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

使用者手册

产品保固

凡泓格科技股份有限公司产品从购买即日起，若无任何材料性缺损保固一年。

免责声明

凡使用本系列产品除质量量所造成的损害，泓格科技股份有限公司不承担任何法律责任。泓格科技股份有限公司有义务提供本系列产品可靠而详尽的数据，但保留修改权利，且不承担使用者非法利用数据对第三方所造成侵害构成的法律责任。

版权

商标

手册中所涉及所有公司商标，商标名称以及产品名称分别属于该商标或名称的拥有者所有。

目录

1 产品信息

1.1 简介

CAN (Controller Area Network；控制器局域网络) 是一种串行式通讯协议，特别适合使用在主系统或子系统下提供更完整的智能网络设备如感应器及驱动器。它提供高安全等级及有效率的分布式实时控制，更具备了侦错和优先权判别的机制。PISO-CAN、PEX-CAN与PCM-CAN使用独立的CAN 控制器，每个CAN 卡可以有两个或四个独立的CAN总线通讯端口，端子部分使用 5-pin的螺钉接线端子或 9-pin D-sub公头接线端子。它可以是主/从端界面，并且应用在不同的CAN应用。另外，这些CAN卡使用NXP SJA1000T CAN控制器与 82C250/251或TJA1042 收发器，提供总线仲裁与错误侦测。这几种CAN卡的差异在于计算机的插槽接口，一些是PCI接口、一些是PCI Express接口及一些是PCI-104 接口，要得知CAN卡详细的特色与规格，请参考 1.2 节与 1.3 节。

1.2 特色

- PCI 总线接口
- 2500Vrms 光隔离保护
- 1/2/4/8 个独立的 CAN 通讯端口
- 相容 CAN 2.0A 与 CAN 2.0B 的规范
- CAN bus 可程序传输速率高达 1 Mbps
- 可跳线选择 120Ω(奥姆)终端电阻
- 直接由 CAN 控制器的内存存取数据
- PISO-CAN200/400
  - 33MHz 32bit 5V 随插即用 PCI v2.1 总线
  - PCI 卡接口支持 5V PCI bus
  - 3 kV_{DC} 隔离保护
  - 具有 2/4 个独立的 CAN 通道
- PISO-CAN100U200U400U/800U
  - 符合 PCI v2.2 32-bit 33MHz
  - Universal PCI 卡、支持 5V 与 3.3V PCI 总线
  - 3 kV_{DC} 隔离保护
  - 具有 1/2/4/8 个独立的 CAN 通道
- PEX-CAN200i
  - 32-bit, 33MHz, X1 PCI Express 总线
  - PCI Express R1.0 规格
  - 3 kV_{DC} 隔离保护
  - 具有 2 个独立 CAN 通道
- PCM-CAN100/200/200P
  - 符合 PCI104 规格
  - 9-pin D-sub 公连接端子
  - 1 kV_{DC} 隔离保护
  - 具有 1/2 个独立 CAN 通道
- 驱动程序支持 Windows 2000/X/P/7 和 Linux 2.6.x ~ 3.2.20 环境

1.3 硬件规格

1.3.1 PCM-CAN100/200/200P

模块名称	PCM-CAN100-D	PCM-CAN200-D	PCM-CAN200P-D
总线接口			
种类	PCI-104		PC/104-Pluse
CAN 界面			
控制器	NXP SJA1000T 搭配 16MHz 震荡器		
收发器	NXP 82C250		
通道数	1	2	
接头	9 针公/母座 D-Sub	9 针公座 D-Sub	
	(CAN_L, CAN_SHLD, CAN_H, 其余脚位空接)		
鲍率(bps)	10 k, 20 k, 50 k, 125 k, 250 k, 500 k, 800 k, 1 M (允许使用者自订义鲍率)		
终端电阻	跳线设定 120 Ω 终端电阻		
电源			
功耗	250 mA @ 5 V		
机构			
尺寸	91mm x 22mm x 96mm (宽 x 长 x 高)		
环境			
工作温度	0 ~ 60 ℃		
储存温度	-20 ~ 70 ℃		
湿度	相对湿度 5 ~ 85% RH, 无结露		

1.3.2 PEX-CAN200i

模块名称	PEX-CAN200i-D	PEX-CAN200i-T
总线接口		
类型	PCIe x 1	
CAN 界面		
控制器	NXP SJA1000T 搭配 16MHz 震荡器	
收发器	NXP 82C250	
通道数	2	
接头	9 针公座 D-Sub	5 针螺丝端子
	(CAN_L, CAN_SHLD, CAN_H, 其余脚位空接)	
鲍率(bps)	10 k, 20 k, 50 k, 125 k, 250 k, 500 k, 800 k, 1 M (允许使用者自订义鲍率)	
终端电阻	跳线设定 120 Ω 终端电阻	
电源		
功耗	100 mA @ 12 V, 100 mA @ 3.3 V	
机构		
尺寸	120mm x 22mm x 85mm (宽 x 长 x 高)	
环境		
工作温度	0 ~ 60 ℃	
储存温度	-20 ~ 70 ℃	
湿度	相对湿度 5 ~ 85% RH, 无结露	

1.3.3 PISO-CAN200/200U

模块名称	PISO-CAN200-D PISO-CAN200-T		PISO-CAN200U-D	PISO-CAN200U-T
总线接口				
类型	PCI 接口, 5 V 讯号, 33 MHz, 32 位, 随插即用		Universal PCI 接口, 支持 3.3 V 与 5 V 讯号, 33 MHz, 32 位, 随插即用	
CAN 界面				
控制器	NXP SJA1000T 搭配 16MHz 震荡器			
收发器	NXP 82C250			
通道数	2			
接头	9 针公座 D-Sub	5 针螺丝端子	9 针公座 D-Sub	5 针螺丝端子
	(CAN_L, CAN_SHLD, CAN_H, 其余脚位空接)			
鲍率(bps)	10 k, 20 k, 50 k, 125 k, 250 k, 500 k, 800 k, 1 M (允许使用者自订义鲍率)			
终端电阻	跳线设定 120 Ω 终端电阻			
电源				
功耗	250 mA @ 5 V			
机构				
尺寸	126mm x 22mm x 85mm (宽 x 长 x 高)			
环境				
工作温度	0 ~ 60 ℃			
储存温度	-20 ~ 70 ℃			
湿度	相对湿度 5 ~ 85% RH, 无结露			

1.3.4 PISO-CAN400/400U

模块名称	PISO-CAN400U-D	PISO-CAN400U-T	PISO-CAN400U-D	PISO-CAN400U-T
总线接口				
类型	PCI 接口, 5 V 讯号, 33 MHz, 32 位, 随插即用		Universal PCI 接口, 支持 3.3 V 与 5 V 讯号, 33 MHz, 32 位, 随插即用	
CAN 界面				
控制器	NXP SJA1000T 搭配 16MHz 震荡器			
收发器	NXP 82C250			
通道数	4			
接头	9 针公座 D-Sub	5 针螺丝端子	9 针公座 D-Sub	5 针螺丝端子
	(CAN_L, CAN_SHLD, CAN_H, 其余脚位空接)			
鲍率(bps)	10 k, 20 k, 50 k, 125 k, 250 k, 500 k, 800 k, 1 M (允许使用者自订义鲍率)			
终端电阻	跳线设定 120 Ω 终端电阻			
电源				
功耗	300 mA @ 5 V			
机构				
尺寸	126mm x 22mm x 85mm (宽 x 长 x 高)			
环境				
工作温度	0 ~ 60 ℃			
储存温度	-20 ~ 70 ℃			
湿度	相对湿度 5 ~ 85% RH, 无结露			

1.3.5 PISO-CAN100U

模块名称	PISO-CAN100U-D	PISO-CAN100U-T
总线接口		
类型	Universal PCI 接口，支持 3.3 V 与 5 V 讯号, 33 MHz, 32 位, 随插即用	
CAN 界面		
控制器	NXP SJA1000T 搭配 16MHz 震荡器	
收发器	NXP 82C250	
通道数	1	
接头	9 针公座 D-Sub	5 针螺丝端子
	(CAN_L, CAN_SHLD, CAN_H，其余脚位空接)	
鲍率(bps)	10 k, 20 k, 50 k, 125 k, 250 k, 500 k, 800 k, 1 M (允许使用者自订义鲍率)	
终端电阻	跳线设定 120 Ω 终端电阻	
电源		
功耗	225 mA @ 5 V	
机构		
尺寸	126mm x 22mm x 85mm (宽 x 长 x 高)	
环境		
工作温度	0 ~ 60 ℃	
储存温度	-20 ~ 70 ℃	
湿度	相对湿度 5 ~ 85% RH，无结露	

1.3.6 PISO-CAN800U

总线接口	
类型	Universal PCI 接口，支持 3.3 V 与 5 V 讯号, 33 MHz, 32 位，随插即用
CAN 界面	
控制器	NXP SJA1000T 搭配 16MHz 震荡器
收发器	NXP TJA1042
通道数	8
接头	37 针公座 D-Sub x 2
鲍率 (bps)	10 k, 20 k, 50 k, 125 k, 250 k, 500 k, 800 k, 1 M (允许使用者自订义鲍率)
终端电阻	跳线设定 120 Ω 终端电阻
电源	
功耗	800 mA @ 5 V
机构	
尺寸	193mm x 22mm x 93mm (宽 x 长 x 高)
环境	
工作温度	0 ~ 60 ℃
储存温度	-20 ~ 70 ℃
湿度	相对湿度 5 ~ 85% RH，无结露

1.4 产品检查清单

除了手册外，包装内含如下项目:

- 一块 PISO-CAN 或 PEX-CAN200 或 PCM-CAN 系列 CAN 卡
- ADP-9 板卡 (仅适用于 PISO-CAN400/PISO-CAN400U)
- 软件光盘

首先，建议使用者先阅读产品发表声明档案，档案中提供下列重要信息：

- 驱动程序、工具软件及范例程序路径。
- 如何安装驱动程序与工具软件
- 诊断程序路径
- 常见问题与解答

注意 !

假如这些任何配件有任何遗失或损害，请联系当地的代理商。请保留出货的相关配件和盒子，以方便您未来要寄送或存放产品.

