

NuDAQ[®] / NuIPC[®]
PCI-7200 / cPCI-7200
12MB/S High Speed
Digital Input/ Output Card
User's Guide



Recycled Paper

©Copyright 1999~2000 ADLINK Technology Inc.

All Rights Reserved.

Manual Rev. 2.20: October 14, 2000

The information in this document is subject to change without prior notice in order to improve reliability, design and function and does not represent a commitment on the part of the manufacturer.

In no event will the manufacturer be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the product or documentation, even if advised of the possibility of such damages.

This document contains proprietary information protected by copyright. All rights are reserved. No part of this manual may be reproduced by any mechanical, electronic, or other means in any form without prior written permission of the manufacturer.

Trademarks

NuDAQ, NuIPC, DAQBench are registered trademarks of ADLINK Technology Inc.

Other product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

Getting service from ADLINK

Customer Satisfaction is always the most important thing for ADLINK Tech Inc. If you need any help or service, please contact us and get it.

ADLINK Technology Inc.			
Web Site	http://www.adlink.com.tw http://www.adlinktechnology.com		
Sales & Service	service@adlink.com.tw		
Technical Support	NuDAQ	nudaq@ adlink.com.tw	
	NuDAM	nudam@ adlink.com.tw	
	NuIPC	nuipc@ adlink.com.tw	
	NuPRO	nupro@ adlink.com.tw	
	Software	sw@ adlink.com.tw	
TEL	+886-2-82265877	FAX	+886-2-82265717
Address	9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan, R.O.C.		

Please inform or FAX us of your detailed information for a prompt, satisfactory and constant service.

Detailed Company Information			
Company/Organization			
Contact Person			
E-mail Address			
Address			
Country			
TEL		FAX	
Web Site			

Questions	
Product Model	
Environment to Use	<input type="checkbox"/> OS _____ <input type="checkbox"/> Computer Brand _____ <input type="checkbox"/> M/B: <input type="checkbox"/> CPU: <input type="checkbox"/> Chipset: <input type="checkbox"/> BIOS: <input type="checkbox"/> Video Card: <input type="checkbox"/> Network Interface Card: <input type="checkbox"/> Other:
Challenge Description	
Suggestions for ADLINK	

Table of Contents

Chapter 1 Introduction.....	1
1.1 Applications.....	1
1.2 Features.....	2
1.3 Specifications	2
1.4 Software Supporting.....	4
1.4.1 <i>Programming Library</i>	4
1.4.2 <i>PCIS-LVIEW: LabVIEW® Driver</i>	5
1.4.3 <i>PCIS-VEE: HP-VEE Driver</i>	5
1.4.4 <i>DAQBench™; ActiveX Controls</i>	5
1.4.5 <i>DASYLab™ PRO</i>	5
1.4.6 <i>PCIS-DDE: DDE Server and InTouch™</i>	5
1.4.7 <i>PCIS-ISG: ISaGRAF™ driver</i>	6
1.4.8 <i>PCIS-ICL: InControl™ Driver</i>	6
1.4.9 <i>PCIS-OPC: OPC Server</i>	6
Chapter 2 Installation.....	7
2.1 What You Have	7
2.2 Unpacking.....	8
2.3 Device Installation for Windows Systems.....	8
2.4 cPCI/PCI-7200's Layout.....	9
2.5 Hardware Installation Outline	11
2.6 Connector Pin Assignments.....	12
2.6.1 <i>PCI-7200 Pin Assignments</i>	12
2.6.2 <i>cPCI-7200 Pin Assignments</i>	14
2.7 8254 for Timer Pacer Generation	15
Chapter 3 Registers Format	16
3.1 I/O Registers Format.....	16
3.2 Digital Input Register (BASE + 10).....	17
3.3 Digital Output Register (BASE + 14)	17
3.4 DIO Status & Control Register (BASE + 18)	17
3.5 Interrupt Status & Control Register (BASE + 1C).....	19
3.6 8254 Timer Registers (BASE + 0)	22
Chapter 4 Operation Theorem	23
4.1 Direct Program Control.....	23
4.2 Timer Pacer Mode.....	24
4.3 External Clock Mode.....	25
4.4 Handshaking.....	25
4.5 Timing Characteristic.....	27

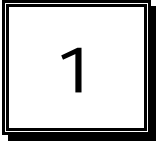
Chapter 5 C/C++ Libraries	30
5.1 Libraries Installation.....	30
5.2 Programming Guide.....	31
5.2.1 Naming Convention	31
5.2.2 Data Types.....	31
5.3 _7200_Initial.....	32
5.4 _7200_Switch_Card_No	33
5.5 _7200_AUX_DI	33
5.6 _7200_AUX_DI_Channel	34
5.7 _7200_AUX_DO.....	34
5.8 _7200_AUX_DO_Channel.....	35
5.9 _7200_DI	35
5.10 _7200_DI_Channel	36
5.11 _7200_DO	36
5.12 _7200_DO_Channel	37
5.13 _7200_Alloc_DMA_Mem	38
5.14 _7200_Free_DMA_Mem.....	39
5.15 _7200_Alloc_DBDMA_Mem.....	39
5.16 _7200_Free_DBDMA_Mem.....	40
5.17 _7200_DI_DMA_Start	40
5.18 _7200_DI_DMA_Status.....	43
5.19 _7200_DI_DMA_Stop	44
5.20 _7200_DblBufferMode.....	44
5.21 _7200_CheckHalfReady	45
5.22 _7200_DblBufferTransfer	45
5.23 _7200_GetOverrunStatus.....	46
5.24 _7200_DO_DMA_Start.....	46
5.25 _7200_DO_DMA_Status.....	47
5.26 _7200_DO_DMA_Stop.....	48
5.27 _7200_DI_Timer	49
5.28 _7200_DO_Timer.....	50
Chapter 6 Double Buffer Mode Principle	52
Chapter 7 Limitation.....	54
Product Warranty/Service.....	55

How to Use This Guide

This manual is designed to help you use the PCI-7200 and cPCI-7200. The functionality of PCI-7200 and cPCI-7200 are the same except that cPCI-7200 has 4 auxiliary digital input and output. Therefore, the "PCI-7200" represents both PCI-7200 and cPCI-7200 if not specified.

The manual describes how to modify various settings on the PCI-7200 card to meet your requirements. It is divided into seven chapters:

- ◆ Chapter 1, "Introduction," gives an overview of the product features, applications, and specifications.
- ◆ Chapter 2, "Installation," describes how to install the PCI-7200. The layout of PCI-7200 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-7200.
- ◆ Chapter 4, "Operation Theorem," describes how the PCI-7200 works.
- ◆ 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, and Win-95 environments.
- ◆ Chapter 6, "Double Buffer Mode Principle," describes the data buffer for double-buffered DMA DI operation.
- ◆ Chapter 7, "Limitation," describes three limitations on using PCI-7200.



Introduction

The PCI-7200/cPCI-7200 is PCI/CompactPCI form factor high-speed digital I/O card, it consists of 32 digital input channels, and 32 digital output channels. High performance designs and the state-of-the-art technology make this card suitable for high-speed digital input and output applications.

The PCI-7200 performs high-speed data transfers using bus-mastering DMA via 32-bit PCI bus architecture. The maximum data transfer rates can be up to 12MB per second. It is very suitable for interfacing high-speed peripherals and your computer system.

Several different digital I/O operation modes are supported:

- 1. Direct Program Control:** the digital inputs and outputs can be accessed and controlled by its corresponding I/O ports directly.
- 2. Timer Pacer Mode:** the digital input and output operations are handled by internal timer pacer clock and transferred by bus mastering DMA.
- 3. External Clock Mode:** the digital input operations are handled by external input strobe signal (I_REQ) and transferred by bus mastering DMA.
- 4. Handshaking:** through REQ signal and ACK signal, the digital I/O data can have simple handshaking data transfer.

1.1 Applications

- ◆ Interface to high-speed peripherals
- ◆ High-speed data transfers from other computers
- ◆ Digital I/O control
- ◆ Interface to external high-speed A/D and D/A converter
- ◆ Digital pattern generator
- ◆ Waveform and pulse generation
- ◆ BCD interface driver

1.2 Features

The PCI-7200 high-speed DIO Card provides the following advanced features:

- ◆ 32 TTL digital input channels
- ◆ 32 TTL digital output channels
- ◆ Transfer up to 12M Bytes per second
- ◆ High output driving and low input loading
- ◆ 32-bit PCI bus, Plug and Play
- ◆ On-board internal timer pacer clock
- ◆ Internal timer controls input sampling rate
- ◆ Internal timer controls digital output rate
- ◆ ACK and REQ for handshaking
- ◆ On-board 32-byte FIFO for both digital input and output
- ◆ Extra 8 Kbytes digital input FIFO for cPCI-7200
- ◆ 4 auxiliary digital input and output channels (cPCI-7200 only)
- ◆ Diode terminators for 32 input channels and control signals (cPCI-7200 only)
- ◆ Multiple interrupt sources are selectable by software

1.3 Specifications

- ◆ **Digital I/O (DIO)**
 - **Number of DI Channels:** 32 TTL compatible
 - **Number of DO Channels:** 32 TTL compatible
 - **Data Transfer Mode**
 - ◆ Program I/O
 - ◆ Internal timer pacer transfer
 - ◆ External I_REG strobe input
 - ◆ Handshake data transfer
 - **Maximum Transfer Speed:**
 - ◆ 3 MHz (12MHz) by external clock, handshake or external strobe
 - ◆ 2 MHz (8MHz) by internal timer pacer transfer
 - **FIFO:** 8 words (32-bit) (for PCI-7200)
2K + 8 words (32-bit) (for cPCI-7200)

- **Input Voltage:**
Low: Min. 0V; Max. 0.8V
High: Min. +2.0V
- **Input Load:**
Low: +0.5V @ -0.6mA max.
High: +2.7V @ +20µA max.
- **Output Voltage:**
Low: Min. 0V; Max. 0.5V
High: Min. +2.7V
- **Driving Capacity:**
Low: Max. +0.5V at 24mA (Sink)
High: Min. 2.4V at -3.0mA (Source)
- ◆ **Programmable Counter**
 - **Device:** 82C54-10, with a 4MHz time base
 - **Timer 0:** DI clock source
 - **Timer 1:** DO clock source
 - **Timer2:** Base clock of Timer #0 and Timer #1
 - **Pacer Output:** 0.00046 Hz ~ 2MHz
- ◆ **General Specifications**
 - **Operating Temperature:** 0°C ~ 60°C
 - **Storage Temperature:** -20°C ~ 80°C
 - **Humidity:** 5 ~ 95%, non-condensing
 - **Connector:**
PCI-7200: one 37-pin D-type and one 40-pin ribbon connector
cPCI-7200: one 100-pin SCSI-type connector
 - **Dimension:**
PCI-7200: Compact size, only 148mm (L) X 102mm(H)
cPCI-7200: Standard 3U CompactPCI form factor
 - **Power Consumption:**
PCI-7200: +5 V @ 720 mA typical
cPCI-7200: +5 V @ 820 mA typical

1.4 Software Supporting

ADLink provides versatile software drivers and packages for users' different approach to built-up a system. We not only provide programming library such as DLL for many Windows systems, but also provide drivers for many software package such as LabVIEW[®], HP VEE[™], DASYLab[™], InTouch[™], InControl[™], ISaGRAF[™], and so on.

