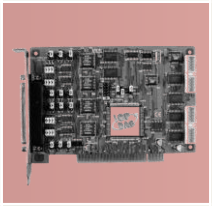
PCI-TMC12 Series User Manual

12-channel Timer/Counter Board Version 2.7, Dec. 2016



Support

This manual relates to the following boards:

PCI-TMC12, PCI-TMC12A and PCI-TMC12AU

Warranty

All products manufactured by ICP DAS are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

Warning

ICP DAS assumes no liability for damages consequent to the use of this product. ICP DAS reserves the right to change this manual at any time without notice. The information furnished by ICP DAS is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS for its use, nor for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademarks

Names are used for identification purposes only and may be registered trademarks of their respective companies.

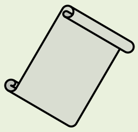
Contact US

If you have any question, feel to contact us by email at:
Email: service@icpdas.com or service.icpdas@gmail.com
We will respond to you within 2 working days.

TABLE OF CONTENTS

Packing List

The shipping package includes the following items:

	One PCI-TMC12/TMC12A/TCM12AU Series board.
	One printed Quick Start Guide
	One software utility CD
	One CA-4002 D-sub connector

Note	*If any of these items are missing or damaged, please contact the local distributor for more information. Save the shipping materials and cartons in case you need to ship the module in the future.*

1. Introduction

The PCI-TMC12AU card is the new generation product that ICP DAS provides to meet RoHS compliance requirement. The new PCI-TMC12AU card is designed as a drop-in replacement for the PCI-TMC12A, and users can replace the PCI-TMC12A by the PCI-TMC12AU directly without software/driver modification.

The PCI-TMC12AU supports both 5 V and 3.3 V PCI bus, while the PCI-TMC12A supports 5 V PCI bus only. They feature twelve 16-bit timers/counters (four 82C54 chips x 3 timers/counters), 16 TTL digital input channels and 16 TTL digital output channels. The two onboard clocks (8 M/1.6 M and 0.8 M/80 K) are jumper selectable and provide a high-resolution clock source for timers/counters. Counters/timers can be used for industrial and laboratory applications such as pulse/event/switch-toggle counting, frequency readings, elapsed time measurement, pulse-width measurement, PWM (pulse-width-modulated) output, and pulse (square wave) and rate generation, etc.

The PCI-TMC12AU also adds a Card ID switch on board. Users can set Card ID on a board and recognize the board by the ID via software when using two or more PCI-TMC12AU cards in one computer.

These cards support a variety of operating systems, such as DOS, Windows 2000, 32-/64-bit Windows XP/2003/2008/7/8 and Windows 10. ICP DAS also provides a DLL and Active X control for the PCI-TMC12A/TMC12AU, together with sample programs in various languages, including Turbo C++, Borland C++, Visual C++, Borland Delphi, Borland C++ Builder, Visual Basic, C#.NET, Visual Basic.NET and LabVIEW, enabling help users to quickly and easily develop their custom applications.

1.1 Features

- Supports the +5 V PCI Bus for PCI-TMC12A
- Supports the +3.3/+5 V PCI Bus for PCI-TMC12AU
- General Purpose Timer/Counter and Digital I/O Board
- 4 Onboard 8254 Timer/Counter Chips
- 12 Independent 16-bit Timers/Counters
- 12 External Clock Inputs
- 12 External Gate Control Inputs
- 12 Timer/Counter Output Channels
- The 16-bit Timer/Counter can be cascaded to create a 32/48-bit Timer/Counter
- Gate Input can be either an External Signal or the Output of a previous Timer/Counter Channel
- Four Interrupt Sources
- Two Internal Clock Sources
- 16-TTL Digital Input Channels and 16 TTL Digital Output Channels
- Supports Card ID (SMD Switch) for PCI-TMC12AU
- More flexible interrupt mechanism
- Hardware mechanism for the generation of two starting-clocks

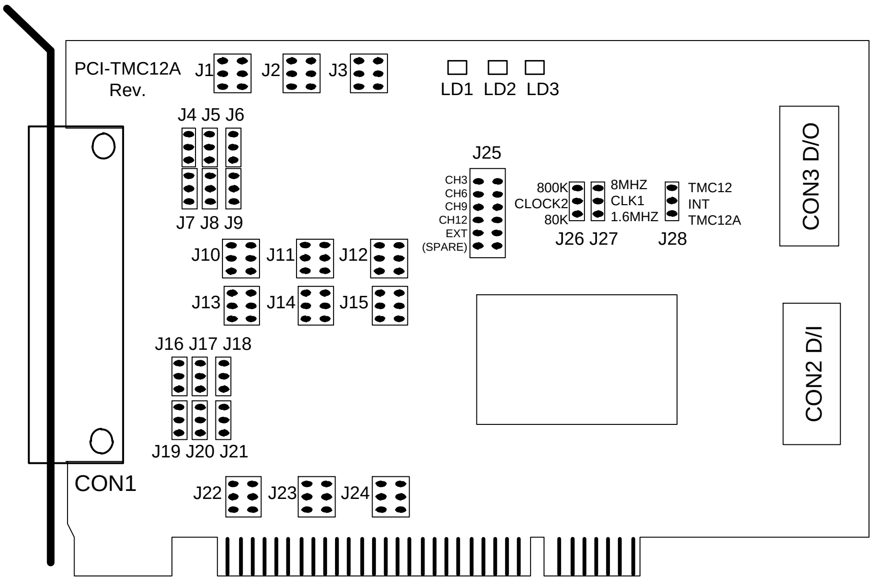
1.2 Specifications

Model Name	PCI-TMC12AU	PCI-TMC12A
Digital Input		
Channels	16	
Compatibility	5 V/TTL	
Input Voltage	Logic 0: 0.8 V Max. Logic 1: 2.0 V Min.	
Response Speed	2.0 MHz (Typical)	
Digital Output		
Channels	16	
Compatibility	5 V/TTL	
Output Voltage	Logic 0: 0.4 V Max. Logic 1: 2.4 V Min.	
Output Capability	Sink: 24 mA @ 0.8 V Source: 15 mA @ 2.0 V	
Response Speed	2.0 MHz (Typical)	
Timer/Counter		
Channels	12 (Independent x 12)	
Resolution	16-bit	
Compatibility	5 V/TTL	
Input Frequency	10 MHz Max.	
Reference Clock	Internal: 8 MHz	
General		
Bus Type	3.3 V/5 V Universal PCI, 32-bit, 33 MHz	5 V PCI, 32-bit, 33 MHz
Data Bus	16-bit	
	Yes (4-bit)	No
I/O Connector	Female DB37 x 1 20-bit Box Header x 2	
Dimensions (L x W x D)	150 mm x 105 mm x 22 mm	
Power Consumption	500 mA @ +5 V	
Operating Temperature	0 ~ 60 °C	
Storage Temperature	-20 ~ 70 °C	
Humidity	5 ~ 85% RH, Non-condensing	

2. Hardware Configuration

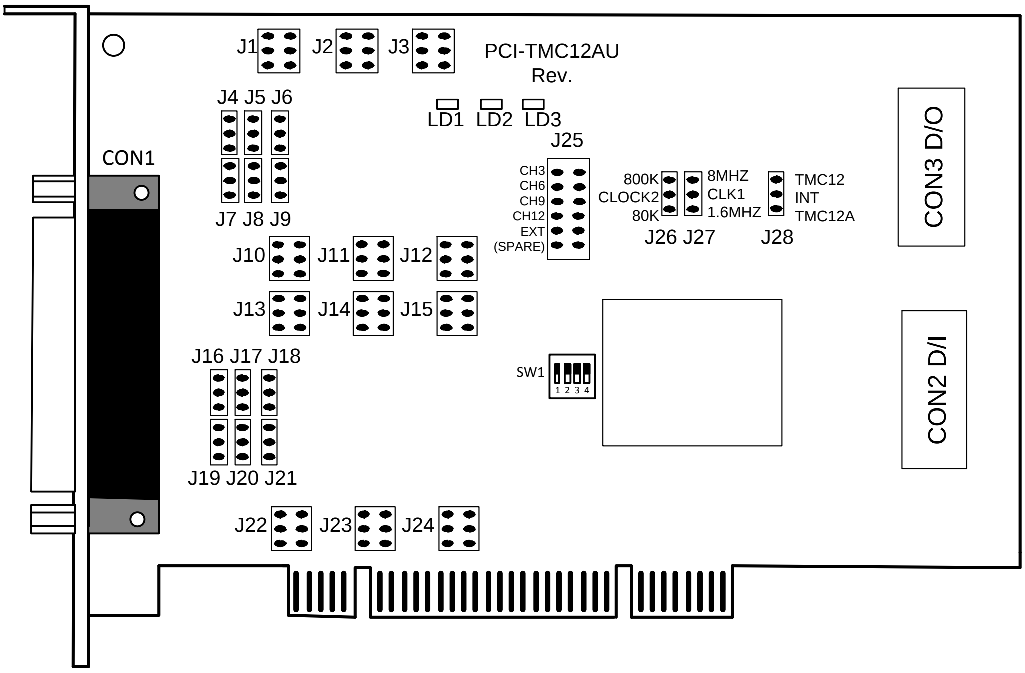
2.1 Board Layout

- The layout of the PCI-TMC12/TMC12A board is illustrated below.



CON1	The terminal for Timer/Counter. Refer to Section 2.5.
CON2	The terminal for Digital Input. Refer to Section 2.5.
CON3	The terminal for Digital Output. Refer to Section 2.5.
J1/J2/J3 **J10/J11/J12/J13/J14/J15** **J22/J23/J24**	The jumper for selecting CLK1 to CLK12. Refer to Section 2.4.3.
J4/J5/J6/J7/J8/J9 **J16/J17/J18/J19/J20/J21**	The jumper for selecting CLK1 to CLK12. Refer to Section 2.4.4.
J25	The jumper for selecting the interrupt source. Refer to Section 2.4.5.
J26/J27	The jumper for selecting CLOCK1 and CLOCK2. Refer to Section 2.4.2.
J28	The jumper for selecting PCI-TMC12 and PCI-TMC12A for the PCI-TMC12A/TMC12AU only. Refer to Section 2.4.1.
LD1/LD2/LD3	LED Indicator for the PCI-TMC12A/TMC12AU only.

➢ The layout of the PCI-TMC12AU board is illustrated below.

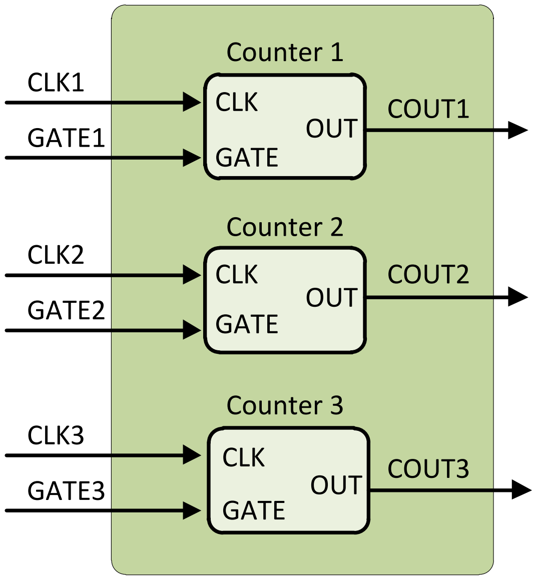


CON1	The terminal for Timer/Counter. Refer to Section 2.5.
CON2	The terminal for Digital Input. Refer to Section 2.5.
CON3	The terminal for Digital Output. Refer to Section 2.5.
J1/J2/J3 **J10/J11/J12/J13/J14/J15** **J22/J23/J24**	The jumper for selecting CLK1 to CLK12. Refer to Section 2.4.3.
J4/J5/J6/J7/J8/J9 **J16/J17/J18/J19/J20/J21**	The jumper for selecting CLK1 to CLK12. Refer to Section 2.4.4.
J25	The jumper for selecting the interrupt source. Refer to Section 2.4.5.
J26/J27	The jumper for selecting CLOCK1 and CLOCK2. Refer to Section 2.4.2.
J28	The jumper for selecting PCI-TMC12 and PCI-TMC12A for the PCI-TMC12A/TMC12AU only. Refer to Section 2.4.1.
LD1/LD2/LD3	LED Indicator for the PCI-TMC12A/TMC12AU only.
SW1	Card ID function. Refer to Section 2.6.

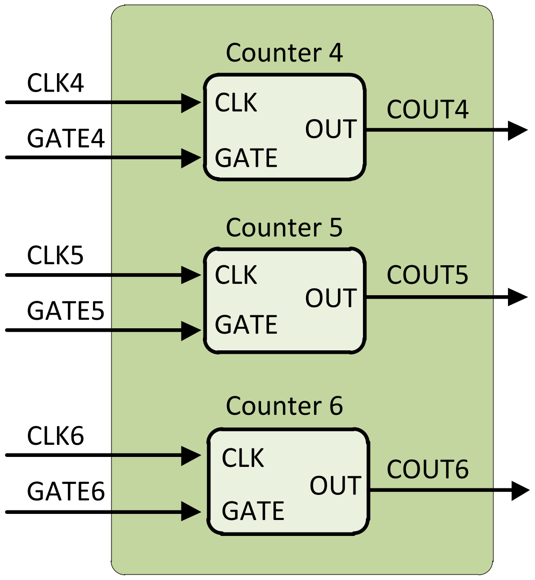
2.2 Counter Architecture

There are four 8254 chips on the PCI-TMC12 series board. The block diagrams for the four chips are as follows:

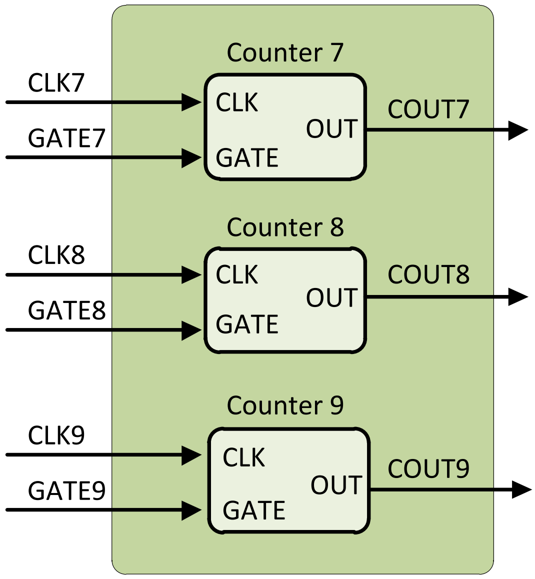
8254 CHIP #1 (U12)



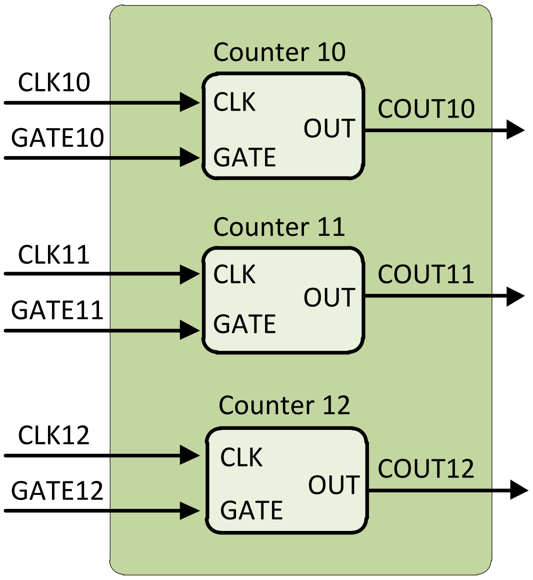
8254 CHIP #2 (U8)



8254 CHIP #3 (U3)

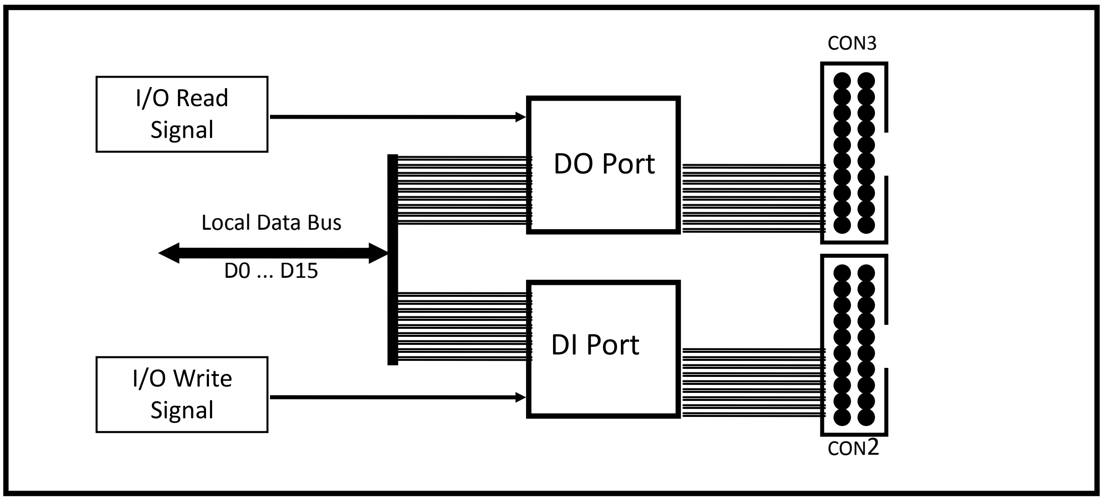


8254 CHIP #4 (U1)



2.3 DI/DO Block Diagram

The PCI-TMC12 series card provides 16 Digital Input and 16 Digital Output channel. All levels are TTL compatible. An illustration of the connections and the block diagram are as follows:



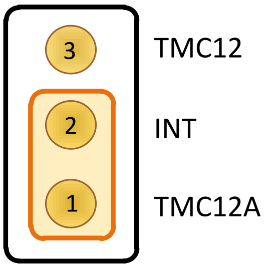
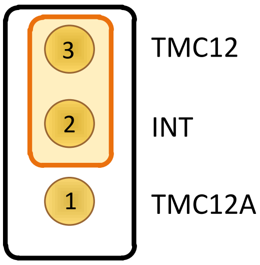
The Digital Input port can be connected to a DB-16P, which is a 16-channel isolated Digital Input daughterboard. The Digital Output port can be connected to either a DB-16R or DB-24PR. The DB-16R is a 16-channel Relay Output daughterboard, while the DB-24R is a 24-channel Power Relay Output daughterboard.

2.4 Jumper Settings

2.4.1 J28: Selecting PCI-TMC12 and PCI-TMC12A

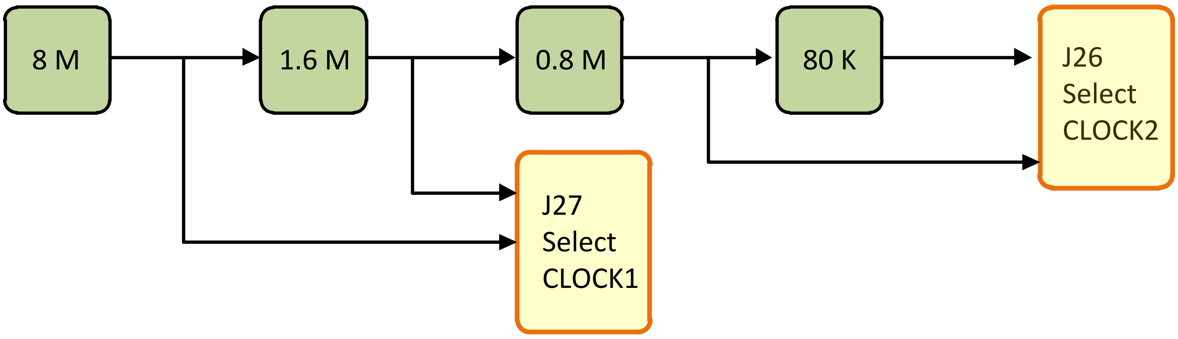
- Any existing programs designed for the PCI-TMC12 can be executed on the PCI-TMC12A without requiring any modification to the code.
- The PCI-TMC12A/TMC12AU provides more features. Refer to Section 3.4 for more detailed information.

The following figure shows the positions of the jumper pins used to select PCI-TMC12 or PCI-TMC12A.

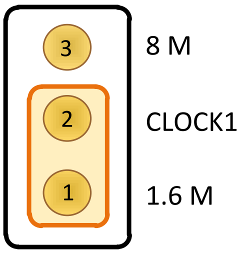
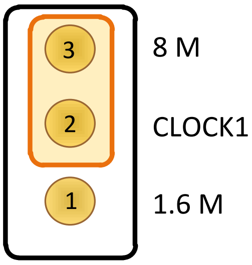
Jumper Positions for J28	
	
PCI-TMC12A **PCI-TMC12AU**	**PCI-TMC12** **(Default Settings)**

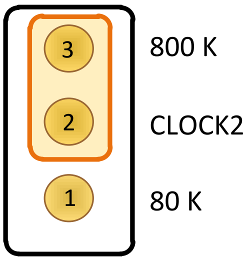
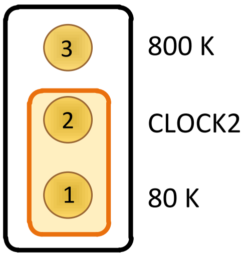
2.4.2 J26, J27: Selecting CLOCK1 and CLOCK2

The PCI-TMC12 series card contains two stable internal clock sources, which are defined as ***CLOCK1 and CLOCK2***. ***CLOCK1*** can be set to either 8 M or 1.6 M, and is select by using jumper **J27**. **CLOCK2** can be set to either 0.8 M or 80 k and is selected using jumper **J26**. The block diagram for the internal clock sources is as follows:



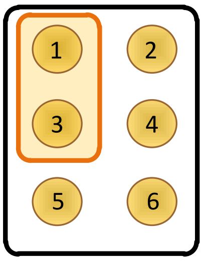
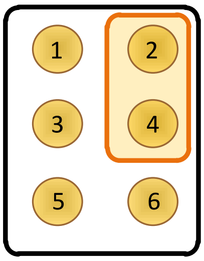
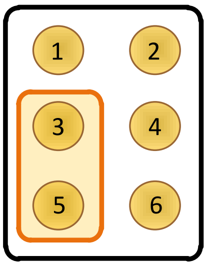
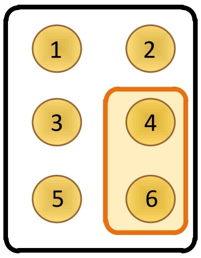
The following figure shows the positions of the jumper pins used to select CLOCK1 and CLOCK2.

Jumper Positions for J27	
	
CLOCK1 = 1.6 M **(Default Settings)**	**CLOCK 1 = 8 M**

Jumper Positions for J26	
	
CLOCK2 = 800 K **(Default Settings)**	CLOCK2 = 80 K

2.4.3 J1/J2/J3/J10/J11/J12/J13/J14/J15/J22/J23/J24: Selecting CLK1 to CLK12

The following figure shows the positions of the jumper pins used to select CLK1 to CLK12.

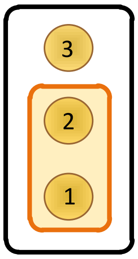
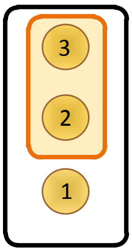
Jumper Positions for J1, J2, J3, J10, J11, J12, J13, J14, J15, J22, J23, J24			
			
Select CLOCK1	**Select CLOCK 2** **(Default Settings)**	Select COUTn-1 (Last Channel)	Select EXT_CLKn (External CLKn)

The following table provides an overview of the Clock sources and jumper positions.

CLK	Jumper	Selected Sources
CLK1	J22	CLOCK1, CLOCK2, COUT6, ECLK1
CLK2	J23	CLOCK1, CLOCK2, COUT1, ECLK2
CLK3	J24	CLOCK1, CLOCK2, COUT2, ECLK3
CLK4	J13	CLOCK1, CLOCK2, COUT3, ECLK4
CLK5	J14	CLOCK1, CLOCK2, COUT4, ECLK5
CLK6	J15	CLOCK1, CLOCK2, COUT5, ECLK6
CLK7	J10	CLOCK1, CLOCK2, COUT12, ECLK7
CLK8	J11	CLOCK1, CLOCK2, COUT7, ECLK8
CLK9	J12	CLOCK1, CLOCK2, COUT8, ECLK9
CLK10	J1	CLOCK1, CLOCK2, COUT9, ECLK10
CLK11	J2	CLOCK1, CLOCK2, COUT10, ECLK11
CLK12	J3	CLOCK1, CLOCK2, COUT11, ECLK12

2.4.4 J4/J5/J6/J7/J8/J9/J16/J17/J18/J19/J20/J21: Selecting GATE1 to GATE2

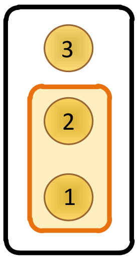
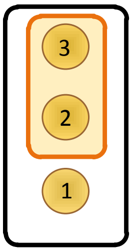
The following figure shows the positions of the jumper pins used to select GATE1 to GATE6.

