

♦ **Programmable Counter**
- **Device:** 82C54-10
- **Digital Input Pacer:** Timer0
- **Digital Output Pacer:** Timer1
- **General Purpose Timer:** Timer2

♦ **General Specifications**
- **Connector:** one 100-pin male SCSI-II style cable connector
- **Operating Temperature:** 0° C ~ 50°C
- **Storage Temperature:** -20° C ~ 80°C
- **Humidity:** 5 ~ 95%, non-condensing
- **Dimension:** Compact size only 98mm(H) X 147mm(L)
- **Power Consumption:**

  +5 V @ 1.5A max. with on-board terminator off

  or

  +5 V @ 1.0A max. with on-board terminator on

  or

  +5 V @ 2.0A max. with on-board terminator on and external device's terminator on:

2

Installation

This chapter describes how to install the PCI-7300A. At first, the contain in the package and unpacking information that you should be careful are described. Because the PCI-7300A is following the PCI design philosophy, it is no more jumpers and DIP switches setting for configuration. The Interrupt and I/O port address are the variables associated with automatic configuration, the resource allocation is managed by the system BIOS. Upon system power-on, the internal configuration registers on the board interact with the BIOS.

2.1 What You Have

In addition to this *User's Manual*, the package includes the following items:

- PCI-7300A 40MB Ultra-High Speed 32 I/O Card
- Manual & Software Utility CD

If any of these items is missing or damaged, contact the dealer from whom you purchased the product. Save the shipping materials and carton in case you want to ship or store the product in the future.

2.2 Unpacking

Your PCI-7300A card contains sensitive electronic components that can be easily damaged by static electricity.

The card should be done on a grounded anti-static mat. The operator should be wearing an anti-static wristband, grounded at the same point as the anti-static mat.

Inspect the card module carton for obvious damage. Shipping and handling may cause damage to your module. Be sure there are no shipping and handling damages on the module before processing.

After opening the card module carton, extract the system module and place it only on a grounded anti-static surface component side up.

Again inspect the module for damage. Press down on all the socketed IC's to make sure that they are properly seated. Do this only with the module place on a firm flat surface.

Note : DO NOT APPLY POWER TO THE CARD IF IT HAS BEEN DAMAGED.

You are now ready to install your PCI-7300A.

2.3 Device Installation for Windows 95/98

While you first plug PCI-7300A card and enter Windows 95/98, the system will detect this device automatically. Please follow the steps to install the device.

1. Click the Next button in the Update Device Driver Wizard window, Win95 will start to search floppy drive A for the PCI-7300A driver information, After fail to find the information in drive A, it will display the message "Windows was unable to locate a driver for this device."
2. Insert ADLink's All-in-one into CD-ROM drive.
3. Click "Other Location…" button in the Update Device Driver Wizard Window, then the Select Other Location windows will appear.
4. Click Browse button to invoke the Browser for Folder window, then select the location

   X*:\Win95Inf\7300** (X*** indicates the CD-ROM drive).

2.4 PCI-7300A's Layout

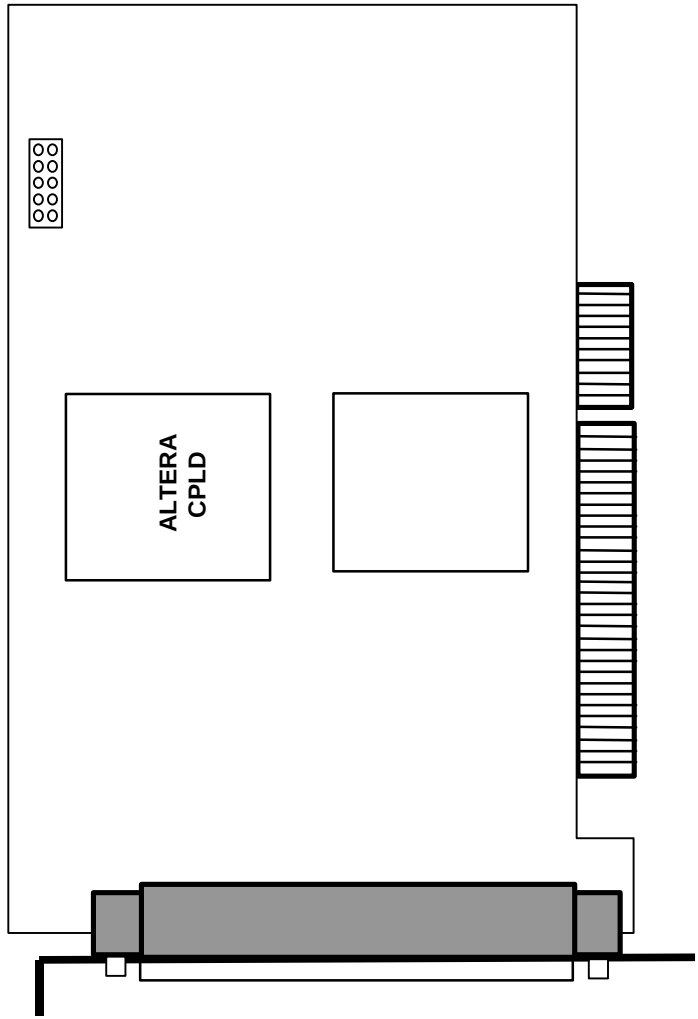


Figure 2.1 PCI-7300A Layout Diagram

2.5 PCI-7300A Installation Outline

2.5.1 Hardware configuration:

1. Plug and Play :

As a plug and play component, the interrupt level is set by PCI plug and play BIOS and saved in the PCI controller. The system BIOS assigns the interrupt level based on the board information and on known system parameters. These system parameters are determined by the installed drivers and the hardware load seen by the system.

2. Configuration :

The board configuration is done on a board-by-board basis for all PCI form factor boards on your system. Because configuration is controlled by the system and software, so there is no jumper for base-address, DMA, and interrupt IRQ need to be set by the user.

The configuration is subject to change with every boot of the system as new boards are added or boards are removed. So, there is no idea what‘s going on to be installed.

3. Trouble shooting :

If your system won‘t boot or if you experience erratic operation with your PCI board in place, it’s likely caused by an interrupt conflict (perhaps because you incorrectly described the ISA setup). In general, the solution, once you determine it is not a simple oversight, is to consult the BIOS documentation that come with your system.

2.5.2 PCI slot selection

Your computer will probably have both PCI and ISA slots. Do not force the PCI-7300A into a PC/AT slot. Also, make sure the PCI slot can support bus master mode when you plug in the PCI-7300A.

2.5.3 Installation Procedures

1. Turn off your computer

2. Turn off all accessories (printer, modem, monitor, etc.) connected to computer.

3. Remove the cover from your computer.

4. Select a 32-bit PCI expansion slot. PCI slot are short than ISA or EISA slots and are usually white or ivory.

Caution !! Don't put PCI-7300A card into ISA or EISA card.

5. Before handling the PCI-7300A, discharge any static buildup on your body by touching the metal case of the computer. Hold the edge and do not touch the components.

6. Position the board into the PCI slot you selected.

7. Secure the card in place at the rear panel of the system unit using screw removed from the slot.

2.5.4 Running the 7300AUTIL.EXE

PCI-7300A can do an automatic configuration of the IRQ, and I/O port address. By using the 7300AUTIL.EXE, you can get above values that are displayed in this utility. Also you can use this utility to check if your PCI-7300A can work properly.

2.6 Connector Pin Assignment

The PCI-7300A comes equipped with one 100-pin SCSI type connector (CN1) located on the rear mounting plate. The pin assignment of CN1 is illustrated in the figure 2.2.

Legend :

Pins	Signal Name	Signal Type	Signal Direction	Description
1…50	GND	POWER		Ground – these lines are the ground reference for all other signals
51..66	PB15…PB0	DATA	I/O	PortB bidirectional data liness-PB15 is the MSB, and PB0 is the LSB.
67	DOACK	CONTROL	I	Digital output Acknowledge lines – In handshaking mode, DOACK carries handshaking status information from the peripheral.
68	DOREQ	CONTROL	O	Request line – In handshaking mode, DOREQ carries handshaking control information to peripheral.
69	DOTRIG	CONTROL	I	DO TRIG- can be used to control the start of data output in all DO modes and to control the stop of pattern generation in pattern generation mode.
70…73	AUXDO3…0	DATA	O	AUX DO 3…0 – can be used as extra output data or can be used as extra control signals.
85..100	PA15…PA0	DATA	I/O	PortA bidirectional data liness-PA15 is the MSB, and PA0 is the LSB.
82	DIACK	CONTROL	O	Digital output Acknowledge lines – In handshaking mode, DIACK carries handshaking status information to the peripheral.
83	DIREQ	CONTROL	I	Request line – In handshaking mode, DOREQ carries handshaking control information from peripheral.
84	DITRIG	CONTROL	I	DI TRIG – can be used to control the start of data acquisition in all DI modes.
78…81	AUXDI3…0	DATA	I	AUX DI 3…0 – can be used as extra input data or can be used as extra control signals.
74…77	TERMPWR	POWER		TERMPWR -- 4.7V active terminator power output

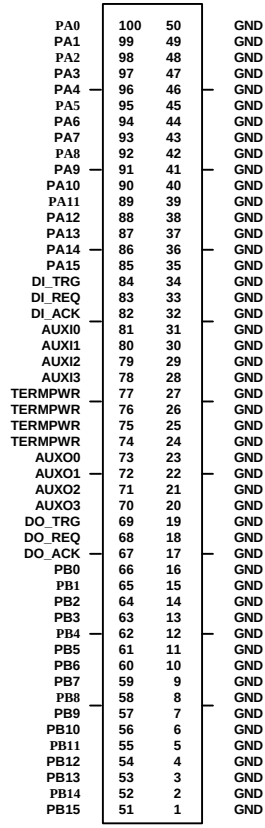


Figure 2.2 CN1 Pin Assignment

2.7 Wiring and Termination

Transmission line effects and environment noise, particularly on clock and control lines, can lead to incorrect data transfers if you do not take care when running signal wires to and from the devices.

Take the following precautions to ensure a uniform transformation line and minimize noise pickup:

1. Use twisted-pair wires to connect digital I/O signals to the device. Twist each digital I/O signal with a GND line. In PCI-7300A, 50 signals are used as GND.
2. Place a shield around the wires connecting digital I/O signal to device.
3. Route signals to the devices carefully. Keep cabling away from noise sources, such as video monitor.

