

PCI-TMC12(A)

User's Manual

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 assume 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

Copyright 2002 by ICP DAS. All rights are reserved.

Trademark

The names used for identification only maybe registered trademarks of their respective companies.

Tables of Contents

1. INTRODUCTION	4
1.1 FEATURES	4
1.2 SPECIFICATIONS	5
1.3 PCI-TMC12 & PCI-TMC12A	6
1.4 PRODUCT CHECK LIST	6
2. HARDWARE CONFIGURATION	7
2.1 BOARD LAYOUT	7
2.2 COUNTER ARCHITECTURE	8
2.3 D/I/O BLOCK DIAGRAM	9
2.4 JUMPER SETTING	10
2.4.1 <i>CLOCK1 & CLOCK2</i>	10
2.4.2 <i>CLK1 to CLK12</i>	11
2.4.3 <i>GATE1 TO GATE12</i>	12
2.4.4 <i>J25: Interrupt Source Selection</i>	13
2.5 DAUGHTER BOARDS	14
2.5.1 <i>DB37</i>	14
2.5.2 <i>DN37 & DN20</i>	14
2.5.3 <i>DB-8125 & DB-8025</i>	14
2.5.4 <i>DB-16P Isolated Input Board</i>	15
2.5.5 <i>DB-16R Relay Board</i>	16
2.5.6 <i>DB-24PR, DB-24POR, DB-24C</i>	17
2.6 PIN ASSIGNMENT	18
3. I/O CONTROL REGISTER	20
3.1 HOW TO FIND THE I/O ADDRESS	20
3.2 THE ASSIGNMENT OF I/O ADDRESS	22
3.3 THE I/O ADDRESS MAP	23
3.3.1 <i>Activate a 8254 chip</i>	23
3.3.2 <i>8254 Timer/Counter Control</i>	24
3.3.3 <i>Digital Input</i>	24
3.3.4 <i>Digital Output</i>	25
3.3.5 <i>Interrupt control/status register of PCI-TMC12</i>	25
3.4 NEW FEATURES OF PCI-TMC12A	28
3.4.1 <i>Default Shipping of PCI-TMC12A</i>	28
3.4.2 <i>Clock input of 8254</i>	29
3.4.3 <i>Xor-control Register of PCI-TMC12A</i>	30

3.4.4	<i>Block Diagram of Interrupt System</i>	31
3.4.5	<i>New Demo Program</i>	32
4.	8254 PROGRAMMING	33
4.1	CONTROL WORD FORMAT	33
4.2	COUNTER LATCH COMMAND	34
4.3	READ BACK COMMAND	34
4.4	STATUS BYTE FORMAT	34
5.	DEMO PROGRAM	35
5.1	DEMO1: USE D/O	36
5.2	DEMO2: USE D/I	37
5.3	DEMO3: WAVE GENERATOR	38
5.4	DEMO4: DELAY ONE MS	40
5.5	DEMO5: 16-BIT EVENT COUNTER	41
5.6	DEMO6: SOFTWARE COUNTER	42
5.7	DEMO7: WATCHDOG TIMER	43
5.8	DEMO8: PULSE WIDTH MEASURE	45
5.9	DEMO9: FREQUENCY MEASURE	47
5.10	DEMO10: FIND CARD NUMBER	49
5.11	DEMO11: COUNT LOW PULSE	50
5.12	DEMO12: LOW PULSE WIDTH	52
5.13	DEMO13: HIGH PULSE WIDTH	55
5.14	NDEMO1: USING LEDs	57
5.15	NDEMO2: GENERATE 2 CLOCKS	58
5.16	NDEMO3: NEW DEMO7	60
5.17	NDEMO4: ACTIVE HIGH INT	62
5.18	NDEMO5: ACTIVE LOW INT	64

1. Introduction

The PCI-TMC12 (A) is a PCI-Bus general purpose timer / counter and digital I/O board. It provides twelve channels of 16-bit timer / counter. Two internal clock source (8 M/1.6 M and 0.8 M/80 k) are selected by jumper. Four 8254 chips provides a variety of powerful timer / counter function modes to match your industrial and laboratory applications.

1.1 Features

- The PCI-TMC12 & PCI-TMC12A is a general purpose counter/timer and digital I/O card
- PC AT compatible PCI bus
- On-board four 8254 chips
- 5 different interrupt sources, 4 internal + 1 external, jumper selectable
- Flexible clock sources and gate control signals selectable
- 2 stable internal clock sources, CLOCK1=8 M/1.6 M, CLOCK2=0.8 M/80 k, jumper selectable
- 12 external clock sources
- 12 external gate control signals
- 16 bits general purpose TTL-compatible D/O or relay (with daughter board DB-16R or DB-24PR)
- 16 bits general propose TTL-compatible D/I or isolated input (with daughter board DB-16P)
- 12 independent 16 bits timer/counter

Note: **PCI_TMC12(A)** = PCI-TMC12 or PCI-TMC12A

1.2 Specifications

Model Name		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	5 V PCI, 32-bit, 33 MHz	
Data Bus	16-bit	
Card ID	No	
I/O Connector	Female DB37 x 1 20-pin 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	

1.3 PCI-TMC12 & PCI-TMC12A

- All old program designed for PCI-TMC12 can be executed on PCI-TMC12A without any modification
- PCI-TMC12A provides more features, refer to Sec. 3.4 for more information.

1.4 Product Check List

Your package includes the following items:

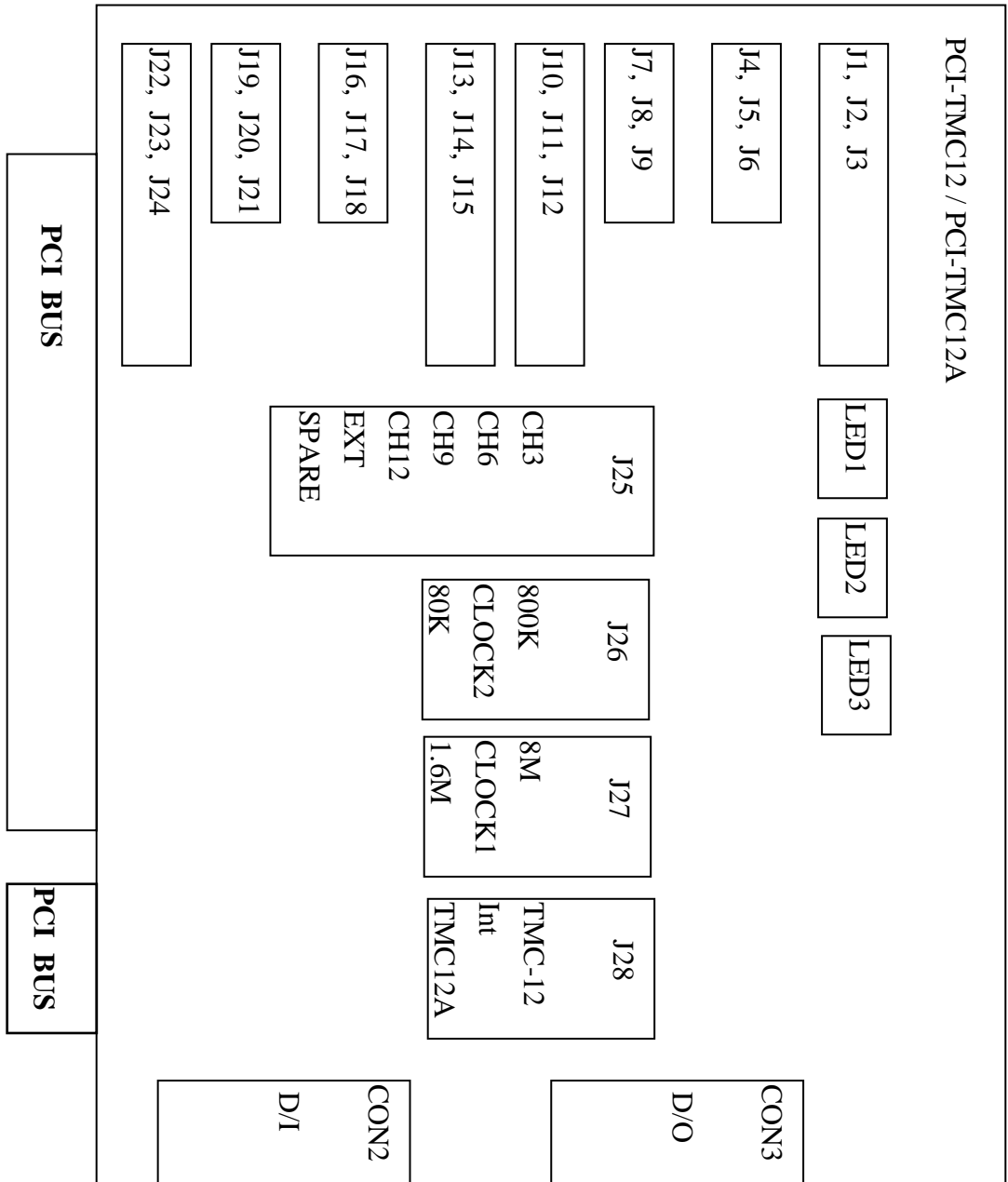
- One PCI-TMC12(A) card
- One companion CD for software driver
- One Quick Start Guide

Attention !

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

2. Hardware configuration

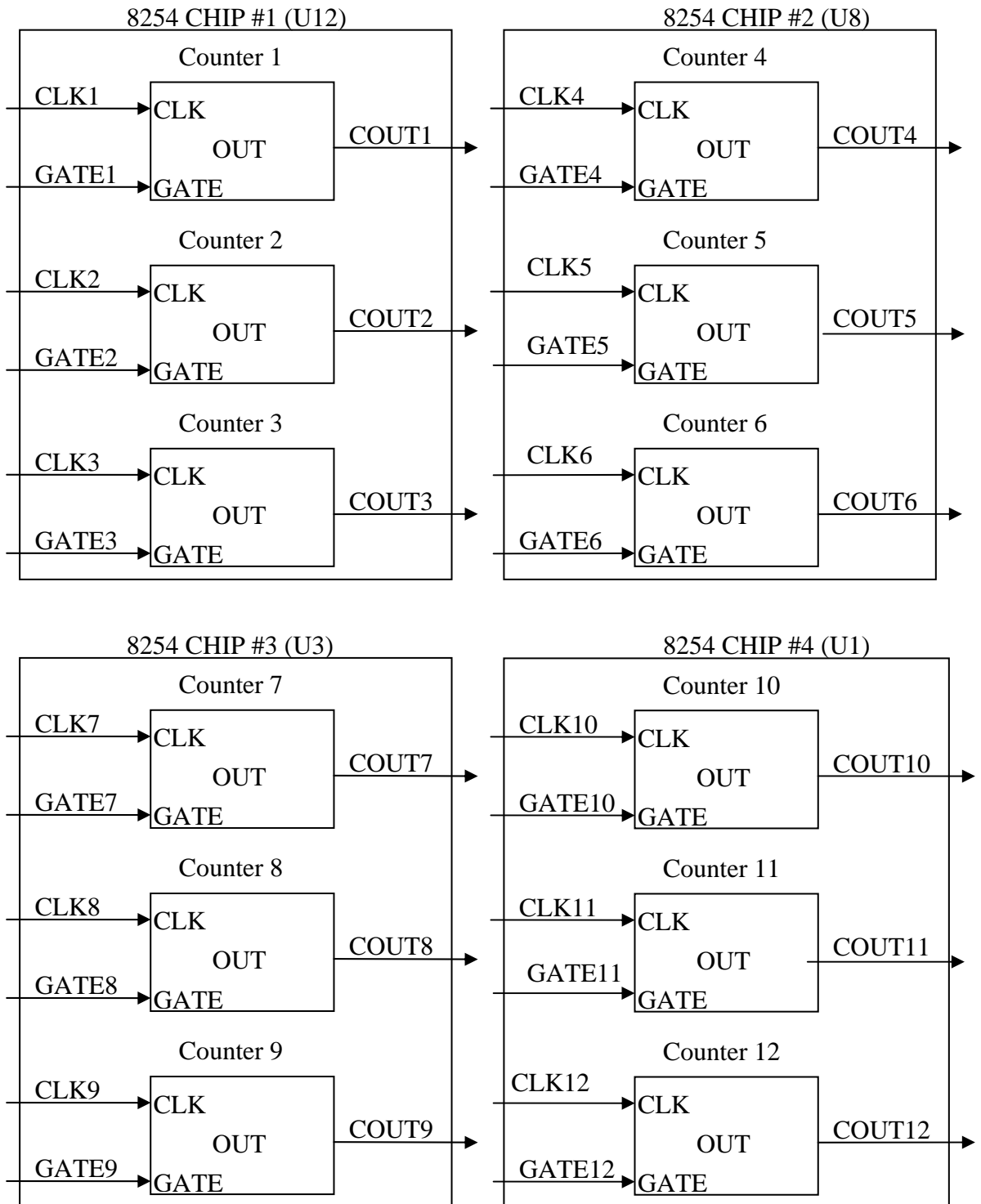
2.1 Board Layout



Note: **J28, LED1, LED2 & LED3** are designed for PCI-TMC12A only.