All the software options are included in the ADLink CD. The non-free software drivers are protected with serial licensed code. Without the software serial number, you can still install them and run the demo version for two hours for demonstration purpose. Please contact with your dealer to purchase the formal license serial code.

1.4.1 Programming Library

For customers who are writing their own programs, we provide function libraries for many different operating systems, including:

- ◆ DOS Library: Borland C/C++ and Microsoft C++, the functions descriptions are included in this user's guide.
- ◆ Windows 95 DLL: For VB, VC++, Delphi, BC5, the functions descriptions are included in this user's guide.
- ◆ PCIS-DASK: Include device drivers and DLL for **Windows 98, Windows NT and Windows 2000**. DLL is binary compatible across Windows 98, Windows NT and Windows 2000. That means all applications developed with PCIS-DASK are compatible across Windows 98, Windows NT and Windows 2000. The developing environment can be VB, VC++, Delphi, BC5, or any Windows programming language that allows calls to a DLL. The user's guide and function reference manual of PCIS-DASK are in the CD. Please refer the PDF manual files under the following directory: \\Manual_PDF\Software\PCIS-DASK
- ◆ PCIS-DASK/X: Include device drivers and shared library for **Linux**. The developing environment can be Gnu C/C++ or any programming language that allows linking to a shared library. The user's guide and function reference manual of PCIS-DASK/X are in the CD. (\\Manual_PDF\Software\PCIS-DASK-X.)

The above software drivers are shipped with the board. Please refer to the "Software Installation Guide" to install these drivers.

1.4.2 PCIS-LVIEW: LabVIEW® Driver

PCIS-LVIEW contains the VIs, which are used to interface with NI's LabVIEW® software package. The PCIS-LVIEW supports Windows 95/98/NT/2000. The LabVIEW® drivers are free shipped with the board. You can install and use them without license. For detail information about PCIS-LVIEW, please refer to the user's guide in the CD.

(\\Manual_PDF\Software\PCIS-LVIEW)

1.4.3 PCIS-VEE: HP-VEE Driver

The PCIS-VEE includes the user objects, which are used to interface with HP VEE software package. PCIS-VEE supports Windows 95/98/NT. The HP-VEE drivers are free shipped with the board. You can install and use them without license. For detail information about PCIS-VEE, please refer to the user's guide in the CD.

(\\Manual_PDF\Software\PCIS-VEE)

1.4.4 DAQBench™: ActiveX Controls

We suggest the customers who are familiar with ActiveX controls and VB/VC++ programming use the DAQBench™ ActiveX Control components library for developing applications. The DAQBench™ is designed under Windows NT/98. For more detailed information about DAQBench, please refer to the user's guide in the CD.

(\\Manual_PDF\Software\DAQBench\DAQBench Manual.PDF)

1.4.5 DASYSLab™ PRO

DASYSLab is an easy-to-use software package, which provides easy-setup instrument functions such as FFT analysis. Please contact us to get DASYSLab PRO, which include DASYSLab and ADLink hardware drivers.

1.4.6 PCIS-DDE: DDE Server and InTouch™

DDE stands for Dynamic Data Exchange specifications. The PCIS-DDE includes the PCI cards' DDE server. The PCIS-DDE server is included in the ADLINK CD. It needs license. The DDE server can be used conjunction with any DDE client under Windows NT.

1.4.7 PCIS-ISG: ISaGRAF™ driver

The ISaGRAF WorkBench is an IEC1131-3 SoftPLC control program development environment. The PCIS-ISG includes ADLink products' target drivers for ISaGRAF under Windows NT environment. The PCIS-ISG is included in the ADLINK CD. It needs license.

1.4.8 PCIS-ICL: InControl™ Driver

PCIS-ICL is the InControl driver which support the Windows NT. The PCIS-ICL is included in the ADLINK CD. It needs license.

1.4.9 PCIS-OPC: OPC Server

PCIS-OPC is an OPC Server, which can link with the OPC clients. There are many software packages on the market can provide the OPC clients now. The PCIS-OPC supports the Windows NT. It needs license.

2

Installation

This chapter describes how to install the PCI-7200. At first, the content of the package and the unpacking information that you should be careful are described. Because the PCI-7200 is a plug and play device, there is no more jumper or DIP switch setting for configuration. The Interrupt number and I/O port address are assigned by the system BIOS during system boot up.

2.1 What You Have

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

- ◆ PCI-7200 Digital I/O & Counter Card
- ◆ Include ACL-10437: 40-pin to 37-pin D-Sub cable
or
- ◆ cPCI-7200 Digital I/O & Counter Module for 3U CompactPCI
- ◆ Include 100-pin SCSI connector assembly
- ◆ ADLINK CD
- ◆ Software Installation Guide

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-7200 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-7200.

2.3 Device Installation for Windows Systems

Once Windows 95/98/2000 has started, the Plug and Play function of Windows system will find the new NuDAQ/NuIPC cards. If this is the first time to install NuDAQ/NuIPC cards in your Windows system, you will be informed to input the device information source. Please refer to the **"Software Installation Guide"** for the steps of installing the device.

2.4 cPCI/PCI-7200's Layout

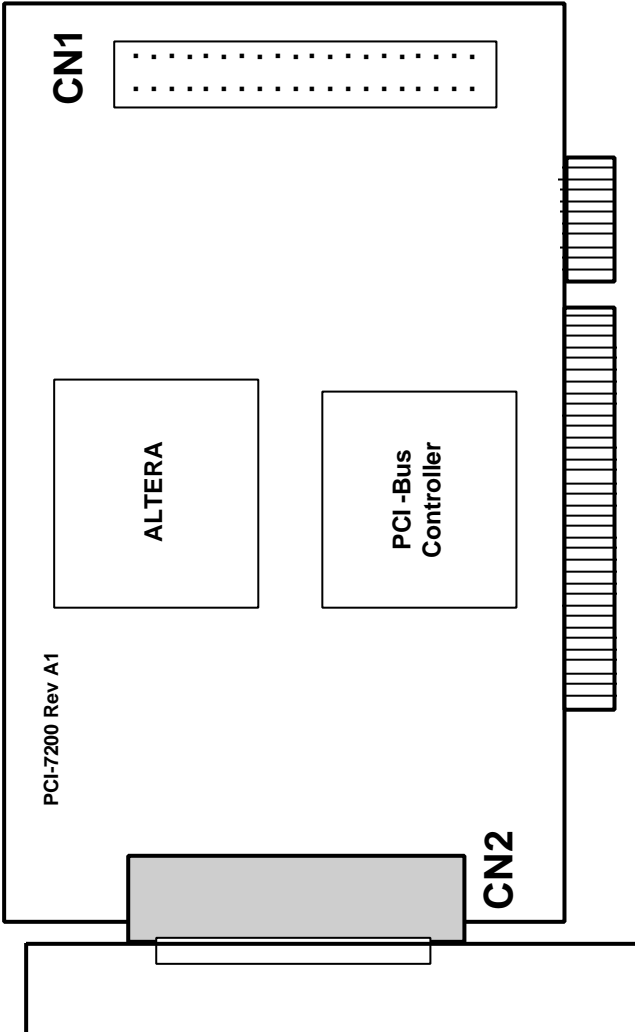


Figure 2.1(a) PCI-7200 Layout Diagram

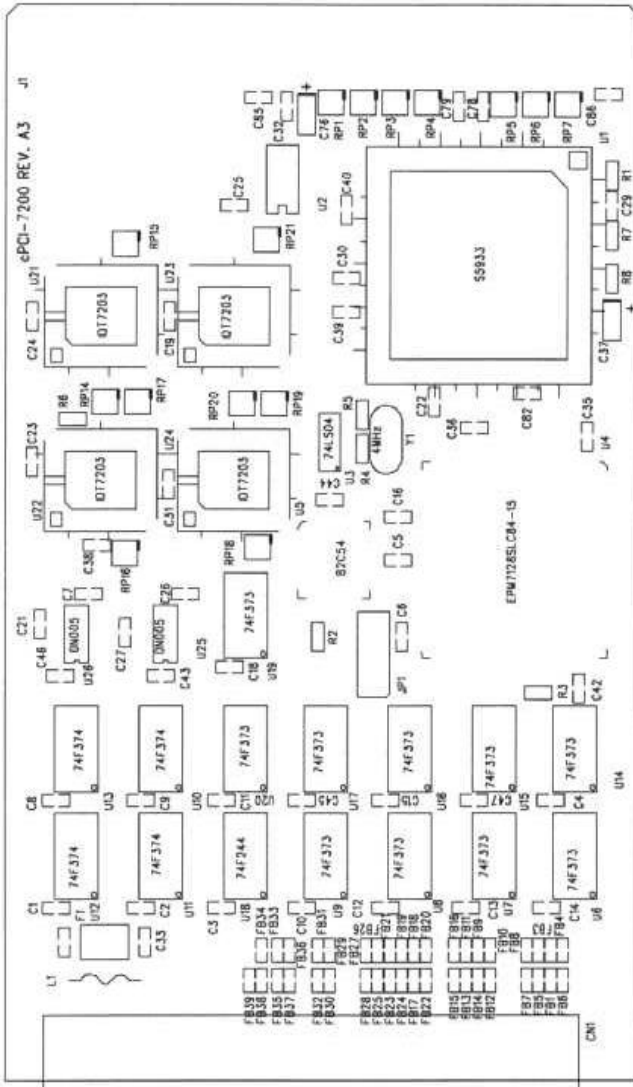


Figure 2.1(b) cPCI-7200 Layout Diagram

2.5 Hardware Installation Outline

Hardware configuration

The PCI cards (or CompactPCI cards) are equipped with plug and play PCI controller, it can request base addresses and interrupt according to PCI standard. The system BIOS will install the system resource based on the PCI cards' configuration registers and system parameters (which are set by system BIOS). Interrupt assignment and memory usage (I/O port locations) of the PCI cards can be assigned by system BIOS only. These system resource assignments are done on a board-by-board basis. It is not suggested to assign the system resource by any other methods.

