

PISO-CM100U-D/T

PEX-CM100-D/T

PCM-CM100-D

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 2007 by ICP DAS. All rights are reserved.

Trademark

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

Tables of Content

1	General Information.....	6
1.1	<i>Introduction.....</i>	<i>6</i>
1.2	<i>Features.....</i>	<i>7</i>
1.3	<i>Specifications</i>	<i>8</i>
1.4	<i>Product Check List</i>	<i>10</i>
2	Hardware Configuration.....	11
2.1	<i>Board Layout.....</i>	<i>11</i>
2.2	<i>Jumper Selection.....</i>	<i>14</i>
2.3	<i>Connector Pin Assignment.....</i>	<i>17</i>
2.3.1	<i>5-pin screw terminal connector</i>	<i>17</i>
2.3.2	<i>9-pin D-sub male connectors</i>	<i>18</i>
2.3.3	<i>Wire connection</i>	<i>19</i>
2.4	<i>LED Indicator & Operation Mode.....</i>	<i>20</i>
2.5	<i>Hardware Installation.....</i>	<i>21</i>
3	Driver Introduction	23
3.1	<i>Software Installation.....</i>	<i>23</i>
3.2	<i>Software Architecture.....</i>	<i>27</i>
4	APIs for the PC's Program	30
4.1	<i>Windows API Definitions and Descriptions.....</i>	<i>30</i>
4.1.1	<i>CM100_GetDllVersion</i>	<i>34</i>
4.1.2	<i>CM100_GetBoardInf</i>	<i>35</i>
4.1.3	<i>CM100_TotalBoard</i>	<i>36</i>
4.1.4	<i>CM100_TotalCM100Board</i>	<i>36</i>
4.1.5	<i>CM100_TotalDNM100Board.....</i>	<i>37</i>
4.1.6	<i>CM100_TotalCPM100Board</i>	<i>37</i>
4.1.7	<i>CM100_GetCM100BoardSwitchNo.....</i>	<i>38</i>
4.1.8	<i>CM100_GetDNM100BoardSwitchNo</i>	<i>39</i>
4.1.9	<i>CM100_GetCPM100BoardSwitchNo</i>	<i>40</i>
4.1.10	<i>CM100_GetCardPortNum</i>	<i>41</i>
4.1.11	<i>CM100_ActiveBoard.....</i>	<i>42</i>
4.1.12	<i>CM100_CloseBoard.....</i>	<i>43</i>
4.1.13	<i>CM100_BoardIsActive.....</i>	<i>44</i>
4.1.14	<i>CM100_AdujstDateTime.....</i>	<i>45</i>
4.1.15	<i>CM100_Reset.....</i>	<i>46</i>
4.1.16	<i>CM100_Init</i>	<i>47</i>
4.1.17	<i>CM100_HardwareReset.....</i>	<i>48</i>

4.1.18	CM100_Check186Mode	49
4.1.19	CM100_Status	50
4.1.20	CM100_AddCyclicTxMsg	52
4.1.21	CM100_DeleteCyclicTxMsg	54
4.1.22	CM100_EnableCyclicTxMsg	55
4.1.23	CM100_DisableCyclicTxMsg	56
4.1.24	CM100_OutputByte	57
4.1.25	CM100_InputByte	58
4.1.26	CM100_IsTxTimeout	59
4.1.27	CM100_SetSystemMsg	60
4.1.28	CM100_ClearSoftBuffer <For default firmware>.....	61
4.1.29	CM100_ClearBufferStatus <For default firmware>.....	62
4.1.30	CM100_ClearDataOverrun <For default firmware>	63
4.1.31	CM100_Config <For default firmware>.....	64
4.1.32	CM100_ConfigWithoutStruct <For default firmware>	67
4.1.33	CM100_RxMsgCount <For default firmware>	68
4.1.34	CM100_ReceiveMsg <For default firmware>	69
4.1.35	CM100_ReceiveWithoutStruct <For default firmware>	71
4.1.36	CM100_SendMsg <For default firmware>	73
4.1.37	CM100_SendWithoutStruct <For default firmware>.....	74
4.1.38	CM100_SJA1000Config <For user-defined firmware>	76
4.1.39	CM100_EnableSJA1000 <For user-defined firmware>	77
4.1.40	CM100_DisableSJA1000 <For user-defined firmware>	78
4.1.41	CM100_DPRAMIntttToCM100 <For user-defined firmware>	79
4.1.42	CM100_DPRAMWriteByte <For user-defined firmware>.....	80
4.1.43	CM100_DPRAMWriteWord <For user-defined firmware>	81
4.1.44	CM100_DPRAMWriteDword <For user-defined firmware>	82
4.1.45	CM100_DPRAMWriteMultiByte <For user-defined firmware>	83
4.1.46	CM100_DPRAMReadByte <For user-defined firmware>	84
4.1.47	CM100_DPRAMReadWord <For user-defined firmware>.....	85
4.1.48	CM100_DPRAMReadDword <For user-defined firmware>.....	86
4.1.49	CM100_DPRAMReadMultiByte <For user-defined firmware>.....	87
4.1.50	CM100_DPRAMMemset <For user-defined firmware>	88
4.1.51	CM100_ReceiveCmd <For user-defined firmware>	89
4.1.52	CM100_SendCmd <For user-defined firmware>.....	91
4.1.53	CM100_InstallUserISR <For user-defined firmware>	92
4.1.54	CM100_RemoveUserISR <For user-defined firmware>.....	93
4.2	Windows API Return Codes Troubleshooting	94

