

---

PISO-CAN200/400  
PISO-CAN100U/200U/400U/800U  
PEX-CAN200i  
PCM-CAN100/200/200P

---

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 2003 by ICP DAS. All rights are reserved.

**Trademark**

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

---

## Tables of Content

<b>1</b>	<b>General Information.....</b>	<b>4</b>
1.1	<i>Introduction.....</i>	<i>4</i>
1.2	<i>Features.....</i>	<i>5</i>
1.3	<i>Hardware Specifications .....</i>	<i>6</i>
1.3.1	<i>PCM-CAN100/200/200P.....</i>	<i>6</i>
1.3.2	<i>PEX-CAN200i.....</i>	<i>7</i>
1.3.3	<i>PISO-CAN200/200U .....</i>	<i>8</i>
1.3.4	<i>PISO-CAN400/400U .....</i>	<i>9</i>
1.3.5	<i>PISO-CAN100U .....</i>	<i>10</i>
1.3.6	<i>PISO-CAN800U .....</i>	<i>11</i>
1.4	<i>Product Check List.....</i>	<i>12</i>
<b>2</b>	<b>Hardware Configuration.....</b>	<b>13</b>
2.1	<i>Board Layout.....</i>	<i>13</i>
2.2	<i>Jumper Selection.....</i>	<i>18</i>
2.3	<i>Connector Pin Assignment.....</i>	<i>22</i>
2.3.1	<i>5-pin screw terminal connector .....</i>	<i>22</i>
2.3.2	<i>9-pin male D-sub connectors .....</i>	<i>23</i>
2.3.3	<i>37-pin female D-sub connectors .....</i>	<i>24</i>
2.4	<i>Installation.....</i>	<i>25</i>
<b>3</b>	<b>Software Installation.....</b>	<b>26</b>
<b>4</b>	<b>Installation DLL Driver.....</b>	<b>29</b>
4.1	<i>DLL Function Definition and Description .....</i>	<i>30</i>
4.1.1	<i>CAN_GetDllVersion .....</i>	<i>33</i>
4.1.2	<i>CAN_TotalBoard.....</i>	<i>33</i>
4.1.3	<i>CAN_GetBoardInf.....</i>	<i>34</i>
4.1.4	<i>CAN_GetCardPortNum .....</i>	<i>35</i>
4.1.5	<i>CAN_ActiveBoard .....</i>	<i>36</i>
4.1.6	<i>CAN_CloseBoard .....</i>	<i>37</i>
4.1.7	<i>CAN_BoardIsActive.....</i>	<i>38</i>
4.1.8	<i>CAN_Reset.....</i>	<i>39</i>
4.1.9	<i>CAN_Init.....</i>	<i>40</i>
4.1.10	<i>CAN_Config.....</i>	<i>41</i>
4.1.11	<i>CAN_ConfigWithoutStructure .....</i>	<i>43</i>
4.1.12	<i>CAN_EnableRxIrq.....</i>	<i>45</i>
4.1.13	<i>CAN_DisableRxIrq.....</i>	<i>46</i>
4.1.14	<i>CAN_RxIrqStatus.....</i>	<i>47</i>
4.1.15	<i>CAN_InstallIrq.....</i>	<i>48</i>

---

4.1.16	<i>CAN_RemoveIrq</i> .....	49
4.1.17	<i>CAN_IrqStatus</i> .....	50
4.1.18	<i>CAN_Status</i> .....	51
4.1.19	<i>CAN_SendMsg</i> .....	52
4.1.20	<i>CAN_SendWithoutStruct</i> .....	54
4.1.21	<i>CAN_RxMsgCount</i> .....	55
4.1.22	<i>CAN_ReceiveMsg</i> .....	56
4.1.23	<i>CAN_ReceiveWithoutStruct</i> .....	58
4.1.24	<i>CAN_ClearSoftBuffer</i> .....	60
4.1.25	<i>CAN_ClearDataOverrun</i> .....	61
4.1.26	<i>CAN_OutputByte</i> .....	62
4.1.27	<i>CAN_InputByte</i> .....	63
4.1.28	<i>CAN_GetSystemFreq</i> .....	64
4.1.29	<i>CAN_InstallUserIsr (only for Windows 2000/XP)</i> .....	65
4.1.30	<i>CAN_RemoveUserIsr (only for Windows 2000/XP)</i> .....	66
4.2	<i>Flow Diagram for Application</i> .....	67
5	Demo Programs for Windows.....	70
6	CANUtility Program for Windows .....	73
7	Appendix .....	79
7.1	<i>Acceptance Filtering</i> .....	79
8	Dimensions .....	82
8.1	<i>PISO-CAN200/400</i> .....	82
8.2	<i>PISO-CAN100U/200U/400U/800U</i> .....	83
8.3	<i>PEX-CAN200i</i> .....	86
8.4	<i>PCM-CAN100/200/200P</i> .....	87

---

# 1 General Information

## 1.1 Introduction

The CAN (Controller Area Network) is a serial communication protocol, which efficiently supports distributed real-time control with a very high level of security. It is especially suited for networking "intelligent" devices as well as sensors and actuators within a system or sub-system. In CAN networks, there is no addressing of subscribers or stations in the conventional sense, but instead prioritized messages are transmitted. As a stand-alone CAN controller, PISO-CAN, PEX-CAN, and PCM-CAN represents an economic solution within which an active CAN board can have two or four independent CAN bus communication ports with either a 5-pin screw terminal connector or a 9-pin D-sub connector. It can be a master/slave interface, and be applied in various CAN applications. In addition, these CAN cards use the new NXP SJA1000T and transceiver 82C250/251, which provide the bus arbitration and error detection. The differences between these CAN cards are the interface of PC. Some are for PCI interface, some are for PCI Express interface, and some are for PCI-104 interface. To get the detail information for the features and the specification of these CAN cards, please refer to the section 1.2 and 1.3.

---

## 1.2 Features

- PCI BUS interface
- 2500Vrms photo-isolation protection
- 1/2/4/8 independent CAN communication ports
- Compatible with CAN specification 2.0 parts A and B
- On-board optical isolation protection
- Programmable transfer-rate up to 1 Mbps
- Jumper select 120Ω terminator resistor for each port
- Direct memory mapping to the CAN controllers
- PISO-CAN200/400
  - 33MHz 32bit 5V PCI bus (V2.1) plug and play technology
  - PCI card, supports 5V PCI bus
  - 3KV galvanic isolation
  - 2/4 independent CAN channels for PISO-CAN200/400
- PISO-CAN100U/200U/400U/800U
  - PCI v2.2 compliant 32-bit 33MHz
  - Universal PCI card, supports both 5V and 3.3V PCI bus
  - 3KV galvanic isolation
  - 1/2/4/8 independent CAN channels for PISO-CAN100U/200U/400U/800U
- PEX-CAN200i
  - 32-bit, 33MHz, X1 PCI Express Bus
  - According to PCI Express specification R1.0
  - 3KV galvanic isolation
  - 2 independent CAN channels
- PCM-CAN100/200/200P
  - PCI104 compliant
  - 9-pin D-sub connector
  - 1KV galvanic isolation
  - 1/2 independent CAN channels
- Driver supported for Windows 2000/XP/7, Linux 2.6.37

## 1.3 Hardware Specifications

### 1.3.1 PCM-CAN100/200/200P

Model Name	PCM-CAN100-D	PCM-CAN200-D	PCM-CAN200P-D
<b>Bus Interface</b>			
Type	PCI-104		PC-104+
<b>CAN Interface</b>			
Controller	NXP SJA1000T with 16 MHz clock		
Transceiver	NXP 82C250		
Channel number	1	2	
Connector	9-pin female and male D-Sub (CAN_L, CAN_SHLD, CAN_H, N/A for others)	9-pin male D-Sub (CAN_L, CAN_SHLD, CAN_H, N/A for others)	
Baud Rate (bps)	10 k, 20 k, 50 k, 125 k, 250 k, 500 k, 800 k, 1 M (allow user-defined baud rate)		
Terminator Resistor	Jumper for 120 $\Omega$ terminator resistor		
<b>Power</b>			
Power Consumption	250 mA @ 5 V		
<b>Mechanism</b>			
<a href="#">Dimensions</a>	91mm x 22mm x 96mm (W x L x H)		
<b>Environment</b>			
Operating Temp.	0 ~ 60 $^{\circ}\text{C}$		
Storage Temp.	-20 ~ 70 $^{\circ}\text{C}$		
Humidity	5 ~ 85% RH, non-condensing		

### 1.3.2 PEX-CAN200i

Model Name	PEX-CAN200i-D	PEX-CAN200i-T
<b>Bus Interface</b>		
Type	33 MHz, 32 bit, X1 PCI Express bus	
<b>CAN Interface</b>		
Controller	NXP SJA1000T with 16 MHz clock	
Transceiver	NXP 82C250	
Channel number	2	
Connector	9-pin male D-Sub	5-pin screwed terminal block
Baud Rate (bps)	10 k, 20 k, 50 k, 125 k, 250 k, 500 k, 800 k, 1 M (allow user-defined baud rate)	
Terminator Resistor	Jumper for 120 $\Omega$ terminator resistor	
<b>Power</b>		
Power Consumption	100 mA @ 12 V, 100 mA @ 3.3 V	
<b>Software</b>		
Driver	Windows 2K/XP/7, Linux 2.6.37, LabView, DASyLab	
Library	VB 6.0, VC++ 6.0, BCB 6.0, Delphi 4.0	
<b>Mechanism</b>		
<a href="#">Dimensions</a>	120mm x 22mm x 85mm (W x L x H)	
<b>Environment</b>		
Operating Temp.	0 ~ 60 $^{\circ}$ C	
Storage Temp.	-20 ~ 70 $^{\circ}$ C	
Humidity	5 ~ 85% RH, non-condensing	

### 1.3.3 PISO-CAN200/200U

Model Name	PISO-CAN200-D	PISO-CAN200-T	PISO-CAN200U-D	PISO-CAN200U-T
<b>Bus Interface</b>				
Type	PCI bus, 5 V, 33 MHz, 32-bit, plug and play		Universal PCI, 3.3 V and 5 V, 33 MHz, 32-bit, plug and play	
<b>CAN Interface</b>				
Controller	NXP SJA1000T with 16 MHz clock			
Transceiver	NXP 82C250			
Channel number	2			
Connector	9-pin male D-Sub	5-pin screwed terminal block	9-pin male D-Sub	5-pin screwed terminal block
Baud Rate (bps)	10 k, 20 k, 50 k, 125 k, 250 k, 500 k, 800 k, 1 M (allow user-defined baud rate)			
Terminator Resistor	Jumper for 120 $\Omega$ terminator resistor			
<b>Power</b>				
Power Consumption	250 mA @ 5 V			
<b>Software</b>				
Driver	Windows 2K/XP/7, Linux 2.6.37, LabView, DASyLab			
Library	VB 6.0, VC++ 6.0, BCB 6.0, Delphi 4.0			
<b>Mechanism</b>				
<a href="#">Dimensions</a>	126mm x 22mm x 85mm (W x L x H)			
<b>Environment</b>				
Operating Temp.	0 ~ 60 $^{\circ}\text{C}$			
Storage Temp.	-20 ~ 70 $^{\circ}\text{C}$			
Humidity	5 ~ 85% RH, non-condensing			



### 1.3.4 PISO-CAN400/400U

Model Name	PISO-CAN400U-D	PISO-CAN400U-T	PISO-CAN400U-D	PISO-CAN400U-T
<b>Bus Interface</b>				
Type	PCI bus, 5 V, 33 MHz, 32-bit, plug and play		Universal PCI, 3.3 V and 5 V, 33 MHz, 32-bit, plug and play	
<b>CAN Interface</b>				
Controller	NXP SJA1000T with 16 MHz clock			
Transceiver	NXP 82C250			
Channel number	4			
Connector	9-pin male D-Sub	5-pin screwed terminal block	9-pin male D-Sub	5-pin screwed terminal block
Baud Rate (bps)	10 k, 20 k, 50 k, 125 k, 250 k, 500 k, 800 k, 1 M (allow user-defined baud rate)			
Terminator Resistor	Jumper for 120 $\Omega$ terminator resistor			
<b>Power</b>				
Power Consumption	300 mA @ 5 V			
<b>Software</b>				
Driver	Windows 2K/XP/7, Linux 2.6.37, LabView, DASyLab			
Library	VB 6.0, VC++ 6.0, BCB 6.0, Delphi 4.0			
<b>Mechanism</b>				
<a href="#">Dimensions</a>	126mm x 22mm x 85mm (W x L x H)			
<b>Environment</b>				
Operating Temp.	0 ~ 60 $^{\circ}\text{C}$			
Storage Temp.	-20 ~ 70 $^{\circ}\text{C}$			
Humidity	5 ~ 85% RH, non-condensing			

### 1.3.5 PISO-CAN100U

Model Name	PISO-CAN100U-D	PISO-CAN100U-T
<b>Bus Interface</b>		
Type	Universal PCI, 3.3 V and 5 V, 33 MHz, 32-bit, plug and play	
<b>CAN Interface</b>		
Controller	NXP SJA1000T with 16 MHz clock	
Transceiver	NXP 82C250	
Channel number	1	
Connector	9-pin male D-Sub	5-pin screwed terminal block
Baud Rate (bps)	10 k, 20 k, 50 k, 125 k, 250 k, 500 k, 800 k, 1 M (allow user-defined baud rate)	
Terminator Resistor	Jumper for 120 $\Omega$ terminator resistor	
<b>Power</b>		
Power Consumption	225 mA @ 5 V	
<b>Software</b>		
Driver	Windows 2K/XP/7, Linux 2.6.37, LabView, DASyLab	
Library	VB 6.0, VC++ 6.0, BCB 6.0, Delphi 4.0, C#.Net, VB.Net	
<b>Mechanism</b>		
<a href="#">Dimensions</a>	126mm x 22mm x 85mm (W x L x H)	
<b>Environment</b>		
Operating Temp.	0 ~ 60 $^{\circ}\text{C}$	
Storage Temp.	-20 ~ 70 $^{\circ}\text{C}$	
Humidity	5 ~ 85% RH, non-condensing	

### 1.3.6 PISO-CAN800U

<b>Bus Interface</b>	
Type	Universal PCI, 3.3 V and 5 V, 33 MHz, 32-bit, plug and play
<b>CAN Interface</b>	
Controller	NXP SJA1000T with 16 MHz clock
Transceiver	NXP TJA1042
Channel number	8
Connector	Female DB-37 x 2
Baud Rate (bps)	10 k, 20 k, 50 k, 125 k, 250 k, 500 k, 800 k, 1 M (allow user-defined baud rate)
Terminator Resistor	Jumper for 120 $\Omega$ terminator resistor
<b>Power</b>	
Power Consumption	800 mA @ 5 V
<b>Software</b>	
Driver	Windows 2K/XP/7, LabView, DASyLab
Library	VB 6.0, VC++ 6.0, BCB 6.0, Delphi 4.0, C#.Net, VB.Net
<b>Mechanism</b>	
<a href="#">Dimensions</a>	193mm x 22mm x 93mm (W x L x H)
<b>Environment</b>	
Operating Temp.	0 ~ 60 $^{\circ}$ C
Storage Temp.	-20 ~ 70 $^{\circ}$ C
Humidity	5 ~ 85% RH, non-condensing

---

## **1.4 Product Check List**

Besides this manual, the package includes the following items:

- Hardware of PISO-CAN or PEX-CAN or PCM-CAN CAN card
- ADP-9 Board (for PISO-CAN400/PISO-CAN400U only)
- Software CD ROM

**It is recommended that users read the release note first. All the important information needed will be provided in the release note as follows:**

- Where you can find the software driver, utility and demo programs.
- How to install software & utility.
- Where is the diagnostic program?
- FAQ's and answers.

### **Attention!**

If any of these items are missing or damaged, please contact your local field agent. Keep aside the shipping materials and carton in case you want to ship or store the product in the future.

---

## 2 Hardware Configuration

This section will describe the hardware settings of the PISO-CAN, PEX-CAN, and PCM-CAN series CAN card. This information includes the wire connection and terminal resistance configuration for the CAN network.

