1 YEAR WARRANTY

OMEGA CARE℠ Extended Warranty Program

User's Guide

Shop online at

omega.com®

OMEGA®

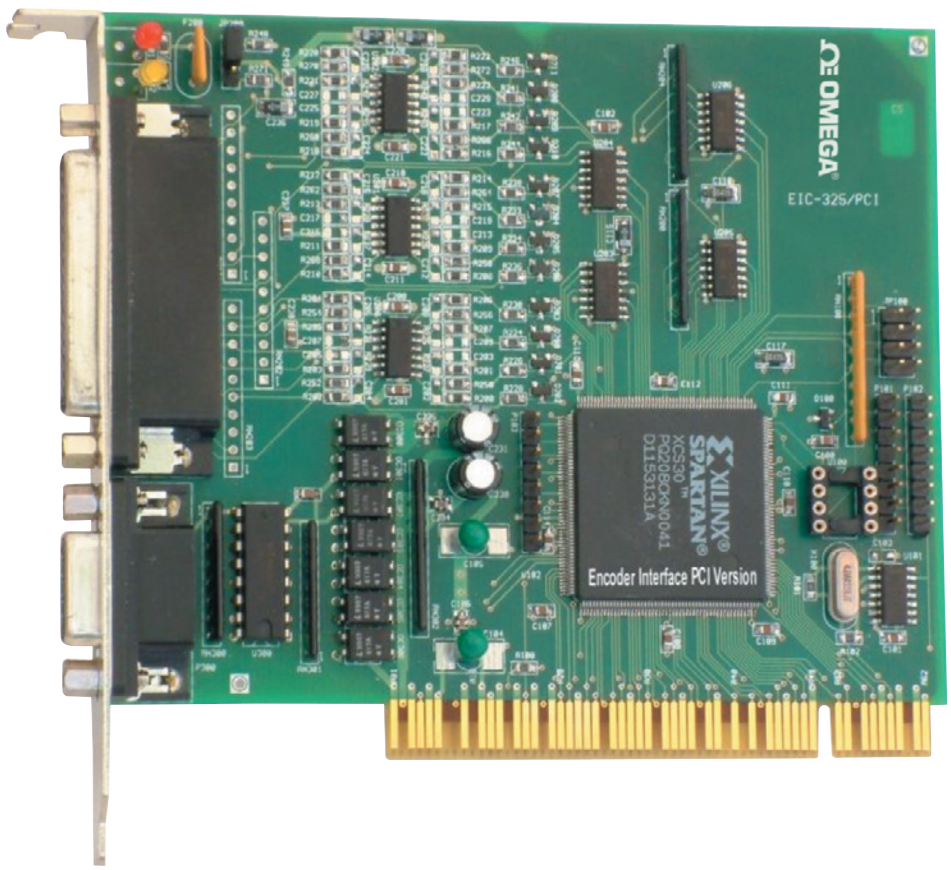
omega.com

e-mail: info@omega.com

For latest product manuals:

omegamanual.info

ISO 9001 CERTIFIED CORPORATE QUALITY	ISO 9002 CERTIFIED CORPORATE QUALITY
STAMFORD, CT	MANCHESTER, UK



EN-EIC-325-PCI Four Axis PCI Encoder Interface Card

omega.com®
ΩEOMEGA®

OMEGAnet® Online Service omega.com	Internet e-mail info@omega.com

Servicing North America:

U.S.A.: ISO 9001 Certified	One Omega Drive, P.O. Box 4047 Stamford, CT 06907-0047 TEL: (203) 359-1660 FAX: (203) 359-7700 e-mail: info@omega.com
Canada:	976 Bergar Laval (Quebec) H7L 5A1, Canada TEL: (514) 856-6928 FAX: (514) 856-6886 e-mail: info@omega.ca

For immediate technical or application assistance:

U.S.A. and Canada:	Sales Service: 1-800-826-6342/1-800-TC-OMEGA® Customer Service: 1-800-622-2378/1-800-622-BEST® Engineering Service: 1-800-872-9436/1-800-USA-WHEN®
Mexico:	En Español: (001) 203-359-7803 e-mail: espanol@omega.com FAX: (001) 203-359-7807 info@omega.com.mx

Servicing Europe:

Czech Republic:	Frystatska 184, 733 01 Karviná, Czech Republic TEL: +420 (0)59 6311899 FAX: +420 (0)59 6311114 Toll Free: 0800-1-66342 e-mail: info@omegashop.cz
Germany/Austria:	Daimlerstrasse 26, D-75392 Deckenpfronn, Germany TEL: +49 (0)7056 9398-0 FAX: +49 (0)7056 9398-29 Toll Free in Germany: 0800 639 7678 e-mail: info@omega.de
United Kingdom: ISO 9002 Certified	One Omega Drive, River Bend Technology Centre Northbank, Irlam, Manchester M44 5BD United Kingdom TEL: +44 (0)161 777 6611 FAX: +44 (0)161 777 6622 Toll Free in United Kingdom: 0800-488-488 e-mail: sales@omega.co.uk

FOUR AXIS ENCODER INTERFACE CARD

MODEL: EN-EIC-325-PCI

User's Guide
Version 2.01,
Oct. 2003

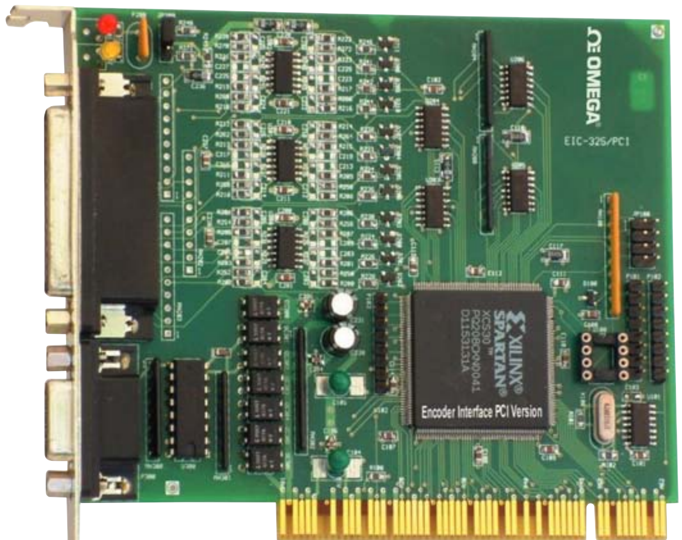


TABLE of CONTENTS

1. GENERAL DESCRIPTION

• The EN-EIC-325-PCI handles four axes of user's encoders. Each user's encoder is directly attached to the Encoder Interface on the card.
• The EN-EIC-325-PCI includes eleven logical inputs, and three general outputs.
• The EN-EIC-325-PCI is I/O mapped.

1.1. ENCODER INTERFACE

Note: Each of the registers IPC, XPC and SPR, mentioned below, represents an unsigned integer 24 bit number ranging from 0 to 16777215. In case the user needs also negative values, he should refer to 16777215 as –1 and so on, thus changing the range 0 – 16777215 to –8388608 – +8388607

Each Encoder Interface includes the following elements:

1.1.1. IPC

The IPC (Internal Position Counter) is updated continuously according to the input from user's encoders.
The updating of the IPC is affected by the Clock Resolution, that may be set to 1, 2 or 4 Clocks/Cycle.

1.1.2. SOFTWARE OUTPUTS

a. The XPC (eXternal Position Counter) is a latch counter being equalized to the IPC upon user's request – either via PC’s software, or by a hardware (real-time) input.
b. "Event Signal" – produced when a pre-defined condition is met.
This *software* Event Signal is supplied also in the *hardware* outputs as described in section 1.2.2 / ii. The user may select one of the following as the pre-defined condition that becomes the trigger of the event:
   - IPC = SPR
     Each Encoder Interface includes an SPR (Set Point Register) being adjustable by the user.
     In case the user selects this condition, an event occurs when IPC = SPR.
   - IPC Overflow
     In case the user selects this condition, an event occurs when there's overflow in the IPC, that is, IPC changes from 16777215 to 0, or vice versa.
   - Index
     In case the user selects this condition, an event occurs when an index (marker) signal arrives from user's encoder.

1.1.3. SOFTWARE INPUTS

a. *Data Request* to request an XPC update.
For example, upon *Data Request* on the A axis, the following operation is done:
$XPC_A \Leftarrow IPC_A$
Notes:
1. *Data Request* may be applied on a single axis, or (all at once) on: axes pair (A+B or C+D), or all four axes.
2. Besides this software *Data Request*, there is also a hardware (real-time) *Data Request* applied on all four axes, as described in section 1.2.1 / ii.

b. *Reset* to clear the IPC.
For example, upon *Reset* on the A axis, the following operation is done:
$IPC_A \Leftarrow 0$
Note: *Reset* may be applied on a single axis, or (all at once) on an axes pair (A+B or C+D).

1.2. CARD'S HARDWARE I/O

1.2.1. HARDWARE INPUTS

i. Inputs #1–#10: Ten general inputs (reflected by software outputs).
ii. Input #11: *Data Request* to request all XPC's update:
Upon this hardware *Data Request*, the following operations are done (all at once):
$XPC_A \Leftarrow IPC_A$; $XPC_B \Leftarrow IPC_B$; $XPC_C \Leftarrow IPC_C$; $XPC_D \Leftarrow IPC_D$.
In other words – this *Data Request* is a hardware real-time equivalent to the software function RequestPositionCounter (*CardHandle*, Encoder4, *OutputsMirror*) (this is function #26 in section 3.5.5.) .) The minimal pulse width should be 20 ns; the 'snapshot' is produced when the pulse goes **low**.

1.2.2. HARDWARE OUTPUTS

a. Three general outputs (reflecting software inputs).
b. Four outputs of the Event Signals, as described in section 1.1.2 / ii. The user may define these outputs to be held until he sends an explicit "Clear" request.

2. CARD'S HARDWARE

The EN-EIC-325-PCI uses the PCI bus of the PC.

2.1. DATA BUS and ADDRESS BUS

* The Data bus is 32 bits connected to the PCI.
* The Address bus is 32 bits (I/O access only, on lower 1MB mode).
* The Address access space is 16 bytes, located from *Base Address* to *Base Address + 15*. The Base Address is allocated by the system each PC power-up.
* Bus controls – see the PCI standard version 2.1.

2.2. PIN LAY-OUT

The drawings of the input/output connections are in Appendix B.