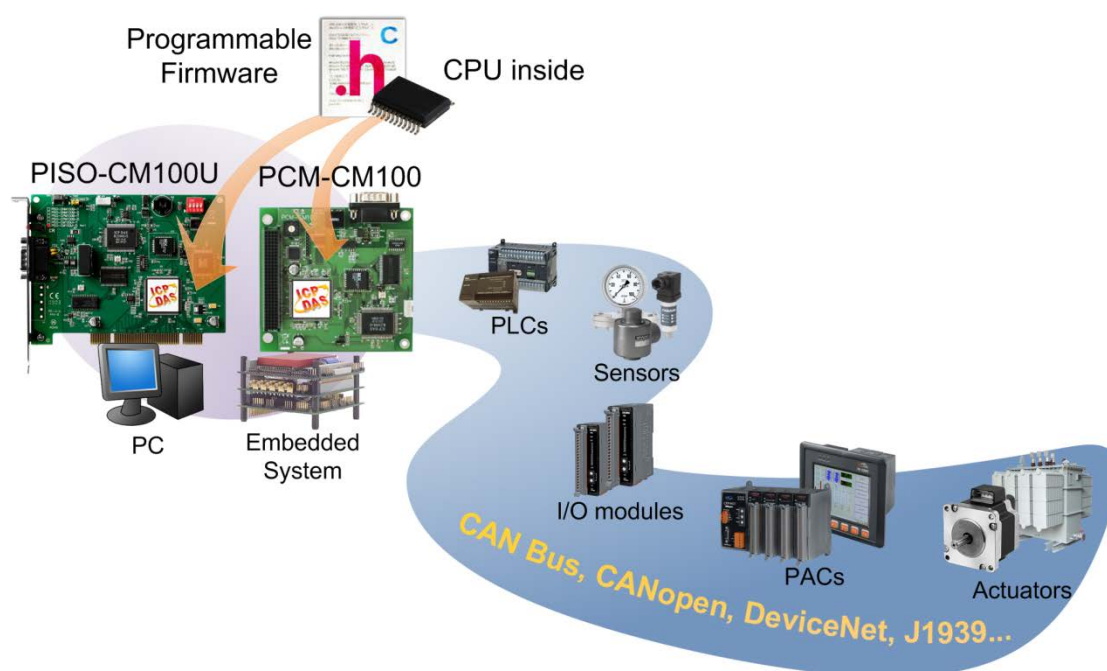
5	Functions of Firmware Library	96
5.1	Firmware Library Definitions and Descriptions	96
5.1.1	L1Off.....	100
5.1.2	L1On	100
5.1.3	L2Off.....	101
5.1.4	L2On	101
5.1.5	DPRAMInttToHost	102
5.1.6	UserDPRAMIrqFunc <must be called once >.....	103
5.1.7	DPRAMWriteByte	104
5.1.8	DPRAMWriteWord	105
5.1.9	DPRAMWriteDword	106
5.1.10	DPRAMWriteMultiByte	107
5.1.11	DPRAMReadByte.....	108
5.1.12	DPRAMReadWord	109
5.1.13	DPRAMReadDword	110
5.1.14	DPRAMReadMultiByte	111
5.1.15	DPRAMMemset.....	112
5.1.16	DPRAMReceiveCmd.....	113
5.1.17	DPRAMSendCmd	114
5.1.18	DebugPrint <assist with CM100_DEBUG_MONITOR.EXE>.....	115
5.1.19	GetKbhit <assist with debug cable and 7188xw.exe>	116
5.1.20	Print <assist with debug cable and 7188xw.exe>	117
5.1.21	GetTime	118
5.1.22	SetTime	119
5.1.23	GetDate	120
5.1.24	SetDate	121
5.1.25	GetWeekDay	122
5.1.26	ReadNVRAM	123
5.1.27	WriteNVRAM	124
5.1.28	GetTimeTicks100us.....	125
5.1.29	GetTimeTicks	126
5.1.30	DelayMs.....	127
5.1.31	CM100_InstallUserTimer	128
5.1.32	T_StopWatchXXX series functions	129
5.1.33	T_CountDownTimerXXX series functions.....	131
5.1.34	CM100_EEPROMReadByte.....	133
5.1.35	CM100_EEPROMReadMultiByte	134
5.1.36	CM100_EEPROMWriteByte.....	135

5.1.37	CM100_EEPROMWriteMultiByte	136
5.1.38	UserCANIrqFunc <must be called once>	137
5.1.39	SJA1000HardwareReset	138
5.1.40	SetCANBaud	139
5.1.41	GetCANBaud	140
5.1.42	SetCANMask	141
5.1.43	GetCANMask	143
5.1.44	CANConfig	144
5.1.45	CANConfigBySJA1000Reg	145
5.1.46	EnableSJA1000	146
5.1.47	DisableSJA1000	146
5.1.48	GetCANStatus	147
5.1.49	ClearDataOverrunStatus	148
5.1.50	SendCANMsg	149
5.1.51	ClearTxSoftBuffer	150
5.1.52	GetCANMsg	151
5.1.53	ClearRxSoftBuffer	153
5.1.54	RxMsgCount	153
5.1.55	ChcekTxStatus	154
5.1.56	AddCyclicTxMsg	155
5.1.57	DeleteCyclicTxMsg	157
5.1.58	EnableCyclicTxMsg	158
5.1.59	EnableCyclicTxMsgWithTimes	159
5.1.60	GetRestCyclicTxCnt	160
5.1.61	DisableCyclicTxMsg	161
5.1.62	ResetCyclicTxBuf	161
5.1.63	SystemHardwareReset	162
5.1.64	SystemInit	162
5.1.65	GetLibVer	163
5.1.66	RefreshWDT	163
5.1.67	UserInitFunc <must be called once>	164
5.1.68	UserLoopFunc <must be called once>	165
5.2	Firmware Library Return Codes Troubleshooting	166
6	Application Programming	167
6.1	Windows Programming With Default Firmware	167
6.2	Introduction of CANUtility Tool	176
6.3	Debug Tools for User-defined Firmware Programming	188
6.4	User-defined Firmware Programming	193

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, the PISO-CM100U/PEX-CM100/PCM-CM100 (CM100s board) represents a powerful and economic solution. It has an internal 80186 compactable CPU for the complex protocol interpretations and implementations. Owing to the real-time DOS-like OS, MiniOS7, the CM100s board can cover most of all time-critical CAN-based applications, such as self-define CAN protocol, CANopen, DeviceNet, J1939, and so forth. Therefore, when users develop their projects, the CM100s board is helpful to handle the process of the CAN messages, and share the CPU loading of the PC or embedded system. Besides, the CM100s board allows users designing the firmware of the CM100s board. Through the library and demos, it is easy to finish the user-defined firmware to satisfy the users' requirements.



1.2 Features

- Follow ISO11898-2 specification
- 2500Vrms photo-isolation protection on CAN side
- Jumper select 120 Ω terminator resistor for CAN bus
- One CAN communication port
- Compatible with CAN specification 2.0 parts A and B
- Provide default baud 10Kbps, 20Kbps, 50Kbps, 125Kbps, 250Kbps, 500Kbps, 800Kbps, and 1Mbps
- Allow user-defined baud
- 2048 records reception buffer and 256 records transmission buffer
- Cyclic transmission precision: $\pm 0.5\text{ms}$ precision when cyclic time is under 10ms , $\pm 1\%$ error when cyclic time exceeds 10ms.
- Provide 5 sets of cyclic transmission.
- Timestamp of CAN message with at least $\pm 1\text{ms}$ precision
- 186 compactable CPU inside
- Arrange the inside DPRAM flexibly
- RTC(Real Time Clock) inside
- Red and green LED indicators
- Support the default firmware or the user-defined firmware
- Support firmware update
- VC++, VB, BCB demos and libraries are given
- C/C++ function libraries of firmware side is given