2 硬件配置

这一节将说明CAN网络中，PISO-CAN、PEX-CAN及PCM-CAN系列的硬件设定、线路连接与终端电阻的架构。

2.1 板卡组件分布图

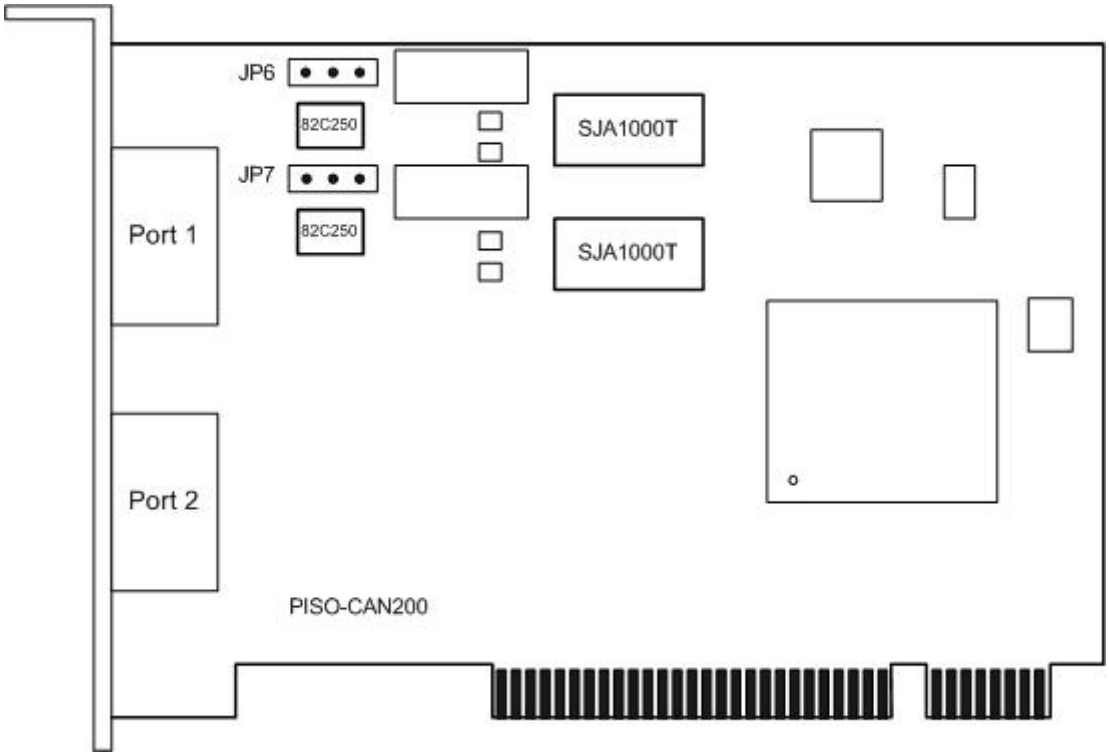


图 2.1　PISO-CAN200 板卡组件分布图

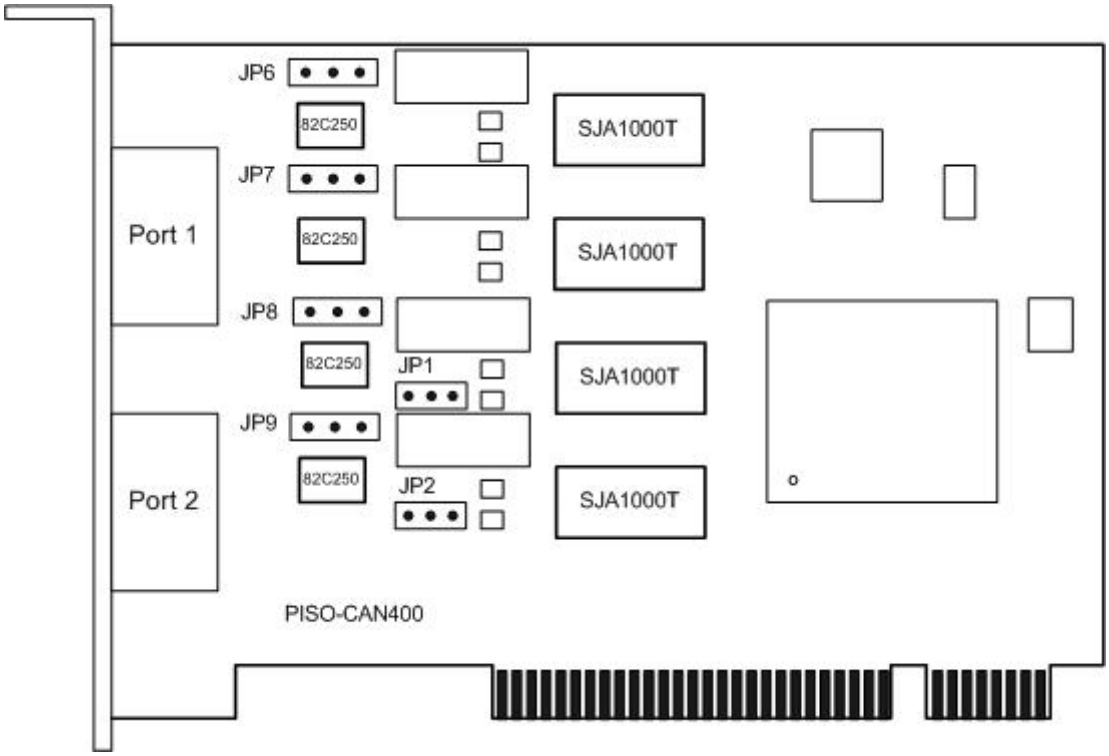


图 2.2　PISO-CAN400 板卡组件分布图

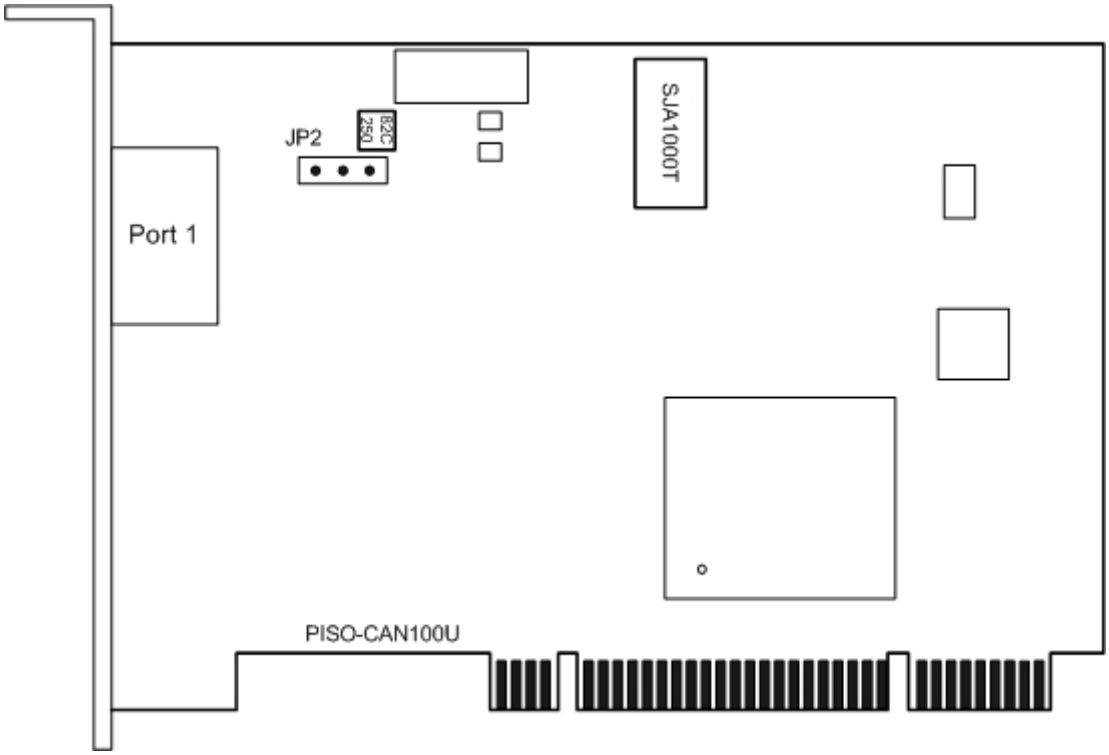


图 2.3　PISO-CAN100U 板卡组件分布图

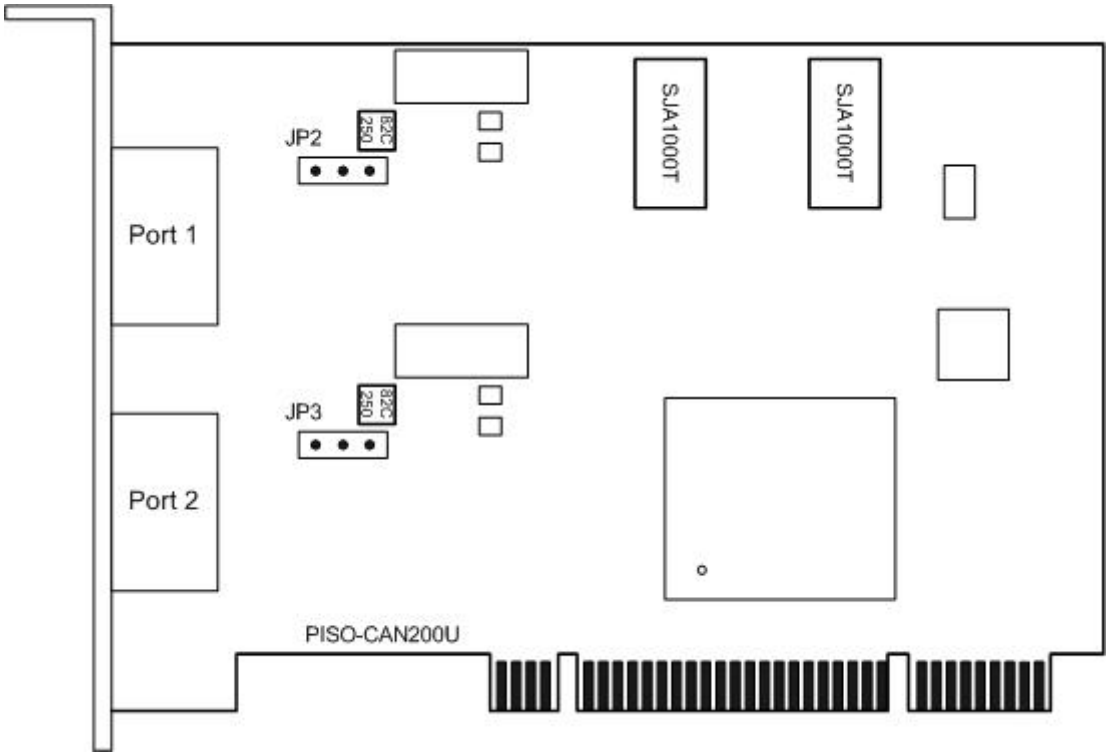


图 2.4　PISO-CAN200U 板卡组件分布图

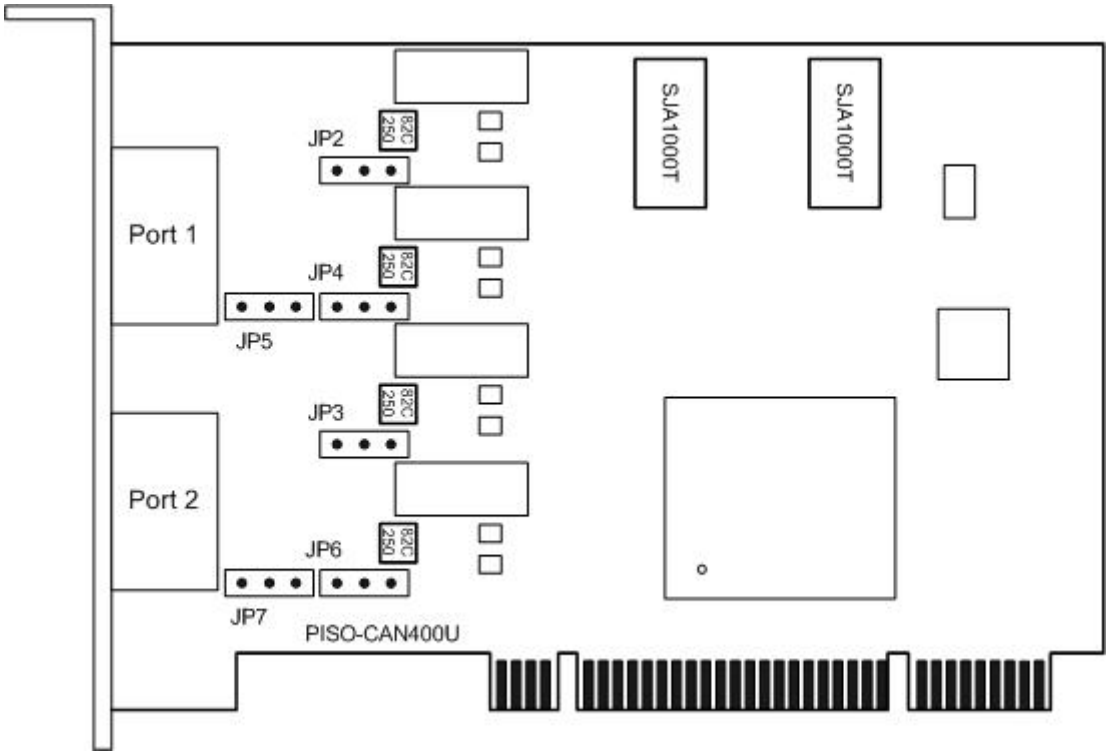


图 2.5　PISO-CAN400U 板卡组件分布图

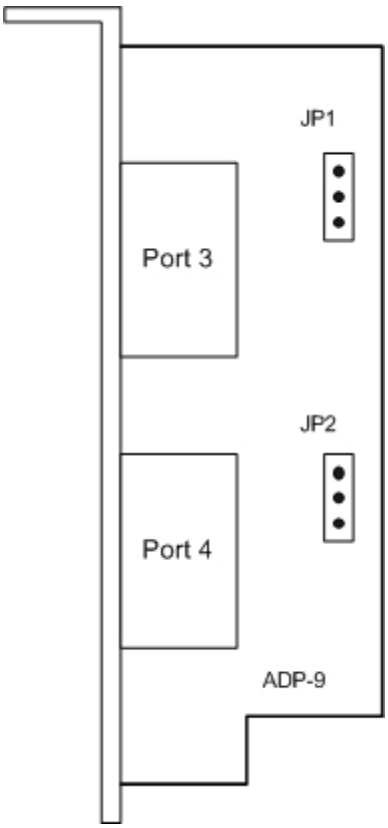


图 2.6　ADP-9 板卡组件分布图(适用于 PISO-CAN400/PISO-CAN400U)

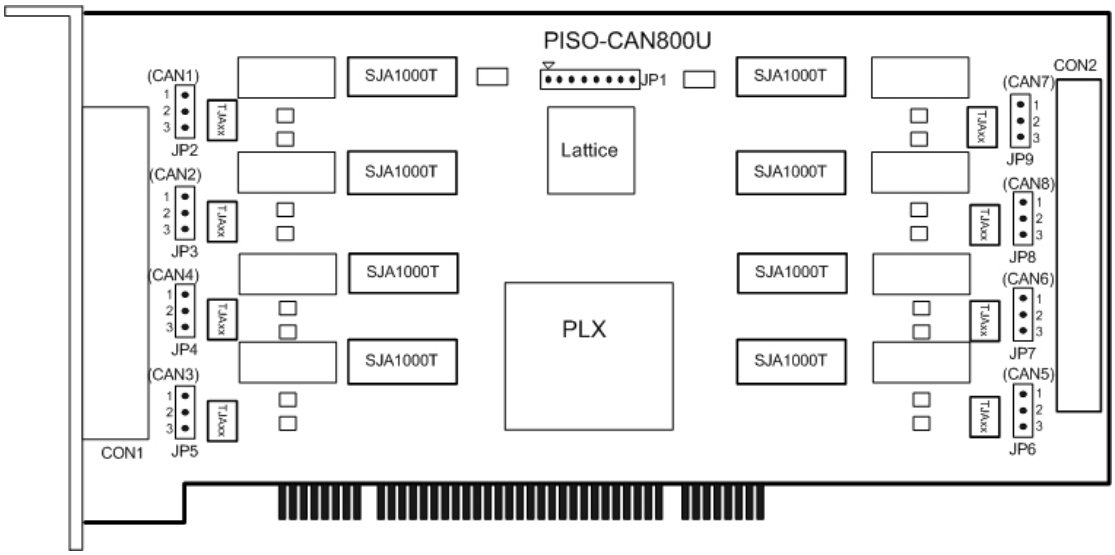


图 2.7　PISO-CAN800U 板卡组件分布图

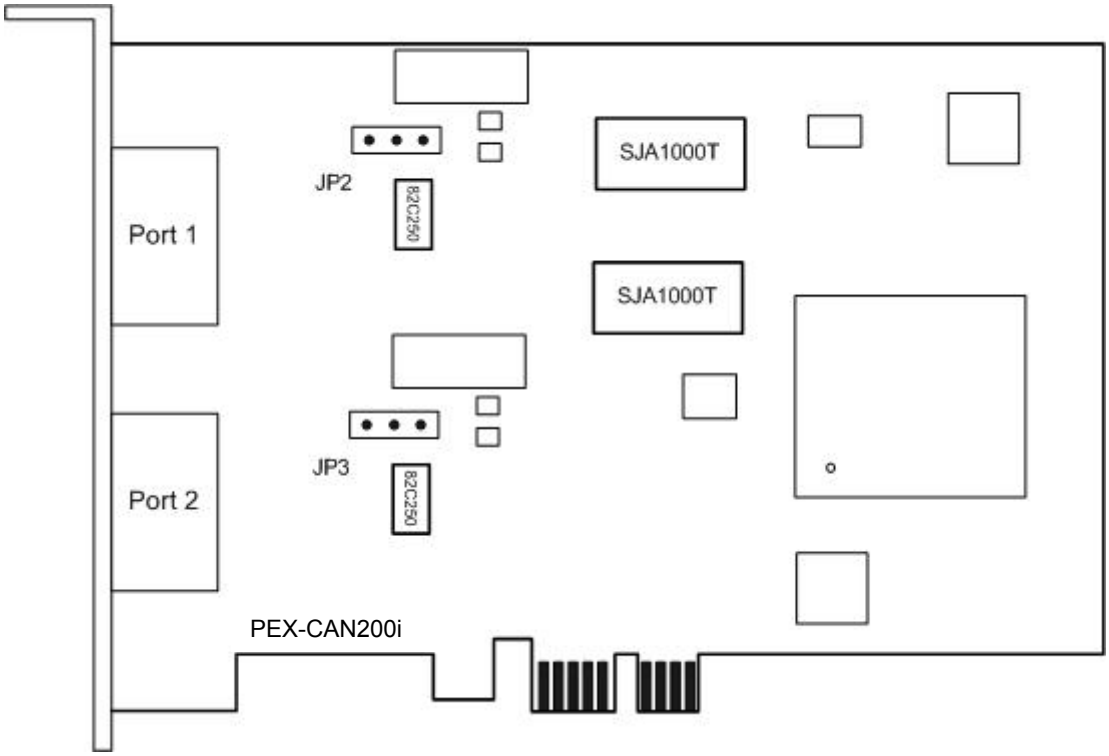


图 2.8　PEX-CAN200i 板卡组件分布图

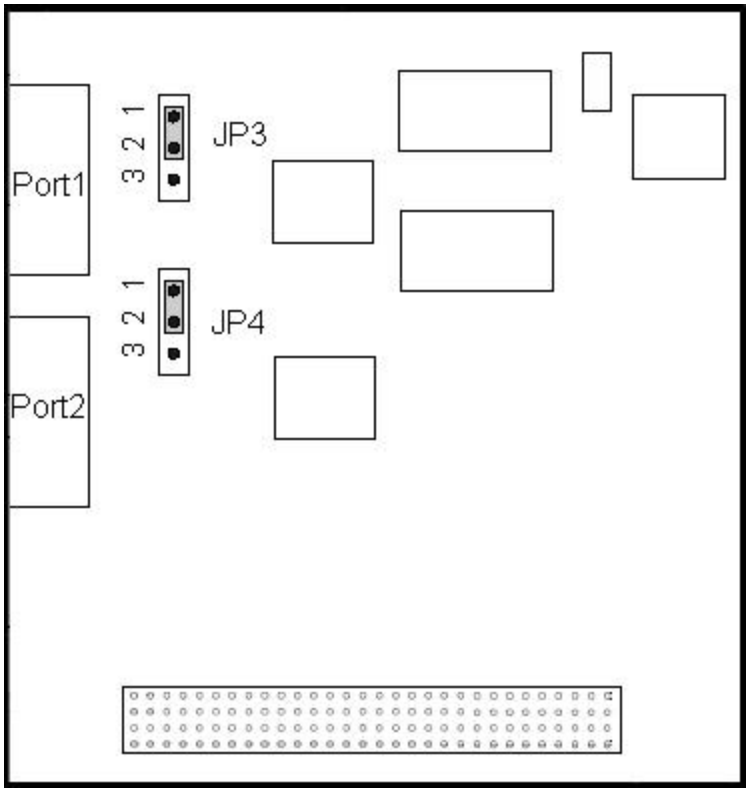


图 2.9 PCM-CAN200 板卡组件分布图

2.2 跳线选择

PISO-CAN200/400

表 2.1 跳线选择

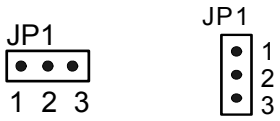
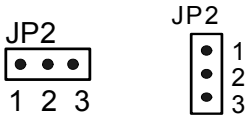
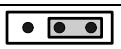
跳线	描述	状态	
JP1	CAN Port 3 端子连接 PISO-CAN400 板卡与 ADP-9 板卡。	 Pin1: CAN_L Pin2: CAN_H Pin3: Shield	
JP2	CAN Port 4 端子连接 PISO-CAN400 板卡与 ADP-9 板卡。	 Pin1: CAN_L Pin2: CAN_H Pin3: Shield	
JP6	Port 1 终端电阻(120Ω)选择	开启 	关闭 
JP7	Port 2 终端电阻(120Ω)选择 ()		
JP8	Port 3 终端电阻(120Ω)选择		
JP9	Port 4 终端电阻(120Ω)选择		

表 2.2 跳线选择

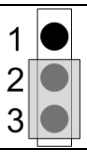
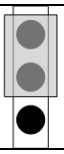
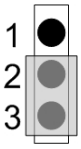
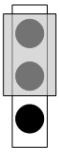
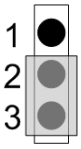
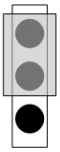
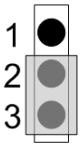
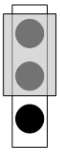
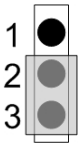
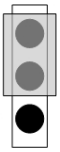
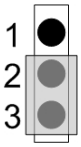
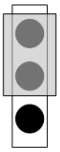
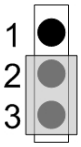
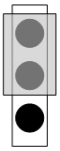
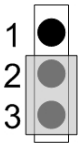
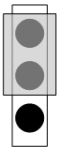
<table>
<tr><th>跳线</th><th>描述</th><th colspan="2">状态</th></tr>
<tr><td>JP5</td><td>CAN Port 3 端子连接 PISO-CAN400U 板卡与 ADP-9 板卡。</td><td colspan="2"> 

Pin1: CAN_L
Pin2: CAN_H
Pin3: Shield</td></tr>
<tr><td>JP7</td><td>CAN Port 4 端子连接 PISO-CAN400U 板卡与 ADP-9 板卡。</td><td colspan="2"> 

Pin1: CAN_L
Pin2: CAN_H
Pin3: Shield</td></tr>
<tr><td rowspan="2">JP2</td><td rowspan="2">Port 1 终端电阻(120Ω)选择</td><td>开启</td><td>关闭</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>JP3</td><td>Port 2 终端电阻(120Ω)选择 (PISO-CAN200U/400U 使用)</td><td> 
</td><td> 
</td></tr>
<tr><td>JP4</td><td>Port 3 终端电阻(120Ω)选择 (PISO-CAN400U 使用)</td><td> 
</td><td> 
</td></tr>
<tr><td>JP6</td><td>Port 4 终端电阻(120Ω)选择 (PISO-CAN400U 使用)</td><td> 
</td><td> 
</td></tr>
</table>

PISO-CAN800U

Table 2.3 Jumper Selections

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

PEX-CAN200i-D/T

表 2.4 跳线选择

跳线	描述	状态	
		开启	关闭
JP2	Port 1 终端电阻(120Ω)选择	 1 2 3	 1 2 3
JP3	Port 2 终端电阻(120Ω)选择	 1 2 3	 1 2 3

PCM-CAN100/200/200P

表 2.3 显示了每个模块的相对应开关与讯号设定。

表 2.5:旋转开关设定

开关位置	插槽模块	CLK	ID 选择	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

表 2.6 跳线选择

跳线	说明	状态	
		开启	关闭
JP3	Port 1 终端电阻(120Ω)选择	 3 2 1	 3 2 1
JP4 (适用于 PCM-CAN200)	Port 2 终端电阻(120Ω)选择	 3 2 1	 3 2 1

2.3 脚位定义

PISO-CAN200/400-T, PISO-CAN100U/200U/400U-T和 PEX-CAN200i-T 配备有 1/2/4 套 **5 针螺钉接线端子**， PISO-CAN200/400-D, PISOCAN100U/200U/400U-D、PEX-CAN200i-D和PCM-CAN100/200 配备有 1/2/4 套 **9 针D-sub 公头接线端子，**PISO-CAN800U配备有 2 套 37 针**D-sub 母头接线端子，搭配使用CA-9-3715D/CA-9-3705 转换线**, 使用者可以將37針母頭接線端子轉成多個9針公頭接線端子，用於CAN bus的电线连接，而接线端子的脚位定义详细的说明如下：

2.3.1 5 针螺钉接线端子

CAN 总线接口的 5-pin 螺钉接线端子如图 2.8 所示，且对应的脚位定义如表 2.6 所示。

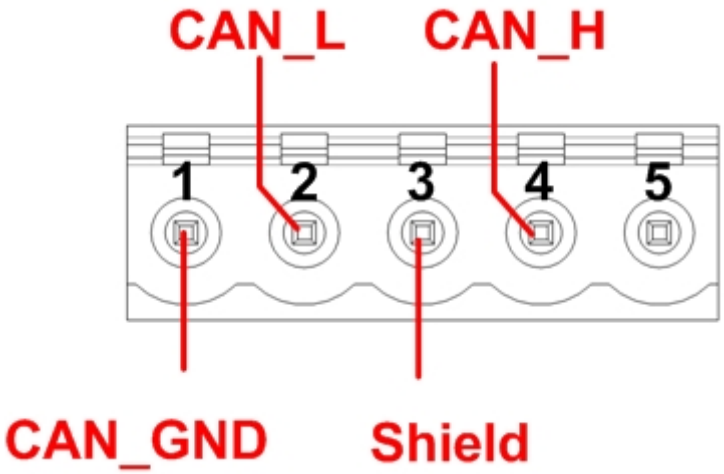


图 2.9　5-pin 螺钉接线端子

表 2.6　5-pin 螺钉接线端子脚位定义

5-pin 螺钉接线端子脚位定义	
1	CAN_GND
2	CAN_L
3	CAN_SHLD
4	CAN_H
5	保留

2.3.2 9 针 D-sub 公头接线端子

CAN 总线接口的 9-pin D-sub 公头接线端子如图 2.9 所示，且对应的脚位定义如表 2.7 所示。

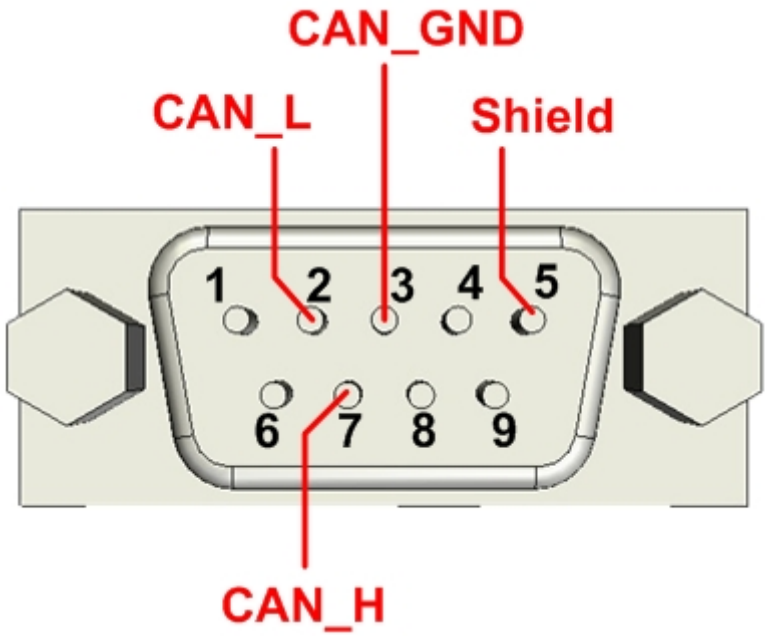


图 2.10 9-pin D-sub 公头接线端子

表 2.7 9-pin D-sub 公头接线端子脚位定义

D-sub 公头接线端子脚位定义	
1	保留
2	CAN_L
3	CAN_GND
4	保留
5	CAN_SHLD
6	保留
7	CAN_H
8	保留
9	保留

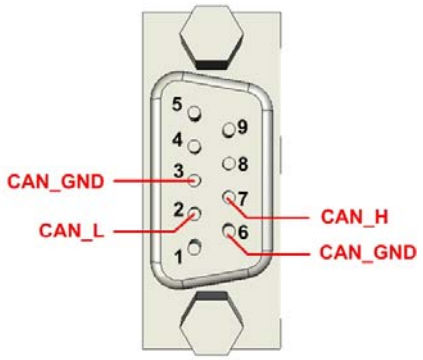
2.3.3 37 针 D-sub 母头接线端子

PISO-CAN800U (CON1) 转 DB-9 接脚定义

(使用 CA-9-3715D/CA-9-3705)

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		

37-Pin Female D-Sub Connector_CAN (CON1)



DB-37 to Male DB-9 Connector_CAN

PISO-CAN800U (CON2) 转 DB-37 接脚定义

CON2			
1	DB-37_**Pin01**	**2**	DB-37_**Pin20**
3	DB-37_**Pin02**	**4**	DB-37_**Pin21**
5	DB-37_**Pin03**	**6**	DB-37_**Pin22**
7	DB-37_**Pin04**	**8**	DB-37_**Pin23**
9	DB-37_**Pin05**	**10**	DB-37_**Pin24**
11	DB-37_**Pin06**	**12**	DB-37_**Pin25**
13	DB-37_**Pin07**	**14**	DB-37_**Pin26**
15	DB-37_**Pin08**	**16**	DB-37_**Pin27**
17	DB-37_**Pin09**	**18**	DB-37_**Pin28**
19	DB-37_**Pin10**	**20**	DB-37_**Pin29**
21	DB-37_**Pin11**	**22**	DB-37_**Pin30**
23	DB-37_**Pin12**	**24**	DB-37_**Pin31**
25	DB-37_**Pin13**	**26**	DB-37_**Pin32**
27	DB-37_**Pin14**	**28**	DB-37_**Pin33**
29	DB-37_**Pin15**	**30**	DB-37_**Pin34**
31	DB-37_**Pin16**	**32**	DB-37_**Pin35**
33	DB-37_**Pin17**	**34**	DB-37_**Pin36**
35	DB-37_**Pin18**	**36**	DB-37_**Pin37**
37	DB-37_**Pin19**	**38**	N.C.
39	N.C.	**40**	N.C.