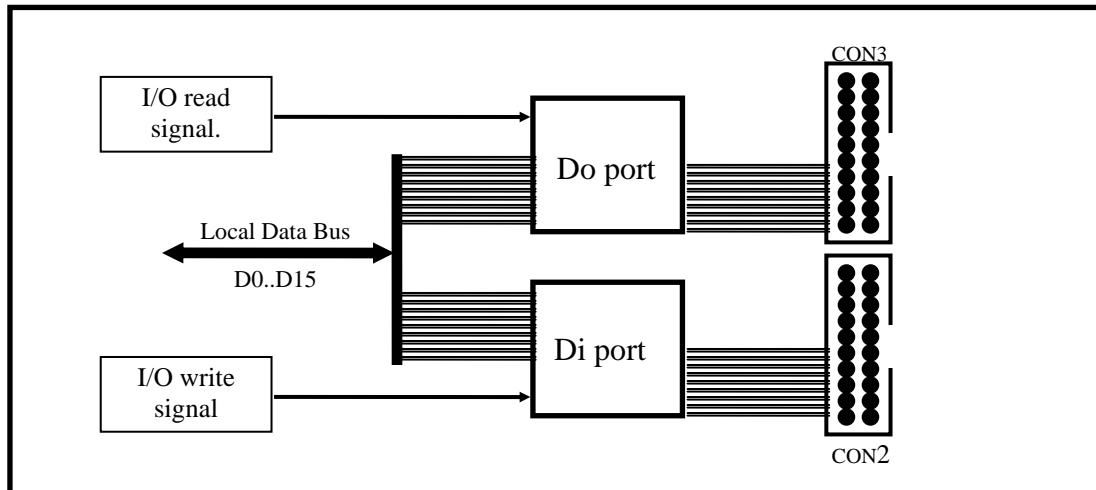
2.2 Counter Architecture

There are four 8254 chips on the PCI-TMC12(A) card. The block diagram is given as following:



2.3 D/I/O Block Diagram

The PCI-TMC12(A) provide 16-channel digital input and 16-channel digital output. All levels are TTL compatible. The connections diagram and block diagram are given as following:

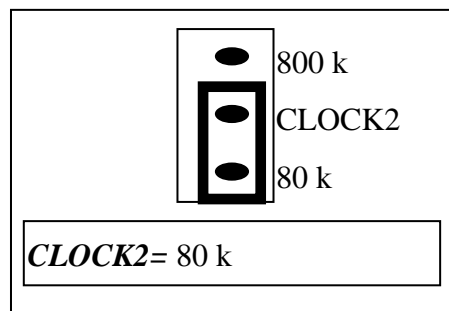
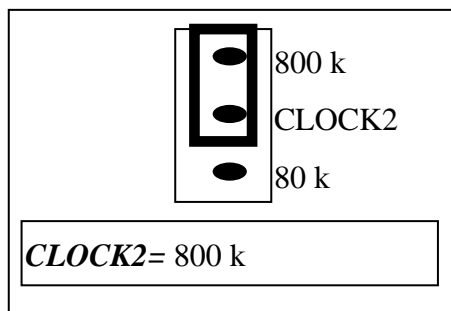
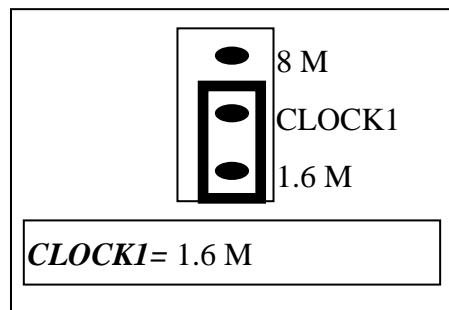
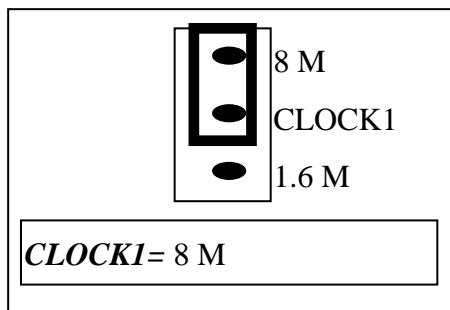
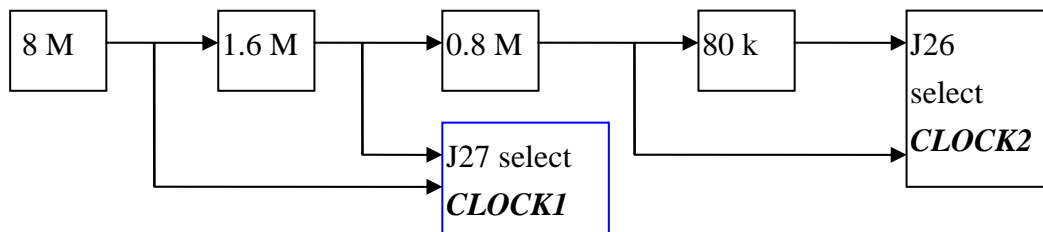


The D/I port can be connected to the DB-16P. The DB-16P is a 16-channel isolated digital input daughter board. The D/O port can be connected to the DB-16R or DB-24PR. The DB-16R is a 16-channel relay output board. The DB-24R is a 24-channel power relay output board.

2.4 Jumper Setting

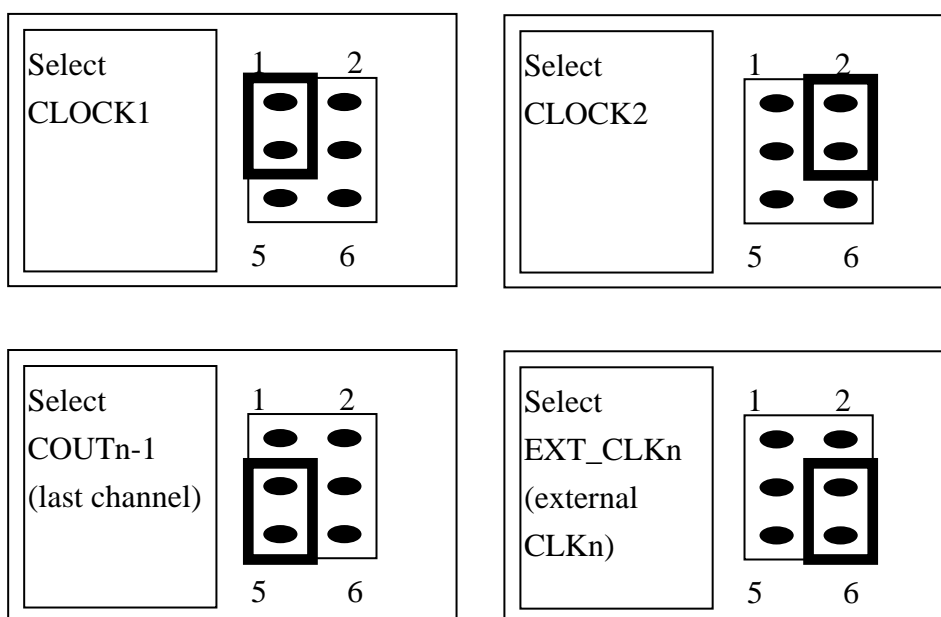
2.4.1 CLOCK1 & CLOCK2

There are two stable internal clock sources in PCI-TMC12(A) which named as **CLOCK1** & **CLOCK2**. The **CLOCK1** may be 8 M or 1.6 M selectable by J27. The **CLOCK2** may be 0.8 M or 80 k selected by J26. The block diagram of internal clock sources is given as following:



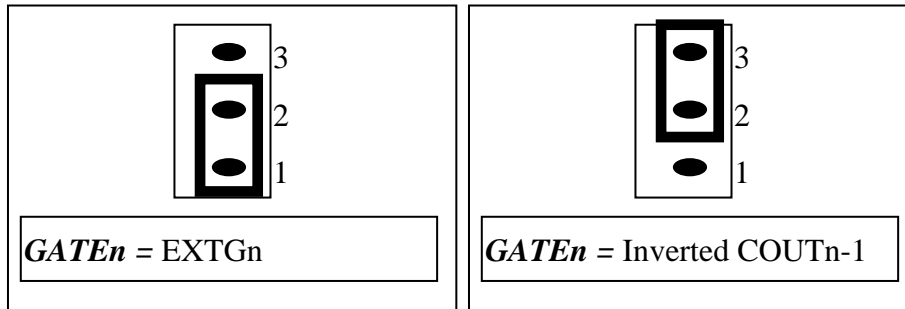
2.4.2 CLK1 to CLK12

- 1: select CLOCK1
- 2: select CLOCK2
- 5: select COUTn-1
- 6: select external CLKn from CN1

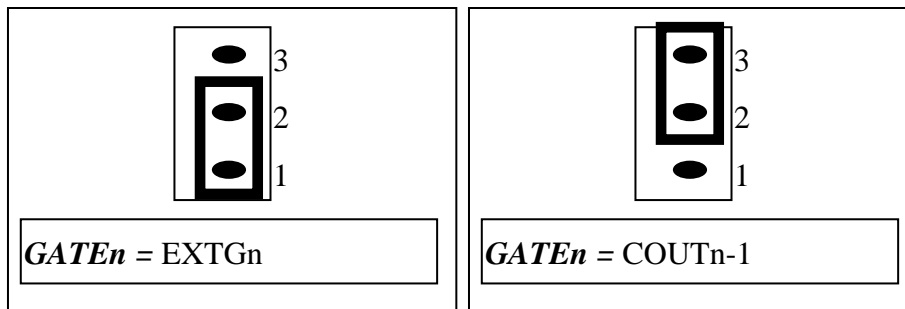


CLK1-12	jumper	Select sources
CLK1	JP22	CLOCK1, CLOCK2, COUT6, ECLK1
CLK2	JP23	CLOCK1, CLOCK2, COUT1, ECLK2
CLK3	JP24	CLOCK1, CLOCK2, COUT2, ECLK3
CLK4	JP13	CLOCK1, CLOCK2, COUT3, ECLK4
CLK5	JP14	CLOCK1, CLOCK2, COUT4, ECLK5
CLK6	JP15	CLOCK1, CLOCK2, COUT5, ECLK6
CLK7	JP10	CLOCK1, CLOCK2, COUT12, ECLK7
CLK8	JP11	CLOCK1, CLOCK2, COUT7, ECLK8
CLK9	JP12	CLOCK1, CLOCK2, COUT8, ECLK9
CLK10	JP1	CLOCK1, CLOCK2, COUT9, ECLK10
CLK11	JP2	CLOCK1, CLOCK2, COUT10, ECLK11
CLK12	JP3	CLOCK1, CLOCK2, COUT11, ECLK12

2.4.3 GATE1 TO GATE12



GATE	Jumper	Select source
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
GETE6	J18	Inverted COUT5, EXTG6