Encoder Inputs

Function		D-Type 44 pin No.
Encoder A	Vcc	31
	Index+	17
	GND	2
	Index–	32
	Sine– / Phase A–	18
	Sine+ / Phase A+	3
	Cosine+ / Phase B+	33
	Cosine– / Phase B–	4
Encoder B	GND	20
	Vcc	5
	Index+	35
	Sine+ / Phase A+	21
	Index–	6
	Sine– / Phase A–	36
	Cosine– / Phase B–	22
	Cosine+ / Phase B+	7
Encoder C	Vcc	23
	GND	38
	Index–	24
	Index+	9
	Sine+ / Phase A+	39
	Cosine+ / Phase B+	25
	Sine– / Phase A–	10
	Cosine– / Phase B–	40
Encoder D	Vcc	41
	Index+	27
	GND	12
	Index–	42
	Sine– / Phase A–	28
	Sine+ / Phase A+	13
	Cosine+ / Phase B+	43
	Cosine– / Phase B–	14
GND		30

Notes:
1. Total max. current 0.7A.
2. The inputs can be sine (sine and cosine) or square (phases A and B).
3. For single ended inputs use the '–' input only.
   The two non-connected wires per each encoder (Sine+ and Cosine+ should be floating, ***not*** grounded.
4. All the five GND pins (2, 20, 38, 12 and 30) are shortened. They appear five times to make the wiring easier.

Logical Inputs

Function	D-Type 44 pin No.
Input #2	16
Input #1	1
Input #3	19
Input #4	34
Input #5	37
Input #11	8
Input #7	26
Input #6	11
Input #8	29
Input #9	44
Input #10	15
GND	30

ENCODERS' EVENT SIGNALS AND GENERAL OUTPUTS

Function		D-Type 9 pin No.
VExt+		1
Event Signal	Encoder A	6
	Encoder B	2
	Encoder C	7
	Encoder D	3
General Output	Output #1	8
	Output #2	4
	Output #3	9
VExt–		5

Notes:
1. Max. (consumed) current 15 mA + outputs drained current.
2. +/– VExt = 5–24 V.

3. SOFTWARE INTERFACE WITH THE CARD

3.1. INSTALLATION

The *main* files and folders of the software package are as follows:

- **Win95_98**
  - Install.exe
  - WRTdevN.VxD (Ten files: N=0,...,9)
  - ImsEncPci.ocx
  - VB
  - VC
- **WinNT**
  - WinRT.sys
  - ImsEic325-PCINT4.reg
- **Win2K**
  - EIC325_P.sys
  - ImsEic325-PCIWDM.inf
- **WinXP**
  - EIC325_P.sys
  - ImsEic325-PCIWDM.inf
- **OCX for WinNT/2K/XP**
  - ImsEncPci.ocx
  - VB
  - VC
    - ReadMe.txt
    - EncPciConst.h
    - EncPciBasicTest
- **DLL for WinNT/2K/XP**
  - EIC325PCI.dll
  - EIC325PCI.lib
  - VB
  - VC
    - ReadMe.txt
    - EncPciConst.h
    - EncPciFunctions.h
    - EncPciBasicTest
- **CPP**
- **More**

Copy these folders (you may skip the irrelevant items) under some new folder on your hard disk, say ‘C:\ImsEnc325-PCI’.

The VB folders

The VB folders include an illustration program, which is essential for *any* user, not only the VB programmer. For any environment you are using – refer to the relevant VB folder.

Note:
If Visual Basic is *NOT* installed on your computer, then before running the (VB produced) program EncPciTest.exe (recommended), please verify that:

1. **Your *WinSysPath* folder includes the following two files:**
   *** Richtx32.ocx**
   *** Riched32.dll**
2. **The Richtx32.ocx file is registered. It should be, as it is self-registering. However, if a manual registering is still required, apply:**
   Start, Run, 'Regsvr32 *WinSysPath*\Richtx32.ocx'

***WinSysPath* is:**

```
Win 95 or 98: \Windows\System
Win NT or 2K: \Winnt\System32
Win XP:       \Windows\System32
```

If needed, you'll find the above two files in the software package.

3.1.1. Windows

The various Windows folders supply the files required for the **driver** installation. The Win95/98 driver includes an ActiveX control. The WinNT/2K/XP drivers include two options: an ActiveX control and a DLL. The DLL uses the popular ‘stdcall’ calling convention recognizable by VB, VC, Delphi etc. The ActiveX control and the DLL are implemented by a WinRT-based solution. They may be installed even if your system already includes WinRT-based solutions, provided that the total number of solutions won't exceed 10 (Win95/98) or 32 (WinNT/2K/XP).

?After performing the driver installation relevant to your Windows (95/98/NT/2K/XP) you may proceed to the card installation (see appendix C).

Win95/98

- Refer to the Win95_98 folder.
- Run the "Install.exe" program, which automatically copies the required files and updates the registry:
  - The file ImsEncPci.ocx is copied to \Windows\System.
  - If the folder \Windows\System\Vmm32 doesn't exist, it's created.
  - The required WinRT file (usually WRTdev0.VxD) is copied to \Windows\System\Vmm32.
  - The registry information is updated.
- Proceed to the **card** installation (see appendix C).

WinNT

- Refer to the WinNT folder.
- Copy the WinRT.sys file into the \Winnt\System32\Drivers folder.
- Execute the ImsEic325-PCINT4.reg file by a double-click.
- Perform the ‘ActiveX Control’ and/or the ‘DLL’ steps under ‘WinNT/2K/XP’.
- Proceed to the **card** installation (see appendix C).

Win2K

- Refer to the Win2K folder.
- Perform the ‘ActiveX Control’ and/or the ‘DLL’ steps under ‘WinNT/2K/XP’.
- Proceed to the **card** installation (see appendix C).
- (Note: The *.sys* and *.inf* files will be used in the ‘card installation’ stage.)

WinXP

- Refer to the WinXP folder.
- Perform the ‘ActiveX Control’ and/or the ‘DLL’ steps under ‘WinNT/2K/XP’.
- Proceed to the **card** installation (see appendix C).
- (Note: The *.sys* and *.inf* files will be used in the ‘card installation’ stage.)

WinNT/2K/XP

- ActiveX Control
  Copy the .ocx file from your ‘OCX for WinNT_2K_XP’ folder to your *WinSysPath* folder and register it by Start, Run,
  'regsvr32 *WinSysPath*\ImsEncPci.ocx'.

- DLL
  Copy the .dll file from your ‘DLL for WinNT_2K_XP’ folder to your *WinSysPath* folder.

WinSysPath **is:**

```
Win NT or 2K: \Winnt\System32
Win XP:       \Windows\System32
```

3.1.2. VB

Select the ‘VB’ folder that corresponds to your driver installation (section 3.1.1). You'll find here a full illustration in VB 5.0 (named "EncPciTest") that demonstrates how to use the various functions to communicate with the card. There are two versions: one using the ActiveX control, the other using the DLL. The DLL-based version includes an Eic325PciDll.bas file that contains the Declare’s of all the DLL functions. There are no such Declare’s in the ActiveX control-based version, as the definitions are included in the control itself. At application start the Active property is set to 1, and in the end to 0. The EncPciTest program illustrates all the functions available. It's recommended for **any** user, not only the VB programmer, to run it. Most of the program is straightforward. Here are some highlights to explain the non-trivial aspects:

- To check primary communication with the card, use the "LEDs" frame. The two LEDs on the card should follow your selection.
- By definition, the program intends to illustrate the basic functions in order to instruct the *programmer* code his application. However, to get some better feeling, there's the "Loop" mode that "Clicks" cyclically:
  - In the Position Counter frame:
    - ALL in the Req. (=Request) column
    - A, B, C, D in the Read column
  - In the I/O frame:
    - Read in the Chip #1 square
    - Read in the Chip #2 square
- The Active Chip frame:
  The required chip is selected automatically upon running any operation. The significance of explicit chip selection by this frame is first to illustrate this operation to the programmer and second to determine on which chip the "Test Active Chip", if clicked, would operate.
- The I/O frame is a bit confusing:
  - The Hardware General Inputs are reflected by the card in its software OUTPUTS.
  - The Voltage Failure card's software output allows the user to check if there is a short-circuit between the output voltage source and the encoders (the thermal resettable fuse in the card will resume normal operation after the short is ended). Voltage Failure = 1 indicates a short, and the user should be instructed in this case to detach the 44 pin connector, then re-check.
  - The EventFlags appear in two separate outputs:
    - Software Outputs
    - Hardware Outputs
  - Clicking "Read" reads card's software outputs of the –
    - Hardware General Inputs
    - Voltage Failure
    - Software EventFlags

- The display of the Hardware EventFlags Output reflects what SHOULD be in these outputs analyzing the Software EventFlags Outputs as well as user's operations that may affect the Hardware EventFlags Output, as described in section 3.5.4, functions #22 and #24. Actually, the EncPciTest program *simulates* card's response and concludes what Hardware EventFlags Output the card should supply.
- The card reflects in its Hardware General Outputs the status of its software INPUTS.
- Clicking "Write" writes into card's software inputs the desired status of card's Hardware General Outputs.
- The desired status of card's Hardware General Outputs may be set by clicking a specific output (1, 2 or 3). However, the output is NOT passed immediately to the card. Click "Write" to validate the new status, or "Undo" to leave out the last change(s).

3.1.3. VC

Select the ‘VC’ folder that corresponds to your driver installation (section 3.1.1). This folder includes four elements:

- The file ReadMe.txt
- The file EncPciConst.h
- The file EncPciFunctions.h
- The subfolder EncPciBasicTest

The first item (ReadMe.txt) includes detailed instructions how to use the ActiveX control or the DLL in Visual C. In order to follow these instructions, you'll need the second and third items (EncPciConst.h and EncPciFunctions.h). The fourth (last) item (the subfolder EncPciBasicTest) includes a sample basic project that was created according to the instructions of ReadMe.txt.

3.1.4. CPP

Here you'll find the required files for the DOS C/C++ programmer.
The programmer should include EncPci.h in his source file, and EncPci.obj in his project.
A simple demo program is supplied. It reads continuously the four axes and displays their values.

3.1.5. More

This folder supplies more information for environments other than the above. The files EncPci.cpp and EncPci.h include the source code (in C++) of driver's functions in DOS. You may utilize this code in order to produce your own driver for your environment.