Pin Assignment Name	Terminal No.		Pin Assignment Name
CAN5_GND	19	37	CAN5_L
CAN5_H	18	36	N.C.
CAN5_GND	17	35	N.C.
N.C.	16	34	N.C.
N.C.	15	33	CAN6_GND
CAN6_L	14	32	CAN6_H
N.C.	13	31	CAN6_GND
N.C.	12	30	N.C.
N.C.	11	29	N.C.
CAN8_GND	10	28	CAN8_L
CAN8_H	09	27	N.C.
CAN8_GND	08	26	N.C.
N.C.	07	25	N.C.
N.C.	06	24	CAN7_GND
CAN7_L	05	23	CAN7_H
N.C.	04	22	CAN7_GND
N.C.	03	21	N.C.
N.C.	02	20	N.C.
N.C.	01		

37-Pin Female D-Sub Connector_CAN (CON2)

2.4 硬件安装

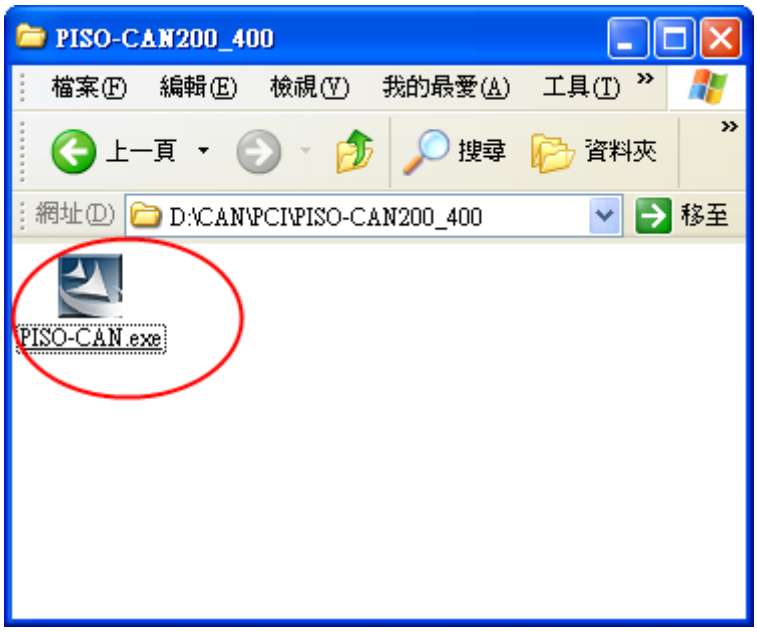
1. 在 PISO-CAN/PEX-CAN/PCM-CAN 上更改跳线设定，以符合您特定的要求。
2. 关闭电源并且移除计算机上盖。
3. 安装 PISO-CAN、PEX-CAN 或 PCM-CAN 系列的 CAN 卡，至适合的空 PCI 插槽。
4. 装回计算机上盖。
5. 安装 CAN bus 接线至 5-pin 螺钉接头或 9-pin D-sub 接头。
6. 当硬件安装完成后，请开启计算机电源。

3 软件安装

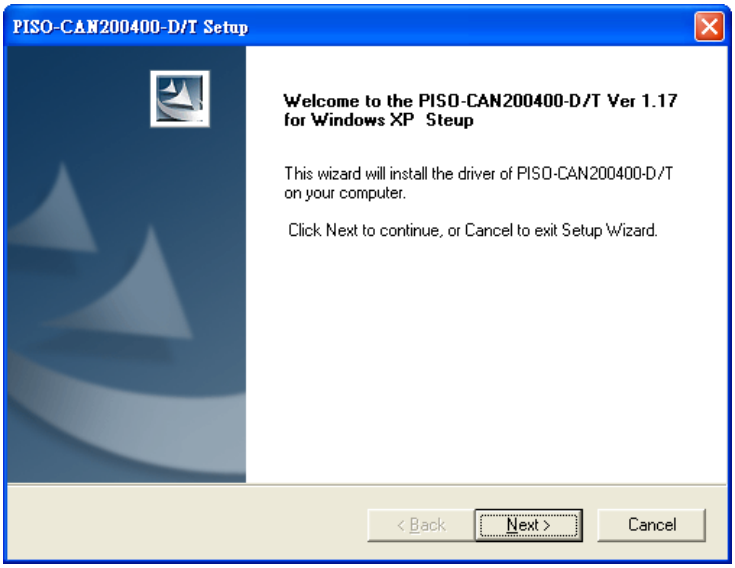
PISO-CAN 或 PCM-CAN 驱动程序可用于 Windows 2K/XP 之环境，使用者可以从软件光盘的“\CAN\PCI\PCM_PISO-CAN_series\driver”路径中，找到并执行“PISO-CAN.exe”来开始安装驱动程序。

安装PISO-CAN 或 PCM-CAN 卡驱动程序

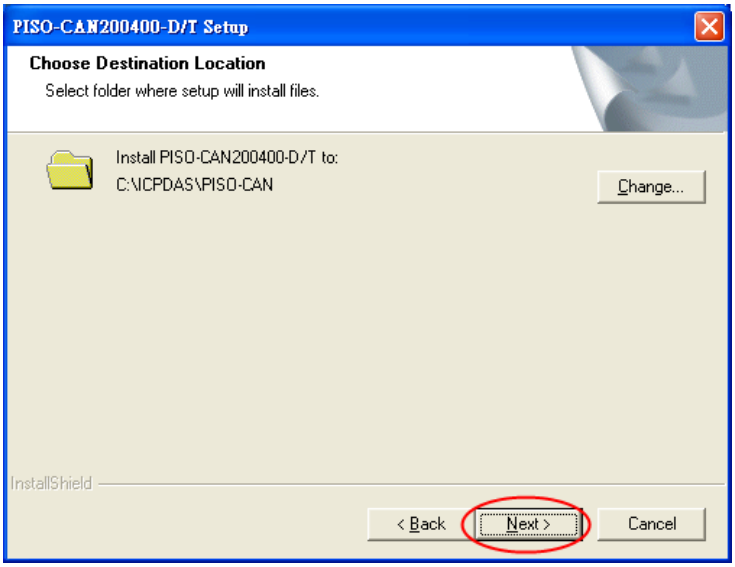
Step 1: 放置产品光盘至光驱并且于路径“\CAN\PCI\PCM_PISO-CAN_series\driver\win_2k_xp_7”(例如:操作系统为 Windows 2000/XP/7)下找到“PISO-CAN.exe”并且执行。



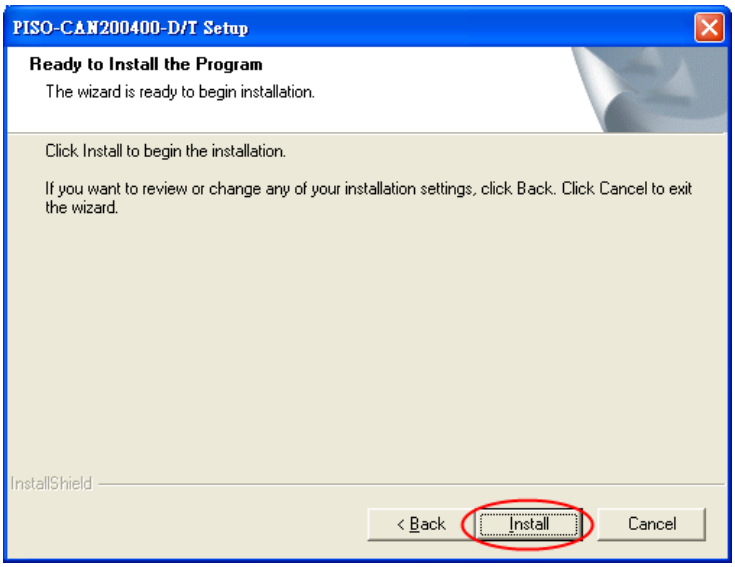
Step 2: 点击「Next」开始安装 PISO-CAN。



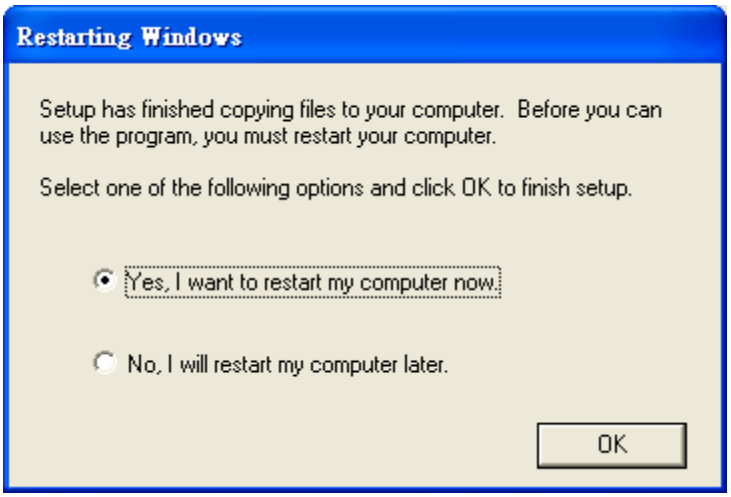
Step 3: 选择 PISO-CAN 安装数据夹，并点击「Next」继续安装。



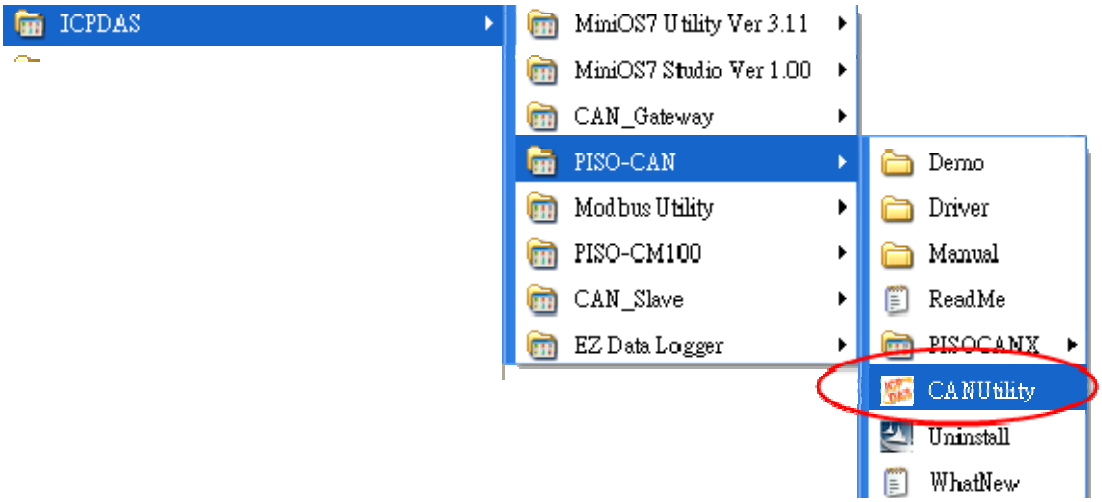
Step 4: 点击「Install」继续安装。



Step 5: 最后，重新启动计算机以完成安装。

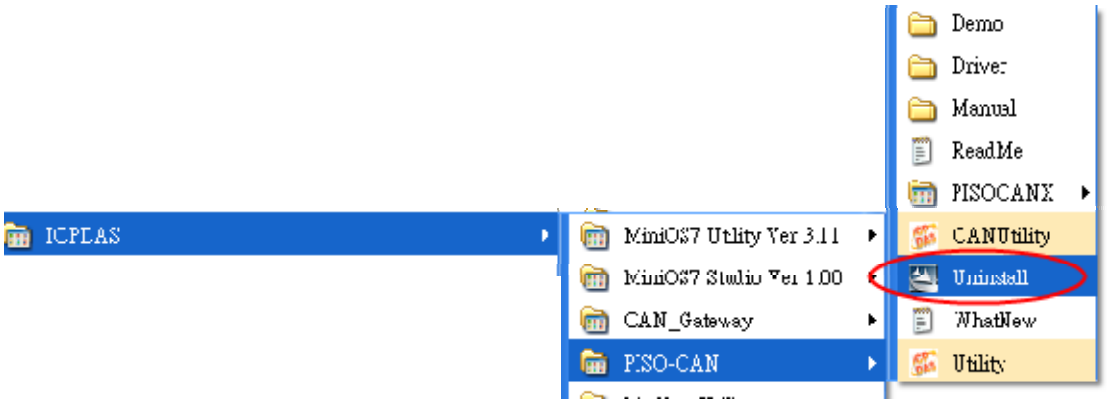


当安装完成后，在 Windows“开始”启动列中，可找到 PISO-CAN 数据夹，如下图。



移除 PISO-CAN 驱动程序

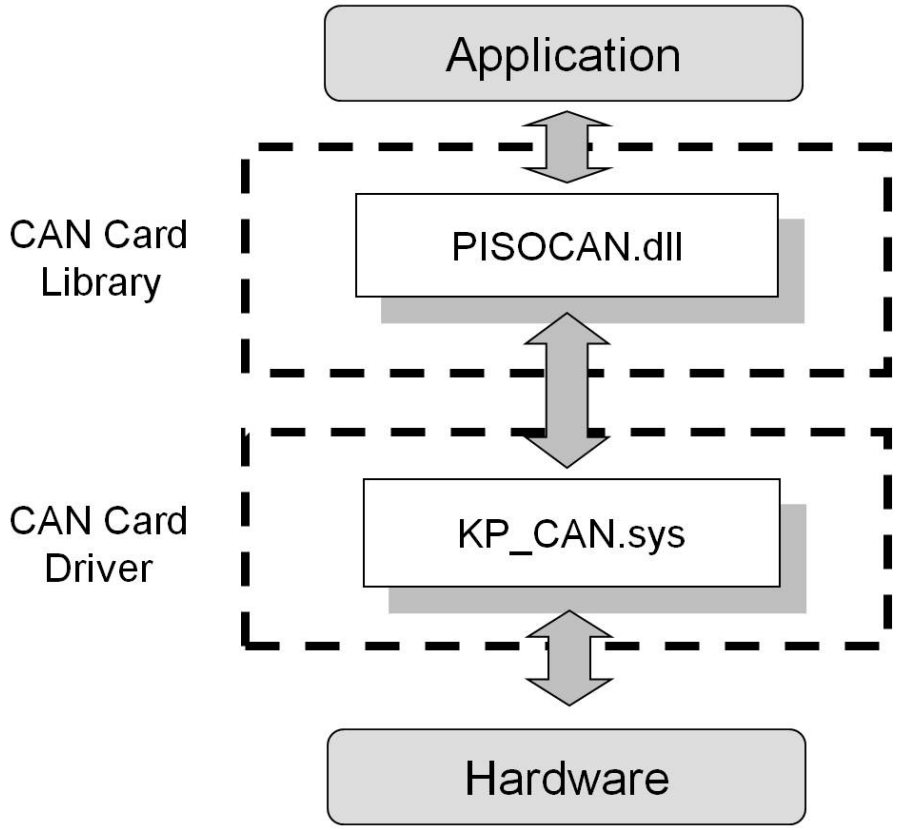
如果使用者要移除 PISO-CAN 驱动程序，请点击「Uninstall」，如下图。



4 DLL 驱动程序安装

Windows DLL 驱动程序

DLL 驱动程序是用于 Windows 2000/XP/7 环境中，能呼叫 PISO-CAN、PEX-CAN 和 PCM-CAN 系列的函式库，其应用架构如下图所示。使用者的应用程序可以使用 VB、VC、Delphi 及 Borland C++ Builder…等开发工具在使用者模式呼叫“PISOCAN.DLL”驱动程序。DLL 驱动程序会将函式呼叫的指令传送至 KP_CAN.sys，以存取硬件系统，如下图所示。