PCI slot selection

The PCI card can be inserted to any PCI slot without any configuration for system resource. CompactPCI peripheral slots are marked with a circle on the backplane. Please note that the PCI and CompactPCI system board must provide bus-mastering capability to operate this board well.

Installation Procedures

1. Turn off your computer
2. Turn off all accessories (printer, modem, monitor, etc.) connected to your computer.
3. Remove the cover from your computer.
4. Setup jumpers on the PCI or CompactPCI card.
5. Select a 32-bit PCI slot. PCI slot are short than ISA or EISA slots, and are usually white or ivory.
6. Before handling the PCI cards, discharge any static buildup on your body by touching the metal case of the computer. Hold the edge and do not touch the components.
7. Position the board into the PCI slot you selected.
8. Secure the card in place at the rear panel of the system.

Running the 7200UTIL.EXE

The IRQ number and I/O port address can be configured by the system. By using the 7200UTIL.EXE, you can get the above values and they are displayed by this utility. A testing program is included in this utility, you can check if your PCI-7200 can work properly. Refer Section 5.2 for further detailed information.

2.6 Connector Pin Assignments

2.6.1 PCI-7200 Pin Assignments

The PCI-7200 comes equipped with one 37-pin D-Sub connector (CN2) located on the rear mounting plate and one 40-pin female flat cable header connector (CN1). The CN2 is located on the rear mounting plate; the CN1 is on front of the board. Refer section 2.2 PCI-7200's layout.

CN2 is used for digital inputs (DI 0 ~ DI 15) and digital outputs (DO 0 ~ DO 15), and the reminder digital I/O channels DI 16 ~ DI 31 and DO 16 ~ DO 31 are presented on the CN1. The pin assignment of CN1 and CN2 is illustrated in the figure 2.2 and 2.3.

Legend:

DO n : Digital Output CH n

DI n : Digital Input CH n

GND : Ground

ACK : ACK Signal of handshaking communication

REQ : REQ Signal of handshaking communication

I_TRG: Input signal to start the DI data sampling

O_TRG: Output signal can be controlled by software

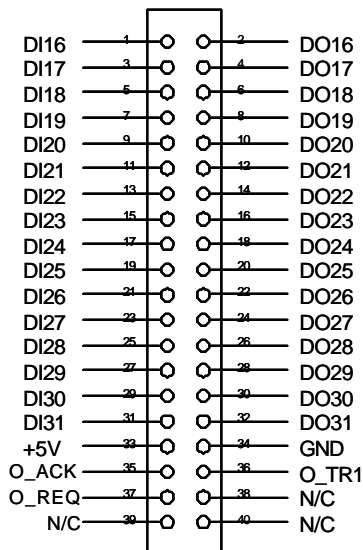


Figure 2.2 CN1 Pin Assignments

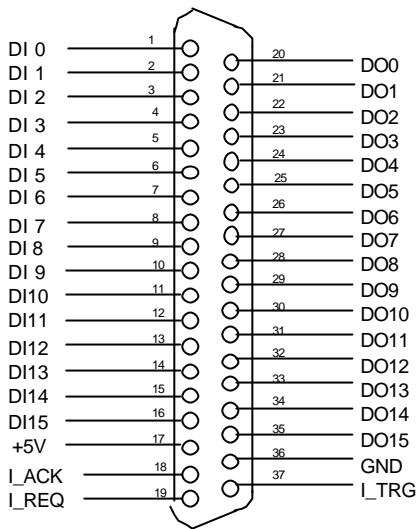
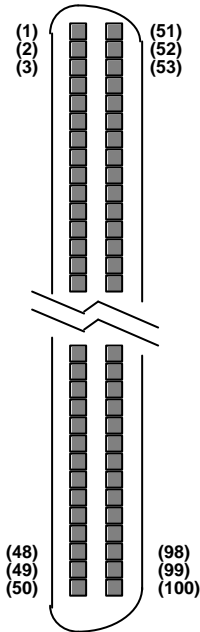


Figure 2.3 CN2 Pin Assignments

2.6.2 cPCI-7200 Pin Assignments



(1) DO0	(26) O_TRG	(51) DO1	(76) GND
(2) DO2	(27) O_REQ	(52) DO3	(77) GND
(3) DO4	(28) O_ACK	(53) DO5	(78) GND
(4) DO6	(29) AUXIN2	(54) DO7	(79) AuxOut2
(5) DO8	(30) AUXIN3	(55) DO9	(80) AuxOut3
(6) DO10	(31) +5Vout	(56) DO11	(81) GND
(7) DO12	(32) +5Vout	(57) DO13	(82) GND
(8) DO14	(33) GND	(58) DO15	(83) GND
(9) GND	(34) DIN0	(59) GND	(84) DIN1
(10) DO16	(35) DIN2	(60) DO17	(85) DIN3
(11) DO18	(36) DIN4	(61) DO19	(86) DIN5
(12) DO20	(37) DIN6	(62) DO21	(87) DIN7
(13) DO22	(38) DIN8	(63) DO23	(88) DIN9
(14) DO24	(39) DIN10	(64) DO25	(89) DIN11
(15) DO26	(40) DIN12	(65) DO27	(90) DIN13
(16) DO28	(41) DIN14	(66) DO29	(91) DIN15
(17) DO30	(42) GND	(67) DO31	(92) GND
(18) GND	(43) DIN16	(68) GND	(93) DIN17
(19) +5Vout	(44) DIN18	(69) GND	(94) DIN19
(20) +5Vout	(45) DIN20	(70) GND	(95) DIN21
(21) AUXIN0	(46) DIN22	(71) AuxOut0	(96) DIN23
(22) AUXIN1	(47) DIN24	(72) AuxOut1	(97) DIN25
(23) I_TRG	(48) DIN26	(73) GND	(98) DIN27
(24) I_REQ	(49) DIN28	(74) GND	(99) DIN29
(25) I_ACK	(50) DIN30	(75) GND	(100) DIN31

2.7 8254 for Timer Pacer Generation

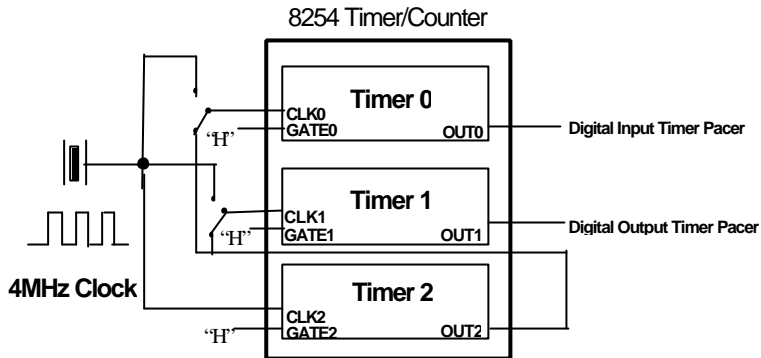


Figure 2.4 8254 configuration

The internal timer/counter 8254 on the PCI-7200 is configured as above diagram (figure 2.4). User can use it to generate the timer pacer for both digital input and digital output trigger.

The digital input timer pacer is from OUT0 (Timer 0), and the digital output timer pacer is from OUT1 (Timer 1). Besides, Timer 0 and Timer 2 can be cascaded together to generate more timer pacer frequency for digital input. Also, the Timer 2 can be cascaded with Timer 1 for digital output.

pacer rate = $4 \text{ MHz} / (C0 * C2)$

if Timer 0 & Timer 2 are cascaded

pacer rate = $4 \text{ MHz} / C0$

if timer 0 & Timer 2 are not cascaded

The maximum pacer signal rate of input and output are $4\text{MHz}/2=2\text{Mhz}$. The minimum signal rate is $4\text{MHz}/65535/65535$, which is a very slow frequency that user may never use it.

For example, if you wish to get a pacer rate 2.5 kHz, you can set $C0 = 40$ and $C2 = 40$. That is

$2.5\text{KHz} = 4\text{MHz} / (40 \times 40)$

3

Registers Format

3.1 I/O Registers Format

The PCI-7200 occupies 8 consecutive 32-bit I/O addresses in the PC I/O address space and the cPCI-7200 occupies 9 consecutive 32-bit I/O addresses. Table 4.1 shows the I/O Map

Address	Read	Write
Base + 0	Counter 0	Counter 0
Base + 4	Counter 1	Counter 1
Base + 8	Counter 2	Counter 2
Base + C	---	CLK Control CW0
Base + 10	Digital Input Reg.	---
Base + 14	Digital Output (Read-back)	Digital Output Reg.
Base + 18	DIO Status & Control	DIO Status & Control
Base + 1C	INT Status & Control	INT Status & Control
Base + 20 (cPCI-7200 only)	AUXDIO Reg.	AUXDO Reg.

Caution: (1) I/O port is 32-bit width

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

3.2 Digital Input Register (BASE + 10)

32 digital input channels can be read from this register

Address: BASE + 10

Attribute: READ Only

Data Format:

Byte	7	6	5	4	3	2	1	0
Base +10	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0
Base +11	DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8
Base +12	DI23	DI22	DI21	DI20	DI19	DI18	DI17	DI16
Base +13	DI31	DI30	DI29	DI28	DI27	DI26	DI25	DI24

3.3 Digital Output Register (BASE + 14)

32 digital output channels can be write and read-back from this register

Address: BASE + 14

Attribute: READ/WRITE

Data Format:

Byte	7	6	5	4	3	2	1	0
Base +14	DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0
Base +15	DO15	DO14	DO13	DO12	DO11	DO10	DO9	DO8
Base +16	DO23	DO22	DO21	DO20	DO19	DO18	DO17	DO16
Base +17	DO31	DO30	DO29	DO28	DO27	DO26	DO25	DO24

The digital output status can be read back through the same location (BASE + 14)

3.4 DIO Status & Control Register (BASE + 18)

The data transfer mode of digital input is controlled and status is checked through this register.