For PCI-7300A, it is important to terminate your cable properly to reduce or eliminate signal reflections in the cable. The PCI-7300A support active terminator on board, you can enable or disable the terminator by software selection. This is a good way to include termination on the signal transmission.

Additional recommendations apply for all signal connection to your PCI-7300A:

1. Separate PCI-7300A device signal lines from high-current or high-voltage line. These lines are capable of inducing currents in or voltages on the PCI-7300A if they run in parallel paths at a close distance. To reduce the magnetic coupling between lines, separate them by a reasonable distance if they run in parallel, or run the lines at right angles to each other.
2. Do not run signal lines through conducts that also contain power lines.
3. Protect signal lines from magnetic fields.

3

Register Format

In this chapter, the register format of the PCI-7300A and cPCI-7300 is described. Please note that the register map of the PCI-7300A Rev.B is different from the PCI-7300A Rev.A

3.1 I/O Registers Format

The PCI-7300A occupies 8 consecutive 32-bit I/O addresses in the PC I/O address space. Table 4.1 shows the I/O Map of the PCI-7300A rev.B.

Address	Read	Write
Base + 0	DI_CSR	DI_CSR
Base + 4	DO_CSR	DO_CSR
Base + 8	AUX_DIO	AUX_DIO
Base + C	INT_CSR	INT_CSR
Base + 10	DI_FIFO	DI_FIFO
Base + 14	DO_FIFO	DO_FIFO
Base + 18	-	FIFO_CR
Base + 1C	POL_CTRL	POL_CTRL
Base + 20	8254_COUNT0	8254_COUNT0
Base + 24	8254_COUNT1	8254_COUNT1
Base + 28	8254_COUNT2	8254_COUNT2
Base + 2C	8254_CONTROL	8254_CONTROL

Legend:

DI_CSR : Digital Input Control & Status Register
DO_SCR : Digital Output Control & Status Register

AUX_DIO : Auxiliary Digital I/O port
INT_CSR : Interrupt Control and Status Register
DI_FIFO : DI FIFO direct access port
DO_FIFO : DO FIFO direct access port
FIFO_CR : FIFO Almost Empty/Full Programming Register
POL_CTRL : Polarity control register for the control signals

Caution:

1. I/O port is 32-bit width

2. 8-bit or 16-bit I/O access is not allowed.

3.1.1 DI_CSR: DI Control & Status Register (BASE + 0)

Digital input control and status checking is done by this register.

Address : BASE + 00

Attribute : READ/WRITE

Data Format :

Nipple	3	2	1	0
Base+0(L)	DI_HND_SHK	DI_CLK_SEL		DI_32
Base+0(H)	RESERVED	PA_TERM_OFF	DI_WAIT_TRIG	-- (1)
	DI_FIFO_FULL	DI_OVER	DI_FIFO_CLR	DI_EN
Base+0(L)	-	-	-	DI-FIFO_EMPTY

(1) This bit is different between Rev.A and Rev.B.

DI_32 (R/W)

0: Input port is not 32-bit wide (16-bit or 8-bit wide)

1: Input port is 32-bit wide, PORTB is configured as the extension of PORTA. That means PORTA is input lines (0…15), and PORTB is input lines (16…31). All PORTB control signals are disabled.

DI_CLK_SEL (R/W)

00 : use timer0 output as input clock
01 : use 20MHz clock as input clock
10 : use 10MHz clock as input clock
11 : use external clock (DI_REQ) as input clock

DI_HND_SHK (R/W)

0: No handshaking
1: REQ/ACK handshaking mode

DI_WAIT_TRIG (R/W)

0: delay input sampling until DITRIG is active

1: start input sampling immediately

PA_TERM_OFF (R/W)
0: PORTA terminator ON
1: PORTA terminator OFF

DI_EN (R/W)
0: Disable digital inputs
1: Enable digital inputs

DI_FIFO_CLR (R/W)
0: No effect
1: Clear digital input FIFO (If both PORTA and PORTB are configured as inputs, both FIFO will be cleared).

DI_OVER (R/W)
0: DI FIFO does not full during input sampling
1: DI FIFO full during input sampling, some input data was lost, write "1" to clear this bit

DI_FIFO_FULL (RO)
0: DI FIFO is not full
1: DI FIFO is full

DI_FIFO_EMPTY (RO)
0: DI FIFO is not empty
1: DI FIFO is empty

3.1.2 DO_CSR : DO Control & Status Register (BASE + 4)

Digital input control and status checking is done by this register.

Address : BASE + 04
Attribute : READ/WRITE
Data Format :

Bits	3	2	1	0
Bit 3-0	DO_WAIT_NAE	DO_MODE		DO_32
Bit 7-4	PG_STOP_TRIG	PB_TERM_OFF	DO_WAIT_TRG	PAT_GEN
Bit 11-8	DO_FIFO_FULL	DO_UNDER	DO_FIFO_CLR	DO_EN
Bit 15-12	-	-	BURST_HNDSH (2)	DO_FIFO_EMPTY

(2) This bit is different between Rev.A and Rev.B.

DO_32 (R/W)
0: Output port is not 32-bit wide (16-bit or 8-bit wide)

1: Output port is 32-bit wide, PORTA is configured as the extension of PORTB. That means PORTB is output lines (0…15), and PORTA is output lines (16…31). All PORTA control signals are disabled.

DO_MODE (R/W)

00 : use timer1 output as output clock
01 : use 20MHz clock as output clock
10 : use 10MHz clock as output clock
11 : REQ/ACK handshaking mode

DO_WAIT_NAE (R/W)

1: do not wait output FIFO not almost empty flag
0: delay output data until FIFO is not almost empty

PAT_GEN(R/W)

0: pattern generation disable (FIFO data do not repeat during data output)
1: pattern generation enable (FIFO data repeat themselves during data output)

DO_WAIT_TRIG (R/W)

0: delay output data until DOTRIG is actived
1: start output data immediately

PB_TERM_OFF (R/W)

0: PORTB terminator ON
1: PORTB terminator OFF

PG_STOP_TRIG (R/W)

0: no effect
1: Stop pattern generation when DOTRIG is deasserted

DO_EN (R/W)

0: Disable digital outputs
1: Enabled digital outputs

DO_FIFO_CLR (R/W)

write “1” to clear the DO_FIFO. If PORTA and PORTB are both configured as output ports. Both FIFOs are cleared. Always get 0 when read.

DI_UNDER (R/W)

0: DO FIFO does not empty during data output
1: DO FIFO is empty during data output, some output data may be output twice. Write 1 to clear this bit

DO_FIFO_FULL (RO)
0: DO FIFO is not full
1: DI FIFO is full

DO_FIFO_EMPTY (RO)
0: DO FIFO is not empty
1: DO FIFO is empty

BURST_HNDSHK (R/W)
0: disable burst handshaking mode
1: enable burst handshake mode

Note: This bit is for Rev.B only.

3.1.3 Auxiliary Digital I/O Register (BASE + 08)

Auxiliary 4-bit digital inputs and 4-bit digital outputs

Address : BASE + 08
Attribute : READ/WRITE
Data Format :

Bits	3	2	1	0
Bit 3-0	DO_AUX_3	DO_AUX_2	DO_AUX_1	DO_AUX_0
Bit 7-4	DI_AUX_3	DI_AUX_2	DI_AUX_1	DI_AUX_0

This auxiliary digital I/O is controlled by porgram I/O only.

DO_AUX_3 ~ DO_AUX_0 (R/W)
4-bit auxiliary output port. Porgram I/O only.

DI_AUX_3 ~ DI_AUX_0 (R)
4-bit auxiliary input port. Program I/O only

3.1.4 INT_CSR : Interrupt Control and Status Register (BASE + 0x 0C)

The interrupt of PCI-7300A is controlled and status is checked through this register.

Address : BASE + 0x0C
Attribute : READ/WRITE
Data Format :

Bits	3	2	1	0
Bit 3-0	T2_INT	AUXIO_INT	T2_EN	AUXDI0_EN
Bit 7-4	-	-	Reserved	Reserved

AUXDI_EN (R/W)
0: Disable AUXDI0 interrupt
1: Interrupt CPU on falling edge of AUXDI0

T2_EN (R/W)
0 : Disable Timer2 interrupt
1: Interrupt CPU on falling edge of Timer 2 output

AUXDI0_INT (R/W)
0: AUXDI does not generate interrupt
1: AUXDI interrupt occurred. Write "1" to clear

T2_EN (R/W)
0: Timer 2 does not generate interrupt
1: Timer 2 interrupt occurred. Write "1" to clear

3.1.5 DI_FIFO : DI FIFO direct access port (BASE + 0x 10)

The digital input FIFO data can be accessed through this port directly.

Address : BASE + 0x10
Attribute : READ/WRITE
Data Format :

Bits	7	6	5	4	3	2	1	0
Bit 7~0	DI_FIFO_8							
Bit 15~8	DI_FIFO_16							
Bit 31_16	DI_FIFO_32							

DI_FIFO_8

Bit 7 ~ Bit 0 of digital input FIFO

DI_FIFO_16

Bit 15 ~ Bit 8 of digital input FIFO if the digital input is configured as 16-bit wide or 32-bit wide.

DI_FIFO_32

Bit 31 ~ Bit 16 of digital input FIFO if the digital input is configured as 32-bit wide

---

Note : Although this port is R/W port, write operation should be avoided in normal operation. If both PORT A and PORT B are configured as output ports, read/write to this port is meaningless.

3.1.6 DO_FIFO : DO external data FIFO direct access port (BASE + 0x14)

The digital output FIFO data can be accessed through this port directly.

Address : BASE + 0x0C
Attribute : READ/WRITE
Data Format :

Bits	7	6	5	4	3	2	1	0
Bit 7~0	DO_FIFO_8							
Bit 15~8	DO_FIFO_16							
Bit 31_16	DO_FIFO_32							

DO_FIFO_8

Bit 7 ~ Bit 0 of digital output FIFO

DO_FIFO_16

Bit 15 ~ Bit 8 of digital output FIFO if the digital output is configured as 16-bit wide or 32-bit wide.

DO_FIFO_32

Bit 31 ~ Bit 16 of digital output FIFO of the digital output is configured as 32-bit wide

---

Note : Although this port is R/W port, read operation should be avoided in normal operation. If both PORTA and PORTB are configured as input ports, read/write to this port is meaningless.

3.1.7 FIFO_CR : FIFO almost empty/full programming register (BASE+ 0x18)

The register is used to control the FIFO programmable almost empty/full flag.