Jumper Positions for J	
	
GATEn = EXTGn **(Default Settings)**	GATEn = Inverted COUTn-1

The following table provides an overview of the GATE and jumper positions.

GATE	Jumper	Selected Sources
GATE1	J19	Inverted COUT6, EXTG1
GATE2	J20	Inverted COUT1, EXTG2
GATE3	J21	Inverted COUT2, EXTG3
GATE4	J16	Inverted COUT3, EXTG4
GATE5	J17	Inverted COUT4, EXTG5
GATE6	J18	Inverted COUT5, EXTG6

The following figure shows the positions of the jumper pins used to select GATE7 to GATE12.

Jumper Positions for J	
	
GATEn = EXTGn **(Default Settings)**	GATEn = COUTn-1

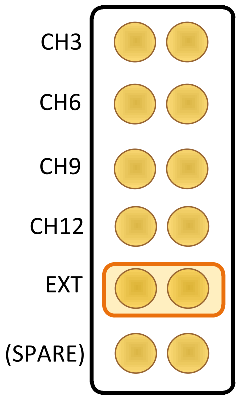
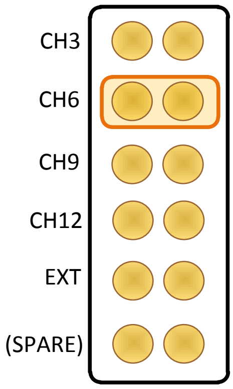
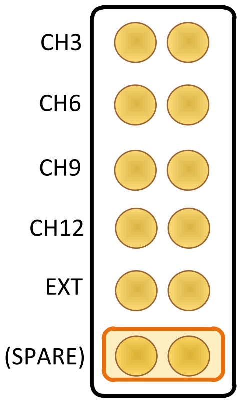
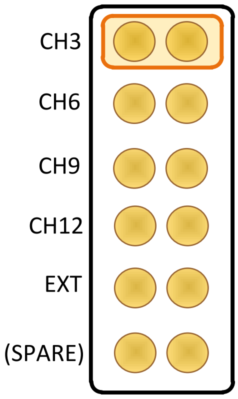
The following table provides an overview of the GATE and jumper positions.

GATE	Jumper	Selected Sources
GATE7	J7	COUT12, EXTG7
GATE8	J8	COUT7, EXTG8
GATE9	J9	COUT8, EXTG9
GATE10	J4	COUT9, EXTG10
GATE11	J5	COUT10, EXTG11
GATE12	J6	COUT11, EXTG12

2.4.5 J25: Selecting the Interrupt Source

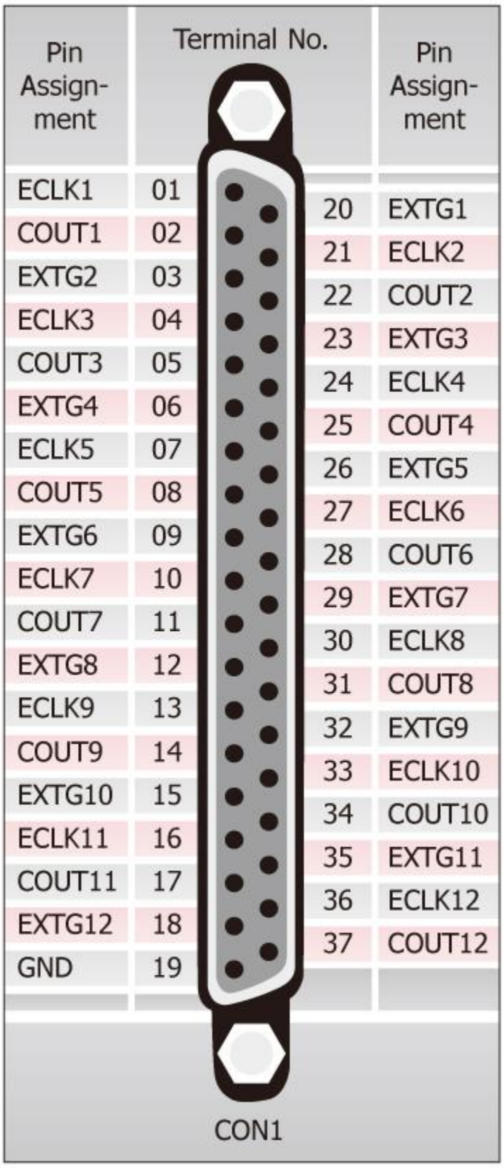
Five signals can be used as interrupt sources: CH3, CH6, CH9, CH12 and EXT, as described below:
CH3: The interrupt source is COUT3, i.e. the output of counter 3
CH6: The interrupt source is COUT6, i.e. the output of counter 6
CH9: The interrupt source is COUT9, i.e. the output of counter 9
CH12: The interrupt source is COUT12, i.e. the output of counter 12
EXT: The interrupt source is ECLK11, i.e. the external CLK for counter 11 via CON1
(SPARE): Indicates that there is no interrupt source

The following figure shows the positions of the jumper pins used to select interrupt source.

Jumper Positions for J25	
	
Interrupt Source = ECLK11	Interrupt Source = COUT6
	
No Interrupt Source **(Default Settings)**	Interrupt Source = COUT3

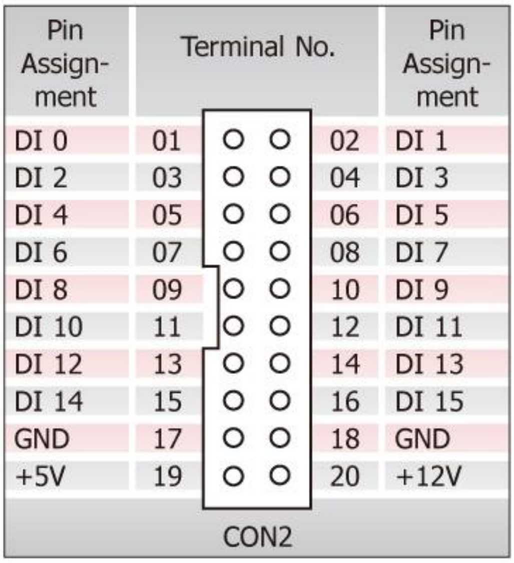
2.5 Pin Assignments

The PCI-TMC12 series card includes three connectors, CON1, CON2 and CON3.



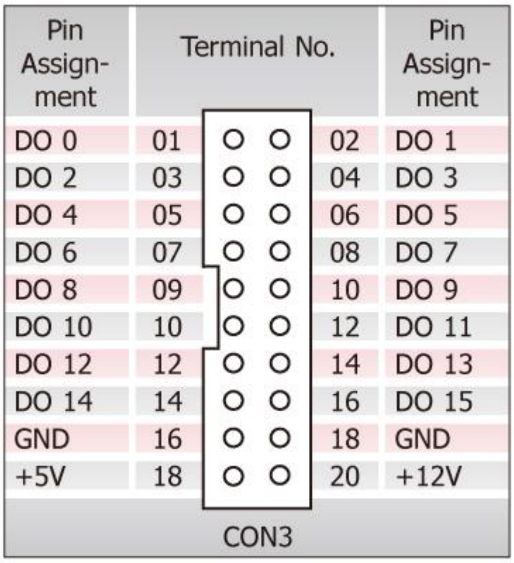
Pin Assign-ment	Terminal No.		Pin Assign-ment
ECLK1	01	20	EXTG1
COUT1	02	21	ECLK2
EXTG2	03	22	COUT2
ECLK3	04	23	EXTG3
COUT3	05	24	ECLK4
EXTG4	06	25	COUT4
ECLK5	07	26	EXTG5
COUT5	08	27	ECLK6
EXTG6	09	28	COUT6
ECLK7	10	29	EXTG7
COUT7	11	30	ECLK8
EXTG8	12	31	COUT8
ECLK9	13	32	EXTG9
COUT9	14	33	ECLK10
EXTG10	15	34	COUT10
ECLK11	16	35	EXTG11
COUT11	17	36	ECLK12
EXTG12	18	37	COUT12
GND	19		

CON1



Pin Assign-ment	Terminal No.		Pin Assign-ment
DI 0	01	02	DI 1
DI 2	03	04	DI 3
DI 4	05	06	DI 5
DI 6	07	08	DI 7
DI 8	09	10	DI 9
DI 10	11	12	DI 11
DI 12	13	14	DI 13
DI 14	15	16	DI 15
GND	17	18	GND
+5V	19	20	+12V

CON2



Pin Assign-ment	Terminal No.		Pin Assign-ment
DO 0	01	02	DO 1
DO 2	03	04	DO 3
DO 4	05	06	DO 5
DO 6	07	08	DO 7
DO 8	09	10	DO 9
DO 10	10	12	DO 11
DO 12	12	14	DO 13
DO 14	14	16	DO 15
GND	16	18	GND
+5V	18	20	+12V

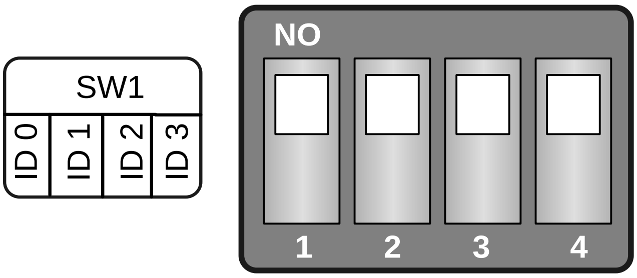
CON3

Notes:

1. ***ECLKn:** External clock source for Counter n*
2. ***EXTGn:** External gate control signal for Counter n*
3. ***COUTn:** Output of Timer/Counter n*
4. *All signals are TTL compatible*

2.6 Card ID Switch

The PCI-TMC12AU has a Card ID switch (SW1) with which users can recognize the board by the ID via software when using two or more PCI-TMC12AU cards in one computer. The default Card ID is 0x0. For detail SW1 Card ID settings, please refer to Table 2.1. *Note that the Card ID function is only supported by the PCI-TMC12AU.*



(Default Settings)

Table 2.1 (*) Default Settings; OFF → 1; ON → 0

Card ID (Hex)	1 ID0	2 ID1	3 ID2	4 ID3
(*) 0x0	ON	ON	ON	ON
0x1	**OFF**	ON	ON	ON
0x2	ON	**OFF**	ON	ON
0x3	**OFF**	**OFF**	ON	ON
0x4	ON	ON	**OFF**	ON
0x5	**OFF**	ON	**OFF**	ON
0x6	ON	**OFF**	**OFF**	ON
0x7	**OFF**	**OFF**	**OFF**	ON
0x8	ON	ON	ON	**OFF**
0x9	**OFF**	ON	ON	**OFF**
0xA	ON	**OFF**	ON	**OFF**
0xB	**OFF**	**OFF**	ON	**OFF**
0xC	ON	ON	**OFF**	**OFF**
0xD	**OFF**	ON	**OFF**	**OFF**
0xE	ON	**OFF**	**OFF**	**OFF**
0xF	**OFF**	**OFF**	**OFF**	**OFF**

3. Hardware Installation

Note:

It is recommended that the driver is installed before installing the hardware as the computer may need to be restarted once the driver is installed in certain operating systems, such as Windows 2000 or Windows XP, etc. Installing the driver first helps reduce the time required for installation and restarting the computer.

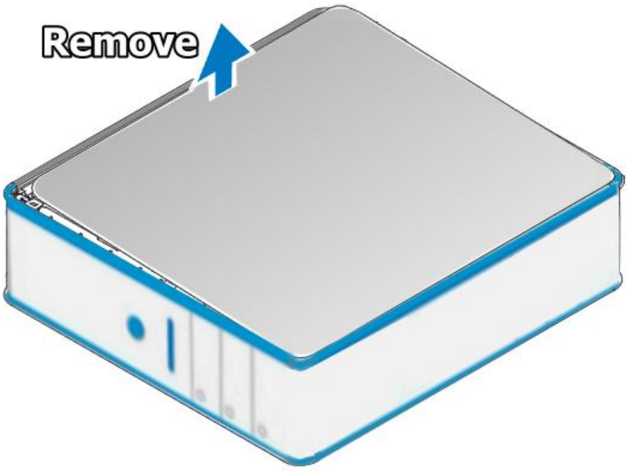
To install your PCI-TMC12 Series board, complete the following steps:

Step 1: Install the driver for your board on Host computer.

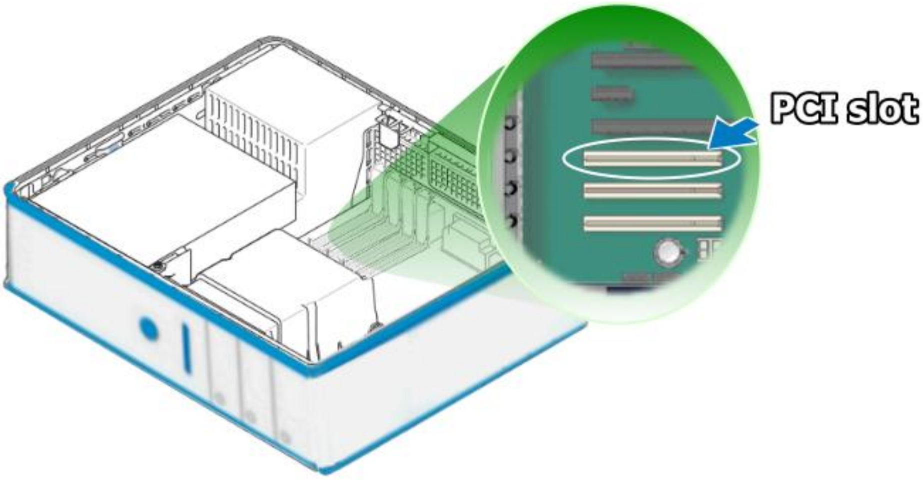
For detailed information about the driver installation, please refer to Chapter 4 Software Installation.

Step 2: Shut down and switch off the power to the computer, and then disconnect the power supply.

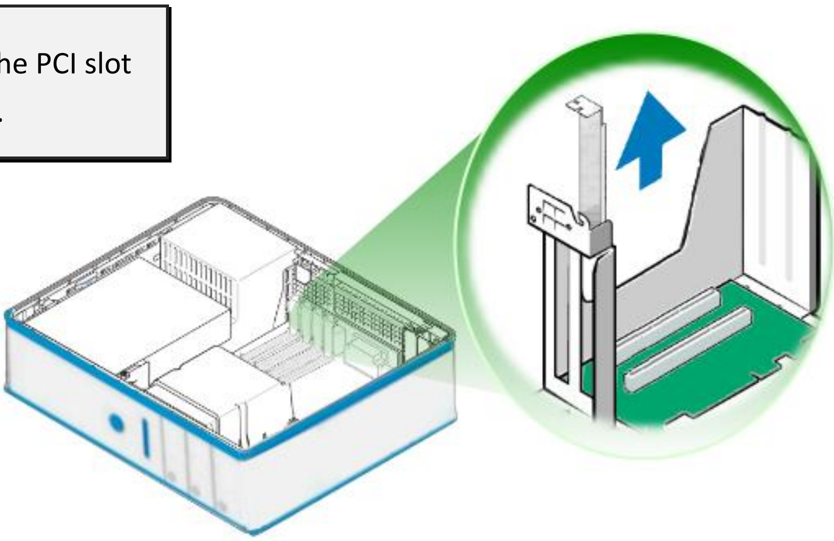
Step 3: Remove the cover from the computer.

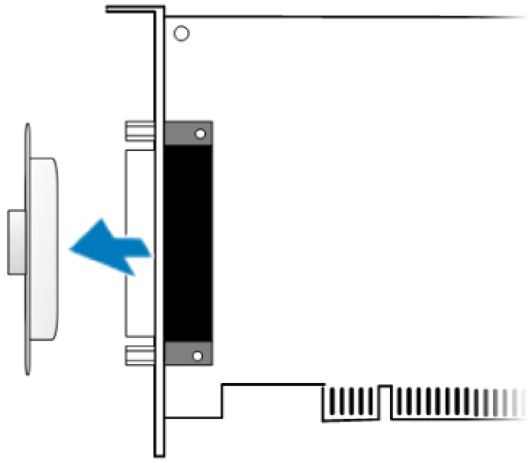


Step 4: Select a vacant PCI slot.



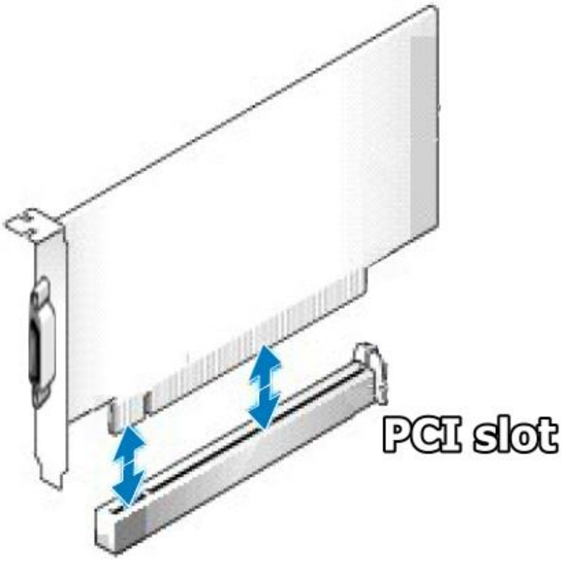
Step 5: Unscrew and remove the PCI slot cover from the computer case.

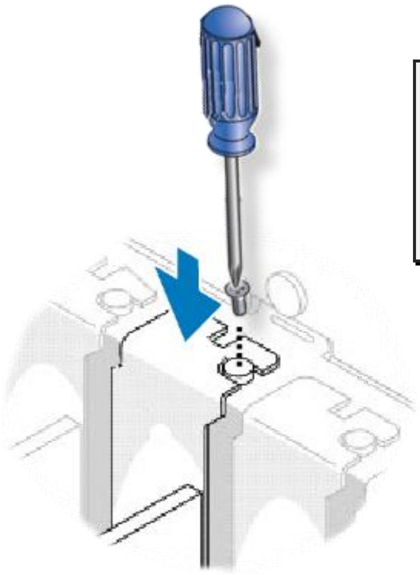




Step 6: Remove the connector cover from your board.

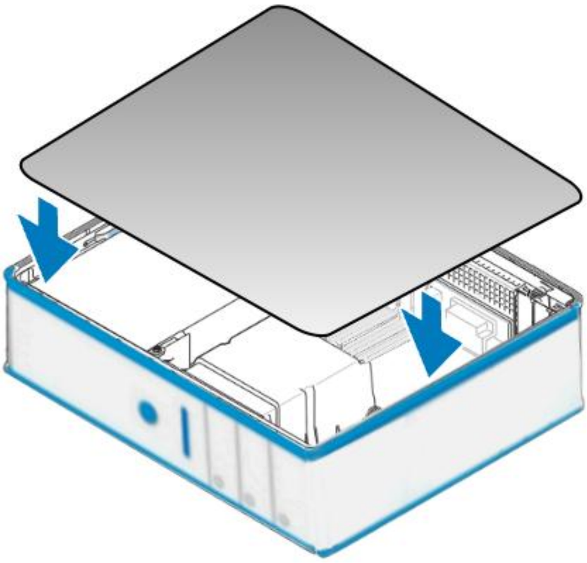
Step 7: Carefully insert your board into the PCI slot by gently pushing down on both sides of the board until it slides into the PCI connector.





Step 8: Confirm that the board is correctly inserted in the motherboard, and then secure your board in place using the retaining screw that was removed in Step 5.

Step 9: Replace the covers on the computer.



Step 10: Re-attach any cables, insert the power cord and then switch on the power to the computer.

Once the computer reboots, follow any message prompts that may be displayed to complete the Plug and Play installation procedure. Refer to Chapter 4 Software Installation for more information.

4. Software Installation

This chapter provides a detailed description of the process for installing the driver for the PCI-TMC12 series board as well as how to verify whether your board was properly installed. PCI-TMC12 series can be used on DOS, Linux and 32/64-bit versions of Windows XP/2003/2008/7/8/10 based systems, and the drivers are fully Plug and Play compliant for easy installation.

4.1 Obtaining/Installing the Driver Installer Package

The driver installation package for PCI-TMC12 series board can be found on the companion CD-ROM, or can be obtained from the ICP DAS FTP web site. Install the appropriate driver for your operating system. The location and website addresses for the installation package are indicated below.

- **UniDAQ Driver/SDK**

Operating System	Windows 2000, 32/64-bit Windows XP, 32/64-bit Windows 2003, 32/64-bit Windows 7, 32/64-bit Windows 2008, 32/64-bit Windows 8 and 32/64-bit Windows 10
Driver Name	**UniDAQ Driver/SDK (unidaq_win_setup_xxxx.exe)**
CD-ROM	CD:\\ NAPDOS\PCI\UniDAQ\DLL\Driver\
Web site	http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/unidaq/dll/driver/
Installing Procedure	To install the UniDAQ driver, follow the procedure described below. **Step 1:** Double-click the **UniDAQ_Win_Setupxxx.exe** icon to begin the installation process.

Installation Procedure	**Step 2:** When the “Welcome to the ICP DAS UniDAQ Driver Setup Wizard” screen is displayed, click the **“Next>”** button to start the installation. **Step 3:** On the “Information” screen, verify that the DAQ board is included in the list of supported devices, then click the **“Next>”** button. **Step 4:** On the “Select Destination Location” screen, click the **“Next>”** button to install the software in the default folder, **C:\ICPDAS\UniDAQ**. **Step 5:** On the “Select Components” screen, verify that the DAQ board is in the list of device, and then click the **“Next>”** button to continue. **Step 6:** On the “Select Additional Tasks” screen, click the **“Next>”** button to continue. **Step 7:** On the “Download Information” screen, click the **“Next>”** button to continue. **Step 8:** Once the installation has completed, click **“No, I will restart my computer later”**, and then click the **“Finish”** button. For more detailed information about how to install the UniDAQ driver, refer to “Section 2.2 Install UniDAQ Driver DLL” of the UniDAQ Software Manual, which can be found in the \NAPDOS\PCI\UniDAQ\Manual\ folder on the companion CD, or can be downloaded from: http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/unidaq/manual/

- **PCI-TMC12 Series Classic Driver**

Operating System	**Windows 95/98/ME, Windows NT, Windows 2000, 32-bit Windows XP, 32-bit Windows 2003, 32-bit Windows 7 and 32-bit Windows 8**
Driver Name	**PCI-TMC12 Series Classic Driver (PCI-TMC12_Win_Setup_xxxx.exe)**
CD-ROM	CD:\\NAPDOS\PCI\PCI-TMC12A\DLL_OCX\Driver
Web site	http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/pci-tmc12a/dll_ocx/driver/
Installing Procedure	Please follow the following steps to setup software: **Step 1:** Double click the **PCI-TMC12 Series Classic Driver** to setup it. **Step 2:** When the Setup Wizard screen is displayed, click the **Next>** button. **Step 3:** Select the folder where the drivers are to install. The **default path is C:\DAQPro\PCI-TMC12**. But if you wish to install the drivers to a different location , click the **“Browse…”** button and select the relevant folder and then click the **Next>** button. **Step 4:** Click the **Install** button to continue. **Step 5:** Click the **Finish** button. For more detailed information about how to install the PCI-TMC12 series classic driver, refer to “Section 2.2 Driver Installing Procedure” of the PCI-TMC12 DLL Software Manual, which can be found in the \NAPDOS\PCI\PCI-TMC12A\Manual\ folder on the companion CD, or can be downloaded from: http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/pci-tmc12a/manual/

4.2 PnP Driver Installation

512 KB Memory Board with DI/DO

3. Hardware Installation

Note:

It is recommended that the driver is installed before installing the hardware as the computer may need to be restarted once the driver is installed in certain operating systems, such as Windows 2000 or Windows XP, etc. Installing the driver first helps reduce the time required for installation and restarting the computer.

To install your PCI-M512 Series board, complete the following steps:

Step 1: Install the driver for your board on Host computer.

For detailed information about the driver installation, please refer to Chapter 4 Software Installation.