Address: BASE + 18

Attribute: READ/WRITE

Data Format:

Byte	7	6	5	4	3	2	1	0
Base +18	O_ACK	DIN_EN	I_TRG	TRGPL	I_FIFO	I_TIME0	I_REQ	I_ACK
Base +19	----	I_OVER	----	----	O_TRG	O_FIFO	O_TIME1	O_REQ
Base +20	----	----	----	----	----	----	----	O_UND
Base +21	----	----	----	----	----	----	----	----

◆ **Digital Input Mode Setting:**

I_ACK: Input ACK Enable

1: Input ACK is enabled (input ACK will be asserted after input data is read by CPU or written to input FIFO)

0: Input ACK is disabled

I_REQ: Input REQ Strobe Enabled

1: Use I_REQ edge to latch input data

0: I_REQ is disabled

I_TIME0: Input Timer 0 Enable

1: Input is sampled by falling edge of Counter 0 output (COUT0)

0: Input Timer 0 is disabled

I_FIFO: Input FIFO Enable Mode

1: Input FIFO is enabled (input data is saved to input FIFO)

0: Input FIFO is disabled

TRGPOL: Input Trigger Polarity

1: I_TRG is Rising Edge Active

0: I_TRG is Falling Edge Active

I_TRG: External Trigger Enable

1: Wait until I_TRG signal is active, digital input sampling will begin after a rising or falling edge of I_TRG is coming.

0: Start input sampling immediately (if input control register is set)

DIN_EN: Digital Input Enable

1: Digital Input Enable

0: Digital Input Disabled, when this bit is set as 0, all digital input operation will be stopped.

◆ **Digital Output Mode Setting:**

O_ACK: Output ACK Enable

1: Output ACK is enabled, the output circuit will wait for O_ACK after O_REQ strobe is asserted.

0: Output ACK is disabled

O_REQ: Output REQ Enable

1: Output REQ is enabled, an O_REQ strobe will be generated after output data is ready

0: Output REQ is disabled

O_TIME1: Output Timer 1 Enable

1: Output Timer 1 is enabled, output data is moved from output FIFO to DO registers when output of Counter1 goes low.

0: Output Counter 1 is disabled

O_FIFO: Output FIFO Enable

1: Output FIFO is enabled (output data is moved from output FIFO)

0: Output FIFO is disabled

O_TRG: Digital Output Trigger Signal
 This bit is used to control the O_TRG output of PCI-7200, the signal is on CN1 pin 36 of PCI-7200 or CN1 pin 26 of cPCI-7200 when
 1: O_TRG 1 goes High (1)
 0: O_TRG 1 goes Low (0)

◆ **Digital I/O FIFO Status:**

I_OVR: Input data overrun
 1: Digital Input FIFO is full (overrun) during input data transfer
 0: No input data overrun occurred
 Input data overrun occurred, the I_OVR bit is set when input FIFO is full and there is new input data coming in. This bit can be cleared by writing "1" to it.

O_UND: Output data FIFO is underrun
 1: Output FIFO is empty during output data transfer
 0: No output data underrun occurred
 Output data underrun, the O_UND bit is set when output FIFO is empty and the output request for new data, this bit can be cleared by writing "1" to it.

3.5 Interrupt Status & Control Register (BASE + 1C)

The interrupt modes/status are set/checked through this register.

Address: BASE + 1C

Attribute: READ/WRITE

Data Format:

Byte	7	6	5	4	3	2	1	0
Base +1C	SI_TO	SI_REQ	SO_ACK	T2_EN	T1_EN	T0_EN	II_REQ	IO_ACK
Base +1D	FIFOFF	FIFOEF	FIFORST	REQ_NEG	T1_T2	T0_T2	SI_T2	SI_T1
Base +1E	----	----	----	----	----	----	----	----
Base +1F	----	----	----	----	----	----	----	----

◆ **Interrupt Control:**

In PCI-7200, the interrupt can be triggered by many signal sources such as O_ACK, I_REQ, timer 0, timer 1, and timer 2. The following bits control the interrupt source:

IO_ACK: Interrupt is triggered by O_ACK signal.

- 1: O_ACK interrupt is enabled
- 0: O_ACK interrupt is disabled

II_REQ: Interrupt is triggered by I_REQ signal.

- 1: I_REQ interrupt is enabled
- 0: I_REQ interrupt is disabled

T0_EN: Interrupt is triggered by timer 0 output.

- 1: Timer 0 interrupt is enabled
- 0: Timer 0 interrupt is disabled

T1_EN: Interrupt is triggered by timer 1 output.

- 1: Timer 1 interrupt is enabled
- 0: Timer 1 interrupt is disabled

T2_EN: Interrupt is triggered by timer 2 output.

- 1: Timer 2 interrupt is enabled
- 0: Timer 2 interrupt is disabled

◆ **Interrupt Status:**

The following bits are used to check interrupt status:

SO_ACK: Status of O_ACK interrupt

- 1: O_ACK Interrupt occurred
- 0: No O_ACK interrupt

SI_REQ: Status of I_REQ interrupt

- 1: I_REQ Interrupt occurred
- 0: No I_REQ Interrupt

SI_T0: Status of timer 0 interrupt

- 1: OUT0 (output of timer 0) Interrupt occurred
- 0: No timer 0 Interrupt

SI_T1: Status of timer 1 interrupt

- 1: OUT1 (output of timer 1) Interrupt occurred
- 0: No timer 1 Interrupt

SI_T2: Status of timer 2 interrupt

- 1: OUT2 (output of timer 2) interrupt occurred
- 0: No timer 2 Interrupt

Note: Writing 1 to the corresponding bit of the register can clear all the interrupt status. In order to make the interrupt work properly, the interrupt service routine has to clear all the interrupt status before end of the ISR.

◆ **Timer Configuration Control:**

The 8254 timer on the PCI-7200 can be configured as either timer 0 cascaded with timer 2 or timer 1 cascaded with timer 2. These configuration are controlled by the following bits:

T0_T2: Timer 0 is cascaded with timer 2

- 1: Timer 0 and timer 2 are cascaded together, output of timer 2 connects to the clock input of timer 0.
- 0: Not cascaded, the 4 MHz clock is connected to the timer 0 clock input.

T1_T2: Timer 1 is cascaded with timer 2

1: Timer 1 and timer 2 are cascaded together, output of timer 2 connects to the clock input of timer 1.

0: Not cascaded, the 4 MHz clock is connected to the timer 1 clock input.

◆ **I_REQ Polarity Selection:**

When the input sampling is controlled by the I_REQ signal only, the I_REQ can be programmed to be rising edge active or falling edge active.

REQ_NEG: I_REQ trigger polarity

1: latch input data on falling edge of I_REQ

0: latch input data on rising edge of I_REQ

◆ **FIFO Control and Status (cPCI-7200 only):**

The cPCI-7200 has an extra 2K samples digital input FIFO. The FIFO can be cleared and monitored by the following bits:

FIFORST (Write only): Clear the on-board DI FIFO

1: Write 1 to clear the data of the FIFO.

0: No operation.

FIFOEF (Read only): Empty flag of the DI FIFO

1: DI FIFO is empty.

0: DI FIFO is not empty.

FIFOFF (Read only): Full flag of the DI FIFO

1: DI FIFO is full.

0: DI FIFO is not full.

Note: The cPCI-7200 has 2 cascaded DI FIFOs. One is located in the PCI controller chip, the other one is on the PCI-7200 board. The above bits only control the on-board FIFO. In order to control the on-chip FIFO, please refer to the AMCC-5933 data book.

3.6 8254 Timer Registers (BASE + 0)

The 8254 timer/ counter IC occupies 4 I/O address. Users can refer to Tundra's or Intel's data sheet for a full description of the 8254 features. You can download the 8254 data sheet from the following web site:

<http://support.intel.com/support/controllers/peripheral/231164.htm> or

<http://www.tundra.com> (for Tundra's 82C54 datasheet.)

Address	Read	Write
Base + 0	Counter 0	Counter 0
Base + 4	Counter 1	Counter 1
Base + 8	Counter 2	Counter 2
Base + C	---	CLK Control CW0

4

Operation Theorem

In PCI-7200, there are four data transfer modes can be used for digital I/O access and control, these modes are:

1. **Direct Program Control:** the digital inputs and outputs can be read/written and controlled by its corresponding I/O port address directly.
2. **Internal Timer Pacer Mode:** the digital input and output operations are paced by internal timer pacer and transferred by bus mastering DMA.
3. **External Clock Mode:** the digital input operation is clocked by external I_REQ strobe and transferred by bus mastering DMA.
4. **Handshaking:** through REQ and ACK signals, the digital I/O can have simple handshaking data transfer.

4.1 Direct Program Control

The digital I/O operations can be controlled by I/O port **BASE+10** for digital input and **BASE+14** for digital output.

The I/O port address BASE is assigned by system BIOS, please refer to Section 5 for more detailed description.

The digital OUT operation is:

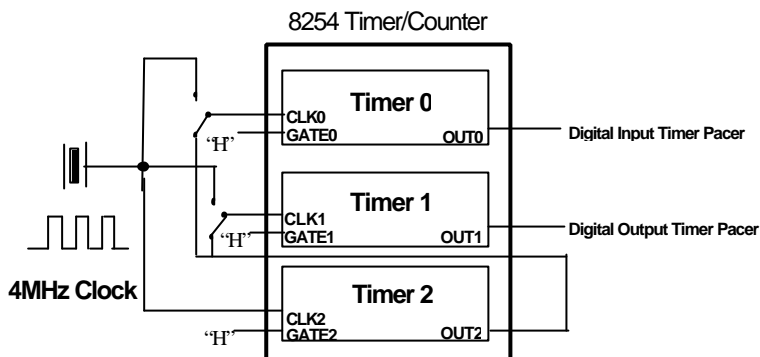
```
outport (BASE+14, 0xAAAAAAAA) // (A : 0 ~ F)
```

The digital IN operation is:

```
value = inport (BASE+10) // The input status is save in the  
// value variable
```

4.2 Timer Pacer Mode

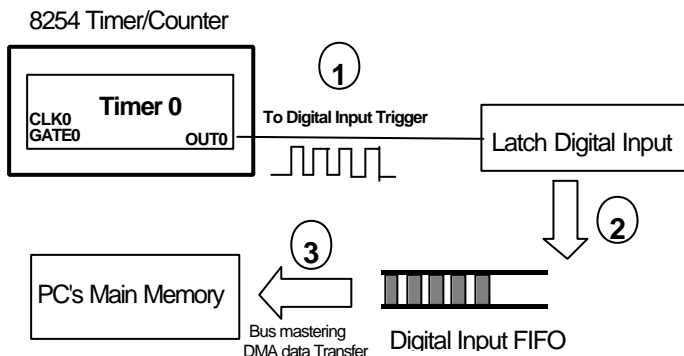
The digital I/O access control is clocked by timer pacer, which is generated by an interval programming timer/counter chip 8254. There are three timers on the 8254. The timer 0 is used to generate timer pacer for digital input, and timer 1 is used for digital output. The configuration is illustrated as below.