3.2. INTRODUCTION TO THE FUNCTIONS

Each encoders pair is handled by a "chip":
Chip #1 includes Encoder Interfaces A and B.
Chip #2 includes Encoder Interfaces C and D.
The following functions serve the DOS C/C++ programmer as well as the Windows programmer.

- The DOS C/C++ programmer should include EncPci.h in his source file, and EncPci.obj in his project. This will make all functions available.
- The Windows programmer should use an ActiveX control (Win95/98, WinNT/2K/XP) or a DLL (WinNT/2K/XP). The ActiveX control and the DLL include all these functions.

  At application start, the Active property should be set to 1, and in the end to 0, as follows:

  In VB:
  * Using the ActiveX control, it will look like:
  Enc.Active = 1
  Enc.Active = 0
  * Using the DLL, it will look like:
  SetActive (1)
  SetActive (0)

  In VC:
  Using either the ActiveX control or the DLL, it will look like:
  SetActive (1) ;
  SetActive (0) ;

Notes:

- Active Chip
  The functions in sections 3.4 and 3.5 operate on the active chip as selected by SetActiveChip (section 3.4.1, function #5).
- Set... functions
  Each Set... function includes the following steps:
  - Write required new value onto the card.
  - Read card's current value.
  - Compare card's read value with the required new value.
  - Respond with a "success" return code only if the values are equal.

  Therefore, a "success" return code indicates not only correct arguments, but also verified communication with the card.
  However, each Set... function has a corresponding Get... function to enable the user read the actual value within the card.

- **Return Code**
  All functions respond with a return code. 1 indicates success, 0 means failure due to either wrong arguments or unsuccessful operation.
- **Arguments**
  The common arguments, that is, those that are not individual to specific functions, are described in section 3.6.

3.3. <u>CARD LEVEL FUNCTIONS</u>

#	Brief Description	Name	Arguments (in C Syntax)	Full Description
1	Supply a 'Card Handle' to the n-th EN-EIC-325-PCI card (if exists). (First n is 0.)	**GetCardHandle**	(long* *CardHandle*, short *CardIndex*)	Call this function in loop in your initialization. Start with *CardIndex=0* and increment it each iteration. Stop the loop upon receiving a 'failure' (0) return code from the function. For each iteration (denoting *CardIndex* by 'n'): If the n-th EN-EIC-325-PCI card is detected: * *CardHandle* is set to a handle to current card, to be used by you as card's identifier in all other functions. Save this *CardHandle*. In case you have more than one EN-EIC-325-PCI card, use an array for the 'save' operation. * *Return Code* is set to 1 ('success'). Otherwise (no 'fresh' card detected): * *CardHandle* is irrelevant. * *Return Code* is set to 0 ('failure').
2	Supply the Revision ID of current card.	**GetRevisionID**	(long *CardHandle*, short* *RevisionID*)	Sets *RevisionID* to the Revision ID of current EN-EIC-325-PCI card.
3	Set a LED	**SetLed**	(long *CardHandle*, short *LedNumber*, short *OnOffMode*)	Turns the specified LED on or off. Useful to check primary communication with the card. Except when turning both LEDs on, has no functional significance. Turning both LEDs on has same effect as RequestPositionCounter (CardHandle, Encoder4, 0), i.e., simultaneous request for all four axes.
4	Get a LED	**GetLed**	(long *CardHandle*, short *LedNumber*, short* *OnOffMode*)	Gets the current status (on or off) of the specified LED.

Note: The description of the common arguments is in section 3.6

3.4. <u>CHIP LEVEL FUNCTIONS</u>

3.4.1. <u>ACTIVE CHIP</u>

#	Brief Description	Name	Arguments (in C Syntax)	Full Description
5	Set Active Chip	**SetActiveChip**	(long *CardHandle*, short *ChipNumber*)	Makes the specified chip active. **All the following functions refer to that active chip.**
6	Get Active Chip	**GetActiveChip**	(long *CardHandle*, short* *ChipNumber*)	Gets the current active chip (Chip1 or Chip2).
7	Test the active chip	**TestActiveChip**	(long *CardHandle*)	Tests the active chip.

Note: The description of the common arguments is in section 3.6

3.4.2. I/O

#	Brief Description	Name	Arguments (in C Syntax)	Full Description
8	Read the inputs of a chip, the EventFlags of its two encoders, and the Voltage Failure.	**ReadInputs**	(long *CardHandle*, short* *Inputs*, short**EncoderEventFlags*)	Reads 5 or 6 chip's logical inputs, the two flags of its EncoderEvents, and, in case of Chip #1, the Voltage Failure. *Inputs*: For Chip #1: Bit #n corresponds to logical input #n+1 (n = 0,...,4). Bit #5 corresponds to Voltage Failure (1 means that Voltage Failure = true). For Chip #2: Bit #n corresponds to logical input #n+6. (n = 0,...,5). *EncoderEventFlags*: Bit #0 corresponds to Encoder1 (A or C). Bit #1 corresponds to Encoder2 (B or D). Notes: 1. Chip #1 has 5 logical inputs (marked 1–5). Chip #2 has 6 logical inputs (marked 6–11). 2. When an encoder event occurs, its software signal is held until **ReadInputs** is called. Afterwards, the software signal is cleared, unless the signal still lasts. This mechanism ensures that **ReadInputs** will hold the last encoder event (in *EncoderEventFlags*) even though the event is already over.
9	Write to chip's outputs.	**WriteOutputs**	(long *CardHandle*, short *Outputs*)	Meaningful only when the active chip is Chip1, in which case its three general outputs (marked 1,2,3) are written. *Outputs*: Bit #n corresponds to general output #n+1 (n=0,1,2). Notes: 1. Due to hardware limitations, the current status of *Outputs* should be kept in order to be passed as argument in the functions **ResetPositionCounter** and **RequestPositionCounter** (named there *OutputsMirror*, refer to section 3.5.5, functions #25–26). 2. Chip2 has no general outputs.

Note: The description of the common arguments is in section 3.6

3.5. **ENCODER LEVEL FUNCTIONS**

3.5.1. RESOLUTION

#	Brief Description	Name	Arguments (in C Syntax)	Full Description
10	Set Encoder Resolution	**SetEncoderResolution**	(long *CardHandle*, short *EncoderNumber*, short *ClocksPerCycle*)	Selects the appropriate resolution that fits user's encoder.
11	Get Encoder Resolution	**GetEncoderResolution**	(long *CardHandle*, short *EncoderNumber*, short* *ClocksPerCycle*)	Gets the current selection of the Encoder Resolution (1, 2 or 4 Clocks/Cycle).

Note: The description of the common arguments is in section 3.6

3.5.2. INDEX

#	Brief Description	Name	Arguments (in C Syntax)	Full Description
12	Set Index Pulse Polarity	**SetIndexPulsePolarity**	(long *CardHandle*, short *EncoderNumber*, short *Polarity*)	Selects the polarity of the index (marker) pulse coming from user's encoder.
13	Get Index Pulse Polarity	**GetIndexPulsePolarity**	(long *CardHandle*, short *EncoderNumber*, short* *Polarity*)	Gets the current status of the Index Pulse Polarity (active on high/low).
14	Set "reset upon index"	**SetIndexResetsPosition Counter**	(long *CardHandle*, short *EncoderNumber*, short *EnabledDisabledMode*)	Defines whether a reset (=clear) of the Internal Position Counter (IPC) should take place upon index (marker) pulse coming from user's encoder. Note: The eXternal Position Counter (XPC) is NOT affected immediately. Run **RequestPositionCounter** to apply the effect on the XPC, and **ReadPositionCounter** to read the XPC (refer to section 3.5.5, functions #26-27).
15	Get "reset upon index"	**GetIndexResetsPosition Counter**	(long *CardHandle*, short *EncoderNumber*, short**EnabledDisabledMode*)	Gets the current status of "reset upon index" (enabled or disabled).

Note: The description of the common arguments is in section 3.6

3.5.3. SETPOINT

#	Brief Description	Name	Arguments (in C Syntax)	Full Description
16	Set Set Point	**SetSetPoint**	(long *CardHandle*, short *EncoderNumber*, long *SetPoint*)	Defines the Set Point Register (SPR). Reaching the SPR (i.e., meeting the condition IPC = SPR) may become the trigger of the Event Signal. *SetPoint*: The desirable SPR. Should conform to an unsigned 24-bit number (ranging from 0 to 16777215). If *SetPoint* exceeds this range, a "failure" return code is responded, and SPR remains changeless. Note: Refer also to section 3.5.4.
17	Get Set Point	**GetSetPoint**	(long *CardHandle*, short *EncoderNumber*, long* *SetPoint*)	Gets the current value of the Set Point Register (SPR).

Note: The description of the common arguments is in section 3.6

3.5.4. EVENT SIGNAL

#	Brief Description	Name	Arguments (in C Syntax)	Full Description
18	Set Source of Encoder's EventFlag	**SetEncoderEventFlag Source**	(long *CardHandle*, short *EncoderNumber*, short *EventFlagSource*)	Selects a condition that becomes the trigger of the Event Signal. Upon meeting this condition, an Event Signal is supplied, both in hardware and in software. Note: The characteristics of the signal are different in software and in hardware: * The polarity and the "hold mode" of the hardware signal are programmable (see functions #20–23). * The polarity of the software signal is constant. * The software signal is always held until activating the function **ReadInputs** (refer to section 3.4.2, function #8), which reads the current signal status and then automatically clears it.
19	Get Source of Encoder's EventFlag	**GetEncoderEventFlag Source**	(long *CardHandle*, short *EncoderNumber*, short* *EventFlagSource*)	Gets the current selection of the Source of Encoder's EventFlag.
20	Set Polarity of Hardware Encoder's EventFlag	**SetEncoderEventFlag Polarity**	(long *CardHandle*, short *EncoderNumber*, short *Polarity*).	Sets the polarity of the hardware Event Signal.
21	Get Polarity of Hardware Encoder's EventFlag	**GetEncoderEventFlag Polarity**	(long *CardHandle*, short *EncoderNumber*, short* *Polarity*)	Gets the current status of the Encoder's EventFlag Polarity (active on high/low).