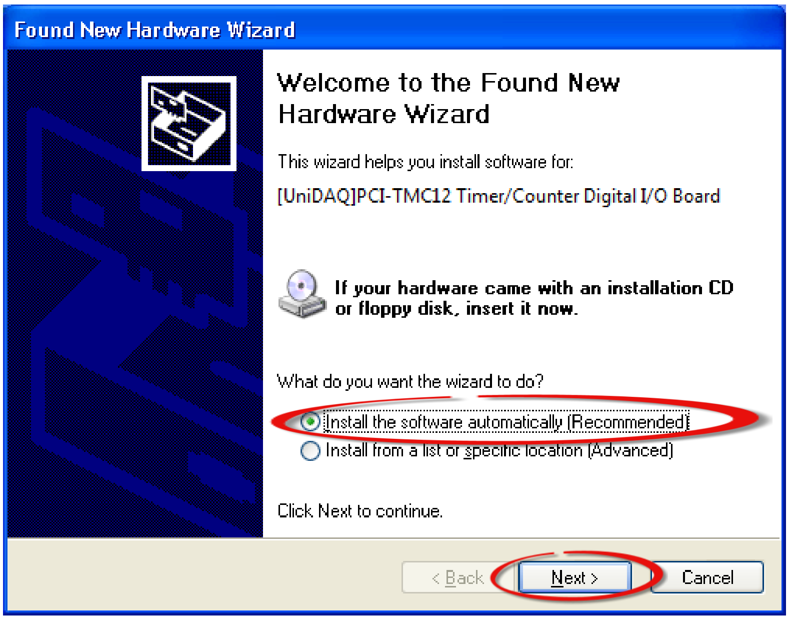
Step 1: Correctly shut down and power off your computer and disconnect the power supply, and then install your board into the computer.

For detailed information about the hardware installation of PCI-TMC12 Series board, please refer to Chapter 3 Hardware Installation.

Step 2: Power on the computer and complete the Plug and Play installation**.**

Note: More recent operating systems, such as Windows 7/8 will automatically detect the new hardware and install the necessary drivers etc., so Steps 3 to 5 can be skipped.

Step 3: Select **"Install the software automatically [Recommended]"** and click the **"Next>"** button.



Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install software for:

[UniDAQ]PCI-TMC12 Timer/Counter Digital I/O Board

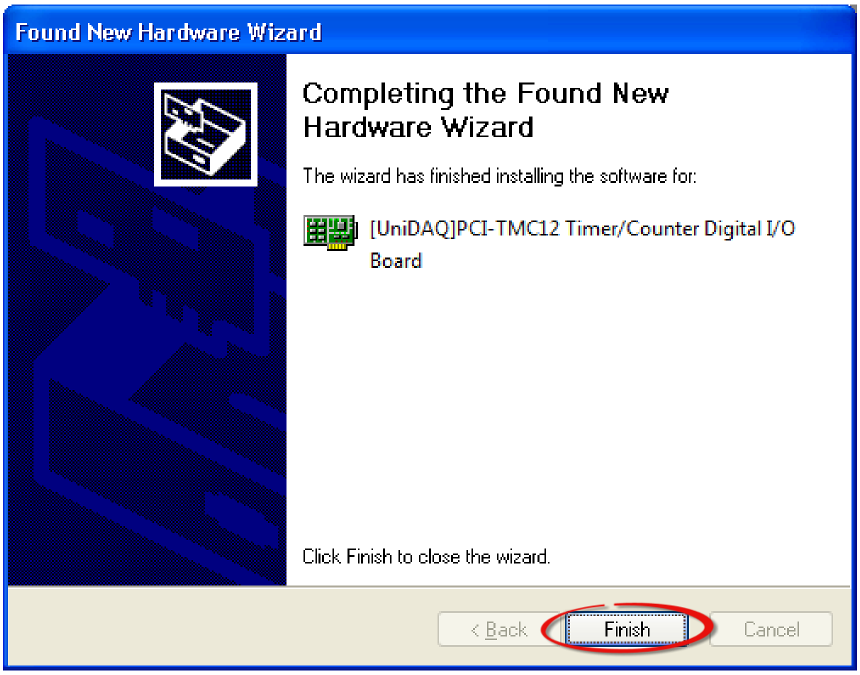
If your hardware came with an installation CD or floppy disk, insert it now.

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.

Step 4: Click the **"Finish"** button.



Found New Hardware Wizard

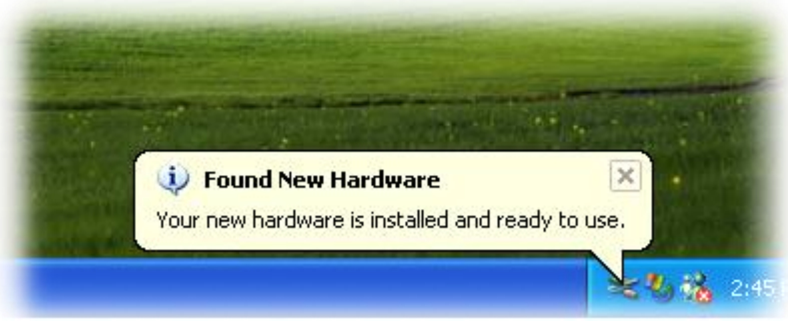
Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

[UniDAQ]PCI-TMC12 Timer/Counter Digital I/O Board

Click Finish to close the wizard.

Step 5: Windows pops up **"Found New Hardware"** dialog box again.



Found New Hardware

Your new hardware is installed and ready to use.

4.3 Verifying the Installation

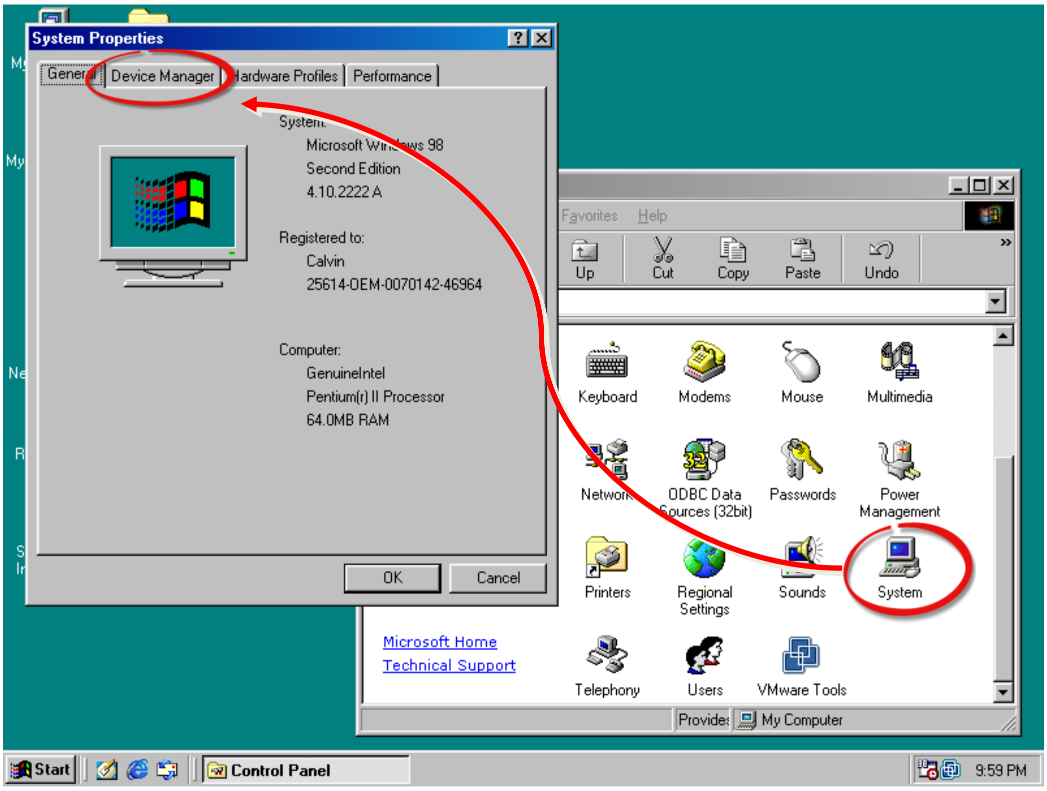
To verify that the driver was correctly installed, use the Windows **Device Manager** to view and update the device drivers installed on the computer, and to ensure that the hardware is operating correctly. The following is a description of how access the Device Manager in each of the major versions of Windows. Refer to the appropriate description for the specific operating system to verify the installation.

4.3.1 Accessing Windows Device Manager

■ Windows 95/98/ME

Step 1: Either right-click the **"My Computer"** icon on the desktop and then click **"Properties"**, or open the **"Control Panel"** and double-click the **"System"** icon to open the System Properties dialog box.

Step 2: In the **System Properties** dialog box, click the "**Device Manager**" tab.



System:
Microsoft Windows 98
Second Edition
4.10.2222 A

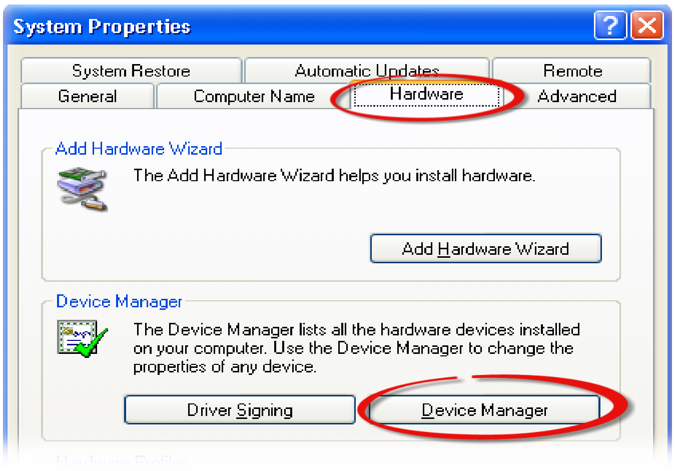
Registered to:
Calvin
25614-OEM-0070142-46964

Computer:
GenuineIntel
Pentium(r) II Processor
64.0MB RAM

■ Windows 2000/XP

Step 1: Click the "**Start"** button and then point to "**Settings" and click** "**Control Panel**". Double-click the "**System**" icon to open the "**System Properties"** dialog box.

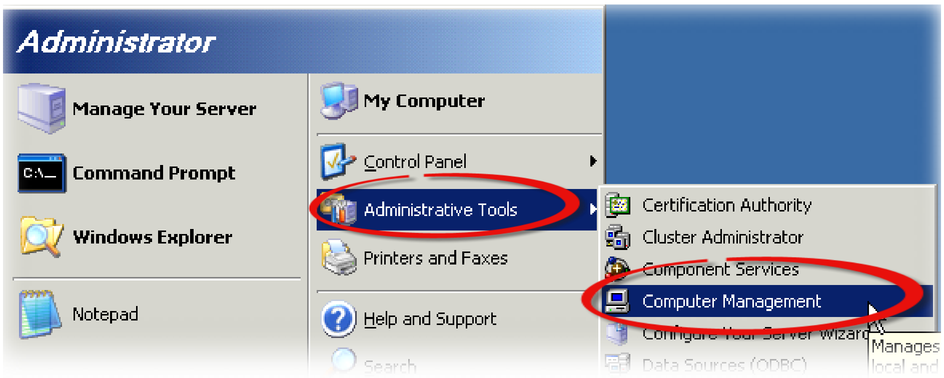
Step 2: Click the "**Hardware**" tab and then click the "**Device Manager**" button.


System Properties
System Restore
Automatic Updates
Remote
General
Computer Name
Hardware
Advanced
Add Hardware Wizard
The Add Hardware Wizard helps you install hardware.
Add Hardware Wizard
Device Manager
The Device Manager lists all the hardware devices installed on your computer. Use the Device Manager to change the properties of any device.
Driver Signing
Device Manager

■ Windows Server 2003

Step 1: Click the **"Start"** button and point to **"Administrative Tools"**, and then click the **"Computer Management"** option.

Step 2: Expand the **"System Tools"** item in the console tree, and then click **"Device Manager"**.



Windows 7

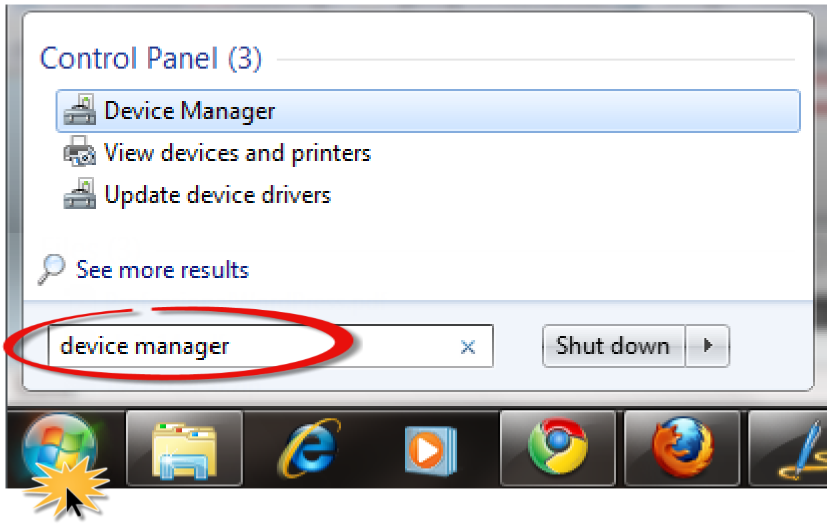
Step 1: Click the **"Start"** button, and then click **"Control Panel"**.

Step 2: Click **"System and Maintenance"**, and then click **"Device Manager"**.

Alternatively,

Step 1: Click the **"Start"** button.

Step 2: In the **Search field,** type **Device Manager** and then press Enter.



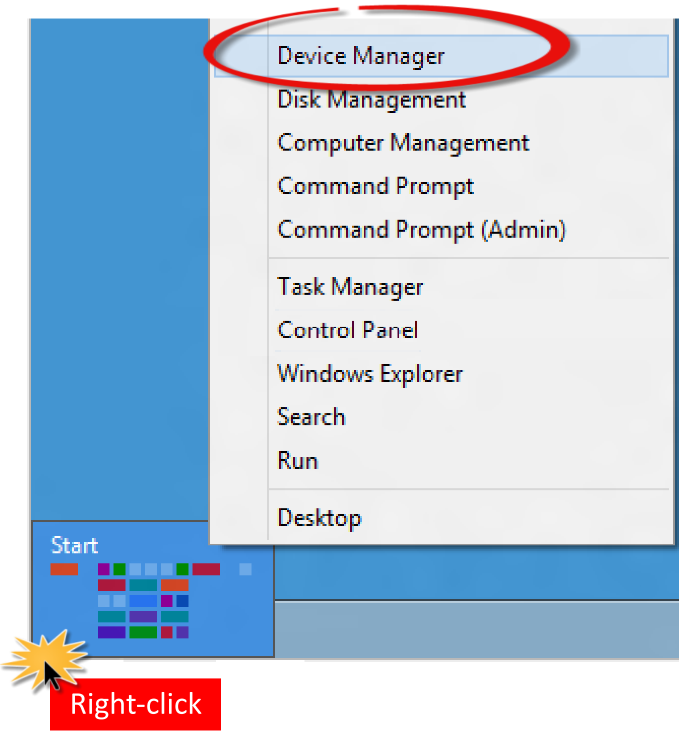
Note that Administrator privileges are required for this operation. If you are prompted for an administrator password or confirmation, enter the password or provide confirmation by clicking the "Yes" button in the User Account Control message.

Windows 8

Step 1: To display the **Start screen icon** from the desktop view, hover the mouse cursor over the **bottom-left corner** of screen.

Step 2: **Right-click** the Start screen icon and then click "**Device Manager**".

Alternatively, press **[Windows Key] +[X]** to open the Start Menu, and then select Device Manager from the options list.



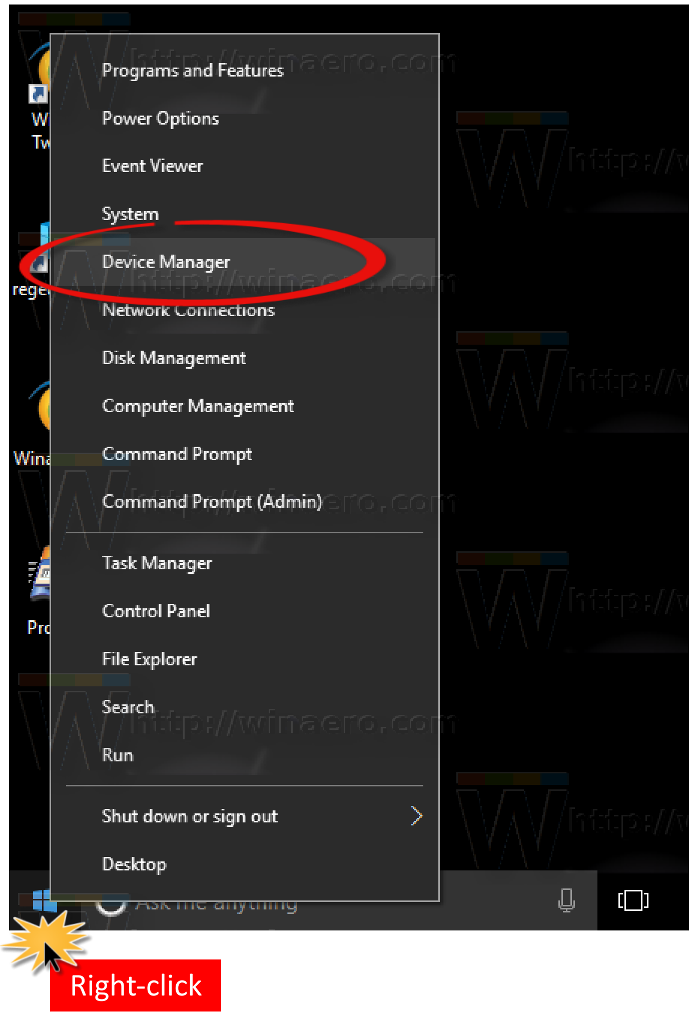
Windows 10

Step 1: Press **[Windows Key] +[X]** shortcut keys together

or

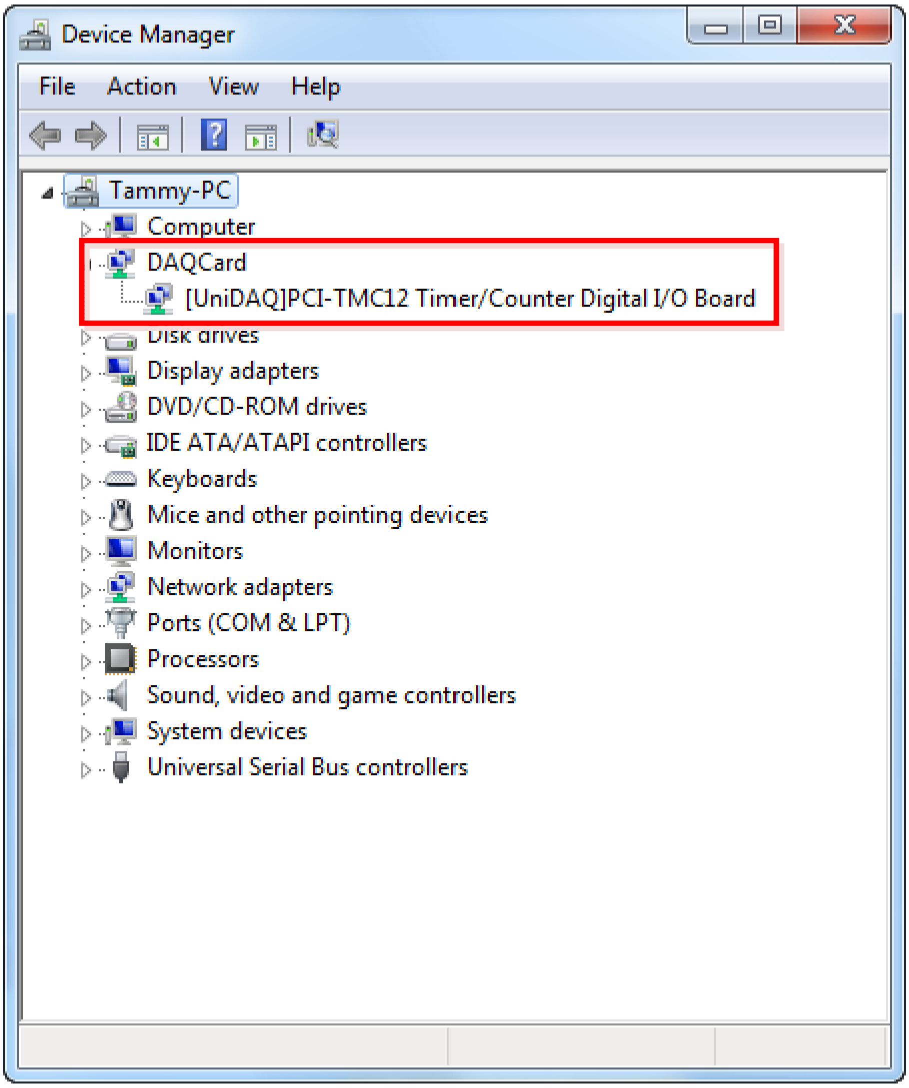
right-click or press and hold on the **Start** ⊞ button, then select **Device Manager** from the context menu.

Step 2: Select the Device Manager item:



4.3.2 Check the Installation

Check that the PCI-TMC12 Series board is correctly listed in the Device Manager, as illustrated below.



Tammy-PC
- Computer
- DAQCard
  - [UniDAQ]PCI-TMC12 Timer/Counter Digital I/O Board
- Disk drives
- Display adapters
- DVD/CD-ROM drives
- IDE ATA/ATAPI controllers
- Keyboards
- Mice and other pointing devices
- Monitors
- Network adapters
- Ports (COM & LPT)
- Processors
- Sound, video and game controllers
- System devices
- Universal Serial Bus controllers

5. Testing the PCI-TMC12 Series Board

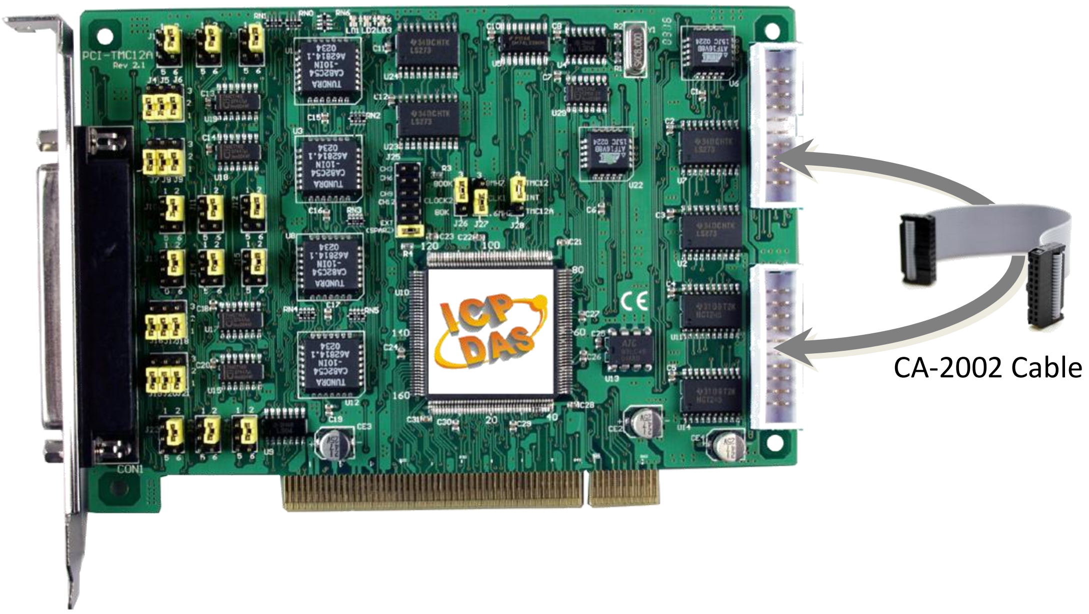
This chapter provides detailed information about the "Self-Test" process, which is used to confirm that the PCI-TMC12 Series board is operating correctly. Before beginning the "Self-Test" process, ensure that both the hardware and driver installation procedures are fully completed. For detailed information about the hardware and driver installation, refer to Chapter 3 Hardware Installation and Chapter 4 Software Installation.

5.1 Self-Test Wiring

The following is a description of how to configure the wiring in order to perform the "Self-Test" procedures for the Digital Input or/and Digital Output.