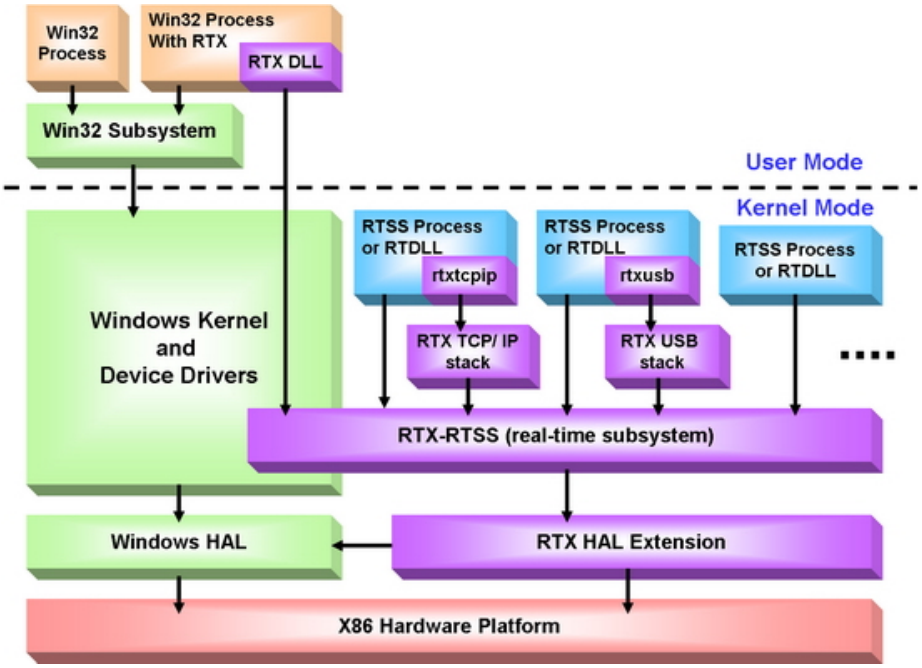
RTX 驱动程序

为了满足用户应用 RTX 系统，泓格提供了 PISO-CAN 系列 CAN 卡的 RTX 驱动程序，如果用户想 CAN 通信接口结合起来至他们的时间关键型(time-critical)系统中，PISO-CAN 系列的 CAN 卡 RTX 驱动程序可以帮助他们轻松快速地做到这一点。此外，RTX 驱动程序 API 的名称和参数都是与 Windows 驱动程序相同的，如果用户之前使用的是 Windows 驱动程序，他们将不需要付出更多的努力学习如何使用 RTX 驱动程序的 API。RTX 驱动程序增加了 PISO-CAN 系列 CAN 卡的附加价值，并满足用户获得高度的实时系统。由于高性价比和高实时性的特点，PISO-CAN 系列 CAN 卡将更广泛的使用在 CAN Bus 的应用环境上。

特点：

1． RTX 驱动程序的 API 的名称和参数与 Windows 驱动程序相同。如果用户先前已使用 PISO-CAN 系列 CAN 卡的话，将不需要学习新的 API 的使用方式
2． 如果 PISO-CAN 系列 CAN 卡可以得到独立的 IRQ，它将支持中断功能
3． 直接 I/ O 控制和高实时性
4． 支持在 Windows2000 SP4 和 Windows XP SP2 操作系统
5． 支持 RTX 8.1 和 2011 版本

备注:
在执行 PISO-CAN 系列 RTX 的 API 之前，用户需要先执行“pisocan_rtx.rtss”文件。



4.1 DLL 函式定义与描述

表 4.1 与表 4.2 列出在 PISO-CAN、PEX-CAN 或 PCM-CAN **(以下简称 PISO-CAN)**中，提供的所有函式，且每一个函式的详细信息会在接下来的小节介绍。然而，为了使描述更为简化与清楚，分别以 **[input]** 和 **[output]** 表示输入与输出参数的函式属性，如下表所示。

关键词	函式呼叫前，由使用者设定参数？	函式呼叫后，从参数取得数据？
[input]	是	否
[output]	否	是

表 4.1 DLL 函式定义

函式定义	章节
WORD CAN_GetDllVersion();	4.1.1
int CAN_TotalBoard();	4.1.2
int CAN_GetBoardInf(BYTE BoardNo, DWORD *dwVID, DWORD *dwDID, DWORD *dwSVID, DWORD *dwSDID, DWORD *dwIrqNo);	4.1.3
int CAN_GetCardPortNum(BYTE BoardNo, BYTE *bGetPortNum);	4.1.4
int CAN_ActiveBoard(BYTE wBoardNo)	4.1.5
int CAN_CloseBoard(BYTE wBoardNo);	4.1.6
int CAN_BoardIsActive(BYTE BoardNo);	4.1.7
int CAN_Reset(BYTE BoardNo, BYTE Port);	4.1.8
int CAN_Init(BYTE wBoardNo, BYTE Port);	4.1.9
int CAN_Config(BYTE BoardNo, BYTE Port, ConfigStruct *CanConfig);	4.1.10
int CAN_ConfigWithoutStructure(BYTE BoardNo, BYTE Port, DWORD AccCode, DWORD AccMask, BYTE BaudRate, BYTE BT0, BYTE BT1);	4.1.11
int CAN_EnableRxIrq(BYTE BoardNo ,BYTE Port);	4.1.12
int CAN_DisableRxIrq(BYTE BoardNo, BYTE Port);	4.1.13
int CAN_RxIrqStatus(BYTE BoardNo, BYTE Port, BYTE *bStatus);	4.1.14
int CAN_InstallIrq(BYTE BoardNo);	4.1.15
int CAN_RemoveIrq(BYTE BoardNo);	4.1.16
int CAN_IrqStatus(BYTE BoardNo, BYTE *bStatus);	4.1.17
int CAN_Status(BYTE BoardNo, BYTE Port, BYTE *bStatus);	4.1.18
int CAN_SendMsg(BYTE BoardNo, BYTE Port, PacketStruct *CanPacket);	4.1.19
int CAN_SendWithoutStruct(BYTE BoardNo, BYTE Port, BYTE Mode, DWORD Id, BYTE Rtr, BYTE Dlen, BYTE *Data)	4.1.20
int CAN_RxMsgCount(BYTE BoardNo, BYTE Port);	4.1.21
int CAN_ReceiveMsg(BYTE BoardNo, BYTE Port, PacketStruct *CanPacket);	4.1.22
int CAN_ReceiveWithoutStruct(BYTE BoardNo, BYTE Port, BYTE *Mode, DWORD *Id, BYTE *Rtr, BYTE *Dlen, BYTE *Data, LONGLONG *MsgTimeStamps);	4.1.23
int CAN_ClearSoftBuffer(BYTE BoardNo, BYTE Port);	4.1.24
int CAN_ClearDataOverrun(BYTE BoardNo, BYTE Port);	4.1.25
void CAN_OutputByte(BYTE BoardNo, BYTE Port, WORD wOffset, BYTE bValue);	4.1.26
BYTE CAN_InputByte(BYTE BoardNo, BYTE Port, WORD wOffset);	4.1.27
LONGLONG CAN_GetSystemFreq(void);	4.1.28
Int CAN_InstallUserIsr(BYTE BoardNo, void(*UserISR)(BYTE BoardNo));	4.1.29
Int CAN_RemoveUserIsr(BYTE BoardNo);	4.1.30
int CAN_BusErrorCode(BYTE BoardNo, BYTE Port, BYTE *bErrorCode);	4.1.31

表 4.2 回传码的说明

回传码	错误名称	备注
0	CAN_NoError	正常
1	CAN_DriverError	驱动程序错误
2	CAN_ActiveBoardError	板卡无法启动
3	CAN_BoardNumberError	板卡编号超出最大板卡数量(7)
4	CAN_PortNumberError	通讯端口编号超出最大通讯端口数量
5	CAN_ResetError	CAN 芯片硬件重新启动错误
6	CAN_SoftResetError	CAN芯片软件重新启动错误
7	CAN_InitError	CAN芯片初始化错误
8	CAN_ConfigError	CAN 芯片配置错误
9	CAN_SetACRError	设定接受码缓存器错误
10	CAN_SetAMRError	设定接受屏蔽缓存器错误
11	CAN_SetBaudRateError	设定鲍率错误
12	CAN_EnableRxIrqFailure	CAN 芯片的接收中断功能启用失败
13	CAN_DisableRxIrqFailure	CAN 芯片的接收中断功能关闭失败
14	CAN_InstallIrqFailure	PCI 板卡的 IRQ 安装失败
15	CAN_RemoveIrqFailure	PCI 板卡的 IRQ 移除失败
16	CAN_TransmitBufferLocked	CAN 芯片的传送缓冲器被锁住
17	CAN_TransmitIncomplete	先前的传输尚未完成
18	CAN_ReceiveBufferEmpty	CAN芯片的RXFIFO是空的
19	CAN_DataOverrun	CAN芯片的RXFIFO没有足够的空间，造成数据遗失
20	CAN_ReceiveError	数据接收未完成
21	CAN_SoftBufferIsEmpty	驱动程序的软件缓冲器是空的
22	CAN_SoftBufferIsFull	驱动程序的软件缓冲器是满的
23	CAN_TimeOut	函式无回应与超时
24	CAN_InstallIsrError	使用者 ISR 安装失败

4.1.1 CAN_GetDllVersion

- **说明:**

  取得 PISOCAN.dll 驱动程序的版本信息

- **语法:**

  WORD CAN_GetDllVersion(viod)

- **参数:**

  无

- **回传:**

  DLL 版本信息。例如，如果传回 101(hex)，表示驱动程序版本为 1.01。

4.1.2 CAN_TotalBoard

- **说明:**

  取得所有安装在 PCI bus 的 CAN 板卡数量。

- **语法:**

  int CAN_TotalBoard(void)

- **参数:**

  无。

- **回传:**

  回传板卡数量。

4.1.3 CAN_GetBoardInf

- **说明:**

  取得 PISO-CAN 板卡的信息，包含供货商名称，设备名称及中断编号。

- **语法:**

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

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号。
  *dwVID: [output]板卡的供货商名称。
  *dwDID: [output]板卡的设备名称。
  *dwSVID: [output]板卡的子供货商名称。
  *dwSDID: [output]板卡的子设备名称。
  *dwSAuxID: [output]板卡的子辅助名称。
  *dwIrq: [output]板卡的逻辑中断编号。

- **回传:**

  CAN_NoError：正常。

  CAN_DriverError： 核心驱动程序无法开启。

  CAN_BoardNumberError：“BoardNo”超出目前所有板卡编号。

4.1.4 CAN_GetCardPortNum

- **Description:**

  取得 PISO-CAN 卡的 CAN 通讯端口编号。

- **Syntax:**

  int CAN_GetCardPortNum(BYTE BoardNo, BYTE *bGetPortNum)

- **Parameter:**

  BoardNo: [input] PISO-CAN 板卡编号 (0~7)。
  * bGetPortNum: [output] CAN 卡的通讯端口编号。

- **Return:**

  CAN_NoError: 正常。

  CAN_DriverError: 核心驱动程序无法开启。

  CAN_BoardNumberError: “BoardNo”值超出目前所有板卡编号。

4.1.5 CAN_ActiveBoard

- **说明:**

  启动设备。**在使用其它 PISO-CAN 板卡的功能之前，必须呼叫此函式一次。**

- **语法:**

  int　CAN_ActiveBoard(BYTE BoardNo)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号 (0~7)。.

- **回传:**

  CAN_NoError: 正常

  CAN_BoardNumberError: “BoardNo”值超出目前所有板卡编号。

  CAN_ActiveBoardError: 此板卡无法启动或核心驱动程序无法开启。

4.1.6 CAN_CloseBoard

- **说明:**

  停止并关闭核心驱动程序且释放计算机端的设备资源。在离开使用者应用程序之前，这个函式必须被呼叫一次。

- **语法:**

  int　CAN_CloseBoard(BYTE BoardNo)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7)。

- **回传:**

  CAN_NoError: 正常。

  CAN_ActiveBoardError:板卡未启动。

  CAN_BoardNumberError: “BoardNo”值超出目前所有板卡编号。

4.1.7 CAN_BoardIsActive

- **说明:**

  检查该版卡是否已经启动。

- **语法:**

  int CAN_BoardIsActive(BYTE BoardNo)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号 (0~7)。

- **回传:**

  0: 表示板卡未启动。

  1: 表示板卡已启动。

4.1.8 CAN_Reset

- **说明:**

  重新启动 CAN 控制器。

- **语法:**

  int CAN_Reset(BYTE BoardNo, BYTE Port)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号 (0~7)。

  Port: [input] CAN 通讯端口编号 (1~8)。

- **回传:**

  CAN_NoError: 正常。

  CAN_DriverError: 核心驱动程序无法开启。

  CAN_BoardNumberError: “BoardNo”不正确，或超出目前所有板卡编号。

  CAN_PortNumberError: “Port”编号不正确。

  CAN_ActiveBoardError: 此板卡未被启动。

4.1.9 CAN_Init

- **说明:**

  初始化 CAN 控制器。

- **语法:**

  int CAN_Init(BYTE BoardNo, BYTE Port)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号 (0~7)。

  Port: [input] CAN 通讯端口编号 (1~8)。

- **回传:**

  CAN_NoError: 正常。

  CAN_DriverError: 核心驱动程序无法开启。

  CAN_BoardNumberError:“ BoardNo”不正确，或超出目前所有板卡的编号。

  CAN_PortNumberError: “Port”编号不正确。

  CAN_ActiveBoardError: 此板卡未被启动。

  CAN_InitError: CAN 控制器初始化失败。

4.1.10 CAN_Config

- **说明:**

  CAN 控制器组态。在呼叫此函式后，CAN 控制器将进入操作模式。

- **语法:**

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

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7).

  Port: [input] CAN 通讯端口编号(1~8)

  *ConfigStruct: [input]“ConfigStruct”指针的结构定义如下：

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

  AccCode[4]: CAN 控制器接受码。

  AccMask[4]: CAN 控制器接受屏蔽。

  BaudRate: 0→使用者定义(必须设定 BT0、BT1)； 1→10Kbps；
  2→20Kbps； 3→50Kbps；4→125Kbps； 5→250Kbps；
  6→500Kbps；7→800Kbps；8→1Mbps。

  BT0, BT1: 使用者定义鲍率 (若“BaudRate=0”时使用)。例如，
  BT0=0x04、 BT1=0x1C、CAN 控制器的鲍率设定便是

100Kbps。更详细的鲍率设定，请参考 SJA1000 CAN 控制器的手册。

- **回传:**

  CAN_NoError: 正常。

  CAN_DriverError: 核心驅動程式无法開啟。

  CAN_BoardNumberError: “BoardNo”不正确，或超出目前所有板卡編號。

  CAN_PortNumberError: “Port”編號不正确。

  CAN_ActiveBoardError: 此板卡未被啟動。

  CAN_SoftResetError: CAN 控制器软件重新启动错误。

  CAN_SetACRError: 设定 CAN 控制器接受码发生错误。

  CAN_SetAMRError: 设定 CAN 控制器接受屏蔽发生错误。

  CAN_SetBaudRateError: 设定 CAN 控制器鲍率发生错误。

  CAN_ConfigError: CAN 控制器进入操作模式失败。

4.1.11 CAN_ConfigWithoutStructure

- **说明:**

  此函式与“**CAN_Config**”相同，但不使用“ConfigStruct”结构型态。提供这个功能是因为部份程序开发平台在编译 PISOCAN.lib 时，结构地址会有不一样的分配方式。因此，如果使用者使用“CAN_Config”却无法正确配置 CAN 卡的时候，就可以使用“CAN_ConfigWithoutStruct”函式来代替。

- **语法:**

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

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7)。

  Port: [input] CAN 通讯端口编号(1~8)。

  AccCode: CAN 控制器接收码，低位对应至 ACR[3]缓存器，高位对应至 ACR[0]缓存器。

  AccMask: CAN 控制器接收屏蔽，低位对应至 AMR[3]缓存器，高位对应至 AMR[0]缓存器。

  BaudRate: 0→使用者定义(必须设定 BT0、BT1)； 1→10Kbps；
  2→20Kbps； 3→50Kbps；4→125Kbps；5→250Kbps；
  6→500Kbps；7→800Kbps； 8→1Mbps。

  BT0, BT1: 使用者定义鲍率(若“BaudRate=0”时使用)。例如，BT0=0x04、 BT1=0x1C，CAN 控制器的鲍率设定便是 100Kbps。更详细的鲍率设定，请参考 SJA1000 CAN 控制器的手册。

- **回传:**

  CAN_NoError: 正常。

  CAN_DriverError: 核心驅動程式无法開啟。

  CAN_BoardNumberError: “BoardNo”不正确，或超出目前所有板卡編號。

  CAN_PortNumberError: “Port”編號不正确。

  CAN_ActiveBoardError: 此板卡未被啟動。

  CAN_SoftResetError: CAN 控制器軟體重新啟動錯誤。

  CAN_SetACRError: 設定CAN控制器接受碼发生錯誤。

  CAN_SetAMRError: 設定CAN控制器接受遮罩发生錯誤。

  CAN_SetBaudRateError: 設定CAN控制器鮑率发生錯誤。

  CAN_ConfigError: CAN控制器進入操作模式失敗。

4.1.12 CAN_EnableRxIrq

- **说明:**

  启用 CAN 控制器的接收中断功能。

- **语法:**

  int CAN_EnableRxIrq(BYTE BoardNo, BYTE Port)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7)。

  Port: [input] CAN 通讯端口编号(1~8)。

- **回传:**

  CAN_NoError: 正常。

  CAN_DriverError: 核心驅動程式无法開啟。

  CAN_BoardNumberError: “BoardNo”不正确，或超出目前所有板卡編號。

  CAN_PortNumberError: “Port”編號不正确。

  CAN_ActiveBoardError: 此板卡未被啟動。

  CAN_EnableRxIrqFailure: 启用接收中断功能失败。

4.1.13 CAN_DisableRxIrq

- **说明:**

  停用 CAN 控制器的接收中断功能。

- **语法:**

  Int CAN_DisableRxIrq(BYTE BoardNo, BYTE Port)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7)。

  Port: [input] CAN 通讯端口编号(1~8)。

- **回传:**

  CAN_NoError: 正常。

  CAN_DriverError: 核心驅動程式无法開啟。

  CAN_BoardNumberError: “BoardNo”不正确，或超出目前所有板卡編號。

  CAN_PortNumberError: “Port”編號不正确。

  CAN_ActiveBoardError: 此板卡未被啟動。

  CAN_DisableRxIrqFailure:停用接收中断功能失败。

4.1.14 CAN_RxIrqStatus

- **说明:**

  取得 CAN 控制器接收中断功能的状态。

- **语法:**

  int CAN_RxIrqStatus(BYTE BoardNo, BYTE Port, BYTE *bStatus)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7)。

  Port: [input] CAN 通讯端口编号(1~8)。

  *bStatus:[output] 0→停用接收中断;

  1→启用接收中断。

- **回传:**

  CAN_NoError: 正常。

  CAN_DriverError: 核心驅動程式无法開啟。

  CAN_BoardNumberError: “BoardNo”不正确, 或超出目前所有板卡編號。

  CAN_PortNumberError: “Port”編號不正确。

  CAN_ActiveBoardError: 此板卡未被啟動。

4.1.15 CAN_InstallIrq

- **说明:**

  对 PISO-CAN board 板卡启用或启动 IRQ 。在呼叫此函式之前，必须先呼叫“**CAN_EnableRxIrq**”函式。

- **语法:**

  int CAN_InstallIrq(BYTE BoardNo)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号 (0~7).