Address : BASE + 0x018
Attribute : READ/WRITE
Data Format :

Bits	7	6	5	4	3	2	1	0
Bit 15~0	PB_PAE_PAF							
Bit 31_16	PA_PAE_PAF							

PB_PAE_PAF (WO)

Programmable almost empty/full threshold of PORTB FIFO, 2
consecutive writes are required to program PORTB FIFO. Programmable almost empty threshold first.

PA_PAE_PAF(WO)

Programmable almost empty/full threshold of PORTA FIFO, 2
consecutive writes are required to program PORTA FIFO. Programmable almost empty threshold first.

3.1.8 POL_CNTRL: Control Signal Polarity Control Register (BASE + 0x 1C)

The register is used to control the control signals' polarity. The control signals include DI_REQ, DI_ACK, DI_TRG, DO_REQ, DO_ACK and DO_TRG. Please note that this register is for PCI-7300A Rev.B and cPCI-7300 only.

Address : BASE + 0x1C
Attribute : READ/WRITE
Data Format :

Bits	3	2	1	0
Bit 3-0	DO_REG_NEG	DI_TRG_NEG	DI_ACK_NEG	DI_REQ_NEG
Bits	**7**	**6**	**5**	**4**
Bit 7-4	-	-	DO_TRG_NEG	DO_ACK_NEG

DI_REQ_NEQ (R/W)
0: DI_REQ is rising edge active
1: DI_REQ is falling edge active

DI_ACK_NEQ (R/W)
0: DI_ACK is rising edge active
1: DI_ACK is falling edge active

DI_TRG_NEQ (R/W)
0: DI_TRG is rising edge active
1: DI_TRG is falling edge active

DO_REQ_NEQ (R/W)
0: DO_REQ is rising edge active
1: DO_REQ is falling edge active

DO_ACK_NEQ (R/W)
0: DO_ACK is rising edge active
1: DO_ACK is falling edge active

DO_TRG_NEQ (R/W)
0: DO_TRG is rising edge active
1: DO_TRG is falling edge active

4

Operation Theorem

Before you are ready to use the PCI-7300A, there are some basic operation modes shall be understood. This section provides detailed operation information for PCI-7300A, including Configuration, Clocking, Timing, Termination, Transfer mode, starting mode, etc.

The PCI-7300A is designed for high speed I/O, so only the strobe I/O, which is data transfer in regular timing or perform handshaking, is provided.

- **Digital I/O Data Path Configuration:**

The 32-bit I/O data path of PCI-7300A can be configured as 8-bit, 16-bit, or 32-bit, the possible configuration modes are listed as below.

Mode	Channel	Description
DI32	PORTA(DI0…DI15) PORTB(DI16..DI31)	Both PORTA and PORTB are configured as input channel
DO32	PORTA(DO16…DO31) PORTB(DO0…DO15)	Both PORTA and PORTB are configured as output channel
DI16DO16 (default mode)	PORTA(DI0…DI15) PORTB(DO0…DO15)	PORTA is 16-CH input PORTB is 16-CH output
DI16DO8	PORTA(DI0…DI15) PORTB(DO0…DO7)	PORTA is 16-CH input PORTB is 8-CH output
DI8DO16	PORTA(DI0…DI7) PORTB(DO0…DO15)	PORTA is 8-CH input PORTB is 16-CH output
DI8DO8	PORTA(DI0…DI7) PORTB(DO0…DO7)	PORTA is 8-CH input PORTB is 8-CH output

Notes:
PORTA is default as Input channel; PORTB is default as output channel.

In DI32 mode, the PORTB has to be configured as the extension of PORTA, that is, PORTB is the input port(DI16…DI31). PORTB control signals are disabled.
In DO32 mode, the PORTA has to be configured as the extension of PORTB, that is, PORTA is the output port(DO16…DO31). PORTA control signals are disabled.
DI0:input LSB, DI31:input MSB;DO0:output LSB, DO31:output MSB.

LSB: Least Significant Bit, MSB: Most Significant Bit

- ***Clocking Mode:***
  1. ***Internal Clock***: the digital input and output operations are handled by internal timer pacer trigger and transferred by bus mastering DMA. There are three timer sources which can trigger both input or output: 20MHz, 10MHz, and Time0 for input or Timer1 for output.

  2. ***External Clock***: this mode is only applied for digital input. the digital inputs are handled by external clock strobe(DIREQ) and transferred by bus mastering DMA.

  3. ***Handshaking***: through REQ input signal and ACK output signal, the digital I/O can have simple handshaking data transfer.

  4. ***Pattern Generator:*** This mode is only available for digital output only. The digital data is output to peripheral device based on the time signals occur at a constant rate and periodically.
  5. ***Burst Handshaking:***

- ***Starting Mode:***
  In addition, the digital I/O data transfer can be started by two modes, NoWait mode or TrigWait mode.

  1. ***NoWait:*** the data transfer is started immediately when a I/O transfer command is issued.

2. ***TrigWait:*** the data transfer is not started until external trigger (DITRIG for digital input, DOTRIG for digital output) is activated.

- **Termination:**

In PCI-7300A, the low capacitance active terminator is installed. You can enable or disable by software selection.

- **Data Transfer:**

Digital I/O data transfer between PCI-7300A and PC's main memory is through bus mastering DMA which is controlled by PCI bridge.

- **Stopping Mode:**

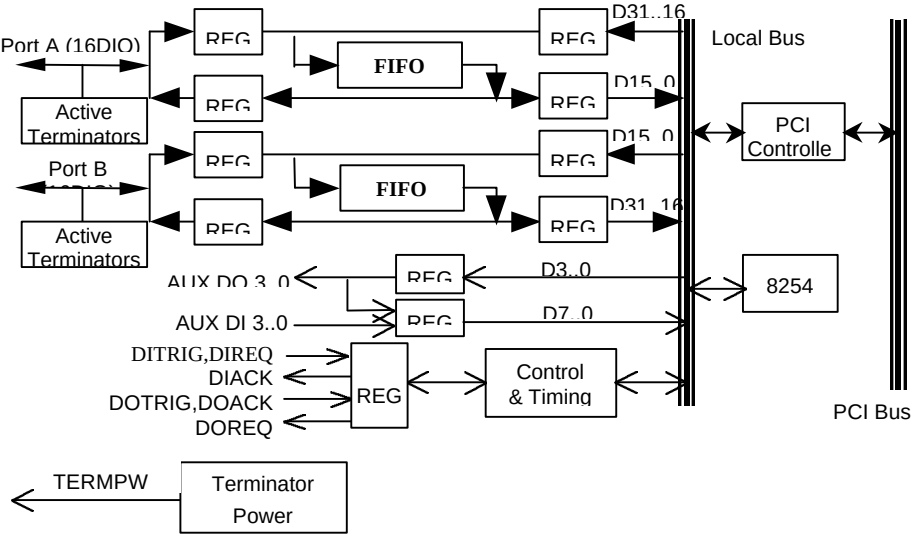
There are four modes to stop the digital I/O operation:

. Data Acquisition :
. Data Output:
. Pattern Generation:
. DMA Stop:

- **Pattern Generation:**

Digital output data transfer between PCI-7300A and external peripheral. The digital pattern can be stored in PC main memory or in PCI-7300A on-board FIFO if the length of pattern is less than or equal to 16K Double words.

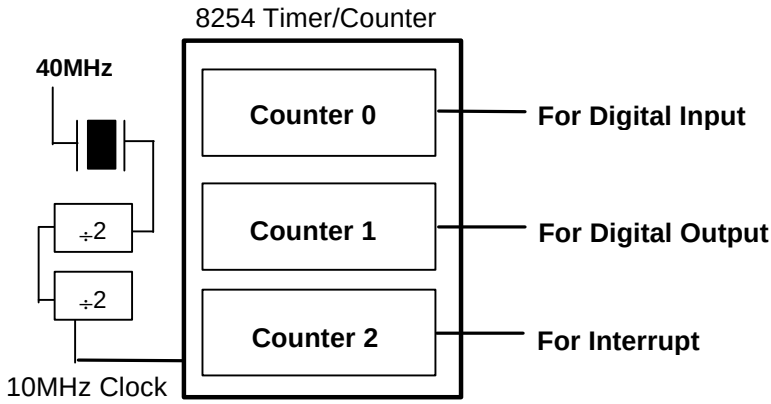
4.1 Function Block Diagram



PORTA :	16 Digital I/O Port, it can set as termination mode or non-termination
PORTB :	16 Digital I/O Port, it can set as termination mode or non-termination
FIFO :	Totally 64KB FIFO for digital I/O data buffer
AUX DO 3..0 :	four auxiliary digital outputs
AUX DI 3..0 :	four auxiliary digital inputs
DITRIG :	Digital input trigger line
DIACK/DIREQ :	Digital input handshaking signals
DOTRIG :	Digital output trigger line
DOACK/DOREQ :	Digital output handshaking signals
TERMPWR:	Terminator power output, provided for external active terminator

4.2 Internal Clock

The digital I/O data transfer can be controlled by internal clock which is generated by a 40MHz oscillator and programming timer/counter chip 8254. There are three counters on the 8254, the counter 0 is used to generate timer pacer for digital input, and counter 1 is used for digital output. The configuration is illustrated as below.



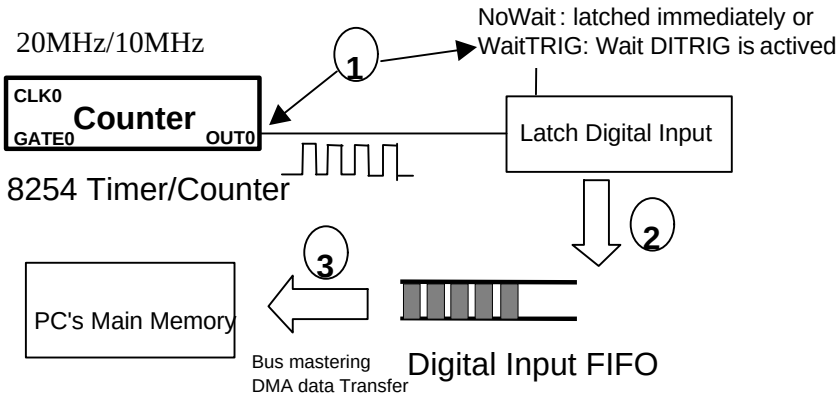
- **The operations sequence of digital input mode are:**

1. Define the input frequency (timer pacer rate), 20MHz, 10MHz, or controlled by 8254 timer/counter chip. Define the data transfer start mode is NoWait or TrigWait.