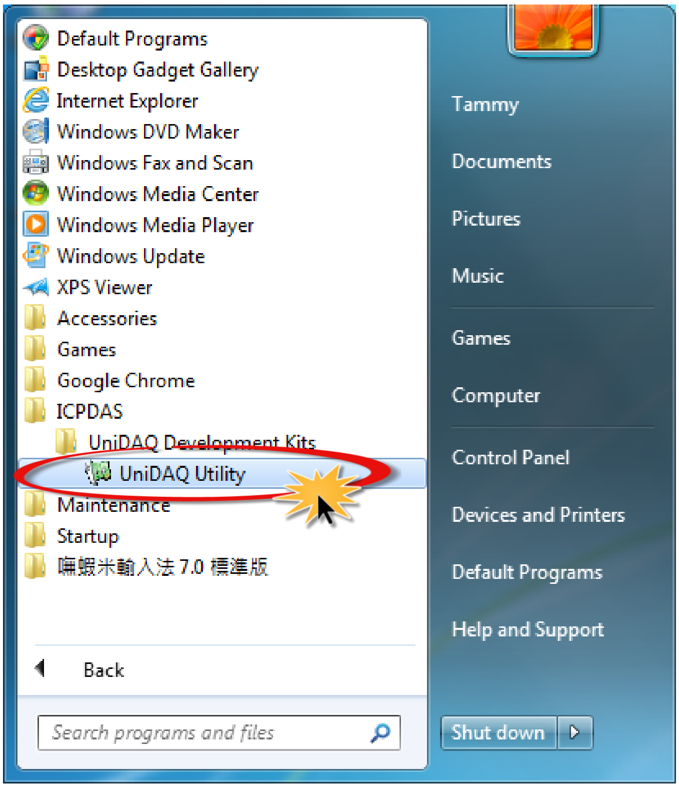
Before beginning the "Self-Test" procedure, ensure that the following items are available:
☑ A CA-2002 Cable
(Optional, Website: http://www.icpdas.com/products/Accessories/cable/cable_selection.htm)

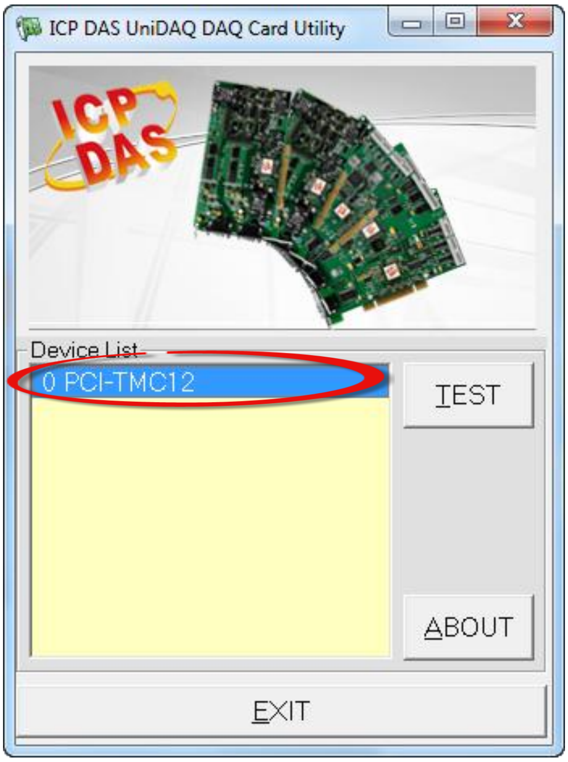
Step 1: Use CA-2002 cable to connect the CON2 with CON3.



5.2 Execute the Test Program

Step 1: In Windows 7, click the “**Start”** button, point to **“All Programs”,** and then click the **“ICPDAS”** folder. Point to **“UniDAQ Development Kits”**and then click the **“UniDAQ Utility**” to execute the UniDAQ Utility Program.





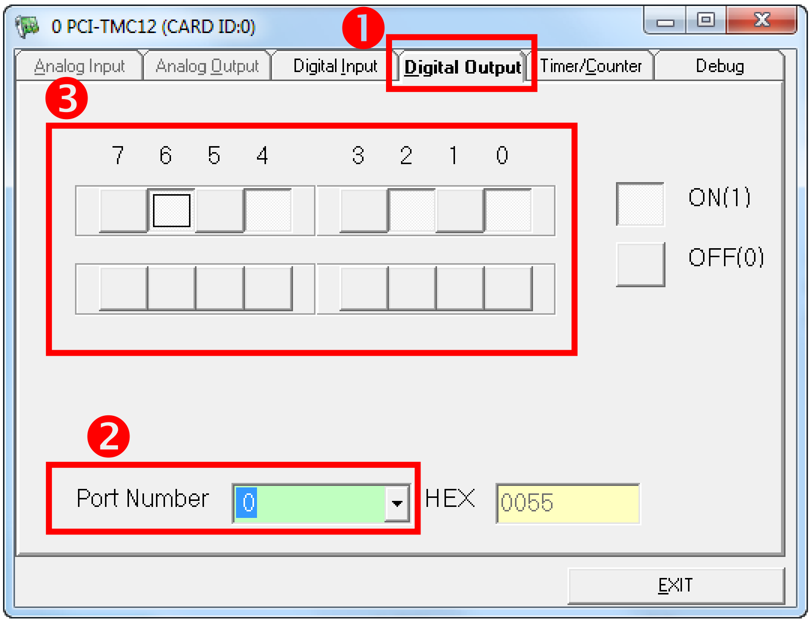
0 PCI-TMC12

Step 2: Confirm that your board has been successfully installed in the Host system. Note that the device numbers start from 0.

Step 3: Click the **“TEST”** button to start the test.

Step 4: Check the results of the **Digital Input/Output** functions test.

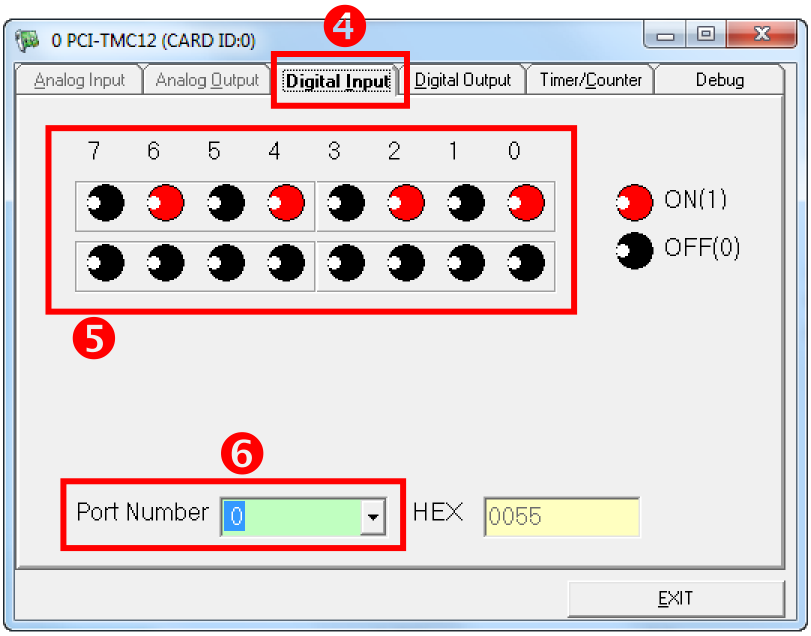
1. Click the **"Digital Output"** tab.
2. Select **"Port0"** from the **"Port Number"** drop-down menu.
3. Check the checkboxes for **channels 0, 2, 4 and 6**.



Port Number 0

HEX 0055

4. Click the **"Digital Input"** tab.
5. Select **"Port0"** from the **"Port Number"** drop-down menu.
6. The DI indicators will turn **red** when the corresponding DO channels 0, 2, 4 and 6 are **ON**.



Port Number 0

HEX 0055

6. I/O Control Register

6.1 Determining the I/O Address

The Plug and Play BIOS will assign an appropriate I/O address for each PCI-TMC12 series card during the power-on stage. The PCI-TMC12 series cards include four fixed ID numbers, which are indicated below:

Table 6-1:

Model	PCI-TMC12	PCI-TMC12A	PCI-TMC12AU
Vendor ID	0x10B5		
Device ID	0x9050		
Sub Vendor ID	0x2129		
Sub Device ID	0x9912		

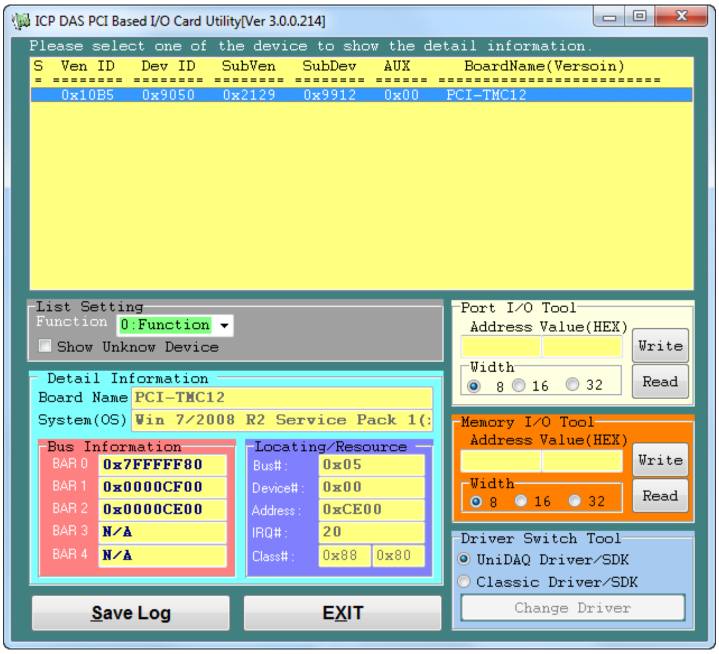
PIO_PISO.EXE Utility for the Windows

The PIO_PISO.EXE utility program will detect and present all information for ICPDAS I/O boards installed in the PC, as shown in the following figure. Details of how to identify the PCI-TMC12 Series board of ICPDAS data acquisition boards based on the **Vendor, Device, Sub-vendor and Sub-device ID** are given in Table 6-1.

The **PIO_PISO.exe** utility is located on the CD as below and is useful for all PISO-DIO series boards.

CD:\NAPDOS\PCI\Utility\Win32\PIO_PISO\

http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/utility/win32/pio_piso/



ICP DAS PCI Based I/O Card Utility[Ver 3.0.0.214]

```
Please select one of the device to show the detail information.
S  Ven ID    Dev ID    SubVen    SubDev    AUX     BoardName(Versoin)
= ======== ======== ======== ======== ====== =========================
   0x10B5    0x9050    0x2129    0x9912    0x00   PCI-TMC12
```

List Setting
Function: 0:Function
□ Show Unknow Device

Detail Information
Board Name: PCI-TMC12
System(OS): Win 7/2008 R2 Service Pack 1(:

Bus Information
BAR 0: 0x7FFFFF80
BAR 1: 0x0000CF00
BAR 2: 0x0000CE00
BAR 3: N/A
BAR 4: N/A

Locating/Resource
Bus#: 0x05
Device#: 0x00
Address: 0xCE00
IRQ#: 20
Class#: 0x88 0x80

Port I/O Tool
Address Value(HEX)
Width: ● 8 ○ 16 ○ 32

Memory I/O Tool
Address Value(HEX)
Width: ● 8 ○ 16 ○ 32

Driver Switch Tool
● UniDAQ Driver/SDK
○ Classic Driver/SDK

6.2 The Assignment of I/O Address

When the system is started (booted), the Plug and Play BIOS will assign an appropriate I/O address for the PCI-TMC12 series card. If only one PCI-TMC12 series card is installed in the system, it will be identified as **card_0**. However, if two or more PCI-TMC12 series cards are installed, identifying which card is **card_0** becomes more difficult. The software driver can support a maximum of 16 boards.

The easiest way to determine the number of the card is to use the **DEM10.EXE** file provided in the **DOS Demo Programs folder (CD:\NAPDOS\PCI\PCI-TMC12A\DOS\)**. This demo program can be used to send a value to the Digital Output and read the details back from the Digital Input. If a 20-pin flat cable is installed between CON2 and CON3, the value read from the Digital Input will be the same as that from the Digital Output. The procedure for determining the board number as follows:

Step 1: Remove all 20-pin flat cables from between CON2 and CON3.

Step 2: Install all PCI-TMC12 Series card into the Host system.

Step 3: Power-on the system and execute the DEMO10.exe file.

Step 4: Check all the DI value displayed. In a normal situation, the DI values should be different from the DO value, since.

Step 5: Install a 20-pin flat cable between the CON2 and CON3 connector of any PCI-TMC12 series card.

Step 6: Check which card has the same vale for both the DI and DO, and then record the number of the board. This the board number for the card where the cable is connected.

Consequently, the card number can be determined very easily if a 20-pin flat cable is individually installed into each PCI-TMC12 series card following the procedure described above.

6.3 I/O Address Mapping

The I/O address for the PCI-TMC12 series card is automatically assigned by the ROM BIOS on the PC and provides Plug and Play capabilities for the PCI-TMC12 series card.

An overview of the I/O address mapping for the PCI-TMC12 series card is shown below:

Address	Read	Write
wBase+0	Active 8254 Counter 0	Active 8254 Counter 0
wBase+4	Active 8254 Counter 1	Active 8254 Counter 1
wBase+8	Active 8254 Counter 2	Active 8254 Counter 2
wBase+0x0C	Active 8254 Control Word	Active 8254 Control Word
wBase+0x10	Reserved	Select the Active 8254 Chip
wBase+0x14	Digital Input Channels 0 ~ 15	Digital Output Channels 0 ~ 15
wBase+0x18	New Control of PCI-TMC12A	Interrupt clear of PCI-TMC12A
wBase+0x1C	Read DO Readback	Reserved
wBase+0x3C	Read Card ID	Reserved

Note: wBase: The base address of the control word for the PCI-TMC12/TMC12A card.

6.3.1 Activating an 8254 chip

The PCI-TMC12 series card contains four 8254 chips that operate simultaneously, but only one of these chips can be activated for reading or writing at any one time. Before accessing a specific 8254 chip, first use “wBase+0x10” to set it as the active chip.

(Write) wBase+0x10: Sets the active 8254 chip

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
x	x	x	x	x	x	D1	D0

D0=0, D1=0: 8254 chip-1 is active
D0=1, D1=0: 8254 chip-2 is active
D0=0, D1=1: 8254 chip-3 is active
D0=1, D1=1: 8254 chip-4 is active

Example:

```
outportb(wBase+0x10,0);      /* select the 8254 chip-1, CNT1 ~CNT3 */
outportb(wBase+0x10,2);      /* select the 8254 chip-3 , CNT10 ~ CNT12 */
```

6.3.2 8254 Timer/Counter Control

The PCI-TMC12 series card contains four 8254 chips that operate simultaneously, but only one of the chips can be activated for reading or writing any one time. Before accessing a specific 8254 chip, first use “wBase+0x10” to set it as the active chip. (See Section 6.3.1 for more details)

The 8254 chip contains four registers, addressed from wBase+0 to wBase+0x0C, and these can be accessed in order to control and monitor the Timer/Counter on the active 8254 chip. For more information, refer to Chapter 7 “Programming the Intel 8254” of this manual and the Intel 82C54 datasheet.

Address	Read	Write
wBase+0	Active 8254 Counter 0	Active 8254 Counter 0
wBase+4	Active 8254 Counter 1	Active 8254 Counter 1
wBase+8	Active 8254 Counter 2	Active 8254 Counter 2
wBase+0x0C	Active 8254 Control Word	Active 8254 Control Word

6.3.3 Digital Input

(Read) wBase+0x14: Reads the status of Digital Input channels 0 to 15

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
DI15	DI14	DI13	DI12	DI11	DI10	DI9	DI8

Example:

```
wDiValue=inport(wBase+0x14);     /* Reads the status of the DI channel */
```

6.3.4 Digital Output

(Write) wBase+0x14: Sets the Digital Output channels 0 to 15

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DO 7	DO 6	DO 5	DO 4	DO 3	DO 2	DO 1	DO0

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
DO 15	DO 14	DO 13	DO 12	DO 11	DO 10	DO 9	DO 8

Example:

```
outport(wBase+0x14,wDoValue);   /* Controls the status of the DO channel */
```

6.3.5 Interrupt Control/Status Register of PCI-TMC12A(U)

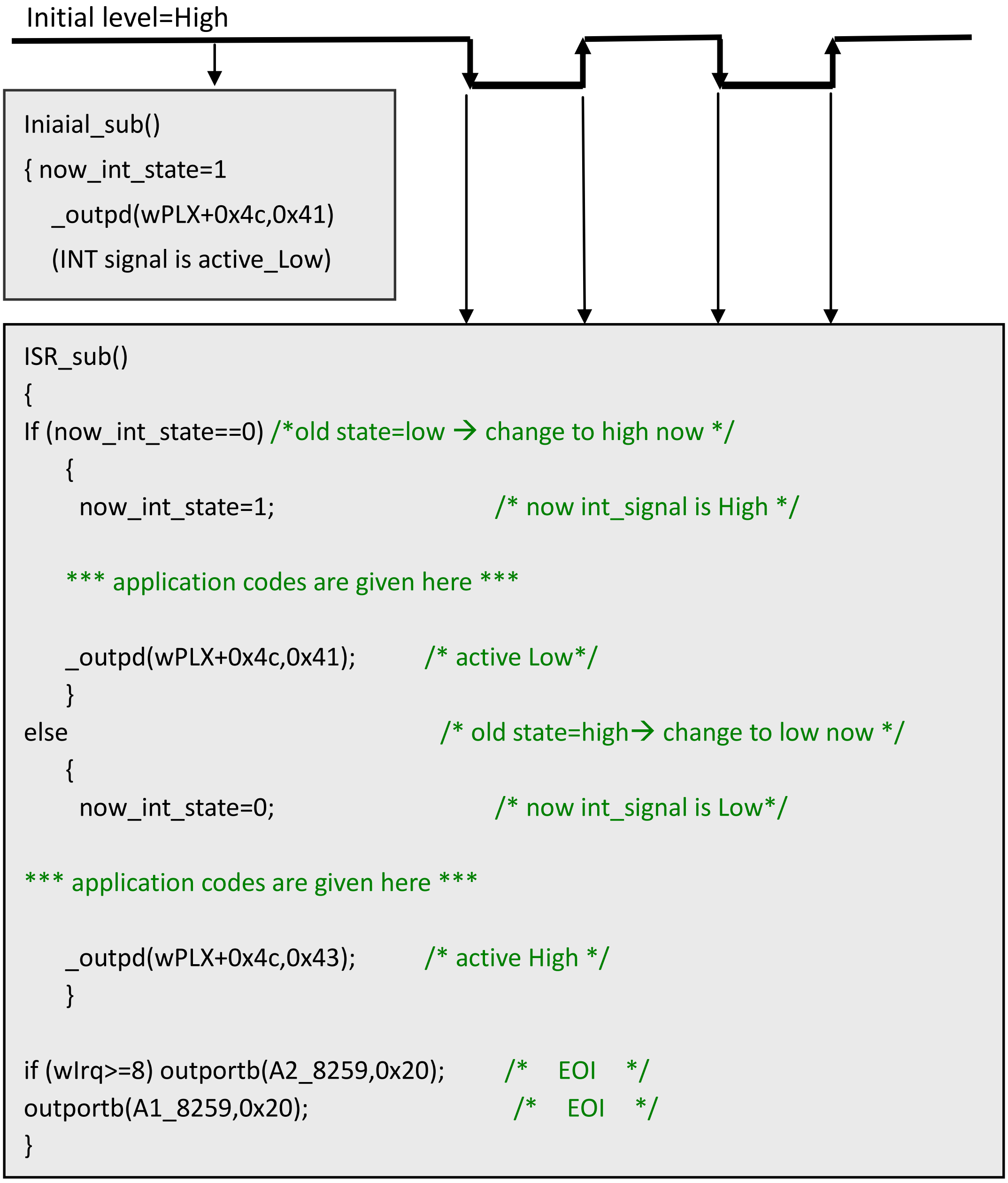
(Read/Write) wPLX+0x4C: interrupt control/status register

Bit	Description
B0	Enable/Disable Interrupt 0 = Disable, 1 = Enable
B1	POLARITY 0 = Active Low, 1 = Active High
B2	Interrupt Status 0 = int not Active, 1 = int is Active
B3	Reserved
B4	Reserved
B5	Reserved
B6	Enable/Disable PCI Interrupt 0 = Disable, 1 = Enable
B7	Software Interrupt. A value of 1 will generate an interrupt
B8 to B31	Reserved

The interrupt for the PCI-TMC12/TMC12A is a **level-trigger**, and the interrupt signal can be programmed as either **active-Low** or **active-High**. The procedure for programming the interrupt for the PCI-TMC12/TMC12A is as follows:

1. Determine whether the initial level is set to High or Low
2. If the initial state is set to **High** → then set the interrupt signal to be initially **active_Low**
3. If the initial state is set to **Low** → then set the interrupt signal to be initially **active_High**
4. If the interrupt signal is active → the program will transfer to the interrupt service routine (ISR) → and **toggle the active_state before returning from the ISR**.

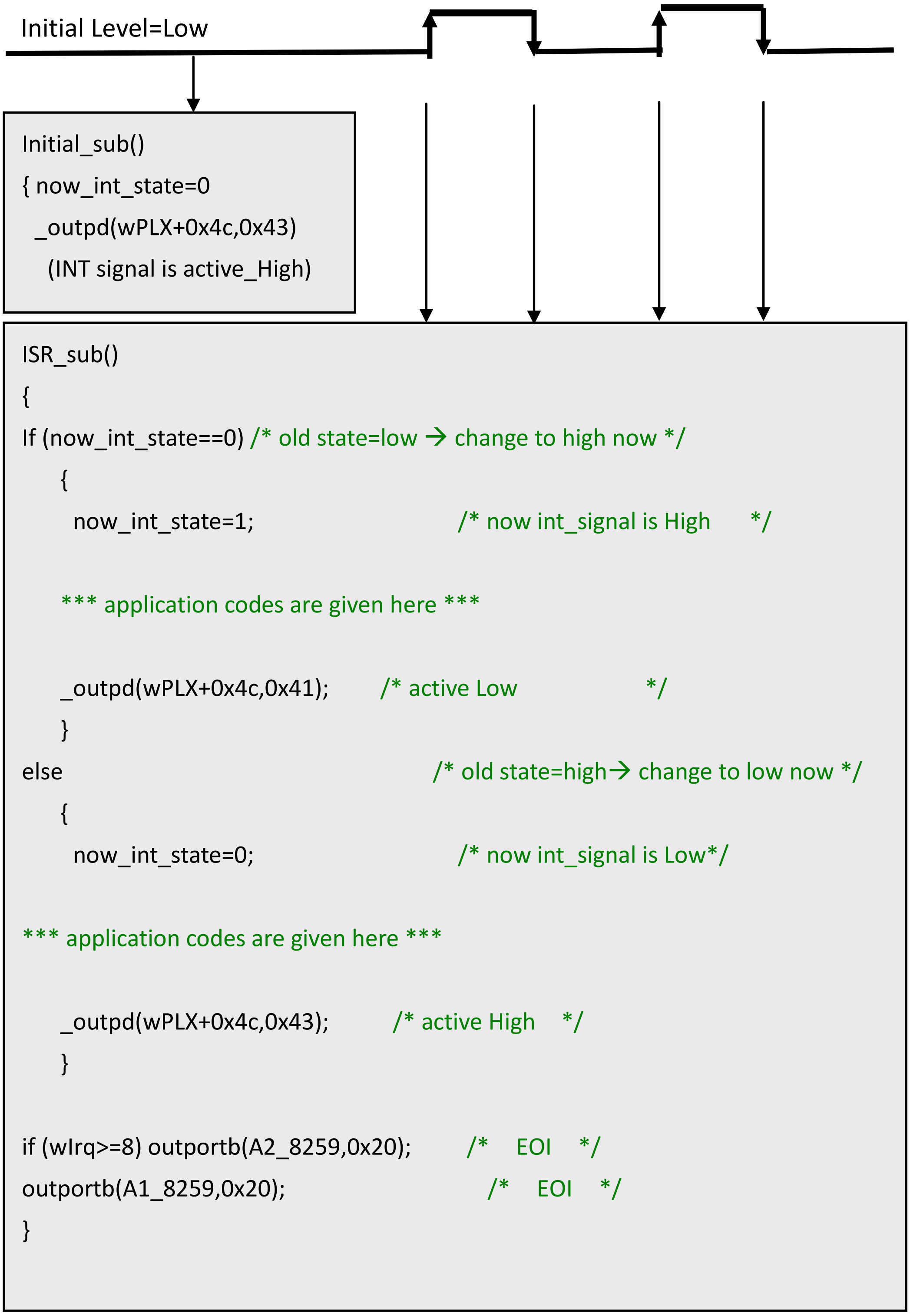
Example 1: Assume initial level=High



```
Iniaial_sub()
{ now_int_state=1
   _outpd(wPLX+0x4c,0x41)
   (INT signal is active_Low)
```

```
ISR_sub()
{
If (now_int_state==0) /*old state=low → change to high now */
    {
     now_int_state=1;                    /* now int_signal is High */

    *** application codes are given here ***

    _outpd(wPLX+0x4c,0x41);        /* active Low*/
    }
else                                  /* old state=high→ change to low now */
    {
     now_int_state=0;                    /* now int_signal is Low*/

*** application codes are given here ***

    _outpd(wPLX+0x4c,0x43);        /* active High */
    }

if (wIrq>=8) outportb(A2_8259,0x20);        /*    EOI    */
outportb(A1_8259,0x20);                      /*    EOI    */
}
```

Example 2: Assume initial level=Low



```
Initial_sub()
{ now_int_state=0
  _outpd(wPLX+0x4c,0x43)
    (INT signal is active_High)
```

```
ISR_sub()
{
If (now_int_state==0) /* old state=low → change to high now */
      {
        now_int_state=1;                                /* now int_signal is High      */

      *** application codes are given here ***

      _outpd(wPLX+0x4c,0x41);        /* active Low                    */
      }
else                                                  /* old state=high→ change to low now */
      {
        now_int_state=0;                                /* now int_signal is Low*/

*** application codes are given here ***

      _outpd(wPLX+0x4c,0x43);          /* active High    */
      }

if (wIrq>=8) outportb(A2_8259,0x20);         /*    EOI    */
outportb(A1_8259,0x20);                              /*    EOI    */
}
```

The ISR_sub function will be active on the **rising edge and the falling edge** of the interrupt signal. Refer to the demo7.c, demo11.c, demo12.c and demo13.c files for more information.