### 2.1 Board Layout

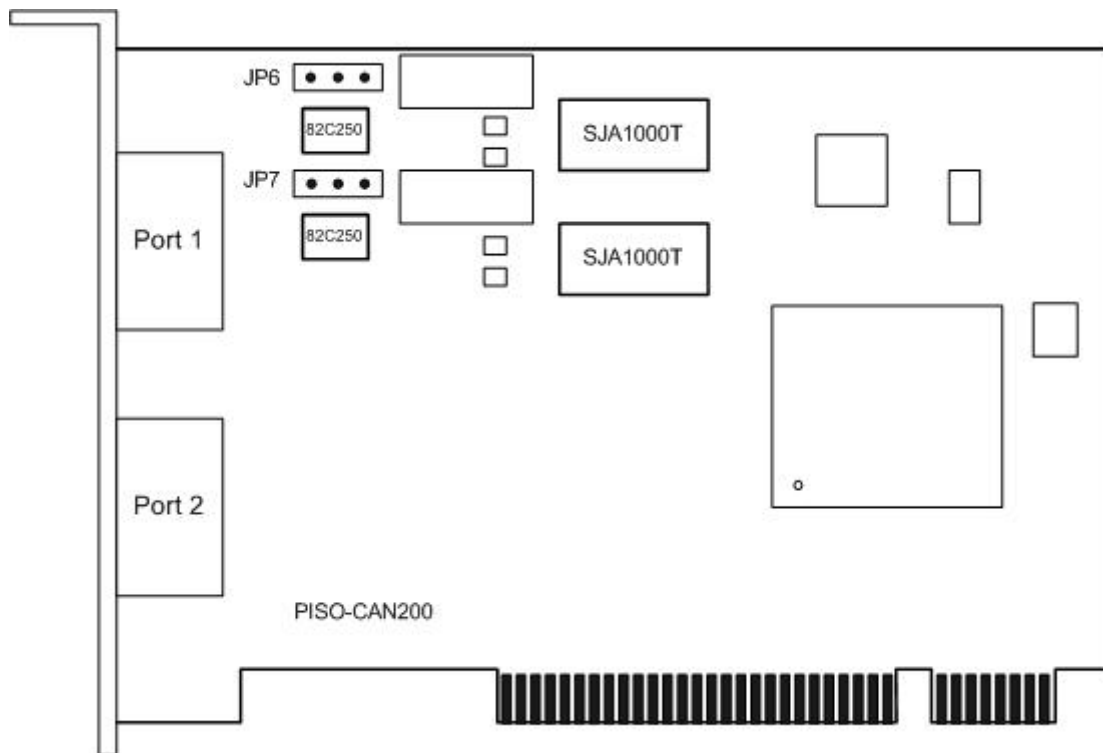


Figure2.1 PISO-CAN200 Board LAYOUT

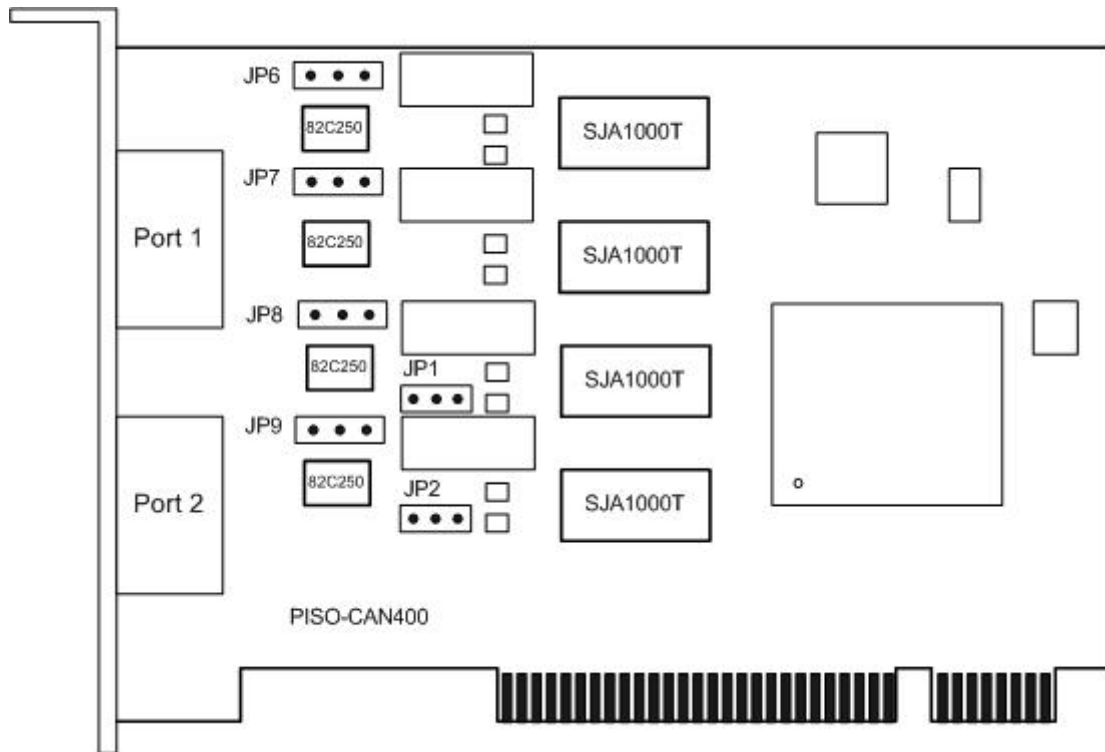


Figure2.2 PISO-CAN400 Board LAYOUT

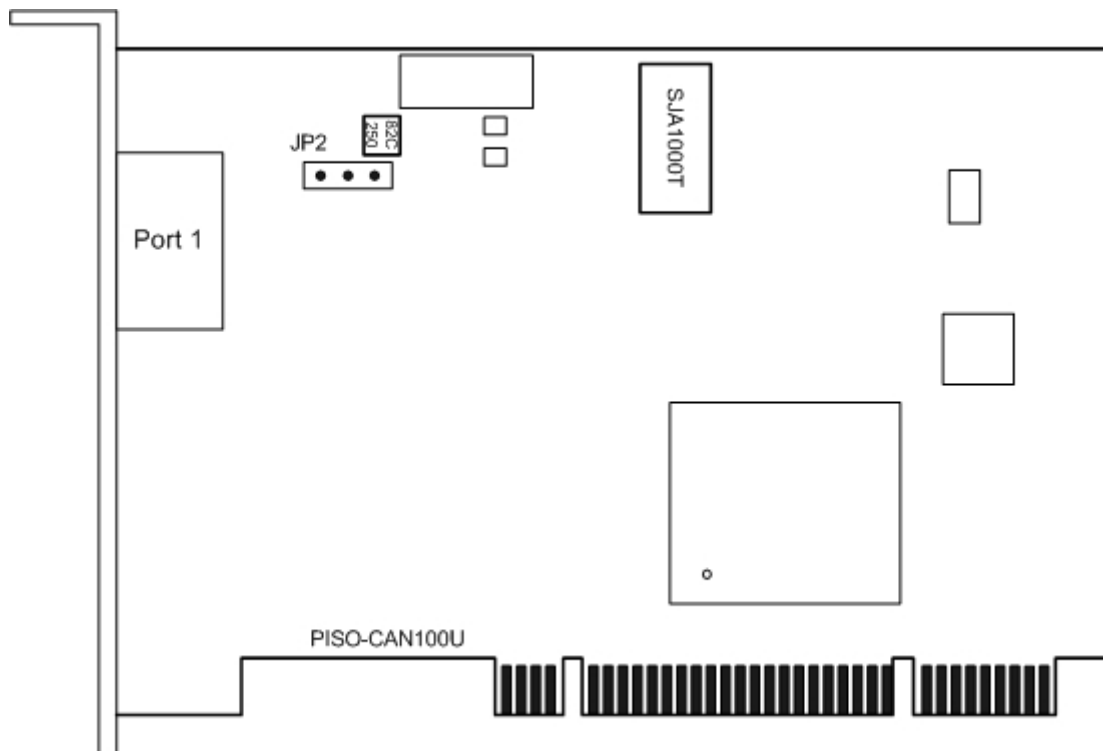


Figure2.3 PISO-CAN100U Board LAYOUT

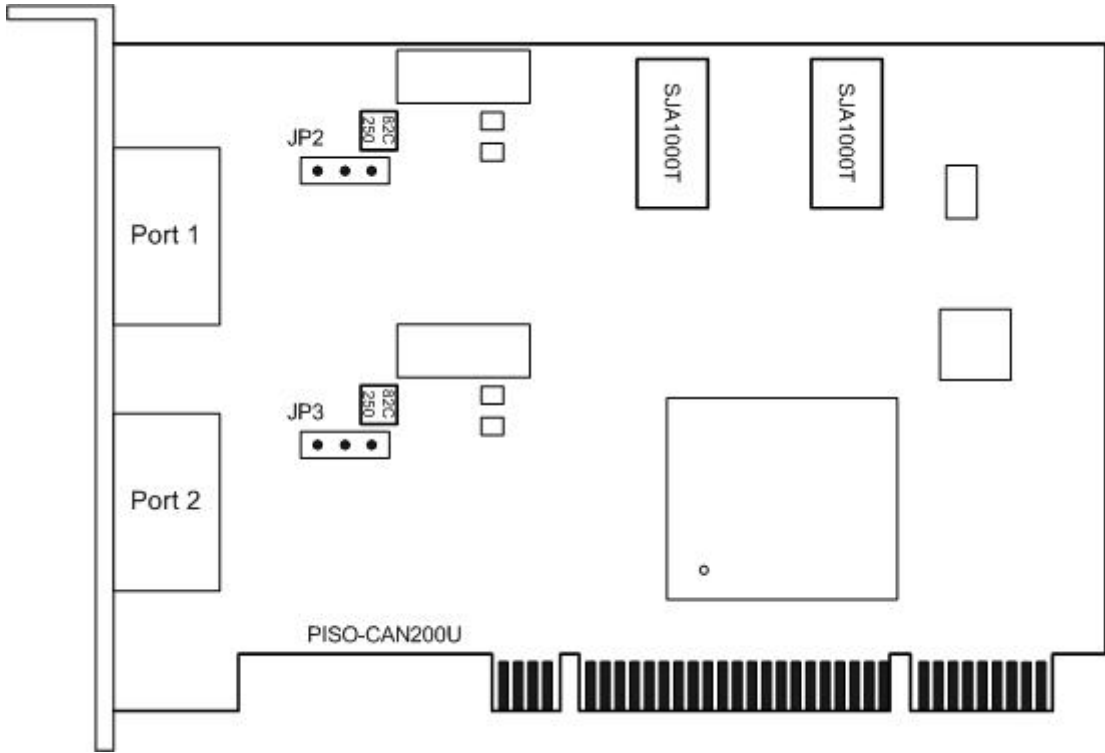


Figure2.4 PISO-CAN200U Board LAYOUT

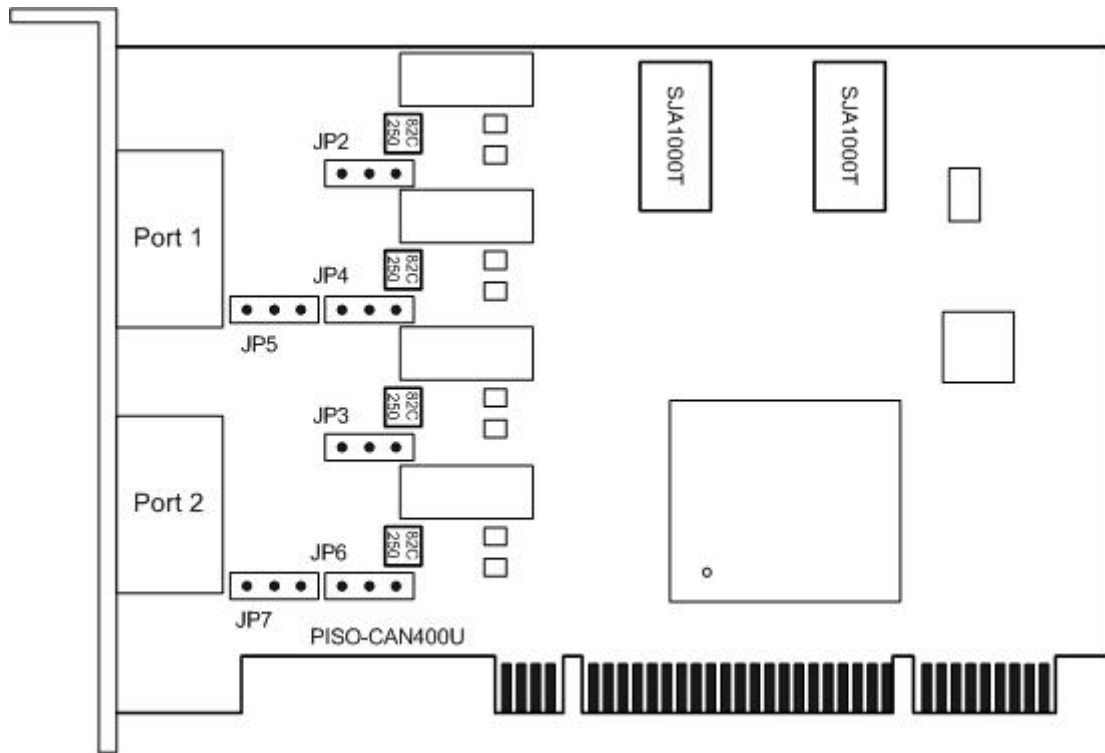


Figure2.5 PISO-CAN400U Board LAYOUT

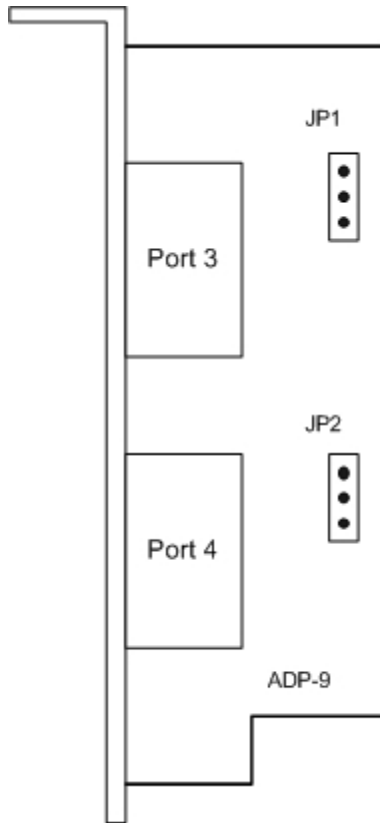


Figure2.6 ADP-9 Board LAYOUT (For PISO-CAN400/400U Only)

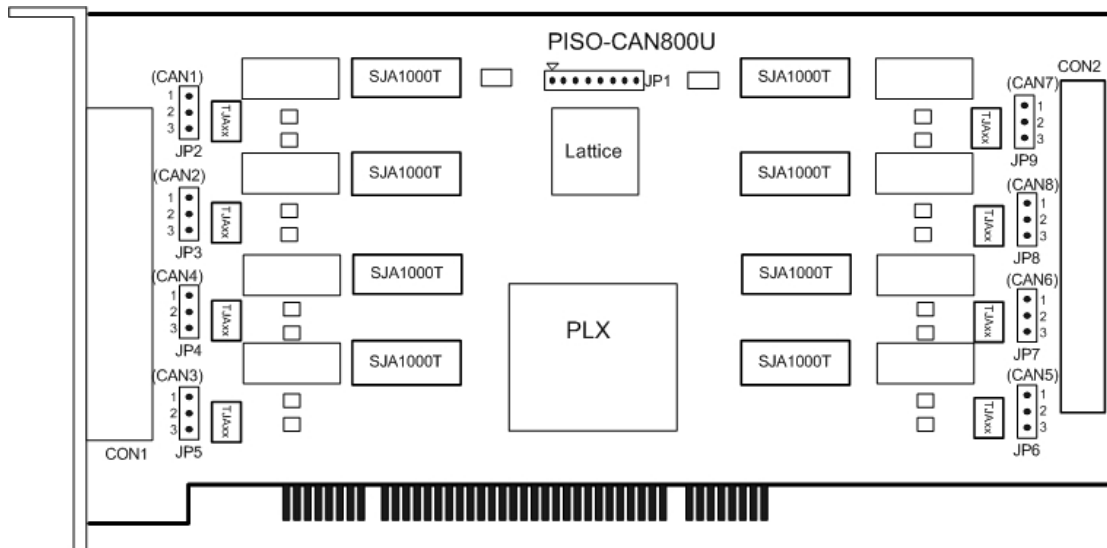


Figure2.7 PISO-CAN800U Board LAYOUT



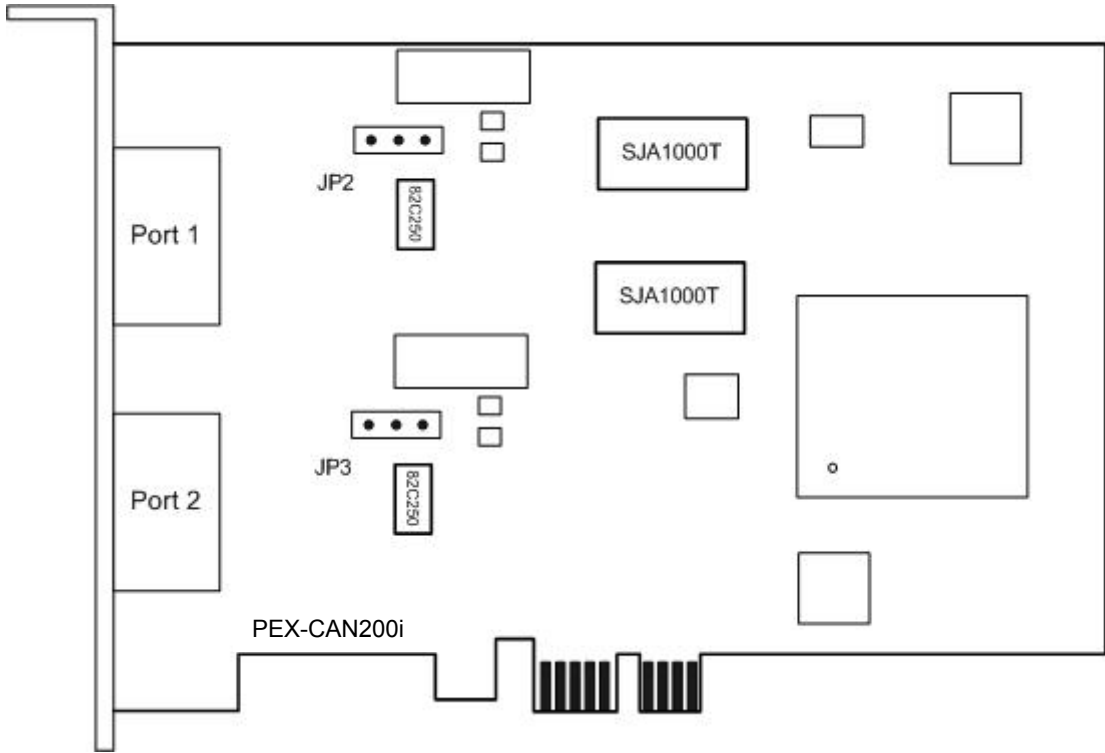


Figure2.8 PEX-CAN200i Board LAYOUT

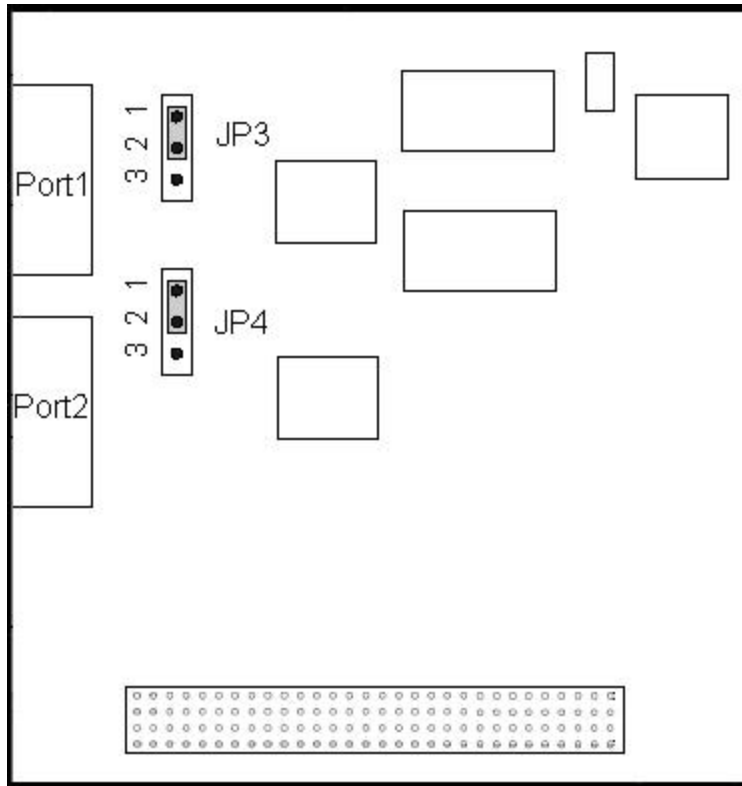

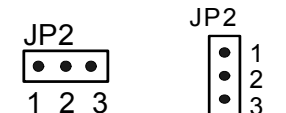
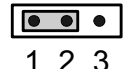
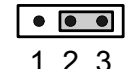
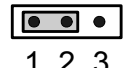
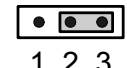
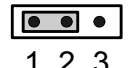
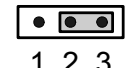
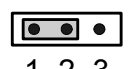
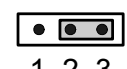
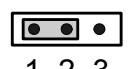
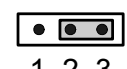
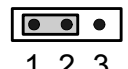
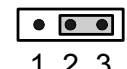
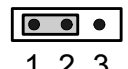
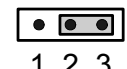


Figure2.9 PCM-CAN200 Board LAYOUT

## 2.2 Jumper Selection





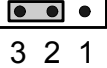
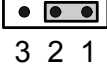
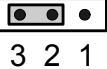
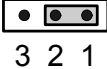
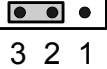
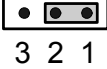
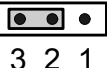
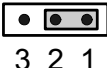
### PISO-CAN200400