GATE	Jumper	Select source
GATE7	J7	COUT12, EXTG7
GATE8	J8	COUT7, EXTG8
GATE9	J9	COUT8, EXTG9
GATE10	J4	COUT9, EXTG10
GATE11	J5	COUT10, EXTG11
GETE12	J6	COUT11, EXTG12

2.4.4 J25: Interrupt Source Selection

There are five signals can be used as interrupt sources: CH3, CH6, CH9, CH12 & EXT as following:

CH3: comes from COUT3, output of counter 3

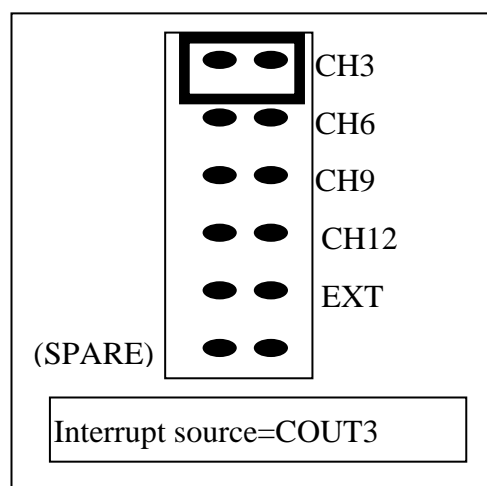
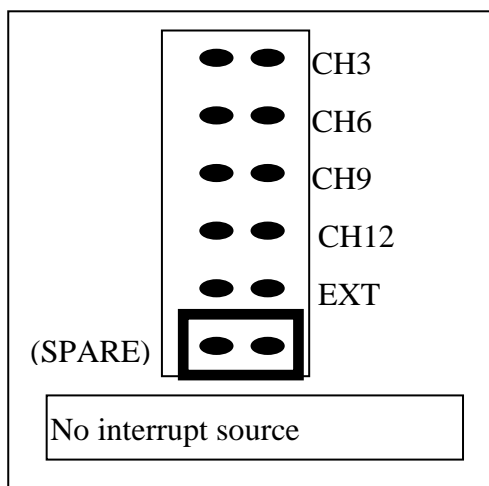
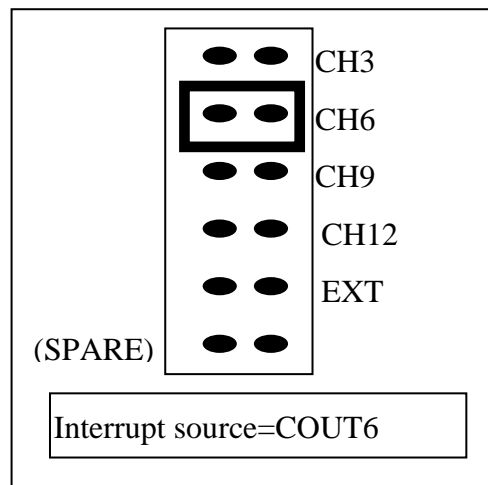
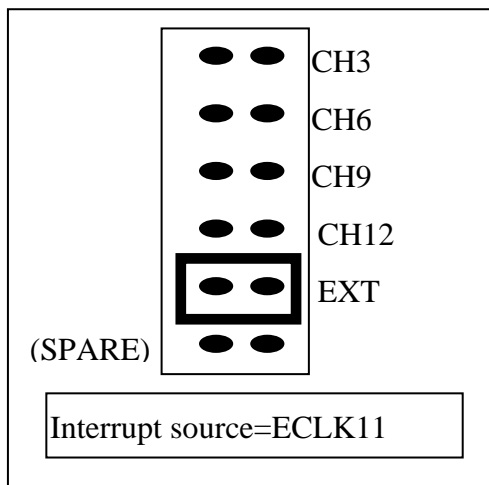
CH6: comes from COUT6, output of counter 6

CH9: comes from COUT9, output of counter 9

CH12: comes from COUT12, output of counter 12

EXT: comes from ECLK11, external CLK for counter 11, from CN1.

(SPARE): no interrupt source



2.5 Daughter Boards

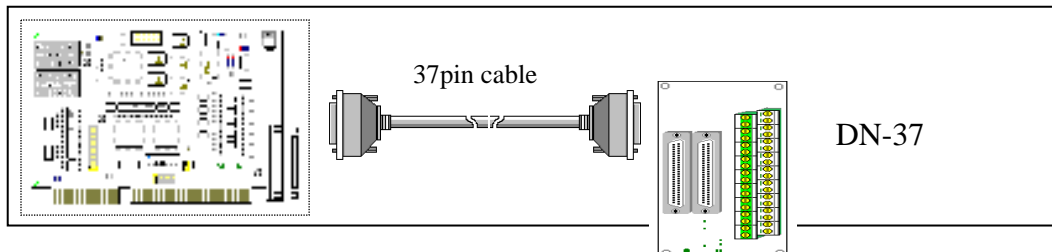
2.5.1 DB37

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



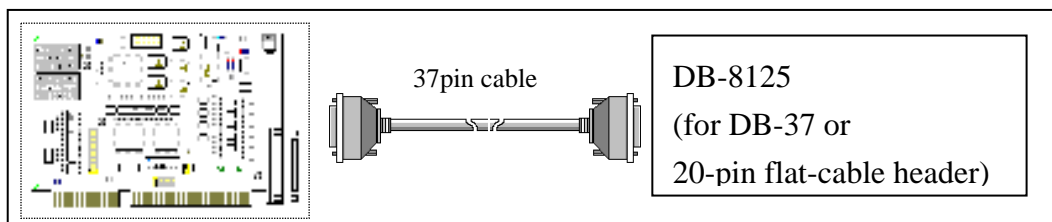
2.5.2 DN37 & DN20

The DN-37 is a general purpose daughter board for DB-37. The DN-20 is designed for 20-pin flat-cable. They are designed for easy wire connection. They are Din-Rail mounting.



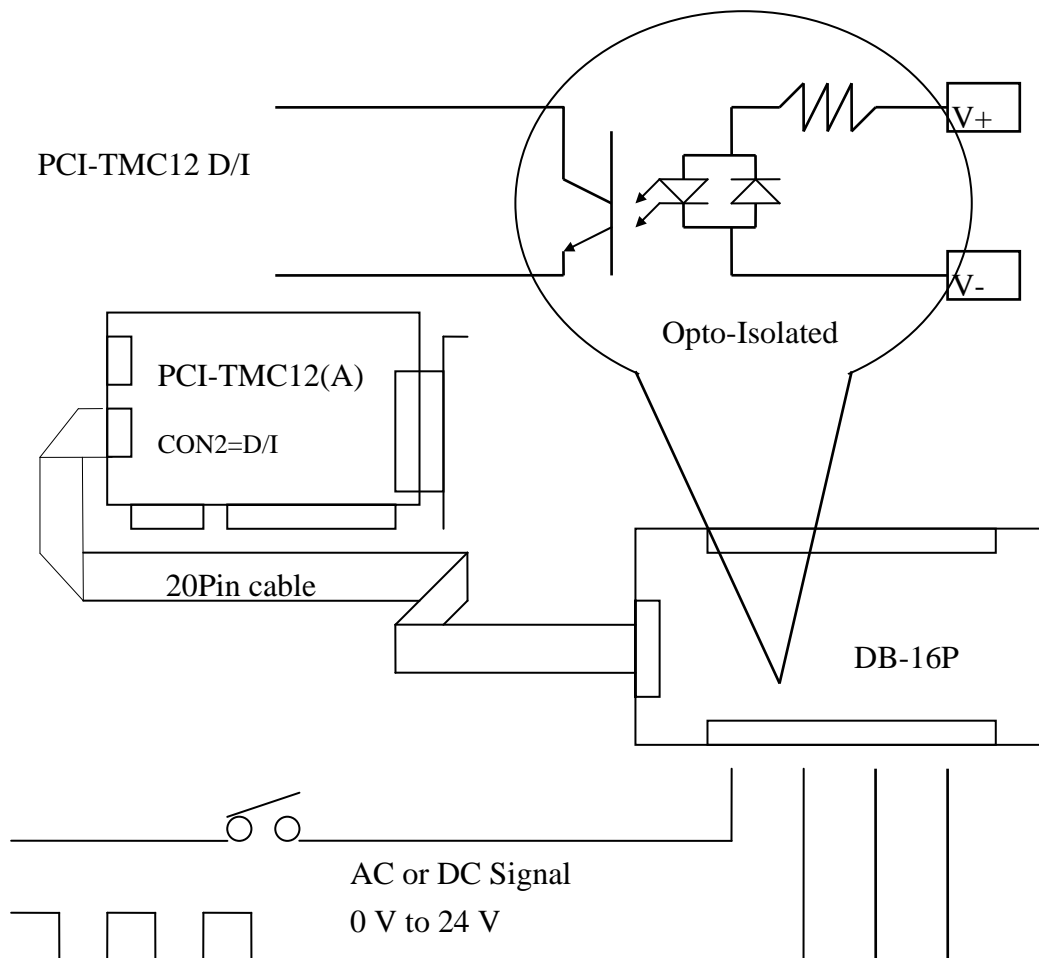
2.5.3 DB-8125 & DB-8025

The DB-8125 is a general purpose screw terminal board. It is designed for easy wire connection. There are one DB-37 & two 20-pin flat-cable header in the DB-8125. The DB-8025 is designed for 20-pin flat-cable header.



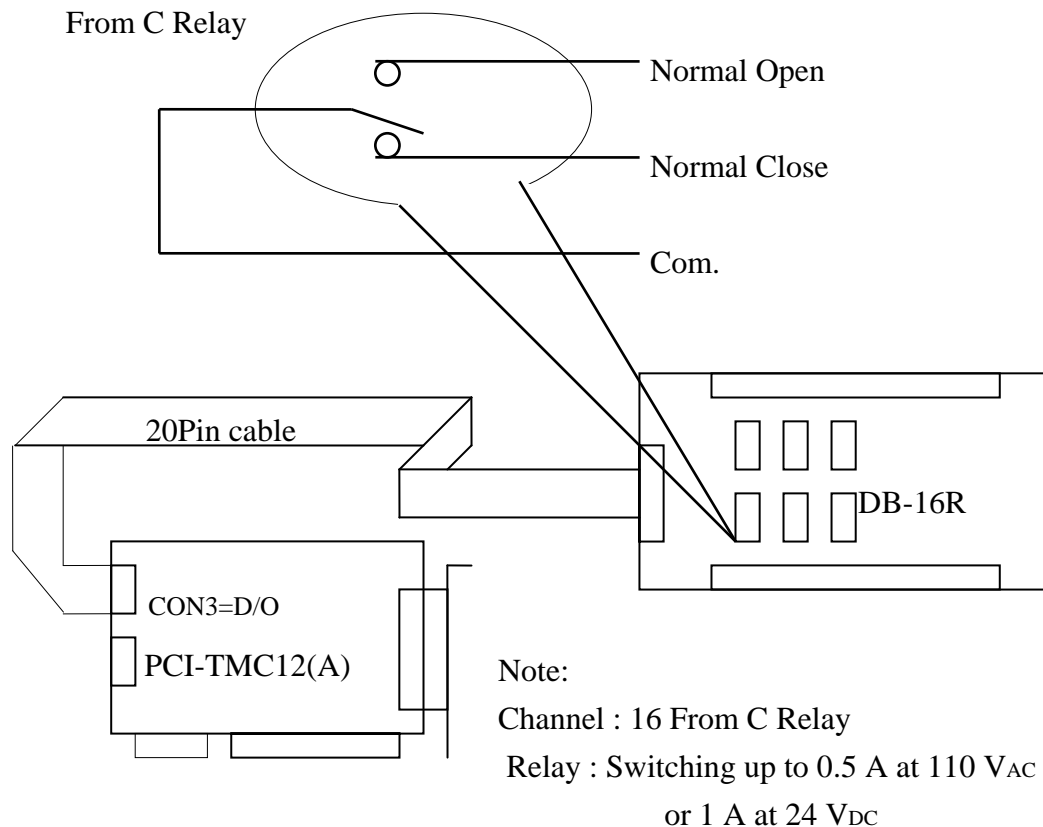
2.5.4 DB-16P Isolated Input Board

The DB-16P is a 16-channel isolated digital input daughter board. The optically isolated inputs of the DB-16P consists of a bi-directional optocoupler with a resistor for current sensing. You can use the DB-16P to sense DC signal from TTL levels up to 24 V or use the DB-16P to sense a wide range of AC signals. You can use this board to isolated the computer from large common-mode voltage, ground loops and transient voltage spike that often occur in industrial environments.