2. The digital input data are saved in FIFO after a DI command is issued, or waiting for DI_TRIG signal is actived. The sampling rate is controlled by timer pacer.

3. The data saved in FIFO will transfer to main memory of your computer system directly and automatically. This is controlled by bus mastering DMA control, this function is supported by PCI controller chip.

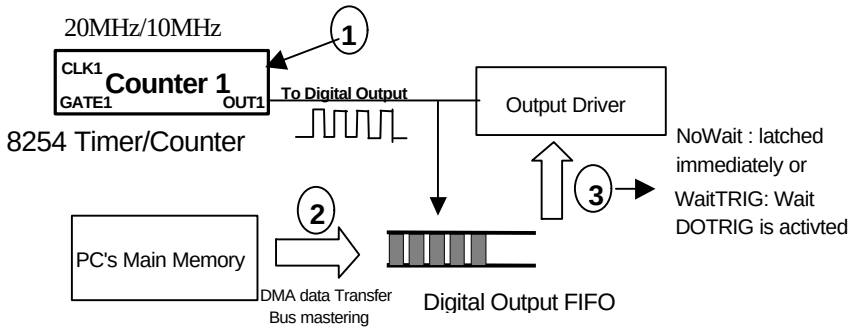
The operation flow is show as below:



- **The operations sequence of digital output mode are:**

1. Define the output frequency (timer pacer rate), 20MHz, 10MHz, or controlled by 8254 timer/counter chip; data transfer start mode either NoWait or TrigWait.
2. The output data saved in PC's main memory will be transferred to FIFO through the PCI controller chip.
3. The digital output data in output FIFO will be transferred to external device after a DO command is issued or DO_TRIG is activated. The output rate is controlled by timer pacer.

The operation flow is show as below:



4.3 External Clock

The digital input data transfer can be controlled by external strobe, which is from pin-83 DIREQ of CN1. The operation sequence is very similar to Internal Clock. Their difference is the clock source is from outside peripheral devices.

- **The operation sequences of digital input mode are:**

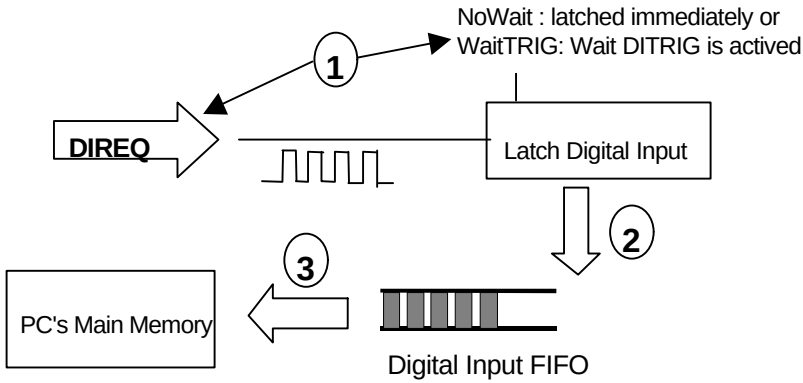
1. The external input strobe is generated from outside device, and go through the Pin 83 (DIREQ) of CN1 and to trigger the digital input operations. And, data transfer start mode is either NoWait or TrigWait.

2. If the Start Mode is TrigWait, the PCI-7300A will wait for the activation of DITRIG. When the DITRIG is activated or the start mode is NoWait, the input data is latched and them started to FIFO on every rising edge (or falling edge) of DIREQ.

3. The data stored in input FIFO will transfer to main memory on your computer system directly. This is controlled by bus

mastering DMA control, this function is supported by PCI controller chip.

4. The DIACK signal indicates the status of PCI-7300A on-board FIFO in external clock mode. When the digital input circuit of PCI-7300A is enabled and its FIFO is not almost full, the DIACK signal will remain asserted.



4.4 Handshaking

In PCI-7300A, it also supports a handshaking digital I/O transfer mode. That is, after input data is ready, a DIREQ is sent form external device, and DIACK will go high to acknowledge the data already accessed. In the same situation, the handshaking mode can be started by either NoWait or TrigWait mode. That is, the handshaking operation will be started either no waiting or waiting for DITRIG or DOTRIG signal is activated.

- **DIREQ & DIACK for Digital Input**

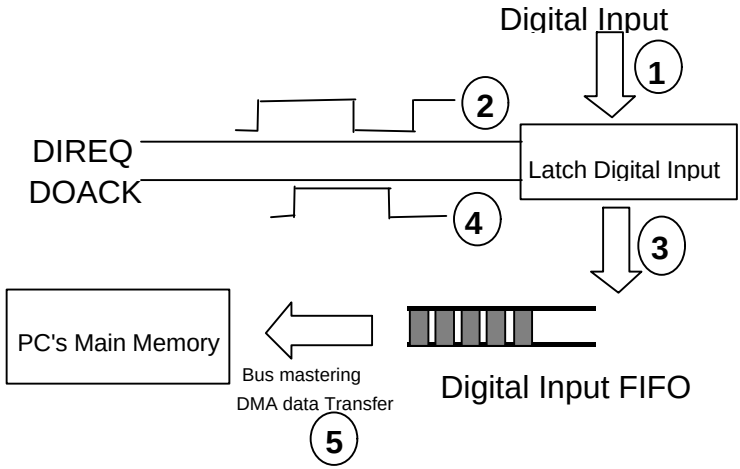
1. Digital Input data is ready (on device side)

2. The peripheral device can then strobe data into the PCI-7300A by assert a **DIREQ** signal, when the digital input data is ready

3. The **DIREQ** signal caused the PCI-7300A to latch digital input data and store it into FIFO

4. The PCI-7300A asserts a **DIACK** signal when it is ready for another input, the step 2 to step 4 will be repeated again.

5. If the FIFO is not empty and PCI bus is not occupied, the data will be transferred to main memory



- **DOREQ & DOACK for Digital Output**

1. Digital Output Data is moved from PC memory to FIFO of PCI-7300A by using DMA data mastering data transformation.

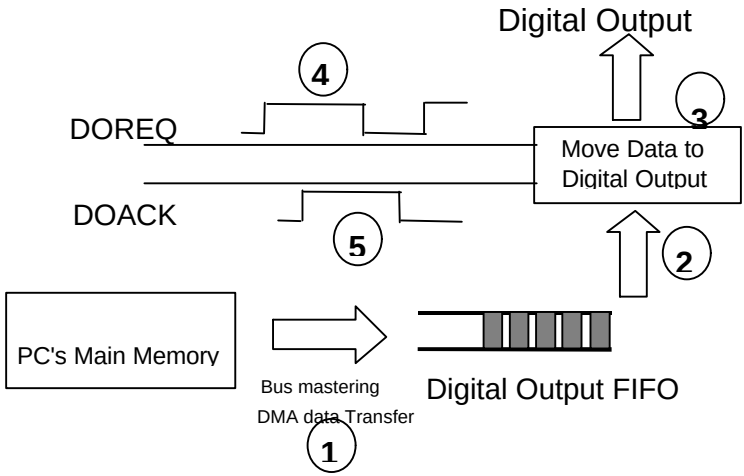
2. Move output data from FIFO to digital output circuit

3. Output data is ready

4. A **DOREQ** signal is generated and sent to outside device.

5. After a **DOACK** is got, the step 2 to step 5 will be repeated again.

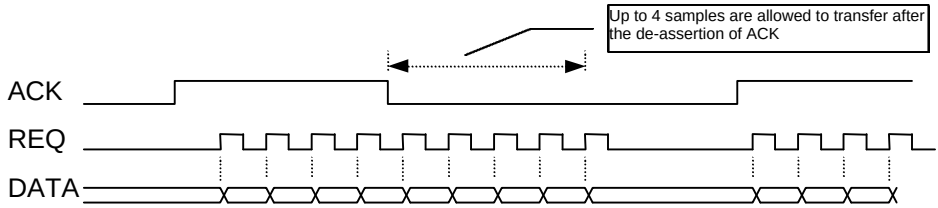
** if the FIFO is not full, the output data is moved form PC‘s main memory to FIFO automatically.



4.5 Burst Handshaking

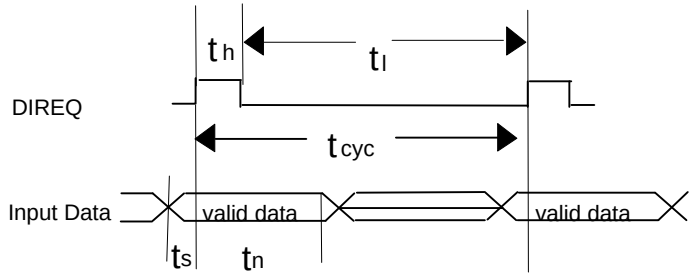
A new handshaking mode, burst handshaking, is address to the new version of PCI-7300A Rev.B. The burst handshaking mode is a fast and reliable data transfer protocol. It has both advantage of handshaking mode, which is reliable, and the advantage of internal / external clock mode, which is fast.