1.3 Specifications

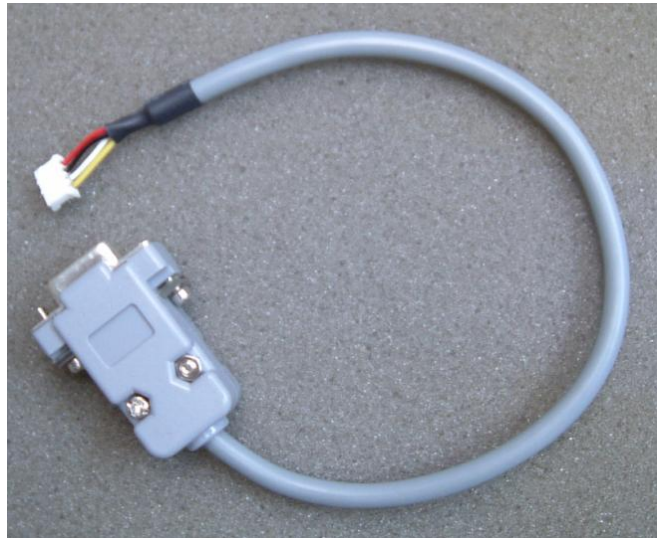
Model Name	PISO-CM100U	PEX-CM100	PCM-CM100
Hardware			
CPU	80186, 80 MHz or compatible		
SRAM	512 KB		
Flash	512 KB (128 KB for system, 384 KB for users' applications), 64 KB for one sector (erase unit), 100,000 erase/write cycles		
EEPROM	16 KB (1 KB for system information, 15 KB for users' applications), 40-year data retention, 1 million erase/write cycles		
DPRAM	8 KB (1 kB for system, others for users' applications)		
NVRAM	31 bytes (battery backup, data valid for up to 10 years)		
RTC (Real Time Clock)	Seconds, minutes, hours, date of week, date of month, month and year, valid from 1980 to 2079		
Bus Interface			
Type	Universal PCI	PCI Express bus	PCI-104
Board No.	By DIP switch		By rotary switch
CAN Interface			
Controller	NXP SJA1000T with 16 MHz clock		
Transceiver	NXP 82C250		
Channel number	1		
Connector	5-pin Screwed Terminal Connector / 9-pin male D-Sub Connector		9-pin male D-Sub
Baud Rate (bps)	10 k, 20 k, 50 k, 125 k, 250 k, 500 k, 800 k, 1 M (allow user-defined baud rate)		
Isolation	3000 Vdc for DC-to-DC, 2500 Vrms for photo-couple		
Terminator Resistor	Jumper for 120 Ω terminator resistor		
LED			
LED Indicator	Green LED, Red LED (In default firmware: Green for Tx/Rx, Red for Err)		
Power			
Power Consumption	300 mA @ 5 V		
Software			
Driver	Windows 2K / XP / 7 / 10 (x32 & x64)		
Library	VC++, VB.net, C#.net for PC, TC1.01, BC3.1 for Firmware		

Mechanism			
Dimensions (W x L)	138mm x 97mm for v1.xx 119mm x 97mm for v2.xx	133mm x 98mm	90mm x 96mm
Environment			
Operating Temp.	0 ~ 60 °C		
Storage Temp.	-20 ~ 70 °C		
Humidity	5 ~ 85% RH, non-condensing		

1.4 Product Check List

Besides this manual, the package includes the following items:

- ☐ CM100s board CAN card
- ☐ Software CD ROM
- ☐ Quickstart
- ☐ One debug cable (model number is 4PCA-0904)



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

- ☐ Where you can find the software driver, utility and demo programs.
- ☐ How to install software & utility.
- ☐ How to program users' applications with CM100s board.
- ☐ The definitions of function library, error code, LED status, and pin assignment.
- ☐ The basic solution of troubleshooting.

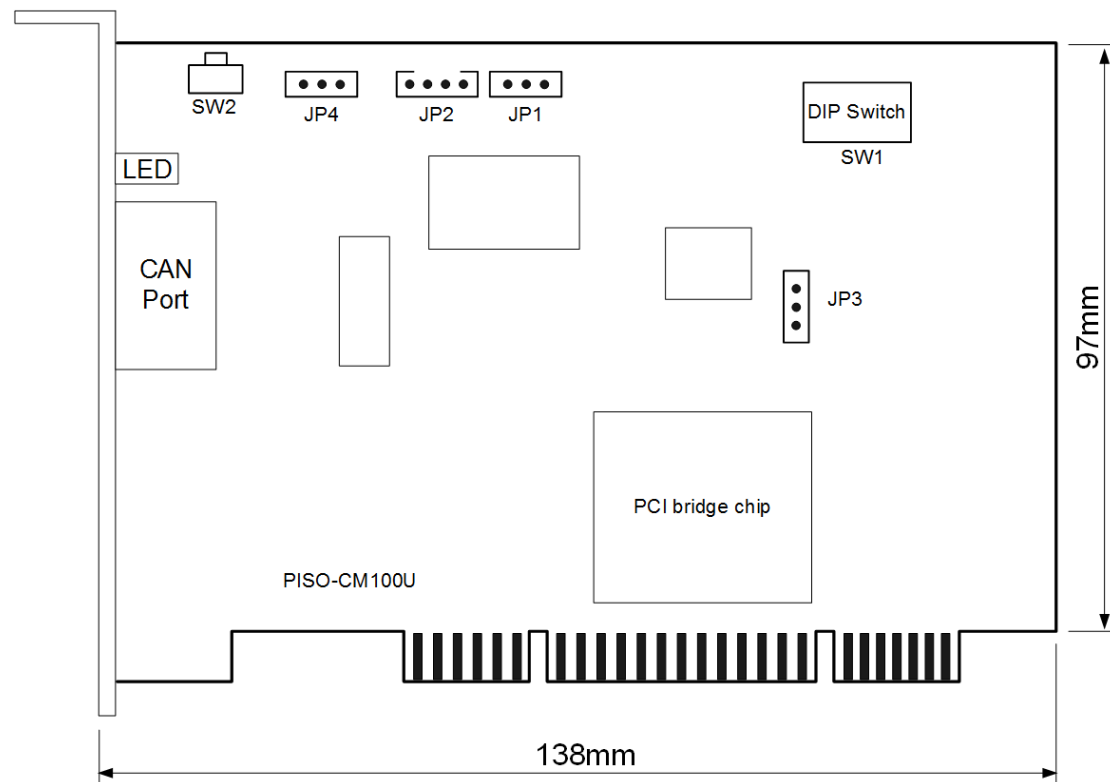
Attention !

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.