2.5.5 DB-16R Relay Board

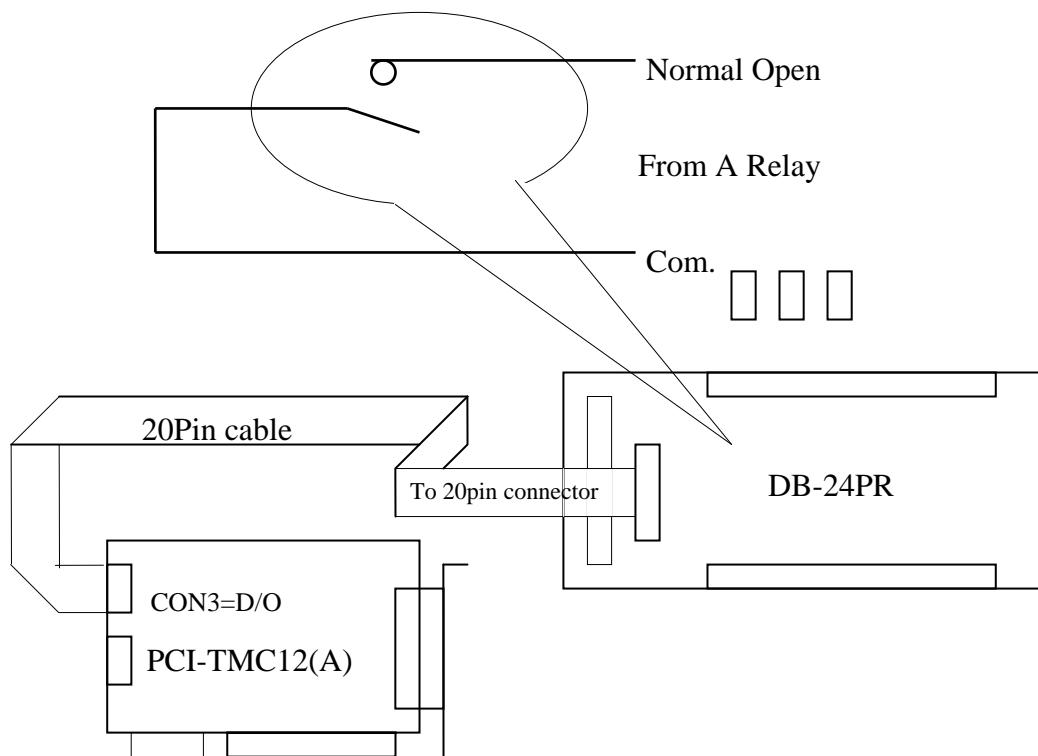
The DB-16R, 16-channel relay output board, consists of 16 form C relays for efficient switch of load by programmed control. It is connector and functionally compatible with 785 series board but with industrial type terminal block. The relays are energized by applying a 5V signal to the appropriate relay channel on the 20-pin flat connector. There are 16 enunciator LEDs for each relay, light when their associated relay is activated. To avoid overloading your PC's power supply, this board provides a screw terminal for external power supply.



2.5.6 DB-24PR, DB-24POR, DB-24C

DB-24PR	24*power relay, 5 A/250 V
DB-24POR	24*photo MOS relay, 0.1 A/350 V _{AC}
DB-24C	24*open collector, 100 mA per channel, 30 V max.

The DB-24PR, 24-channel power relay output board, consists of 8 form C and 16 form A electromechanical relays for efficient switching of load programmed control. The contact of each relay can control a 5 A load at 250 V_{AC}/30V_{DC}. The relay is energized by applying a 5 voltage signal to the appropriate relay channel on the 20-pin flat cable connector(just used 16 relays) or 50-pin flat cable connector.(OPTO-22 compatible, for DIO-24 series). Twenty - four enunciator LEDs, one for each relay, light when their associated relay is activated. To avoid overloading your PC' s power supply , this board needs a +12 V_{DC} or +24 V_{DC} external power supply.



Note:

50-Pin connector(OPTO-22 compatible), for DIO-24, DIO-48, DIO-144

20-Pin connector for 16 channel digital output, A-82X, A-62X, DIO-64, ISO-DA16/DA8

Channel : 16 From A Relay , 8 From C Relay

Relay : switching up to 5 A at 110 V_{AC} / 5 A at 30 V_{DC}

2.6 Pin Assignment

The CON1 is a 37 pin D-type female connector.

Pin Number	Description	Pin Number	Description
1	ECLK1	20	EXTG1
2	COUT1	21	ECLK2
3	EXTG2	22	COUT2
4	ECLK3	23	EXTG3
5	COUT3	24	ECLK4
6	EXTG4	25	COUT4
7	ECLK5	26	EXTG5
8	COUT5	27	ECLK6
9	EXTG6	28	COUT6
10	ECLK7	29	EXTG7
11	COUT7	30	ECLK8
12	EXTG8	31	COUT8
13	ECLK9	32	EXTG9
14	COUT9	33	ECLK10
15	EXTG10	34	COUT10
16	ECLK11	35	EXTG11
17	COUT11	36	ECLK12
18	EXTG12	37	COUT12
19	GND	XXXXXXX	This pin not available

ECLKn: external clock source for counter n

EXTGn: external gate control signal for counter n

COUTn: output of timer/counter n

All signals are TTL compatible.

CON2: pin assignment of digital input connector.

Pin	Name	Pin	Name
1	Digital input 0	2	Digital input 1
3	Digital input 2	4	Digital input 3
5	Digital input 4	6	Digital input 5
17	Digital input 6	8	Digital input 7
9	Digital input 8	10	Digital input 9
11	Digital input 10	12	Digital input 11
13	Digital input 12	14	Digital input 13
15	Digital input 14	16	Digital input 15
17	PCB ground	18	PCB ground
19	PCB +5 V	20	PCB +12 V

CON3: pin assignment of the digital output connector.

Pin	Name	Pin	Name
1	Digital output 0	2	Digital output 1
3	Digital output 2	4	Digital output 3
5	Digital output 4	6	Digital output 5
17	Digital output 6	8	Digital output 7
9	Digital output 8	10	Digital output 9
11	Digital output 10	12	Digital output 11
13	Digital output 12	14	Digital output 13
15	Digital output 14	16	Digital output 15
17	PCB ground	18	PCB ground
19	PCB +5 V	20	PCB +12 V

3. I/O Control Register

3.1 How to Find the I/O Address

The plug&play BIOS will assign a proper I/O address to every PCI-TMC12(A) card in the power-on stage. The Ids of PCI-TMC12(A) are given as following:

- **Vendor ID = 10B5**
- **Device ID = 9050**
- **Sub-vendor ID= 2129**
- **Sub-device ID = 9912**

We provide all necessary functions as following:

1. **PTMC12_DriverInit(&wBoard)**

This function can detect how many PCI-TMC12(A) cards in the system. It is implemented based on the PCI plug&play mechanism-1. It will find all PCI-TMC12(A) cards installed in this system & save all their resource in the library.

- wBoard=1 → only one PCI-TMC12(A) in this PC system.
- wBoard=2 → there are two PCI-TMC12(A) in this PC system.

2. **PTMC12_GetConfigAddressSpace(wBoardNo,*wBase,*wIrq,*wPLX)**

The user can use this function to save resource of all PCI-TMC12(A) installed in this system. Then the application program can control all functions of PCI-TMC12(A) directly.

- wBoardNo=0 to N → totally N+1 cards of PCI-TMC12(A)
- wBase → base address of the board control word
- wIrq → allocated IRQ channel number of this board
- wPLX → base address of PCI-interface-IC

The sample program source is given as following:

```
/* step1: detect all PCI-TMC12(A) card first */
wRetVal=PTMC12_DriverInit(&wBoards);
printf("Threr are %d PCI-TMC12 Cards in this PC\n",wBoards);

/* step2: save resource of all PCI-TMC12(A) cards installed in this PC */
for (i=0; i<wBoards; i++)
{
PTMC12_GetConfigAddressSpace(i,&wBase,&wIrq,&wPLX);
printf("\nCard_%d: wBase=%x, wIrq=%x, wPLX=%x", i,wBase,wIrq,wPLX);
wConfigSpace[i][0]=wBaseAddress; /* save all resource of this card */
wConfigSpace[i][1]=wIrq;          /* save all resource of this card */
wConfigSpace[i][2]=wPLX;          /* save all resource of this card */
}

/* step3: control the PCI-TMC12(A) directly */
wBase=wConfigSpace[0][0];          /* get base address the card_0          */
outport(wBase+0x14,wDoValue);      /* control the D/O states of card_0 */
wDiValue=inport(wBase+0x14);      /* read the D/I states of card_0    */

wBase=wConfigSpace[1][0];          /* get base address of card_1          */
outport(wBase+0x14,wDoValue);      /* control the D/O states of card_1 */
wDiValue=inport(wBase+0x14);      /* read the D/I states of card_1    */

wPLX=wConfigSpace[2][2];          /* get PCI-interface base address of card-
2 */
_outpd(wPLX+0x4c,0x41);           /* channel_1, interrupt active_Low   */
..
..
_outpd(wPLX+0x4c,0);              /* disable all interrupt              */
```

3.2 The Assignment of I/O Address

The plug&play BIOS will assign the proper I/O address to PCI-TMC12. If there is only one PCI-TMC12, the user can identify the board as card_0. If there are two PCI-TMC12 cards in the system, the user will be very difficult to identify which board is card_0? The software driver can support 16 boards max. Therefore the user can install 16 boards of PCI-TMC12 in one PC system. How to find the card_0 & card_1 ?

The simplest way to find the card number is to use DEM10.EXE given in DOS demo program. This demo program will send a value to D/O and read back from D/I. If the user install a 20-pin flat cable between CON2 & CON3, the value read from D/I will be the same as D/O. The operation steps are given as following:

Remove all 20-pin flat cable between CON2 and CON3

Install all PCI-TMC12 cards into this PC system

Power-on and run DEM10.EXE

Now all D/I value will be different from D/O value

Install a 20-pin flat cable into CON2 & CON3 of any PCI-TMC12 card

There will be one card's D/I value = D/O value, the card number is also show in screen

Therefore the user can find the card number very easy if he install a 20-pin flat cable into PCI-TMC12 one-by-one.

3.3 The I/O Address Map

The I/O address of PCI-TMC12(A) is automatically assigned by the main board ROM BIOS. The I/O address can also be re-assigned by user. **It is strongly recommended not to change the I/O address by user. The plug&play BIOS will assign proper I/O address to each PCI-TMC12(A) very well.** The hardware I/O ports are described as following:

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 channel 0-15	Digital output channel 0-15
wBase+0x18	New control of PCI-TMC12A	Interrupt clear of PCI-TMC12A

Note. Refer to Sec. 3.1 for more information about wBase.

3.3.1 Activate a 8254 chip

There are four 8254 chips working simultaneously on PCI-TMC12(A) card, but only one 8254 chip can be activated for reading/writing at a time. Before accessing a 8254, use wBase+0x10 to set it as the active chip.

(WRITE) wBase+0x10: select 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

Note. Refer to Sec. 3.1 for more information about wBase.

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

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

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

3.3.2 8254 Timer/Counter Control

There are four 8254 chips working simultaneously on PCI-TMC12(A) card, but only one 8254 can be activated for reading/writing at a time. Before accessing an 8254, use wBase+0x10 to set it as the active chip. (Sec. 3.3.1)

The 8254 has 4 registers addressed from wBase+0 to wBase+0x0C, users can access the registers to control and monitor the Timer/Counter on the active 8254 chip. For more information, please refer to Chapter 4 and 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

Note. Refer to Sec. 3.1 for more information about wBase.