- **回传:**

  CAN_NoError: 正常。

  CAN_DriverError: 核心驱动程序无法开启。

  CAN_BoardNumberError:“BoardNo”不正确，或超出目前所有板卡编号。

  CAN_ActiveBoardError: 此板卡未被启动。

  CAN_InstallIrqFailure: 启用或启动 IRQ 失败。

4.1.16 CAN_RemoveIrq

- **说明:**

  停用或停止 PISO-CAN 板卡的 IRQ。在呼叫此函式后，板卡上的所有 CAN 控制器的 IRQ 功能将被停用。

- **回传:**

  int CAN_RemoveIrq(BYTE BoardNo)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7)。

- **回传:**

  CAN_NoError: 正常。

  CAN_DriverError: 核心驱动程序无法开启。

  CAN_BoardNumberError: “BoardNo”不正确，或超出目前所有板卡编号。

  CAN_ActiveBoardError: 此板卡未被启动。

  CAN_RemoveIrqFailure: 停用或停止 IRQ 失败。

4.1.17 CAN_IrqStatus

- **说明:**

  取得 PISO-CAN 板卡的 IRQ 状态。

- **语法:**

  int CAN_IrqStatus(BYTE BoardNo, BYTE *bStatus)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号 (0~7)。

  *bStatus:[output] 0→停用 IRQ。

  1→启用 IRQ。

- **回传:**

  CAN_NoError: 正常。

  CAN_DriverError: 核心驱动程序无法开启。

  CAN_BoardNumberError: “BoardNo”不正确，或超出目前所有板卡编号。

  CAN_ActiveBoardError: 此板卡未被启动。

4.1.18 CAN_Status

- **说明:**

  取得 CAN 控制器的状态。

- **语法:**

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

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7)。

  *bStatus:[output] CAN 控制器的状态值。

表 4.3“bStatus”的位说明。

位	名称	值	状态
bit 7	总线状态	1	总线关闭
		0	总线开启
bit 6	错误状态	1	错误
		0	正常
bit 5	传输状态	1	传输
		0	闲置
bit 4	接收状态	1	接收
		0	闲置
bit 3	传输完成状态	1	完成
		0	未完成
bit 2	传输缓冲器状态	1	释放
		0	锁住
bit 1	数据溢出状态	1	溢出
		0	不存在
bit 0	接收缓冲器状态	1	满的/未满的
		0	空的

- **回传:**

  CAN_NoError: 正常。

  CAN_BoardNumberError:“BoardNo”不正确，或超出目前所有板卡编号。

  CAN_PortNumberError:“Port”编号不正确。

  CAN_ActiveBoardError: 此板卡未被启动。

4.1.19 CAN_SendMsg

- **说明:**

  立即发送 CAN 讯息。

- **语法:**

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

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7)。

  Port: [input] CAN 通讯端口编号(1~8)。

  *CanPacket: [input]“CanPacket”结构定义如下：

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

  MsgTimeStamps:此参数在本函式中没有作用。

  mode: 0→ 11-bit 的识别码； 1 → 29-bit 的识别码。

  id: 识别码。

  rtr: 远程传输请求。

  len: 数据长度。

data[8]: 数据字节。

- **回传:**

  CAN_NoError: 正常。

  CAN_BoardNumberError: “BoardNo”不正确，或超出目前所有板卡编号。

  CAN_PortNumberError: “Port”编号不正确。

  CAN_ActiveBoardError: 此板卡未被启动。

  CAN_TransmitBufferLocked: CAN 芯片的传输缓冲器被锁住。

  CAN_TransmitIncomplete: 尚未完成传输。

  CAN_ConfigError: 通讯端口尚未配置成功。

4.1.20 CAN_SendWithoutStruct

- **说明:**

  此函式与“**CAN_SendMsg**”函式相同，但不使用“PacketStruct”结构型态。如果使用者在部份应用程序开发环境，如.NET 2003，使用“CAN_SendMsg”，却无法正确发送 CAN 讯息，此时便可用“CAN_SendWithoutStruct”函式代替。

- **语法:**

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

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7)。

  Port: [input] CAN 通讯端口编号(1~8)。

  Mode: 0→ 11-bit 的识别码；1 → 29-bit 的识别码。

  Id: 识别码。

  Rtr: 远程传输请求。

  Dlen: 数据长度。

  *Data: 数据字节。

- **回传:**

  CAN_NoError: 正常。

  CAN_BoardNumberError: “BoardNo”不正确。

  CAN_PortNumberError: “Port”编号不正确。

  CAN_ActiveBoardError: 此板卡未被启动。

  CAN_TransmitBufferLocked: CAN 芯片的传输缓冲器被锁住。

  CAN_TransmitIncomplete: 传输尚未完成。

  CAN_ConfigError: 通讯端口尚未配置成功。

4.1.21 CAN_RxMsgCount

- **说明:**

  取得 CAN 控制器的 RXFIFO 或软件缓冲器(4KBytes)里面，可用的 CAN 讯息数量。在呼叫“**CAN_EnableRxIrq**”和“**CAN_InstallIrq**”函式后，所得到的 CAN 讯息数量是软件缓冲区内部的，否则就是 CAN 控制器 RXFIFO 内部的。

- **语法:**

  int CAN_RxMsgCount(BYTE BoardNo, BYTE Port);

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号 (0~7)。

  Port: [input] CAN 通讯端口编号(1~8)。

- **回传:**

  CAN 讯息的数量。

  注意. 如果“*BoardNo*”或“Port”参数不正确，回传值将一直是“0”。

4.1.22 CAN_ReceiveMsg

- **说明:**

  从 CAN 控制器的 RXFIFO 或软件缓冲器中，取得接收讯息。在呼叫“**CAN_EnableRxIrq**”和 “**CAN_InstallIrq**”后，讯息不是在软件缓冲器内，就是在 CAN 控制器的 RXFIFO 内。

  注意! 如果使用者的计算机进入“待机模式”或“睡眠模式”，此函式将不能接收任何讯息。

- **语法:**

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

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7)。

  Port: [input] CAN 通讯端口编号(1~8)。

  *CanPacket: [output]“CanPacket”结构定义如下：

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

  MsgTimeStamps: 此参数是纪录当 CAN 讯息从 SJA1000 接收起来时的

时间标记，在 Windows 98/Me/NT4 环境中，是用系统频率计数器来记录此时间，而在 Windows 2000 /XP 环境中，则以 100-ns 为单位的系统时间中断计数器记录时间。其中系统频率计数器是在计算机开机后，系统就会开始计数。如果在 64 位的 SJA1000 FIFO 里有接收并储存 1 个以上的 CAN 讯息时，这几笔 CAN 讯息的时间标记可能会很接近。

mode: 0→ 11-bit 的识别码；1 → 29-bit 的识别码。

id: 识别码。

rtr: 远程传输请求。

len: 数据长度。

data[8]: 数据字节。

- **回传:**

CAN_NoError: 正常。

CAN_BoardNumberError: “BoardNo”不正确, 或超出目前所有板卡編號。

CAN_PortNumberError: “Port”編號不正确。

CAN_ActiveBoardError: 此板卡未被啟動。

CAN_ConfigError: 通讯端口尚未配置成功。

CAN_ReceiveBufferEmpty: CAN 控制器的 RXFIFO 是空的。

CAN_SoftBufferIsEmpty: 软件的 RX 缓冲器是空的。

CAN_SoftBufferIsFull: 软件的 RX 缓冲器是满的。

4.1.23 CAN_ReceiveWithoutStruct

- **说明:**

  此函式与“**CAN_ReceiveMsg**”函式相同，但不使用“PacketStruct”结构型态。提供这个功能是因为部份程序开发平台在编译 PISOCAN.lib 时，结构地址会有不一样的分配方式，如 .NET 2003。因此，如果使用“CAN_ReceiveMsg”函式，却无法正确的接收 CAN 讯息，那么可以使用“CAN_ReceiveWithoutStruct”函式来代替。

- **语法:**

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

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号 (0~7)。

  Port: [input] CAN 通讯端口编号(1~8)。

  *Mode: 0→ 11-bit 的识别码；1 → 29-bit 的识别码。

  *Id: 识别码。

  *Rtr: 远程传输请求。

  *Dlen: 数据长度。

  *Data: 数据字节

  *H_MsgTimeStamps, *L_MsgTimeStamps: 这些参数是纪录当 CAN 讯息从 SJA1000 接收起来时的时间标记，在 Windows 98/Me/NT4 环境中，是用系统频率计数器来记录此时间，而在 Windows 2000/XP 环境中，则以 100-ns 为单位的系统时间中断计数器记录时间。参数“*H_MsgTimeStamps”是时间标记的高位 DWORD 而参数 *L_MsgTimeStamps 是时间标记的低位 DWORD。其中

系统频率计数器是在计算机开机后，系统就会开始计数。如果在64位的SJA1000 FIFO里有接收并储存1个以上的CAN讯息时，这几笔CAN讯息的时间标记可能会很接近。

- **回传:**

CAN_NoError: 正常。

CAN_BoardNumberError: “BoardNo”不正确, 或超出目前所有板卡編號。

CAN_PortNumberError: “Port”編號不正确。

CAN_ActiveBoardError: 此板卡未被啟動。

CAN_ConfigError: 通讯端口尚未成功启动。

CAN_ReceiveBufferEmpty: CAN 控制器的 RXFIFO 是空的。

CAN_SoftBufferIsEmpty: 软件的 RX 缓冲器是空的。

CAN_SoftBufferIsFull: 软件的 RX 缓冲器是空的。

4.1.24 CAN_ClearSoftBuffer

- **说明:**

  清除 PISOCAN.DLL 驱动程序的软件缓冲器。

- **语法:**

  int CAN_ClearSoftBuffer(BYTE BoardNo, BYTE Port)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7) 。

  Port: [input] CAN 通讯端口编号(1~8)。

- **回传:**

  CAN_NoError: 正常。

  CAN_BoardNumberError: BoardNo”不正确，或超出目前所有板卡编号。

  CAN_PortNumberError: 通讯端口不正确。

4.1.25 CAN_ClearDataOverrun

- **说明:**

  清除 CAN 控制器的数据溢位状态位。

- **语法:**

  int CAN_ClearDataOverrun(BYTE BoardNo, BYTE Port)

- **参数**

  BoardNo: [input] PISO-CAN 板卡编号 (0~7) 。

  Port: [input] CAN 通讯端口编号 (1~8) 。

- **回传:**

  CAN_NoError: 正常。

  CAN_BoardNumberError:“BoardNo”不正确，或超出目前所有板卡编号。

  CAN_PortNumberError: 通讯端口不正确。

  CAN_ActiveBoardError: 此板卡未被启动。

  CAN_ConfigError: CAN 控制器进入操作模式失败。

4.1.26 CAN_OutputByte

- **说明:**

  将数据写入 CAN 芯片 (SJA1000) 。

- **语法:**

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

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号 (0~7)。

  Port: [input] CAN 通讯端口编号(1~8)。

  wOffset: [input] 地址偏移量。

  bValue: [input] 数据字节。

- **回传:**

  無.

4.1.27 CAN_InputByte

- **说明:**

  从 CAN 芯片读取数据(SJA1000)。

- **语法:**

  BYTE CAN_InputByte(BYTE BoardNo, BYTE Port, WORD wOffset)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号 (0~7)。

  Port: [input] CAN 通讯端口编号(1~8) 。

  wOffset: [input] 地址偏移量。

- **回传:**

  CAN 芯片的数据字节。

4.1.28 CAN_GetSystemFreq

- **说明:**

  取得频率频率。此函式对于计算接收讯息的时间标记来说很有用。

- **语法:**

  LONGLONG CAN_GetSystemFreq(void)

- **参数:**

  无。

- **回传:**

  在 Windows 98/Me/NT4 回传的是频率频率，而在 Windows 2000/XP 则回传 10000000。

4.1.29 CAN_InstallUserIsr (适用于 Windows 2000/XP)

- **说明:**

  这个函式允许使用者请求中断服务程序(ISR)，当使用者将自定义的 ISR 放到此函式中后，接收 CAN 讯息的中断将触发此 ISR。

- **语法:**

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

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7)

  (*UserISR)(BYTE BoardNo): [input] 此指标指向一个函式，格式为“void XXX(BYTE BoardNo)”。其中“XXX”是使用者的 ISR 的函式名称，而参数“BoardNo”表示产生中断讯号的板卡编号。

- **回传:**

  CAN_NoError: 正常。

  CAN_DriverError: 核心驱动程序无法打开。

  CAN_BoardNumberError: “BoardNo”值不正确，或超出目前所有板卡的编号。

  CAN_ActiveBoardError: 此板卡未被启动。

  CAN_InstallIrqFailure: 启用或启动 IRQ 失败。

  CAN_InstallIsrError: 启用或启动 ISR 失败。

4.1.30 CAN_RemoveUserIsr (only for Windows 2000/XP)

- **说明:**

  当使用者不需要 ISR 功能时，可呼叫此函式移除 ISR。

- **语法:**

  Int　CAN_RemoveUserIsr(BYTE BoadNo) 。

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7) 。

- **回传:**

  CAN_NoError: 正常。

  CAN_DriverError: 核心驱动程序无法打开。

  CAN_BoardNumberError:“BoardNo”值不正确，或超出目前所有板卡编号。

  CAN_ActiveBoardError: 此板卡未被启动。

  CAN_RemoveIrqFailure: 禁用或停止 IRQ 失败。

4.1.31 CAN_BusErrorCode

- **说明:**

  取得 CAN 控制器 Error Code Capture(ECC)缓存器的数值。

- **语法:**

  int CAN_BusErrorCode(BYTE BoardNo, BYTE Port,BYTE *bErrorCode)

- **参数:**

  BoardNo: [input] PISO-CAN 板卡编号(0~7)。

  Port: [input] CAN 通讯端口编号(1~8)。

  *bErrorCode:[output] CAN 控制器的 Error code capture 缓存器数值。

表 4.4 Error code capture 缓存器的位说明。

Bit	SYMBOL	NAME	VALUE	FUNCTION
ECC.7(1)	ERRC1	Error Code 1	-	-
ECC.6(1)	ERRC0	Error Code 0	-	-
ECC.5(2)	DIR	Direction	1	RX; error occurred during reception
			0	TX; error occurred during transmission
ECC.4(2)	SEG4	Segment 4	-	-
ECC.3(2)	SEG3	Segment 3	-	-
ECC.2(2)	SEG2	Segment 2	-	-
ECC.1(2)	SEG1	Segment 1	-	-
ECC.0(2)	SEG0	Segment 0	-	-

备注:

1. ECC.7 及 ECC.6 缓存器的位说明请参考表 4.5

2. ECC.4 ~ ECC.0 缓存器的位说明请参考表 4.6

Table 4.5 ECC.7 及 ECC.6 缓存器的位说明

BIT ECC.7	BIT ECC.6	功能
0	0	Bit error
0	1	Form error
1	0	Stuff error
1	1	Other type of error

Table 4.6　ECC.4 ~ ECC.0 缓存器的位说明

BIT ECC.4	BIT ECC.3	BIT ECC.2	BIT ECC.1	BIT ECC.0	功能
0	0	0	1	1	Start of frame
0	0	0	1	0	ID.28 to ID.21
0	0	1	1	0	ID.20 to ID.18
0	0	1	0	0	Bit SRTR
0	0	1	0	1	Bit IDE
0	0	1	1	1	ID.17 to ID.13
0	1	1	1	1	ID.12 to ID.5
0	1	1	1	0	ID.4 to ID.0
0	1	1	0	0	Bit RTR
0	1	1	0	1	Reserved bit 1
0	1	0	0	1	Reserved bit 0
0	1	0	1	1	Data length code
0	1	0	1	0	Data field
0	1	0	0	0	CRC sequence
1	1	0	0	0	CRC delimiter
1	1	0	0	1	Acknlwledge slot
1	1	0	1	1	Acknowledge delimiter
1	1	0	1	0	End of frame
1	0	0	1	0	Intermission
1	0	0	0	1	Active error flag
1	0	1	1	0	Passive error flag
1	0	0	1	1	Tolerate dominant bits
1	0	1	1	1	Error delimiter
1	1	1	0	0	Overload flag

- **回传:**

  CAN_NoError: 正常。

  CAN_BoardNumberError:“BoardNo”不正确，或超出目前所有板卡编号。

  CAN_PortNumberError:“Port”编号不正确。

4.2 应用流程图

在这一节，我们将显示 PISO-CAN/PEX-CAN /PCM-CAN 板卡在发送与接收 CAN 讯息的操作程序。 图 4.1 介绍 “CAN 讯息发送”程序。 图 4.2 与 4.3 分别代表中断模式及轮循模式的“CAN 讯息接收”程序。为了能正确且容易的经由 CAN 网络，发送及接收 CAN 讯息，须遵循 PISO-CAN/PEX-CAN/PCM-CAN 板卡的操作规则。更详细的信息，请参考第 5 章的范例程序。

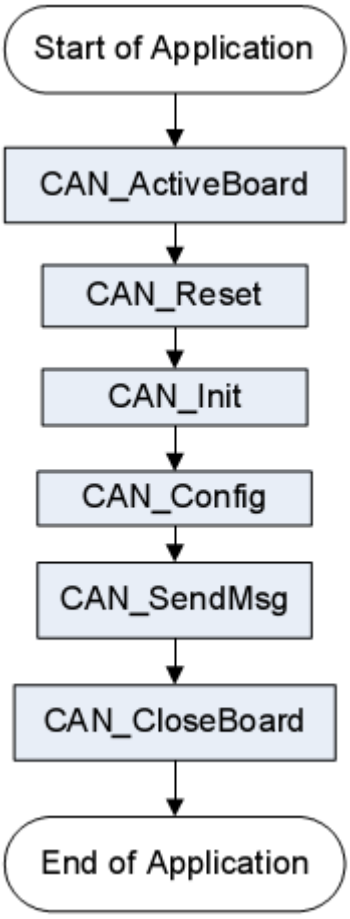


图 4.1 “CAN 讯息发送” 流程图

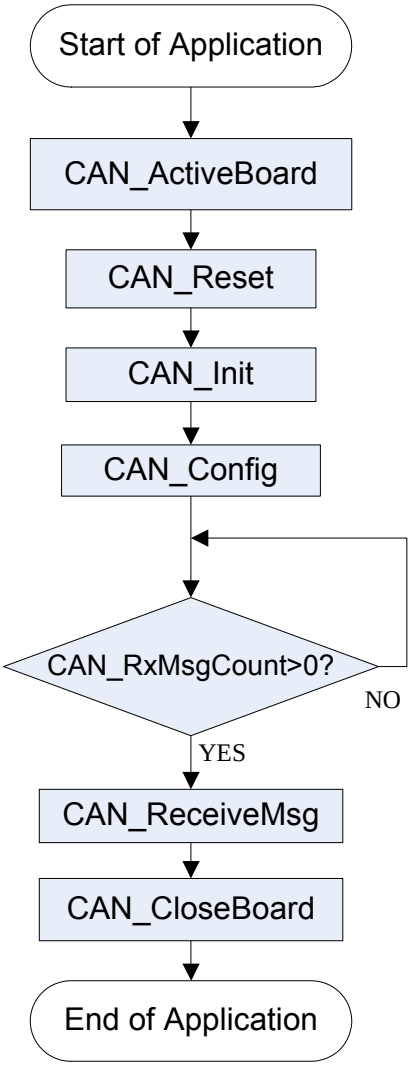


图 4.2 “CAN 讯息接收”流程图

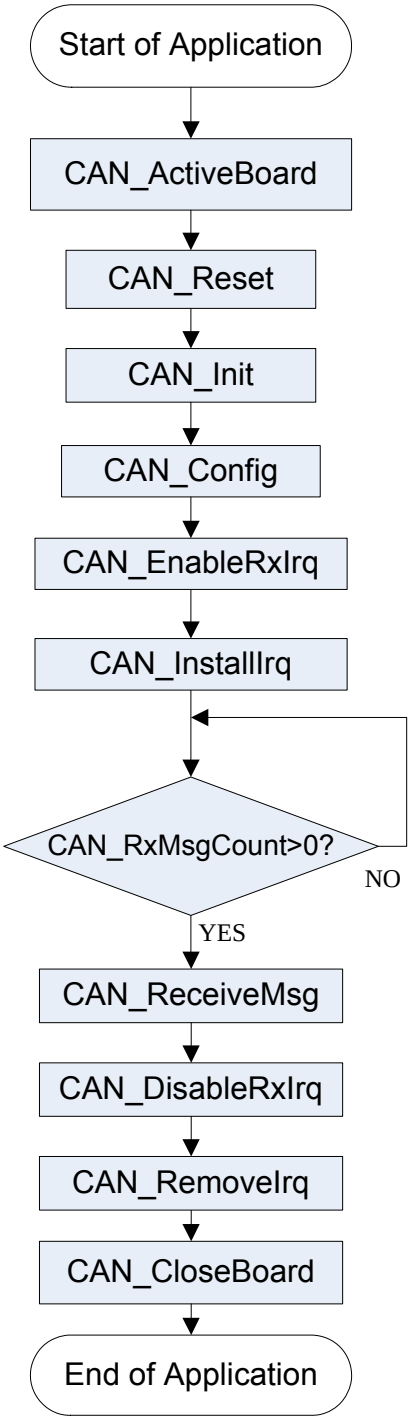


图 4.3“用 IRQ 接收 CAN 讯息”流程图

5 范例程序(适用于 Windows)