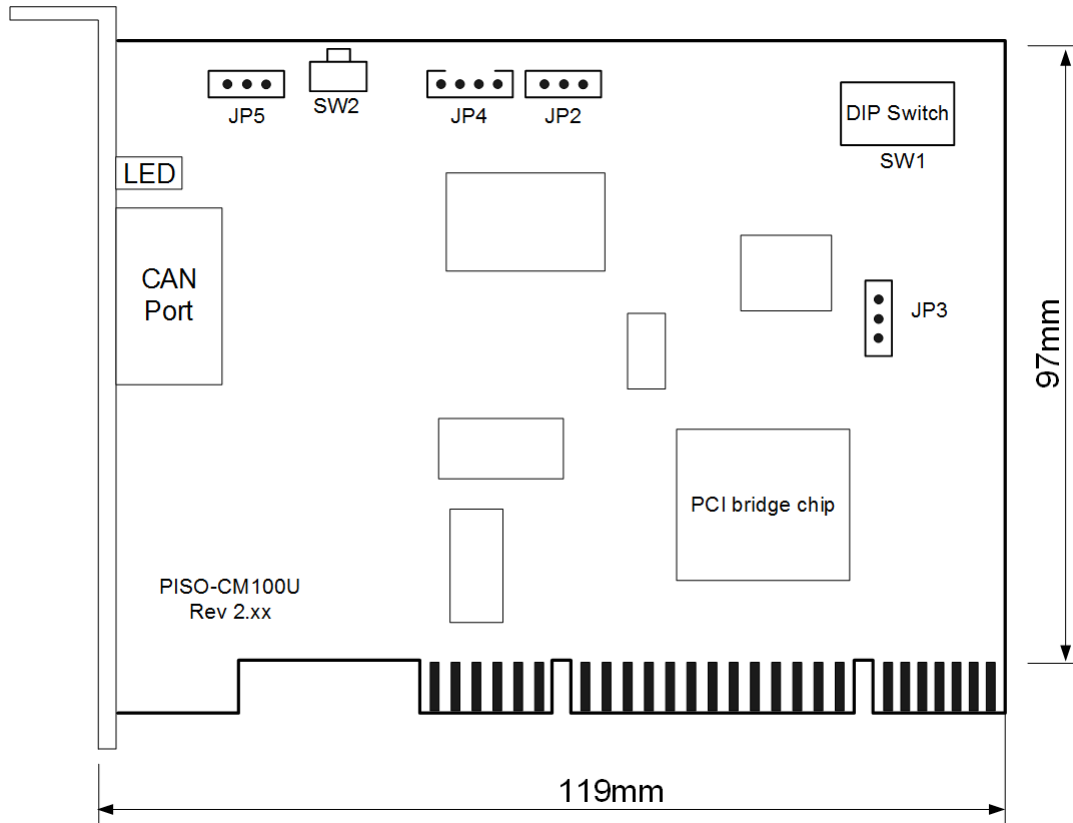
2 Hardware Configuration

This section describes the hardware settings of the PISO-CM100U-D/T, PEX-CM100-D/T and PCM-CM100-D. This information includes the wire connection and terminal resistance configuration for the CAN network. The PISO-CM100U-T/PEX-CM100-T layout is similar with the PISO-CM100U-D/PEX-CM100-D. The only difference is the position of CAN port connector.

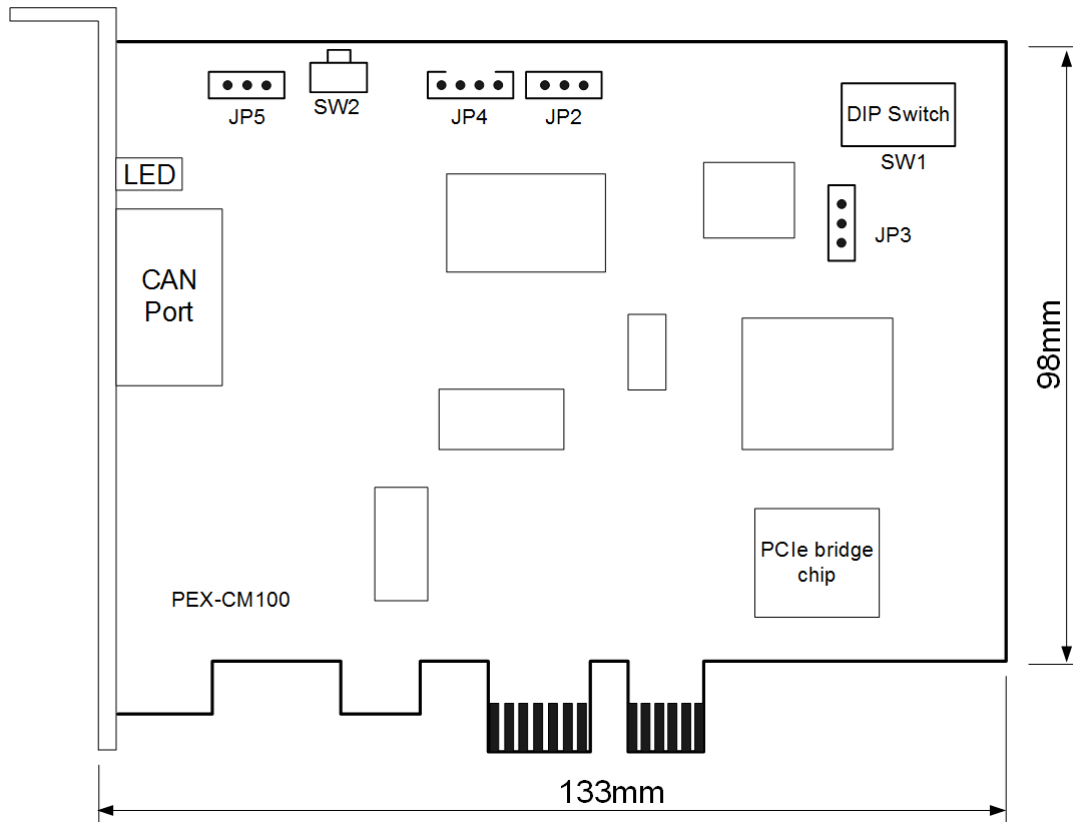
2.1 Board Layout



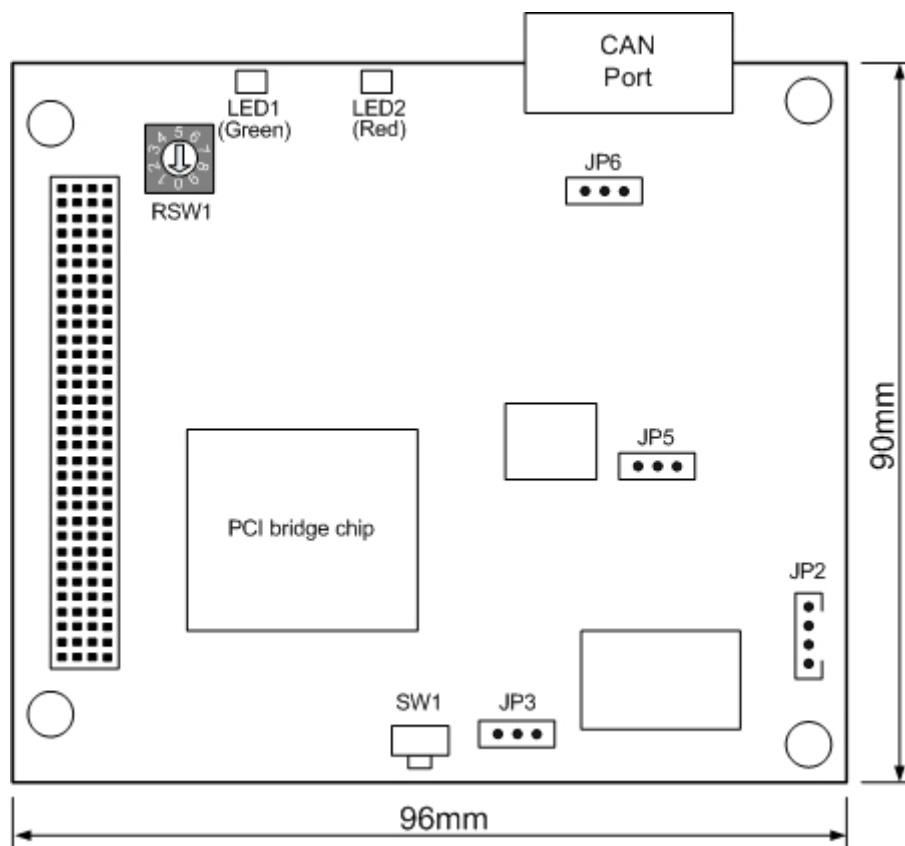
PISO-CM100U-D board layout for PCB Rev 1.xx



PISO-CM100U-D board layout for PCB Rev 2.xx



PEX-CM100-D board layout



PCM-CM100-D board layout

2.2 Jumper Selection

The following table shows the definition of jumpers or DIP switch. Users need to refer to the following table to configure the PISO-CM100U-D/T hardware. Note that, there are two layouts (Rev1.xx and Rev2.xx) for PISO-CM100U and their jumpers are some different. The different jumpers will be indicated in brackets.