Table 2.1 Jumper Selections

Jumper	Description	Status	
JP1	CAN Port 3 Connector, connecting PISO-CAN400 board and ADP-9 board.	 <p>Pin1: CAN_L Pin2: CAN_H Pin3: Shield</p>	
JP2	CAN Port 4 Connector, connecting PISO-CAN400 board and ADP-9 board.	 <p>Pin1: CAN_L Pin2: CAN_H Pin3: Shield</p>	
JP6	Port 1 terminator resistor(120Ω) selection	<b>Enable</b>	<b>Disable</b>
			
JP7	Port 2 terminator resistor(120Ω) selection		
			
JP8	Port 3 terminator resistor(120Ω) selection		
			
JP9	Port 4 terminator resistor(120Ω) selection		
			

## PISO-CAN100U/200U/400U

Table 2.2 Jumper Selections

Jumper	Description	Status	
JP5	CAN Port 3 Connector, connecting PISO-CAN400U board and ADP-9 board.	 Pin1: CAN_L Pin2: CAN_H Pin3: Shield	 Pin1: CAN_L Pin2: CAN_H Pin3: Shield
JP7	CAN Port 4 Connector, connecting PISO-CAN400U board and ADP-9 board.	 Pin1: CAN_L Pin2: CAN_H Pin3: Shield	 Pin1: CAN_L Pin2: CAN_H Pin3: Shield
JP2	Port 1 terminator resistor(120Ω) selection	<b>Enable</b>	<b>Disable</b>
		 3 2 1	 3 2 1
JP3	Port 2 terminator resistor(120Ω) selection (only for PISO-CAN200U/400U)	 3 2 1	 3 2 1
JP4	Port 3 terminator resistor(120Ω) selection (only for PISO-CAN400U)	 3 2 1	 3 2 1
JP6	Port 4 terminator resistor(120Ω) selection (only for PISO-CAN400U)	 3 2 1	 3 2 1





# PISO-CAN800U

Table 2.3 Jumper Selections

Jumper	Description	Status	
		Enable	Disable
JP2	Port 1 terminator resistor(120Ω) selection	1 ● 2 ● 3 ●	1 ● 2 ● 3 ●
JP3	Port 2 terminator resistor(120Ω) selection	1 ● 2 ● 3 ●	1 ● 2 ● 3 ●
JP4	Port 4 terminator resistor(120Ω) selection	1 ● 2 ● 3 ●	1 ● 2 ● 3 ●
JP5	Port 3 terminator resistor(120Ω) selection	1 ● 2 ● 3 ●	1 ● 2 ● 3 ●
JP6	Port 5 terminator resistor(120Ω) selection	1 ● 2 ● 3 ●	1 ● 2 ● 3 ●
JP7	Port 6 terminator resistor(120Ω) selection	1 ● 2 ● 3 ●	1 ● 2 ● 3 ●
JP8	Port 8 terminator resistor(120Ω) selection	1 ● 2 ● 3 ●	1 ● 2 ● 3 ●
JP9	Port 7 terminator resistor(120Ω) selection	1 ● 2 ● 3 ●	1 ● 2 ● 3 ●

## PEX-CAN200i-D/T

Table 2.4 Jumper Selections

Jumper	Description	Status	
		Enable	Disable
JP2	Port 1 terminator resistor(120Ω) selection		
		1 2 3	1 2 3
JP3	Port 2 terminator resistor(120Ω) selection		
		1 2 3	1 2 3





## PCM-CAN100 and PCM-CAN200

Table 2.5 shows the appropriate switch setting and signals used for each module in the stack.

Table 2.5 Rotary Switch Settings

Switch Position	Module Slot	CLK	ID Select	INT
0 or 4 or 8	1	CLK0	IDSEL0	INTA
1 or 5 or 9	2	CLK1	IDSEL1	INTB
2 or 6	3	CLK2	IDSEL2	INTC
3 or 7	4	CLK3	IDSEL3	INTD

Table 2.6 Jumper Selections

Jumper	Description	Status	
		Enable	Disable
JP3	Port 1 terminator resistor(120Ω) selection		
		3 2 1	3 2 1
JP4 (only for PCM-CAN200)	Port 2 terminator resistor(120Ω) selection		
		3 2 1	3 2 1

---

## 2.3 Connector Pin Assignment

The PISO-CAN200/400-T and PISO-CAN100U/200U/400U-T, and PEX-CAN200i-T are equipped with 1/2/4 sets of **5-pin screw terminal connectors**, the PISO-CAN200/400-D and PISO-CAN100U/200U/400U-D, PEX-CAN200i-D and PCM-CAN100/200 are equipped with 1/2/4 sets of **9-pin male D-sub connectors** for wire connection of the CAN bus. And the PISO-CAN800U is equipped with 2 sets of female DB-37 connector. Via the **CA-9-3715D/CA-9-3705** cable, user can convert the female DB-37 connector to **9-pin male D-sub connectors**. The connector's pin assignment is specified as follows:

### 2.3.1 5-pin screw terminal connector

The 5-pin screw terminal connector of the CAN bus interface is shown in Figure 2.8. The details for the pin assignment are presented in Table 2.7.

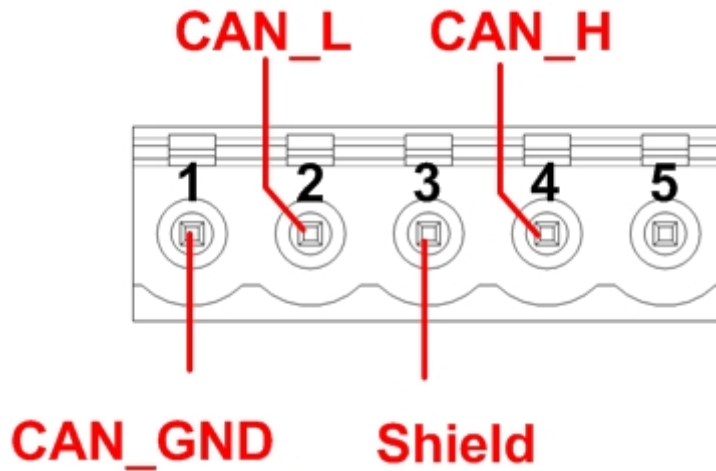


Figure2.9 5-pin screw terminal connector

Table 2.7 Pin assignment of 5-pin screw terminal connector

5-pin screw terminal connectors pin assignment	
1	CAN_GND
2	CAN_L
3	CAN_SHLD
4	CAN_H
5	Reserved

### 2.3.2 9-pin male D-sub connectors

The 9-pin male D-sub connector of the CAN bus interface is shown in Figure 2.9 and the corresponding pin assignments are given in Table 2.8.

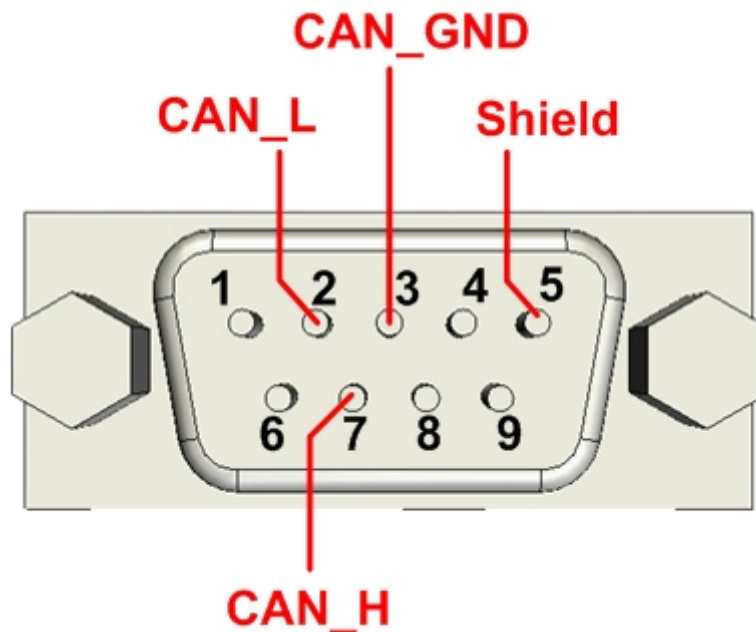


Figure 2.10 9-pin male D-sub connector

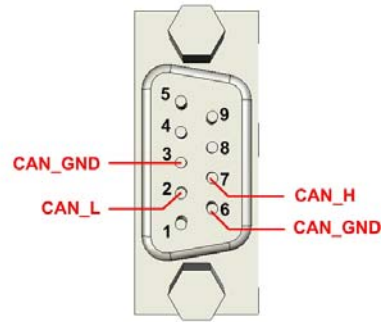
Table 2.8 Pin assignment of the 9-pin male D-sub connector

D-sub male connector pin assignment	
1	Reserved
2	CAN_L
3	CAN_GND
4	Reserved
5	CAN_SHLD
6	Reserved
7	CAN_H
8	Reserved
9	Reserved

### 2.3.3 37-pin female D-sub connectors

#### CA-9-3715D/CA-9-3705 DB-37 to DB-9 Pin Assignment for PISO-CAN800U (CON1)

Pin Assignment Name	Terminal No.	Pin Assignment Name
CAN1_GND	19	37 CAN1_L
CAN1_H	18	36 N.C.
CAN1_GND	17	35 N.C.
N.C.	16	34 N.C.
N.C.	15	33 CAN2_GND
CAN2_L	14	32 CAN2_H
N.C.	13	31 CAN2_GND
N.C.	12	30 N.C.
N.C.	11	29 N.C.
CAN4_GND	10	28 CAN4_L
CAN4_H	09	27 N.C.
CAN4_GND	08	26 N.C.
N.C.	07	25 N.C.
N.C.	06	24 CAN3_GND
CAN3_L	05	23 CAN3_H
N.C.	04	22 CAN3_GND
N.C.	03	21 N.C.
N.C.	02	20 N.C.
N.C.	01	



DB-37 to Male DB-9 Connector\_CAN

37-Pin Female D-Sub Connector\_CAN (CON1)

#### DB-37 Pin Assignment for PISO-CAN800U (CON2)

CON2				Pin Assignment Name	Terminal No.	Pin Assignment Name
1	DB-37 Pin01	2	DB-37 Pin20	CAN5_GND	19	37 CAN5_L
3	DB-37 Pin02	4	DB-37 Pin21	CAN5_H	18	36 N.C.
5	DB-37 Pin03	6	DB-37 Pin22	CAN5_GND	17	35 N.C.
7	DB-37 Pin04	8	DB-37 Pin23	N.C.	16	34 N.C.
9	DB-37 Pin05	10	DB-37 Pin24	N.C.	15	33 CAN6_GND
11	DB-37 Pin06	12	DB-37 Pin25	CAN6_L	14	32 CAN6_H
13	DB-37 Pin07	14	DB-37 Pin26	N.C.	13	31 CAN6_GND
15	DB-37 Pin08	16	DB-37 Pin27	N.C.	12	30 N.C.
17	DB-37 Pin09	18	DB-37 Pin28	N.C.	11	29 N.C.
19	DB-37 Pin10	20	DB-37 Pin29	CAN8_GND	10	28 CAN8_L
21	DB-37 Pin11	22	DB-37 Pin30	CAN8_H	09	27 N.C.
23	DB-37 Pin12	24	DB-37 Pin31	CAN8_GND	08	26 N.C.
25	DB-37 Pin13	26	DB-37 Pin32	N.C.	07	25 N.C.
27	DB-37 Pin14	28	DB-37 Pin33	N.C.	06	24 CAN7_GND
29	DB-37 Pin15	30	DB-37 Pin34	CAN7_L	05	23 CAN7_H
31	DB-37 Pin16	32	DB-37 Pin35	N.C.	04	22 CAN7_GND
33	DB-37 Pin17	34	DB-37 Pin36	N.C.	03	21 N.C.
35	DB-37 Pin18	36	DB-37 Pin37	N.C.	02	20 N.C.
37	DB-37 Pin19	38	N.C.	N.C.	01	
39	N.C.	40	N.C.			

37-Pin Female D-Sub Connector\_CAN (CON2)



---

## **2.4 Installation**

1. Configure the jumper settings on your PISO-CAN/PEX-CAN/PCM-CAN in accordance with your particular requirements.
2. Shutdown your system and take off the chassis of your machine.
3. Plug in your PISO-CAN, PEX-CAN, or PCM-CAN series CAN card into a suitable empty PCI slot.
4. Replace your chassis.
5. Plug your CAN bus cable(s) into the 5-pin screw terminal connector or the 9-pin D-sub connector.
6. When the hardware installation is complete, please turn on the computer again.

---

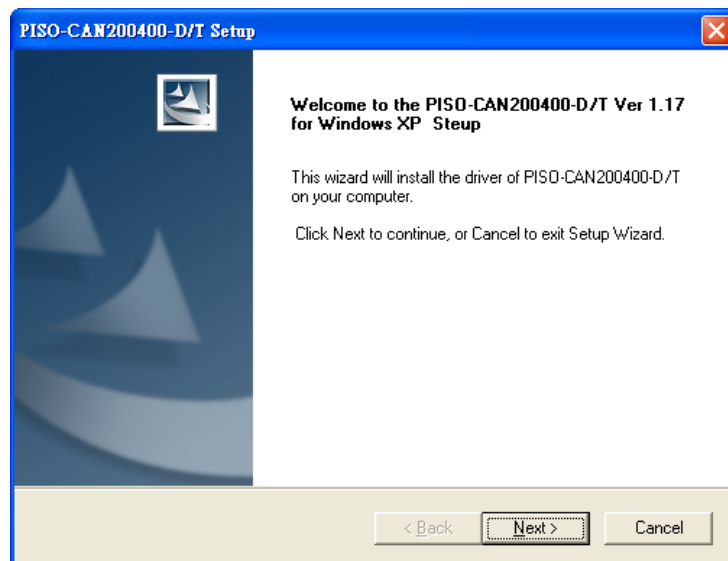
### 3 Software Installation

The driver of PISO-CAN or PCM-CAN can be used in 2K/XP/7 Windows environments. Users can find the driver in the path of “\CAN\PCI\PCM\_PISO-CAN\_series\driver\” in the Fieldbus\_CD. Execute the PISO-CAN.exe file to start install the driver.

#### **Install the PISO-CAN or PCM-CAN card driver**

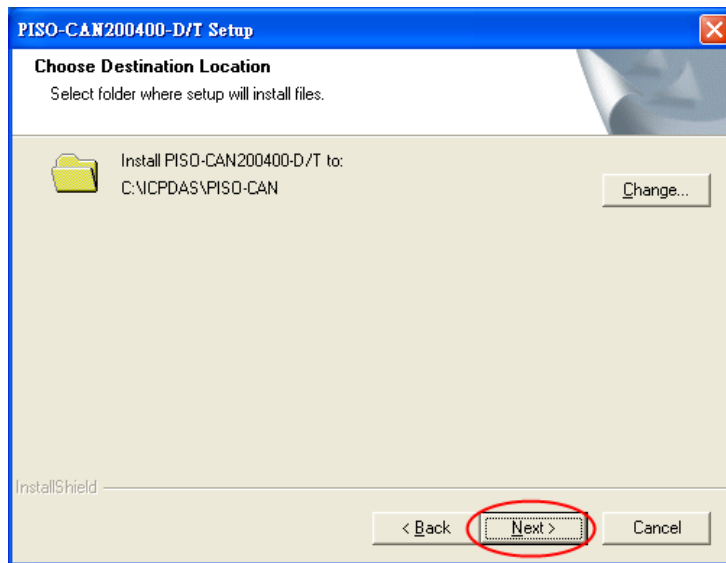
**Step 1:** Insert the product CD into the CD-ROM and find the path \CAN\PCI\ PCM\_PISO-CAN\_series\Driver\win2k\_xp\_7\ (ex: the OS is Windows 2000/XP/7). Then execute the PISO-CAN.exe to install the PISO-CAN card driver.

**Step 2:** Click “Next” to start the PISO-CAN installation.

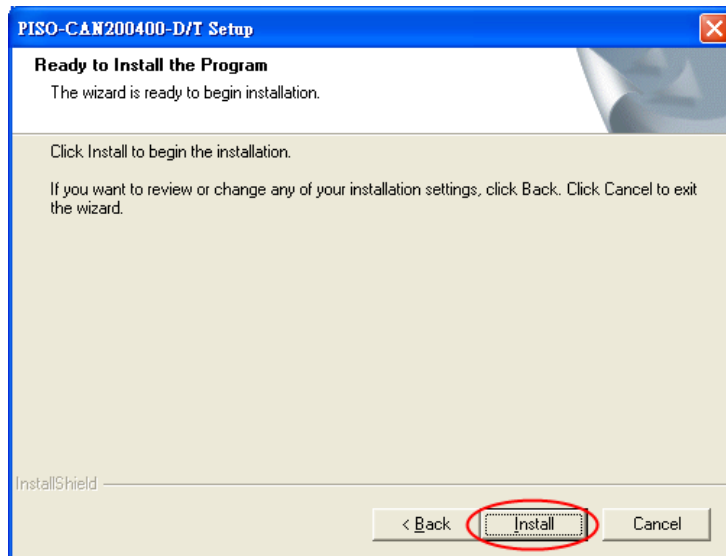


---

**Step 3:** Select the folder where the PISO-CAN setup would be installed and click “Next” button to continue.



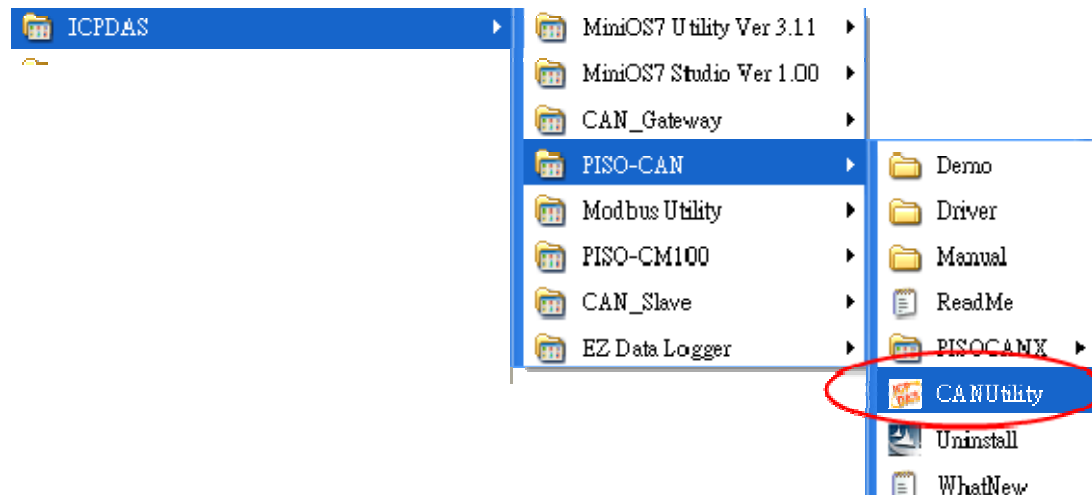
**Step 4:** Click the button “Install” to continue.