3.3.3 Digital Input

(READ) wBase+0x14: read the digital input channel 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

Note. Refer to Sec. 3.1 for more information about wBase.

```
wDiValue=inport(wBase+0x14); /* read the D/I states */
```

3.3.4 Digital Output

(WRITE) wBase+0x14: set the digital output channel 0 to 15

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DO7	DO6	DO5	DO4	DO3	DO2	DO1	DO0

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
DO15	DO14	DO13	DO12	DO11	DO10	DO9	DO8

Note. Refer to Sec. 3.1 for more information about wBase.

```
output(wBase+0x14,wDoValue); /* control the D/O states */
```

3.3.5 Interrupt control/status register of PCI-

TMC12

(READ/WRITE) wPLX+0x4C: interrupt control/status register

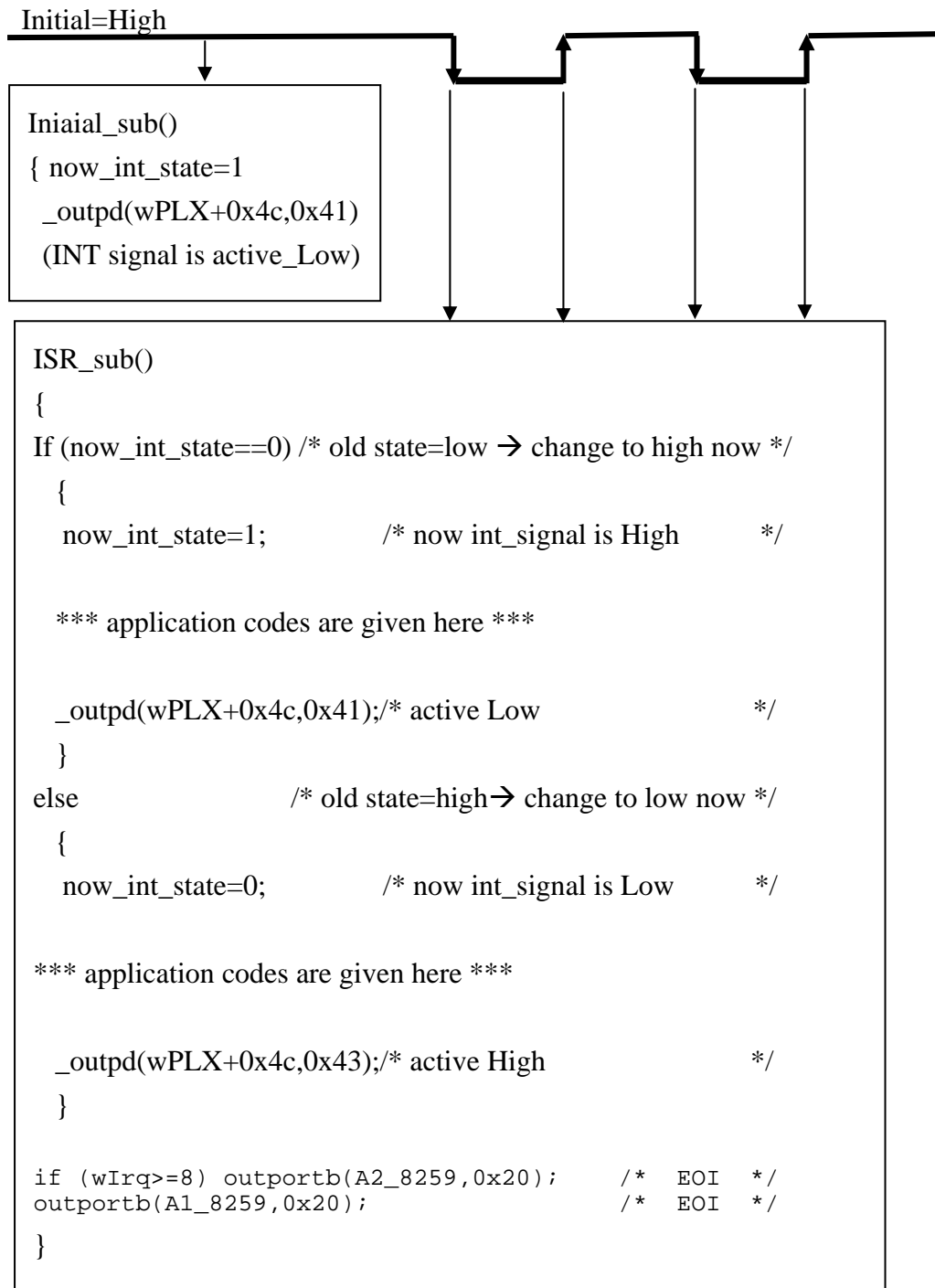
Bit	Description
B0	INTERRUPT enable, 0=disable, 1=enable
B1	POLARITY, 1=active HIGH, 0=active LOW
B2	INTERRUPT status, 0=int not active, 1=int is active
B3	reserved
B4	reserved
B5	reserved
B6	PCI interrupt enable, 0=disable, 1=enable
B7	Software interrupt, a value of 1 will generate interrupt
B8 to B31	reserved

Refer to DEMO7.C, DEMO11.C, DEMO12.C & DEMO13.C for more information.

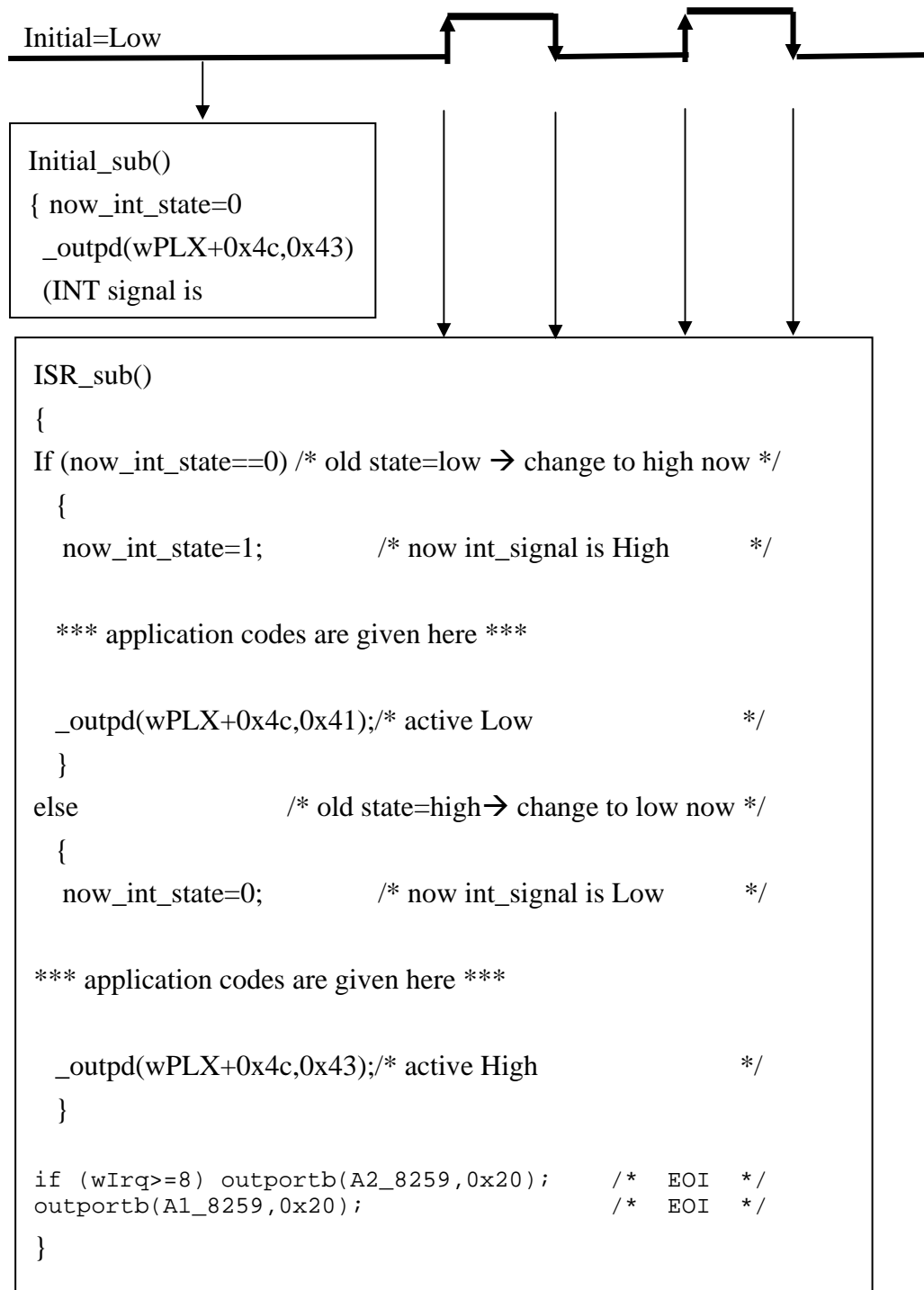
The interrupt of PCI-TMC12 is **level-trigger**. The interrupt signal can be **active-low or active-high** programmable. The procedures of programming are given as following:

1. make sure **the initial level is High or Low**
2. if the initial state is High → set the interrupt signal is active_low initially
3. if the initial state is Low → set the interrupt signal is active_high initially
4. If the interrupt signal is active → program will transfer into the interrupt service routine → **toggle the active_state before return from the ISR.**

Example 1: assume initial level=High



Example 2: assume initial level=Low



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

3.4 New features of PCI-TMC12A

3.4.1 Default Shipping of PCI-TMC12A

The default shipping of J28 is selected in TMC12(Sec. 3.4.4), it is equivalent to PCI-TMC12. So the interrupt system of PCI-TMC12A in the default shipping is compatible to PCI-TMC12. Refer to Sec. 3.4.4 for interrupt block diagram of PCI-TMC12 & PCI-TMC12A.

All Xor? of PCI-TMC12A are clear to their Low states in the first power up stage, so all clock sources of PCI-TMC12A are compatible to those of PCI-TMC12. Refer to Sec. 3.4.2 for block diagram.