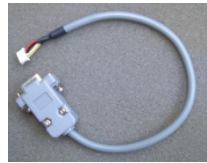
PISO-CM100U-D/T (Rev 1.xx) :

Jumper	Description	Status
JP2	The debug port for the user-defined firmware. Users can connect the debug port with the PC RS-232 port via the debug cable.	 4-pin connector for Jump D-Sub 9 pin connector for PC RS-232 port
JP3	The Flash protection jumper. If users would like to protect the data of the Flash. Enable this jumper. In this case, the firmware can't be updated from the utility.	JP3  Enable JP3  Disable
JP4	The 120Ω terminal resistance of the CAN bus.	JP4  Enable JP4  Disable
SW2	The reset button for into the download mode. If users would like force the CAN board into download mode, enable this jumper to reset the firmware and let the PISO-CM100U into the download mode.	 SW2
DIP switch	The DIP switch for the PISO-CM100U board No. If the left-hand-side switch (bit No. 1) is ON, the board No. is set to 1. The range of the board No. is from 0 to 15. Be careful that the board No. of each CM100s, DNM100s and CPM100s in your PC must be unique.	DIP switch  This situation indicates the board No. 1.

PISO-CM100U-D/T (Rev 2.xx) and PEX-CM100 :

Jumper	Description	Status
JP4	The debug port for the user-defined firmware. Users can connect the debug port with the PC RS-232 port via the debug cable.	 4-pin connector for Jump D-Sub 9 pin connector for PC RS-232 port
JP3	The Flash protection jumper. If users would like to protect the data of the Flash. Enable this jumper. In this case, the firmware can't be updated from the utility.	JP3  Enable JP3  Disable
JP5	The 120Ω terminal resistance of the CAN bus.	JP4  Enable JP4  Disable
SW2	The reset button for into the download mode. If users would like force the CAN board into download mode, enable this jumper to reset the firmware and let the CM100s board into the download mode.	 SW2
DIP switch	The DIP switch for the CM100s board No. If the left-hand-side switch (bit No. 1) is ON, the board No. is set to 1. The range of the board No. is from 0 to 15. Be careful that the board No. of each CM100s, DNM100s and CPM100s in your PC must be unique.	DIP switch  This situation indicates the board No. 1.

PCM-CM100-D :

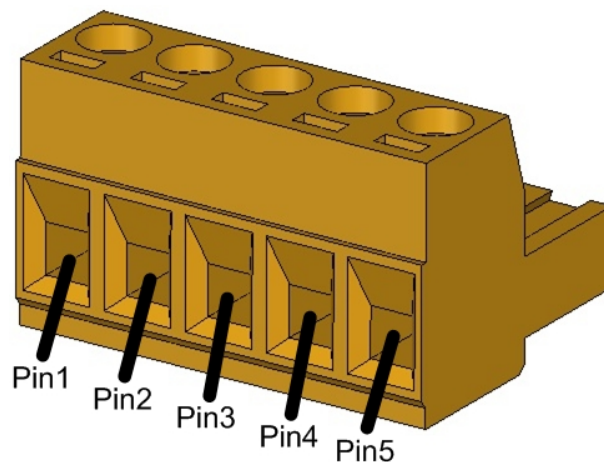
Jumper	Description	Status
JP2	The debug port for the user-defined firmware. Users can connect the debug port with the PC RS-232 port via the debug cable.	 4-pin connector for JP2 D-Sub 9 pin connector for PC RS-232 port
JP5	The Flash protection jumper. If users would like to protect the data of the Flash. Enable this jumper. In this case, the firmware can't be updated from the utility.	JP5  Enable JP5  Disable
JP6	The 120Ω terminal resistance of the CAN bus.	JP6  Enable JP6  Disable
SW1	The reset button for into the download mode. If users would like force the CAN board into download mode, enable this jumper to reset the firmware and let the PCM-CM100 into the download mode.	 SW1
RSW1	The rotary switch for the PCM-CM100 board No. and interrupt line. There are only 4 interrupt lines for the PCI-104 architecture. So, only the value 0 ~ 3 are useful. If users set the rotary switch to 4 ~ 9, the board No will be 4 ~ 9, but the interrupt line is still one of the 0 ~ 3.	 This situation indicates the board No. 0.

2.3 Connector Pin Assignment

The PISO-CM100U-T is equipped with one **5-pin screw terminal connector** and the PISO-CM100U-D/PCM-CM100-D is equipped with one **9-pin D-sub male connector** for wire connection of the CAN bus. The connector's pin assignment is specified as following:

2.3.1 5-pin screw terminal connector

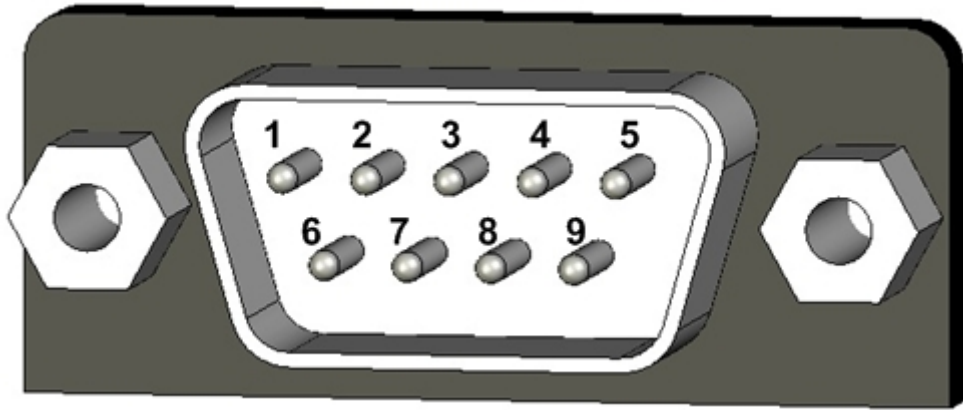
The 5-pin screw terminal connector of the CAN bus interface is shown in the following figure and table.