**Step 5:** Finally, restart the computer to complete the installation.

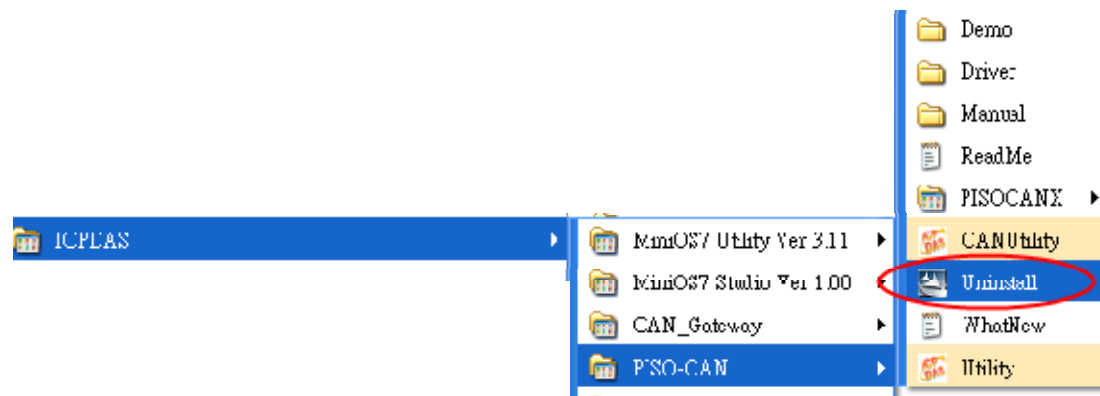


When finishing the installation. The PISO-CAN folder would be found at the Start menu shown as below.



### **Remove the PISO-CAN driver**

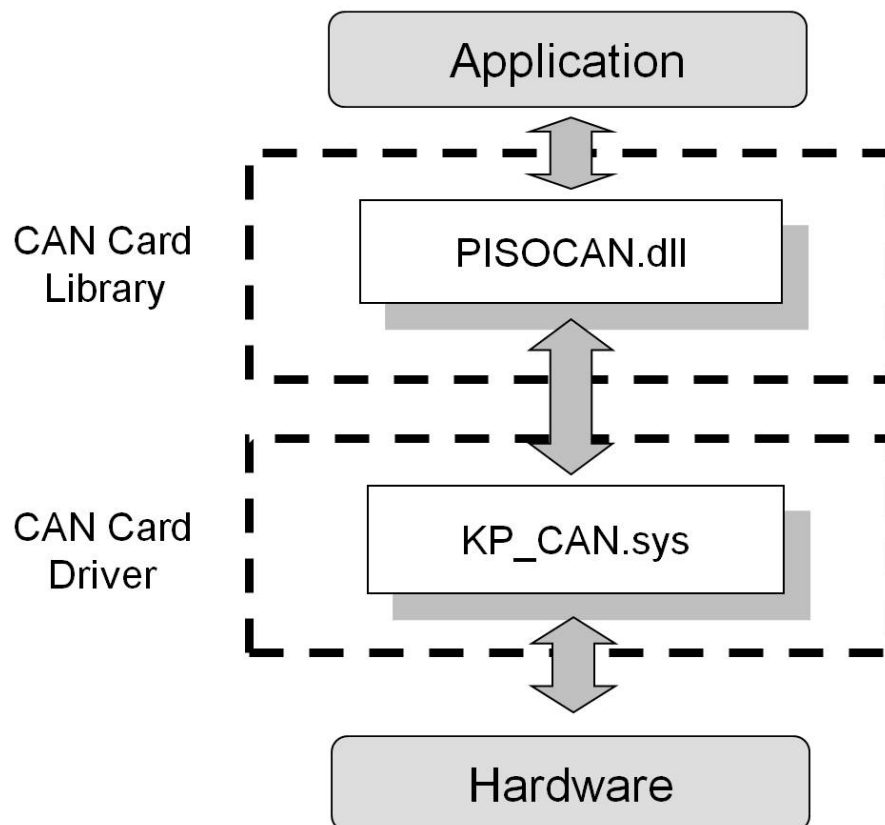
If the PISO-CAN driver is not used any more, users can click the “Uninstall” to remove the PISO-CAN driver below.



---

## 4 Installation DLL Driver

The DLL driver is the collection of function calls on the PISO-CAN, PEX-CAN and PCM-CAN series cards used for Windows 98/Me/NT4/2000/XP systems. The application structure is presented in the following figure. The user application programs which have been developed by the following designated tools: VB, VC, Delphi and Borland C++ Builder...etc, can call the PISOCAN.DLL driver in user mode. And then the DLL driver will bypass the function call into the KP\_CAN.sys to access the hardware system, as shown in the following Figure.



---

## 4.1 DLL Function Definition and Description

All the functions provided in the PISO-CAN, PEX-CAN, or PCM-CAN (**hereinafter referred to as PISO-CAN**) are listed in the following table and detailed information for every function is presented in the following sub-section. However, in order to make the descriptions more simplified and clear, the attributes for the both the input and output parameter functions are given as **[input]** and **[output]** respectively, as shown in following table.

Keyword	Set parameter by user before calling this function?	Get the data from this parameter after calling this function?
[ input ]	Yes	No
[ output ]	No	Yes

Table 4.1 DLL function definition

Function definition	Section
WORD CAN_GetDllVersion();	<a href="#">4.1.1</a>
int CAN_TotalBoard();	<a href="#">4.1.2</a>
int CAN_GetBoardInf(BYTE BoardNo, DWORD *dwVID, DWORD *dwDID, DWORD *dwSVID, DWORD *dwSDID, DWORD *dwIrqNo);	<a href="#">4.1.3</a>
int CAN_GetCardPortNum(BYTE BoardNo, BYTE *bGetPortNum);	<a href="#">4.1.4</a>
int CAN_ActiveBoard(BYTE wBoardNo)	<a href="#">4.1.5</a>
int CAN_CloseBoard(BYTE wBoardNo);	<a href="#">4.1.6</a>
int CAN_BoardIsActive(BYTE BoardNo);	<a href="#">4.1.7</a>
int CAN_Reset(BYTE BoardNo, BYTE Port);	<a href="#">4.1.8</a>
int CAN_Init(BYTE wBoardNo, BYTE Port);	<a href="#">4.1.9</a>
int CAN_Config(BYTE BoardNo, BYTE Port, ConfigStruct *CanConfig);	<a href="#">4.1.10</a>
int CAN_ConfigWithoutStructure(BYTE BoardNo, BYTE Port, DWORD AccCode, DWORD AccMask, BYTE BaudRate, BYTE BT0, BYTE BT1);	<a href="#">4.1.11</a>
int CAN_EnableRxIrq(BYTE BoardNo, BYTE Port);	<a href="#">4.1.12</a>
int CAN_DisableRxIrq(BYTE BoardNo, BYTE Port);	<a href="#">4.1.13</a>
int CAN_RxIrqStatus(BYTE BoardNo, BYTE Port, BYTE *bStatus);	<a href="#">4.1.14</a>
int CAN_InstallIrq(BYTE BoardNo);	<a href="#">4.1.15</a>
int CAN_RemoveIrq(BYTE BoardNo);	<a href="#">4.1.16</a>
int CAN_IrqStatus(BYTE BoardNo, BYTE *bStatus);	<a href="#">4.1.17</a>
int CAN_Status(BYTE BoardNo, BYTE Port, BYTE *bStatus);	<a href="#">4.1.18</a>
int CAN_SendMsg(BYTE BoardNo, BYTE Port, PacketStruct *CanPacket);	<a href="#">4.1.19</a>
int CAN_SendWithoutStruct(BYTE BoardNo, BYTE Port, BYTE Mode, DWORD Id, BYTE Rtr, BYTE Dlen, BYTE *Data)	<a href="#">4.1.20</a>
int CAN_RxMsgCount(BYTE BoardNo, BYTE Port);	<a href="#">4.1.21</a>
int CAN_ReceiveMsg(BYTE BoardNo, BYTE Port, PacketStruct *CanPacket);	<a href="#">4.1.22</a>
int CAN_ReceiveWithoutStruct(BYTE BoardNo, BYTE Port, BYTE *Mode, DWORD *Id, BYTE *Rtr, BYTE *Dlen, BYTE *Data, LONGLONG *MsgTimeStamps);	<a href="#">4.1.23</a>
int CAN_ClearSoftBuffer(BYTE BoardNo, BYTE Port);	<a href="#">4.1.24</a>
int CAN_ClearDataOverrun(BYTE BoardNo, BYTE Port);	<a href="#">4.1.25</a>
void CAN_OutputByte(BYTE BoardNo, BYTE Port, WORD wOffset, BYTE bValue);	<a href="#">4.1.26</a>
BYTE CAN_InputByte(BYTE BoardNo, BYTE Port, WORD wOffset);	<a href="#">4.1.27</a>
LONGLONG CAN_GetSystemFreq(void);	<a href="#">4.1.28</a>
Int CAN_InstallUserIsr(BYTE BoardNo, void(*UserIsr)(BYTE BoardNo));	<a href="#">4.1.29</a>
Int CAN_RemoveUserIsr(BYTE BoardNo);	<a href="#">4.1.30</a>

Table 4.2 Interpretation of the return code

Return Code	Error ID	Comment
0	CAN_NoError	OK
1	CAN_DriverError	Driver error
2	CAN_ActiveBoardError	This board can't be activated.
3	CAN_BoardNumberError	The board number exceeds the maximum board number (7).
4	CAN_PortNumberError	The port number exceeds the maximum port number.
5	CAN_ResetError	CAN chip hardware reset error
6	CAN_SoftResetError	CAN chip software reset error
7	CAN_InitError	CAN chip initiation error
8	CAN_ConfigError	CAN chip configure error
9	CAN_SetACRError	Set to Acceptance Code Register error
10	CAN_SetAMRError	Set to Acceptance Mask Register error
11	CAN_SetBaudRateError	Set Baud Rate error
12	CAN_EnableRxIrqFailure	Enable CAN chip receive interrupt failure
13	CAN_DisableRxIrqFailure	Disable CAN chip receive interrupt failure
14	CAN_InstallIrqFailure	Installing PCI board IRQ failure
15	CAN_RemoveIrqFailure	Removing PCI board IRQ failure
16	CAN_TransmitBufferLocked	Transmit buffer in CAN chip is locked
17	CAN_TransmitIncomplete	Previously transmission is not yet completed
18	CAN_ReceiveBufferEmpty	CAN chip RXFIFO is empty
19	CAN_DataOverrun	Data was lost because there was not enough space in CAN chip RXFIFO
20	CAN_ReceiveError	Receive data is not completed
21	CAN_SoftBufferIsEmpty	Software buffer in driver is empty
22	CAN_SoftBufferIsFull	Software buffer in driver is full
23	CAN_TimeOut	Function no response and timeout
24	CAN_InstallIsrError	Installing user ISR failure



---

### **4.1.1 CAN\_GetDllVersion**

- **Description:**  
Obtain the version information of PISOCAN.dll driver.
- **Syntax:**  
WORD CAN\_GetDllVersion(void)
- **Parameter:**  
None
- **Return:**  
DLL version information. For example: If 101(hex) is return, it means driver version is 1.01.

### **4.1.2 CAN\_TotalBoard**

- **Description:**  
Obtain the amount of all CAN boards installed in the PCI bus.
- **Syntax:**  
int CAN\_TotalBoard(void)
- **Parameter:**  
None
- **Return:**  
Return the amount of all board.

---

### 4.1.3 CAN\_GetBoardInf

- **Description:**

Obtain the information of PISO-CAN boards, which include vender ID, device ID and interrupt number.

- **Syntax:**

```
int CAN_GetBoardInf(BYTE BoardNo, DWORD *dwVID, DWORD *dwDID, DWORD *dwSVID, DWORD *dwSDID, DWORD *dwSAuxID, DWORD *dwIrqNo)
```

- **Parameter:**

BoardNo:	[input] PISO-CAN board number
*dwVID:	[output] vendor ID of this board
*dwDID:	[output] device ID of this board
*dwSVID:	[output] sub-vendor ID of this board
*dwSDID:	[output] sub-device ID of this board
*dwSAuxID:	[output] sub-auxiliary ID of this board
*dwIrq:	[output] logical interrupt number of this board

- **Return:**

CAN\_NoError: OK

CAN\_DriverError: Kernel driver can not be opened.

CAN\_BoardNumberError: BoardNo exceeds the current total board number.

---

#### 4.1.4 CAN\_GetCardPortNum

- **Description:**

Call this function to Get CAN port number of the PISO-CAN card.

- **Syntax:**

```
int CAN_GetCardPortNum(BYTE BoardNo, BYTE *bGetPortNum)
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7).

\* bGetPortNum: [output] Port number of the CAN card

- **Return:**

CAN\_NoError: OK

CAN\_DriverError: Kernel driver can not be opened.

CAN\_BoardNumberError: BoardNo exceeds the current total board  
number.

---

### 4.1.5 CAN\_ActiveBoard

- **Description:**

Activate the device. **It must be called once before using other functions of PISO-CAN board.**

- **Syntax:**

int CAN\_ActiveBoard(BYTE BoardNo)

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7).

- **Return:**

CAN\_NoError: OK

CAN\_BoardNumberError: BoardNo exceeds the current total board number.

CAN\_ActiveBoardError: This board can not be activated or kernel driver can not be found.

---

### 4.1.6 CAN\_CloseBoard

- **Description:**

Stop and close the kernel driver and release the device resource from computer device resource. This method must be called once before exiting the user's application program.

- **Syntax:**

int CAN\_CloseBoard(BYTE BoardNo)

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7).

- **Return:**

CAN\_NoError: OK

CAN\_ActiveBoardError: The board is not activated

CAN\_BoardNumberError: BoardNo exceeds the current total board  
number.

---

### 4.1.7 CAN\_BoardIsActive

- **Description:**

Obtain the information about the specific board is active or not.

- **Syntax:**

```
int CAN_BoardIsActive(BYTE BoardNo)
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7).

- **Return:**

0: means the board is inactive.

1: means the board is active.

---

### 4.1.8 CAN\_Reset

- **Description:**

Hardware reset CAN controller.

- **Syntax:**

int CAN\_Reset(BYTE BoardNo, BYTE Port)

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7).

Port: [input] CAN port number (1~4 or 1~2)

- **Return:**

CAN\_NoError: OK

CAN\_DriverError: Kernel driver can't be opened.

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_PortNumberError: Port number is not correct.

CAN\_ActiveBoardError: This board is not activated.

---

### 4.1.9 CAN\_Init

- **Description:**

Initiate CAN controller.

- **Syntax:**

int CAN\_Init(BYTE BoardNo, BYTE Port)

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7).

Port: [input] CAN port number (1~4 or 1~2)

- **Return:**

CAN\_NoError: OK

CAN\_DriverError: Kernel driver can't be opened.

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_PortNumberError: Port number is not correct.

CAN\_ActiveBoardError: This board is not activated.

CAN\_InitError: Initiating CAN controller failure



---

#### 4.1.10 CAN\_Config

- **Description:**

Configure CAN controller. After calling this function, the CAN controller will enter operating mode.

- **Syntax:**

```
int CAN_Config(BYTE BoardNo, BYTE Port,ConfigStruct
               *CanConfig);
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7).

Port: [input] CAN port number (1~4 or 1~2)

\*ConfigStruct: [input] The point of structure for ConfigStruct is defined as

following,

```
typedef struct config
{
    BYTE AccCode[4];
    BYTE AccMask[4];
    BYTE BaudRate;
    BYTE BT0, BT1;
} ConfigStruct;
```

AccCode[4]: Acceptance code for CAN controller.

AccMask[4]: Acceptance mask for CAN controller.

BaudRate: 0→user-defined(must to set BT0,BT1), 1→10Kbps,

2→20Kbps, 3→50Kbps, 4→125Kbps, 5→250Kbps,

6→500Kbps, 7→800Kbps, 8→1Mbps.

BT0, BT1: user-defined baud rate (used only if BaudRate=0)). For

---

example, BT0=0x04, BT1=0x1C, then baud rate setting for the CAN controller is 100Kbps. For more detail baud rate setting, please refer to manual of SJA1000 CAN controller.

- **Return:**

CAN\_NoError: OK

CAN\_DriverError: Kernel driver can't be opened.

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current total board number.

CAN\_PortNumberError: Port number is not correct.

CAN\_ActiveBoardError: This board is not activated.

CAN\_SoftResetError: CAN controller software reset error.

CAN\_SetACRError: Set Acceptance code to CAN controller error

CAN\_SetAMRError: Set Acceptance mask to CAN controller error

CAN\_SetBaudRateError: Set baud rate to CAN controller error

CAN\_ConfigError: CAN controller enter operating mode failure.

---

#### 4.1.11 CAN\_ConfigWithoutStructure

- **Description:**

This function is the same as **CAN\_Config**. But this function doesn't use ConfigStruct structure type. To provide this function is for that the structure address of some application development is allocated different from the PISOCAN.lib. So if users use CAN\_Config and can't configure CAN card correctly, the CAN\_ConfigWithoutStruct function can instead.

- **Syntax:**

```
int CAN_ConfigWithoutStructure(BYTE BoardNo, BYTE Port,  
                               DWORD AccCode, DWORD AccMask, BYTE BaudRate,  
                               BYTE BT0, BYTE BT1);
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7).

Port: [input] CAN port number (1~4 or 1~2)

AccCode: Acceptance code for CAN controller.

AccMask: Acceptance mask for CAN controller.

BaudRate: 0→user-defined(must to set BT0,BT1), 1→10Kbps,  
2→20Kbps, 3→50Kbps, 4→125Kbps, 5→250Kbps,  
6→500Kbps, 7→800Kbps, 8→1Mbps.

BT0, BT1: user-defined baud rate (used only if BaudRate=0)). For example, BT0=0x04, BT1=0x1C, then baud rate setting for the CAN controller is 100Kbps. For more detail baud rate setting, please refer to manual of SJA1000 CAN controller.

- **Return:**

CAN\_NoError: OK

---

CAN\_DriverError: Kernel driver can't be opened.

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_PortNumberError: Port number is not correct.

CAN\_ActiveBoardError: This board is not activated.

CAN\_SoftResetError: CAN controller software reset error.

CAN\_SetACRError: Set Acceptance code to CAN controller error

CAN\_SetAMRError: Set Acceptance mask to CAN controller error

CAN\_SetBaudRateError: Set baud rate to CAN controller error

CAN\_ConfigError: CAN controller enter operating mode failure.

---

#### 4.1.12 *CAN\_EnableRxIrq*

- **Description:**

Enable receive interrupt for CAN controller.

- **Syntax:**

int CAN\_EnableRxIrq(BYTE BoardNo, BYTE Port)

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7).

Port: [input] CAN port number (1~4 or 1~2)

- **Return:**

CAN\_NoError: OK

CAN\_DriverError: Kernel driver can't be opened.

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_PortNumberError: Port number is not correct.

CAN\_ActiveBoardError: This board is not activated.

CAN\_EnableRxIrqFailure: Enable receives interrupt failure.

---

### 4.1.13 CAN\_DisableRxIrq

- **Description:**

Disable receive interrupt of the CAN controller.

- **Syntax:**

Int CAN\_DisableRxIrq(BYTE BoardNo, BYTE Port)

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7)

Port: [input] CAN port number (1~4 or 1~2)

- **Return:**

CAN\_NoError: OK

CAN\_DriverError: Kernel driver can't be opened.