When using this mode, the sender has to check the availability of receiver indicated by the ACK signal before it starts to send data. Once the ACK is asserted, the receiver has to keep the ACK signal asserted before its input buffer becomes too small. When the ACK is de-asserted, indicating the receiver’s buffer has not much space for new data, the sender is still allowed to send 4 data to the receiver, and the receiver has to receive these data. The following figure illustrates the operation of the burst handshaking mode:



4.6 Timing Characteristic

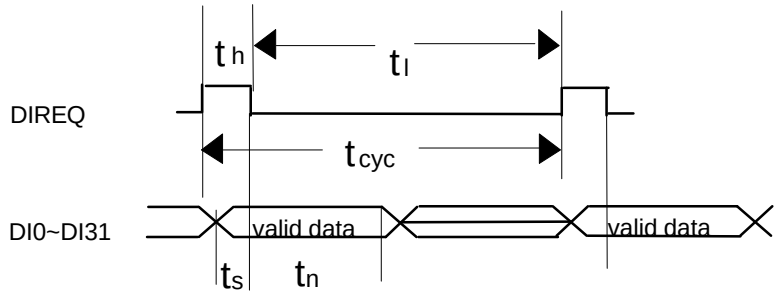
1. DIREQ as input data strobe (when Rising Edge Active)



$t_h \geqslant 10ns$ $t_l \geqslant 10ns$ $t_{cyc} \geqslant 25ns$

$t_s \geqslant 5ns$ $t_n \geqslant 10ns$

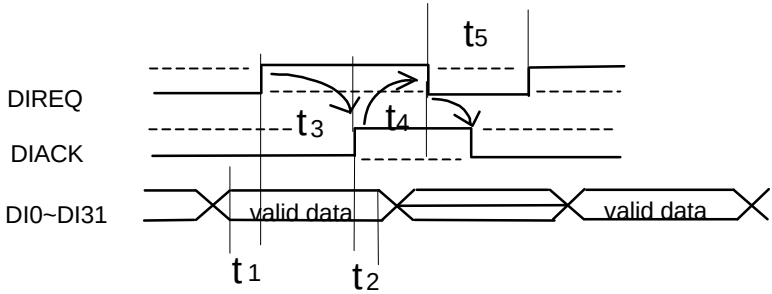
2. DIREQ as input data strobe (when Falling Edge Active)



$t_h \geqslant 10\ ns$ $t_l \geqslant 10\ ns$ $t_{cyc} \geqslant 25\ ns$

$t_s \geqslant 5\ ns$ $t_n \geqslant 10\ ns$

3. DIREQ & DIACK Handshaking



$t_1 \geq$ 0ns $t_5 \geq$ 50ns $t_3 \geq$ 35 ns

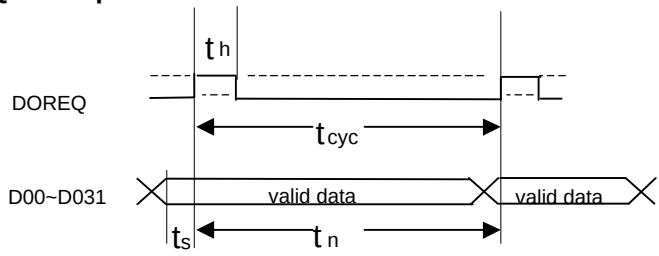
$t_2 \geq$ 0ns $t_4 \geq$ 0 ns

Note : DIREQ must be asserted until DIACK asserts, DIACK will be asserted until DIREQ de-asserted.

Note : The dashed line in the timing diagram represents the opposite polarity of the control signal. All the six control signals can be set to positive edge active or negative edge active individually through its associated bit in the POL_CTRL register mentioned in Section 3.1.8.

---

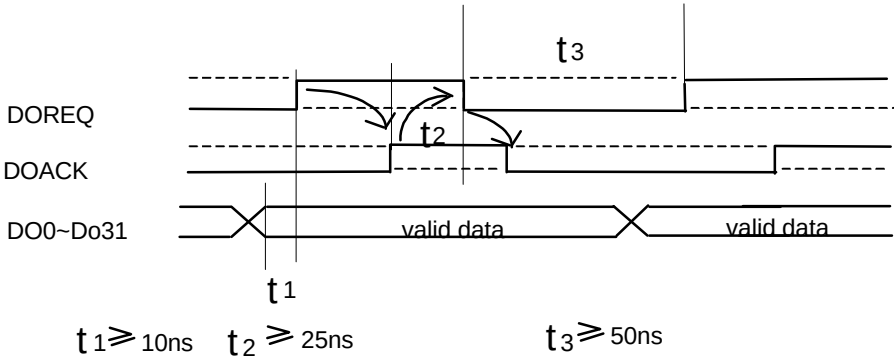
4. DOREQ as output data strobe



$t_s \geq$ 10ns t_h = 25 or 50 ns $t_{cyc} \geq$ 50ns

$t_s \geq$ 25ns

5. DOREQ & DOACK Handshaking



$t_1 \geqslant 10ns$ $t_2 \geqslant 25ns$ $t_3 \geqslant 50ns$

Note : DOACK must be deserted before DOREQ asserts, DOACK can be asserted any time after DOREQ asserts, DOREQ will be reasserted after DOACK is asserted.

5

C/C++ & DLL Libraries

In this chapter, the PCI-7300A's software drivers: C/C++ language library for DOS and DLL driver for Windows 95 are described.

5.1 Installation

5.1.1 Installation

The *PCI-7300A Software Library supplied with PCI-7300a* includes DOS C-language library, Windows 95 DLL library, and some demonstration programs which can help you reduce programming work.

Due to the installation on different platforms should follow different procedures. The installation procedures are classified to two operating systems.

¨ *MS-DOS Installation:*

The procedures should be followed as:

1. Turn your PC's power switch on
2. Put the ADLink's "All-in-one" CD into the appropriate CD drive.
3. To Install for **DOS** environment, type the command (X indicates the CD ROM drive):

X:\> **cd NUDAQPCI\7300\DOS**
X:\NUDAQPCI\7300\DOS> **SETUP**

4. An installation complete message will be shown on the screen.

After installation, all the files *of PCI-7300a Library & Utility for DOS* are stored in C:\ADLink\7300\DOS directory.

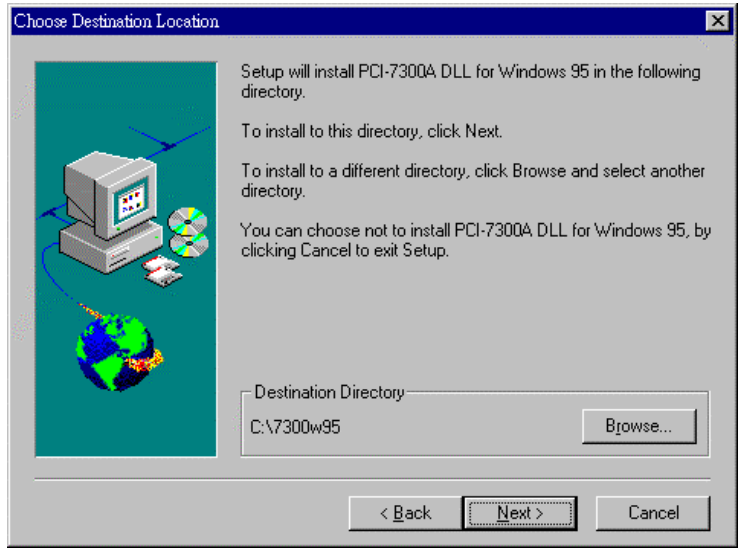
¨ *Win-95 Installation:*

1. Put the ADLink's "All-in-one" CD into the appropriate CD drive.

2. If autorun setup program is not invoked automatically, please execute ***X:\Setup.exe***.(***X*** indicates the CD ROM drive)

3. Select NuDAQ PCI→Drivers→Win95/98→PCI-7300A to setup PCI-7300A DLL for Windows 95.

After a welcome dialog box, Setup prompts the following dialog box for you to specify the destination directory. The default path is C:\ADLink\7300\W95. If you want to install *PCI-7300A DLL for Windows 95* in another directory, please click Browse button to change the destination directory.



Then you can click Next to begin installing *PCI-7300A DLL for Windows 95*.

After you complete the installation of PCI-7300A Software, PCI-7300A's DLL (7300A.DLL) is copied to Windows System directory (default is C:\WINDOWS\SYSTEM) and the driver files (W95_7300.VXD and PCIW95.VXD) are also copied to the appropriate directory. 7300.lib is the import library for Visual C/C++. 7300_bc.lib is the import library for Borland C++.

(PCI-7300A is a plug & play card, so please refer to section 2.3 of this manual for the detailed description of device installation in Windows 95).

5.2 Software Driver Naming Convention

The functions of PCI-7300A's software drivers are using full-names to represent the functions' real meaning. The naming convention rules are:

- **DOS**

  _{hardware_model}_{action_name}. e.g. **_7300_Initial** ().

- **Windows 95**

  In order to recognize the difference between DOS library and Windows library, A capital "**W**" is put on the head of each function name of the Windows DLL driver. e.g. **W_7300_Initial** ()

There are 25 function calls provided by each driver for PCI-7300A Digital I/O cards; all drivers (DOS and Win-95) provide the same function capability. The function names using in Windows is only a capital "**W**" put on the head of each function name of DOS library.

The detailed descriptions of each function are specified in the following sections.

5.3 _7300_Initial

@ **Description**

A PCI-7300A card is initialized according to the card number. Because the PCI-7300A is PCI bus architecture and meets the plug and play design, the IRQ and base address (pass-through address) are assigned by system BIOS directly. Every PCI-7300A card has to be initialized by this function before calling other functions.

Note : Because configuration of PCI-7300A is handled by the system, there is no jumpers or DMA selection on the PCI boards that need to be set up by the users.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_Initial (int card_number, int *pcic_base_addr, int *lb_base_addr, int *irq_no, int *pci_master)

Visual Basic (Windows 95)

W_7300_Initial (ByVal card_number As Long, pcic_base_addr As Long, lb_base_addr As Long, irq_no As Long, pci_master As Long) As Long

C/C++ (DOS)

int _7300_Initial (int card_number, int *pcic_base_addr, int *lb_base_addr, int *irq_no, int *pci_master)

@ **Argument**

card_number :	the card number to be initialized, only four cards can be initialized, the card number must be CARD_1, CARD_2, CARD_3 or CARD_4.
pcic_base_addr :	the I/O port base address of the PCI controller on card, it is assigned by system BIOS.
lb_base_addr :	the I/O port base address of the card, it is assigned by system BIOS.
irq_no :	system will give an available interrupt number to this card automatically.
pci_master :	**TRUE:** BIOS enabled PCI bus mastering **FALSE:** BIOS did not enable PCI bus mastering

@ **Return Code**

NoError
PCICardNumErr
PCIBiosNotExist
PCICardNotExist
PCIBaseAddrErr

5.4 _7300_Close

@ **Description**

Close a previously initialized PCI-7300A card.

@ **Syntax**

Visual C/C++ (Windows 95)
int W_7300_Close (int card_number)

Visual Basic (Windows 95)
W_7300_Close (ByVal card_number As Long) As Long

C/C++ (DOS)
int _7300_Close (int card_number)

@ **Argument**

card_number : The card number of the PCI-7300A card.