如果 DLL 驱动程序未正确地安装，范例程序将不能正常的运作。在安装 DLL 驱动程序期间，安装精灵将对操作系统注册正确的核心驱动程序，并且复制 DLL 驱动程序及范例程序到操作系统(windows 2000、XP、7)的相对应位置。在安装完成后，针对不同的开发环境，其相关的范例程序、开发组件库(library)及标头档(Header file)介绍如下：

```
|--\Demo                          →示范程序
   |--\BCB3                       →适用于 Borland C++ Builder 3
   |      |--\CAN.H               →标头档
   |      |--\PISOCAN.LIB         →BCB 的连结库
   |
   |
   |--\Delphi4                    →适用于 Delphi 4
   |      |--\CAN.PAS             →宣告檔
   |
   |--\VC6                        →适用于 Visual C++ 6
   |      |--\CAN.H               →标头档
   |      |--\PISOCAN.LIB         →VC6 的连结库
   |
   |--\VB6                        →适用于 Visual Basic 6
   |      |--\CAN.BAS             →宣告檔
   |
   |--\C#,Net                     →适用于 C#.Net
   |      |--\ PISOCAN_Net.DLL    →动态连结函式库
   |
   |--\VB,Net                     →适用于 VB.Net
   |      |--\ PISOCAN_Net.DLL    →动态连结函式库
```

范例程序清单：

TxRxCAN_NoIRQ:　传送与接收 CAN 讯号。
TxRxCAN_IRQ:　用 IRQ 传送与接收 CAN 讯号。

范例程序简介

TxRxCAN_NoIRQ:

Demo1 的例子用于启动 PISO-CAN/PEX-CAN /PCM-CAN 板卡。此示范程序是设计由同一块 PISO-CAN/PEX-CAN/PCM-CAN 板卡的 Port 1 发送出 CAN 讯息，并由 Port 2 立即接收 CAN 讯息。在执行此程序前，使用者需先将 CAN 的 Port1 与 Port2 之间的接线连接。在此范例中，使用者可以输入 CAN 讯息到 Port1 的讯息框，并且点击「Send」发送至 Port2。如果在 Port2 的讯息框点击「Receive」，由 Port2 接收的讯息将会显示在“文字”方块中。 屏幕截图显示如下。请注意，如果 Port2 显示一个警告讯息如 CAN 数据溢位，这表示在 64 字节的 RXFIFO CAN 缓冲器被其它讯息覆盖而无法读取。这意味着讯息从 CAN bus 接收时，错误的逻辑运算(and/or)，导致部份讯息遗失。接着使用者可点击「Clear Overrun」清除控制器内的 RXFIFO 缓冲器溢位状态。

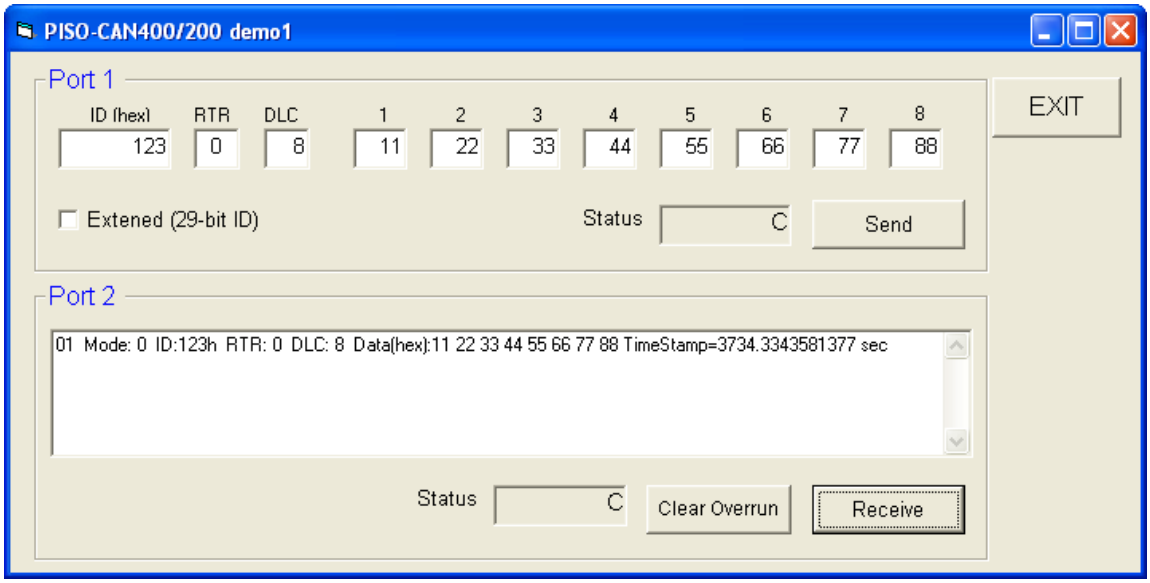


图 5.1: Demo1 的样式

TxRxCAN_IRQ:

在 demo 2，我们提供一个实例示范如何经由“port 1”发送出 CAN 讯号，且以中断模式从“port 2”接收 CAN 讯号。在此范例中，使用者可以输入 CAN 讯息到 port1 的讯息框，并且点击「Send」发送出 CAN 讯息。 同时，port 2 以中断模式接收 CAN 讯息。如下图所示，port 2 可以自动地接收 CAN 讯息，并且储存在软件缓冲器的 4K byte 里。当使用者点击「Receive」，所有储存在 4K bytes 缓冲器里的讯息，将全部显示在文字编辑区域，如下图所示。

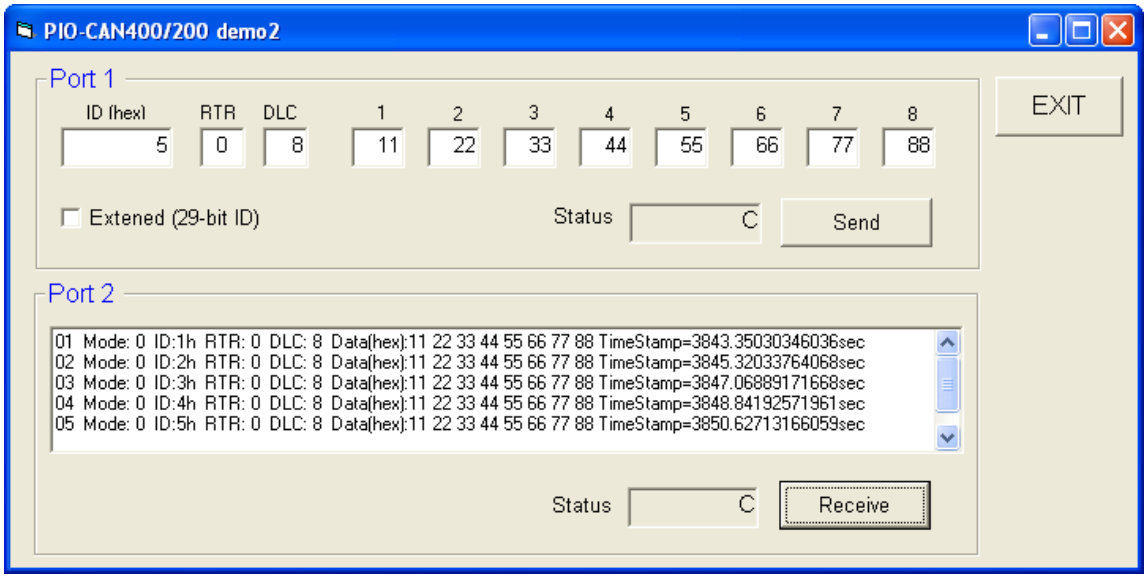


图 5.2: Demo2 的样式

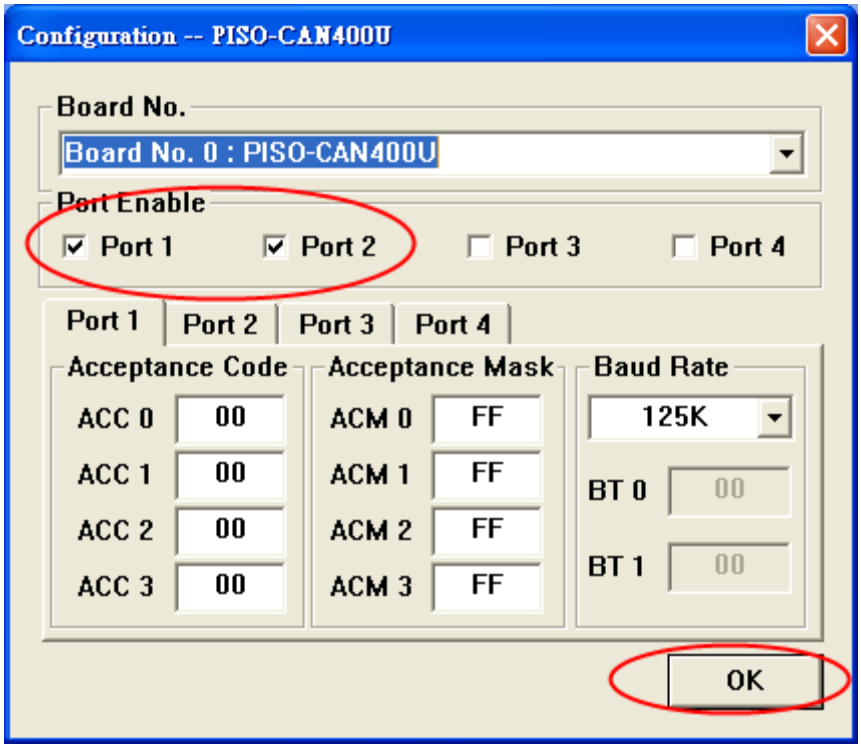
6 CANUtility 工具软件(适用于 Windows)

针对 PISO-CAN、PEX-CAN 或 PCM-CAN，我们提供一个友善的 CAN bus 工具软件，让使用可以在 CAN 网络上轻松的发送及接收 CAN 讯息。此工具软件可以在 CAN 网络上监控讯息或测试设备，而且支持数种功能，如发送 CAN 讯息、接收 CAN 讯息、储存 CAN 讯息、循环传输、等等。其操作方式将在接下来的小节一一介绍。

(1) CAN 配置对话框

请点击「Board No.」选单，选择已装在计算机上，并想要使用的版卡型号。

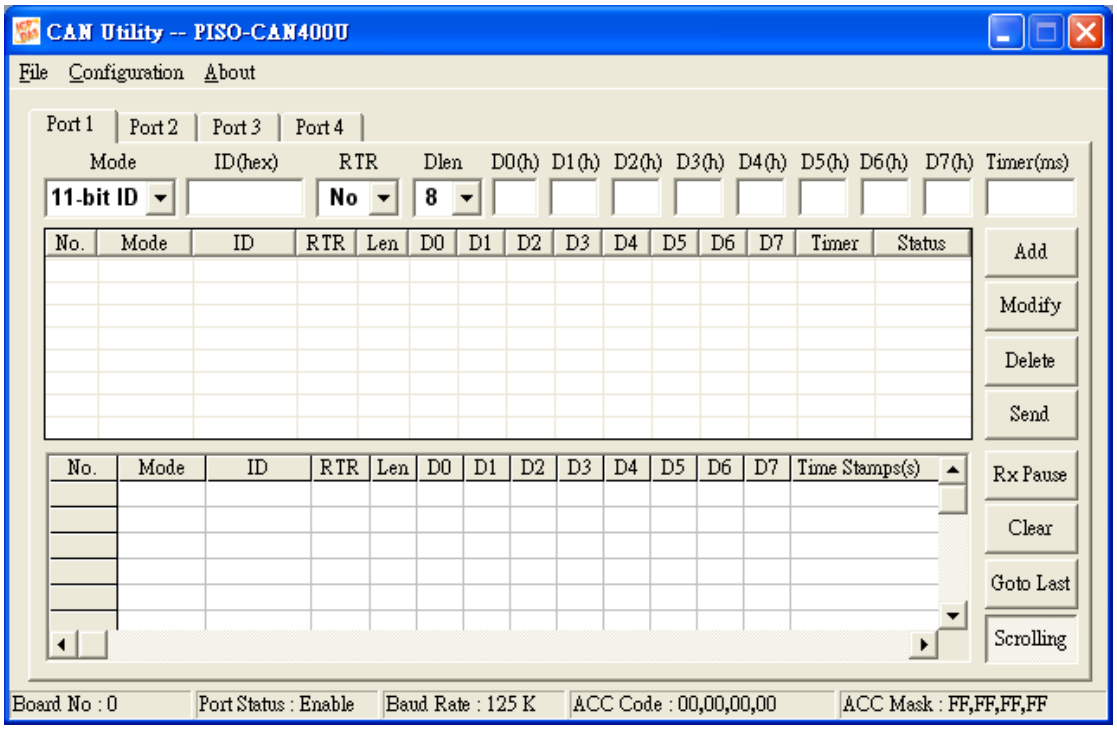
勾选通讯端口启用复选框来启动 CAN 通讯端口，接着点选 CAN 通讯端口标签。根据每个 CAN 通讯的要求，使用者需设定适当的鲍率、接受码及接受屏蔽。鲍率的选项有 8 种，分别为 10K、20K、50K、125K、250K、500K、800K 及 1M。使用者也可以选择使用者定义模式，藉由 BT0 及 BT1 来设定特殊的鲍率，但前提是，使用者需要了解 SJA1000。接着，点击「OK」储存设定。



(2) 主对话框

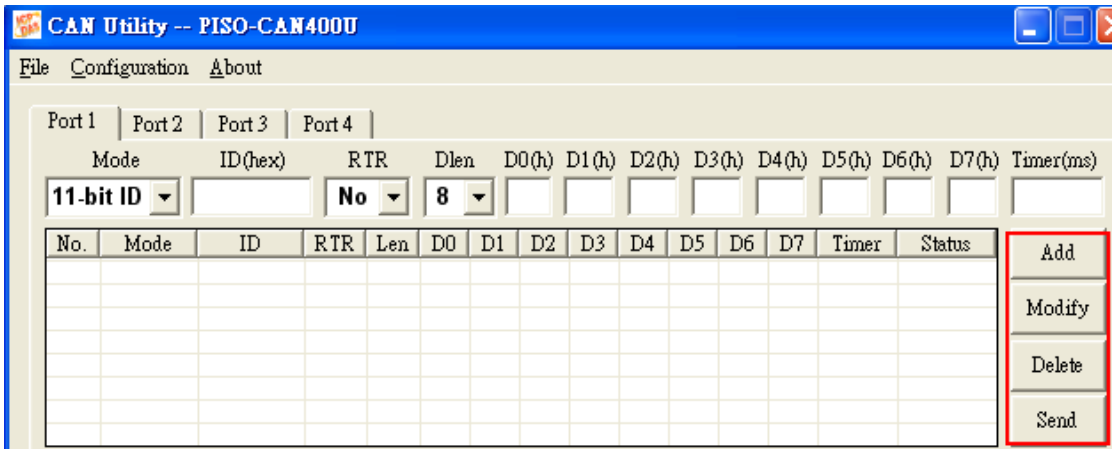
CAN 工具软件的主要对话框如下图所示。会有 1 个页签、2 个页签、4 个页签与 8 个页签的情况分别代表 1 个埠(PCM-CAN100)、2 个埠(PISO-CAN 200/200U, PEX-CAN200i, PCM-CAN200), 4 个埠(PISO-CAN400/400U)及 8 个埠(PISO-CAN800U)。在主对话框的底端，状态列显示被选择的通讯端口的 5 种

参数，分别是板卡编号、通讯端口状态、鲍率、接受码、接受屏蔽。



(3) CAN 传输功能

CAN 通讯端口传输的部份页面如下图所示，其中传输清单包含了下列 4 种功能。



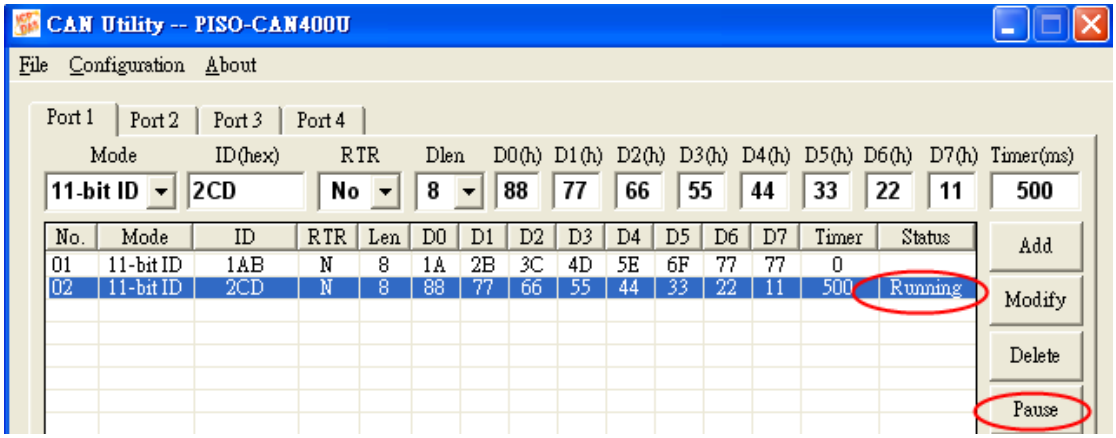
Add 按钮: 使用者可以在文字编辑区内输入 CAN 的讯息内容，接着点击「Add」新增一笔讯息至传输清单。传输清单最多可容纳 20 笔讯息。在新增讯息到传输清单后，可使用「Send」发送讯息至 CAN 网络。

Modify 按钮:传输清单中，如果使用者想修改 CAN 的 讯息内容，首先选择欲修改的 CAN 讯息。接着该讯息的信息显示在传输清单上方的文字编辑区内。使用者可以直接在文字辑区内修改讯息。最后点击「Modify」储存修改设定。

Delete 按钮:在传输清单中，若有一些讯息未使用，可以选取该讯息并点击

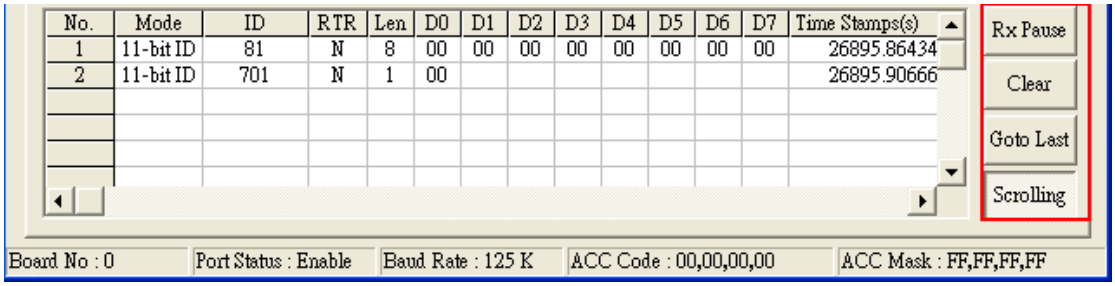
「Delete」来删除讯息。

Send 按钮:从传输清单中选择一笔 CAN 讯息，点击「Send」发送时，会从所选择的通讯端口发送讯息一次。如果 CAN 讯息的 Timer 参数不为 0，讯息发送的时间将取决于 Timer 参数周期。在这例子中，传输清单中的 CAN 讯息状态显示为“Running”，且在「Send」按钮上的文字改变为“Pause”。如果使用者想停止讯息的传输，请再一次点击此按钮。最多允许 5 笔 CAN 讯息，同时从同一个 CAN 通讯端口循环发送讯息。