Pin No.	Signal	Description
1	CAN_GND	Ground
2	CAN_H	CAN_H bus line (dominant high)
3	CAN_SHLD	Optional CAN Shield
4	CAN_L	CAN_L bus line (dominant low)
5	N/A	No use

2.3.2 9-pin D-sub male connectors

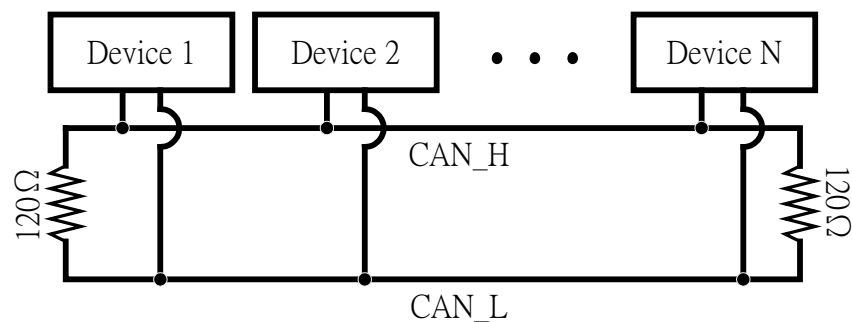
The 9-pin D-sub male connector of the CAN bus interface is shown in the following figure and table.



Pin No.	Signal	Description
1	N/A	No use
2	CAN_L	CAN_L bus line (dominant low)
3	CAN_GND	Ground
4	N/A	No use
5	CAN_SHLD	Optional CAN Shield
6	CAN_GND	Ground
7	CAN_H	CAN_H bus line (dominant high)
8	N/A	No use
9	N/A	No use

2.3.3 Wire connection

In order to minimize the reflection effects on the CAN bus line, the CAN bus line has to be terminated at both ends by two terminal resistances as in the following figure. According to the ISO 11898-2 spec, each terminal resistance is 120Ω (or between 108Ω~132Ω). The length related resistance should have 70 mΩ/m. Users should check the resistances of the CAN bus, before they install a new CAN network.



Moreover, to minimize the voltage drop over long distances, the terminal resistance should be higher than the value defined in the ISO 11898-2. The following table can be used as a good reference.

Bus Length (meter)	Bus Cable Parameters		Terminal Resistance (Ω)
	Length Related Resistance (mΩ/m)	Cross Section (Type)	
0~40	70	0.25(23AWG)~ 0.34mm ² (22AWG)	124 (0.1%)
40~300	< 60	0.34(22AWG)~ 0.6mm ² (20AWG)	127 (0.1%)
300~600	< 40	0.5~0.6mm ² (20AWG)	150~300
600~1K	< 20	0.75~0.8mm ² (18AWG)	150~300

2.4 LED Indicator & Operation Mode

The LED status will be changed when CM100s board is in the different modes. There are three modes, and each mode describes as follows:

1. Download mode: In this case, the green LED and red LED will be alternatively flashed once per second. (When the green LED is ON, the red LED is OFF, and vice versa.). The CM100s board is ready for updating the firmware from the Utility. Users can use the software to download the newer default firmware or the user-defined firmware.
2. Firmware mode: If the CM100s board uses default firmware, the green LED is flashed once as the CM100s board receives or transmits one CAN message to CAN bus successfully. If the bus loading is heavy, the green LED may be turned on always. If some error occurs, the red LED is turned on. Users can use `CM100_Status()` function to obtain the CAN control status except the buffer status. The buffer statuses are obtained from the return codes of the transition or reception functions as users read or send a CAN message. If the CM100s board runs the user-defined firmware, the actions of the green LED or red LED can be designed by the user-defined firmware.
3. Firmware reset mode: If users enable SW2 of the PISO-CM100U/ PEX-CM100 or the SW1 of the PCM-CM100 described in section 2.2, both the red and green LEDs will be turned on about 1 second, and the CM100s board is forced to enter the download mode. If the CM100Us board is out of control because of the bugs of the user-defined firmware or other problems, use this method to reset the firmware and download the newer firmware again.

Note: The color of the LED 1 and LED2 of the PCM-CM100 are green and red respectively.

2.5 Hardware Installation

When users want to use the PISO-CM100U Rev1.xx, the hardware installation needs to be finished as the following steps.

1. Shutdown your personal computer.
 2. Configure the DIP switch and the JP4 for the board No. and the terminal resistor of the PISO-CM100U. The more detail of the switch and jumper positions can be found at section 2.2.
 3. Check the JP3 of the PISO-CM100U. If necessary, enable it.
 4. Find an empty PCI slot for the PISO-CM100U on the mother board of the personal computer. Plug the configured PISO-CM100U into this empty PCI slot.
 5. Plug the CAN bus cable(s) into the 5-pin screw terminal connector or the 9-pin D-sub connector due to the model number of the PISO-CM100U.
- After the procedure described above is completed, turn on the PC.

The hardware installation for the PISO-CM100U Rev 2.xx or PEX-CM100 is shown as the following steps.

1. Shutdown your personal computer.
2. Configure the DIP switch and the JP5 for the board No. and the terminal resistor of the PISO-CM100U/PEX-CM100. The more detail of the switch and jumper positions can be found at section 2.2.
3. Check the JP3 of the PISO-CM100U/PEX-CM100. If necessary, enable it.
4. Find an empty PCI/PCIe slot for the PISO-CM100U/PEX-CM100 on the mother board of the personal computer. Plug the configured PISO-CM100U/PEX-CM100 into this empty PCI/PCIe slot.
5. Plug the CAN bus cable(s) into the 5-pin screw terminal connector or the 9-pin D-sub connector due to the model number of the PISO-CM100U/PEX-CM100.

After the procedure described above is completed, turn on the PC.

The hardware installation for the PCM-CM100 is shown as the following steps.

1. Shutdown your embedded system.
2. Configure the rotary switch and the JP6 for the board No. and the terminal resistor of the PCM-CM100. The more detail of the switch and jumper positions can be found at section 2.2.
3. Check the JP5 of the PCM-CM100. If necessary, enable it.
4. Plug the PCM-CM100 on the top of the PCI-104 slot of your embedded system. One embedded may have up to 4 PCM-CM100.
5. Plug the CAN bus cable(s) into the 9-pin D-sub connector.

After the procedure described above is completed, turn on the embedded system.