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit

5.5 _7300_Configure

@ **Description**

Set the port DI/O configuration, terminator control, and control signal polarity for the PCI-7300A card.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_Configure (int card_number, int dio_config, int term_cntrl, int cntrl_pol)

Visual Basic (Windows 95)

W_7300_Configure (ByVal card_number As Long, ByVal dio_config As Long, ByVal term_cntrl As Long, ByVal cntrl_pol As Long) As Long

C/C++ (DOS)

int _7300_Configure (int card_number, int dio_config, int term_cntrl, int cntrl_pol)

@ **Argument**

card_number : The card number of the PCI-7300A card.

dio_config : The port configuration
DI32: input port is 32-bit wide, PORTB is configured as the extension of PORTA.
DO32: output port is 32-bit wide, PORTA is configured as the extension of PORTB.
DI8DO8: PORTA is 8-bit input and PORTB is 8-bit output
DI8DO16: PORTA is 8-bit input and PORTB is 16-bit output
DI16DO8: PORTA is 16-bit input and PORTB is 8-bit output
DI16DO16: PORTA is 16-bit input and PORTB is 16-bit output

term_cntrl : the terminator control
PAOFF_PBOFF: PORTA terminator OFF, PORTB terminator OFF
PAOFF_PBON: PORTA terminator OFF, PORTB terminator ON
PAON_PBOFF: PORTA terminator ON, PORTB terminator OFF
PAON_PBON: PORTA terminator ON, PORTB terminator ON

cntrl_pol : The polarity configuration. This argument is an integer expression formed from one or more of the manifest constants defined in 7300.h. There are six groups of constants:

(1) DIREQ

DIREQ_POS: DIREQ signal is rising edge active

DIREQ_NEG: DIREQ signal is falling edge active

(2) DIACK

DIACK_POS: DIACK signal is rising edge active

DIACK_NEG: DIACK signal is falling edge active

(3) DITRIG

DITRIG_POS: DITRIG signal is rising edge active

DITRIG_NEG: DITRIG signal is falling edge active

(4) DOREQ

DOREQ_POS: DOREQ signal is rising edge active

DOREQ_NEG: DOREQ signal is falling edge active

(5) DOACK

DOACK_POS: DOACK signal is rising edge active

DOACK_NEG: DOACK signal is falling edge active

(6) DOTRIG

DOTRIG_POS: DOTRIG signal is rising edge active

DOTRIG_NEG: DOTRIG signal is falling edge active

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit
InvalidDIOConfigure

5.6 _7300_DI_Mode

@ **Description**

Set the clock mode and start mode for the PCI-7300A DI operation.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_DI_Mode (int card_number, int clk_mode, int start_mode)

Visual Basic (Windows 95)

W_7300_DI_Mode (ByVal card_number As Long, ByVal clk_mode As Long, ByVal start_mode As Long) As Long

C/C++ (DOS)

int _7300_DI_Mode (int card_number, int clk_mode, int start_mode)

@ **Argument**

card_number : The card number of the PCI-7300A card.

clk_mode :	**DI_CLK_TIMER:** use timer0 output as input clock **DI_CLK_20M:** use 20MHz clock as input clock **DI_CLK_10M:** use 10MHz clock as input clock **DI_CLK_REQ:** use external clock (DI_REQ) as input clock **DI_CLK_REQACK:** REQ/ACK handshaking mode
start_mode :	**DI_WAIT_TRIG:** delay input sampling until DITRIG is active **DI_NO_WAIT:** start input sampling immediately

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit
InvalidDIOMode

5.7 _7300_DO_Mode

@ **Description**

Set the clock mode and start mode for the PCI-7300A DO operation.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_DO_Mode (int card_number, int clk_mode, int start_mode, int fifo_threshold)

Visual Basic (Windows 95)

W_7300_DO_Mode (ByVal card_number As Long, ByVal clk_mode As Long, ByVal start_mode As Long, ByVal fifo_threshold As Long) As Long

C/C++ (DOS)

int _7300_DO_Mode (int card_number, int clk_mode, int start_mode, int fifo_threshold)

@ **Argument**

card_number :	The card number of the PCI-7300A card.
clk_mode :	**DO_CLK_TIMER:** use timer1 output as output clock **DO_CLK_20M:** use 20MHz clock as output clock **DO_CLK_10M:** use 10MHz clock as output clock **DO_CLK_ACK:** REQ/ACK handshaking **DO_CLK_TIMER_ACK:** burst handshaking mode by using timer1 output as output clock **DO_CLK_10M_ACK:** burst handshaking mode by using 10MHz clock as output clock **DO_CLK_20M_ACK:** burst handshaking mode by using 20MHz clock as output clock
start_mode :	**DO_WAIT_TRIG:** delay output data until DOTRIG is active **DO_NO_WAIT:** start output data immediately **DO_WAIT_FIFO:** delay output data until FIFO is not almost empty **DO_WAIT_BOTH:** delay output data until DOTRIG is active and FIFO is not almost empty.
fifo_threshold :	programmable almost empty threshold of both PORTB FIFO and PORTA FIFO (if PORTA is set as output).

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit
InvalidDIOMode

5.8 _7300_AUX_DI

@ **Description**

Read data from auxiliary digital input port. You can get all 4 bits input data by using this function.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_AUX_DI (int card_number, int *aux_di)

Visual Basic (Windows 95)

W_7300_AUX_DI (ByVal card_number As Long, aux_di As Long) As Long

C/C++ (DOS)

int _7300_AUX_DI (int card_number, int *aux_di)

@ **Argument**

card_number : The card number of the PCI-7300A card.
aux_di : returns 4-bit value from auxiliary digital input port.

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit

5.9 _7300_AUX_DI_Channel

@ **Description**

Read data from auxiliary digital input channel. There are 4 digital input channels on the PCI-7300A auxiliary digital input port. When performs this function, the auxiliary digital input port is read and the value of the corresponding channel is returned.

** channel means each bit of digital input port.*

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_AUX_DI_Channel (int card_number, int di_ch_no, int *aux_di)

Visual Basic (Windows 95)

W_7300_AUX_DI_Channel (ByVal card_number As Long, ByVal di_ch_no As Long, aux_di As Long) As Long

C/C++ (DOS)

int _7300_AUX_DI_Channel (int card_number, int di_ch_no, int *aux_di)

@ **Argument**

card_number : The card number of the PCI-7300A card.
di_ch_no : the DI channel number, the value has to be set within 0 and 3.
aux_di : return value, either 0 or 1.

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit
InvalidDIOChNum

5.10 _7300_AUX_DO

@ **Description**

Write data to auxiliary digital output port. There are 4 auxiliary digital outputs on the PCI-7300A.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_AUX_DI (int card_number, int do_data)

Visual Basic (Windows 95)

W_7300_AUX_DI (ByVal card_number As Long, ByVal do_data As Long) As Long

C/C++ (DOS)
int _7300_AUX_DI (int card_number, int do_data)

@ **Argument**
card_number : The card number of the PCI-7300A card.
do_data : value will be written to auxiliary digital output port

@ **Return Code**
NoError
PCICardNumErr
PCICardNotInit

5.11 _7300_AUX_DO_Channel

@ **Description**
Write data to auxiliary digital output channel (bit). There are 4 auxiliary digital output channels on the PCI-7300A. When performs this function, the digital output data is written to the corresponding channel.

** channel means each bit of digital output port.*

@ **Syntax**
Visual C/C++ (Windows 95)
int W_7300_AUX_DO_Channel (int card_number, int do_ch_no, int do_data)

Visual Basic (Windows 95)
W_7300_AUX_DO_Channel (ByVal card_number As Long, ByVal do_ch_no As Long, ByVal do_data As Long) As Long

C/C++ (DOS)
int _7300_AUX_DO_Channel (int card_number, int do_ch_no, int do_data)

@ **Argument**
card_number : The card number of the PCI-7300A card.

do_ch_no : the DO channel number, the value has to be set within 0 and 3.

do_data : either 0 (OFF) or 1 (ON).

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit
InvalidDIOChNum
InvalidDOData

5.12 _7300_Alloc_DMA_Mem

@ **Description**

Contact Windows 95 system to allocate a memory for DMA transfer. This function is only available in Windows 95 version.

@ **Syntax**

Visual C/C++ (Windows 95)

```
int W_7300_Alloc_DMA_Mem (U32 buf_size, HANDLE *memID,
        U32 *linearAddr)
```

Visual Basic (Windows 95)

```
W_7300_Alloc_DMA_Mem (ByVal buf_size As Long, memID As
        Long, linearAddr As Long) As Long
```

@ **Argument**

buf_size: Bytes to allocate. Please be careful, the unit of this argument is BYTE, not SAMPLE.

memID: If the memory allocation is successful, driver returns the ID of that memory in this argument. Use this memory ID in **`W_7300_DI_DMA_Start`** or **`W_7300_DO_DMA_Start`** function call.

linearAddr: The linear address of the allocated DMA memory. You can use this linear address as a pointer in C/C++ to access (read/write) the DMA data.

@ **Return Code**

NoError

AllocDMAMemFailed

5.13 _7300_Free_DMA_Mem

@ Description

Deallocate a system DMA memory under Windows 95 environment. This function is only available in Windows 95 version.

@ Syntax

Visual C/C++ (Windows 95)

int W_7300_Free_DMA_Mem (HANDLE memID)

Visual Basic (Windows 95)

W_7300_Free_DMA_Mem (ByVal memID As Long) As Long

@ Argument

memID: The memory ID of the system DMA memory to deallocate.

@ Return Code

NoError

5.14 _7300_DI_DMA_Start

@ Description

The function will perform digital input with DMA data transfer. It will take place in the background which will not stop until the N-th input data is transferred or your program execute **_7300_DI_DMA_Abort** function to stop the process.

After executing this function, it is necessary to check the status of the operation by using the function **_7300_DI_DMA_Status**. The PCI-7300A Bus mastering DMA is different from traditional PC style DMA. Its description is as follows:

Bus Mastering DMA mode of PCI-7300A :

PCI bus mastering offers the highest possible speed available on the PCI-7300A. When the function **_7300_DI_DMA_Start** is executed,

it will enable PCI bus master operation. This is conceptually similar to DMA (Direct Memory Access) transfers in a PC but is really PCI bus mastering. It does not use an 8237-style DMA controller in the host computer and therefore isn't blocked in 64K maximal groups. PCI-7300A bus mastering works as follows:

1. To set up bus mastering, first do all normal PCI-7300A initialization necessary to control the board in status mode. This includes testing for the presence of the PCI BIOS, determining the base addresses, slot number, vendor and device ID's, I/O or memory, space allocation, etc. Please make sure your PCI-7300A is plug in a bus master slot, otherwise this function will not be workable.

2. Load the PCI controller with the count and 32-bit physical address of the start of previously allocated destination memory which will accept data. This count is the number of *bytes* (not longwords!) transferred during the bus master operation and can be a large number up to 8 million (2^23) bytes. Since the PCI-7300A transfers are always longwords, this is 2 million longwords (2^21).