Note: The description of the common arguments is in section 3.6

22	Set Hold Mode of Hardware Encoder's EventFlag	**SetHoldEncoderEvent Flag**	(long *CardHandle*, short *EncoderNumber*, short *OnOffMode*)	Determines the "hold" characteristic of the hardware Event Signal: *OnOffMode*: Off: Hardware signal remains "as is" – no holding mechanism. On: Hardware signal is held until running an explicit "clear" operation by **ClearEncoderEventFlag** (function #24). Note: All functions of sections 3.5.1, 3.5.2 and 3.5.4 also have (as a by-product) the same effect as **ClearEncoderEventFlag**.
23	Get Hold Mode of Hardware Encoder's EventFlag	**GetHoldEncoderEvent Flag**	(long *CardHandle*, short *EncoderNumber*, short* *OnOffMode*)	Gets the current selection of the Hold Mode of Encoder's EventFlag (on or off).
24	Clear Hardware Encoder's EventFlag	**ClearEncoderEvent Flag**	(long *CardHandle*, short *EncoderNumber*)	Clears the hardware Event Signal. Meaningful only when the Hold Mode of Hardware Encoder's EventFlag is on (refer to function #22). Note: All functions of sections 3.5.1, 3.5.2 and 3.5.4 also have (as a by-product) the same effect as **ClearEncoderEventFlag**.

Note: The description of the common arguments is in section 3.6

3.5.5. POSITION COUNTER

#	Brief Description	Name	Arguments (in C Syntax)	Full Description
25	Reset (=clear) Position Counter	**ResetPositionCounter**	(long *CardHandle*, short *EncoderNumber*, short *OutputsMirror*)	Clears the Internal Position Counter (IPC) of one or two axes. *OutputsMirror*: This argument is described within the function **WriteOutputs** (section 3.4.2, function #9). Note: The eXternal Position Counter (XPC) is NOT affected directly. Run **RequestPositionCounter** to apply the effect on the XPC, and **ReadPositionCounter** to read the XPC.
26	Request an updated reading of Position Counter	**RequestPositionCounter**	(long *CardHandle*, short *EncoderNumber*, short *OutputsMirror*)	Updates the eXternal Position Counter (XPC) according to the Internal Position Counter (IPC): XPC $\Leftarrow$ IPC. *OutputsMirror*: This argument is described within the function **WriteOutputs** (section 3.4.2, function #9). Note: Run **ReadPositionCounter** (function #27) to read the XPC.
27	Read Position Counter	**ReadPositionCounter**	(long *CardHandle*, short *EncoderNumber*, long* *PositionCounter*)	Reads the eXternal Position Counter (XPC): PositionCounter $\Leftarrow$ XPC. Note: To update the XPC, call **RequestPositionCounter** (function #26).

Note: The description of the common arguments is in section 3.6

3.6. ARGUMENTS

The constants of the arguments are available as follows:
* In VB they are included in the example source file.
* In VC they are in the EncPciConst.h file in the VC folder; refer to the ReadMe.txt file.
* For DOS C/C++, the arguments, along with the headers of the functions, are in the file CPP\EncPci.h.
* For other environments, refer to the file More\EncPci.h. It contains the arguments, along with the headers of the functions, coded in C syntax. Transform the code to the proper syntax.

// *CardHandle:*
// Handle to current card as retrieved by the function GetCardHandle.

// *ChipNumber*:
#define Chip1 1 // Standard chip (always should be present).
#define Chip2 2 // Optional chip.

// *EncoderNumber*:
// The codes of *EncoderNumber* are of three types:
// Type 1 codes (one encoder):
//   Available for all the functions using *EncoderNumber*.
// Type 2 codes (pair of encoders):
//   Available only for **ResetPositionCounter** and **RequestPositionCounter**.
// Type 3 code (all four encoders):
//   Available only for **RequestPositionCounter**.

// Type 1 *EncoderNumber* codes (one encoder):
#define Encoder1 1 // First encoder in the selected chip (A or C).
#define Encoder2 2 // Second encoder in the selected chip (B or D).
// Type 2 *EncoderNumber* codes (pair of encoders):
#define Encoder3 3 // Both encoders in the selected chip (A+B or C+D).
// Type 3 *EncoderNumber* codes (all 4 encoders):
#define Encoder4 4 // All 4 encoders (A+B+C+D).

// *LedNumber*:
#define LedYellow 1
#define LedRed 2

// *ClocksPerCycle*:
#define Clock1 1 // One clock per cycle.
#define Clock2 2 // Two clocks per cycle.
#define Clock4 4 // Four clocks per cycle.

// *EventFlagSource*:
#define EventFlagSourceNone 1
#define EventFlagSourceSetPoint 2
#define EventFlagSourceOverflow 3
#define EventFlagSourceIndex 4

// *OnOffMode*:

```
#define TurnOn 1
#define TurnOff 0

// EnabledDisabledMode:
#define SetEnabled 1
#define SetDisabled 0

// Polarity:
#define ActiveOnHigh 1
#define ActiveOnLow 0
```

APPENDIX A: SPECIFICATION

Introduction

- ❖ A half-size PC card.
- ❖ Supports four optical encoders with either Square-wave or Sine output, both linear and rotary types.
- ❖ Directly connected to the encoders and also provides the excitation.
- ❖ Single / Multiple encoder pulses: Interpolation = 1, 2 or 4 (software selectable).
- ❖ Independent operation mode for each axis.
- ❖ General inputs, an index input per each encoder, and an input to request a snapshot of all counters in real-time.
- ❖ The user may instruct the card to generate an "Event Signal" output upon reaching a pre-defined set-point, counter overflow, or index (marker) signal.
- ❖ General outputs, an "Event Signal" output per each encoder, and a "Voltage Failure" indicator.
- ❖ All outputs are opto-isolated.
- ❖ Max. input pulse rate: 500 kpps.
- ❖ Software included:
  - ❖ Drivers for Win95/98/NT/2K/XP and how to communicate under DOS.
  - ❖ Useful utility that allows immediate read/test of the encoders.
  - ❖ Detailed example in VB and basic example in Visual C.

Encoder Counters

- ❖ 24 bit up/down latch counter for each axis. Counter's range is from 0 to 16777215. In case the user needs also negative values, he may consider the range 0 – 16777215 as -8388608 – +8388607.
- ❖ Each counter has a corresponding programmable set-point value.
- ❖ The user may instruct the card to generate an "Event Signal" output when card's counter reaches its corresponding set-point value, or at counter overflow.
- ❖ User may request a snapshot of a counter without stopping the counting process, thus not losing data. The operation may be accomplished either by software or, in real-time, by a hardware input.
- ❖ A counter may be reset by encoder's index output and/or by request via user's software.

<u>I/O</u>

- Index input per each axis (indicating Marker/Home/Zero). The user may instruct the card to reset its counter and/or generate an "Event Signal" output upon recognizing the index input.
- 11 Logical Inputs. The inputs are TTL/CMOS compatible Schmidt trigger single ended. Ten of them are general inputs and one is being used for requesting a real-time counters snapshot of all four channels.
- "Event Signal" opto-isolated output per each axis that may be used for Set Point, Carry (Overflow) or Index (Marker) indication + three general opto-isolated outputs.
- "Voltage Failure" indicator allows the user to know if there is a short-circuit between the output voltage source and the encoders (the thermal resettable fuse in the card will resume normal operation after the short is ended).

<u>Input Signals from the Encoders</u>

- **<u>Square wave signal:</u>**
  - Square wave Phase A and Phase B (Sine and Cosine) shifted by 90°.
  - Max. input pulse rate: 500 kpps.
  - Signals may be either single ended or differential.

- **<u>Sine wave signal:</u>**
  - Two incremental sinusoidal signals Phase A and Phase B (Sine and Cosine) shifted by 90°.
  - Max. input pulse rate: 500 kpps.

- **<u>General:</u>**
  - Index (marker) signals may be either single ended or differential.
  - HTL 0.5–5V or TTL compatible.
  - Characteristics of the electrical signals (encoder output impedance < 1 KΩ):
    - Current output encoders: > ±100µA.
    - Voltage output encoders:
      Differential: >±100mV for phases A, B (Sine and Cosine) and for the reference index (marker) pulse.
      Single ended: >2V amplitude (not recommended for Sine wave signal).
    - Excitation to the encoders: 5V DC.
    - Light source current: Max. 900 mA total. Protected by a polyswitch resettable fuse.

Software

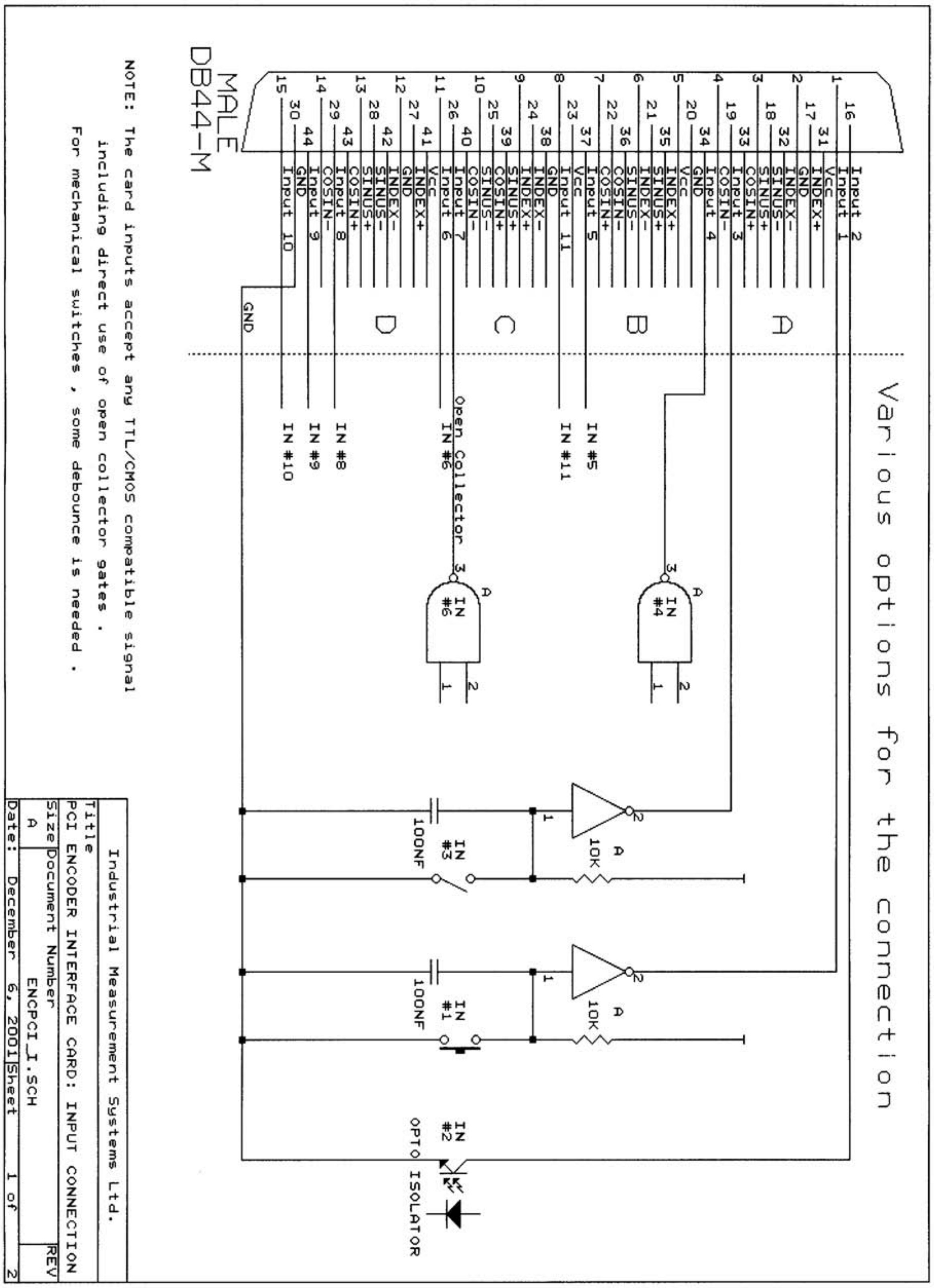
- Method of communication with PCI bus: I/O ports.
- Software included:
  - An ActiveX control to communicate with the card in Win95/98/NT/2K/XP. For WinNT/2K/XP there's also a DLL option instead of the ActiveX control.
  - A basic example in Visual C and a detailed example in VB.
  - Example and source code on how to communicate with the board under DOS.