6.3.6 DO Readback Register

(Read) wBase+0x1C: Reads the Digital Output value

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DO 7	DO 6	DO 5	DO 4	DO 3	DO 2	DO 1	DO0

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
DO 15	DO 14	DO 13	DO 12	DO 11	DO 10	DO 9	DO 8

Example:

```
DigitalIn=inportb(BaseAddr+0x1C);          /* Reads DO Readback */
```

6.3.7 Card ID Register

(Read) wBase+0x3C: Reads card ID (SW1)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	DI3	DI2	DI1	DI0

Example:

```
wCardID=inport(wBase+0x3C);      /* Reads card ID*/
```

6.4 PCI-TMC12A Features

6.4.1 Default Shipping Settings for the PCI-TMC12A(U)

When shipped, the **J28** jumper (See Section 2.4.1) on the PCI-TMC12 series card is set so that the module will operate in PCI-TMC12 series mode by default, meaning that the module is equivalent to the PCI-TMC12 series and the interrupt system is compatible with the PCI-TMC12 series. Refer to Section 6.4.4 for the interrupt block diagrams for both the PCI-TMC12 series card.

All Xor? control registers for the PCI-TMC12A(U) card are cleared to their Low states during the initial power up stage, so all clock sources on the PCI-TMC12A(U) are compatible with those of the PCI-TMC12. Refer to Section 6.4.2 for block diagram.

In general, any PCI-TMC12AU card can be installed and used as a PCI-TMC12A. All original programs designed for a PCI-TMC12A card can be executed using a PCI-TMC12AU without requiring any modification to the code.

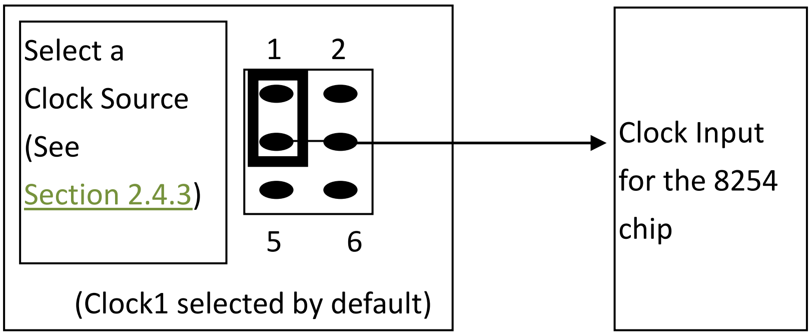
The most important aspect to remember is that the PCI-TMC12AU is equivalent to the PCI-TMC12A by default when shipped.

The following is an overview of the new features of the PCI-TMC12A(U) board:

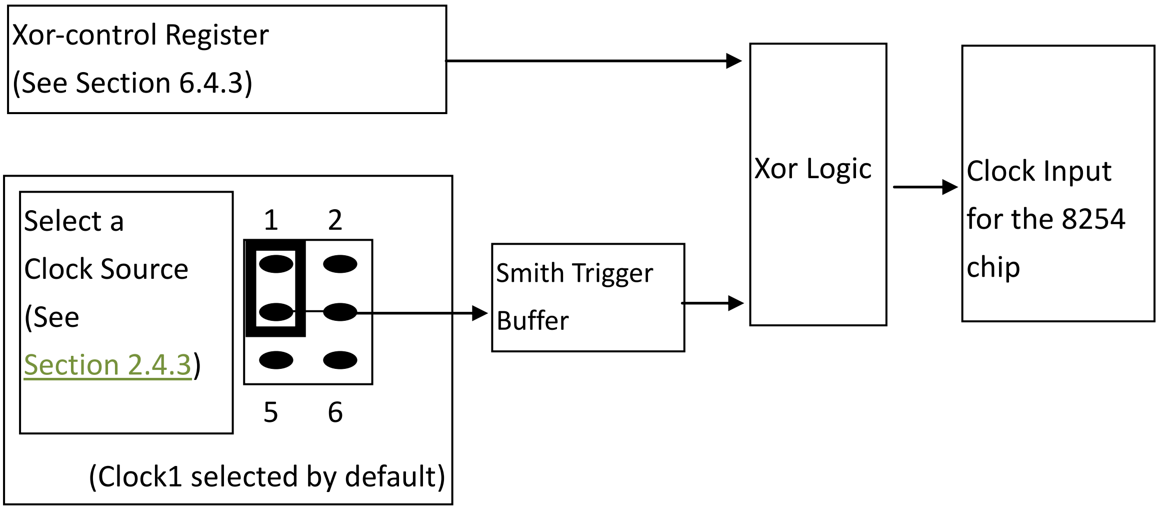
- New interrupt mechanism (See Section 6.4.4)
- Xor? bits for generation of the two clocks (See Section 6.4.2)
- Three LEDs as status indicators (See Section 6.4.3 and Section 2.1)
- Equipped with a smith trigger buffer for the selected clock source (See Section 6.4.2)
- An additional **DO port, wBase+0x18,** for the Xor-bits, XorInt and LED ON/OFF control. Refer to Section 6.4.3 for more information.
- An additional **DI port, wBase+0x18**, used to enable/disable the interrupt. The initial routine and the ISR must be input from the wBase+0x18 to enable the next interrupt operation. Refer to Section 6.4.4 for more information.
- Refer to the new demo programs described in Section 6.4.5 for more details of how to use the new features
- Refer to Section 2.1 for a diagram of the PCB layout for the PCI-TMC12A(U) card

6.4.2 Clock Input for the 8254 Chip

- The clock input for the 8254 chips on the PCI-TMC12 card is as follows:



- The clock input for the 8254 chips on the PCI-TMC12A(U) card is as follows:



The following is an overview of the new features of the PCI-TMC12A(U) board:

- A smith trigger buffer has been added to remove noise in the selected clock source
- An Xor-control register has been added to invert/non-invert the selected clock source. This mechanism can be used to generate two additional starting clocks for the 8254 chip.

Note: All Xor-control registers are cleared to 0 when the PCI-TMC12A(U) card is first powered-up, so the initial state for the PCI-TMC12A(U) is compatible with the PCI-TMC12.

As referred to in Section 8.1.15 Ndemo2: Generating Two Staring Clocks, the twelve Xor-bits are used to generate the two starting clocks. So the initial value for the 8254 chip can be verified after the two starting clocks are generated. They are then used to generate one single clock for testing purposes. In general, however, these Xor-bits are only designed for the generation of the two starting clocks.

6.4.3 Xor-control Register for the PCI-TMC12A(U)

(Write) wBase+18: Sets the Xor-control register

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Xor8	Xor7	Xor6	Xor5	Xor4	Xor3	Xor2	Xor1

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
LED3	LED2	LED1	XorInt	Xor12	Xor11	Xor10	Xor9

Note: All bits for this register will be cleared to zero during the power-up stage.

Xor1 → invert/non-invert the clock source selected for CLK1

Xor2 → invert/non-invert the clock source selected for CLK2

Xor11 → invert/non-invert the clock source selected for CLK11

Xor12→ invert/non-invert the clock source selected for CLK12

Xor?=0 → non-invert, i.e., the power-up value

Xor?=1 → invert

XorInt → invert/non-invert the selected interrupt source

Led1 → Led1=0 → Turn LED1 ON, Led1=1 → turn LED1 OFF

Led2 → Led2=0 → Turn LED2 ON, Led2=1 → turn LED2 OFF

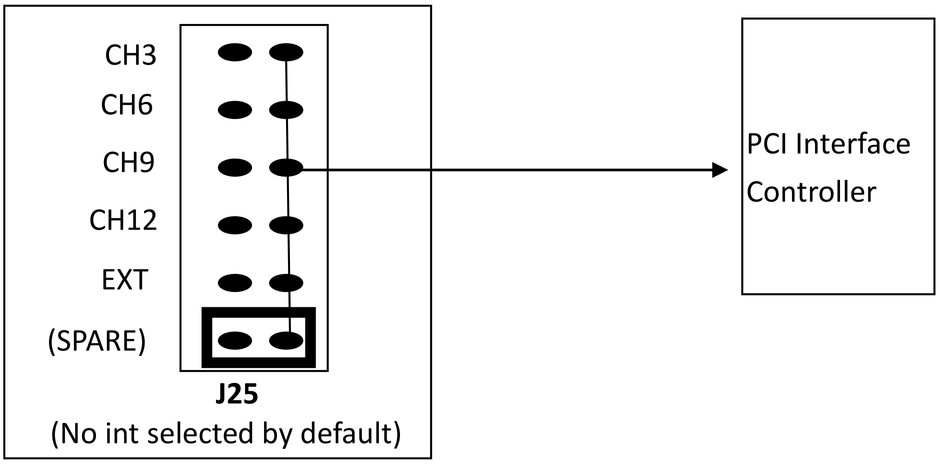
Led3 → Led3=0 → Turn LED3 ON, Led3=1 → turn LED3 OFF

- Xor? is designed to generate the two starting clocks for the 8254 chip
- XorInt is used to invert/non-invert the interrupt source to a Low state, i.e., if the initial state for the interrupt source is set to High, this bit will be set to High to invert it to the Low state. Refer to Section 8.1. 18 Ndemo5: Implementing an Active Low Interrupt for demo program.
- When the PCI-TMC12A card is first powered-up, the initial values are all set to zero, meaning that LED1/2/3 are all set to ON. LED1/2/3 are designed as status indicators, and can be assigned to any purpose depending on specific requirements.

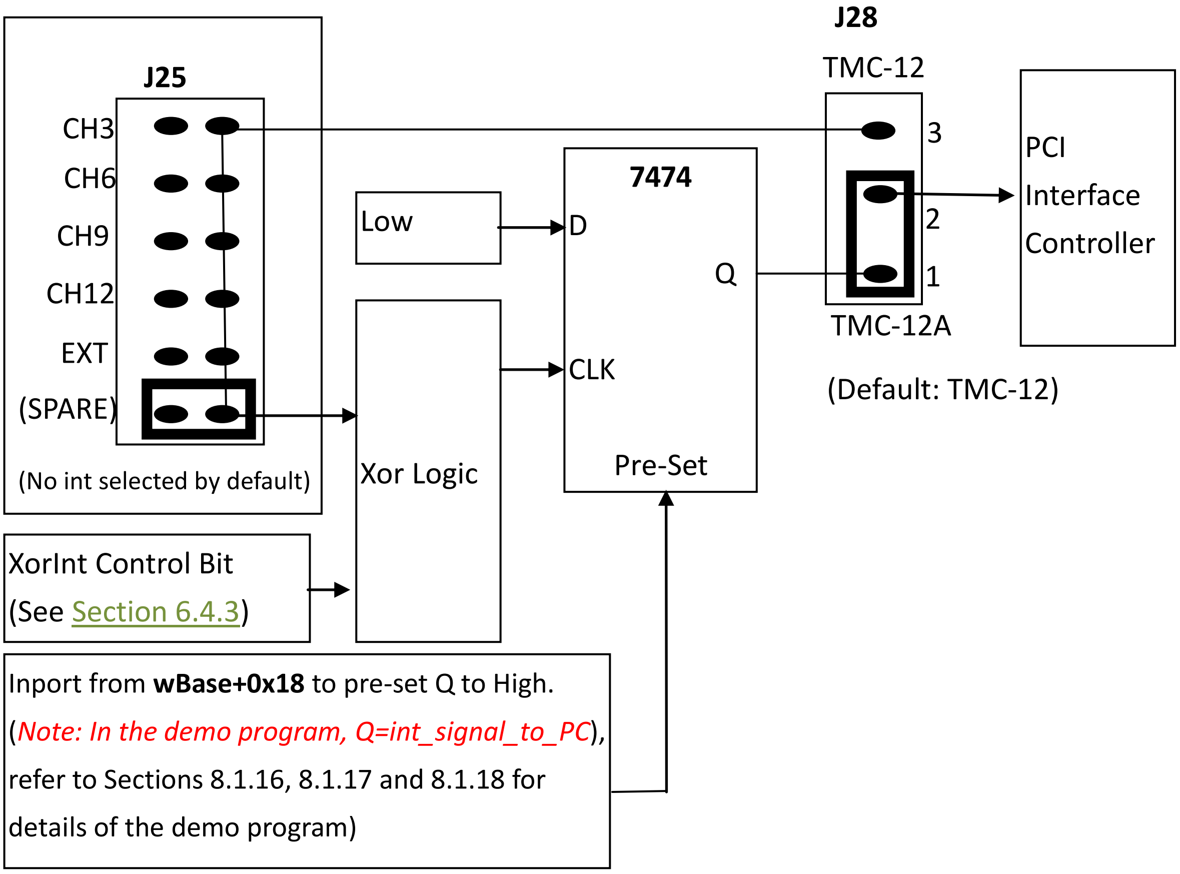
As referred to Section 8.1.15 Ndemo2: Generating Two Starting Clocks, the twelve Xor-bits are used to generate the two starting clocks. So the initial value for the 8254 can be verified after the two starting clocks are generated. They are then used to generate single clock for testing purposes. In general, however, these Xor-bits are only designed for the generation of the two starting clocks.

6.4.4 Block Diagram for the Interrupt System

The block diagram for the interrupt system (J25) on the PCI-TMC12 is as follows:



- The block diagram for the interrupt system (J25) on the PCI-TMC12A(U) is as follows:



Inport from **wBase+0x18** to pre-set Q to High. (*Note: In the demo program, Q=int_signal_to_PC*), refer to Sections 8.1.16, 8.1.17 and 8.1.18 for details of the demo program)

The interrupt mechanism on the PCI-TMC12 can be set to either active Low or active High, and the interrupt system of the PCI bus is level triggered. Consequently, the Windows driver for the PCI-TMC12 must create a thread to handle all the conditions where the interrupt is active. As there are so many possible conditions, the interrupt performance will be greatly reduced.

The new interrupt mechanism on the PCI-TMC12A(U) is designed to improve the performance of the Windows driver in the following manner:

- The initial subroutine and ISR will inport from wBase+0x18 to pre-set the int_signal_to_PC value (shown as Q in the block diagram above) to High state to enable the next interrupt operation
- If the initial value of the interrupt source is set to Low, XorInt will be set to 0, i.e., the rising-edge of the interrupt
- If the initial value of the interrupt source is set to High, XorInt will be set to 1, i.e., the falling-edge of the interrupt
- The software driver is designed for rising-edge or falling-edge interrupts

When the interrupt ISR is executed, the int_signal_to_PC value (shown as Q in the block diagram above) is set to the Low state, so the interrupt ISR must inport from wBase+0x18 to pre-set the int_signal_to_PC value to the High state to enable the next interrupt operation. Refer to Section 8.1.16, Section 8.1.17 and Section 8.1.18 for details of the demo programs.

6.4.5 New Demo Program

- New Demo Program 1, How to use the Status Indicator LEDs.
  (Refer to Section 8.1.14 Ndemo1: Using the LEDs)

- New Demo Program 2, How to Generate the two Starting Clocks for 8254 chips.
  (Refer to Section 8.1.15 Ndemo2: Generating Two Starting Clocks)

- New Demo Program 3, Modifying Demo Program7 (designed for the PCI-TMC12 card) to match the new interrupt mechanism on the PCI-TMC12A(U))
  (Refer to Section 8.1.16 Ndemo3: Implementing a watchdog Timer (Modified version of Demo7))

- New Demo Program 4, Interrupt source = initial High, active low
  (Refer to Section 8.1.17 Ndemo4: Implementing an Active High Interrupt)

- New Demo Program 5, Interrupt source = initial Low, active High
  (Refer to Section 8.1.18 Ndemo5: Implementing an Active Low Interrupt)

7. Programming the Intel 8254

The Intel 8254 is a Timer/Counter device designed to solve common timing control problems in computer system design. The device provides three independent 16-bit counters, each capable of handling clock inputs up to 10 MHz and all modes are software programmable.

For more detailed information related to 8254 programming, refer to the relevant Intel documentation.

7.1 Control Word Format

After power-up, the state of the 8254 is undefined, as well as the Mode, count value, and output of all Counters. How each Counter operates is determined when it is programmed, and each Counter must be programmed before it can be used. Unused counters need not be programmed.

Counters are programmed by writing a Control Word and then setting an initial count value. The Control Word itself specifies which Counter is being programmed, and the format of the initial count is determined by the Control Word used.

The format of the Control Word is a follows:

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

➢ Selection Counter

SC1	SC0	Description
0	0	Selects Counter0
0	1	Selects Counter1
1	0	Selects Counter2
1	1	Read Back command

- Read/Write

RL1	RL0	Description
0	0	Counter Latch Command
0	1	Reads/writes the LSB only
1	0	Reads/writes MSB only
1	1	Reads/writes the LSB first, then reads/writes the MSB

- Mode Selection

M2	M1	M0	Working Mode	Mode Type
0	0	0	Mode 0	Interrupt on Terminal Count
0	0	1	Mode 1	Hardware Retriggerable one-shot
x	1	0	Mode 2	Rate Generator
x	1	1	Mode 3	Square Wave Mode
1	0	0	Mode 4	Software Triggered Strobe
1	0	1	Mode 5	Hardware Triggered Strobe (Retriggerable)

Note: "Don't Care" bits (x) should be set to 0to ensure compatibility with other Inter products.

- BCD

BCD	Description
0	Binary Counter, 16-bits
1	Binary Coded Decimal (BCD) Counter (4 decades)

7.2 Counter Latch Command

Like a Control Word, this command is written to the Control Word Register. Also like a Control Word, the SC0, SC1 bits are used to select one of the three Counters, but two other bits, D5 and D4, distinguish this command from a Control Word.

The output latch of the selected Counter latches the count at the time the Counter Latch Command is received. This count is held in the latch until it is read by the CPU, or until the counter is reprogrammed.

The format for the Counter Latch Command is as follows:

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	0	0	x	x	x	x

Note: "Don't Care" bits (x) should be set to 0to ensure compatibility with other Inter products.

SC1	SC0	Description
0	**0**	Latches Counter 0
0	**1**	Latches Counter 1
1	**0**	Latches Counter 2
1	**1**	Read Back command

7.3 Read Back Command

This command allows the user to check the count value, programmed Mode, or current states related to the OUT pin and the Null Count flag for the selected counter(s).

The command applies to the counters that have been selected by setting their corresponding bits, D3, D2, D1 = 1, and is written into the Control Word Register.

The read-back command may be used to latch multiple counter output latches by setting the COUNT bit D5 to 0 and then selecting the desired counter(s).

The format is as follows:

D7	D6	D5	D4	D3	D2	D1	D0
1	1	/COUNT	/STATUS	CNT2	CNT1	CNT0	0

Dx	Value	Description
D5	0	Latches the counter value of the selected counters
D4	0	Latches the status of the selected counters
D3	1	Selects Counter2
D2	1	Selects Counter1
D1	1	Selects Counter0

7.4 Status Byte Format

The read-back command (see Section 7.3 above) may also be used to latch status information of selected counter(s) by setting STATSU bit D4 to 0. Note that the status muster be latched in order to be read. The status of a counter can be accessed by performing a read operation from the specific counter.

The counter status byte format is as follows:

D7	D6	D5	D4	D3	D2	D1	D0
COUT	Null Count	RW1	RW2	M2	M1	M0	BCD

Dx	Value	Description

D7	0
	1
		Cout = Low
		Cout = High

D6	0 1	Count available for reading Null Count
D5 to D0	Setting value read back	

The Null Count bit D6 indicates when the last count written to the counter register has been loaded into the counting element. The exact time this happens depends on the Mode of the counter and is described in the Mode Definitions, but until the count is loaded into the counting element, it cannot be read from the counter. If the count is latched or is read before this time, the count value will not reflect the new count value just written.

Note that if multiple status latch operations are performed on the counter(s) without reading the status of the counter at the time the first status read-back command was issued.

8. Demo Programs

ICP DAS provides a range of demo programs, together with the source code for the library, that can be used in either a DOS or a Windows environment using a variety of programming languages, including Turbo C (DOS), Delphi, Visual Basic, Visual C, VB.NET2005, and C#.NET2005 (Windows).

Detailed information about the demo programs is provided below.

Sample Program	UniDAQ SDK/Driver	PCI-TMC12 Series Class Driver	DOS
TC	-	-	✓
BC	-	-	-
MSC	-	-	-
Borland C^{++} Builder 4	-	-	-
Borland C^{++} Builder 6	✓	-	-
Delphi 4	-	✓	-
Delphi 6	✓	-	-
Visual Basic 6	✓	✓	-
Visual C^{++} 5	✓	✓	-
VB.NET 2005 (32-bit)	✓	✓	-
VB.NET 2005 (64-bit)	✓	-	-
C#.NET 2005 (32-bit)	✓	✓	-
C#.NET 2005 (64-bit)	✓	-	-
VC.NET 2005 (32-bit)	✓	-	-
VC.NET 2005 (64-bit)	✓	-	-
MATLAB	✓	-	-
LabVIEW	✓	✓	-

8.1 Demo Programs for DOS

Programming for the Intel 8254 chip can be very complicated, so, to help users easily solve real world problems, ICP DAS provides a range of demo programs that can be used in a DOS environment, together with the source code for the library.