The operation sequences are:

1. Define the frequency (timer pacer rate)
2. The digital input data are saved in FIFO after a timer pacer pulse is generated. The sampling is controlled by timer pacer.
3. The data saved in FIFO will be transferred 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.

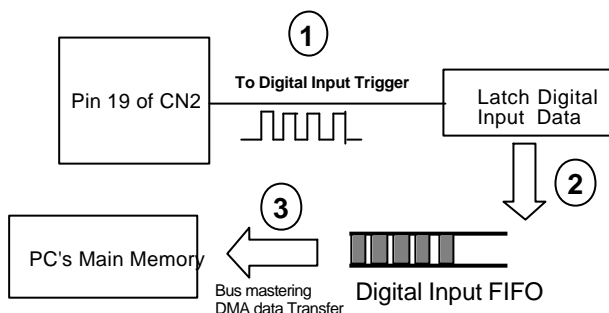
The operation flow is show as following:



4.3 External Clock Mode

The digital input is clocked by external strobe, which is from the Pin 19 (I_REQ) of CN2 (PCI-7200) or Pin 24 of CN1 (cPCI-7200). The operation sequence is very similar to Timer Pacer Trigger. The only difference is the clock source.

1. The external input strobe is generated from outside device, and go through the Pin 19 (I_REQ) of CN2 and to latch the digital input.
2. The digital input data are saved in FIFO after an I/O strobe signal is coming in.
3. The data saved in input FIFO will be transferred to main memory on your computer system directly. This is controlled by bus mastering DMA control, this function is supported by PCI

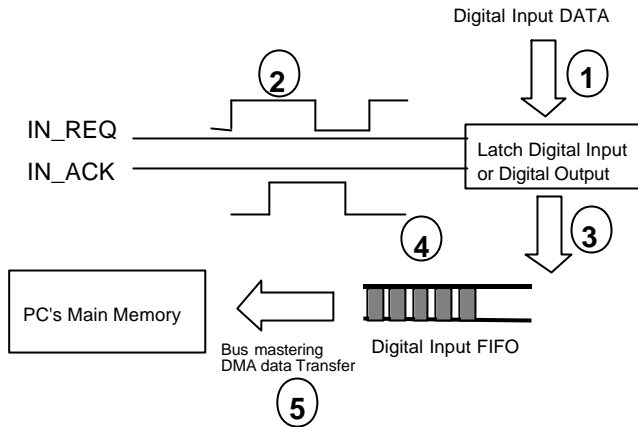


4.4 Handshaking

In PCI-7200, it also supports a handshaking digital I/O transfer mode. That is, after input data is ready, an I_REQ is sent from external device, and I_ACK will go high to acknowledge the data already accessed.

I_REQ & I_ACK for Digital Input

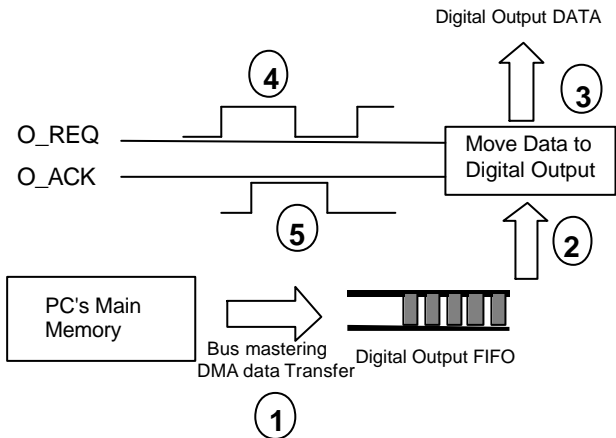
1. Digital Input Data is ready
2. An I_REQ signal is generated for digital input operation
3. Digital input data is saved to FIFO
4. An I_ACK signal is generated and sent to outside device
5. If the FIFO is not empty and PCI bus is not occupied, the data will be transferred to main memory



O_REQ & O_ACK for Digital Output

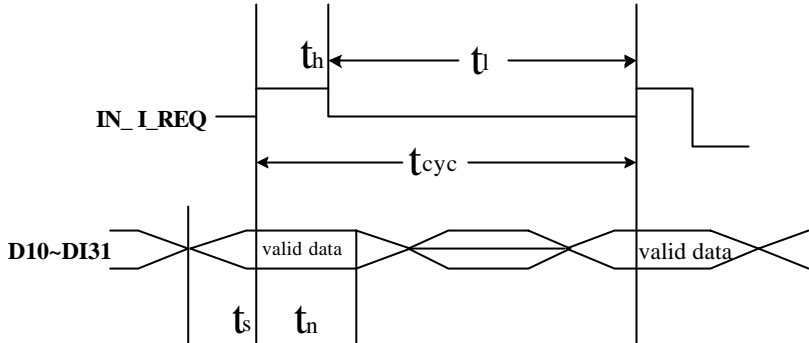
1. Digital Output Data is moved from PC memory to FIFO of PCI-7200 by using DMA data mastering data transfer.
2. Move output data from FIFO to digital output circuit.
3. Output data is ready.
4. An O_REQ signal is generated and sent to outside device.
5. After an O_ACK is got, the step 2 to step 5 will be repeated again.

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



4.5 Timing Characteristic

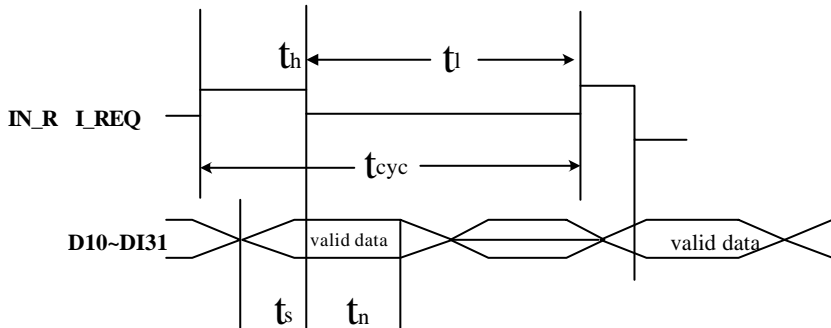
1. I_REQ as input data strobe (Rising Edge Active)



$t_h \cong 60\text{ns}$ $t_l \cong 60\text{ns}$ $t_{cyc} \cong 5 \text{ PCI CLK Cycle}$

$t_s \cong 2\text{ns}$ $t_n \cong 30\text{ns}$

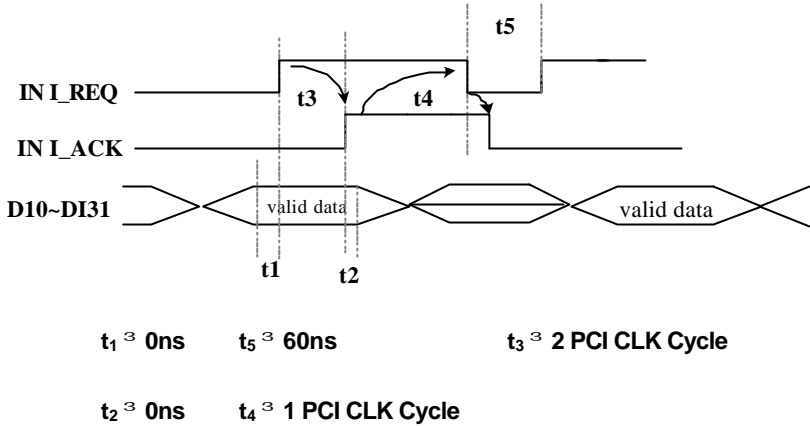
2. I_REQ as input data strobe (Falling Edge Active)



$t_h \cong 60\text{ns}$ $t_l \cong 60\text{ns}$ $t_{cyc} \cong 5 \text{ PCI CLK Cycle}$

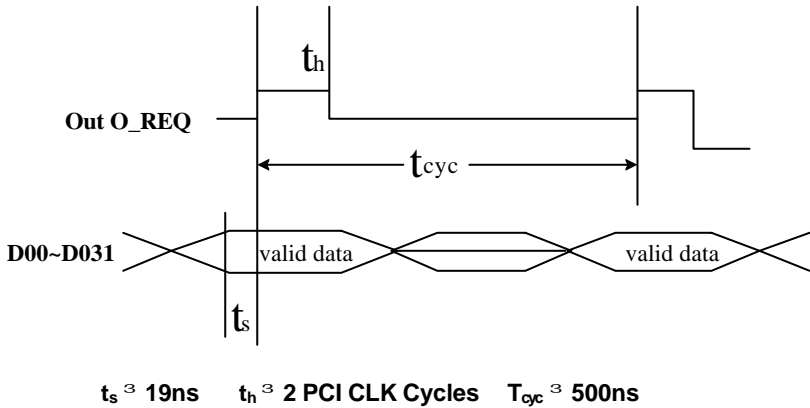
$t_s \cong 2\text{ns}$ $t_n \cong 30\text{ns}$

3. I_REQ & I_ACK Handshaking

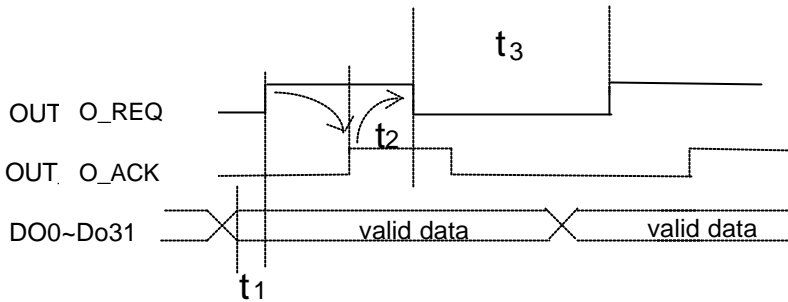


Note: I_REQ must be asserted until I_ACK asserts, I_ACK will be asserted until I_REQ de-asserts.

4. O_REQ as output data strobe



5. O_REQ & O_ACK Handshaking



$t_1 \geq 19\text{ns}$ $t_2 \geq 1 \text{ PCI CLK Cycle}$ $t_3 \geq 5 \text{ PCI CLK Cycle}$

Note: O_ACK must be de-asserted before O_REQ asserts, O_ACK can be asserted any time after O_REQ asserts, O_REQ will be reasserted after O_ACK is asserted.

5

C/C++ Libraries

This chapter describes the software library for operating this card. Only the functions in DOS library and Windows 95 DLL are described. Please refer to the PCIS-DASK function reference manual, which included in ADLINK CD, for the descriptions of the Windows 98/NT/2000 DLL functions.