General

- Board dimensions: Half-size PC card.
- PCI bus for PC.
- Power (all supplied from the PC bus):
  - +5V: 200 mA max.
  - +12V: 100 mA max.
  - –12V: 50 mA max.
- Connectors:
  - Encoders + Logical Inputs: Mini DIP 44 pin.
  - Outputs: DIP 9 pin.
- Environmental:
  - Operating temperature: 0–50° C (32–122° F).
  - Humidity: Up to 80% non-condensing.

APPENDIX B: DRAWINGS

Connector Wiring for the Inputs

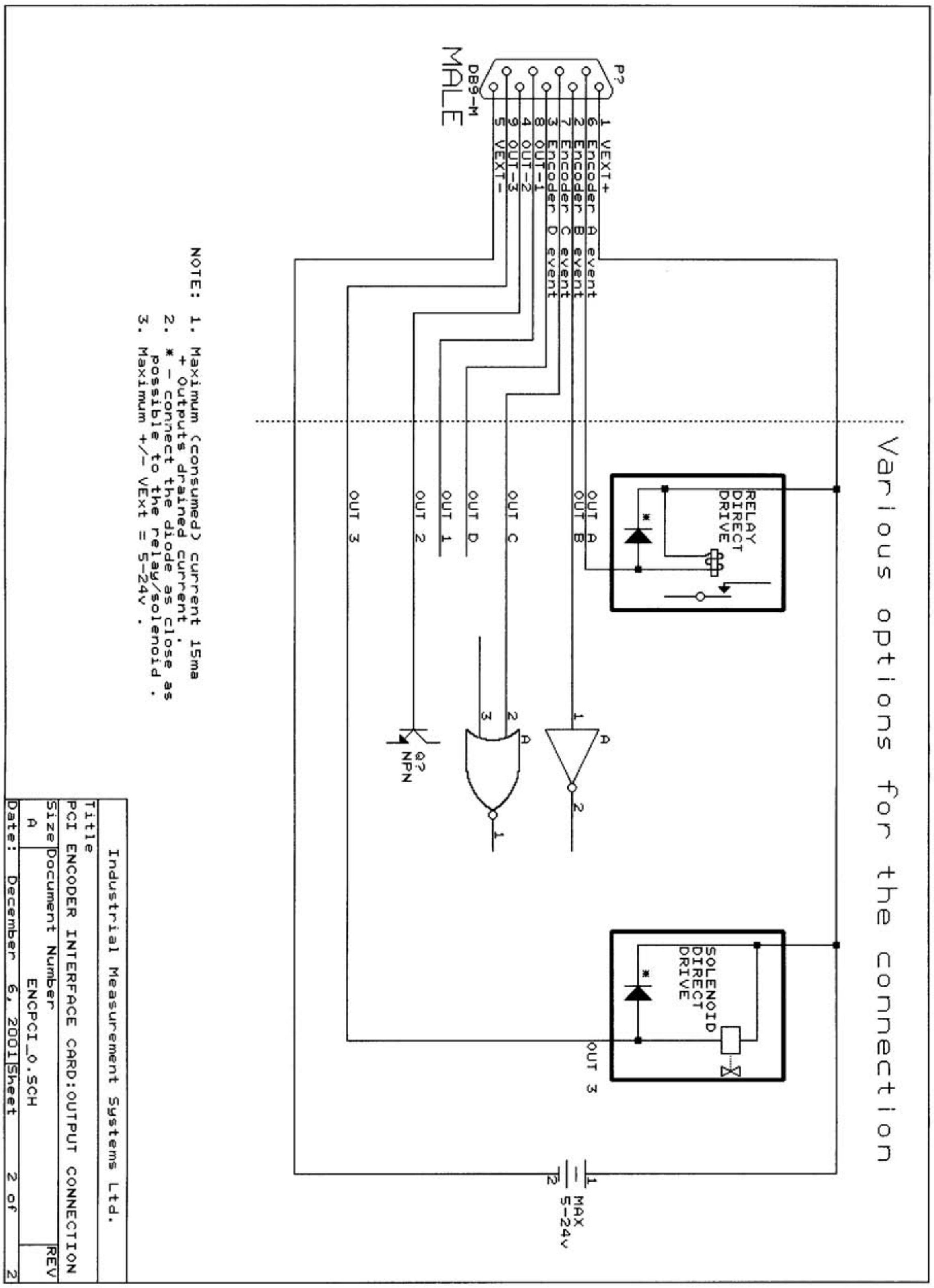


NOTE: The card inputs accept any TTL/CMOS compatible signal including direct use of open collector gates .
For mechanical switches , some debounce is needed .

Industrial Measurement Systems Ltd.

Title		
PCI ENCODER INTERFACE CARD: INPUT CONNECTION		
Size	Document Number	REV
A	ENCPCI_I.SCH	
Date: December 6, 2001	Sheet 1 of 2	

Connector Wiring for the Outputs



NOTE: 1. Maximum (consumed) current 15ma
+ Outputs drained current .
2. * - connect the diode as close as
possible to the relay/solenoid .
3. Maximum +/- VExt = 5-24v .

APPENDIX C: CARD INSTALLATION

?** **Install your card only <u>after</u> driver installation (section 3.1.1).

Win95/98

1. Shut down your PC (i.e., power off).

2. Insert the new card into a free PCI slot.

3. Turn on your PC.

4. During the Windows 95/98 startup, the following window will appear:

Add New Hardware Wizard

This wizard searches for new drivers for:
PCI Card
A device driver is a software program that makes a hardware device work.

Click:
<u>Next</u>

5. Select the following option:
Display a list of all the drivers in a specific location, so you can select the driver you want.
...and click:
<u>Next</u>

6. In the Devices List select:
Other Devices
and click:
<u>Next</u>

7. Wizard shows a '?' and displays in the 'Models' square:
Unsupported Device
Click:
<u>Next</u>

8. Wizard shows an 'Update Driver Warning':
The driver that you have chosen was not written specifically for the selected hardware and may not work correctly. Installing this driver is not recommended. Are you sure you want to use this driver?

This is normal. Click:
<u>Yes</u>

9. Click:
Next

10. Wizard displays:
Unsupported Device. Windows has not installed a driver for this device.

This is normal either. Click:
Finish

WinNT

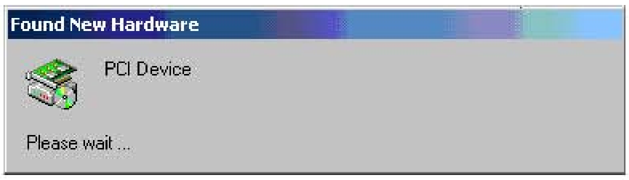
1. Shut down your PC (i.e., power off).
2. Insert the new card into a free PCI slot.
3. Turn on your PC.

? ***For Win2K/XP instructions please refer to the following pages.***

Win2000

* Shut down your PC (i.e., power off).
* Insert the new card to a free PCI slot.
* Turn on your PC.

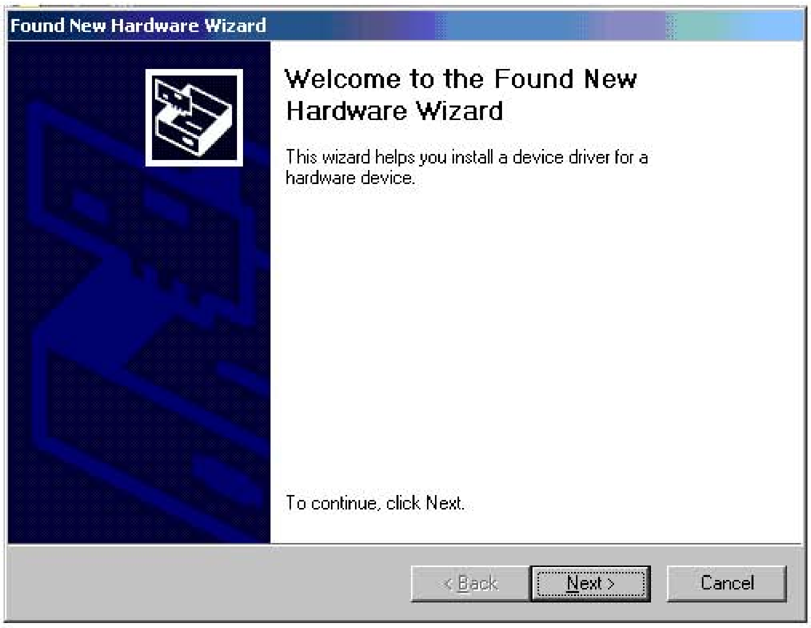
Win2000 will display the following message:


Found New Hardware

PCI Device

Please wait ...