In general, you can buy one PCI-TMC12A & use it as PCI-TMC12. **All old application program designed for PCI-TMC12 can be executed in PCI-TMC12A without any modification.**

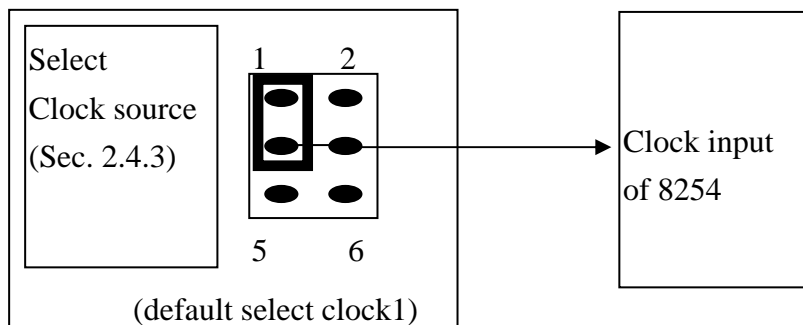
Key point → default shipping of PCI-TMC12A=PCI-TMC12

The new features of PCI-TMC12A are given as follows:

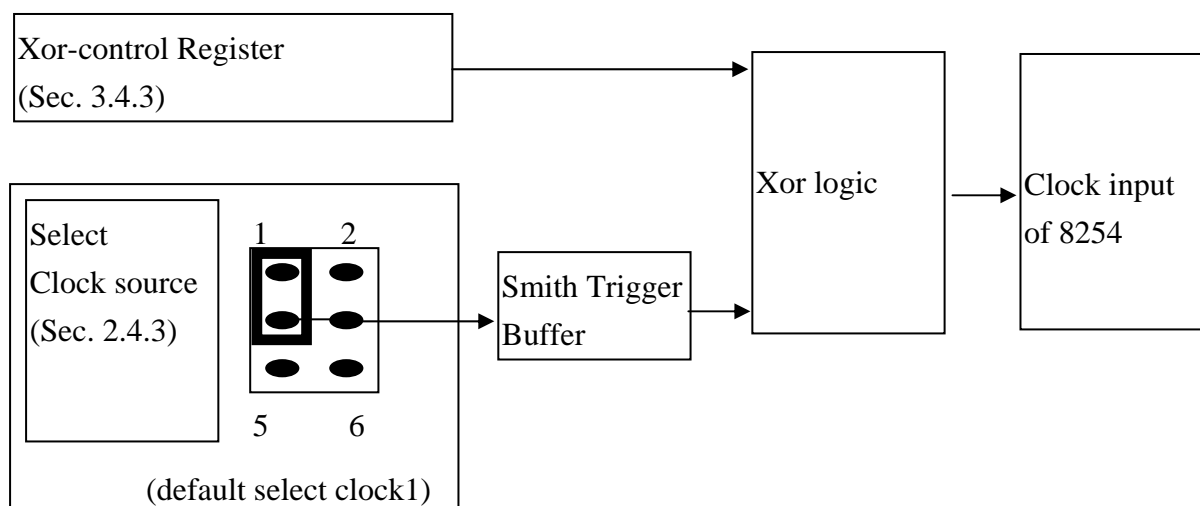
- The new interrupt mechanism (Sec. 3.4.4)
- The Xor? bits for 2 clocks generation (Sec. 3.4.2)
- There are 3 LEDs for status indicators (Sec. 3.4.3 & Sec. 2.1)
- It equips one smith trigger buffer for the selected clock source (Sec. 3.4.2)
- **One new D/O port, wBase+0x18**, for Xor-bits, XorInt & LED on/off control. Refer to Sec. 3.4.3 for more information.
- **One new D/I port, wBase+0x18**, for interrupt enable. The initial routine & ISR must inport from wBase+0x18 to enable next interrupt operation. Refer to Sec. 3.4.4 for more information.
- Refer to new demo programs given in Sec. 3.4.5 for how to use these new features
- Refer to Sec. 2.1 for PCB layout of PCI-TMC12A

3.4.2 Clock input of 8254

The clock input of 8254 chips in PCI-TMC12 is given as follows:



The clock input of 8254 chips in PCI-TMC12A is given as follows:



The new features of PCI-TMC12A are given as follows:

- A smith trigger buffer is added to remove noises in the selected clock source
- A Xor-control register is added to invert/non-inverted the selected clock source. This mechanism can be used to generate 2 extra starting clocks to 8254.

Note: The Xor-control register is clear to 0 when the PCI-TMC12A is first power-up. So the initial state of PCI-TMC12A is exactly compatible to PCI-TMC12.

Refer to Sec. 5.15 Ndemo2: Generate 2 Clocks, the twelve Xor-bits are used to generate the 2 starting clocks. So the initial value of 8254 can be verified after these 2 starting clocks are generated. Then they are used to generate one single clock for testing. In general, these Xor-bits are designed for generation of 2 starting clocks only.

3.4.3 Xor-control Register of PCI-TMC12A

(WRITE) wBase+0x18: set 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 1. Refer to Sec. 3.1 for more information about wBase.

Note 2. All bits of this register will be clear to zero in the power-up stage.

Xor1 --> invert/non-invert the selected clock source of CLK1

Xor2 --> invert/non-invert the selected clock source of CLK2

.....

Xor11 --> invert/non-invert the selected clock source of CLK11

Xor1 2--> invert/non-invert the selected clock source of CLK12

Xor?=0 --> non-invert, it is the power-up value

Xor?=1 --> invert

XorInt-->inverted/non-inverted 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

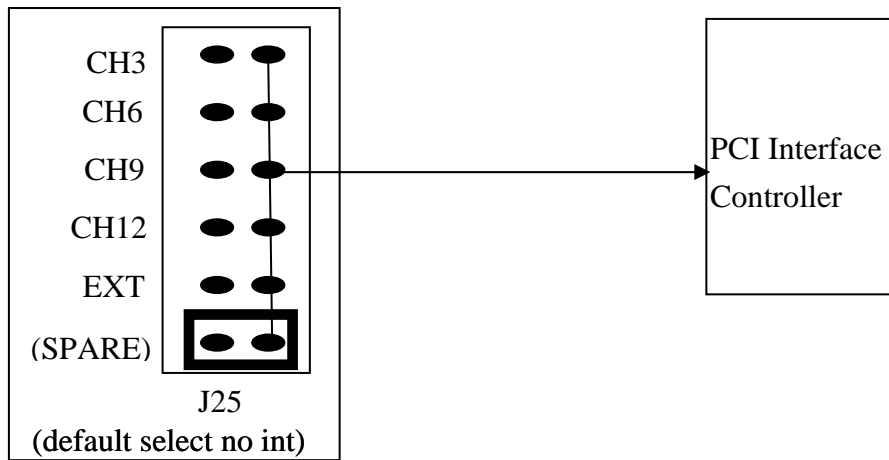
Led3 --> Led3=0 --> Turn LED3 ON, Led3=1 --> turn LED3 Off

- The Xor? is designed to generate the starting 2 clocks for 8254
- The XorInt is used to invert/non-invert the interrupt source to Low state, that is to say, **if the initial value of interrupt source is High, set this bit to High to invert it to Low state.** Refer to Sec. 5. 18 Ndemo5:Active Low Int for demo program.
- When the TMC12A is first power-up, the initial values are all zero. So Led1/2/3 are all turn ON. The Led1/2/3 are designed for status indicators. User can use them based on their special requirements.

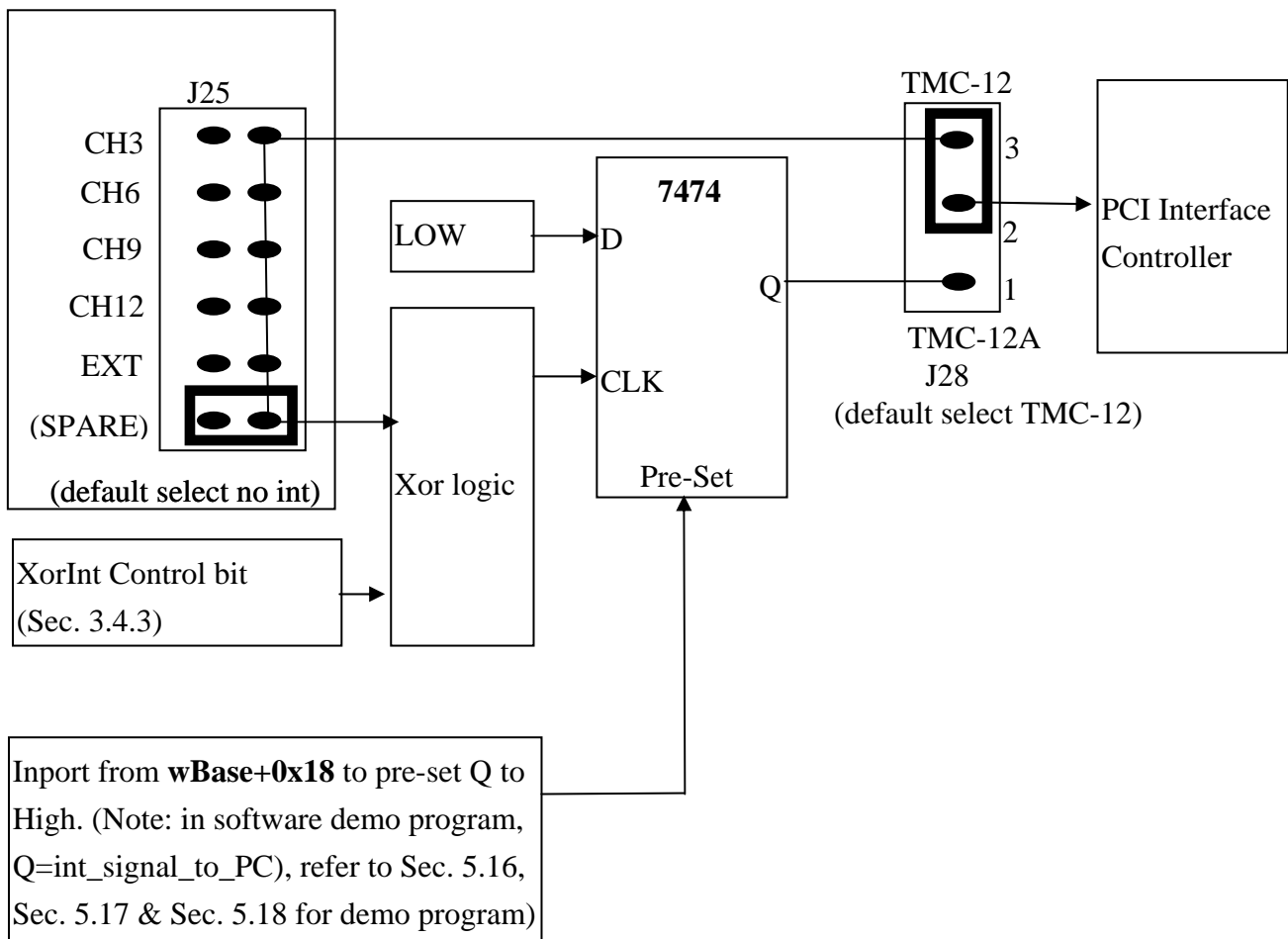
Refer to Sec. 5.15 Ndemo2: Generate 2 Clocks, the twelve Xor-bits are used to generate the 2 starting clocks. So the initial value of 8254 can be verified after these 2 starting clocks are generated. Then they are used to generate single clock for testing. In general, these Xor-bits are designed for generation of 2 starting clocks only.

3.4.4 Block Diagram of Interrupt System

The block diagram of interrupt system in PCI-TMC12 is given as follows:



The block diagram of interrupt system in PCI-TMC12A is given as follows:



The interrupt mechanism of PCI-TMC12 can be active Low or active High. And the interrupt system of PCI bus is level trigger. So the Windows driver of PCI-TMC12 must create a thread to handle all interrupt active conditions. There are so many possible conditions, so the interrupt performance will be reduced very much.

The new interrupt mechanism of PCI-TMC12A is designed to improve the performance of Windows driver as follows:

- initial subroutine & ISR will inport from **wBase+0x18** to pre-set int_signal_to_PC (Q in Sec. 3.4.4) to High state to enable the next interrupt operation
- if the initial value of interrupt source is Low, set XorInt to 0 → rising-edge interrupt
- if the initial value of interrupt source is High, set XorInt to 1 → falling-edge interrupt
- the software driver is designed for rising-edge or falling-edge interrupt

When the interrupt ISR is executed, the int_signal_to_PC (Q in Sec. 3.4.4) is in Low state, so the interrupt ISR must inport from **wBase+0x18** to pre-set int_signal_to_PC to High state to enable next interrupt operation. Refer to Sec. 5.16, Sec. 5.17 & Sec. 5.18 for demo program

3.4.5 New Demo Program

- New demo program 1 → How to Use Status Indicators LEDs
(Refer to Sec. 5.14 Ndemo1: Using LEDs)
- New demo program 2 → How to Generate the Starting 2 Clocks for 8254
(Refer to Sec. 5.15 Ndemo2: Generate 2 Clocks)
- New demo program 3 → Modify demo7 (designed for PCI-TMC12) to fit the new interrupt mechanism of PCI-TMC12A)
(Refer to Sec. 5.16 Ndemo3: New Demo7)
- New demo program 4 → interrupt source = initial low, active High
(Refer to Sec. 5.17 Ndemo4: Active Low Int)
- New demo program 5 → interrupt source = initial High, active low
(Refer to Sec. 5.18 Ndemo5: Active High Int)

4. 8254 Programming

4.1 Control Word Format

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

SC1	SC0	Description
0	0	Select counter_0
0	1	Select counter_1
1	0	Select counter_2
1	1	Read back command

RW1	RW0	Description
0	0	Counter latch command
0	1	Read/write LSB ONLY
1	0	Read/write MSB ONLY
1	1	Read/write LSB first, then read/write MSB

M2	M1	M0	Working mode
0	0	0	Mode 0
0	0	1	Mode 1
Don't care	1	0	Mode 2
Don't care	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD	Description
0	Binary counter, 16-bits
1	Binary coded decimal (BCD) counter (4 decades)

4.2 Counter latch command

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

SC1	SC0	Description
0	0	Latch counter_0
0	1	Latch counter_1
1	0	Latch counter_2
1	1	Read back command

4.3 Read back command

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

- D5=0 → latch counter value of selected counters
- D4=0 → latch status of selected counters
- D3=1 → select counter 2
- D2=1 → select counter 1
- D1=1 → select counter 0

4.4 Status byte format

D7	D6	D5	D4	D3	D2	D1	D0
Cout	Null count	RW1	RW2	M2	M1	M0	BCD

- D7=0 → Cout=Low, D7=1 → Cout=High
- D6=0 → count available for reading, D6=1 → null count
- D5 to D0 → setting value read back

5. Demo Program

The application program of 8254 is very complicated. There are about 10 demo program given in the DOS floppy disk. The program source of library & demo program are all given in the disk. These demo program will help the user to solve their real world problem more easy.