The function prototypes and some useful constants are defined in the header files LIB directory (DOS) and INCLUDE directory (Windows 95). For Windows 95 DLL, the developing environment can be Visual Basic 4.0 or above, Visual C/C++ 4.0 or above, Borland C++ 5.0 or above, Borland Delphi 2.x (32-bit) or above, or any Windows programming language that allows calls to a DLL. It provides the C/C++, VB, and Delphi include files.

5.1 Libraries Installation

Please refer to the “**Software Installation Guide**” for the detail information about how to install the software libraries for DOS, or Windows 95 DLL, or PCIS-DASK for Windows 98/NT/2000.

The device drivers and DLL functions of Windows 98/NT/2000 are included in the PCIS-DASK. Please refer the PCIS-DASK user’s guide and function reference, which included in the ADLINK CD, for detailed programming information.

5.2 Programming Guide

5.2.1 Naming Convention

The functions of the NuDAQ PCI cards or NuIPC CompactPCI cards' software driver are using full-names to represent the functions' real meaning. The naming convention rules are:

In DOS Environment :

`_{hardware_model}_{action_name}`. e.g. `_7200_Initial()`.

All functions in PCI-7200 driver are with 7200 as {hardware_model}. But they can be used by PCI-7200, cPCI-7200.

In order to recognize the difference between DOS library and Windows 95 library, a capital "W" is put on the head of each function name of the Windows 95 DLL driver. e.g. `w_7200_Initial()`.

5.2.2 Data Types

We defined some data type in `Pci_7200.h` (DOS) and `Acl_pci.h` (Windows 95). These data types are used by NuDAQ Cards' library. We suggest you to use these data types in your application programs. The following table shows the data type names and their range.

Type Name	Description	Range
U8	8-bit ASCII character	0 to 255
I16	16-bit signed integer	-32768 to 32767
U16	16-bit unsigned integer	0 to 65535
I32	32-bit signed integer	-2147483648 to 2147483647
U32	32-bit single-precision floating-point	0 to 4294967295
F32	32-bit single-precision floating-point	-3.402823E38 to 3.402823E38
F64	64-bit double-precision floating-point	-1.797683134862315E308 to 1.797683134862315E309
Boolean	Boolean logic value	TRUE, FALSE

5.3 _7200_Initial

@ *Description*

A PCI-7200 card is initialized according to the card number.

Because the PCI-7200 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-7200 card has to be initialized by this function before calling other functions.

Note: Because configuration of PCI-7200 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++ (Windows 95)

```
int W_7200_Initial (U8 card_number, U16 *base_addressss, U8 *irq_no)
```

Visual Basic (Windows 95)

```
W_7200_Initial (ByVal card_number As Byte, base_addressss As Integer, irq_no As Byte) As Long
```

C/C++ (DOS)

```
int _7200_Initial (U8 card_number, U16 *base_addressss, U8 *irq_no)
```

@ *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.

base_address :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.

@ *Return Code*

```
ERR_NoError  
ERR_InvalidBoardNumber  
ERR_PCIBiosNotExist  
ERR_PCICardNotExist  
ERR_PCIIrqNotExist  
ERR_BaseAddressError
```

5.4 _7200_Switch_Card_No

@ Description

After initialized more than one PCI-7200 card, this function is used to select which card is used currently.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_Switch_Card_No (U8 card_number)
```

Visual Basic (Windows 95)

```
W_7200_Switch_Card_No (ByVal card_number As Byte) As Long
```

C/C++ (DOS)

```
int _7200_Switch_Card_No (U8 card_number)
```

@ Argument

card_number : The card number to be initialized, four cards can be initialized, the card number must be CARD_1, CARD_2, CARD_3, or CARD_4, but only one card is active.

@ Return Code

```
ERR_NoError  
ERR_InvalidBoardNoInit
```

5.5 _7200_AUX_DI

@ Description

Read data from auxiliary digital input port of cPCI-7200 card. You can get all 4 bits input data by using this function.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_AUX_DI (U32 *aux_di)
```

Visual Basic (Windows 95)

```
W_7200_DI (aux_di As Long) As Long
```

C/C++ (DOS)

```
int _7200_DI (U32 *aux_di)
```

@ Argument

aux_di : returns 4-bit value from auxiliary digital input port.

@ Return Code

```
ERR_NoError  
ERR_FunctionNotAvailable
```

5.6 _7200_AUX_DI_Channel

@ *Description*

Read data from auxiliary digital input channel of cPCI-7200 card. There are 4 digital input channels on the cPCI-7200 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++ (Windows 95)

```
int W_7200_AUX_DI_Channel (U8 di_ch_no, Boolean
*aux_data)
```

Visual Basic (Windows 95)

```
W_7200_AUX_DI_Channel (ByVal di_ch_no As Byte,
aux_data As Byte) As Long
```

C/C++ (DOS)

```
int _7200_AUX_DI_Channel (U8 di_ch_no, Boolean
*aux_data)
```

@ *Argument*

di_ch_no : the DI channel number, the value has to be set within 0 and 3.

aux_data : return value, either 0 or 1.

@ *Return Code*

```
ERR_NoError, ERR_InvalidDIChannel, ERR_FunctionNotAvailable
```

5.7 _7200_AUX_DO

@ *Description*

Write data to auxiliary digital output port. There are 4 auxiliary digital outputs on the cPCI-7200.

@ *Syntax*

Visual C++ (Windows 95)

```
int W_7200_AUX_DO (U32 aux_do)
```

Visual Basic (Windows 95)

```
W_7200_AUX_DO (ByVal aux_do As Long) As Long
```

C/C++ (DOS)

```
int _7200_AUX_DO (U32 aux_do)
```

@ *Argument*

aux_do : value will be written to auxiliary digital output port

@ *Return Code*

```
ERR_NoError
ERR_FunctionNotAvailable
```

5.8 _7200_AUX_DO_Channel

@ *Description*

Write data to auxiliary digital output channel (bit). There are 4 auxiliary digital output channels on the cPCI-7200. When performs this function, the digital output data is written to the corresponding channel.

channel means each bit of digital input port

@ *Syntax*

Visual C++ (Windows 95)

```
int W_7200_AUX_DO_Channel (U8 do_ch_no, Boolean aux_data)
```

Visual Basic (Windows 95)

```
W_7200_AUX_DO_Channel (ByVal do_ch_no As Byte, ByVal  
aux_data As Byte) As Long
```

C/C++ (DOS)

```
int _7200_AUX_DO_Channel (U8 do_ch_no, Boolean aux_data)
```

@ *Argument*

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

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

@ *Return Code*

```
ERR_NoError  
ERR_InvalidDOChannel  
ERR_FunctionNotAvailable
```

5.9 _7200_DI

@ *Description*

This function is used to read data from digital input port. There are 32-bit digital inputs on the PCI-7200. You can get all 32 input data from _7200_DI by using this function.

@ *Syntax*

Visual C++ (Windows 95)

```
int W_7200_DI (U32 *di_data)
```

Visual Basic (Windows 95)

```
W_7200_DI (di_data As Long) As Long
```

C/C++ (DOS)

```
int _7200_DI (U32 *di_data)
```

@ *Argument*

di_data : returns all 32-bit value from digital port.

@ *Return Code*

```
ERR_NoError
```

5.10 _7200_DI_Channel

@ Description

This function is used to read data from digital input channels (bit). There are 32 digital input channels on the PCI-7200. When performs this function, the 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++ (Windows 95)

```
int W_7200_DI_Channel (U8 di_ch_no, Boolean *di_data)
```

Visual Basic (Windows 95)

```
W_7200_DI_Channel (ByVal di_ch_no As Byte, di_data As Byte)  
As Long
```

C/C++ (DOS)

```
int _7200_DI_Channel (U8 di_ch_no, Boolean *di_data)
```

@ Argument

di_ch_no : the DI channel number, the value has to be set within 0 and 31.

di_data : return value, either 0 or 1.

@ Return Code

```
ERR_NoError  
ERR_InvalidDIChannel
```

5.11 _7200_DO

@ Description

This function is used to write data to digital output port. There are 32 digital outputs on the PCI-7200.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_DO (U32 do_data)
```

Visual Basic (Windows 95)

```
W_7200_DO (ByVal do_data As Long) As Long
```

C/C++ (DOS)

```
int _7200_DO (U32 do_data)
```

@ Argument

do_data : value will be written to digital output port

@ Return Code

```
ERR_NoError
```

5.12 _7200_DO_Channel

@ **Description**

This function is used to write data to digital output channels (bit). There are 32 digital output channels on the PCI-7200. When performs this function, the digital output data is written to the corresponding channel.

channel means each bit of digital input port

@ **Syntax**

Visual C++ (Windows 95)

```
int W_7200_DO_Channel (U8 do_ch_no, Boolean do_data)
```

Visual Basic (Windows 95)

```
W_7200_DO_Channel (ByVal do_ch_no As Byte, ByVal do_data As  
Byte) As Long
```

C/C++ (DOS)

```
int _7200_DO_Channel (U8 do_ch_no, Boolean do_data)
```

@ **Argument**

do_ch_no : the DO channel number, the value has to be set within 0 and 31.

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

@ **Return Code**

ERR_NoError

ERR_InvalidDOChannel

5.13 _7200_Alloc_DMA_Mem

@ Description

Contact Windows 95/98 system to allocate a block of contiguous memory for single-buffered DMA transfer. This function is only available in Windows 95/98 version.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_Alloc_DMA_Mem (U32 *buff, U32 *handle, U32
buf_size, U32 *actual_size)
```

Visual Basic (Windows 95)

```
W_7200_Alloc_DMA_Mem (buff As Long, handle As Long, ByVal
buf_size As Long, actual_size As Long ) As Long
```

@ Argument

buff : The start address of the user buffer for DMA data transfer. This buffer will be attached to the DMA memory allocated by this function. When using this DMA memory handle as an argument of `W_7200_DI_DMA_Start` function, DI data will be copied to this buffer. When using this DMA memory handle as an argument of `W_7200_DO_DMA_Start` function, the data stored in this buffer will be the DO data.

handle: The handle of system DMA memory returned from system. Use this handle in `_7200_DI_DMA_Start` or `_7200_DO_DMA_Start`.

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

actual_size: The actual size system allocate for DMA memory. The unit is BYTE. If system is not able to get a block of contiguous memory of specified *buf_size*, it will allocate a block of memory as large as it can. In this case, this function returns `ERR_SmallerDMAMemAllocated`, and *actual_size* denotes the actual size of allocated memory.