(4) CAN 接收功能

下图显示所选择的 CAN 通讯端口的接收部份，接收清单包含了下列 4 种功能。



Rx Pause 按钮: 点击此按钮来停止特定 CAN 通讯端口的讯息接收，再点一次便可继续接收讯息。

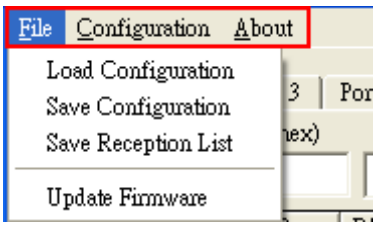
Clear 按钮: 点击此按钮可删除所有显示在接收列表的 CAN 讯息。

Goto Last 按钮: 点击此按钮可直接跳到最后接收的讯息字段。

Scrolling 按钮: 当此按钮被按下时，接受清单将自动卷动至最后一笔讯息。如果这个按钮被按起来，接受清单将停止卷动，但还是持续从 CAN 通讯端口接收讯息。此按钮的预设状态是被按下的。

(5) 功能选项

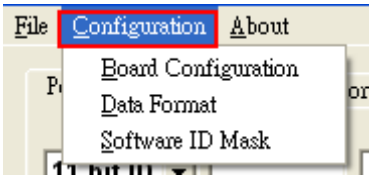
CANUtility 软件工具有三个功能选项。



File 项目:

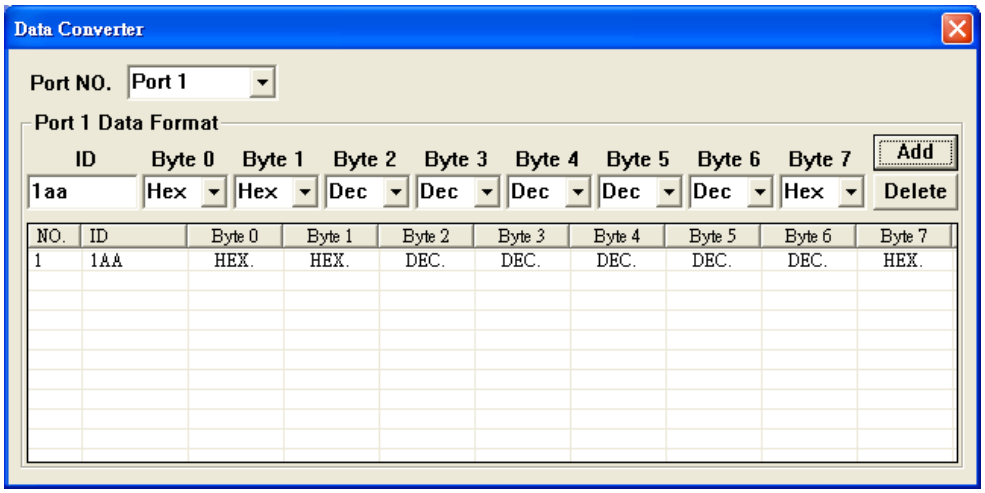
- **Load Configuration:** 如果使用者在使用 CANUtility 之前，已有储存配置设定，可点击此功能加载原有的设定。
- **Save Configuration:** 此功能会将每一个 CAN 通讯端口的传输清单、数据格式清单及 ID 屏蔽清单储存至一个“.txt”文字文件中。
- **Save Reception List:** 此功能用于储存接收清单中的讯息。每一个不同 CAN 通讯端口的接受清单中，除了没有讯息的之外，其余的数据被储存至“.txt”文字文件中。例如，如果使用者想将接收清单中的数据储存到“test.txt”檔。一般来说，当使用者用 PISO-CAN400 时，数据会被储存到四个“.txt”檔，text_port01.txt、text_port02.txt、text_port03.txt、与 text_port04.txt。如果 Port 2 的接受清单没有数据，那么就不会产生 text_port02.txt 这个档案。
- **Update Firmare:** 更新 CAN 板卡的韧体。此功能只适用于 PISO-CM100/U、 PISO-CPM100/U 及 PISO-DNM100/U，而不能用于 PISO-CAN、PEX-CAN 及 PCM-CAN。

Configuration 项目:

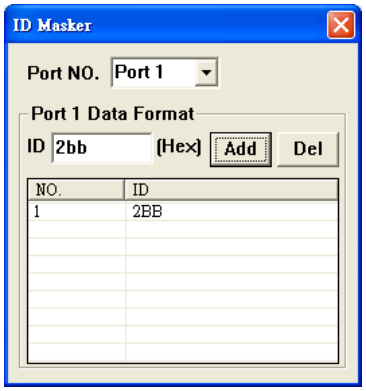


- **Board Configuration:** 使用者点击「Board Configuration」重新配置 CAN 板卡。详细信息请参考本节的“(1) CAN 配置对话框”。
- **Data Format:** 使用者可以对特定 ID 设定显示指定的格式 (如十六进制、十进制或 ASCII)，设定对话框如下图所示。例如，以“ID=0x1AA”CAN 讯息的 Byte2 到 Byte6 的数据格式设定为十进制。然后，接受清单就会以十进制显示“ID=0x1AA”的 Byte2 到

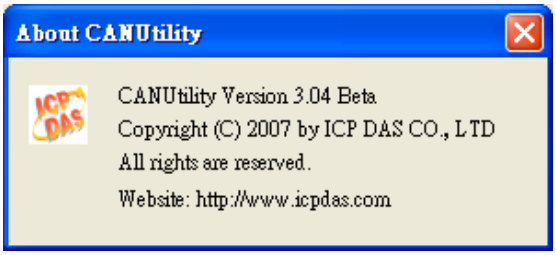
byte 6 数据格式，而其它字节以十六进制表示。若有讯息没配置数据格式，将一律以十六进制表示。在此对话框中，使用者最多可配置 20 组不同的 ID 讯息。



- **Software ID Mask:** 如果不想显示接受表中的特定 ID 讯息，可使用此屏蔽功能。如下图所示，在 ID 屏蔽清单中，使用者最多可设定 20 笔不同 ID 讯息。设定完成之后，如果 CAN 通讯端口接收的讯息与 ID 屏蔽清单中所设定的一样，则 CAN 讯息不会显示在接受清单中。



About: 显示关于 CANUtility 工具软件的版本及泓格科技网址的信息。



7 附录

7.1 接受滤波器

四个 8 位接受缓存器(AC0、AC1、AC2 及 AC3)与接受屏蔽缓存器(AM0、AM1、AM2 及 AM3)用于各种息讯的滤波器。这些缓存器用于控制 4 字节的滤波器，它可以检查 CAN 讯息的特定位并且决定 CAN 板卡是否接收该讯息。讯息滤波器的基本概念如图 A.1 所示。接受码缓存器主要用于决定 CAN 可以接受何种 ID 讯息，而接受屏蔽缓存器主要用于决定 ID 讯息中，哪些位须使用接受码缓存器来检查。如果接受屏蔽中任一位设为“0”，表示 ID 讯息中相同位置的位需要检查。

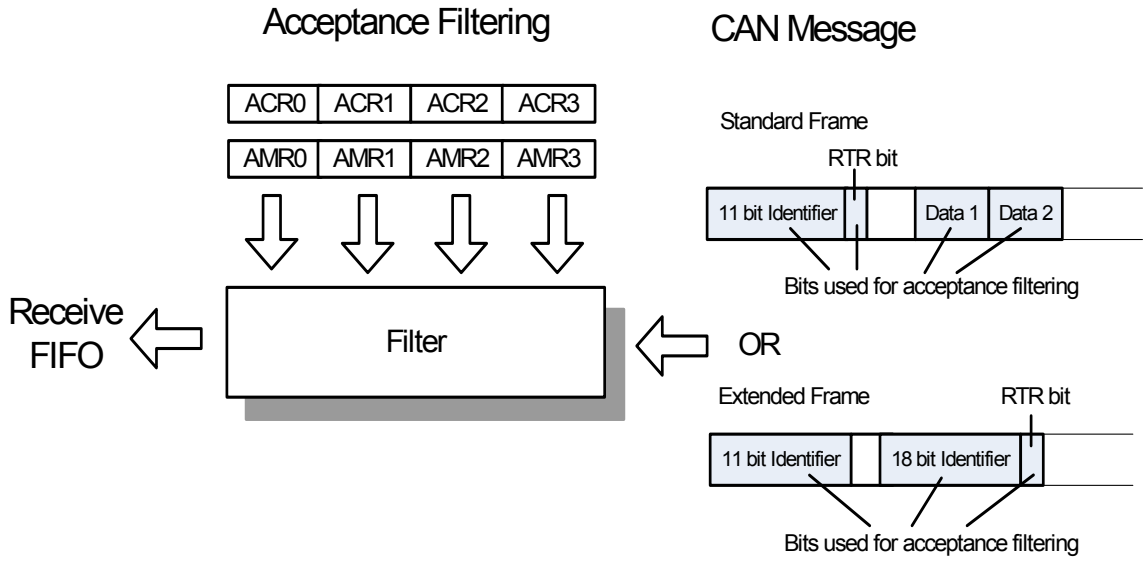


图 A.1 接受滤波器

Example 1:

假设有一个标准帧(**Standard Frame**)的讯息，其接受码缓存器(ACRn) 和接受屏蔽缓存器(AMRn)设定如下。

n	0	1 (较高 4 位)	2	3
ACRn	01xx x010	xxxx	xxxx xxxx	xxxx xxxx
AMRn	0011 1000	1111	1111 1111	1111 1111
接受讯息 (ID.28..ID.18 RTR)	01xx x010 xxxx			

("x"=忽略, 只有 ACR1 与 AMR1 较高的 4 位被使用)

在这个例子中，ACR0 和 AMR0 用于讯息 ID 中最高的 8 位；ACR1 和 AMR1 中较高的 4 位用于 ID 中较低的 3 位以及 RTR 位；ACR1 和 AMR1 中较低的 4 位未使用；ACR2 和 AMR2 用于 CAN 讯息的第一个字节；ACR3 和 AMR3 用于 CAN 讯息的第二个字节。因此，此 CAN 讯息不管是否为远程传输请求讯息，只要该讯息 ID 的格式为“01xx x010 xxx”就会被接受。(x 代表“忽略”)

Example 2:

假设有一个**延伸帧(Extended Frame)**的讯息，其接受码缓存器(ACRn)和接受屏蔽缓存器(AMRn)设定如下。

n	0	1	2	3(较高 6 位)
ACRn	1011 0100	1011 000x	1100 xxxx	0011 0xxx
AMRn	0000 0000	0000 0001	0000 1111	0000 0111
接受讯息 (ID.28..ID.0 RTR)	1011 0100	1011 000x	1100 xxxx	0011 0x

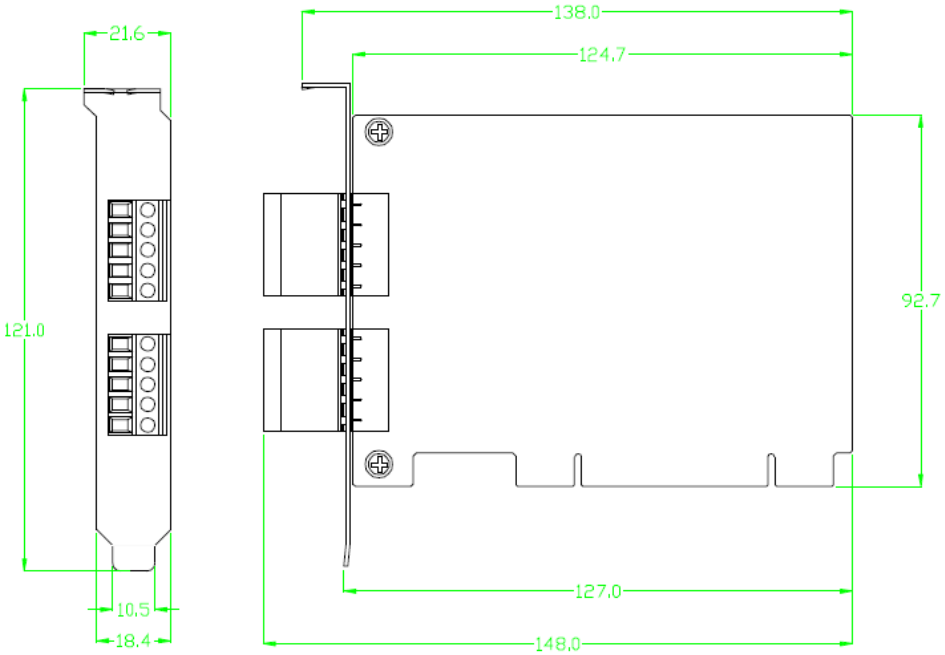
(”x”=忽略，只有 ACR3 和 AMR3 较高的 6 位被使用)

在这个例子中，AMR3 和 AMR3 较低的 2 位未使用。接受码与接受屏蔽的其它位，全部用于 29 位的讯息 ID 及 RTR 位。因此，不管 CAN 讯息是否为 RTR(远程传输请求)，只要讯息 ID 格式如 “1011 0100 1011 000x 1100 xxxx 0011 0x ” (x 代表“忽略”)就会被接受。