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_PortNumberError: Port number is not correct.

CAN\_ActiveBoardError: This board is not activated.

CAN\_DisableRxIrqFailure: Disable receives interrupt failure.

---

#### 4.1.14 CAN\_RxIrqStatus

- **Description:**

Obtain receive interrupt status of the CAN controller.

- **Syntax:**

int CAN\_RxIrqStatus(BYTE BoardNo, BYTE Port, BYTE \*bStatus)

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7)

Port: [input] CAN port number (1~4 or 1~2)

\*bStatus:[output] 0→receive interrupt disable;

1→ receive interrupt enable.

- **Return:**

CAN\_NoError: OK

CAN\_DriverError: Kernel driver can't be opened.

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_PortNumberError: Port number is not correct.

CAN\_ActiveBoardError: This board is not activated.

---

#### 4.1.15 CAN\_InstallIrq

- **Description:**

Enable or start IRQ for PISO-CAN board. Before calling this function, **CAN\_EnableRxIrq** must to be called first.

- **Syntax:**

```
int CAN_InstallIrq(BYTE BoardNo)
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7).

- **Return:**

CAN\_NoError: OK

CAN\_DriverError: Kernel driver can't be opened.

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_ActiveBoardError: This board is not activated.

CAN\_InstallIrqFailure: Enable or start IRQ failure.



---

#### 4.1.16 CAN\_RemoveIrq

- **Description:**

Disable or stop IRQ for PISO-CAN board. After calling this function, the interrupts for all CAN controllers on board will be disabled.

- **Syntax:**

```
int CAN_RemoveIrq(BYTE BoardNo)
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7).

- **Return:**

CAN\_NoError: OK

CAN\_DriverError: Kernel driver can't be opened.

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_ActiveBoardError: This board is not activated.

CAN\_RemoveIrqFailure: Disable or stop IRQ failure.

---

#### 4.1.17 CAN\_IrqStatus

- **Description:**

Obtain IRQ status of the PISO-CAN board.

- **Syntax:**

```
int CAN_IrqStatus(BYTE BoardNo, BYTE *bStatus)
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7).

\*bStatus:[output] 0→IRQ disable;

1→ IRQ enable.

- **Return:**

CAN\_NoError: OK

CAN\_DriverError: Kernel driver can't be opened.

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_ActiveBoardError: This board is not activated.

---

#### 4.1.18 CAN\_Status

- **Description:**

Obtain the status of CAN controller for PISO-CAN board.

- **Syntax:**

```
int CAN_Status(BYTE BoardNo, BYTE Port, BYTE *bStatus)
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7).

\*bStatus:[output] Status value of CAN controller.

Table 4.3 Bit interpretation of the bStatus.

Bit	NAME	VALUE	STATUS
bit 7	Bus Status	1	bus-off
		0	bus-on
bit 6	Error Status	1	error
		0	ok
bit 5	Transmit Status	1	transmit
		0	idle
bit 4	Receive Status	1	receive
		0	idle
bit 3	Transmission Complete Status	1	complete
		0	incomplete
bit 2	Transmit Buffer Status	1	release
		0	locked
bit 1	Data Overrun Status	1	overrun
		0	absent
bit 0	Receive Buffer Status	1	full/not empty
		0	empty

- **Return:**

CAN\_NoError: OK

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_PortNumberError: Port number is not correct.

CAN\_ActiveBoardError: This board is not activated.

---

#### 4.1.19 CAN\_SendMsg

- **Description:**

Send a CAN message immediately.

- **Syntax:**

```
int CAN_SendMsg(BYTE BoardNo, BYTE Port, PacketStruct  
                *CanPacket)
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7)

Port: [input] CAN port number (1~4 or 1~2)

\*CanPacket: [input] The point of structure for CanPacket is defined as  
following,

```
typedef struct packet  
{  
    LONGLONG MsgTimeStamps;  
    BYTE mode;  
    DWORD id;  
    BYTE rtr;  
    BYTE len;  
    BYTE data[8];  
} PacketStruct;
```

MsgTimeStamps: Not use in this function.

mode: 0 → 11-bit identifier, 1 → 29-bit identifier.

id: Identifier

rtr: Remote transmission request

len: Data length

data[8]: data byte

---

- **Return:**

CAN\_NoError: OK

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_PortNumberError: Port number is not correct.

CAN\_ActiveBoardError: This board is not activated.

CAN\_TransmitBufferLocked: Transmit buffer in CAN chip is locked.

CAN\_TransmitIncomplete: Transmission is not yet completed.

CAN\_ConfigError: Port has not been configured successfully.

---

#### 4.1.20 CAN\_SendWithoutStruct

- **Description:**

This function is the same as **CAN\_SendMsg**. But this function doesn't use PacketStruct structure type. If users use CAN\_SendMsg and can't send CAN message correctly with some application development like dot Net 2003, the CAN\_SendWithoutStruct function can instead.

- **Syntax:**

```
int CAN_SendWithoutStruct(BYTE BoardNo, BYTE Port, BYTE Mode,  
                          DWORD Id, BYTE Rtr, BYTE Dlen, BYTE *Data)
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7)

Port: [input] CAN port number (1~4 or 1~2)

Mode: 0 → 11-bit identifier, 1 → 29-bit identifier.

Id: Identifier

Rtr: Remote transmission request

Dlen: Data length

\*Data: data byte

- **Return:**

CAN\_NoError: OK

CAN\_BoardNumberError: BoardNo is not correct.

CAN\_PortNumberError: Port number is not correct.

CAN\_ActiveBoardError: This board is not activated.

CAN\_TransmitBufferLocked: Transmit buffer in CAN chip is locked.

CAN\_TransmitIncomplete: Transmission is not yet completed.

CAN\_ConfigError: Port has not been configured successfully.

---

#### 4.1.21 CAN\_RxMsgCount

- **Description:**

Obtain the amount of CAN messages available within the CAN controller's RXFIFO or the software buffer (4KBytes). After calling the functions **CAN\_EnableRxIrq** and **CAN\_InstallIrq**, the amount of CAN messages is within the software buffer; otherwise it is within the CAN controller's RXFIFO.

- **Syntax:**

```
int CAN_RxMsgCount(BYTE BoardNo, BYTE Port);
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7)

Port: [input] CAN port number (1~4 or 1~2)

- **Return:**

The amount of CAN messages.

**Note.** If the parameter for *BoardNo* or *Port* isn't correct, the return value will always be 0.

---

### 4.1.22 CAN\_ReceiveMsg

- **Description:**

Obtain receive message from CAN controller's RXFIFO or software buffer. After calling the functions **CAN\_EnableRxIrq** and **CAN\_InstallIrq**, the messages is within the software buffer, otherwise it is within the CAN controller's RXFIFO.

**Note! If users' PC go into "Standby mode" or "Sleep mode", this function will can't receive any message.**

- **Syntax:**

```
int CAN_ReceiveMsg(BYTE BoardNo, BYTE Port, PacketStruct  
                  *CanPacket)
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7)

Port: [input] CAN port number (1~4 or 1~2)

\*CanPacket: [output] The structure for CanPacket is defined below,

```
typedef struct packet  
{  
    LONGLONG MsgTimeStamps;  
    BYTE mode;  
    DWORD id;  
    BYTE rtr;  
    BYTE len;  
    BYTE data[8];  
} PacketStruct;
```

MsgTimeStamps: This parameter in Windows 98/Me/NT4 will record the



---

time with system clock counter and in Windows 2000 /XP will record the system interrupt-time count of 100-ns unit when the CAN message is received from SJA1000. The system clock counter starts to count after the PC boots up. If more than one CAN messages are received and stored in the 64-byte SJA1000 FIFO, the time stamps of these CAN messages may be closed.

mode: 0 → 11-bit identifier, 1 → 29-bit identifier.

id: Identifier

rtr: Remote transmission request

len: Data length

data[8]: data byte

- **Return:**

CAN\_NoError: OK

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current total board number.

CAN\_PortNumberError: Port number is not correct.

CAN\_ActiveBoardError: This board is not activated.

CAN\_ConfigError: Port has not been configured successfully.

CAN\_ReceiveBufferEmpty: CAN controller's RXFIFO is empty.

CAN\_SoftBufferIsEmpty: Software RX Buffer is empty.

CAN\_SoftBufferIsFull: Software RX Buffer Is full.

---

### 4.1.23 CAN\_ReceiveWithoutStruct

- **Description:**

This function is the same as **CAN\_ReceiveMsg**. But this function doesn't use PacketStruct structure type. To provide this function is for that the structure address of some application development is allocated different from the PISOCAN.lib like dot Net 2003. So if users use CAN\_ReceiveMsg and can't receive CAN message correctly, the CAN\_ReceiveWithoutStruct function can instead.

- **Syntax:**

```
int CAN_ReceiveWithoutStruct(BYTE BoardNo, BYTE Port, BYTE
                             *Mode, DWORD *Id, BYTE *Rtr, BYTE *Dlen,
                             BYTE *Data, DWORD *H_MsgTimeStamps,
                             DWORD *L_MsgTimeStamps)
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7)

Port: [input] CAN port number (1~4 or 1~2)

\*Mode: 0 → 11-bit identifier, 1 → 29-bit identifier.

\*Id: Identifier

\*Rtr: Remote transmission request

\*Dlen: Data length

\*Data: data byte

\*H\_MsgTimeStamps, \*L\_MsgTimeStamps: These parameters in Windows 98/Me/NT4 will record the time with system clock counter and in Windows 2000/XP will record the system interrupt-time count of 100-ns unit when the CAN message is received from SJA1000. The \*H\_MsgTimeStamps is the high

---

DWORD and the \*L\_MsgTimeStamps is the low DWORD The system clock counter starts to count after the PC boots up. If more than one CAN messages are received and stored in the 64-byte SJA1000 FIFO, the time stamps of these CAN messages may be closed.

- **Return:**

CAN\_NoError: OK

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current total board number.

CAN\_PortNumberError: Port number is not correct.

CAN\_ActiveBoardError: This board is not activated.

CAN\_ConfigError: Port has not been configured successfully.

CAN\_ReceiveBufferEmpty: CAN controller's RXFIFO is empty.

CAN\_SoftBufferIsEmpty: Software RX Buffer is empty.

CAN\_SoftBufferIsFull: Software RX Buffer is full.

---

#### 4.1.24 CAN\_ClearSoftBuffer

- **Description:**

Clear the software buffer of the PISOCAN.DLL driver.

- **Syntax:**

int CAN\_ClearSoftBuffer(BYTE BoardNo, BYTE Port)

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7)

Port: [input] CAN port number (1~4 or 1~2)

- **Return:**

CAN\_NoError: OK

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_PortNumberError: Port number is not correct.

---

#### 4.1.25 CAN\_ClearDataOverrun

- **Description:**

Clear the data overrun status bit for the CAN controller.

- **Syntax:**

int CAN\_ClearDataOverrun(BYTE BoardNo, BYTE Port)

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7)

Port: [input] CAN port number (1~4 or 1~2)

- **Return:**

CAN\_NoError: OK

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_PortNumberError: Port number is not correct.

CAN\_ActiveBoardError: This board is not activated.

CAN\_ConfigError: CAN controller enter operating mode failure.

---

#### 4.1.26 CAN\_OutputByte

- **Description:**

Write data to CAN chip (SJA1000).

- **Syntax:**

```
void CAN_OutputByte(BYTE BoardNo, BYTE Port, WORD wOffset,  
BYTE bValue)
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7)

Port: [input] CAN port number (1~4 or 1~2)

wOffset: [input] Address offset from base address

bValue: [input] Data byte

- **Return:**

None.

---

#### 4.1.27 CAN\_InputByte

- **Description:**

Read data from CAN chip (SJA1000).

- **Syntax:**

BYTE CAN\_InputByte(BYTE BoardNo, BYTE Port, WORD wOffset)

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7)

Port: [input] CAN port number (1~4 or 1~2)

wOffset: [input] Address offset from base address

- **Return:**

Data Byte of CAN chip.

---

#### 4.1.28 *CAN\_GetSystemFreq*

- **Description:**

Get the clock frequency. It is useful for calculate the time of the time stamp for reception message.

- **Syntax:**

LONGLONG *CAN\_GetSystemFreq*(void)

- **Parameter:**

None

- **Return:**

In Windows 98/Me/NT4 is clock frequency, in Windows 2000/XP is always 10000000.



---

#### 4.1.29 CAN\_InstallUserIsr (only for Windows 2000/XP)

- **Description:**

Using this function can allow users to apply ISR (interrupt service routine). When users put their ISR into this function, the interrupt of receiving CAN message will trigger the users' ISR.

- **Syntax:**

```
int CAN_InstallUserIsr(BYTE BoardNo,  
                      void(*UserISR)(BYTE BoardNo))
```

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7)

(\*UserISR)(BYTE BoardNo): [input] The pointer which points a function with format "void XXX(BYTE BoardNo)". The XXX is the function name of users' ISR. The parameter, BoardNo, indicates the number of the board which produces an interrupt signal.

- **Return:**

CAN\_NoError: OK

CAN\_DriverError: Kernel driver can't be opened.

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current total board number.

CAN\_ActiveBoardError: This board is not activated.

CAN\_InstallIrqFailure: Enable or start IRQ failure.

CAN\_InstallIsrError: Enable or start ISR failure.

---

#### 4.1.30 *CAN\_RemoveUserIsr (only for Windows 2000/XP)*

- **Description:**

When users don't need the ISR function, call this function to remove users ISR.

- **Syntax:**

Int CAN\_RemoveUserIsr(BYTE BoardNo)

- **Parameter:**

BoardNo: [input] PISO-CAN board number (0~7)

- **Return:**

CAN\_NoError: OK

CAN\_DriverError: Kernel driver can't be opened.

CAN\_BoardNumberError: BoardNo is not correct or exceeds the current  
total board number.

CAN\_ActiveBoardError: This board is not activated.

CAN\_RemoveIrqFailure: Disable or stop IRQ failure.

---

## 4.2 Flow Diagram for Application

In this section, we will show the operation procedure of PISO-CAN/PEX-CAN /PCM-CAN board for sending and receiving CAN message. Figure 4.1 presents the “Send CAN Message” procedure. Figure 4.2 and 4.3 stand for the “receiving CAN Message” in polling and in interrupt mode, respectively. Users need to follow the operation principle of PISO-CAN/PEX-CAN/PCM-CAN board for correctly and easily send and receive the CAN message through CAN network. For more detail information, please refer to the demo programs in section 5.

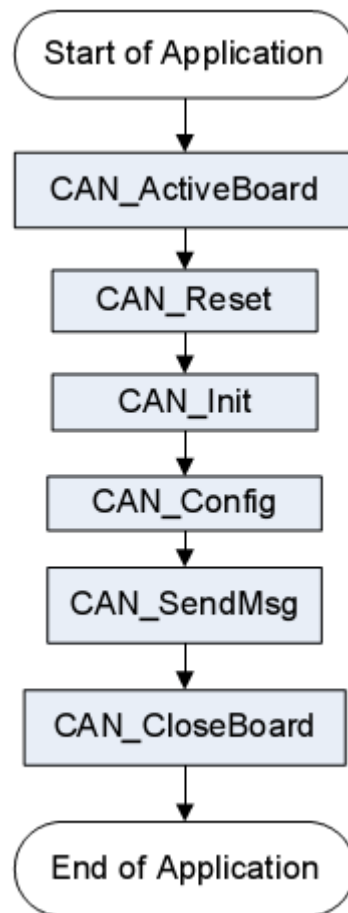


Figure 4.1 Flow Chart of “Send CAN Message”

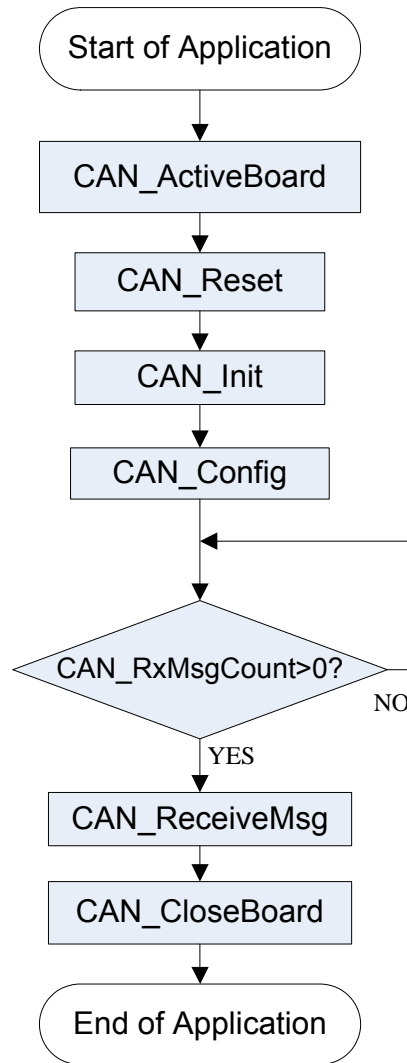


Figure 4.2 Flow Chart of “Receive CAN Message”

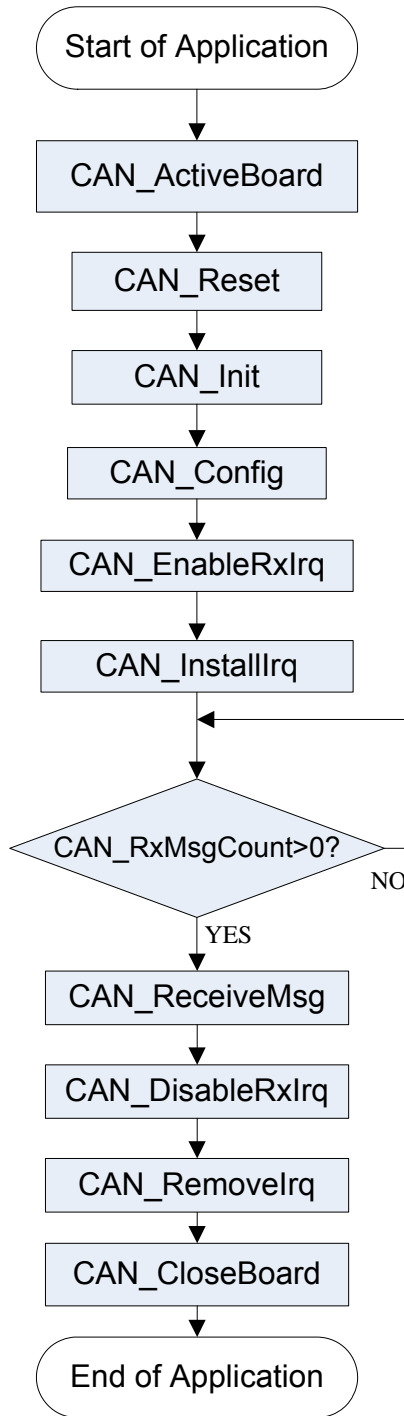


Figure 4.3 Flow Chart of “Receive CAN Message with IRQ”

---

## 5 Demo Programs for Windows

All of demo programs will not work normally if DLL driver would not be installed correctly. During the installation process of DLL driver, the install-shields will register the correct kernel driver to the operation system and copy the DLL driver and demo programs to the correct position based on the driver software package you have selected (windows 2K/XP/7). After driver installation, the related demo programs and development library and declaration header files for different development environments are presented as follows.