@ Return Code

```
ERR_NoError
ERR_SmallerDMAMemAllocated
```

5.14 _7200_Free_DMA_Mem

@ Description

Release the system DMA memory under Windows 95/98 environment. This function is only available in Windows 95/98 version.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_Free_DMA_Mem (U32 handle)
```

Visual Basic (Windows 95)

```
W_7200_Free_DMA_Mem (ByVal handle As Long ) As Long
```

@ Argument

handle: The handle of system DMA memory to release.

@ Return Code

```
ERR_NoError
```

5.15 _7200_Alloc_DBDMA_Mem

@ Description

Contact Windows 95/98 system to allocate a block of contiguous memory as circular buffer for double-buffered DMA DI transfer. This function is only available in Windows 95/98 version. For double-buffered transfer principle, please refer to Section 6 “*Double Buffered Mode Principle*”.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_Alloc_DBDMA_Mem (U32 *buff, U32 *handle, U32  
buf_size, U32 *actual_size)
```

Visual Basic (Windows 95)

```
W_7200_Alloc_DBDMA_Mem (buff As Long, handle As Long, ByVal  
buf_size As Long, actual_size As Long) As Long
```

@ Argument

buff : There is a dummy buffer attached to the DMA memory this function will allocate. But this buffer need to have size equal to or more than *buf_size* bytes.

handle: The handle of system DMA memory returned from system. Use this handle in *_7200_DI_DMA_Start*.

buf_size: Bytes to allocate. This is the half size of circular buffer in byte. That is, this is the size of each half buffer in byte.

actual_size: The actual DMA memory size system allocate for each half buffer. If system is not able to get a block of contiguous memory of specified *buf_size*, it will allocate a block of memory as large as it can. In this case, this function returns *ERR_SmallerDMAMemAllocated*, and *actual_size* denotes

the actual size of allocated memory for each half of circular buffer.

@ Return Code

```
ERR_NoError  
ERR_SmallerDMAMemAllocated
```

5.16 _7200_Free_DBDMA_Mem

@ Description

Release a system circular buffer DMA memory under Windows 95/98 environment. This function is only available in Windows 95/98 version. For double-buffered transfer principle, please refer to Section 6 “*Double Buffered Mode Principle*”.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_Free_DBDMA_Mem (U32 handle)
```

Visual Basic (Windows 95)

```
W_7200_Free_DBDMA_Mem (ByVal handle As Long ) As Long
```

@ Argument

handle: The handle of system DMA memory to release.

@ Return Code

```
ERR_NoError
```

5.17 _7200_DI_DMA_Start

@ Description

The function will perform digital input N times with DMA data transfer by using one of the following four sampling modes :

1. pacer trigger (internal timer trigger)
2. external rising edge I_IRQ
3. external falling edge I_IRQ
4. I_REQ & I_ACK handshaking

It will take place in the background which will not stop until the Nth input data is transferred or your program execute _7200_DI_DMA_Stop function to stop the process.

After executing this function, it is necessary to check the status of the operation by using the function _7200_DI_DMA_Status. The PCI-7200 Bus mastering DMA is different from traditional PC style DMA. Its description is as follow :

Bus Mastering DMA mode of PCI-7200 :

PCI bus mastering offers the highest possible speed available on the PCI-7200. When the function `_7200_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 max. groups. PCI-7200 bus mastering works as follows:

1. To set up bus mastering, first do all normal PCI-7200 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-7200 is plugged 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 long words!) transferred during the bus master operation and can be a large number up to 64 million (2^{26}) bytes. Since the PCI-7200 transfers are always long words, this is 16 million long words (2^{24}).
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-7200 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).

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

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_DI_DMA_Start (U8 mode, U32 count, U32 handle,  
Boolean wait_trg, U8 trg_pol, Boolean clear_fifo, Boolean  
disable_di)
```

Visual Basic (Windows 95)

```
W_7200_DI_DMA_Start (ByVal mode As Byte, ByVal count As Long,  
ByVal handle As Long, ByVal wait_trg as Byte, ByVal trg_pol  
As Byte, ByVal clear_fifo As Byte, ByVal disable_di As Byte)  
As Long
```

C/C++ (DOS)

```
int _7200_DI_DMA_Start (U8 mode, U32 count, U32 *di_buffer,  
Boolean wait_trig, U8 trig_pol, Boolean clear_fifo, Boolean  
disable_di)
```

@ Argument

mode : Digital Input trigger modes

DI_MODE0 : Internal timer pacer (TIME 0)

DI_MODE1 : external signal I_REQ rising edge

DI_MODE2 : external signal I_REQ falling edge

DI_MODE3 : I_REQ & I_ACK handshaking

count : For non-double-buffered DI, this parameter denotes the number of digital input samples to read. For double-buffered DI, it is the size of circular buffer (in samples, not in bytes!).

handle (Win 95): the handle of system DMA memory. In Windows 95 environment, before calling `_7200_DI_DMA_Start`, depending on using double-buffer mode or not, either `_7200_Alloc_DMA_Mem` or `_7200_Alloc_DBDMA_Mem` must be called to allocate a contiguous DMA memory and get the handle of it.

di_buffer (DOS): If double buffer mode is disabled, this is the start address of the memory buffer to store the DI data. If double buffer mode is enabled, this memory buffer is actually of no use. But the buffer size still must be larger than the number of *count* (that is, $\text{count} \times 4$ bytes). You can use this buffer as transfer buffer in `_7200_DblBufferTransfer` to make use of this buffer.

****This memory should be double-word alignment.**

wait_trig : The waiting status of trigger

DI_NONWAITING : the input sampling will be
start immediately

DI_WAITING : the input samples waiting rising or falling edge trigger
to start DI

trig_pol : trigger polarity

DI_RISING : rising edge trigger

DI_FALLING : falling edge trigger

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

```
ERR_NoError  
ERR_BoardNoInit  
ERR_InvalidDIOMode  
ERR_InvalidDIOCnt  
ERR_NotDWordAlign  
ERR_DMATransferNotAllowed
```

5.18 _7200_DI_DMA_Status

@ Description

Since the _7200_DI_DMA_Start function is executed in background, you can issue this function to check its operation status. This function only works when double-buffer mode is set as disable.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_DI_DMA_Status (U8 *status, U32 *count)
```

Visual Basic (Windows 95)

```
W_7200_AD_Status (status As Byte, count As Long ) As Long
```

C/C++ (DOS)

```
int _7200_AD_DMA_Status (U8 *status, U32 *count)
```

@ Argument

status : status of the DMA data transfer

- 0 : DI_DMA_STOP : DMA is completed
- 1 : DI_DMA_RUN : DMA is not completed

count : the numbers of DI data which has been transferred.

@ Return Code

```
ERR_NoError
```

5.19 _7200_DI_DMA_Stop

@ Description

This function is used to stop the DMA data transferring. After executing this function, the _7200_DI_DMA_Start function is stopped. The function returns the number of the data which has been transferred, no matter if the digital input DMA data transfer is stopped by this function or by the DMA terminal count ISR.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_DI_DMA_Stop (U32 * count)
```

Visual Basic (Windows 95)

```
W_7200_DI_DMA_Stop ( count As Long ) As Long
```

C/C++ (DOS)

```
int _7200_DI_DMA_Stop (U32 *count)
```

@ Argument

count : the number of DI data which has been transferred.

@ Return Code

ERR_NoError

ERR_BoardNoInit

5.20 _7200_DblBufferMode

@ Description

This function is used to enable or disable double buffer mode for DMA DI operation.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_DblBufferMode (Boolean db_flag)
```

Visual Basic (Windows 95)

```
W_7200_DblBufferMode (ByVal db_flag As Byte) As Long
```

C/C++ (DOS)

```
int _7200_CheckHalfReady (Boolean db_flag)
```

@ Argument

db_flag : 1 : double buffer mode enabled
0 : double buffer mode disabled

@ Return Code

ERR_NoError

5.21 _7200_CheckHalfReady

@ *Description*

When you use `_7200_DI_DMA_Start` to sample digital input data and double buffer mode is set as enable. You must use `_7200_CheckHalfReady` to check data ready (data half full) or not in the circular buffer, and using `_7200_DblBufferTransfer` to get data.

@ *Syntax*

Visual C++ (Windows 95)

```
int W_7200_CheckHalfReady (Boolean * halfReady)
```

Visual Basic (Windows 95)

```
W_7200_CheckHalfReady (halfReady As Byte) As Long
```

C/C++ (DOS)

```
int _7200_CheckHalfReady (Booelan *halfReady)
```

@ *Argument*

halfReady: 1 (TRUE) or 0 (FALSE)

@ *Return Code*

```
ERR_NoError
```

5.22 _7200_DblBufferTransfer

@ *Description*

Using this function to copy the input data in the circular buffer to the transfer buffer. It copies half of the circular buffer, either first half or second half, to the transfer buffer.

@ *Syntax*

Visual C++ (Windows 95)

```
int W_7200_DblBufferTransfer (U32 *userBuffer)
```

Visual Basic (Windows 95)

```
W_7200_DblBufferTransfer (userBuffer As Long) As Long
```

C/C++ (DOS)

```
int _7200_DblBufferTransfer (U32 *userBuffer)
```

@ *Argument*

userBuffer: the start address of the transfer buffer. `W_7200_DblBufferTransfer` function copies half of the circular buffer to `userBuffer`.

@ *Return Code*

```
ERR_NoError, ERR_NotHalfReady
```

5.23 _7200_GetOverrunStatus

@ Description

When you use `_7200_DI_DMA_Start` to convert Digital I/O data with double buffer mode enabled, and if you do not use `_7200_DblBufferTransfer` to move converted data then the double buffer overrun will occur, using this function to check overrun count.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_GetOverrunStatus (U32 * overrunCount)
```

Visual Basic (Windows 95)

```
int W_7200_GetOverrunStatus (overrunCount As Long) As Long
```

C/C++ (DOS)

```
int _7200_GetOverrunStatus (U32 *overrunCount )
```

@ Argument

overrunCount : number of overrun counts.