- \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

5.1 Demo1: Use D/O

```
/* 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_tmcl2_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 )
    {
        putchar(0x07); putchar(0x07); putchar(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_tmcl2_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_tmcl2_do(WORD wDo)
{
    outport(wBaseAddr+0x14,wDo);
    return(NoError);
}

```

5.2 Demo2: Use D/I

- If there is only one PCI-TMC12, this program will test this only card.
- If there are over one PCI-TMC12 cards installed in the PC system, this program will **test the second card**.
- How can we know which card is the second card ? Please refer to Sec. 3.2 for more information.

```
/* 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_tmcl2_do(WORD wDo);
void pci_tmcl2_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_tmcl2_do(j); pci_tmcl2_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_tmcl2_di(WORD *wDi)
{
    WORD wRetVal;
    (*wDi)=(inport(wBase+0x14))&0xffff;
}

```

5.3 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_tmcl2_select8254(char cChip);
WORD pci_tmcl2_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_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 )
    {
        putchar(0x07); putchar(0x07); putchar(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_tmcl2_select8254(0); /* select 8254-chip-1 */
    pci_tmcl2_c0(0x36, 2, 0); /* CH-1, mode-3, low=2, high=0, cout=4M */
    pci_tmcl2_c1(0x76, 4, 0); /* CH-2, mode-3, low=4, high=0, cout=2M */
    pci_tmcl2_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_tmcl2_select8254(1); /* select 8254-chip-2 */
    pci_tmcl2_c0(0x36, 16, 0); /* CH-4, mode-3, low=16, high=0, cout=500K */
    pci_tmcl2_c1(0x76, 32, 0); /* CH-5, mode-3, low=32, high=0, cout=250K */
    pci_tmcl2_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_tmcl2_select8254(2); /* select 8254-chip-3 */
    pci_tmcl2_c0(0x36, 128, 0); /* CH-7, mode-3, low=128, high=0, cout=64K */
    pci_tmcl2_c1(0x76, 0, 1); /* CH-8, mode-3, low=0, high=1, cout=32K */
    pci_tmcl2_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_tmcl2_select8254(3); /* select 8254-chip-4 */
pci_tmcl2_c0(0x36,0,4); /* CH-10,mode-3,low=0,high=4,cout=8K */
pci_tmcl2_c1(0x76,0,8); /* CH-11,mode-3,low=0,high=8,cout=4K */
pci_tmcl2_c2(0xb6,0,16); /* CH-12,mode-3,low=0,high=16,cout=2K */

PTMCl2_DriverClose();
}

/* ----- */

WORD pci_tmcl2_select8254(char cChip)
{
outportb(wBaseAddr+0x10,cChip);
return(NoError);
}

WORD pci_tmcl2_c0(char cConfig, char cLow, char cHigh)
{
outportb(wBaseAddr+0x0C,cConfig);
outportb(wBaseAddr      ,cLow);
outportb(wBaseAddr      ,cHigh);
return(NoError);
}

WORD pci_tmcl2_c1(char cConfig, char cLow, char cHigh)
{
outportb(wBaseAddr+0x0C,cConfig);
outportb(wBaseAddr+4   ,cLow);
outportb(wBaseAddr+4   ,cHigh);
return(NoError);
}

WORD pci_tmcl2_c2(char cConfig, char cLow, char cHigh)
{
outportb(wBaseAddr+0x0C,cConfig);
outportb(wBaseAddr+8   ,cLow);
outportb(wBaseAddr+8   ,cHigh);
return(NoError);
}

```

5.4 Demo4: Delay One Ms

- This demo use CNT1 to implement a **machine independent timer**. So you can run this demo **in any speed PC** & find the * shown in screen every seconds. The machine independent timer is useful in industry application.

```
/* 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_tmcl2_select8254(char cChip);
WORD pci_tmcl2_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_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 --> 0xffff */
delay_one_ms()
{
    int low, high;
    pci_tmcl2_select8254(0); /* select 8254-chip-0 */
    pci_tmcl2_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 */
    }
}
```

5.5 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_tmcl2_select8254(char cChip);
WORD pci_tmcl2_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_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_tmcl2_select8254(0); /* select 8254-chip-0 */
pci_tmcl2_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_Count + 2

Note2: If the count > 65536 → this 16-bit counter will be overflow.

So → refer to DEMO6 for infinite-bit counter.

5.6 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_tmcl2_select8254(char cChip);
WORD pci_tmcl2_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_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_tmcl2_select8254(0); /* select 8254-chip-0 */
    pci_tmcl2_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_Count + 2

Note 4: This software counter can be nearly infinite-bits.

5.7 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_tmcl2_select8254(char cChip);
WORD pci_tmcl2_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_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_tmcl2_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_tmcl2_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 Sec. 3.3.5 for more information.

```

5.8 Demo8: Pulse Width Measure

```
/* 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_tmcl2_di(WORD *wDi);
WORD pci_tmcl2_select8254(char cChip);
WORD pci_tmcl2_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_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) Threre 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_tmcl2_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 */

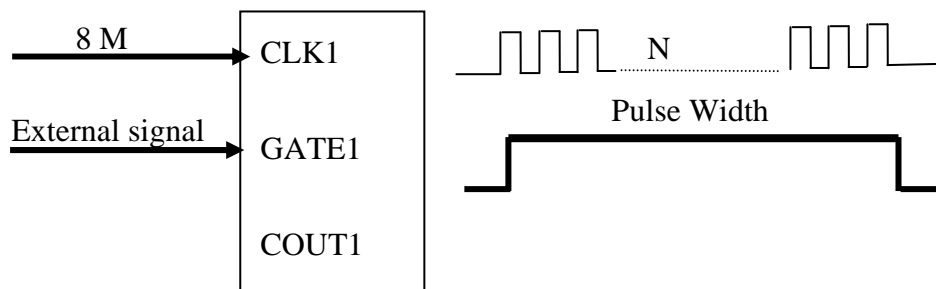
pci_tmcl2_select8254(0); /* select 8254-chip-0 */
pci_tmcl2_c0(0x30,0xff,0xff); /* CH-1,mode-0 down count ffff */

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();
}

```



- $N = \text{number of down count in CNT1}(8 \text{ M clock})$
- $\text{Pulse width} = 8\text{M_width} * N$

5.9 Demo9: Frequency Measure

```
/* 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_tmcl2_di(WORD *wDi);
WORD pci_tmcl2_select8254(char cChip);
WORD pci_tmcl2_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_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 )
    {
        putchar(0x07); putchar(0x07); putchar(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_tmcl2_select8254(0); /* select 8254-chip-0 */
    pci_tmcl2_c0(0x30,0xff,0xff); /* CH-1,mode-0 down count ffff */
    pci_tmcl2_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;}
    }

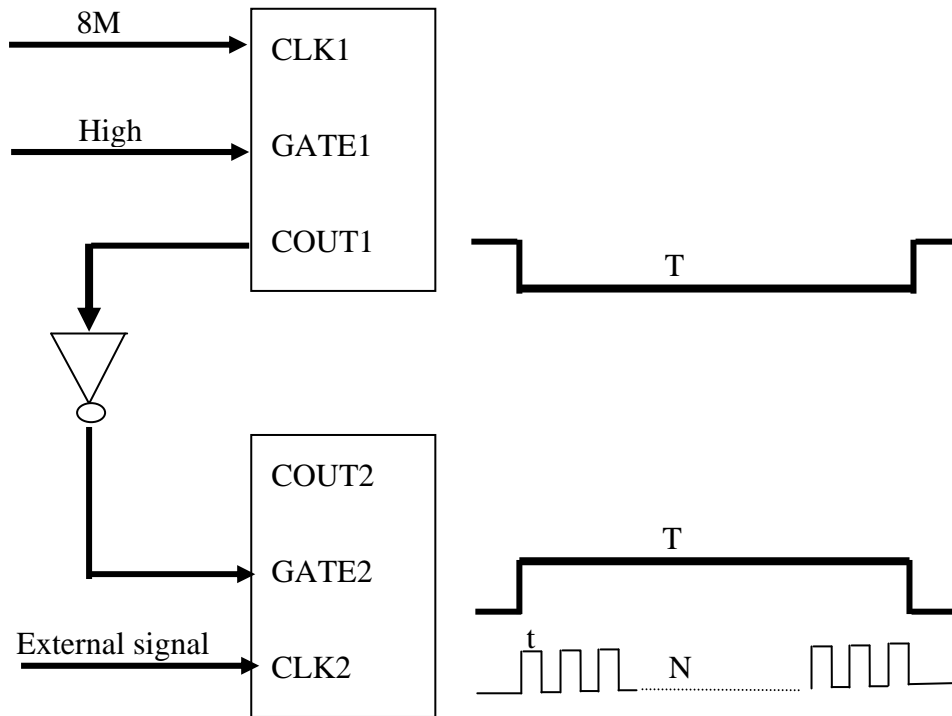
    outportb(wBaseAddr+0x0C,0x40); /* latch counter_1 */
    low=inportb(wBaseAddr+0x04);
    high=inportb(wBaseAddr+0x04);
    count=(0xff-high)*256+(0xff-low)+2;
}
```

```

/* 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=number of down count in CNT2
- $t=T/\text{Down_count2}$
- $f=1/t$
- The CNT1 can be changed to CNT3/4/5/6. The COUT of CNT 8/9/10/11/12/13 are directly connected to next counter without inverter. So they can not be used to replace CNT1.
- The 12 CNTs of TMC-12 are divided into two groups: inverter group & non-inverted group. The inverted group includes CNT 1/2/3/4/5/6. The non-inverted group included CNT 7/8/9/10/11/12. The user has to select his proper group for different application.

5.10 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_tmcl2_do(WORD wDo);
void pci_tmcl2_di(WORD *wDi);
WORD pci_tmcl2_select8254(char cChip);
WORD pci_tmcl2_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_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 )
    {
        putchar(0x07); putchar(0x07); putchar(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_tmcl2_do(j); pci_tmcl2_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");
}

```

5.11 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 demoll.exe */
/* ----- */

#include "PCITMC12.H"

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

WORD pci_tmcl2_select8254(char cChip);
WORD pci_tmcl2_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_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_tmcl2_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_tmcl2_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 Sec. 3.3.5 for more information.