--\Demo	→ Demo program
--\BCB3	→ for Borland C++ Builder 3
--\CAN.H	→ Header file
--\PISOCAN.LIB	→ Linkage library for BCB
--\Delphi4	→ for Delphi 4
--\CAN.PAS	→ Declaration file
--\VC6	→ for Visual C++ 6
--\CAN.H	→ Header file
--\PISOCAN.LIB	→ Linkage library for VC6
--\VB6	→ for Visual Basic 6
--\CAN.BAS	→ Declaration file
--\C#,Net	→ for C#.Net
--\ PISOCAN_Net.DLL	→ Dll file
--\VB,Net	→ for VB.Net
--\ PISOCAN_Net.DLL	→ Dll file

### The list of demo programs:

TxRxCAN\_NoIRQ: Transmit and receive CAN messages.  
TxRxCAN\_IRQ: Transmit and receive CAN messages with IRQ

---

## A brief introduction of the demo programs

### TxRxCAN NoIRQ:

Demo1 is the example used for starting the PISO-CAN/PEX-CAN /PCM-CAN board. This demo program is designed to send out the CAN message through Port 1 and receive the CAN message immediately at port 2 in the same PISO-CAN/PEX-CAN/PCM-CAN board. Before exercising this demo, the user needs to finish the CAN median wiring connection between port 1 and port 2. Based on this demo, the user can key in the CAN message into the port 1 frame area and then click the “Send” button in order to send out the CAN message to port 2. If you click the “Receive” button in the CAN port 2 frame area, the CAN message received by CAN port 2 will be presented in “TEXT” box. This is shown in the below screenshot. Note that if port 2 displays a warning message like CAN Data Overrun, then it is an indication that the un-read messages within the 64 bytes RXFIFO CAN buffer have been covered by another message. This means that the messages that are being received from the CAN bus may be in error and/or they may be missing part of the message. Then the user can click on the “Clear Overrun” button to clear the RXFIFO buffer overrun status within the CAN controller.

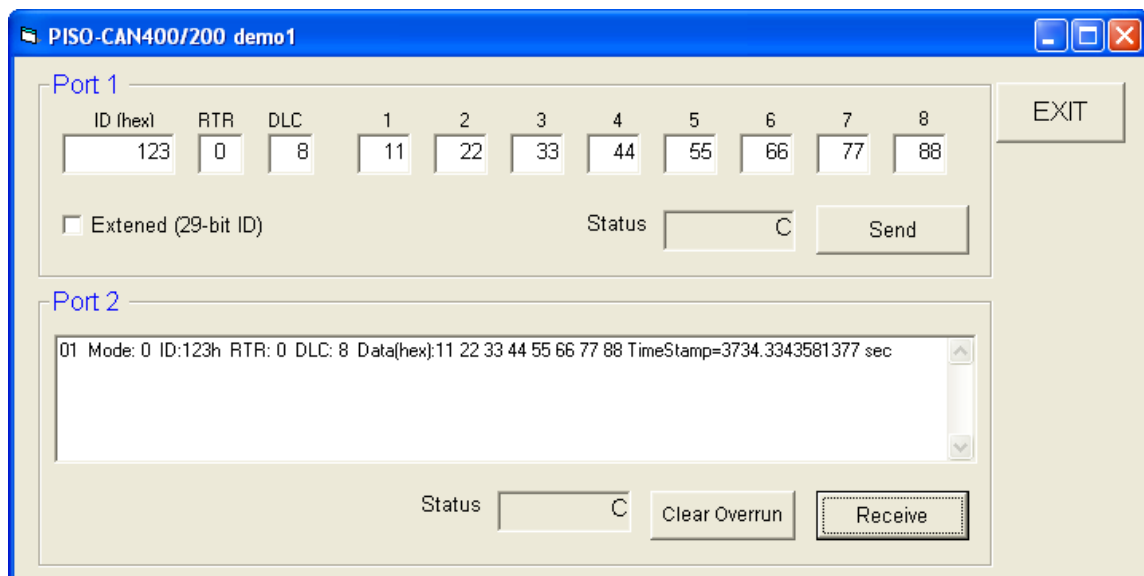


Figure 5.1: The form of demo1 program

## TxRxCAN IRQ:

In demo 2, we provide a demonstration on how to send out a CAN message through port 1 and receive the CAN message in port 2 by means of the interrupt mode. Contained within this operation, the user can key in the CAN message into the port 1 frame area and click on the “Send” button to send out the CAN message. At the same time, the CAN message will be received at port 2 by means of the interrupt mode. As shown in the following figure, port 2 can automatically receive the CAN message and store it within the 4K bytes of buffer software. When the user clicks the “Receive” button, all the messages stored in the 4K bytes buffer will all be presented in the TEXT edit area, as shown in the following figure.

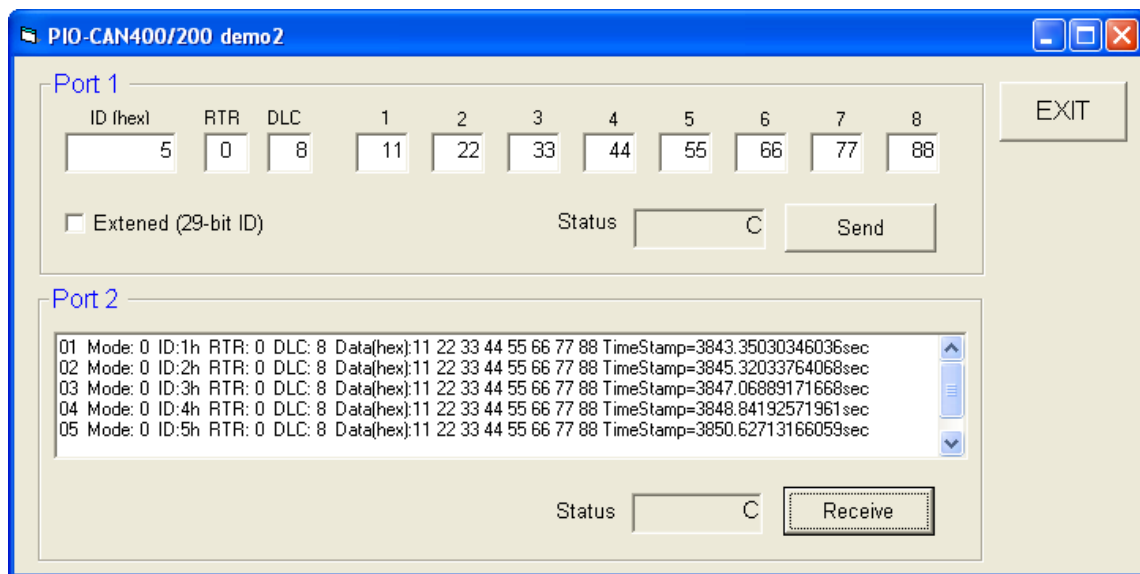


Figure 5.2: The form of demo2 program



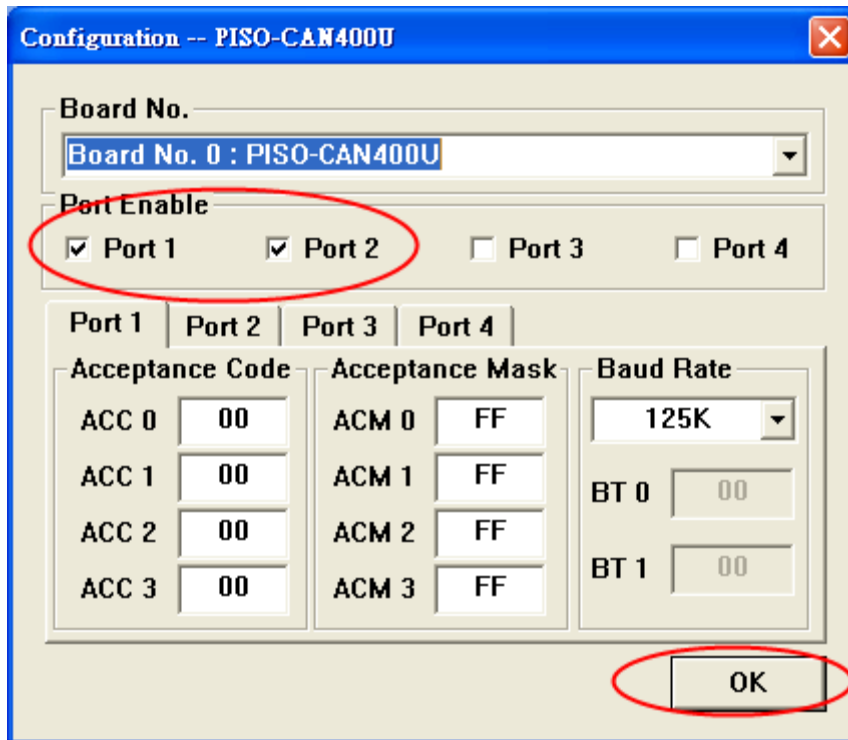
## 6 CANUtility Program for Windows

For PISO-CAN, PEX-CAN, or PCM-CAN, we provide a friendly CAN bus utility tool to allow users to send/receive the CAN messages to/from CAN network easily. This utility tool can be thought as a useful tool for monitoring CAN messages or testing CAN devices on the CAN network. It supplies several functions, such as sending CAN messages, receiving CAN messages, storing CAN messages, cyclic transmission, and so forth. The operation principle will be addressed in the following sub-section.

### (1) CAN Configure Dialog

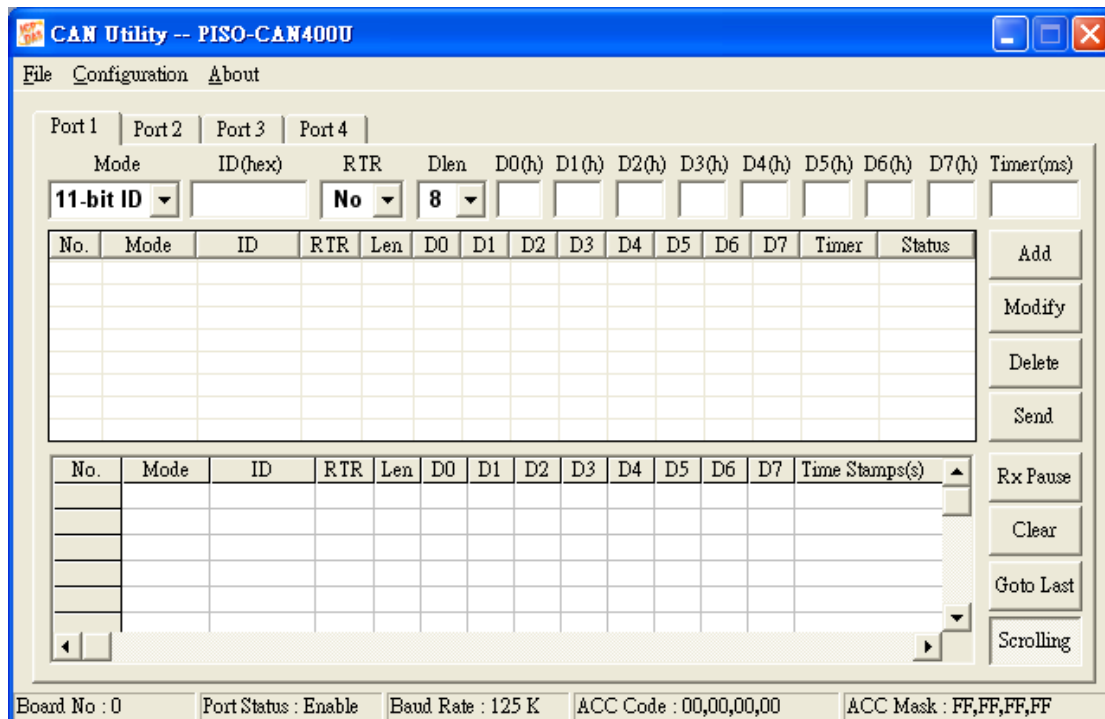
Please click the Board No combo box to select which CAN board plugged on the pc will be used.

Check the Port Enable check boxes to enable CAN ports. Then select CAN port tag. According to CAN communication requirement, users need to set the proper baud rate, acceptance code and acceptance mask. The Baud Rate combo box has eight kinds of baud, 10K, 20K, 50K, 125K, 250K, 500K, 800K, and 1M. Users can also define the special baud by using BT0 and BT1 field. If users select the user defined baud rate, users must have the background of the CAN chip, SJA1000. Afterwards, click OK to save the configuration.



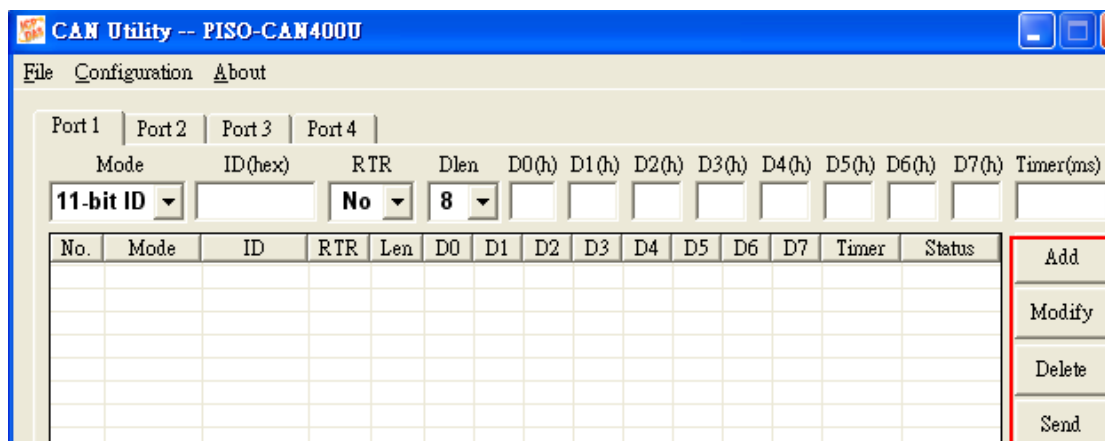
## (2) Main Dialog

The CAN Utility main dialog is as following figure. There are 1 tag, 2 tags and 4 tags for one-port card (PCM-CAN100), two-port card (PISO-CAN200/200U, PEX-CAN200i, and PCM-CAN200), four-port card (PISO-CAN400/400U) and eight-port card (PISO-CAN800U) respectively. In the bottom of the main dialog, the status bar shows five parameters, board number, port status, baud rate, acceptance code, and acceptance mask for the selected port.



## (3) CAN Transmission Function

In the CAN port transmits part page as follow figure, there are four function buttons for transmission list to use.

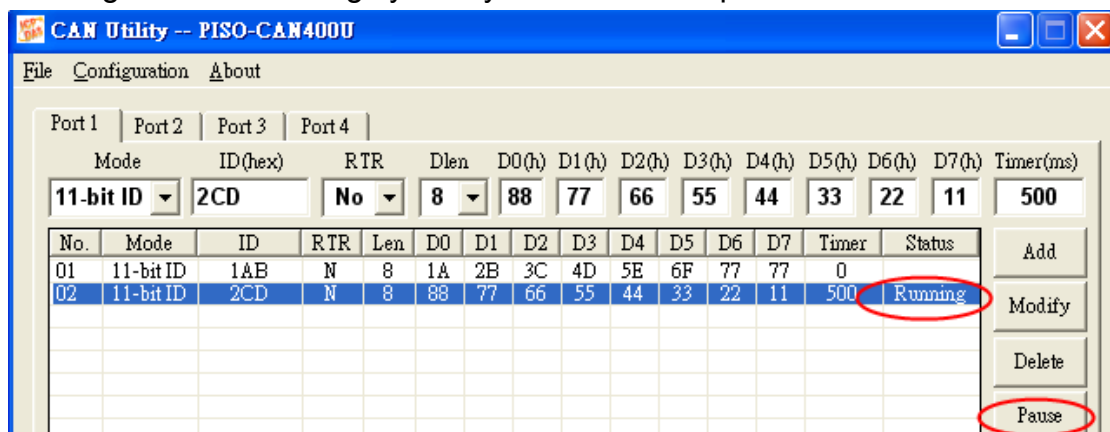


**Add Button:** User can key in the CAN message into the text boxes above the transmission list. Then click add button to insert this CAN message into transmission list. The transmission list can include maximum 20 CAN messages. After adding the message into transmission list, users can send this message to CAN network by using Send button.

**Modify Button:** If users want to modify the content of some CAN message in the transmission list, select this CAN messages in the transmission list firstly. Then, this CAN message information will be shown in the text boxes above the transmission list. Users can modify the CAN message in these text boxes directly. Finally, click Modify button to save the modification in the transmission list.

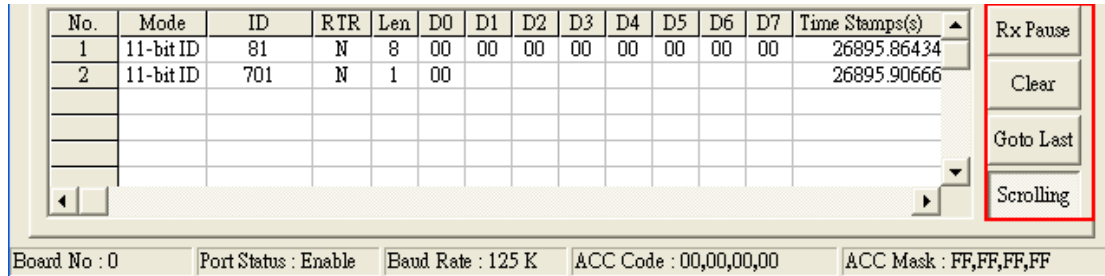
**Delete Button:** If some CAN message in the transmission list is useless, users can select it and click delete button to delete this CAN message from transmission list.

**Send Button:** After users select one CAN message from transmission list, click Send button to send this CAN message once from the selected CAN port. If the timer parameter of this CAN message is not 0, the CAN message will be send depending on this timer parameter periodically. In this case, the status filed of this CAN message in transmission list will display “Running”, and the text shown on the Send button will be changed to “Pause”. If uses want to stop the message transmission, click this button again. There are only 5 CAN message can be sending cyclically from one CAN port at the same time.



#### (4) CAN Reception Function

The following figure shows the receive part of a selected CAN port. There are four functions for reception list.



**Rx Pause:** Click this button to stop the CAN message reception from specific CAN port. Click it again to continue the message reception.

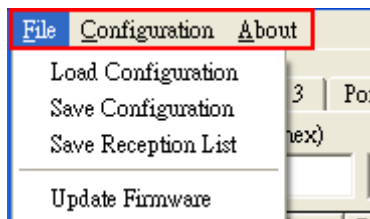
**Clear Button:** Click this button to delete all CAN messages shown in the reception list.

**Goto Last:** Click this button to show the last received CAN message.

**Scrolling Button:** When the button is pushed down, the reception list is always scrolled automatically to the last received CAN message. If this button is pushed up, the reception list will stop to scroll automatically, but reception list still get the CAN messages from CAN port. The default status of this button is pushed down.

#### (5) Menu Function

There are three functions on the CAN Utility menu.