3 Driver Introduction

3.1 Software Installation

The CM100s board can be used in the Windows 2000/XP environments. Users need to get the proper driver for their operation system. These drivers are in the Field Bus CD in the CM100s board package. The path is CAN\PCI\PISO-CM100U. Also, users can find them from our website as following. (Take a note that the PISO-CM100U, PEX-CM100 and the PCM-CM100 use the same driver.)

ftp://ftp.icpdas.com/pub/cd/fieldbus_cd/can/pci/piso-cm100u/

The recommended installation procedure is given as below:

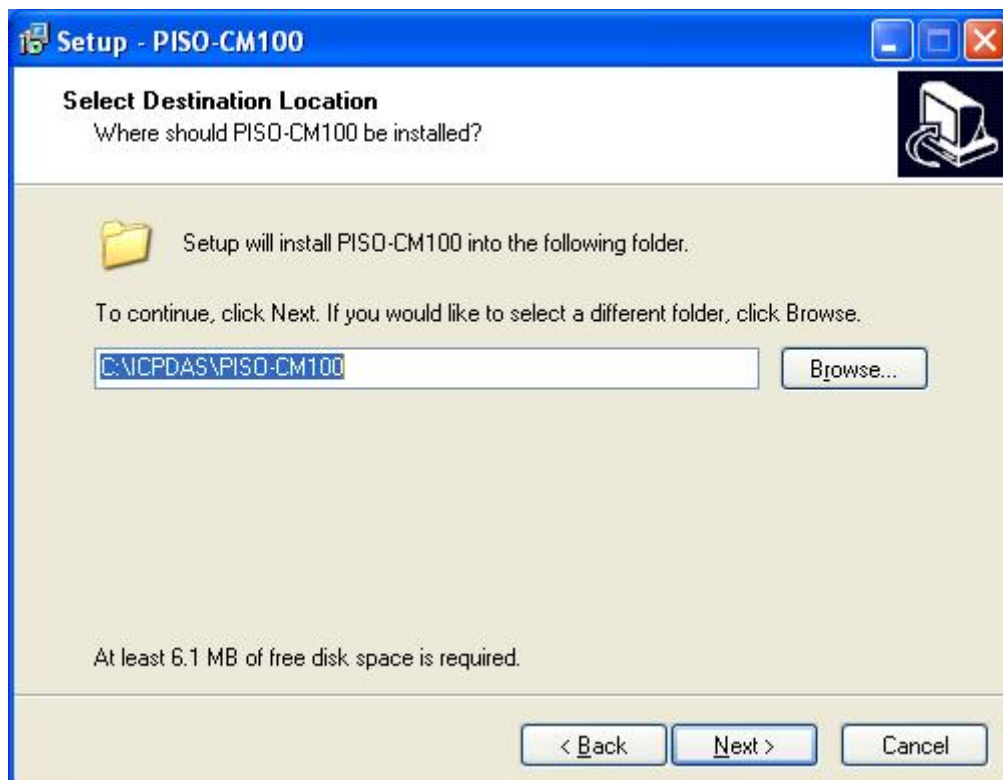
- Step 1: Shut down your system.
- Step 2: Plug your CM100s board into your system.
- Step 3: Boot up your system. When system detects a new card and pop up a wizard dialog for driver installation, cancel this dialog and skip the procedure of the driver installation.
- Step 4: Get the proper CM100s board driver for your operation system. These drivers can be found in CD of CM100s board package or on the ICP DAS product webpage.
- Step 5: Install the driver and reboot your system. In the following description, the installation procedure for Windows XP is given for an example. The installation procedures for other operation systems are similar with the one for the Windows XP.

The driver installation procedure for Window XP is shown as below:

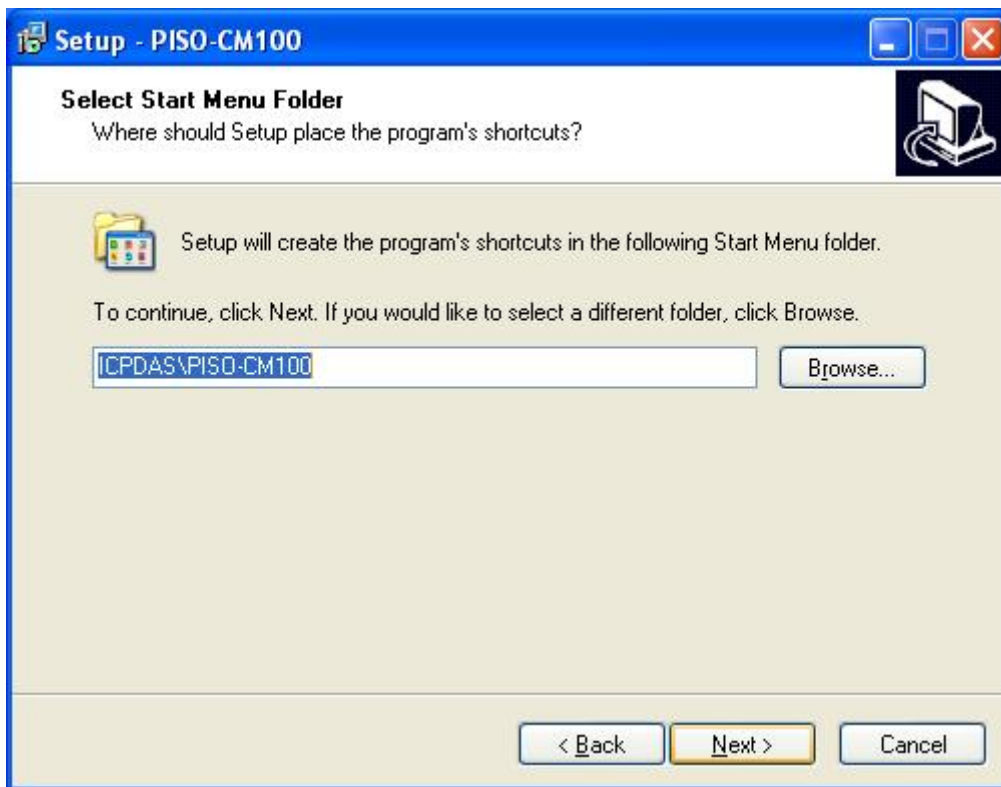
Step1: Execute the driver, piso-cm100_2k_xp.exe, which can be found on the product CD or the website of the ICP DAS.



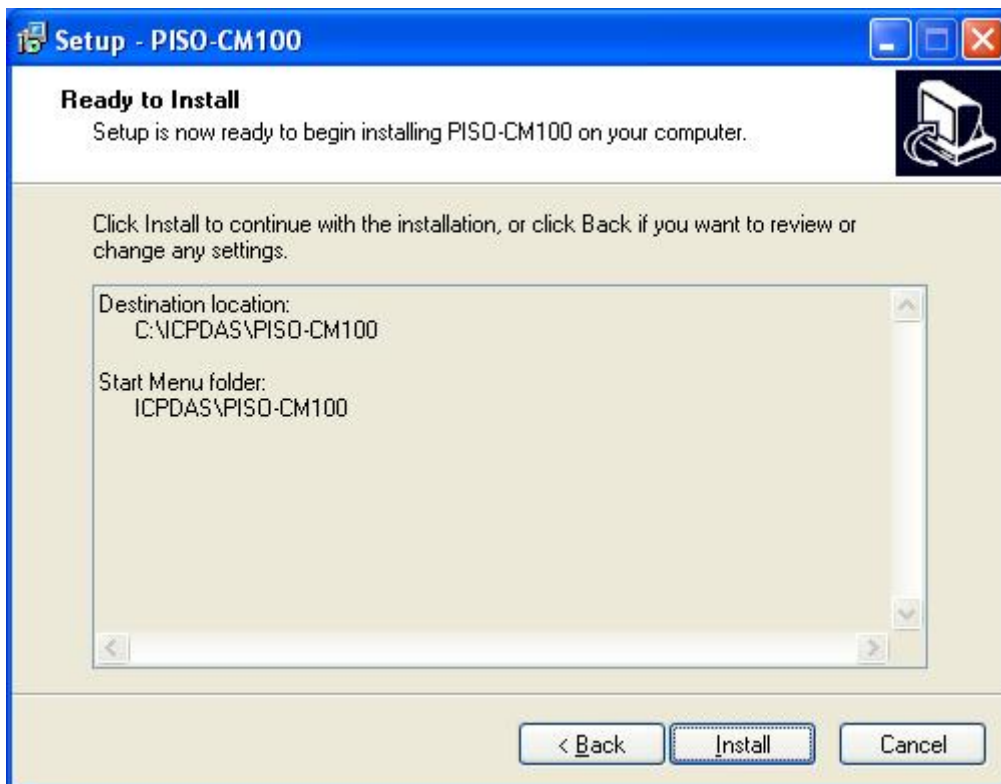
Step2: Confirm the driver installation path. This may concern with where the demos and the utility tool are.