3. After the input sampling is started, the input data is stored in the FIFO of PCI controller. Each bus mastering data transfer continually tests if any data in the FIFO and then blocks transfer, the system will continuously loop until the conditions are satisfied again *but will not exit the block transfer cycle if the block count is not complete*. If there is momentarily no input data, the PCI-7300A will relinquish the bus temporarily but returns immediately when more input data appear. This operation continues until the whole block is done.

4. This operation proceeds transparently until the PCI controller transfer byte count is reached. All normal PCI bus operation applies here such as a receiver which cannot accept the transfers, higher priority devices requesting the PCI bus, etc. Remember that only one PCI initiator can have bus mastering at any one time. However, review the PCI priority and "fairness" rules. Also study the effects of the Latency Timer.

And be aware that the PCI priority strategy (round robin rotated, fixed priority, custom, etc.) is unique to your host PC and is explicitly *not* defined by the PCI standard. You must determine this priority scheme for your own PC (or replace it).

5. The interrupt request from the PCI controller can be optionally set up to indicate that this longword count is complete although this can also be determined by polling the PCI controller.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_DI_DMA_Start (int card_number, HANDLE memID, U32 count, int clear_fifo, int disable_di)

Visual Basic (Windows 95)

W_7300_DI_DMA_Start (ByVal card_number As Long, ByVal memID As Long, ByVal count As Long, ByVal clear_fifo As Long, ByVal disable_di As Long) As Long

C/C++ (DOS)

int _7300_DI_DMA_Start (int card_number, int mode, U32 *buffer, U32 count, int clear_fifo, int disable_di)

@ **Argument**

card_number : The card number of the PCI-7300A card.

mode (DOS): **CHAIN_DMA:** chaining DMA mode. By using the scatter-gather capability of PCI-7300A, the input data is put to several buffers which chained together.
NON_CHAIN_DMA: The input data is stored in a block of contiguous memory.

memID (Win-95): the memory ID of the allocated system DMA memory. In Windows 95 environment, before calling **`W_7300_DI_DMA_Start`**, **`W_7300_Alloc_DMA_Mem`** must be called to allocate a DMA memory.
`W_7300_Alloc_DMA_Mem` will return a memory ID for identifying the allocated DMA memory, as well

as the linear address of the DMA memory for user to access the data.

buffer (DOS): With non-chaining mode, this is the start address of the memory buffer to store the DI data. With chaining-mode (scatter-gather), this is the address (pointer) of first DMA descriptor node.
***With non-chaining mode, this memory should be double-word alignment. With chaining-mode, this address should be 16-byte alignment. Also the pointer of all DMA descriptor nodes should be 16-byte alignment.*

count : With non-chaining mode, this is the number of digital input to transfer. The unit is double-word (4-byte). The value of *count* can not exceed 2^21 (about 2 million). With chaining mode, please set this argument to 0. The number of digital input is determined by the information in DMA descriptor nodes.

clear_fifo : **0**: retain the FIFO data
1: clear FIFO data before perform digital input

disable_di : **0**: digital input operation still active after DMA transfer complete
1: disable digital input operation immediately when DMA transfer complete

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit
DMATransferNotAllowed
InvalidDIOCount
BufNotDWordAlign
DMADscrBadAlign

5.15 _7300_DI_DMA_Status

@ **Description**

Since the **_7300_DI_DMA_Start** function is executed in background, you can issue this function to check its operation status.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_DI_DMA_Status (int card_number, int *status)

Visual Basic (Windows 95)

W_7300_DI_DMA_Status (ByVal card_number As Long, status As Long) As Long

C/C++ (DOS)

int _7300_DI_DMA_Status (int card_number, int *status)

@ **Argument**

card_number : The card number of the PCI-7300A card.

status : status of the DMA data transfer

0 (DMA_DONE): DMA is completed

1 (DMA_CONTINUE): DMA is not completed

@ **Return Code**

ERR_NoError
PCICardNumErr
PCICardNotInit

5.16 _7300_DI_DMA_Abort

@ **Description**

This function is used to stop the DMA DI operation. After executing this function, the DMA transfer operation is stopped.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_DI_DMA_Abort (int card_number)

Visual Basic (Windows 95)

W_7300_DI_DMA_Abort (ByVal card_number As Long) As Long

C/C++ (DOS)
int _7300_DI_DMA_Stop (int card_number)

@ **Argument**
card_number : The card number of the PCI-7300A card.

@ **Return Code**
NoError
PCICardNumErr
PCICardNotInit

5.17 _7300_GetOverrunStatus

@ **Description**
When you use **_7300_DI_DMA_Start** to input data, the input data is stored in the FIFO of PCI controller. The data then transfer to memory through PCI-bus if PCI-bus is available. If the FIFO is full and next data is written to the FIFO, overrun situation occurs. Using this function to check overrun status.

@ **Syntax**
Visual C/C++ (Windows 95)
int W_7300_GetOverrunStatus (int card_number, int *overrun)

Visual Basic (Windows 95)
int W_7300_GetOverrunStatus (ByVal card_number As Long,
overrun As Long) As Long

C/C++ (DOS)
int _7300_GetOverrunStatus (int card_number, int *overrun)

@ **Argument**
card_number : The card number of the PCI-7300A card.
overrun : **0:** overrun sitation did not occur.
1: overrun situation occurred.

@ **Return Code**
NoError
PCICardNumErr

PCICardNotInit

5.18 _7300_DO_DMA_Start

@ Description

The function will perform digital output N times with DMA data transfer. It will takes place in the background which will not be stop until the Nth conversion has been completed or your program execute **_7300_DO_DMA_Abort** function to stop the process. After executing this function, it is necessary to check the status of the operation by using the function **_7300_DO_DMA_Status**.

@ Syntax

Visual C/C++ (Windows 95)

```
int W_7300_DO_DMA_Start (int card_number, HANDLE memID,
        U32 count)
```

Visual Basic (Windows 95)

```
W_7300_DO_DMA_Start (ByVal card_number As Long, ByVal
        memID As Long, ByVal count As Long) As Long
```

C/C++ (DOS)

```
int _7300_DO_DMA_Start (int card_number, U32 *buff, U32 count,
        int repeat, DMA_DSCR *dma_dscr_ptr)
```

@ Argument

card_number : The card number of the PCI-7300A card.

memID (Win-95) : the memory ID of the allocated system DMA memory. In Windows 95 environment, before calling **W_7300_DO_DMA_Start**, **W_7300_Alloc_DMA_Mem** must be called to allocate a DMA memory. **W_7300_Alloc_DMA_Mem** will return a memory ID for identifying the allocated DMA memory, as well as the linear address of the DMA memory for user to access the data. So you should write the output data to this memory before calling **W_7300_DO_DMA_Start**.

buff (DOS) : If repeat is set as 0, this is the start address of the memory buffer to store the DO data. If repeat is set as 1, this argument is of no use.
*** This memory should be double-word alignment*

count : For non-chaining mode, this is the total number of digital output data in double-words (4-byte). The value of *count* can not exceed 2^21 (about 2 million). For chaining mode, please set this argument as 0. The number of digital output is determined by the information in DMA descriptor nodes.

repeat (DOS) : **0:** Use non-chaining mode DMA transfer. The digital output data is stored in *buff*.
1: Use chaining mode DMA transfer. The digital output data is stored in several buffers. The information of the buffers is stored in DMA description nodes. All description nodes are chained together.

dma_dscr_ptr (DOS) : the pointer to the first DMA description node. Since the DMA description nodes are chained together, with giving this pointer, data in all buffers will be transferred.

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit
DMATransferNotAllowed
InvalidDIOCount
BufNotDWordAlign
DMADscrBadAlign

5.19 _7300_DO_DMA_Status

@ **Description**

Since the **_7300_DO_DMA_Start** function is executed in background, you can issue the function **_7300_DO_DMA_Status** to check its operation status.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_DO_DMA_Status (int card_number, int *status)

Visual Basic (Windows 95)

W_7300_DO_DMA_Status (ByVal card_number As Long, status As Long) As Long

C/C++ (DOS)

int _7300_DO_DMA_Status (int card_number, int *status)

@ **Argument**

card_number : The card number of the PCI-7300A card.

status : status of the DMA data transfer
0 (DMA_DONE): DMA is completed
1 (DMA_CONTINUE): DMA is not completed

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit

5.20 _7300_DO_DMA_Abort

@ **Description**

This function is used to stop the DMA DO operation. After executing this function, the **`_7300_DO_DMA_Start`** function is stopped.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_DO_DMA_Abort (int card_number)

Visual Basic (Windows 95)

W_7300_DO_DMA_Abort (ByVal card_number As Long) As Long

C/C++ (DOS)

int _7300_DO_DMA_Abort (int card_number)

@ **Argument**

card_number : The card number of the PCI-7300A card.

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit

5.21 _7300_DO_PG_Start

@ **Description**

The function will perform pattern generation with the data stored in *buff_ptr*. It will takes place in the background which will not be stop until your program execute **_7300_DO_PG_Stop** function to stop the process.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_DO_PG_Start (int card_number, void *buff_ptr, U32 count)

Visual Basic (Windows 95)

W_7300_DO_PG_Start (ByVal card_number As Long, buff_ptr As Any, ByVal count As Long) As Long

C/C++ (DOS)

int _7300_DO_PG_Start (int card_number, void *buff_ptr, U32 count)

@ **Argument**

card_number :	The card number of the PCI-7300A card.
buff_ptr :	the start address of the memory buffer to store the output data of pattern generation. *** This memory should be double-word alignment*
count :	the total number of pattern generation samples. The size of the sample depends on the port configuration. For example, if port is set as DO32, each sample contains 4 bytes; if port is

set as DI16DO8 or DI8DO8, each sample is 1 byte.

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit
DMATransferNotAllowed
InvalidDIOCount
BufNotDWordAlign
DMADscrBadAlign

5.22 _7300_DO_PG_Stop

@ **Description**

This function is used to stop the pattern generation operation. After executing this function, the **_7300_DO_PG_Start** function is stopped.

@ **Syntax**

Visual C/C++ (Windows 95)
int W_7300_DO_PG_Stop (int card_number)

Visual Basic (Windows 95)
W_7300_DO_PG_Stop (ByVal card_number As Long) As Long

C/C++ (DOS)
int _7300_DO_PG_Stop (int card_number)

@ **Argument**

card_number : The card number of the PCI-7300A card.