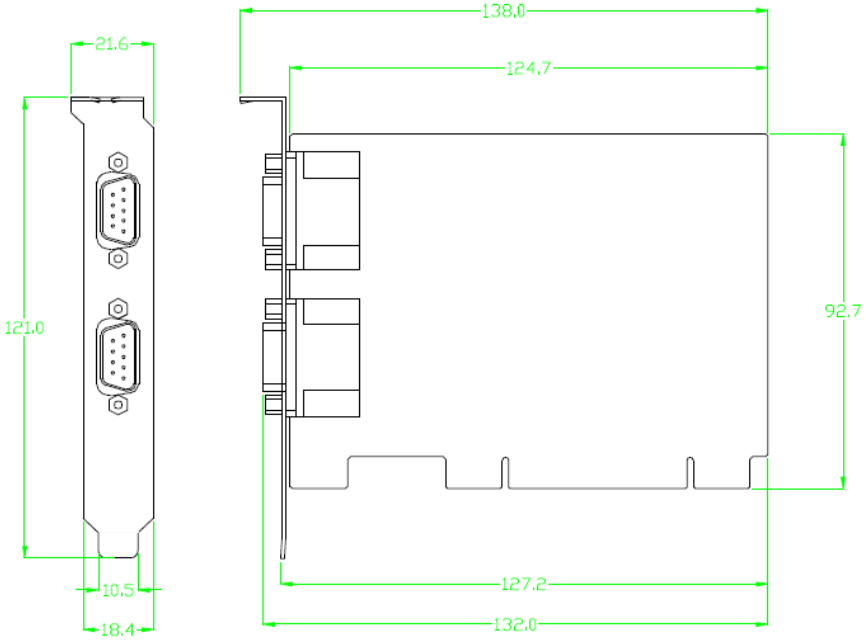
8 尺寸图

8.1 PISO-CAN200/400

PISO-CAN200/400-D/T



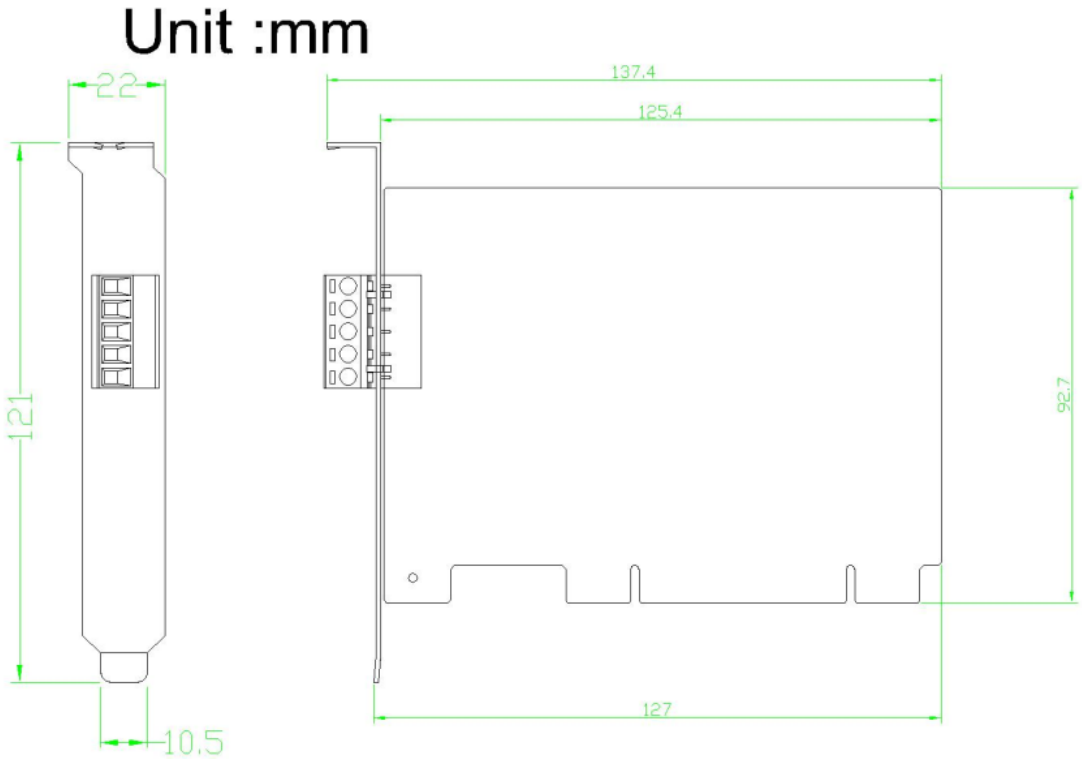
PISO-CAN200/400-T



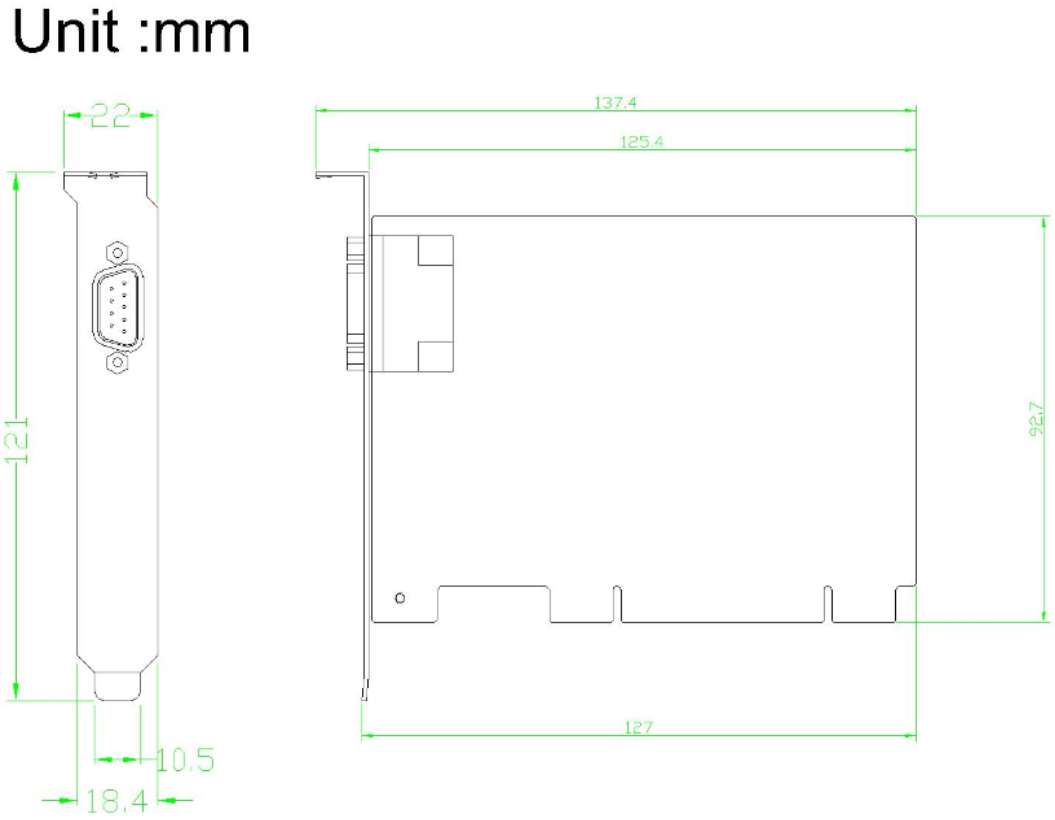
PISO-CAN200/400-D

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

PISO-CAN100U-D/T

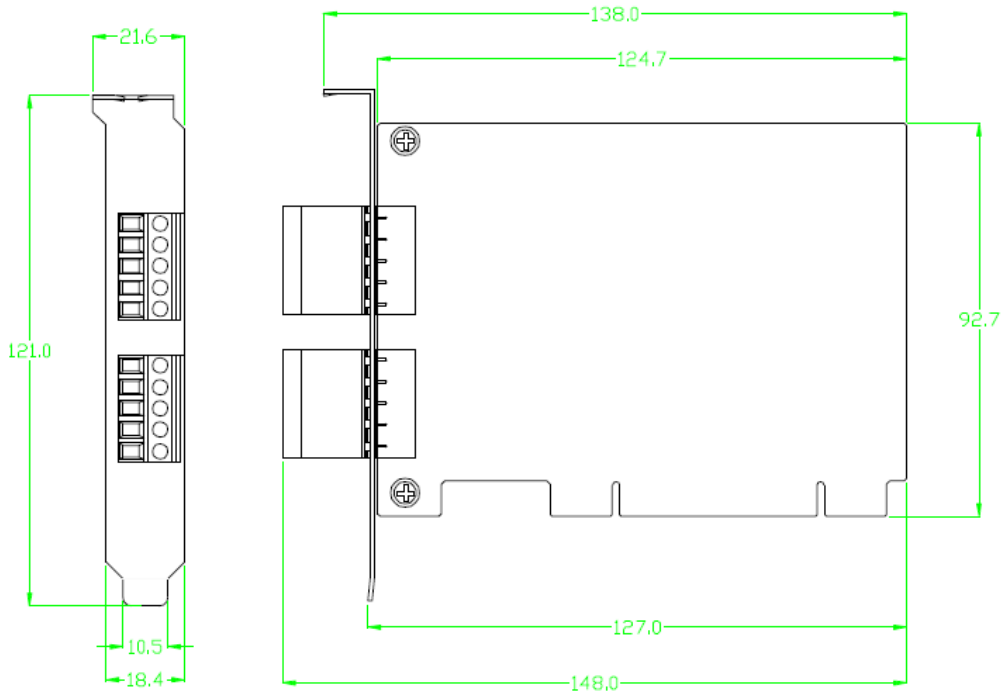


PISO-CAN100U-T

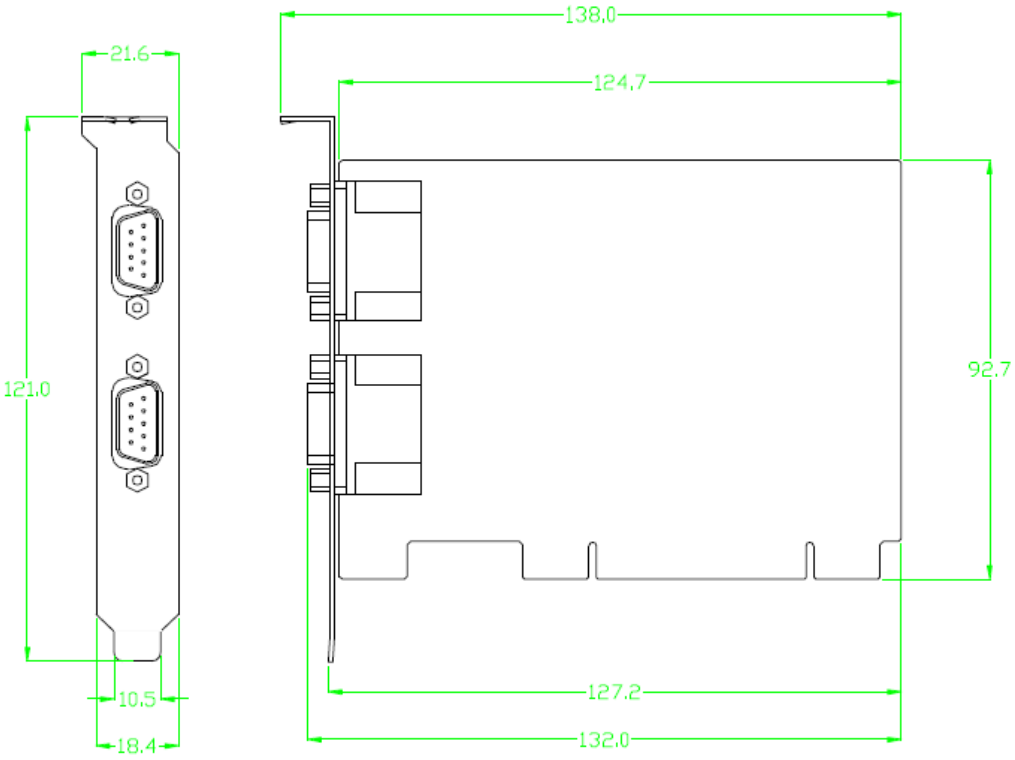


PISO-CAN100U-D

PISO-CAN200U/400U-D/T

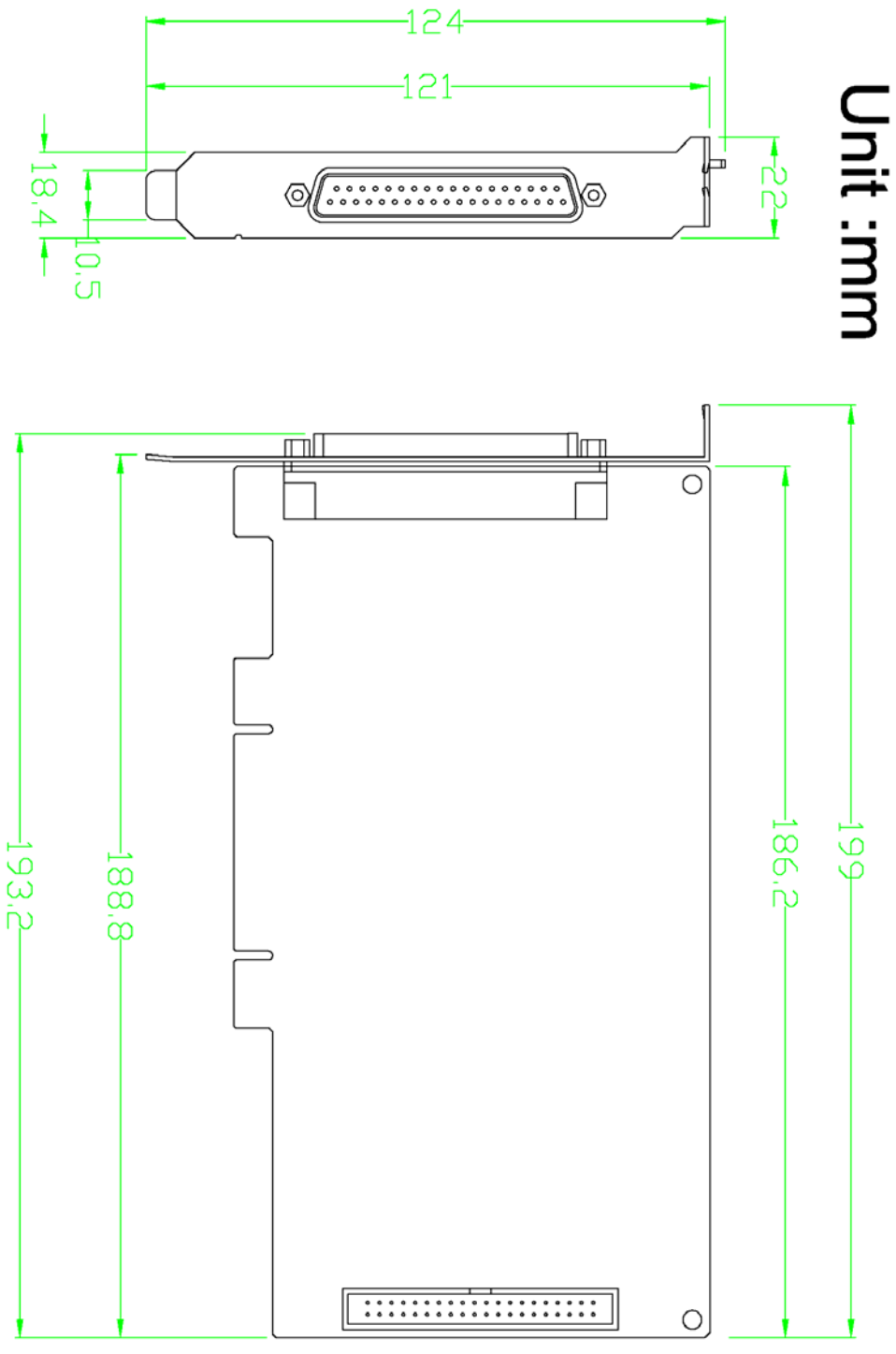


PISO-CAN200U/400U-T



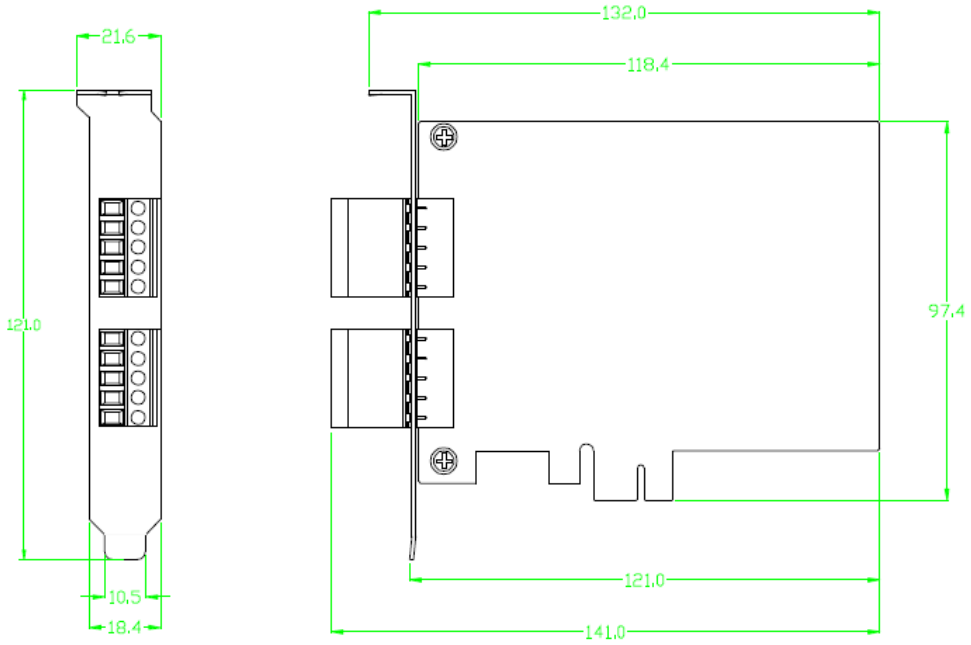
PISO-CAN200U/400U-D

PISO-CAN800U-D/T

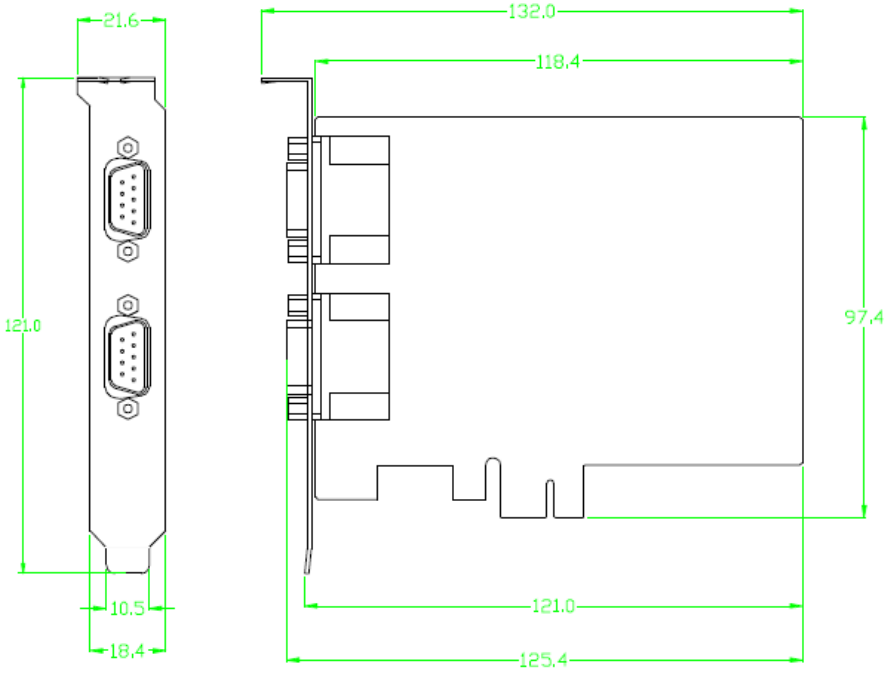


8.3 PEX-CAN200i

PEX-CAN200i-D/T



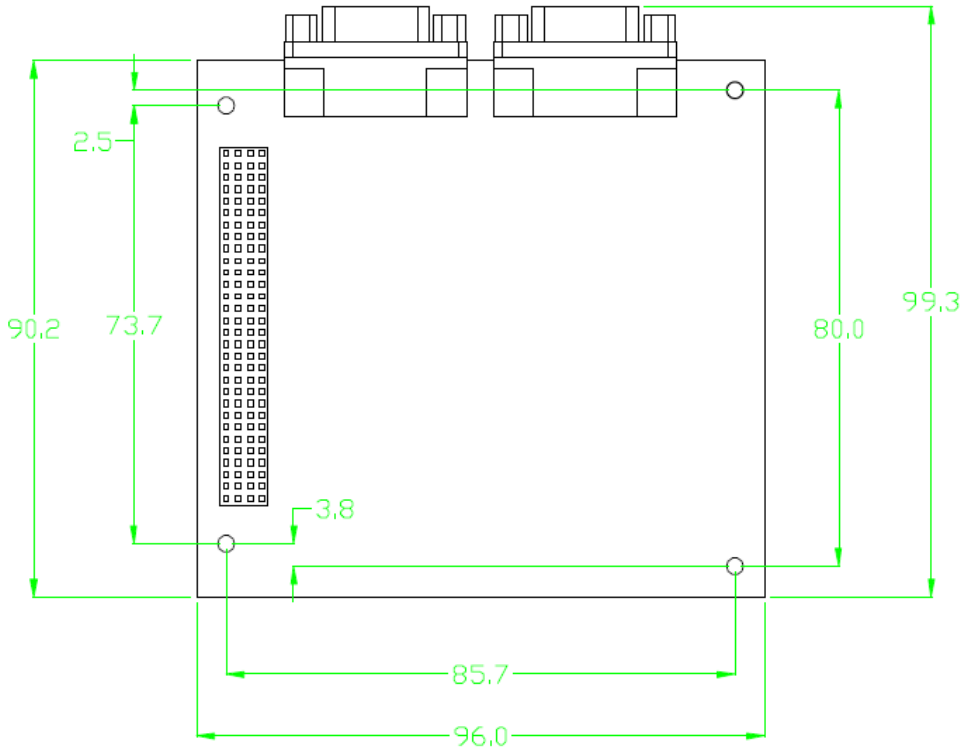
PEX-CAN200i-T



PEX-CAN200i-D

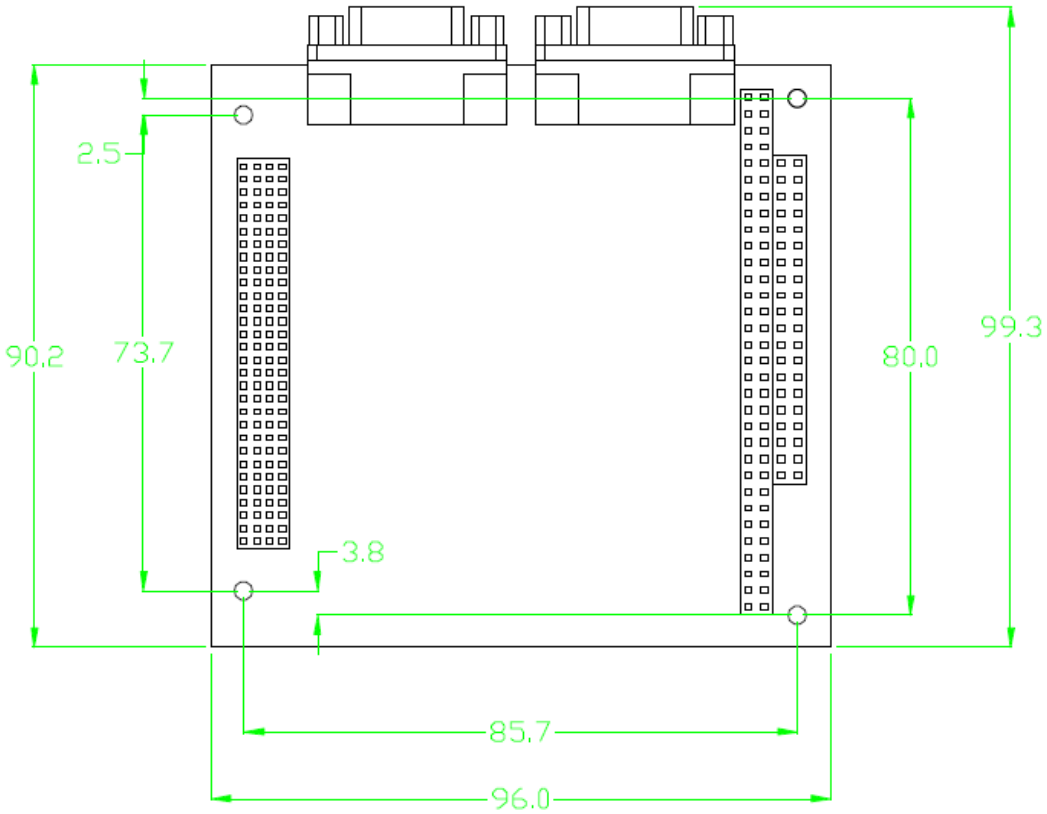
8.4 PCM-CAN100/200

PCM-CAN100/200-D



Unit :mm

PCM-CAN200P-D



Unit :mm