... and the 'Found New Hardware Wizard' will appear:

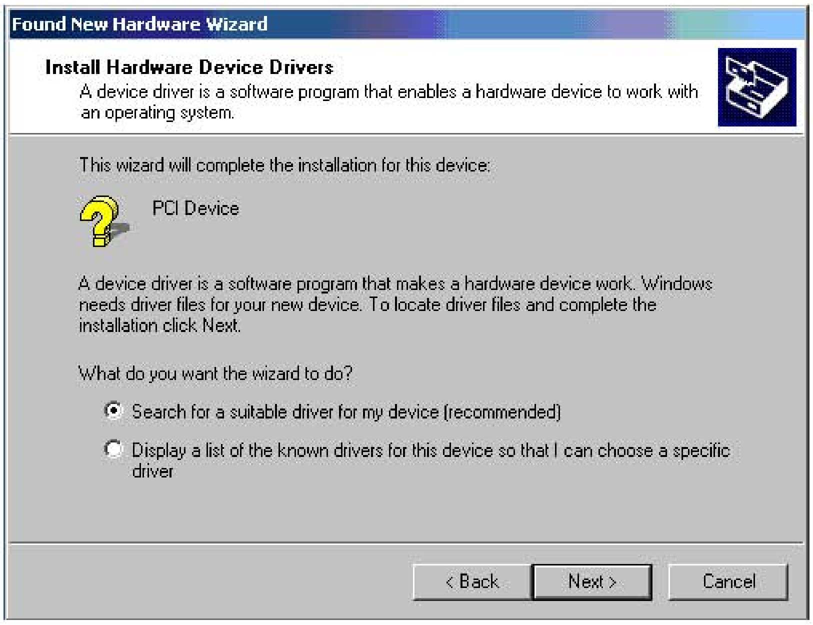

Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install a device driver for a hardware device.

To continue, click Next.

Click 'Next' and you'll get the following:


Found New Hardware Wizard

Install Hardware Device Drivers

A device driver is a software program that enables a hardware device to work with an operating system.

This wizard will complete the installation for this device:

PCI Device

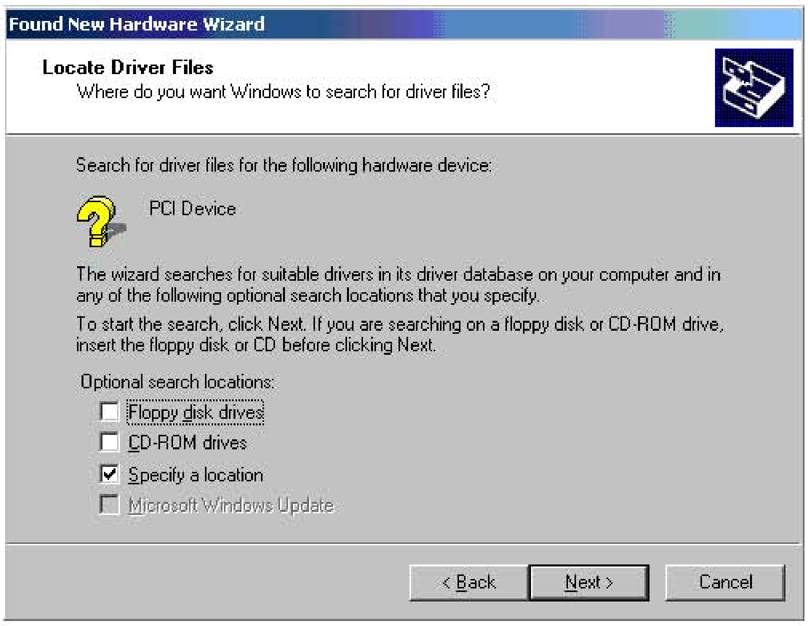
A device driver is a software program that makes a hardware device work. Windows needs driver files for your new device. To locate driver files and complete the installation click Next.

What do you want the wizard to do?

[✓ Search for a suitable driver for my device (recommended)]

Display a list of the known drivers for this device so that I can choose a specific driver

Click 'Next' and get the following:


Found New Hardware Wizard

Locate Driver Files

Where do you want Windows to search for driver files?

Search for driver files for the following hardware device:

PCI Device

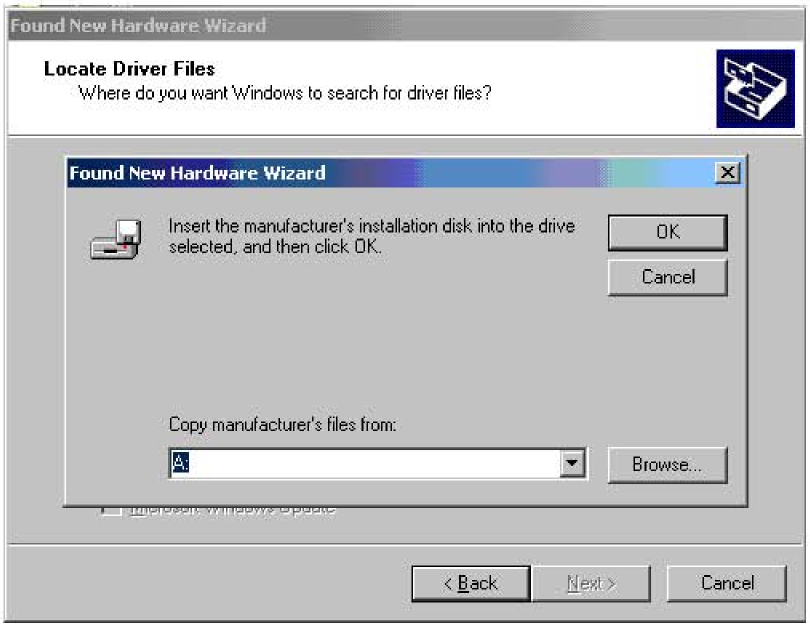
The wizard searches for suitable drivers in its driver database on your computer and in any of the following optional search locations that you specify.

To start the search, click Next. If you are searching on a floppy disk or CD-ROM drive, insert the floppy disk or CD before clicking Next.

Optional search locations:

- [ ] Floppy disk drives
- [ ] CD-ROM drives
- [x] Specify a location
- [ ] Microsoft Windows Update

Click 'Next' and get the following:


Found New Hardware Wizard

Locate Driver Files

Where do you want Windows to search for driver files?

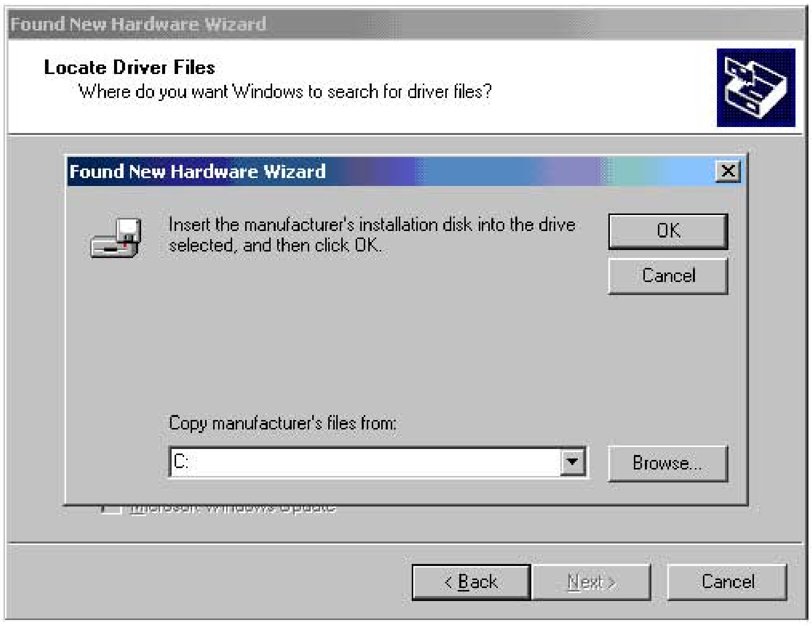
Found New Hardware Wizard

Insert the manufacturer's installation disk into the drive selected, and then click OK.

Copy manufacturer's files from:

A:

Change the 'A:' by typing the letter of your hard disk, say 'C:':


Found New Hardware Wizard

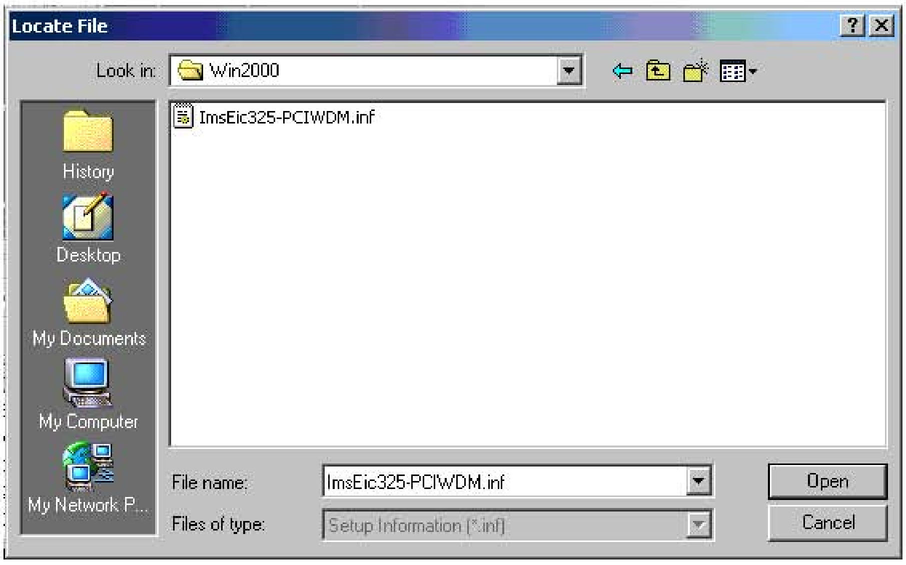
Locate Driver Files
Where do you want Windows to search for driver files?

Found New Hardware Wizard

Insert the manufacturer's installation disk into the drive selected, and then click OK.