5.12 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_tmcl2_select8254(char cChip);
WORD pci_tmcl2_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_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 )
    {
        putchar(0x07); putchar(0x07); putchar(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_tmcl2_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_tmcl2_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_tmcl2_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 */
    }
}

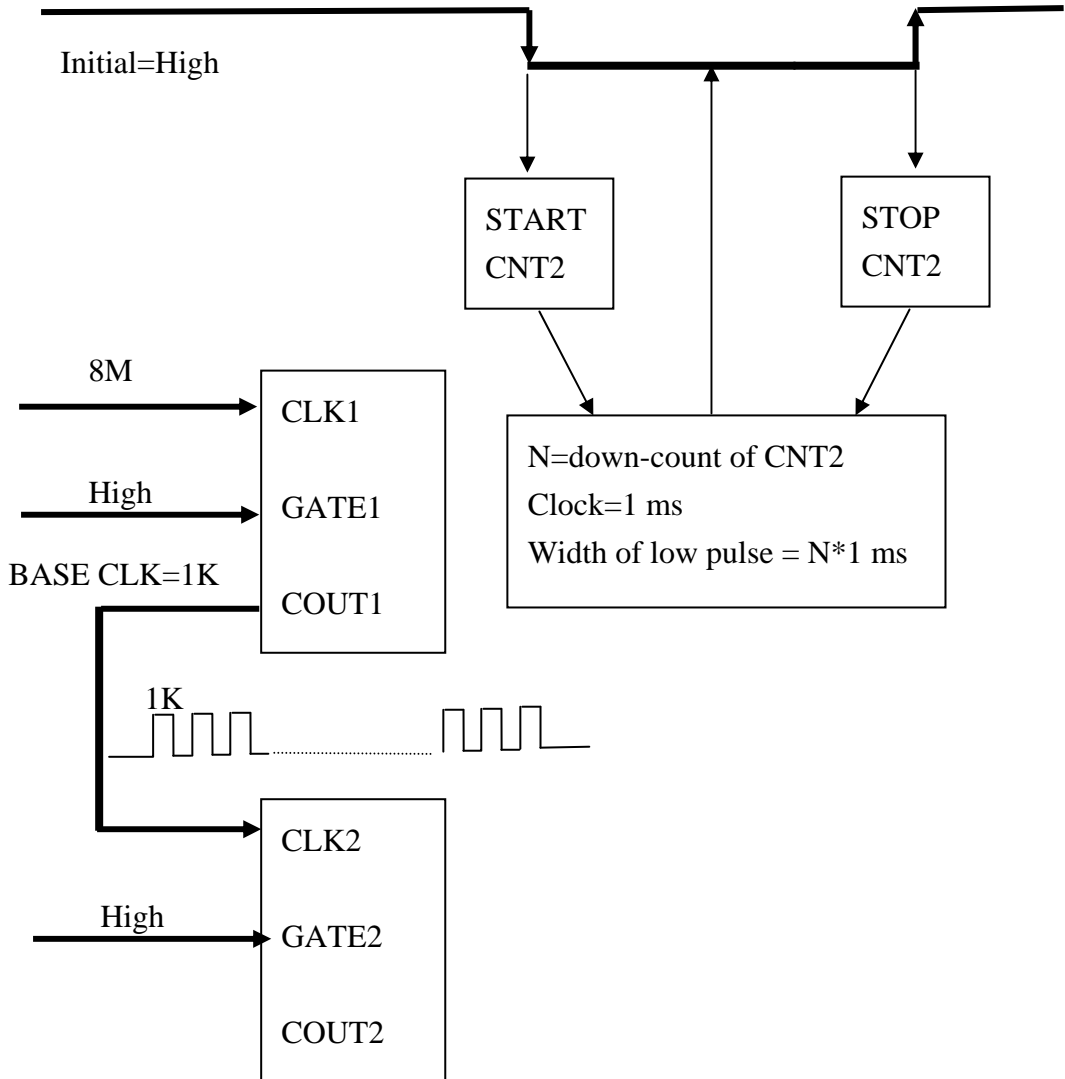
```

```

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 Sec. 3.3.5 for more information.

5.13 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();
}

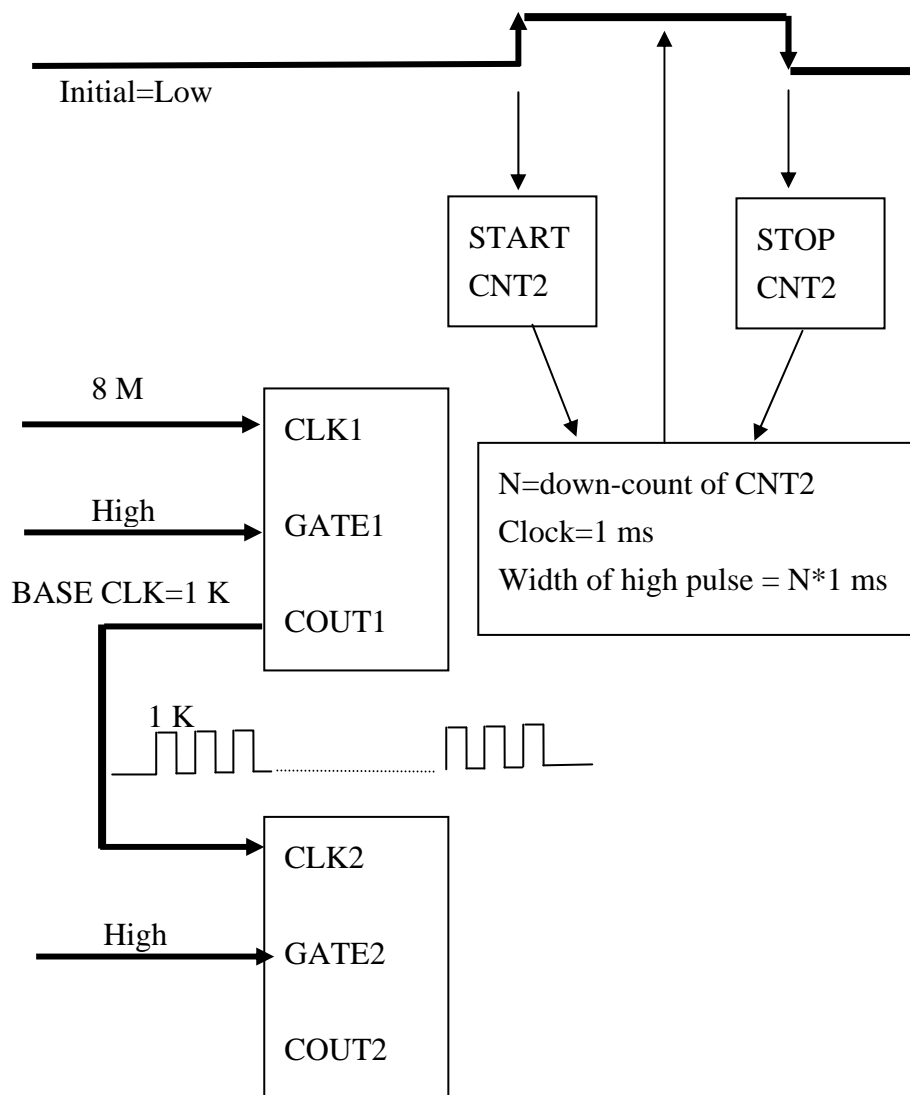
void interrupt irq_service()
{
    if (now_int_state==0)
    {
        COUNT3++; /* find a high_pulse */
        if (COUNT3==2) /* start to down-count */
            pci_tmc12_c1(0x70,0xff,0xff); /* CH-2,mode-0 down count ffff */
            _outpd(wPLX+0x4c,0x41); /* channel_1, interrupt active_Low */
            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 Sec. 3.3.5 for more information.

5.14 Ndemo1: Using 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_tmcl2_do(WORD wDo);
WORD pci_tmcl2_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 )
    {
        putchar(0x07); putchar(0x07); putchar(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_tmcl2_do2(0xe000); printf("\nAll LED off, press any key to
continue"); getch();
    pci_tmcl2_do2(0xc000); printf("\nLED1 on, press any key to
continue"); getch();
    pci_tmcl2_do2(0xa000); printf("\nLED2 on, press any key to
continue"); getch();
    pci_tmcl2_do2(0x6000); printf("\nLED3 on, press any key to
continue"); getch();

    PTMC12_DriverClose();
}

/* ----- */

WORD pci_tmcl2_do(WORD wDo)
{
    output(wBaseAddr+0x14,wDo);
    return(NoError);
}

/* ----- */

WORD pci_tmcl2_do2(WORD wXor)
{
    output(wBaseAddr+0x18,wXor);
    return(NoError);
}
```

5.15 Ndemo2: Generate 2 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 counter2 */
/* ----- */

#include "PCITMC12.H"

WORD wBaseAddr,wIrq,wPLX;
WORD pci_tmcl2_do(WORD wDo);
WORD pci_tmcl2_do2(WORD wXor);
WORD pci_tmcl2_select8254(char cChip);
WORD pci_tmcl2_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_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 )
    {
        putchar(0x07); putchar(0x07); putchar(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_tmcl2_select8254(0);
    pci_tmcl2_c0(0x30,0xfe,0xff);
    pci_tmcl2_c1(0x70,0xfd,0xff);
    pci_tmcl2_c2(0xb0,0xfc,0xff);

    pci_tmcl2_select8254(1);
    pci_tmcl2_c0(0x30,0xfb,0xff);
    pci_tmcl2_c1(0x70,0xfa,0xff);
    pci_tmcl2_c2(0xb0,0xf9,0xff);
}
```

```

pci_tmcl2_select8254(2);
pci_tmcl2_c0(0x30,0xf8,0xff);
pci_tmcl2_c1(0x70,0xf7,0xff);
pci_tmcl2_c2(0xb0,0xf6,0xff);

pci_tmcl2_select8254(3);
pci_tmcl2_c0(0x30,0xf5,0xff);
pci_tmcl2_c1(0x70,0xf4,0xff);
pci_tmcl2_c2(0xb0,0xf3,0xff);

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

delay(1);
pci_tmcl2_do2(0);
pci_tmcl2_do2(0x0fff);
pci_tmcl2_do2(0);
pci_tmcl2_do2(0x0fff);
pci_tmcl2_do2(0);

for (;;)
{
pci_tmcl2_select8254(0);
read_c0(1); /* Counter 1 */
read_c1(2); /* Counter 2 */
read_c2(3); /* Counter 3 */

pci_tmcl2_select8254(1);
read_c0(4); /* Counter 4 */
read_c1(5); /* Counter 5 */
read_c2(6); /* Counter 6 */

pci_tmcl2_select8254(2);
read_c0(7); /* Counter 7 */
read_c1(8); /* Counter 8 */
read_c2(9); /* Counter 9 */

pci_tmcl2_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_tmcl2_do2(0x0fff);
pci_tmcl2_do2(0);

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

PTMC12_DriverClose();
}
/* ----- */
WORD pci_tmcl2_do(WORD wDo)
{
outport(wBaseAddr+0x14,wDo);
return(NoError);
}
/* ----- */
WORD pci_tmcl2_do2(WORD wXor)
{
outport(wBaseAddr+0x18,wXor);
return(NoError);
}

```

```
}
```

5.16 Ndemo3: New 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_tmcl2_select8254(char cChip);
WORD pci_tmcl2_c0(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_c1(char cConfig, char cLow, char cHigh);
WORD pci_tmcl2_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_tmcl2_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_tmcl2_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);}

```

5.17 Ndemo4: Active High Int

```
/* ndemo4 : interrupt demo, int source=initial low, active High */
/* step 1 : connect D01 (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_tmcl2_do(WORD wDo);
WORD pci_tmcl2_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 )
    {
        putchar(0x07); putchar(0x07); putchar(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_tmcl2_do(0); /* D01=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_tmcl2_do(1); /* generate a High pulse to */
        pci_tmcl2_do(0); /* D01=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_tmcl2_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);
}

```

5.18 Ndemo5: Active Low Int

```
/* ndemo5 : interrupt demo, int source=initial High, active Low */
/* step 1 : connect D01 (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_tmcl2_do(WORD wDo);
WORD pci_tmcl2_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 )
    {
        putchar(0x07); putchar(0x07); putchar(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_tmcl2_do(1); /* D01=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_tmcl2_do(0); /* generate a Low pulse to */
        pci_tmcl2_do(1); /* D01=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_tmcl2_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);
}

```