##### File Menu:

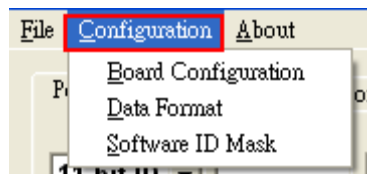
- **Load Configuration:** If users had have saved the configuration by using CANUtility before, users can click Load Configuration function to load the older records into these lists of CAN Utility.
- **Save Configuration:** The function is used for saving the transmission list, data format list, and ID mask list of each CAN

---

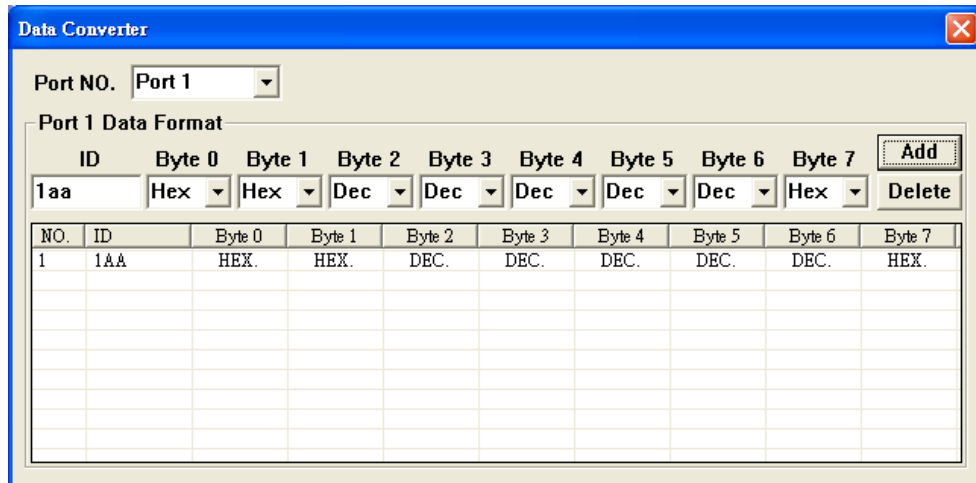
port to a .txt file.

- **Save Reception List:** Save Receive List, is used for saving the CAN messages that is received on the reception list. The data in the reception list of each different CAN port will be saved into different .txt file except that the reception list has no message. For example, if users want to save the data in the reception list to “test.txt” file, generally, these data will be saved to four .txt files, text\_port01.txt, text\_port02.txt, text\_port03.txt, and text\_port04.txt when users using PISO-CAN400. If the reception list of the port 2 has no data, the text\_port02.txt file will not be produced.
- **Update Firmare:** Update firmware of the CAN board. This function can not be used to the PISO-CAN, PEX-CAN, and PCM-CAN, it only for PISO-CM100/U, PISO-CPM100/U, and PISO-DNM100/U.

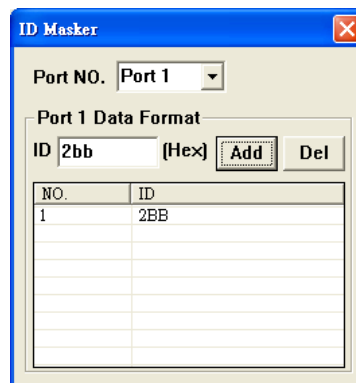
### Configuration Menu:



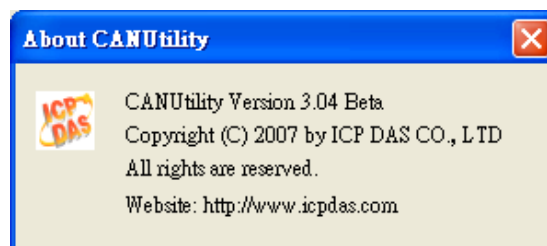
- **Board Configuration:** Users can click “Board Configuration” to re-configure the CAN board. Please refer to “(1) CAN Configure Dialog” of this section for more detail information.
- **Data Format:** In this function, user can set what kind of format (such as hexadecimal, decimal, or ASCII) the CAN message with specific ID will be displayed on the reception list. The setting dialog is as follows. For example, set the data format of the byte2 ~ byte6 of the CAN message with ID 0x1AA to the decimal format. Then, the reception list will display the byte2 ~ byte 6 data of the message with ID 0x1AA by using decimal format, and display the other bytes of this message by using hexadecimal format. Any message without configuring data format will be shown by using hexadecimal format. Users can configure maximum 20 different ID messages in this dialog.



- Software ID Mask:** If users don't want to show some message with specific ID on the reception list, the ID mask function is useful for that. As following figure, users can set maximum 20 different ID message in the ID mask list. Afterwards, if the CAN port receives the message with the ID set in ID mask list, the CAN message will not shown in the reception list.



**About:** Show the information about the CAN Utility version and the ICP DAS home page.



## 7 Appendix

### 7.1 Acceptance Filtering

Four 8-bits Acceptance Code registers (AC0, AC1, AC2 and AC3) and Acceptance Mask registers (AM0, AM1, AM2 and AM3) are available for a various filtering of messages. These registers can be used for controlling a 4-byte filter, which can check the specific bits of a CAN message and decide if this message will be passed to the CAN card or not. The message filter general concept is shown in Figure A.1. The Acceptance Code Register is mainly used for deciding what kind of message ID the CAN card will accept. The Acceptance Mask Register is mainly used for deciding which bit of message ID will need to check by using the Acceptance Code Register. If the bit of the Acceptance Mask is set to 0, it means that the bit in the same position of message ID needs to be checked.

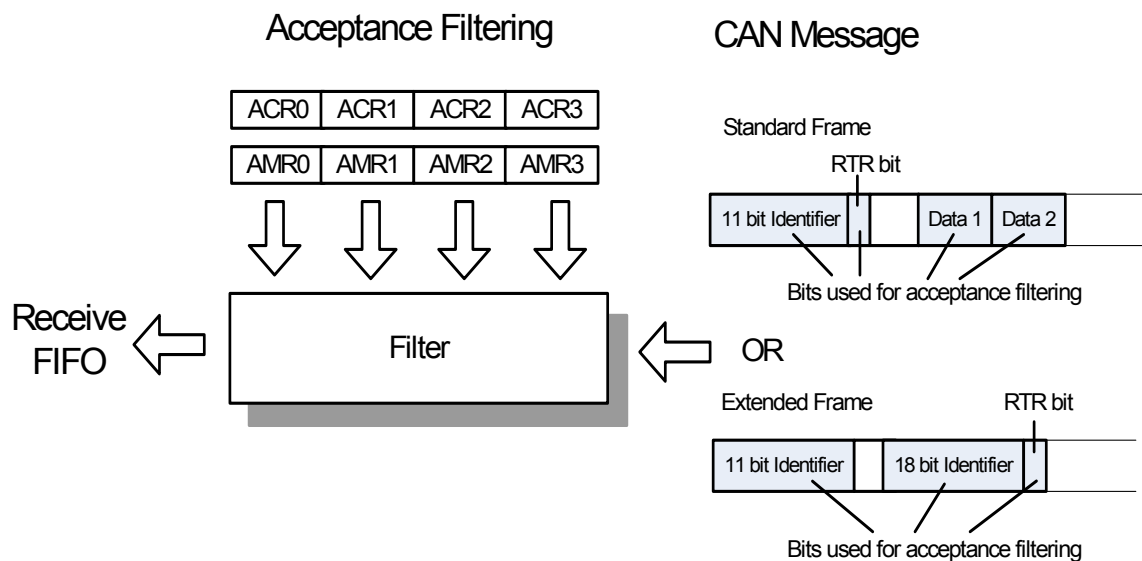


Figure A.1 Acceptance Filter

---

### Example 1:

Assume that a message with a **Standard Frame** is considered. The Acceptance Code Registers (ACRn) and Acceptance Mask Registers (AMRn) is set as follows.

n	0	1 (upper 4 bits)	2	3
ACRn	01xx x010	xxxx	xxxx xxxx	xxxx xxxx
AMRn	0011 1000	1111	1111 1111	1111 1111
Accepted messages (ID.28..ID.18 RTR)	01xx x010 xxxx			

("x"=don't care, only the upper 4 bits of ACR1 and AMR1 are used)

In this case, the ACR0 and the AMR0 are used for the upper 8 bits of message ID. The upper 4 bits of the ACR1 and AMR1 are used for the lower 3 bits of the message ID and RTR bit. The lower 4 bits of the ACR1 and AMR1 are useless. The ACR2 and AMR2 are used for the first data byte of the CAN message. The ACR3 and AMR3 are used for the second data byte of the CAN message. Therefore, no matter the CAN message is remote transmit request message or not, the message ID with the format 01xx x010 xxx will be accepted. (x means "don't care").



---

## Example 2:

Assume that a message with an **Extended Frame** is considered. The Acceptance Code Registers (ACRn) and Acceptance Mask Registers (AMRn) is set as follows.

n	0	1	2	3(upper 6 bits)
ACRn	1011 0100	1011 000x	1100 xxxx	0011 0xxx
AMRn	0000 0000	0000 0001	0000 1111	0000 0111
Accepted messages (ID.28..ID.0 RTR)	1011 0100	1011 000x	1100 xxxx	0011 0x

("x"=don't care, only the upper 6 bits of ACR3 and AMR3 are used)

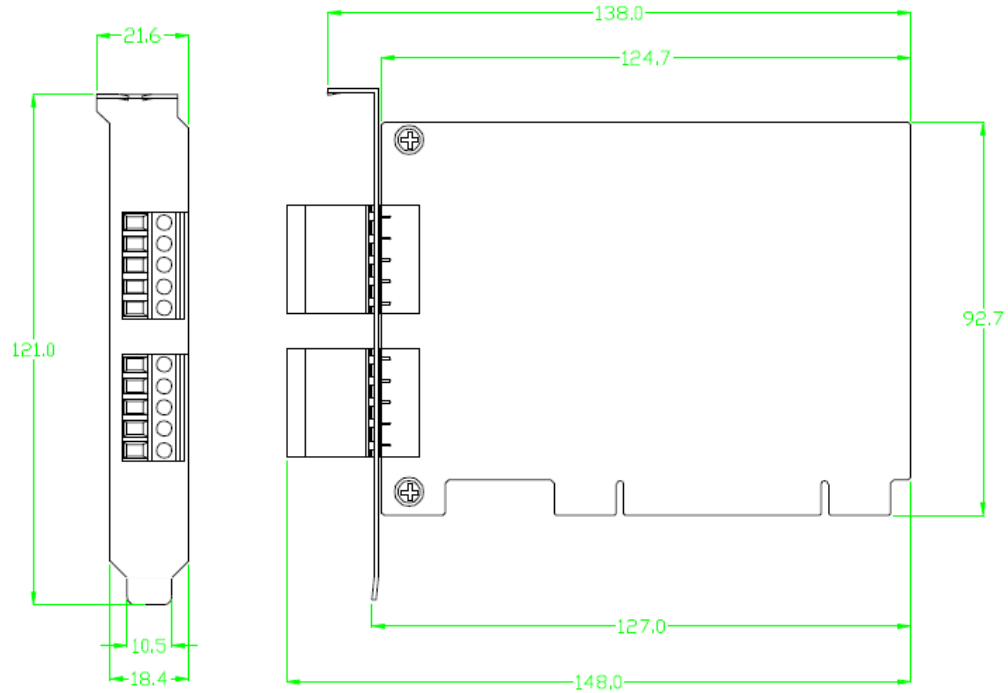
In this case, the lower 2 bits of AMR3 and AMR3 are useless. All the other bits of Acceptance Code and Acceptance Mask will be used for the 29-bit message ID and the RTR bit. Therefore, no matter the CAN message is RTR (remote transmit request) message or not, the message ID follows the format 1011 0100 1011 000x 1100 xxxx 0011 0x (x means "don't care") will be accepted.