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit

5.23 _7300_DI_Timer

@ **Description**

This function is used to set the internal timer pacer for digital input. Timer pacer frequency = 10Mhz / C0.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_DI_Timer (int card_number, U16 c0)

Visual Basic (Windows 95)

W_7300_DI_Timer (ByVal card_number As Long, ByVal c0 As Integer) As Long

C/C++ (DOS)

int _7300_DI_Timer (int card_number, U16 c0)

@ **Argument**

card_number : The card number of the PCI-7300A card.
c0 : frequency divider of Counter #0. Valid value ranges from 2 to 65535.

Note : Since the Integer type in Visual Basic is signed integer. It's range is within -32768 and 32767. In Visual Basic, if you want to set c0 as value larger than 32767, please set it as the intended value minus 65536. For example, if you want to set c0 as 40000, please set c0 as 40000-65536=-25536.

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit

5.24 _7300_DO_Timer

@ **Description**

This function is used to set the internal timer pacer for digital output. Timer pacer frequency = 10Mhz / C1.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_DO_Timer (int card_number, U16 c1)

Visual Basic (Windows 95)
W_7300_DO_Timer (ByVal card_number As Long, ByVal c1 As Integer) As Long

C/C++ (DOS)
int _7300_DO_Timer (int card_number, U16 c1)

@ **Argument**

card_number : The card number of the PCI-7300A card.
c1 : frequency divider of Counter #1

Note : Since the Integer type in Visual Basic is signed integer. It's range is within -32768 and 32767. In Visual Basic, if you want to set c1 as value larger than 32767, please set it as the intended value minus 65536. For example, if you want to set c1 as 40000, please set c1 as 40000-65536 = -25536.

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit

5.25 _7300_Int_Timer

@ **Description**

This function is used to set Counter #2.

@ **Syntax**

Visual C/C++ (Windows 95)
int W_7300_Int_Timer (int card_number, U16 c2)

Visual Basic (Windows 95)
W_7300_Int_Timer (ByVal card_number As Long, ByVal c2 As Integer) As Long

C/C++ (DOS)
int _7300_Int_Timer (int card_number, U16 c2)

@ **Argument**

card_number : The card number of the PCI-7300A card.
c2 : frequency divider of Counter #2

Note : Since the Integer type in Visual Basic is signed integer. It's range is within -32768 and 32767. In Visual Basic, if you want to set c2 as value larger than 32767, please set it as the intended value minus 65536. For example, if you want to set c1 as 40000, please set c1 as 40000-65536 = -25536.

@ **Return Code**

NoError
PCICardNumErr
PCICardNotInit

5.26 _7300_Get_Sample

@ **Description**

For the language without pointer support such as Visual Basic, programmer can use this function to access the *index*-th data in input DMA buffer. This function is only available in Windows 95 version.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_Get_Sample (U32 linearAddr, U32 index, U32 *data_value, U32 portWidth)

Visual Basic (Windows 95)

W_7300_Get_Sample (ByVal linearAddr As Long, ByVal index As Long, data_value As Long, ByVal portWidth As Long) As Long

@ **Argument**

linearAddr : The linear address of the allocated DMA memory.
index : The index of the sample. The first sample is with index 0.
dataValue : The sample retrieved. The width of retrieved data is different with the different portWidth value.

portWidth : The port width of the digital input port. The possible values are 8, 16, or 32.

@ **Return Code**

NoError

5.27 _7300_Set_Sample

@ **Description**

For the language without pointer support such as Visual Basic, programmer can use this function to write the output data to the *index*-th position in output DMA buffer. This function is only available in Windows 95 version.

@ **Syntax**

Visual C/C++ (Windows 95)

int W_7300_Set_Sample (U32 linearAddr, U32 index, U32 data_value, U32 portWidth)

Visual Basic (Windows 95)

W_7300_Get_Sample (ByVal linearAddr As Long, ByVal index As Long, ByVal data_value As Long, ByVal portWidth As Long) As Long

@ **Argument**

linearAddr : The linear address of the allocated DMA memory.

index : The position the data is written to. The first sample is with index 0.

dataValue : The data to put to output buffer. The data width is different with the different portWidth value.

portWidth : The port width of the digital output port. The possible values are 8, 16, or 32.

@ **Return Code**

NoError

Appendix A 8254 Programmable Interval Timer

Note : The material of this section is adopted from
"Intel Microprocessor and Peripheral Handbook Vol. II --Peripheral"

A.1 The Intel (NEC) 8254

The Intel (NEC) 8254 contains three independent, programmable, multi-mode 16 bit counter/timers. The three independent 16 bit counters can be clocked at rates from DC to 5 MHz. Each counter can be individually programmed with 6 different operating modes by appropriately formatted control words. The most commonly uses for the 8254 in microprocessor based system are:

- programmable baud rate generator
- event counter
- binary rate multiplier
- real-time clock
- digital one-shot
- motor control

For more information about the 8254, please refer to the NEC Microprocessors and peripherals or Intel Microprocessor and Peripheral Handbook.

A.2 The Control Byte

The 8254 occupies 8 I/O address locations in the PCI-7300A I/O map. As shown below.

Base + 0	LSB OR MSB OF COUNTER 0
Base + 4	LSB OR MSB OF COUNTER 1
Base + 8	LSB OR MSB OF COUNTER 2

Base + C	CONTROL BYTE for Chip 0

Before loading or reading any of these individual counters, the **control byte** (Base + C**)** must be loaded first. The format of control byte is :

Control Byte : (Base + 7, Base + 11)

Bit	7	6	5	4	3	2	1	0
	SC1	SC0	RL1	RL0	M2	M1	M0	BCD

- SC1 & SC1 - Select Counter (Bit7 & Bit 6)

SC1	SC0	COUNTER
0	0	0
0	1	1
1	0	2
1	1	ILLEGAL

- RL1 & RL0 - Select Read/Load operation (Bit 5 & Bit 4)

RL1	RL0	OPERATION
0	0	COUNTER LATCH
0	1	READ/LOAD LSB
1	0	READ/LOAD MSB
1	1	READ/LOAD LSB FIRST, THEN MSB

- M2, M1 & M0 - Select Operating Mode (Bit 3, Bit 2, & Bit 1)

M2	M1	M0	MODE
0	0	0	0
0	0	1	1
x	1	0	2
x	1	1	3
1	0	0	4
1	0	1	5

- BCD - Select Binary/BCD Counting (Bit 0)

0	BINARY COUNTER 16-BITS
1	BINARY CODED DECIMAL (BCD) COUNTER (4 DECADES)

Note:

1. The count of the binary counter is from 0 up to 65,535.
2. The count of the BCD counter is from 0 up to 99,999.

A.3 Mode Definition

In 8254, there are six different operating modes can be selected. The they are :

- **Mode 0** : interrupt on terminal count

  The output will be initially low after the mode set operation. After the count is loaded into the selected count register, the output will remain low and the counter will count. When terminal count is reached, the output will go high and remain high until the selected count register is reloaded with the mode or a new count is loaded. The counter continues to decrement after terminal count has been reached.

  Rewriting a counter register during counting results in the following:

  (1) Write 1st byte stops the current counting.
  (2) Write 2nd byte starts the new count.

- **Mode 1** : Programmable One-Shot.

  The output will go low on the count following the rising edge of the gate input. The output will go high on the terminal count. If a new count value is loaded while the output is low it will not affect the duration of the one-shot pulse until the

succeeding trigger. The current count can be read at anytime without affecting the one-shot pulse.

The one-shot is re-triggerable, hence the output will remain low for the full count after any rising edge of the gate input.

- **Mode 2** : Rate Generator.

  Divided by N counter. The output will be low for one period of the input clock. The period from one output pulse to the next equals the number of input counts in the count register. If the count register is reloaded between output pulses the present period will not be affected, but the subsequent period will reflect the new value.

  The gate input when low, will force the output high. When the gate input goes high, the counter will start form the initial count. Thus, the gate input can be used to synchronized by software.

  When this mode is set, the output will remain high until after the count register is loaded. The output then can also be synchronized by software.

- **Mode 3** : Square Wave Rate Generator.

  Similar to MODE 2 except that the output will remain high until one half the count has been completed (or even numbers) and go low for the other half of the count. This is accomplished by decrement the counter by two on the falling edge of each clock pulse. When the counter reaches terminal count, the state of the output is changed and the counter is reloaded with the full count and the whole process is repeated.

  if the count is odd and the output is high, the first clock pulse (after the count is loaded) decrements the count by 1. Subsequent clock pulses decrement the clock by 2 after time-out, the output goes low and the full count is reloaded. The first clock pulse (following the reload) decrements the counter

by 3. Subsequent clock pulses decrement the count by 2 until time-out. Then the whole process is repeated. In this way, if the count is odd, the output will be high for (N + 1)/2 counts and low for (N - 1)/2 counts.

In Modes 2 and 3, if a CLK source other then the system clock is used, GATE should be pulsed immediately following Way Rate of a new count value.

- **Mode 4** : Software Triggered Strobe.

  After the mode is set, the output will be high. When the count is loaded, the counter will begin counting. On terminal count, the output will go low for one input clock period, then will go high again.

  If the count register is reloaded during counting, the new count will be loaded on the next CLK pulse. The count will be inhibited while the GATE input is low.

- **Mode 5** : Hardware Triggered Strobe.

  The counter will start counting after the rising edge of the trigger input and will go low for one clock period when the terminal count is reached. The counter is re-triggerable. the output will not go low until the full count after the rising edge of any trigger.

The detailed description of the mode of 8254, please refer to the Intel Microsystem Components Handbook.

Product Warranty/Service

Seller warrants that equipment furnished will be free form defects in material and workmanship for a period of one year from the confirmed date of purchase of the original buyer and that upon written notice of any such defect, Seller will, at its option, repair or replace the defective item under the terms of this warranty, subject to the provisions and specific exclusions listed herein.

This warranty shall not apply to equipment that has been previously repaired or altered outside our plant in any way as to, in the judgment of the manufacturer, affect its reliability. Nor will it apply if the equipment has been used in a manner exceeding its specifications or if the serial number has been removed.

Seller does not assume any liability for consequential damages as a result from our products uses, and in any event our liability shall not exceed the original selling price of the equipment.

The equipment warranty shall constitute the sole and exclusive remedy of any Buyer of Seller equipment and the sole and exclusive liability of the Seller, its successors or assigns, in connection with equipment purchased and in lieu of all other warranties expressed implied or statutory, including, but not limited to, any implied warranty of merchant ability or fitness and all other obligations or liabilities of seller, its successors or assigns.

The equipment must be returned postage-prepaid. Package it securely and insure it. You will be charged for parts and labor if you lack proof of date of purchase, or if the warranty period is expired.