The relevant DOS software and demo programs can be found in the:
CD:\NAPDOS\PCI\PCI-TMC12A\DOS\ folder on the companion CD, or can be downloaded from
http://ftp.icpdas.com/pub/cd/iocard/pci/napdos/pci/pci-tmc12a/dos/

An overview of the demo programs and library files for use with Turbo C 2.xx or above is provided below:

\TC*.*	→ for Turbo C 2.xx or above
\TC\LARGE\*.*	→ for large model
\TC\LARGE\LIB\*.*	→ for library source code
\TC\LARGE\DEMO?\*.*	→ demo program source code

\TC\LARGE\LIB\PCITMC12.H	→ library header file
\TC\LARGE\LIB\PCITMC12.C	→ library source file
\TC\LARGE\LIB\A.BAT	→ compiler file
\TC\LARGE\LIB\B.BAT	→ link file
\TC\LARGE\LIB\PCITMC12.lib	→ library file

\TC\LARGE\DEMO1\PCITMC12.H	→ library header file
\TC\LARGE\DEMO1\DEMO1.C	→ demo1 source file
\TC\LARGE\DEMO1\DEMO1.PRJ	→ TC project file
\TC\LARGE\DEMO1\IOPORTL.LIB	→ I/O port library file
\TC\LARGE\DEMO1\PCITMC12.LIB	→ library file
\TC\LARGE\DEMO1\DEMO1.EXE	→ demo1 execution file

8.1.1 Demo1: Using the Digital Output

```
/* -------------------------------------------------------------------- */
/* demo 1 : D/O demo                                                     */
/* step 1 : connect a DB-16R to CON3 of PCI-TMC12    */
/* step 2 : run DEMO1.EXE                                              */
/* step 3 : check the LEDs of DB-16R will turn on sequentially */
/* ------------------------------------------------------------------ */

#include "PCITMC12.H"

WORD pci_tmc12_do(WORD wDo);
WORD wBaseAddr, wIrq, wPLX;

int main()
{
int    i,j;
WORD wBoards,wRetVal;
char c;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
     {
     putch(0x07);    putch(0x07);    putch(0x07);
     printf("(1) There are no PCI-TMC12 card in this PC !!!\n");
     exit(0);
     }

printf("\n(2) Show the Configuration Space of all PCI-TMC12:");
for(i=0; i<wBoards; i++)
     {
     PTMC12_GetConfigAddressSpace(i,&wBaseAddr,&wIrq,&wPLX);
     printf("\nCard_%d: wBaseAddr=%x, wIrq=%x, wPLX=%x"
                    ,i,wBaseAddr,wIrq,wPLX);
     }

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* card_0 */
printf("\n(3) *** Card_0 D/O test, wBaseAddr=%x ***",wBaseAddr);
j=1;
for(i=0; i<16; i++)
     {
     pci_tmc12_do(j); printf("\nTEST_%2d --> DO = %x",i,j);
     c=getch(); if ((c=='q') || (c=='Q')) return;
     j=j<<1; if (j==0) j=1;
     }

PTMC12_DriverClose();
}

/* ------------------------------------------------------------ */

WORD pci_tmc12_do(WORD wDo)
{
  outport(wBaseAddr+0x14,wDo);
  return(NoError);
}
```

8.1.2 Demo2: Using the Digital Input

- If only one PCI-TMC12 series card is installed in the system, this program will test this card.
- If there is more than one PCI-TMC12 series card installed in the system, this program will **only test the second card**.
- For details of how to determine which card is the second card, refer to Section 6.2 .

```
/* ------------------------------------------------------------------ */
/* demo 2: D/I demo                                                    */
/* step 1: connect a CON2 & CON3 of PCI-TMC12 with a */
/*                  20-pin 1-to-1 flat cable                       */
/* step 2: run DEMO2.EXE                                   */
/* ------------------------------------------------------------------ */

#include "PCITMC12.H"
WORD pci_tmc12_do(WORD wDo);
void pci_tmc12_di(WORD *wDi);
WORD wBase,wIrq,wPLX;

int main()
{
int   i,j,k;
WORD wBoards,wRetVal;
char c;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);

if (wBoards>1)
       PTMC12_GetConfigAddressSpace(1,&wBase,&wIrq,&WPLX);/* card_1 */
else PTMC12_GetConfigAddressSpace(0,&wBase,&wIrq,&wPLX);/* card_0 */

printf("\n(3) *** D/I/O test , wBase=%x ***",wBase);
j=1;
for(i=0; i<16; i++)
    {
    pci_tmc12_do(j); pci_tmc12_di(&k);
    printf("\nTEST_%2d --> DO = %x , DI=%x",i,j,k);
    if (j!=k) printf(" <-- TEST ERROR");
    else         printf(" <-- TEST OK");
    j=j<<1; if (j==0) j=1;
    }

PTMC12_DriverClose();
}

/* ------------------------------------------------------------ */

void pci_tmc12_di(WORD *wDi)
{
WORD wRetVal;
(*wDi)=(inport(wBase+0x14))&0xffff;
}
```

8.1.3 Demo3: Wave Generator

```
/* ----------------------------------------------------------- */
/* demo 3 : Square Wave Generator                              */
/* step 1 : all CLK select clock1=8M                           */
/* step 2 : run DEMO3.EXE                                      */
/* step 3 : check all Cout of four 8254 by scope                */
/* ------------------------------------------------------------ */

#include "PCITMC12.H"

WORD pci_tmc12_select8254(char cChip);
WORD pci_tmc12_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c2(char cConfig, char cLow, char cHigh);
WORD wBaseAddr,wIrq,wPLX;

int main()
{
int   i,j;
WORD wBoards,wRetVal;
char c;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
   {
   putch(0x07);   putch(0x07);   putch(0x07);
   printf("(1) There are no PCI-TMC12 card in this PC !!!\n");
   exit(0);
   }
printf("\n(2) Show the Configuration Space of all PCI-TMC12:");
for(i=0; i<wBoards; i++)
   {
   PTMC12_GetConfigAddressSpace(i,&wBaseAddr,&wIrq,&wPLX);
   printf("\nCard_%d: wBaseAddr=%x, wIrq=%x, wPLX=%x"
            ,i,wBaseAddr,wIrq,wPLX);
   }

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* card_0 */
printf("\n(3) *** Card_0, wBaseAddr=%x ***",wBaseAddr);

printf("\n(4) *** Square Wave Generator for CH1 to CH3 ***");
pci_tmc12_select8254(0);     /* select 8254-chip-1                         */
pci_tmc12_c0(0x36,2,0);      /* CH-1,mode-3,low=2,high=0,cout=4M           */
pci_tmc12_c1(0x76,4,0);      /* CH-2,mode-3,low=4,high=0,cout=2M           */
pci_tmc12_c2(0xb6,8,0);      /* CH-3,mode-3,low=8,high=0,cout=1M           */

printf("\n(5) *** Square Wave Generator for CH4 to CH6 ***");
pci_tmc12_select8254(1);     /* select 8254-chip-2                        */
pci_tmc12_c0(0x36,16,0);     /* CH-4,mode-3,low=16,high=0,cout=500K */
pci_tmc12_c1(0x76,32,0);     /* CH-5,mode-3,low=32,high=0,cout=250K */
pci_tmc12_c2(0xb6,64,0);     /* CH-6,mode-3,low=64,high=0,cout=125K */

printf("\n(6) *** Square Wave Generator for CH7 to CH9 ***");
pci_tmc12_select8254(2);      /* select 8254-chip-3                       */
pci_tmc12_c0(0x36,128,0);   /* CH-7,mode-3,low=128,high=0,cout=64K */
pci_tmc12_c1(0x76,0,1);       /* CH-8,mode-3,low=0,high=1,cout=32K     */
pci_tmc12_c2(0xb6,0,2);        /* CH-9,mode-3,low=0,high=2,cout=16K      */
printf("\n(7) *** Square Wave Generator for CH10 to CH12 ***");
pci_tmc12_select8254(3);     /* select 8254-chip-4                        */
pci_tmc12_c0(0x36,0,4);      /* CH-10,mode-3,low=0,high=4,cout=8K    */
pci_tmc12_c1(0x76,0,8);      /* CH-11,mode-3,low=0,high=8,cout=4K    */
pci_tmc12_c2(0xb6,0,16);     /* CH-12,mode-3,low=0,high=16,cout=2K */

PTMC12_DriverClose();
}

/* ---------------------------------------------------------- */

WORD pci_tmc12_select8254(char cChip)
{
outportb(wBaseAddr+0x10,cChip);
return(NoError);
}

WORD pci_tmc12_c0(char cConfig, char cLow, char cHigh)
{
outportb(wBaseAddr+0x0C,cConfig);
outportb(wBaseAddr        ,cLow);
outportb(wBaseAddr        ,cHigh);
return(NoError);
}

WORD pci_tmc12_c1(char cConfig, char cLow, char cHigh)
{
outportb(wBaseAddr+0x0C,cConfig);
outportb(wBaseAddr+4     ,cLow);
outportb(wBaseAddr+4     ,cHigh);
return(NoError);
}

WORD pci_tmc12_c2(char cConfig, char cLow, char cHigh)
{
outportb(wBaseAddr+0x0C,cConfig);
outportb(wBaseAddr+8     ,cLow);
outportb(wBaseAddr+8     ,cHigh);
return(NoError);
}
```

8.1.4 Demo4: Generating a Delay of 1 ms

This demo program uses CNT1 to implement a **machine independent timer**. The demo can be run on any speed PC and the * value will be displayed on the screen at intervals of 1 second. A machine independent timer is very useful in industrial applications.

```
/* -------------------------------------------------------------- */
/* demo 4 : delay 1 ms Using CH-1                                  */
/* step 1 : CLK-1 select clock1=8M                                 */
/* step 2 : run demo4.exe                                               */
/* -------------------------------------------------------------- */

#include "PCITMC12.H"

WORD pci_tmc12_select8254(char cChip);
WORD pci_tmc12_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c2(char cConfig, char cLow, char cHigh);
WORD wBaseAddr,wIrq,wPLX;

int main()
{
int   i,j;
WORD wBoards,wRetVal;
char c;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* card_0 */
printf("\n(3) *** Card_0, wBaseAddr=%x ***",wBaseAddr);

printf("\n(4) *** Delay 1 ms ***\n");
for (;;)
     {
     for (i=0; i<1000; i++) delay_one_ms();
     printf("*");
     if (kbhit()!=0) {getch(); return;}
     }
PTMC12_DriverClose();
}
/* CLK-1=8M –> count 0x1f40 = count 8000 = 1 ms                    */
/* down count from 8000 –> 7999 –> ..... –> 1 –> 0 –> 0xfff */
delay_one_ms()
{
int low,high;
pci_tmc12_select8254(0);           /* select 8254-chip-0               */
pci_tmc12_c0(0x30,0x40,0x1f);      /* CH-1,mode-0 down count 8000      */
for (;;)
     {
     outportb(wBaseAddr+0x0C,0x00);      /* latch counter_0 */
     low=inportb(wBaseAddr);
     high=inportb(wBaseAddr);
     if (high>0x20) return;              /* overflow → time up          */
     }
}
```

8.1.5 Demo5: 16-bit Event Counter

```
/* ---------------------------------------------------------------- */
/* demo 5 : 16-bit event down counter                                 */
/* step 1 : CNT1 select ECLK1 (JP22)                                  */
/* step 2 : run demo5.exe                                             */
/* step 3 : connect the external CNT signal to pin1 of CON1           */
/* ---------------------------------------------------------------- */

#include "PCITMC12.H"
WORD pci_tmc12_select8254(char cChip);
WORD pci_tmc12_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c2(char cConfig, char cLow, char cHigh);
WORD wBaseAddr,wIrq,wPLX;

int main()
{
int   i,j;
WORD wBoards,wRetVal;
char c;
unsigned int high,low,count;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
   {
   putch(0x07);   putch(0x07);   putch(0x07);
   printf("(1) There are no PCI-TMC12 card in this PC !!!\n");
   exit(0);
   }

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* card_0 */
printf("\n(3) *** Card_0, wBaseAddr=%x ***",wBaseAddr);

printf("\n(4) *** 16-bit event down counter ***\n");
pci_tmc12_select8254(0);                        /* select 8254-chip-0       */
pci_tmc12_c0(0x30,0xff,0xff);             /* CH-1,mode-0 down count ffff */
for (;;)
    {
    outportb(wBaseAddr+0x0C,0x00);          /* latch counter_0 */
    low=inportb(wBaseAddr);
    high=inportb(wBaseAddr);
    count=(0xff-high)*256+(0xff-low)+2;
    printf("\nhigh=%x, low=%x, count=%u",high,low,count);
    if (kbhit()!=0) {getch(); break;}
    }

PTMC12_DriverClose();
}
```

Note1: The starting two ECLK will be used to initialize 8254.
So → Total_Count = 0xffff - Current_Counnt + 2
Note2: If the count > 65536 → this 16-bit counter will be overflow.
So → refer to DEMO6 for infinite-bit counter.

8.1.6 Demo6: Software Counter

```
/* ----------------------------------------------------------------- */
/* demo 6 : software event down counter                                              */
/* step 1 : CNT1 select ECLK1 (JP22)                                                    */
/* step 2 : run demo6.exe                                                                             */
/* step 3 : connect the external CNT signal to pin1 of CON1      */
/* ----------------------------------------------------------------- */

#include "PCITMC12.H"
WORD pci_tmc12_select8254(char cChip);
WORD pci_tmc12_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c2(char cConfig, char cLow, char cHigh);
WORD wBaseAddr,wIrq,wPLX;

float c65536,software_count;
int main()
{
int   i,j;
WORD wBoards,wRetVal;
char c,s0;
unsigned int high,low;

c65536=0; s0=0;
clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* card_0 */
printf("\n(3) *** Card_0, wBaseAddr=%x ***",wBaseAddr);

printf("\n(4) *** 16-bit event down counter ***\n");

pci_tmc12_select8254(0);                          /* select 8254-chip-0               */
pci_tmc12_c0(0x30,0xff,0xff);           /* CH-1,mode-0 down count ffff */
for (;;)
      {
      outportb(wBaseAddr+0x0C,0x00);         /* latch counter_0 */
      low=inportb(wBaseAddr);
      high=inportb(wBaseAddr);
      if (high < 0x80) s0=1;
      if ((high > 0x80 ) && (s0==1))
           {
           c65536 += 1.0; s0=0;
           }
      software_count=c65536*65536.0+(0xff-high)*256+(0xff-low)+2;
      printf("\nhigh=%x, low=%x, c65536=%f, software_count=%f"
      ,high,low,c65536,software_count);
      if (kbhit()!=0) {getch(); break;}
      }

PTMC12_DriverClose();
}
```

Note 1: The starting two ECLK will be used to initialize 8254.
Note 2: c65536 will be increment by 1 every 65536 counts
Note 3: So → Total_Count = c65536*65536 + 0xffff - Current_Counnt + 2
Note 4: This software counter can be nearly infinite-bits.

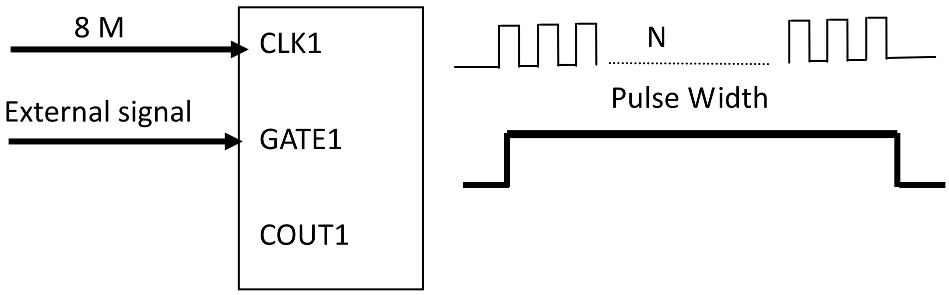
8.1.7 Demo7: Watchdog Timer

```
/* ---------------------------------------------------------------- */
/* demo 7 : watchdog timer using CH-3                                */
/* step 1 : CLK-3 select clock2=80K (J24)                             */
/* step 2 : INT select CH3 (J2                                        */
/* step 3 : run demo7.exe                                                  */
/* ---------------------------------------------------------------- */

#include "PCITMC12.H"
#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI  0x20

WORD pci_tmc12_select8254(char cChip);
WORD pci_tmc12_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c2(char cConfig, char cLow, char cHigh);
WORD init_watchdog();
WORD wBaseAddr,wIrq,wPLX;

static void interrupt irq_service();
int watchdog,irqmask;

int main()
{
int   i,j;
WORD wBoards,wRetVal;
char c;
DWORD dwVal;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
     {
     putch(0x07);   putch(0x07);   putch(0x07);
     printf("(1) There are no PCI-TMC12 card in this PC !!!\n");
     exit(0);
     }

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* card_0 */
printf("\n(3)Card_0, wIrq=%x, wPLX=%x ",wIrq,wPLX);

watchdog=0;
pci_tmc12_select8254(0);                /* select 8254-chip-0                */
printf("\n(4) *** start refresh watchdog **\n");
init_watchdog();

for (;;)
       {
       refresh_watchdog();
       printf("\npress any key to simulate PC fail,watch=%d",watchdog);
       if (kbhit()!=0) {getch(); break;}
       }

printf("\nWait watchdog failure");
for (;;)
    {
    if (watchdog != 0)
        {
        printf("\nwatchdog is failure now");
        break;
        }
    if (kbhit()!=0) {getch(); break;}
    }

PTMC12_DriverClose();
_outpd(wPLX+0x4c,0);    /* disable all interrupt */
}
/* ------------------------------------------------------- */

WORD init_watchdog()
{
DWORD dwVal;

disable();

refresh_watchdog();
_outpd(wPLX+0x4c,0x41);        /* channel_1, interrupt active_Low */

if (wIrq<8)
    {
    irqmask=inportb(A1_8259+1);
    outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
    setvect(wIrq+8, irq_service);
    printf("<%x>",wIrq);
    }
else
    {
    irqmask=inportb(A1_8259+1);
    outportb(A1_8259+1,irqmask & 0xfb);                /* IRQ2 */
    outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
    irqmask=inportb(A2_8259+1);
    outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
    setvect(wIrq-8+0x70, irq_service);
    printf("[%x]",wIrq);
    }

enable();
}

/* 80K*65536_count=0.8192 sec --> high_width=0.4096 sec       */
/* --> the user has to refresh the watchdog before 0.4 sec */
refresh_watchdog()
{
pci_tmc12_c2(0xb6,0xff,0xff);          /* mode_3, CNT2--> CH3                  */
return(NoError);
}

void interrupt irq_service()
{
watchdog++;
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

Refer to Section 6.3.5 for more information.

8.1.8 Demo8: Pulse Width Measurement

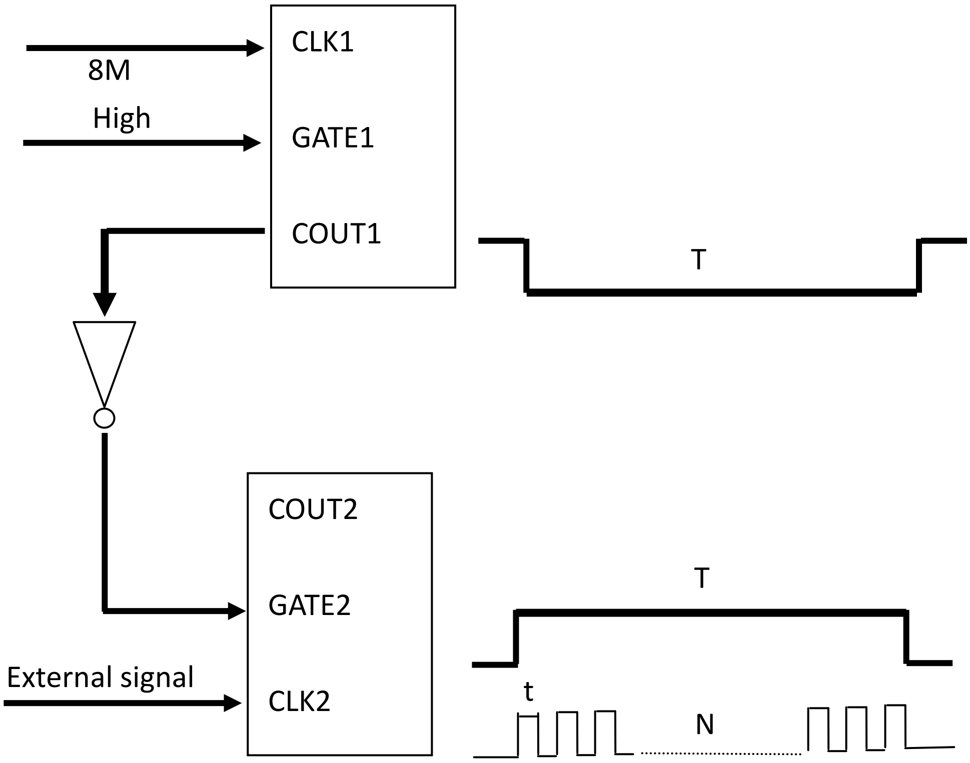
```
/* ----------------------------------------------------------------------- */
/* demo 8 : Pulse Width Measure                                             */
/* step 1 : J19 select EXTG1, J22 select CLOCL1=8M hz           */
/* step 2 : connect pin20 of CON1 to pin1 of CON2                     */
/* step 3 : connect external signal to (pin20,pin19)                   */
/* step 4 : run demo8.exe, the width of active high pulse will */
/*              be shown in the screen. (8 ms max.)                      */
/* ---------------------------------------------------------------------- */

#include "PCITMC12.H"

void pci_tmc12_di(WORD *wDi);
WORD pci_tmc12_select8254(char cChip);
WORD pci_tmc12_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c2(char cConfig, char cLow, char cHigh);
WORD wBaseAddr,wIrq,wPLX;

int main()
{
int    i,j,k;
WORD wBoards,wRetVal;
char c,cc[80];
unsigned int high,low,count;
float ms;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
    {
    putch(0x07);    putch(0x07);    putch(0x07);
    printf("(1) There are no PCI-TMC12 card in this PC !!!\n");
    exit(0);
    }

printf("\n(2) Show the Configuration Space of all PCI-TMC12:");
for(i=0; i<wBoards; i++)
    {
    PTMC12_GetConfigAddressSpace(i,&wBaseAddr,&wIrq,&wPLX);
    printf("\n       Card_%d: wBaseAddr=%x, wIrq=%x, wPLX=%x"
              ,i,wBaseAddr,wIrq,wPLX);
    }

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* card_0 */
printf("\n(3) *** Card_0, wBaseAddr=%x ***",wBaseAddr);

printf("\n(4) *** read EXTG1 & show 80-read ***\n",wBaseAddr);
for (i=0; i<80; i++)
     {
     pci_tmc12_di(&k);
     cc[i]=k;
     }

for (i=0; i<80; i++)
     {
     j=cc[i]&0x01;
     if (j==0) printf("0"); else printf("1");
      }
while (((inport(wBaseAddr+0x14))&1)==0);/* wait EXG1=High          */
while (((inport(wBaseAddr+0x14))&1)!=0);/* wait EXG1=Low           */

pci_tmc12_select8254(0);                  /* select 8254-chip-0           */
pci_tmc12_c0(0x30,0xff,0xff);          /* CH-1,mode-0 down count ffff */

while (((inport(wBaseAddr+0x14))&1)==0);/* wait EXG1=High          */
while (((inport(wBaseAddr+0x14))&1)!=0);/* wait EXG1=Low           */

outportb(wBaseAddr+0x0C,0x00);                 /* latch counter_0 */
low=inportb(wBaseAddr);
high=inportb(wBaseAddr);
count=(0xff-high)*256+(0xff-low)+2;
ms=0.000125*(float)count;
printf("\nhigh=%x, low=%x, count=%d : %f ms",high,low,count,ms);

PTMC12_DriverClose();
}
```



- **N = The down count value for CNT1(8 M clock)**
- **Pulse width = 8M_width * N**

8.1.9 Demo9: Frequency Measurement

```
/* -------------------------------------------------------------- */
/* demo 9 : Signal Frequency Measure                               */
/* step 1 : J19 select EXTG1, J22 select CLOCL1=8M hz              */
/* step 2 : J20 select \COUT1,J23 select ECLK2                     */
/* step 3 : connect external signal to (pin21,pin19)               */
/* step 4 : run demo9.exe, the frequency of input signal will      */
/*          be shown in the screen. (125 Hz min.)                  */
/* -------------------------------------------------------------- */

#include "PCITMC12.H"

void pci_tmc12_di(WORD *wDi);
WORD pci_tmc12_select8254(char cChip);
WORD pci_tmc12_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c2(char cConfig, char cLow, char cHigh);
WORD wBaseAddr,wIrq,wPLX;

int main()
{
int   i,j,k;
WORD wBoards,wRetVal;
char c,cc[80];
unsigned int high,low,count,cout0;
float f,t;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
   {
   putch(0x07);   putch(0x07);   putch(0x07);
   printf("(1) There are no PCI-TMC12 card in this PC !!!\n");
   exit(0);
   }

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* card_0 */
printf("\n(3) *** Card_0, wBaseAddr=%x ***",wBaseAddr);

printf("\n(4) *** frequency must be > 125 Hz ***\n",wBaseAddr);

pci_tmc12_select8254(0);                /* select 8254-chip-0          */
pci_tmc12_c0(0x30,0xff,0xff);       /* CH-1,mode-0 down count ffff */
pci_tmc12_c1(0x70,0xff,0xff);       /* CH-2,mode-0 down count ffff */

for (;;)
    {
    outportb(wBaseAddr+0x0C,0xE2);      /* latch status of counter0     */
    low=inportb(wBaseAddr);
    high=inportb(wBaseAddr);
    cout0=low&0x80;
    if (cout0!=0) break;
    if (kbhit()!=0) {getch(); break;}
    }

outportb(wBaseAddr+0x0C,0x40);          /* latch counter_1 */
low=inportb(wBaseAddr+0x04);
high=inportb(wBaseAddr+0x04);
count=(0xff-high)*256+(0xff-low)+2;
/* COUT0 = 65536*0.000125=8.192 ms */
t=8.192/(float)count;      /* ms  */
f=(1.0/t)*1000.0; /* f=1/T */
printf("\nhigh=%x, low=%x, count=%d : frequency = %f Hz",high,low,count,f);

PTMC12_DriverClose();
}
```



- **Down_count2 = The down count value for CNT2**
- **t = T/Down_count2**
- **f = 1/t**
- **CNT1 can be changed to CNT3/4/5/6. The COUT of CNT 8/9/10/11/12/13 is directly connected to the next counter without including an inverter, so cannot be used to replace CNT1**.
- **The 12 CNTs on the TMC-12 are divided into two groups: the inverter group and the non-inverted group. The inverted group includes CNT 1/2/3/4/5/6, while the non-inverted group includes CNT 7/8/9/10/11/12. The appropriate group should be selected for the different applications.**

8.1.10 Demo10: Find Card Number

```
/* -------------------------------------------------------------- */
/* demo 10: Find card number demo                                 */
/* step 1 : run DEMO10.EXE                                        */
/* step 2 : connect a 20-pin flat cable to CON2&CON3 of card_? */
/* step 3 : The card number is shown in screen as TEST OK        */
/* -------------------------------------------------------------- */

#include "PCITMC12.H"