@ Return Code

```
ERR_NoError
```

5.24 _7200_DO_DMA_Start

@ Description

The function will perform digital output N times with DMA data transfer by using the following four sampling modes :

1. pacer trigger (internal timer trigger, TIME 1)
2. Internal timer pacer with O_REQ enable
3. O_REQ & O_ACK handshaking

It will takes place in the background which will not be stop until the Nth conversion has been completed or your program execute `_7200_DO_DMA_Stop` function to stop the process. After executing this function, it is necessary to check the status of the operation by using the function `_7200_DO_DMA_Status`.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_DO_DMA_Start (U8 mode, U32 count, U32 handle,  
Boolean repeat)
```

Visual Basic (Windows 95)

```
W_7200_DO_DMA_Start (ByVal mode As Byte, ByVal count As Long,  
ByVal handle As Long, ByVal repeat as Byte) As Long
```


C/C++ (DOS)

```
int _7200_DO_DMA_Start (U8 mode, U32 count, U32 *do_buffer,  
    Boolean repeat)
```

@ Argument

mode : Digital output trigger modes

DO_MODE_0 : Internal timer pacer (TIME 1)

DO_MODE_1 : Internal timer pacer with O_REQ enable

DO_MODE_2 : O_REQ & I_REQ handshaking

count : the sample number of digital output data (in samples, not in bytes!)

handle (Win 95): the handle of system DMA memory. In Windows 95 environment, before calling W_7200_DO_DMA_Start, W_7200_Alloc_DMA_Mem must be called to allocate a contiguous DMA memory and get the handle of it. Also W_7200_Alloc_DMA_Mem will attach a buffer to DMA memory. The DO data is stored in the buffer attached to this handle.

do_buffer (DOS) : the start address of the memory buffer to store the DO data.

**** This memory should be double-word alignment**

repeat : The digital output will be continuous or only one shot.

CONTINUOUS: digital output will be continuous until the _7200_DO_DMA_STOP is called.

ONE_SHOT : digital output only one-shot.

@ Return Code

```
ERR_NoError  
ERR_InvalidDIOMode  
ERR_InvalidBoardNumber  
ERR_BoardNoInit  
ERR_InvalidDIOCnt  
ERR_NotDWordAlign  
ERR_DMATransferNotAllowed
```

5.25 _7200_DO_DMA_Status

@ Description

Since the _7200_DO_DMA_Start function is executed in background, you can issue the function _7200_DO_DMA_Status to check its operation status.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_DO_DMA_Status (U8 *status, U32 * count)
```

Visual Basic (Windows 95)

```
W_7200_DO_Status ( status As Byte, count As Long ) As Long
```

C/C++ (DOS)

```
int _7200_DO_DMA_Status (U8 *status , U32 *count)
```

@ Argument

status : status of the DMA data transfer

0 : DO_DMA_STOP : DMA is completed

1 : DO_DMA_RUN : DMA is not completed

count : the numbers of DO data which has been transferred.

@ Return Code

```
ERR_NoError
```

5.26 _7200_DO_DMA_Stop

@ Description

This function is used to stop the DMA DO operation. After executing this function, the _7200_DO_DMA_Start function is stopped. The function returns the number of the data which has been transferred, no matter if the digital output DMA data transfer is stopped by this function or by the DMA terminal count ISR.

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_DO_DMA_Stop (U32 *count)
```

Visual Basic (Windows 95)

```
W_7200_DO_DMA_Stop (count As Long) As Long
```

C/C++ (DOS)

```
int _7200_DO_DMA_Stop (U32 *count)
```

@ Argument

count : the number of digital output data which has been transferred.

@ Return Code

```
ERR_NoError
```

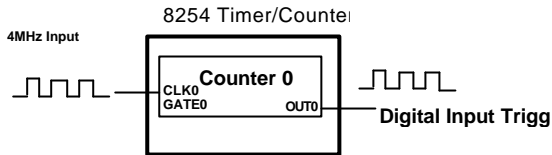
```
ERR_BoardNoInit
```

5.27 _7200_DI_Timer

@ Description

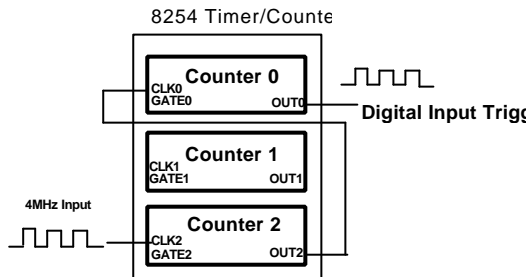
This function is used to set the internal timer pacer for digital input. There are two configurations for the internal timer pacer :

1. Non-cascaded (One COUNTER 0 only)



Timer pacer frequency = $4\text{MHz} / C0$

2. Cascaded (TIME2 cascaded with COUNTER0)



Timer pacer frequency = $4\text{MHz} / (C0 * C2)$

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_DI_Timer (U16 c0, U16 c2, Boolean mode)
```

Visual Basic (Windows 95)

```
W_7200_DI_Timer ( ByVal c0 As Integer, ByVal c2 As Integer,  
ByVal mode As Byte ) As Long
```

C/C++ (DOS)

```
int _7200_DI_Timer (U16 c0, U16 c2, Boolean mode)
```

@ Argument

- c0**: frequency divider of Counter #0. Valid value ranges from 2 to 65535.
- c2**: frequency divider of Counter #2. Valid value ranges from 2 to 65535.

Note : Since the Integer type in Visual Basic is signed integer. Its range is within -32768 and 32767. In Visual Basic, if you want to set c0 or c2 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.

mode : TIMER_NONCASCADE or TIMER_CASCADE

@ Return Code

```

ERR_NoError
ERR_InvalidBoardNumber
ERR_InvalidTimerMode
ERR_BoardNoInit

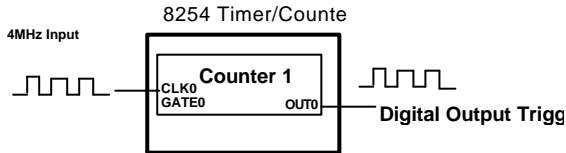
```

5.28 _7200_DO_Timer

@ Description

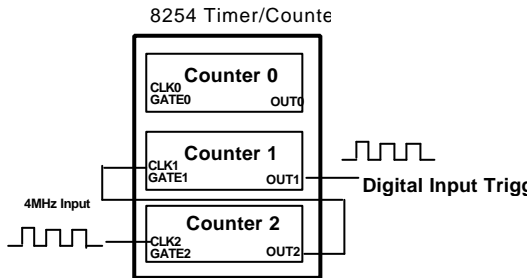
This function is used to set the internal timer pacer for digital output. There are two configurations for the internal timer pacer :

1. Non-cascaded (One COUNTER 0 only)



Timer pacer frequency = 4Mhz / C1

2. Cascaded (TIME2 cascaded with COUNTER0)



Timer pacer frequency = 4Mhz / (C1 * C2)

@ Syntax

Visual C++ (Windows 95)

```
int W_7200_DO_Timer (U16 c1, U16 c2, Boolean mode)
```

Visual Basic (Windows 95)

```
W_7200_DO_Timer (ByVal c1 As Integer, ByVal c2 As Integer,  
ByVal mode As Byte) As Long
```

C/C++ (DOS)

```
int _7200_DO_Timer (U16 c1, U16 c2, Boolean mode)
```

@ Argument

- c1:** frequency divider of Counter #1
- c2:** frequency divider of Counter #2

Note: Since the Integer type in Visual Basic is signed integer. Its range is within -32768 and 32767. In Visual Basic, if you want to set c1 or 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.

mode: TIMER_NONCASCADE or TIMER_CASCADE

@ Return Code

```
ERR_NoError  
ERR_InvalidBoardNumber  
ERR_InvalidTimerMode  
ERR_BoardNoInit
```

6

Double Buffer Mode Principle

The data buffer for double-buffered DMA DI operation is a circular buffer logically divided into two equal halves. The double-buffered DI begins when device starts writing data into the first half of the circular buffer (Figure 6-1a). After device begins writing to the second half of the circular buffer, you can copy the data from the first half into the transfer buffer (Figure 6-1b). You now can process the data in the transfer buffer according to application needs. After the board has filled the second half of the circular buffer, the board returns to the first half buffer and overwrites the old data. You now can copy the second half of the circular buffer to the transfer buffer (Figure 6-1c). The data in the transfer buffer is again available for process. The process can be repeated endlessly to provide a continuous stream of data to your application (Figure 6-1d).

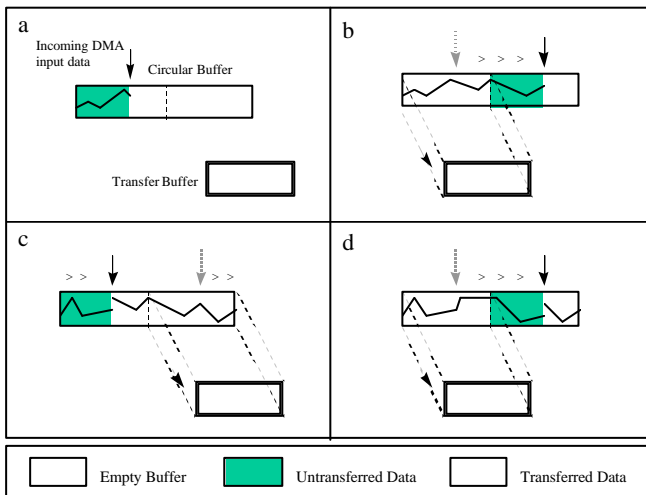


Figure 6-1

The PCI-7200 double buffer mode functions were designed according to the principle described above. If you use `_7200_DblBufferMode()` to enable double buffer mode, the following `_7200_DI_DMA_Start()` will perform double-buffered DMA DI. You can call `_7200_CheckHalfReady()` to check if data in the circular buffer is half-full and ready for copying to the transfer buffer. Then you can call `_7200_DblBufferTransfer()` to copy data from the ready half buffer to the transfer buffer.

In Win-95 version, `W_7200_Alloc_DBDMA_Mem()` is needed to allocate a contiguous DMA memory for the circular buffer. The `buf_size` argument of `W_7200_Alloc_DBDMA_Mem()` is the half size of circular buffer in byte, that is, the size of each half buffer in byte. The DMA memory is referenced by the return parameter handle. This memory is system memory, users are not allowed to access it directly.

7

Limitation

The 12 MB/sec data transfer rate can only be possibly achieved in a system in which the PCI-7200 card is the only device using the bus, but the speed can not be guaranteed due to the limited FIFO depth.

PCI-7200 supports three input clock modes, internal clock, external clock, and handshaking modes. The first two modes cannot guarantee the input data integrity in high-speed data rate because of the limited FIFO depth and the PCI-bus latency variation. The handshaking mode is the only mode that data integrity can be guaranteed. In handshaking mode, you can expect 12 MB/sec data rate in average but the speed is not guaranteed.

The guaranteed data rate with internal clock or external clock mode is 1MB/sec in a machine that PCI-7200 card is the only device using the bus.

The largest transfer size (in bytes) PCI-7200 can perform is 64 MB.

Product Warranty/Service

Seller warrants that equipment furnished will be free from 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.