Copy manufacturer's files from:
C:

Click 'Browse...' and browse to the folder where you copied the files during driver installation (section 3.1.1), for example:

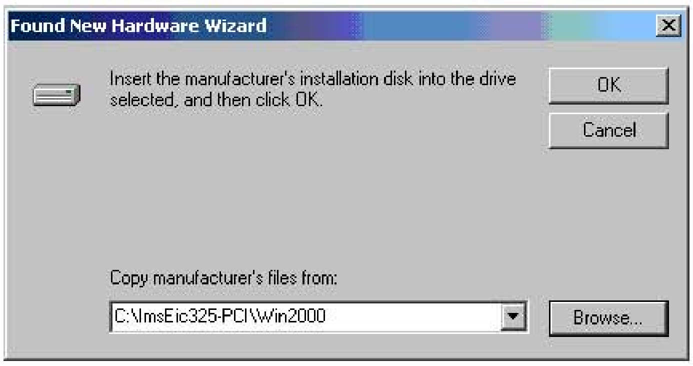

Locate File

Look in: Win2000

ImsEic325-PCIWDM.inf

File name: ImsEic325-PCIWDM.inf
Files of type: Setup Information (*.inf)

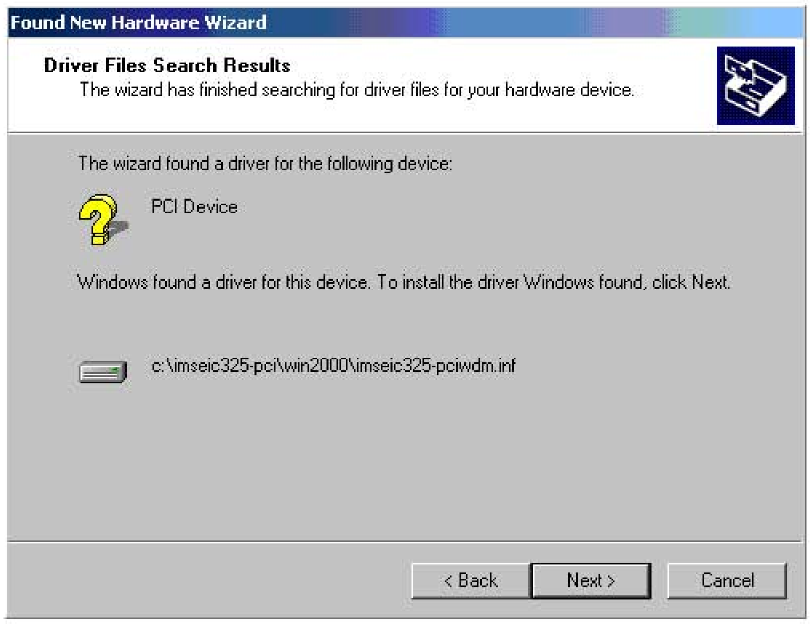
Click 'Open' and you'll get the following:


Found New Hardware Wizard

Insert the manufacturer's installation disk into the drive selected, and then click OK.

Copy manufacturer's files from:
C:\ImsEic325-PCI\Win2000

Click 'OK' and after a while you'll get the following:


Found New Hardware Wizard

Driver Files Search Results

The wizard has finished searching for driver files for your hardware device.

The wizard found a driver for the following device:

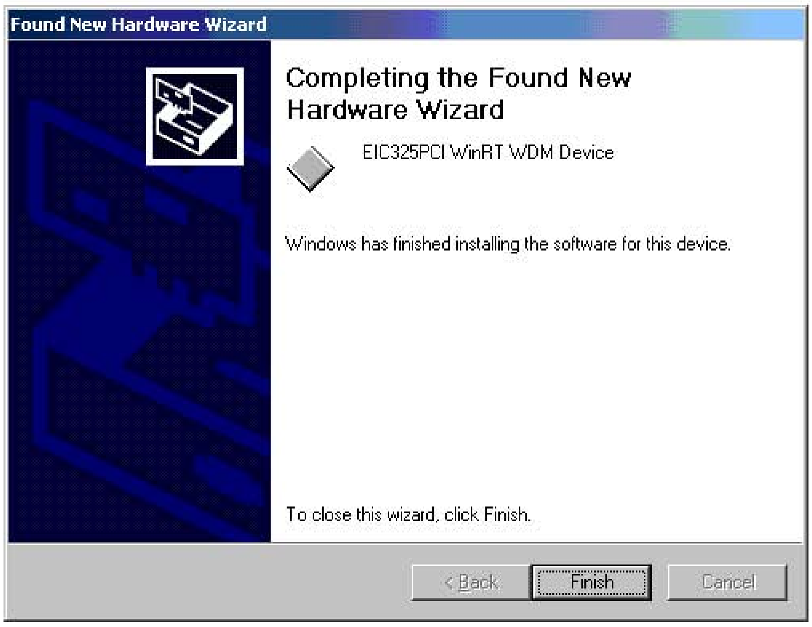
PCI Device

Windows found a driver for this device. To install the driver Windows found, click Next.

c:\imseic325-pci\win2000\imseic325-pciwdm.inf

Click 'Next'; the new software will be installed.

You'll get the following display:


Found New Hardware Wizard

Completing the Found New Hardware Wizard

EIC325PCI WinRT WDM Device

Windows has finished installing the software for this device.

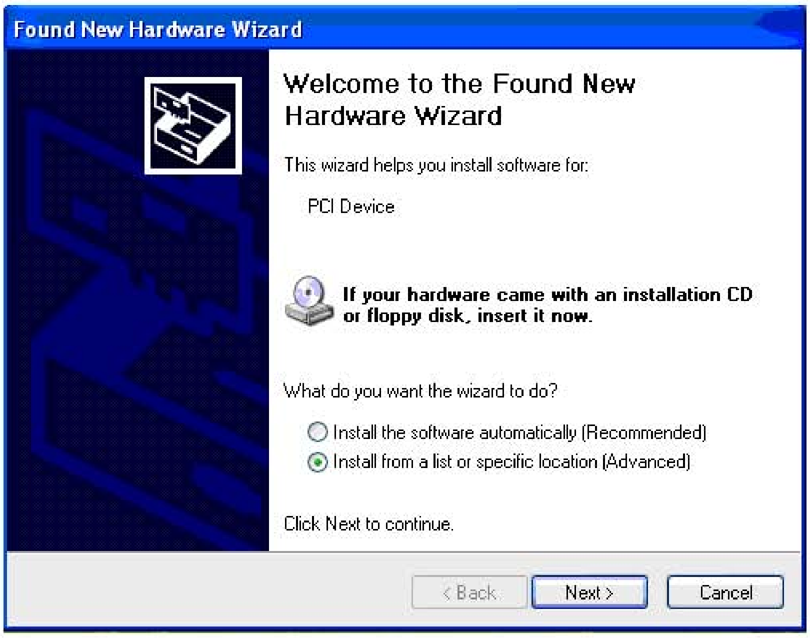
To close this wizard, click Finish.

Click 'Finish'. That's it!

WinXP

* Shut down your PC (i.e., power off).
* Insert the new card to a free PCI slot.
* Turn on your PC.

... and the 'Found New Hardware Wizard' will appear:



Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install software for:

PCI Device

If your hardware came with an installation CD or floppy disk, insert it now.

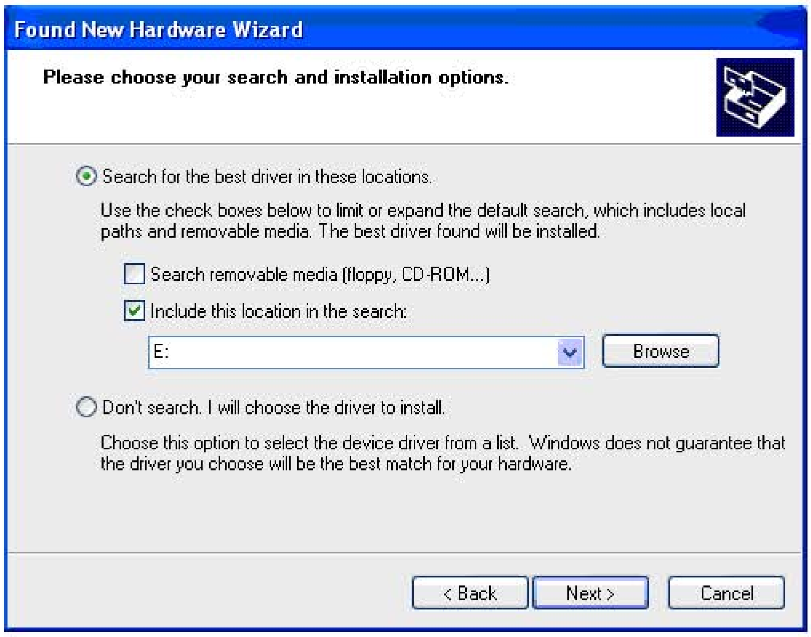
What do you want the wizard to do?

- ○ Install the software automatically (Recommended)
- ◉ Install from a list or specific location (Advanced)

Click Next to continue.

1. Select the second option (as shown).

2. Click 'Next' and you'll get the following:



Found New Hardware Wizard

Please choose your search and installation options.

◉ Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

☐ Search removable media (floppy, CD-ROM...)