WORD pci_tmc12_do(WORD wDo);
void pci_tmc12_di(WORD *wDi);
WORD pci_tmc12_select8254(char cChip);
WORD pci_tmc12_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c2(char cConfig, char cLow, char cHigh);
WORD wBaseAddr,wIrq;

int main()
{
int   i,j,k;
WORD wBoards,wRetVal;
char c;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
    {
    putch(0x07);   putch(0x07);   putch(0x07);
    printf("(1) There are no PCI-TMC12 card in this PC !!!\n");
    exit(0);
    }

for (;;)
     {
     printf("\n------------ press any key to stop ------------");
     for (i=0; i<wBoards; i++) test_card(i);
     for (i=0; i<1000; i++) delay_one_ms();   /* delay 1 sec */
     if (kbhit()!=0) {getch(); break;}
     }
PTMC12_DriverClose();
}

/* ------------------------------------------------------- */

test_card(int card)
{
int i,j,k,ok;

PTMC12_GetConfigAddressSpace(card,&wBaseAddr,&wIrq);
j=1; ok=1;
for(i=0; i<16; i++)
    {
    pci_tmc12_do(j); pci_tmc12_di(&k);
    if (j!=k) ok=0;
    j=j<<1; if (j==0) j=1;
    }
printf("\nCard Number=%d, wBaseAddr=%x",card,wBaseAddr);
if (ok==1) printf(", Test OK"); else printf(", Test ERROR");
}
```

8.1.11 Demo11: Count Low Pulse

```
/* ------------------------------------------------------------------------ */
/* demo 11: count low pulse                                                 */
/*              (Use CH-3 to simulate external pulse)                       */
/* step 1 : CLK-3 select clock2=80K                                         */
/* step 2 : J25 select CH3                                                  */
/* step 3 : run demo11.exe                                                  */
/* ----------------------------------------------------------------------- */

#include "PCITMC12.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI  0x20

WORD pci_tmc12_select8254(char cChip);
WORD pci_tmc12_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c2(char cConfig, char cLow, char cHigh);
WORD init_CH3();
WORD wBaseAddr,wIrq,wPLX;

static void interrupt irq_service();
int COUNT3,irqmask,now_int_state;

int main()
{
int   i,j;
WORD wBoards,wRetVal;
char c;
DWORD dwVal;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
    {
    putch(0x07);   putch(0x07);   putch(0x07);
    printf("(1) There are no PCI-TMC12 card in this PC !!!\n"); exit(0);
    }

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* card_0 */
printf("\n(3) *** Card_0, wBaseAddr=%x ***",wBaseAddr);

COUNT3=0;
pci_tmc12_select8254(0);                /* select 8254-chip-0                */
printf("\n(4) *** show the count of low_pulse **\n");
init_CH3();

for (;;)
     {
     printf("\nCOUNT3=%d",COUNT3);
     if (kbhit()!=0) {getch(); break;}
     }

PTMC12_DriverClose();
_outpd(wPLX+0x4c,0);    /* disable all interrupt */
}
/* ---------------------------------------------------------------- */
/* Use CH3 to simulate the external signal                          */
/* The user can must set the J25=CH3 in this demo.                  */
/* The user can set the J25=EXT in real world application.     */

WORD init_CH3()
{
DWORD dwVal;

disable();
pci_tmc12_c2(0xb6,0xff,0xff);  /* mode_3, CNT2--> CH3              */
/* 80K*65536_count=0.8192 sec --> high_width=0.4096 sec           */
/* --> high_width=0.4 sec, low_width=0.4 sec,              */
now_int_state=1;                        /* now COUT3 is High         */

_outpd(wPLX+0x4c,0x41); /* channel_1, interrupt active_Low     */
if (wIrq<8)
    {
    irqmask=inportb(A1_8259+1);
    outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
    setvect(wIrq+8, irq_service);
    }
else
    {
    irqmask=inportb(A1_8259+1);
    outportb(A1_8259+1,irqmask & 0xfb);                   /* IRQ2 */
    outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
    irqmask=inportb(A2_8259+1);
    outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
    setvect(wIrq-8+0x70, irq_service);
    }

enable();
}

void interrupt irq_service()
{
if (now_int_state==0)             /* old state=low → change to high now */
    {
                                           /* find a high_pulse here                        */

    now_int_state=1;                        /* now int_signal is High                      */
    _outpd(wPLX+0x4c,0x41);   /* channel_1, interrupt active_Low */
    }
else
    {                                    /* old state=high → change to low now   */
                                            /* find a low_pulse                        */
    now_int_state=0;                     /* now int_signal is low                     */
    COUNT3++;                            /* only count low pulse                    */
   _outpd(wPLX+0x4c,0x43); /* channel_1, interrupt active_High   */
   }

if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

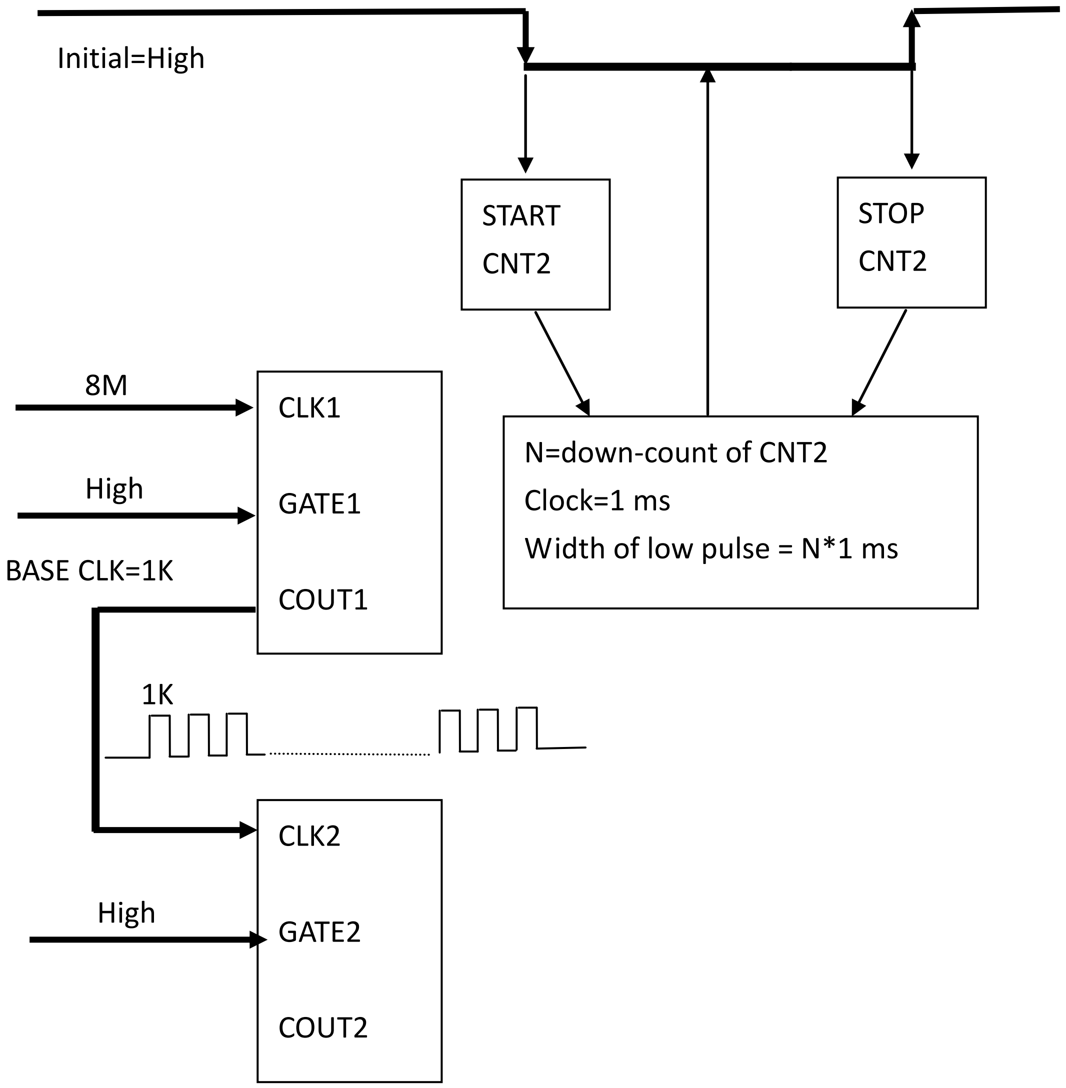
Refer to Section 6.3.5 for more information.

8.1.12 Demo12: Low Pulse Width

```
/* -------------------------------------------------------------------------- */
/* demo 12: detect the pulse_width of low_pulse                         */
/*             (Use CH-3 to simulate external pulse)                           */
/* step 1 : CLK-3 select clock2=80K --> simulate ext signal          */
/* step 2 : CLK-1 select clock1=8M    --> generate BASE clock      */
/* step 3 : CLK-2 select COUT1=1K     --> measure pulse-width*/
/* step 4 : J25 select CH3                                                          */
/* step 5 : run demo12.exe                                                          */
/* -------------------------------------------------------------------------- */

#include "PCITMC12.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI  0x20

WORD pci_tmc12_select8254(char cChip);
WORD pci_tmc12_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c2(char cConfig, char cLow, char cHigh);
WORD init_CH3();
WORD wBaseAddr,wIrq,wPLX;

static void interrupt irq_service();
int COUNT3,WIDTH3,CNT_H,CNT_L,irqmask,now_int_state;

int main()
{
int    i,j;
WORD wBoards,wRetVal,count;
char c;
DWORD dwVal;
float low_pulse_width;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
     {
     putch(0x07);    putch(0x07);    putch(0x07);
     printf("(1) There are no PCI-TMC12 card in this PC !!!\n"); exit(0);
     }

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* card_0 */
printf("\n(3) *** Card_0, wBaseAddr=%x ***",wBaseAddr);

printf("\n***(4) detect the pulse_width of low_pulse ***");
pci_tmc12_select8254(0);                    /* select 8254-chip-0                 */
for(;;)
     {
     printf("\npress any key to continue, Q to stop");
     c=getch(); if ((c=='q') || (c=='Q')) goto ret_label;
     COUNT3=0;
     init_CH3();
     while (COUNT3 < 4)
          {
          if (kbhit()!=0) {getch(); break;}
          }
count=(0xff-CNT_H)*256+(0xff-CNT_L)+2;
    /* COUT0 = 1 ms */
    low_pulse_width=(float)count*1.0;
    printf("\nCNT_H=%x, CNT_L=%x, Low_pulse=%f",CNT_H,CNT_L,low_pulse_width);
    }

ret_label:
PTMC12_DriverClose();
_outpd(wPLX+0x4c,0);     /* disable all interrupt */
}
/* ------------------------------------------------------------------------- */
/* Use CH3 to simulate the external signal                                        */
/* The user can must set the J25=CH3 in this demo.                           */
/* The user can set the J25=EXT in real world application.                   */
WORD init_CH3()
{
DWORD dwVal;

disable();

pci_tmc12_c2(0xb6,0xff,0xff);  /* mode_3, CNT2--> CH3                       */
/* 80K*65536_count=0.8192 sec --> high_width=0.4096 sec                    */
/* --> high_width=0.4 sec, low_width=0.4 sec                              */

pci_tmc12_c0(0x36,0,32);                /* CH-1,mode-3,low=0,high=32,cout=1K*/
_outpd(wPLX+0x4c,0x41);               /* channel_1, interrupt active_Low         */
now_int_state=1;                           /* now int_signal is High                              */
if (wIrq<8)
    {
    irqmask=inportb(A1_8259+1);
    outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
    setvect(wIrq+8, irq_service);
    }
else
    {
    irqmask=inportb(A1_8259+1);
    outportb(A1_8259+1,irqmask & 0xfb);                      /* IRQ2 */
    outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
    irqmask=inportb(A2_8259+1);
    outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
    setvect(wIrq-8+0x70, irq_service);
    }

enable();
}
void interrupt irq_service()
{
if (now_int_state==0)                  /* old state=low → change to high now */
    {
    COUNT3++;                               /* find a HIGH_pulse                                */
    if (COUNT3==4)                      /* stop down-count & read-counter                  */
        {
        outportb(wBaseAddr+0x0C,0x40); /* latch counter1                         */
        CNT_L=inportb(wBaseAddr+0x04);
        CNT_H=inportb(wBaseAddr+0x04);
        _outpd(wPLX+0x4c,0);             /* disable all interrupt                */
        }
    _outpd(wPLX+0x4c,0x41);            /* channel_1, interrupt active_Low */
    now_int_state=1;                          /* now int_signal is High            */
    }
```

```
else                              /* old state=low → change to high now    */
     {
     COUNT3++;                                    /* find a low_pulse                   */
     if (COUNT==3)                                /* start counter                             */
          pci_tmc12_c1(0x70,0xff,0xff);   /* CH-2,mode-0 down count ffff */
     else
          _outpd(wPLX+0x4c,0x43);                   /* channel_1, interrupt active_High*/
     now_int_state=0;                             /* now int_signal is Low              */
     }

if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

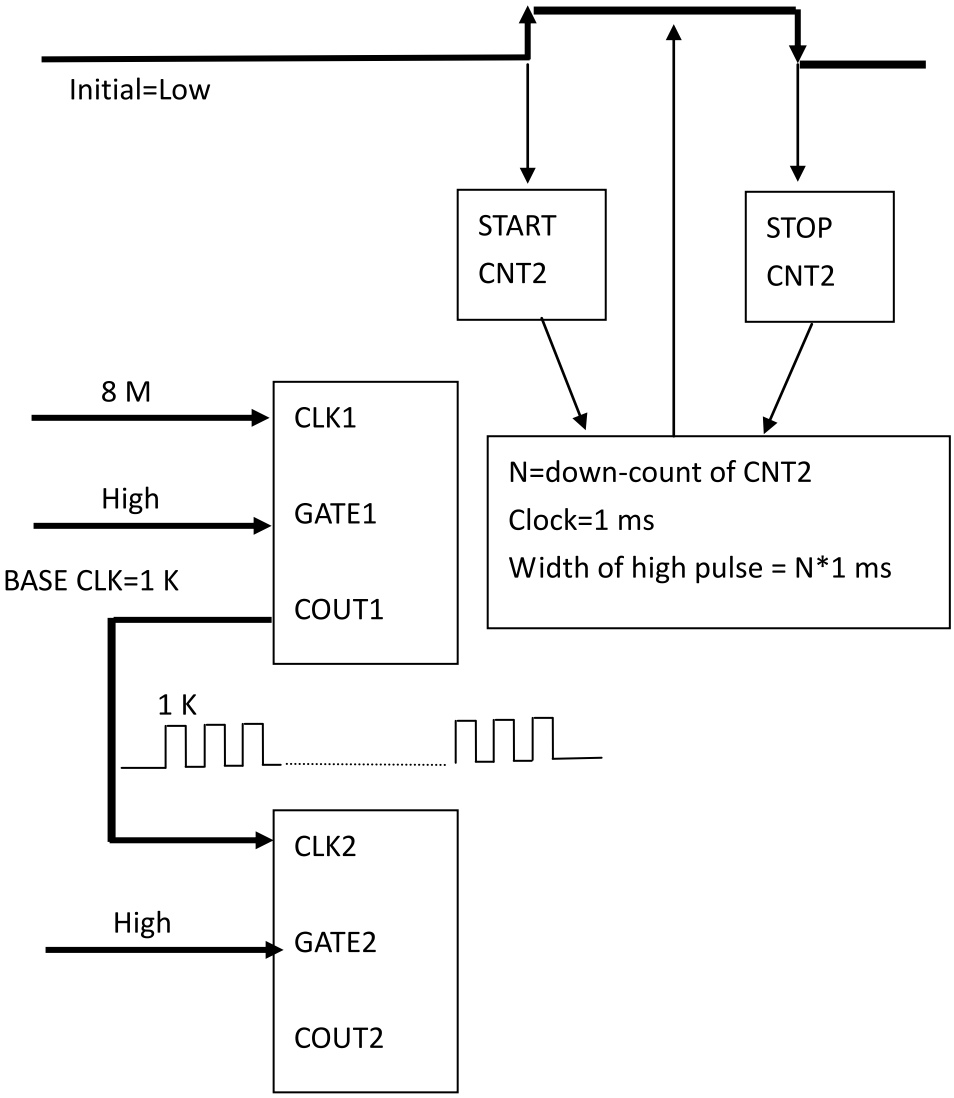
Refer to Section 6.3.5 for more information.



8.1.13 Demo13: High Pulse Width

```
/* ------------------------------------------------------------------------ */
/* demo 13 detect the pulse_width of high_pulse                              */
/*            (Use CH-3 to simulate external pulse)                           */
/* step 1 : CLK-3 select clock2=80K --> simulate ext signal                  */
/* step 2 : CLK-1 select clock1=8M   --> generate BASE clock                 */
/* step 3 : CLK-2 select COUT1=1K    --> measure pulse-width                 */
/* step 4 : J25 select CH3                                                    */
/* step 5 : run demo13.exe                                                    */
/* ------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------*/
/* Use CH3 to simulate the external signal                                   */
/* The user can must set the J25=CH3 in this demo.                           */
/* The user can set the J25=EXT in real world application.                  */
/* ----------------------------------------------------------------------- */

WORD init_CH3()
{
DWORD dwVal;

disable();

pci_tmc12_c2(0xb6,0xff,0xff);  /* mode_3, CNT2--> CH3                        */
/* 80K*65536_count=0.8192 sec --> high_width=0.4096 sec                      */
/* --> high_width=0.4 sec, low_width=0.4 sec                                  */

pci_tmc12_c0(0x36,0,32); /* CH-1,mode-3,low=0,high=32,cout=1K  */
_outpd(wPLX+0x4c,0x41); /* channel_1, interrupt active_Low        */
now_int_state=1;                    /* now int_signal is High                       */
if (wIrq<8)
    {
    irqmask=inportb(A1_8259+1);
    outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
    setvect(wIrq+8, irq_service);
    }
else
    {
    irqmask=inportb(A1_8259+1);
    outportb(A1_8259+1,irqmask & 0xfb);                        /* IRQ2 */
    outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
    irqmask=inportb(A2_8259+1);
    outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
    setvect(wIrq-8+0x70, irq_service);
    }

enable();
}

void interrupt irq_service()
{
if (now_int_state==0)
    {
    COUNT3++;                                          /* find a high_pulse                        */
    if (COUNT3==2)                                     /* start to down-count                      */
        pci_tmc12_c1(0x70,0xff,0xff);        /* CH-2,mode-0 down count ffff */
    _outpd(wPLX+0x4c,0x41);            /* channel_1, interrupt active_Low */
    now_int_state=1;                               /* now int_signal is High                   */
    }
else
    {
    COUNT3++;                                         /* find a low_pulse                                    */
    if (COUNT3==3)                                     /* stop the down-count & read-count*/
        {
        outportb(wBaseAddr+0x0C,0x40); /* latch counter1                         */
        CNT_L=inportb(wBaseAddr+0x04);
        CNT_H=inportb(wBaseAddr+0x04);
        _outpd(wPLX+0x4c,0);                          /* disable all interrupt               */
        }
    else
        _outpd(wPLX+0x4c,0x43);                    /* channel_1, interrupt active_High*/
    now_int_state=0;                            /* now int signal is Low              */
    }

if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

Refer to Section 6.3.5 for more information.



8.1.14 Ndemo1: Using the LEDs

```
/* -------------------------------------------------------------------- */
/* ndemo1 : LED1, LED2, LED3 demo                                       */
/* step 1 : default shipping of PCI-TMC12A                          */
/* step 2 : run NDEMO1.EXE                                              */
/* step 3 : the LED1/2/3 of TMC12A will turn on sequentially        */
/* ------------------------------------------------------------------ */

#include "PCITMC12.H"
WORD pci_tmc12_do(WORD wDo);
WORD pci_tmc12_do2(WORD wXor);
WORD wBaseAddr,wIrq,wPLX;
int main()
{
int   i,j;
WORD wBoards,wRetVal;
char c;
clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
     {
     putch(0x07);   putch(0x07);   putch(0x07);
     printf("(1) There are no PCI-TMC12 card in this PC !!!\n"); exit(0);
     }
printf("\n(2) Show the Configuration Space of all PCI-TMC12:");
for(i=0; i<wBoards; i++)
     {
     PTMC12_GetConfigAddressSpace(i,&wBaseAddr,&wIrq,&wPLX);
     printf("\n      Card_%d: wBaseAddr=%x, wIrq=%x, wPLX=%x",i,wBaseAddr,wIrq,wPLX);
     }
PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /*select card_0 */
printf("\n(3) *** Card_0 LED test, wBaseAddr=%x ***",wBaseAddr);
pci_tmc12_do2(0xe000); printf("\nAll LED off, press any key to continue"); getch();
pci_tmc12_do2(0xc000); printf("\nLED1 on,        press any key to continue"); getch();
pci_tmc12_do2(0xa000); printf("\nLED2 on,        press any key to continue"); getch();
pci_tmc12_do2(0x6000); printf("\nLED3 on,        press any key to continue"); getch();

PTMC12_DriverClose();
}

/* --------------------------------------------------------- */
WORD pci_tmc12_do(WORD wDo)
{
outport(wBaseAddr+0x14,wDo);
return(NoError);
}

/* --------------------------------------------------------- */
WORD pci_tmc12_do2(WORD wXor)
{
outport(wBaseAddr+0x18,wXor);
return(NoError);
}
```

8.1.15 Ndemo2: Generating Two Starting Clocks

```
/* ------------------------------------------------------------------ */
/* ndemo2 : generate 2 starting clock demo                            */
/* step 1 : all clock sources select external_clock                   */
/* step 2 : run NDEMO2.EXE                                            */
/* step 3 : read the counter value of counter1 to counter12           */
/* ---------------------------------------------------------------- */

#include "PCITMC12.H"

WORD wBaseAddr,wIrq,wPLX;
WORD pci_tmc12_do(WORD wDo);
WORD pci_tmc12_do2(WORD wXor);
WORD pci_tmc12_select8254(char cChip);
WORD pci_tmc12_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c2(char cConfig, char cLow, char cHigh);
void read_c0(int B);
void read_c1(int B);
void read_c2(int B);

int main()
{
int   i,j;
WORD wBoards,wRetVal;
char c;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
   {
   putch(0x07);   putch(0x07);   putch(0x07);
   printf("(1) There are no PCI-TMC12 card in this PC !!!\n"); exit(0);
   }

printf("\n(2) Show the Configuration Space of all PCI-TMC12:");
for(i=0; i<wBoards; i++)
   {
   PTMC12_GetConfigAddressSpace(i,&wBaseAddr,&wIrq,&wPLX);
   printf("\n      Card_%d: wBaseAddr=%x, wIrq=%x, wPLX=%x",i,wBaseAddr,wIrq,wPLX);
   }

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* select card_0 */
printf("\n(3) *** Card_0 LED test, wBaseAddr=%x ***",wBaseAddr);

/* initial count */

pci_tmc12_select8254(0);
pci_tmc12_c0(0x30,0xfe,0xff);
pci_tmc12_c1(0x70,0xfd,0xff);
pci_tmc12_c2(0xb0,0xfc,0xff);

pci_tmc12_select8254(1);
pci_tmc12_c0(0x30,0xfb,0xff);
pci_tmc12_c1(0x70,0xfa,0xff);
pci_tmc12_c2(0xb0,0xf9,0xff);
pci_tmc12_select8254(2);
pci_tmc12_c0(0x30,0xf8,0xff);
pci_tmc12_c1(0x70,0xf7,0xff);
pci_tmc12_c2(0xb0,0xf6,0xff);

pci_tmc12_select8254(3);
pci_tmc12_c0(0x30,0xf5,0xff);
pci_tmc12_c1(0x70,0xf4,0xff);
pci_tmc12_c2(0xb0,0xf3,0xff);

/* generate 2 starting clocks for all channels (Counter1~Counter12) */

delay(1);
pci_tmc12_do2(0);
pci_tmc12_do2(0x0fff);
pci_tmc12_do2(0);
pci_tmc12_do2(0x0fff);
pci_tmc12_do2(0);

for (;;)
    {
    pci_tmc12_select8254(0);
    read_c0(1);    /* Counter 1 */
    read_c1(2); /* Counter 2 */
    read_c2(3); /* Counter 3 */

    pci_tmc12_select8254(1);
    read_c0(4); /* Counter 4 */
    read_c1(5); /* Counter 5 */
    read_c2(6); /* Counter 6 */

    pci_tmc12_select8254(2);
    read_c0(7); /* Counter 7 */
    read_c1(8); /* Counter 8 */
    read_c2(9); /* Counter 9 */

    pci_tmc12_select8254(3);
    read_c0(10); /* Counter 10 */
    read_c1(11); /* Counter 11 */
    read_c2(12); /* Counter 12 */

    /* generate one clock to all channels for testing only */
    pci_tmc12_do2(0x0fff);
    pci_tmc12_do2(0);

    printf("\n-----------------------");
    c=getch();
    if ((c=='q') || (c=='Q')) return;
    }

PTMC12_DriverClose();
}

/* -------------------------------------------------------- */
WORD pci_tmc12_do(WORD wDo)
{
outport(wBaseAddr+0x14,wDo);
return(NoError);
}

/* -------------------------------------------------------- */
WORD pci_tmc12_do2(WORD wXor)
{
outport(wBaseAddr+0x18,wXor);
return(NoError);
}
```