---

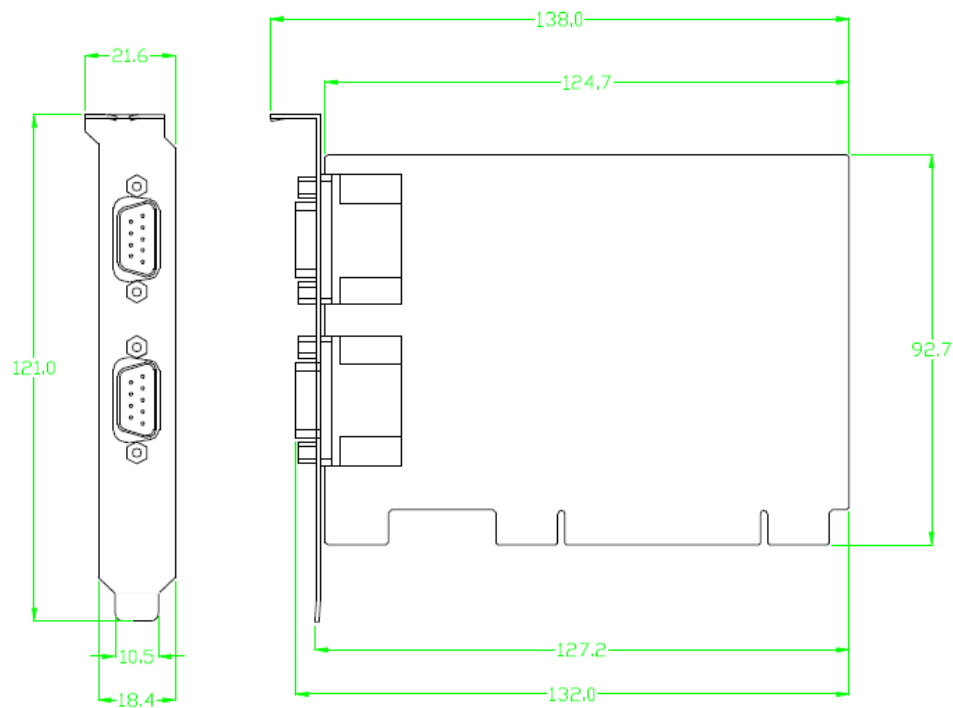
## 8 Dimensions

### 8.1 PISO-CAN200/400

#### PISO-CAN200/400-D/T



#### PISO-CAN200/400-T

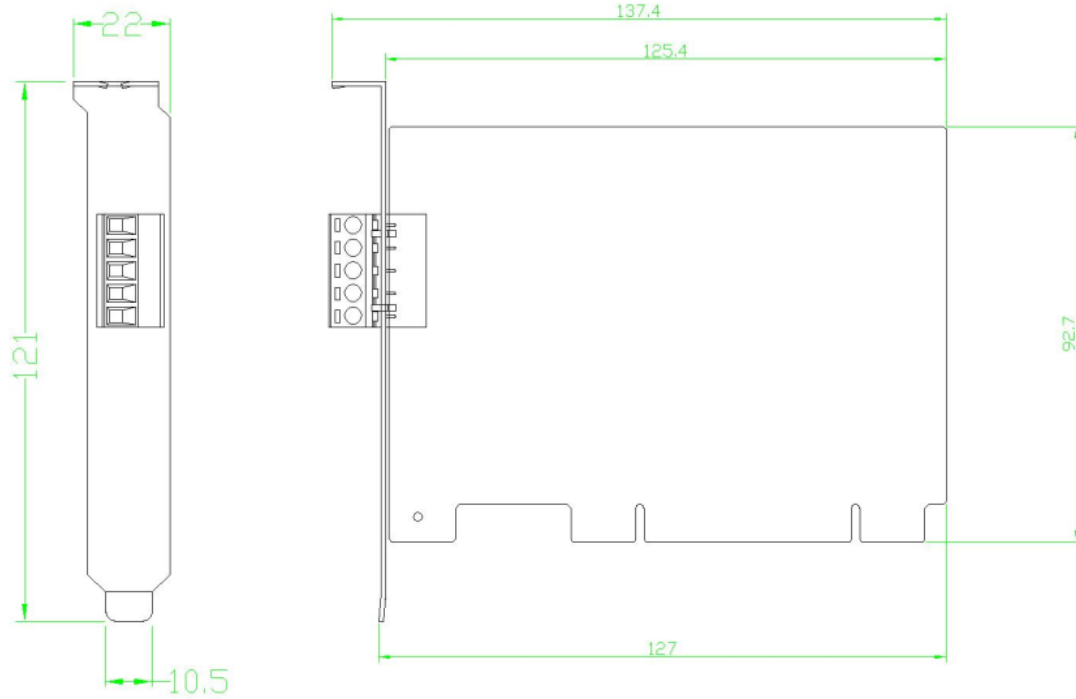


#### PISO-CAN200/400-D

## 8.2 PISO-CAN100U/200U/400U/800U

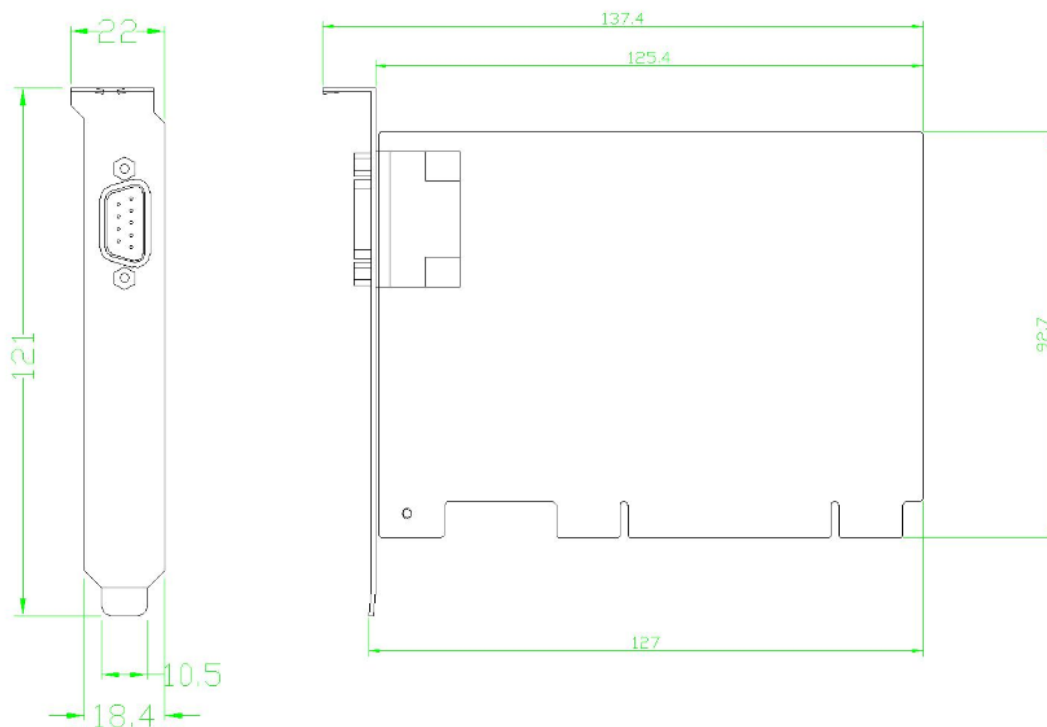
### PISO-CAN100U-D/T

Unit :mm



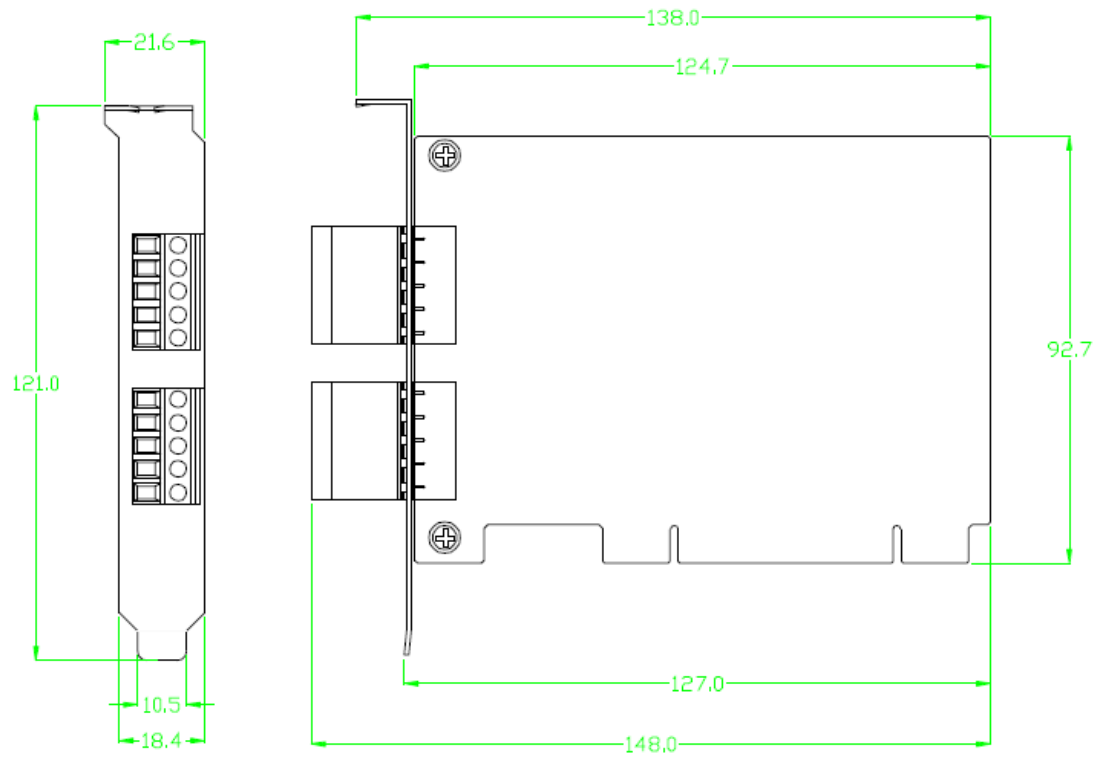
### PISO-CAN100U-T

Unit :mm

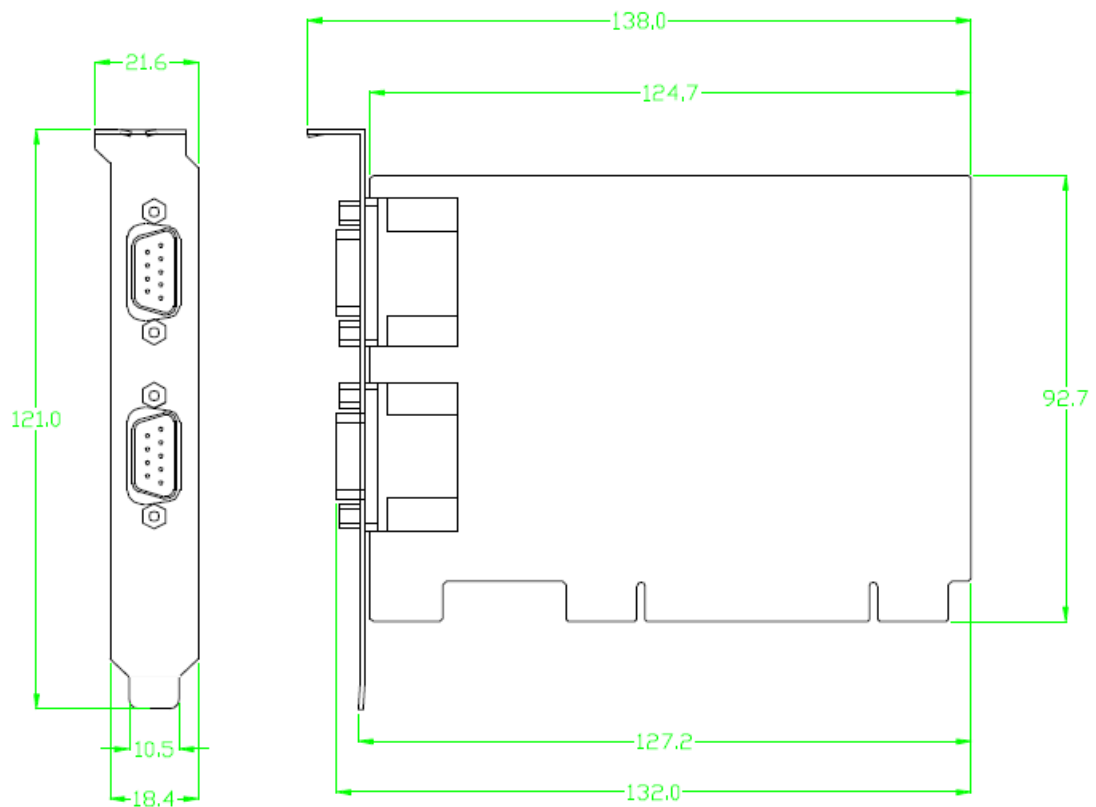


### PISO-CAN100U-D

## PISO-CAN200U/400U-D/T

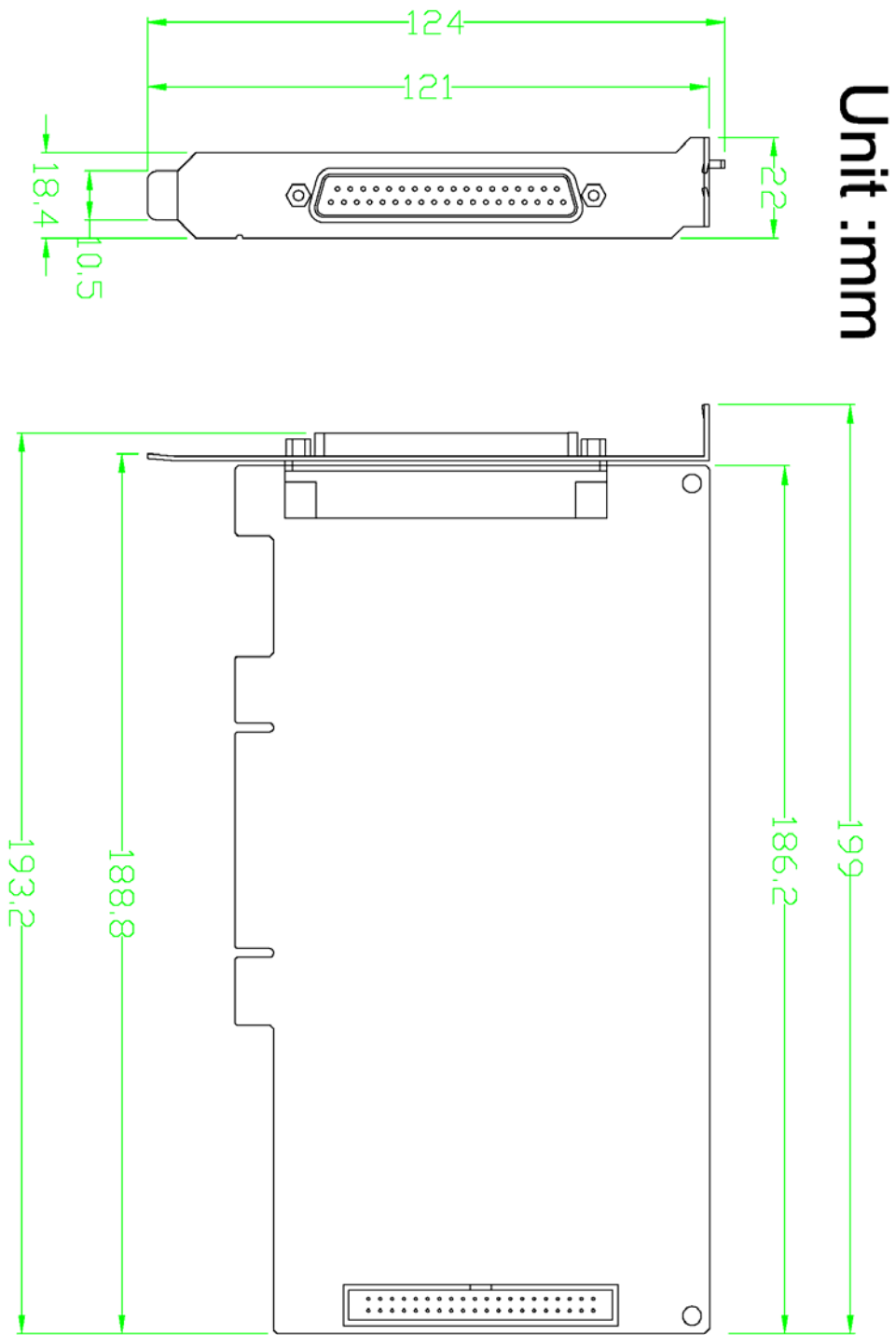


## PISO-CAN200U/400U-T



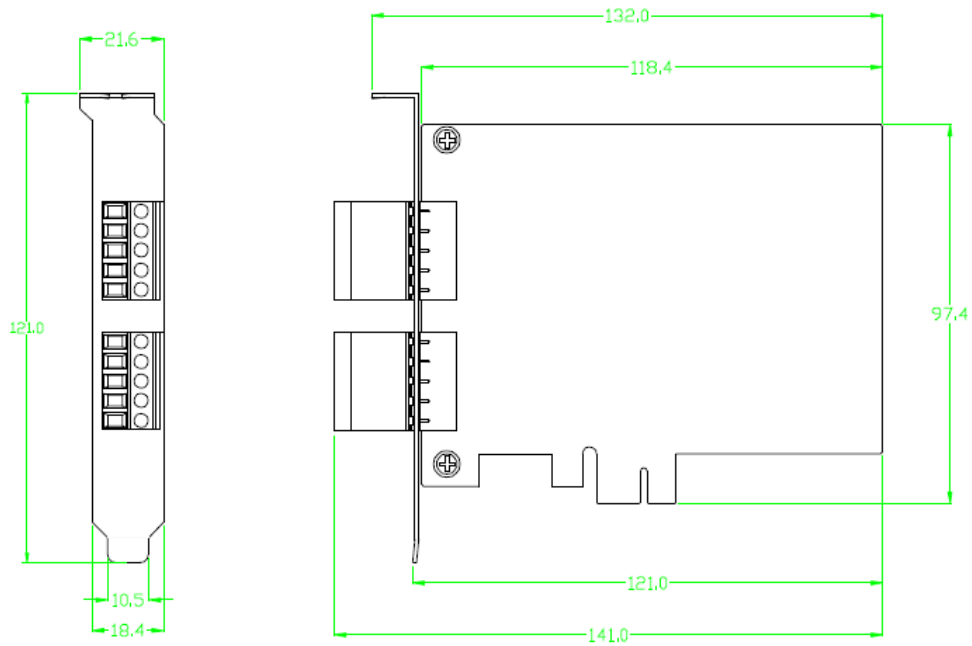
## PISO-CAN200U/400U-D

PISO-CAN800U

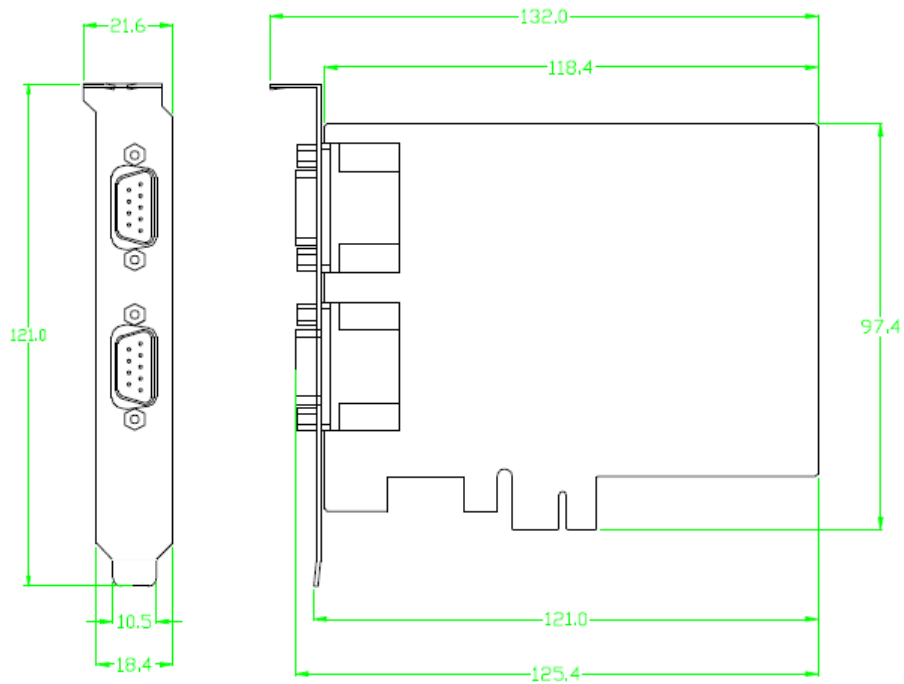


### 8.3 PEX-CAN200i

#### PEX-CAN200i-D/T



#### PEX-CAN200i-T

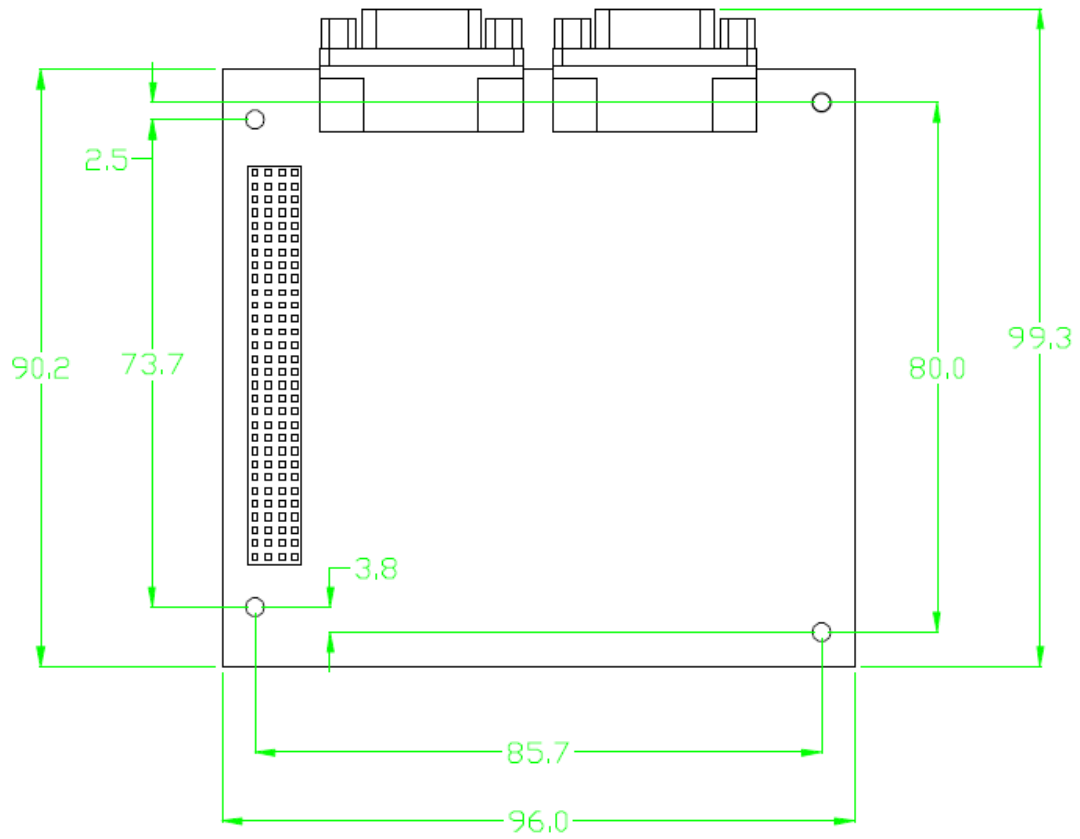


#### PEX-CAN200i-D

---

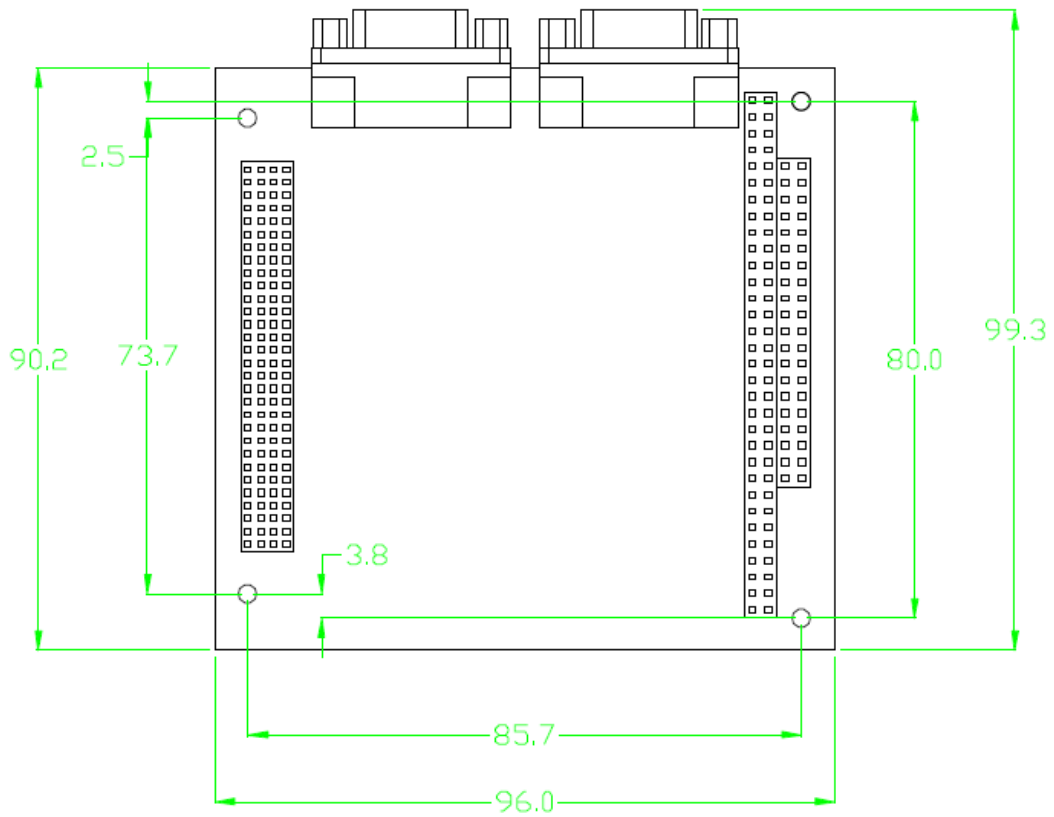
## 8.4 PCM-CAN100/200/200P

### PCM-CAN100/200-D



**Unit :mm**

# PCM-CAN200P-D



Unit :mm