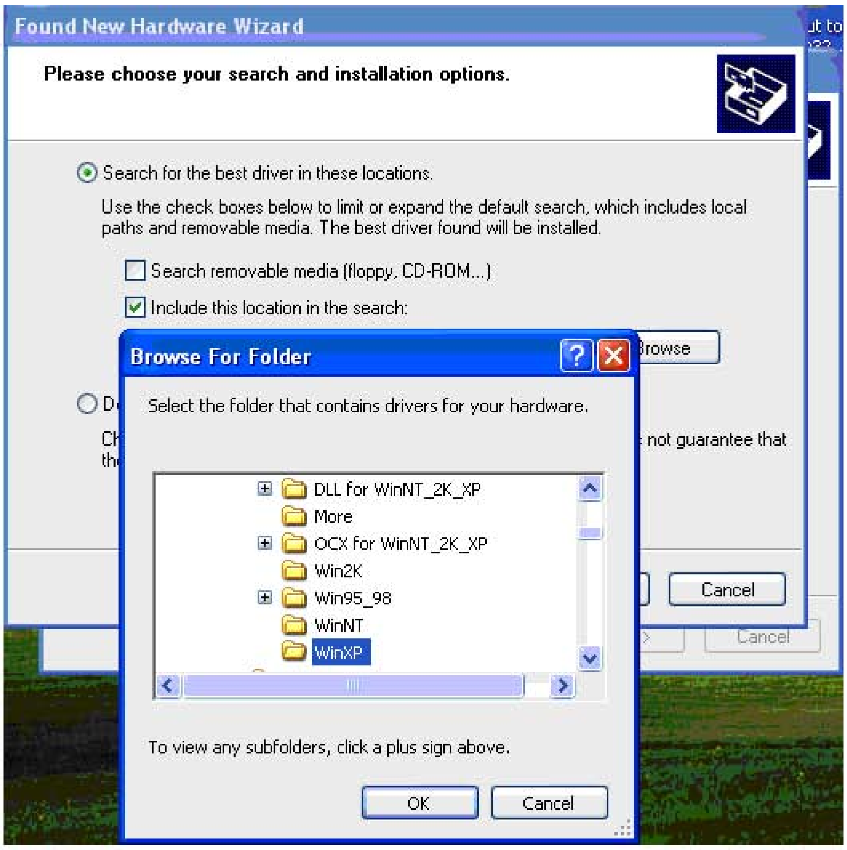
☑ Include this location in the search:

E:

○ Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

Click 'Browse...' and browse to the folder where you copied the files during driver installation (section 3.1.1), for example:


Found New Hardware Wizard

Please choose your search and installation options.

● Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

☐ Search removable media (floppy, CD-ROM...)

☑ Include this location in the search:

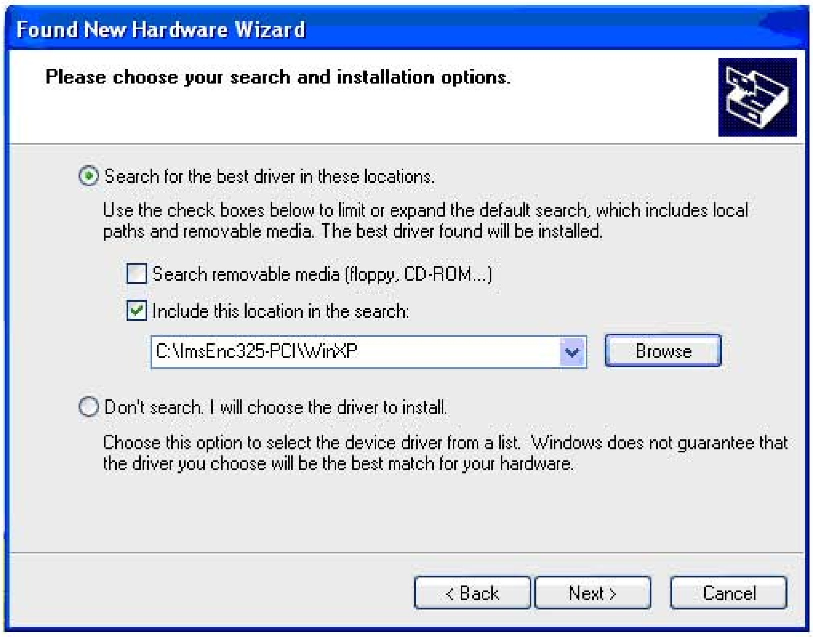
Browse For Folder

Select the folder that contains drivers for your hardware.

- ⊞ DLL for WinNT_2K_XP
- More
- ⊞ OCX for WinNT_2K_XP
- Win2K
- ⊞ Win95_98
- WinNT
- WinXP

To view any subfolders, click a plus sign above.

Click 'OK' and you'll get the following:


Found New Hardware Wizard

Please choose your search and installation options.

● Search for the best driver in these locations.

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

☐ Search removable media (floppy, CD-ROM...)

☑ Include this location in the search:

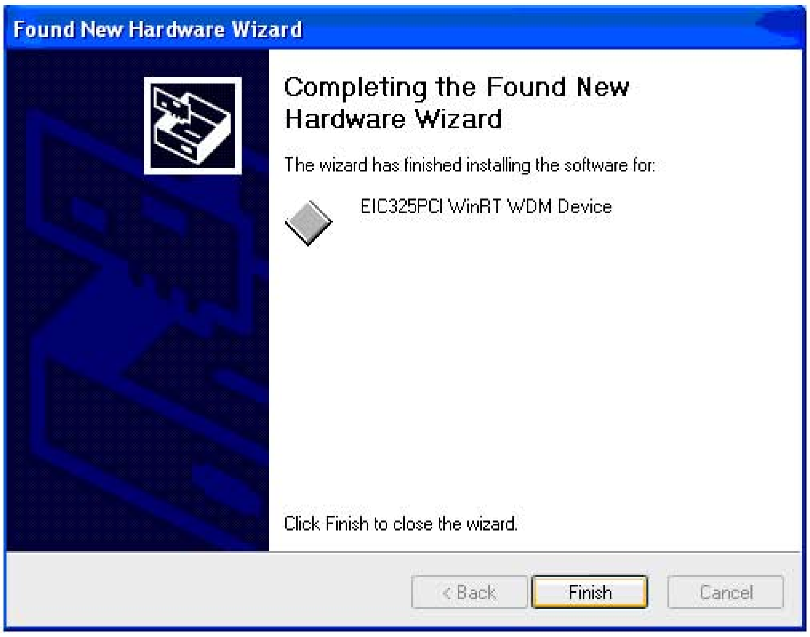
C:\ImsEnc325-PCI\WinXP

○ Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

Click 'Next'; the new software will be installed.

You'll get the following display:



Found New Hardware Wizard

Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

EIC325PCI WinRT WDM Device

Click Finish to close the wizard.

Click 'Finish'. That's it!

APPENDIX D: PRODUCT DEVELOPMENT

Main Milestones in Product Development:

- Dec. '01: V1.00: Product launching.
- Apr. '02: V1.02: Support under WinNT/2K/XP (refer to section 3.1.1 & appendix C).
- May '02: V1.03: In addition to the OCX, a DLL is supplied too (WinNT/2K/XP only).
- May '03: V2.00: New feature: Option to request a snapshot of the position counters in real-time, using a hardware input (in addition to the veteran software function for this operation). The new feature uses the last input (input #11); hence, input #11 is no longer operational as a general input, and there are now only 10 (vs. 11 before) general inputs. For details, refer to section 1.2.1 and appendix A.
- Oct. '03: V2.01: Better support for installation on WinXP (though it was possible also before based on the procedure for Win2K).

MADE IN USA

WARRANTY/DISCLAIMER

OMEGA ENGINEERING, INC. warrants this unit to be free of defects in materials and workmanship for a period of **13 months** from date of purchase. OMEGA's WARRANTY adds an additional one (1) month grace period to the normal **one (1) year product warranty** to cover handling and shipping time. This ensures that OMEGA's customers receive maximum coverage on each product.

If the unit malfunctions, it must be returned to the factory for evaluation. OMEGA's Customer Service Department will issue an Authorized Return (AR) number immediately upon phone or written request. Upon examination by OMEGA, if the unit is found to be defective, it will be repaired or replaced at no charge. OMEGA's WARRANTY does not apply to defects resulting from any action of the purchaser, including but not limited to mishandling, improper interfacing, operation outside of design limits, improper repair, or unauthorized modification. This WARRANTY is VOID if the unit shows evidence of having been tampered with or shows evidence of having been damaged as a result of excessive corrosion; or current, heat, moisture or vibration; improper specification; misapplication; misuse or other operating conditions outside of OMEGA's control. Components in which wear is not warranted, include but are not limited to contact points, fuses, and triacs.

OMEGA is pleased to offer suggestions on the use of its various products. However, OMEGA neither assumes responsibility for any omissions or errors nor assumes liability for any damages that result from the use of its products in accordance with information provided by OMEGA, either verbal or written. OMEGA warrants only that the parts manufactured by the company will be as specified and free of defects. OMEGA MAKES NO OTHER WARRANTIES OR REPRESENTATIONS OF ANY KIND WHATSOEVER, EXPRESSED OR IMPLIED, EXCEPT THAT OF TITLE, AND ALL IMPLIED WARRANTIES INCLUDING ANY WARRANTY OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE HEREBY DISCLAIMED. LIMITATION OF LIABILITY: The remedies of purchaser set forth herein are exclusive, and the total liability of OMEGA with respect to this order, whether based on contract, warranty, negligence, indemnification, strict liability or otherwise, shall not exceed the purchase price of the component upon which liability is based. In no event shall OMEGA be liable for consequential, incidental or special damages.

CONDITIONS: Equipment sold by OMEGA is not intended to be used, nor shall it be used: (1) as a "Basic Component" under 10 CFR 21 (NRC), used in or with any nuclear installation or activity; or (2) in medical applications or used on humans. Should any Product(s) be used in or with any nuclear installation or activity, medical application, used on humans, or misused in any way, OMEGA assumes no responsibility as set forth in our basic WARRANTY/DISCLAIMER language, and, additionally, purchaser will indemnify OMEGA and hold OMEGA harmless from any liability or damage whatsoever arising out of the use of the Product(s) in such a manner.

RETURN REQUESTS/INQUIRIES

Direct all warranty and repair requests/inquiries to the OMEGA Customer Service Department. BEFORE RETURNING ANY PRODUCT(S) TO OMEGA, PURCHASER MUST OBTAIN AN AUTHORIZED RETURN (AR) NUMBER FROM OMEGA'S CUSTOMER SERVICE DEPARTMENT (IN ORDER TO AVOID PROCESSING DELAYS). The assigned AR number should then be marked on the outside of the return package and on any correspondence.

The purchaser is responsible for shipping charges, freight, insurance and proper packaging to prevent breakage in transit.

FOR **WARRANTY** RETURNS, please have the following information available BEFORE contacting OMEGA:

1. Purchase Order number under which the product was PURCHASED,
2. Model and serial number of the product under warranty, and
3. Repair instructions and/or specific problems relative to the product.

FOR **NON-WARRANTY** REPAIRS, consult OMEGA for current repair charges. Have the following information available BEFORE contacting OMEGA:

1. Purchase Order number to cover the COST of the repair,
2. Model and serial number of the product, and
3. Repair instructions and/or specific problems relative to the product.

OMEGA's policy is to make running changes, not model changes, whenever an improvement is possible. This affords our customers the latest in technology and engineering.

OMEGA is a registered trademark of OMEGA ENGINEERING, INC.

M4272/0106