8.1.16 Ndemo3: Implementing a Watchdog Timer (Modified Version of Demo7)

```
/* -------------------------------------------------------------------- */
/* ndemo3 : watchdog timer using CH-3 (modified from demo7)       */
/*            (only add 2 lines to pre-set int_signal_to_PC)              */
/* step 1 : CLK-3 select clock2=80K                                        */
/* step 2 : J25 select CH3                                                      */
/* step 3 : run ndemo3.exe                                                      */
/* ------------------------------------------------------------------- */

#include "PCITMC12.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI  0x20

WORD pci_tmc12_select8254(char cChip);
WORD pci_tmc12_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmc12_c2(char cConfig, char cLow, char cHigh);
WORD init_watchdog();
WORD wBaseAddr,wIrq,wPLX;

static void interrupt irq_service();
int watchdog,irqmask;

int main()
{
int   i,j;
WORD wBoards,wRetVal;
char c;
DWORD dwVal;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
    {
    putch(0x07);   putch(0x07);   putch(0x07);
    printf("(1) There are no PCI-TMC12 card in this PC !!!\n"); exit(0);
    }

printf("\n(2) Show the Configuration Space of all PCI-TMC12:");
for(i=0; i<wBoards; i++)
    {
    PTMC12_GetConfigAddressSpace(i,&wBaseAddr,&wIrq,&wPLX);
    printf("\n      Card_%d: wBaseAddr=%x, wIrq=%x, wPLX=%x",i,wBaseAddr,wIrq,wPLX);
    }

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* select card_0 */
printf("\n(3) *** Card_0, wBaseAddr=%x ***",wBaseAddr);

watchdog=0;
pci_tmc12_select8254(0);                /* select 8254-chip-0                */
printf("\n(4) *** start refresh watchdog **\n");
init_watchdog();
for (;;)
    {
    refresh_watchdog();
    printf("\npress any key to simulate PC fail, watchdog=%d",watchdog);
    if (kbhit()!=0) {getch(); break;}
    }

printf("\nWait watchdog failure");
for (;;)
    {
    if (watchdog != 0)
       {
       printf("\nwatchdog is failure now");
       break;
       }
    if (kbhit()!=0) {getch(); break;}
    }

PTMC12_DriverClose();
_outpd(wPLX+0x4c,0);    /* disable all interrupt */
}

/* -------------------------------------------------------------- */
WORD init_watchdog()
{
DWORD dwVal;

inport(wBaseAddr+0x18); /* pre-set int_signal_to_PC, added line 1 */
disable();
refresh_watchdog();
_outpd(wPLX+0x4c,0x41);       /* channel_1, interrupt active_Low */

if (wIrq<8)
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
   setvect(wIrq+8, irq_service);
   }
else
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & 0xfb);                   /* IRQ2 */
   outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
   irqmask=inportb(A2_8259+1);
   outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
   setvect(wIrq-8+0x70, irq_service);
   }

enable();
}

/* 80K*65536_count=0.8192 sec --> high_width=0.4096 sec      */
/* --> the user has to refresh the watchdog before 0.4 sec */
refresh_watchdog()
{
pci_tmc12_c2(0xb6,0xff,0xff);          /* mode_3, CNT2--> CH3                 */
return(NoError);
}

void interrupt irq_service()
{
watchdog++;
inport(wBaseAddr+0x18); /* pre-set int_signal_to_PC, added line 2 */
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

8.1.17 Ndemo4: Implementing an Active High Interrupt

```
/* ------------------------------------------------------------------------ */
/* ndemo4 : interrupt demo, int source=initial low, active High              */
/* step 1 : connect DO1 (pin1 of CON3) to ECLK11 (pin16 of CON1) */
/* step 2 : J25 select EXT                                                          */
/* step 3 : run ndemo4.exe                                                          */
/* step 4 : press any key to test, press Q to stop                      */
/* ----------------------------------------------------------------------- */

#include "PCITMC12.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI  0x20

WORD pci_tmc12_do(WORD wDo);
WORD pci_tmc12_do2(WORD wDo);
WORD init_interrupt();
WORD wBaseAddr,wIrq,wPLX,int_count;

static void interrupt irq_service();
int irqmask;

int main()
{
int    i,j;
WORD wBoards,wRetVal,old_count;
char c;
DWORD dwVal;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
     {
     putch(0x07);    putch(0x07);    putch(0x07);
     printf("(1) There are no PCI-TMC12 card in this PC !!!\n"); exit(0);
     }

printf("\n(2) Show the Configuration Space of all PCI-TMC12:");
for(i=0; i<wBoards; i++)
     {
     PTMC12_GetConfigAddressSpace(i,&wBaseAddr,&wIrq,&wPLX);
     printf("\n       Card_%d: wBaseAddr=%x, wIrq=%x, wPLX=%x",i,wBaseAddr,wIrq,wPLX);
     }

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* select card_0 */
printf("\n(3) *** Card_0, wBaseAddr=%x ***",wBaseAddr);

printf("\n(4) *** start test interrupt      **\n");

pci_tmc12_do(0); /* DO1=int source --> initial low, active High */

init_interrupt();
old_count=1;
for (;;)
    {
    if (old_count != int_count)
       {
       printf("\nint_High_count=%d",int_count);
       old_count=int_count;
       }

    if (kbhit()!=0)
       {
       c=getch();
       if ((c=='q') || (c=='Q')) break;
       pci_tmc12_do(1); /* generate a High pulse to    */
       pci_tmc12_do(0); /* DO1=ECLK11=J25=int source */
       printf(" --> Generate a High interrupt pulse");
       }
    }

PTMC12_DriverClose();
_outpd(wPLX+0x4c,0);     /* disable all interrupt */
}
/* ------------------------------------------------------------ */

WORD init_interrupt()
{
DWORD dwVal;

int_count=0;
pci_tmc12_do2(0);          /* set IntXor OFF to non-invert the int source */
inport(wBaseAddr+0x18); /* pre-set int_signal_to_PC to High value    */
                                     /* to enable next interrupt operation        */

disable();

_outpd(wPLX+0x4c,0x41);      /* channel_1, interrupt active_Low                    */

if (wIrq<8)
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
   setvect(wIrq+8, irq_service);
   }
else
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & 0xfb);                        /* IRQ2 */
   outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
   irqmask=inportb(A2_8259+1);
   outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
   setvect(wIrq-8+0x70, irq_service);
   }

enable();
}

void interrupt irq_service()
{
/* now the int_signal_to_PC is in Low state */
inport(wBaseAddr+0x18);       /* pre-set int_signal_to_PC to High value */
                                     /* to enable next interrupt operation        */
int_count++;
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

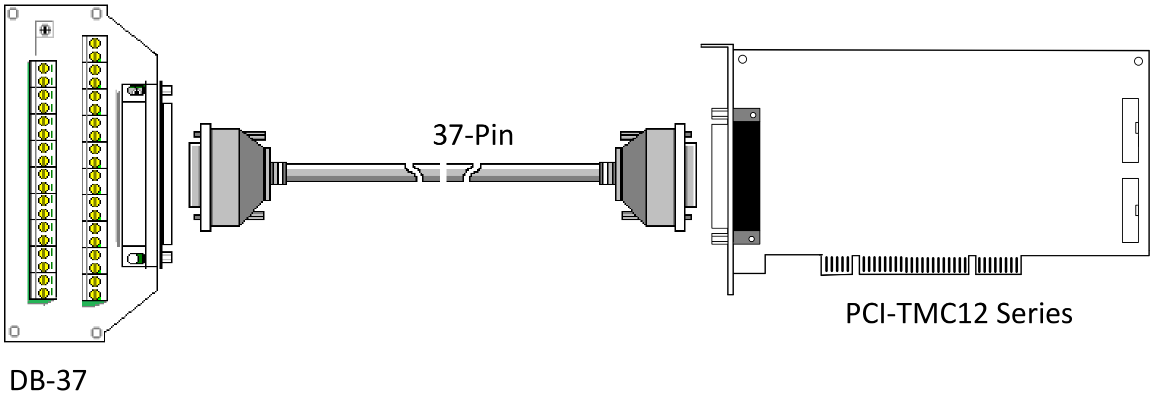
8.1.18 Ndemo5: Implementing an Active Low Interrupt

```
/* -------------------------------------------------------------------- */
/* ndemo5 : interrupt demo, int source=initial High, active Low               */
/* step 1 : connect DO1 (pin1 of CON3) to ECLK11 (pin16 of CON1) */
/* step 2 : J25 select EXT                    */
/* step 3 : run ndemo5.exe       */
/* step 4 : press any key to test, press Q to stop */
/* ------------------------------------------------------------------ */

#include "PCITMC12.H"

#define A1_8259 0x20
#define A2_8259 0xA0
#define EOI  0x20

WORD pci_tmc12_do(WORD wDo);
WORD pci_tmc12_do2(WORD wDo);
WORD init_interrupt();
WORD wBaseAddr,wIrq,wPLX,int_count;

static void interrupt irq_service();
int irqmask;

int main()
{
int    i,j;
WORD wBoards,wRetVal,old_count;
char c;
DWORD dwVal;

clrscr();
wRetVal=PTMC12_DriverInit(&wBoards);
printf("\n(1) Threr are %d PCI-TMC12 Cards in this PC",wBoards);
if ( wBoards==0 )
     {
     putch(0x07);    putch(0x07);    putch(0x07);
     printf("(1) There are no PCI-TMC12 card in this PC !!!\n"); exit(0);
     }

printf("\n(2) Show the Configuration Space of all PCI-TMC12:");
for(i=0; i<wBoards; i++)
     {
     PTMC12_GetConfigAddressSpace(i,&wBaseAddr,&wIrq,&wPLX);
     printf("\n       Card_%d: wBaseAddr=%x, wIrq=%x, wPLX=%x",i,wBaseAddr,wIrq,wPLX);
     }

PTMC12_GetConfigAddressSpace(0,&wBaseAddr,&wIrq,&wPLX); /* select card_0 */
printf("\n(3) *** Card_0, wBaseAddr=%x ***",wBaseAddr);

printf("\n(4) *** start test interrupt     **\n");

pci_tmc12_do(1); /* DO1=int source --> initial High, active Low */

init_interrupt();
old_count=1;
for (;;)
    {
    if (old_count != int_count)
       {
       printf("\nint_count=%d",int_count);
       old_count=int_count;
       }

    if (kbhit()!=0)
       {
       c=getch();
       if ((c=='q') || (c=='Q')) break;
       pci_tmc12_do(0); /* generate a Low   pulse to   */
       pci_tmc12_do(1); /* DO1=ECLK11=J25=int source */
       printf(" --> Generate a Low interrupt pulse");
       }
    }

PTMC12_DriverClose();
_outpd(wPLX+0x4c,0);     /* disable all interrupt */
}

/* ----------------------------------------------------------------- */

WORD init_interrupt()
{
DWORD dwVal;

int_count=0;
pci_tmc12_do2(0x1000); /* set IntXor On to invert the int source */
inport(wBaseAddr+0x18);/* pre-set int_signal_to_PC to High value */
                                  /* to enable next interrupt operation        */
disable();

_outpd(wPLX+0x4c,0x41);        /* channel_1, interrupt active_Low */

if (wIrq<8)
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
   setvect(wIrq+8, irq_service);
   }
else
   {
   irqmask=inportb(A1_8259+1);
   outportb(A1_8259+1,irqmask & 0xfb);                   /* IRQ2 */
   outportb(A1_8259+1,irqmask & (0xff ^ (1 << wIrq)));
   irqmask=inportb(A2_8259+1);
   outportb(A2_8259+1,irqmask & (0xff ^ (1 << (wIrq-8))));
   setvect(wIrq-8+0x70, irq_service);
   }

enable();
}

void interrupt irq_service()
{
/* now the int_signal_to_PC is in Low state */
inport(wBaseAddr+0x18); /* pre-set int_signal_to_pc to High value */
                                       /* to enable next interrupt operation        */
int_count++;
if (wIrq>=8) outportb(A2_8259,0x20);
outportb(A1_8259,0x20);
}
```

Appendix: Daughter Board

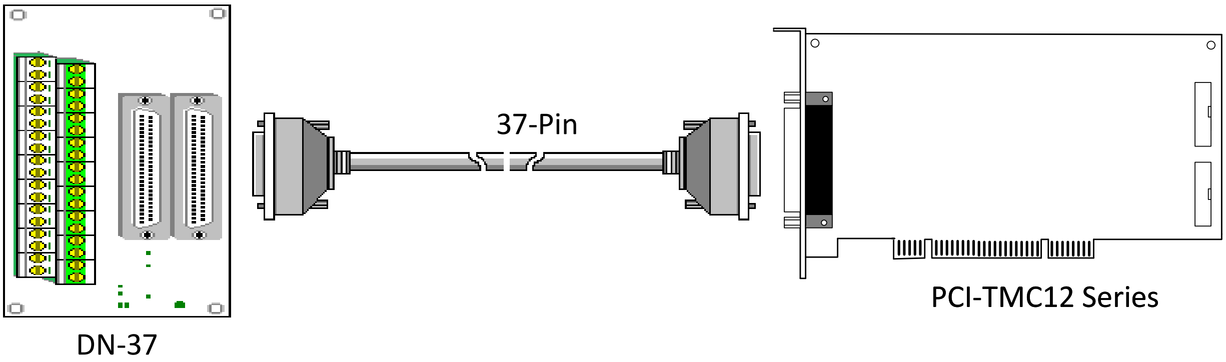
A1. DB-37

The DB-37 is a general purpose daughter board for D-sub 37 pins. It is designed for easy wire connection via pin-to-pin.



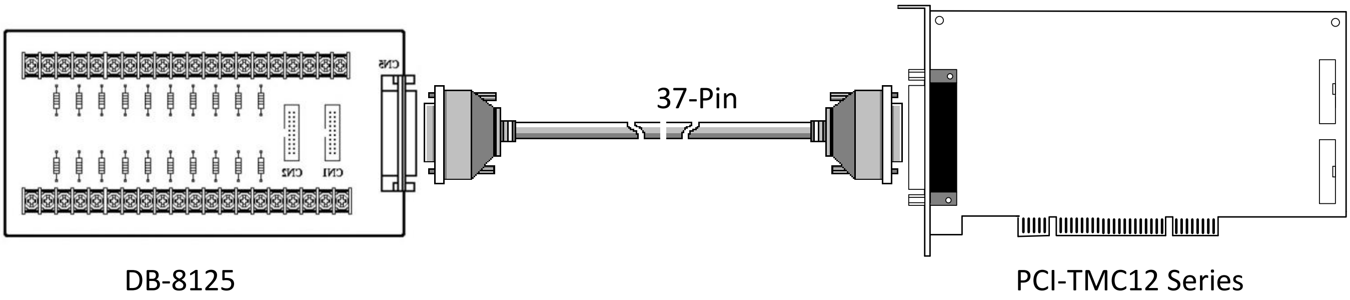
A2. DN-37 and DN-20

The DN-37 is a general-purpose daughter board for the DB-37. The DN-20 is designed for a 20-pin flat-cable. Both are designed for easy wiring, and are DIN-Rail mountable.



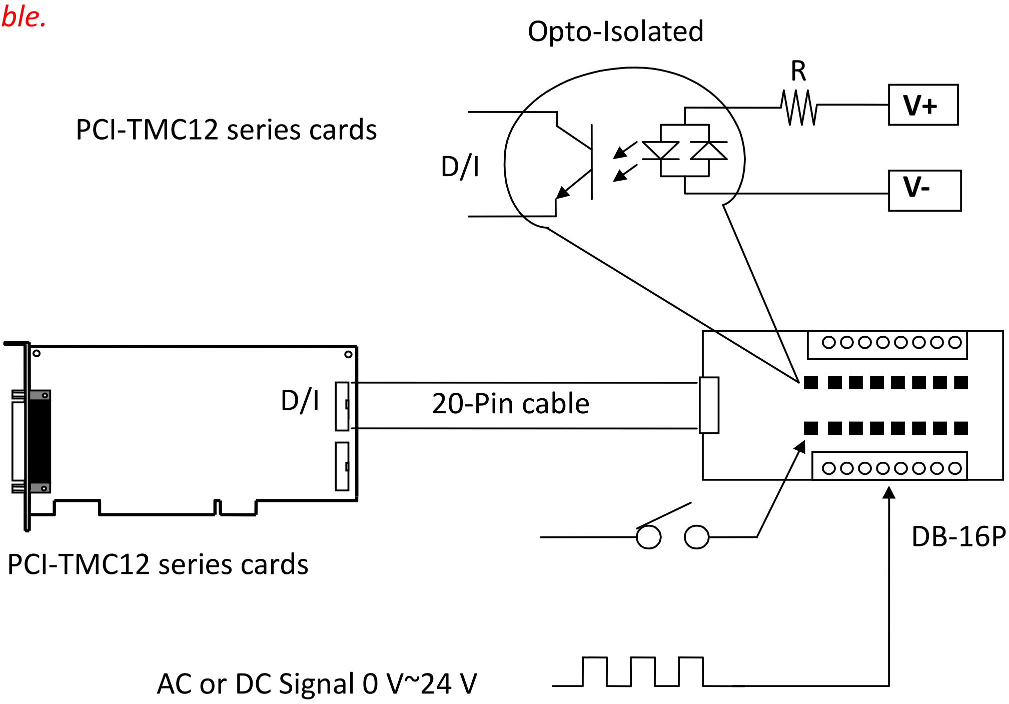
A3. DB-8125 and DB-8025

The DB-8125 is a general-purpose screw terminal board that is designed for easy wiring, and includes one DB-37 and two 20-pin flat-cable headers. The DB-8025 is designed only for a 20-pin flat-cable header.



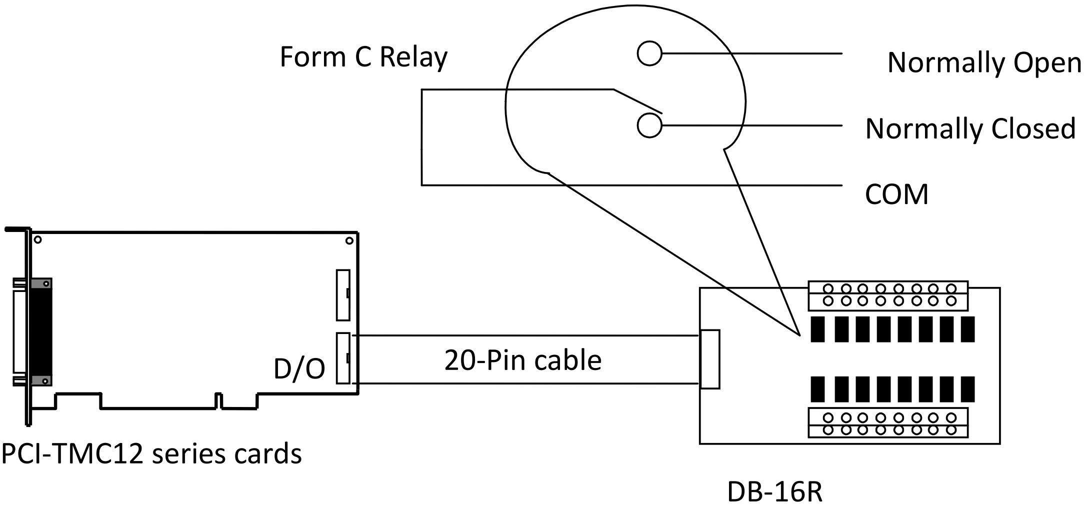
A4. DB-16P

The DB-16P is a 16-channel isolated Digital Input daughter board with optically isolated inputs that consist of a bi-directional optocoupler with a resistor to allow current sensing. The DB-16P can be used to sense DC signals from TTL levels up to +24 V, or can be used to sense a wide range of AC signals. This board can also be used to isolate the Host computer from high common-mode voltages, ground loops and transient voltage spikes that often occur in industrial environments. *Note: The lowest nibbles, bit_0 to bit_3, are used by the PCI-M512, so only the highest 12-bits, bit_4 to bit_15, are available.*



A5. DB-16R Relay Board

The DB-16R is a 16-channel Relay Output daughterboard that provides of 16 Form C Relay Output channels for that enable the efficient switching of loads through a programmable control. Both the connectors and the functionality are compatible with 785 series boards, and the DB-16R contains an industrial type terminal block. The relays are powered by applying a 5 V signal to the appropriate relay channel via the 20-pin flat cable connector. The DB-16R provides 16 LEDs, one for each relay, which are illuminated the associated relay is activated. To avoid overloading the power supply of the Host computer, the DB-16R includes a screw terminal to allow an external power supply to be connected.



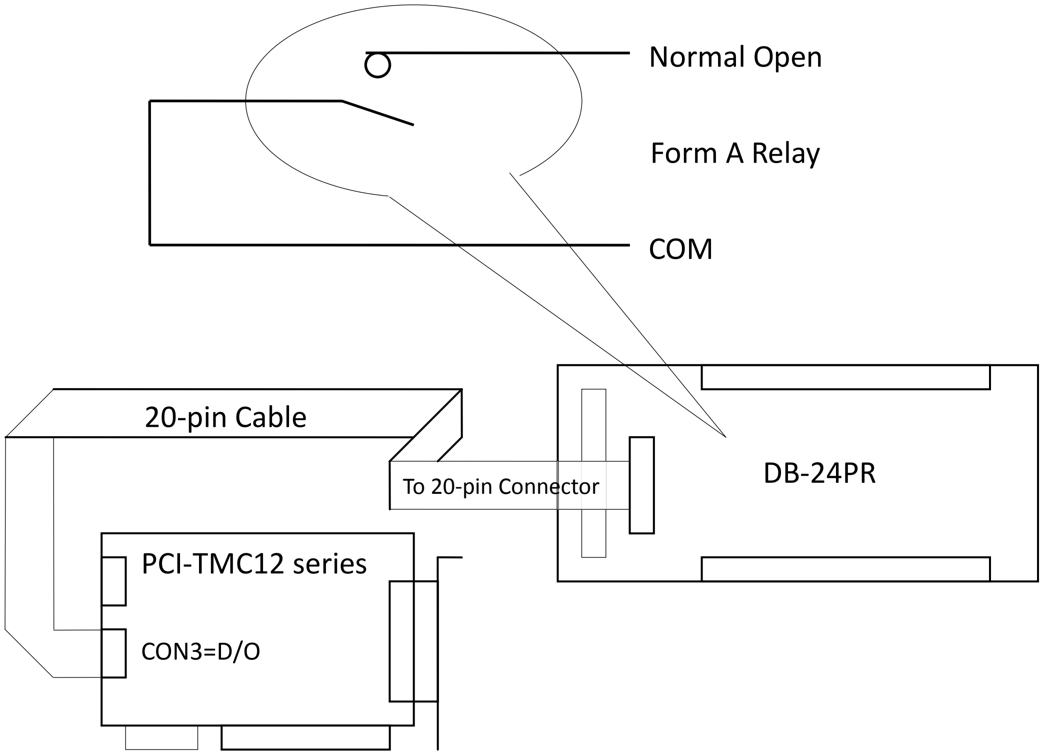
Note:

Channel: 16 Form C Relays
Relay: Switching up to 0.5 A @ 110 V_{AC}/ 1 A @ 24 V_{DC}

A6. DB-24PR, DB-24POR, DB-24C

DB-24PR	24 x Power Relay channels, 5A/250 V
DB-24POR	24 x PhotoMOS Relay channels, 0.1 A/350 V_{AC}
DB-24C	24 x Open Collectors, 100 mA per channel, 30 V Max.

The DB-24PR is a 24-channel Power Relay Output daughterboard that provides 8 Form C and 16 Form A electromechanical relay channels to enable the efficient switching of loads through a programmable control. The contact of each relay can be used to control a 5 A load at 250 V_{AC}/30 V_{DC}. The relays are powered by applying a 5 V signal to the appropriate relay channel via the 20-pin flat cable connector, which only uses 16 relays, or 50-pin flat cable connector, which is OPTO-22 compatible, for the DIO-24 series. The DB-24PR contains 24 LEDs, one for each relay, which are illuminated when the associated relay is activated. To avoid overloading the power supply of the Host PC, the DB-24R requires either a +12 V_{DC} or +24 V_{DC} external power supply to be connected.



Note:

1. The 50-pin connector (OPTO-22 compatible) is for use with DIO-24, DIO-48 and DIO-144 modules.

2. The 20-pin connector is for use with 16-channel Digital Output modules, including A-82X, A-62X, DIO-64 and ISO-DA16/DA8.

3. Channels: 16 Form A Relays, 8 Form C Relays

4. Relays: Switching up to 5 A @ 110 V_{AC} / 5 A @ 30 V_{DC}