Step3: Confirm the name of the shortcut, and click the next button.



Step4: The dialog shows the installation path and the name of the shortcut. Click the install button to go on the installation.

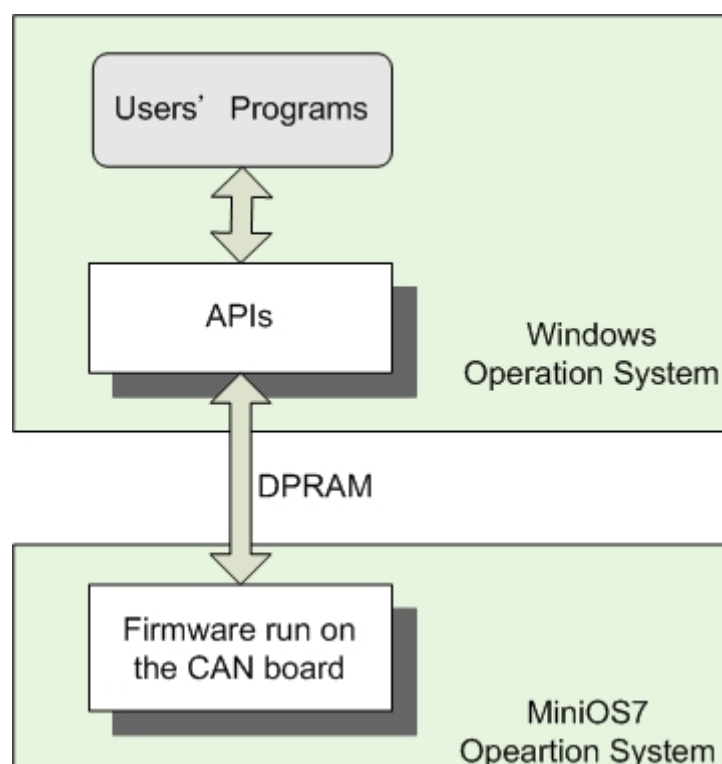


Step5: When all procedures are finished, click the finish button and reset the PC.



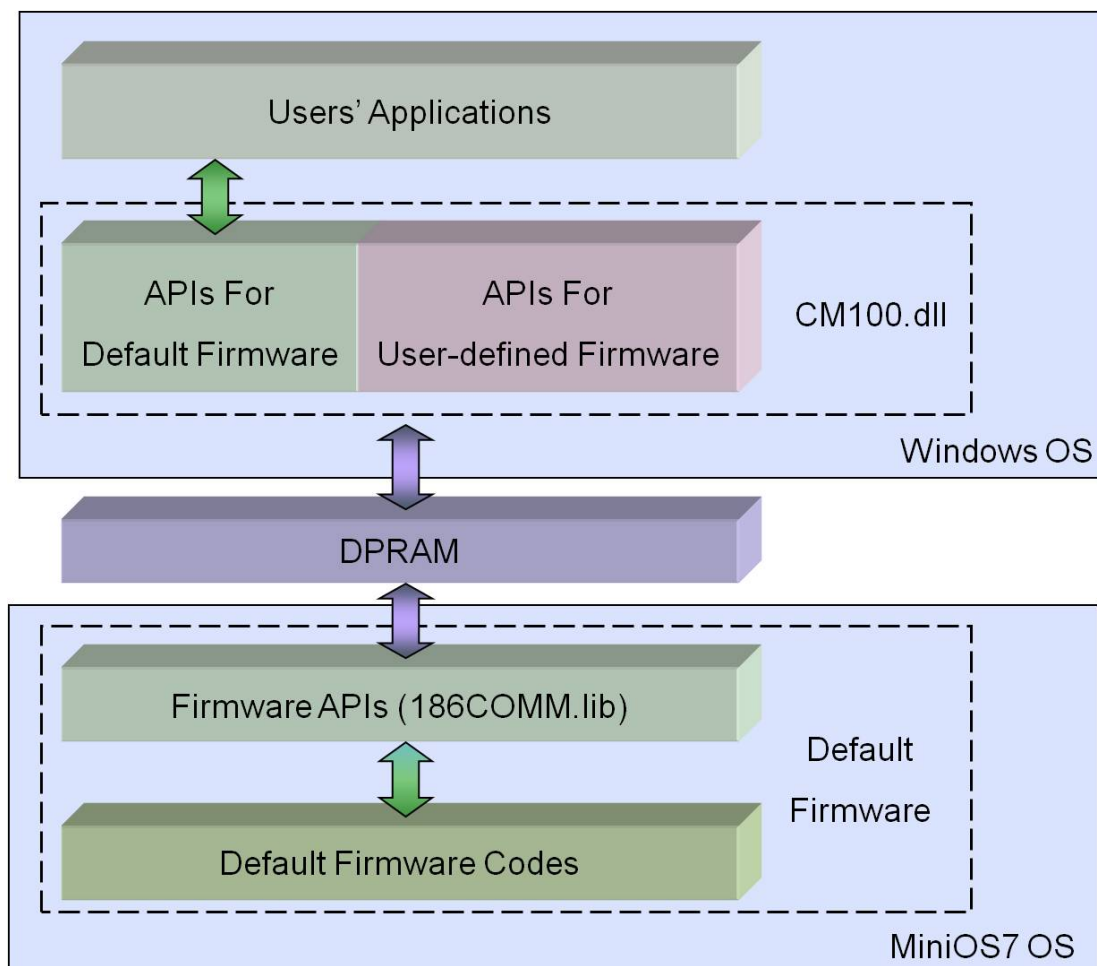
3.2 Software Architecture

The basic software architecture of the CM100s board is shown in the following figure. Users' programs can communicate with the firmware of the CM100s board via APIs. The APIs provide functions for CAN message transmission or reception, DPRAM access, the RTC adjustment and the user-defined firmware communication. Users can apply the default firmware to access the CAN network as a normal CAN card, or program the user-defined firmware to handle the CAN application protocols and process the algorithms as an intelligent CAN card.

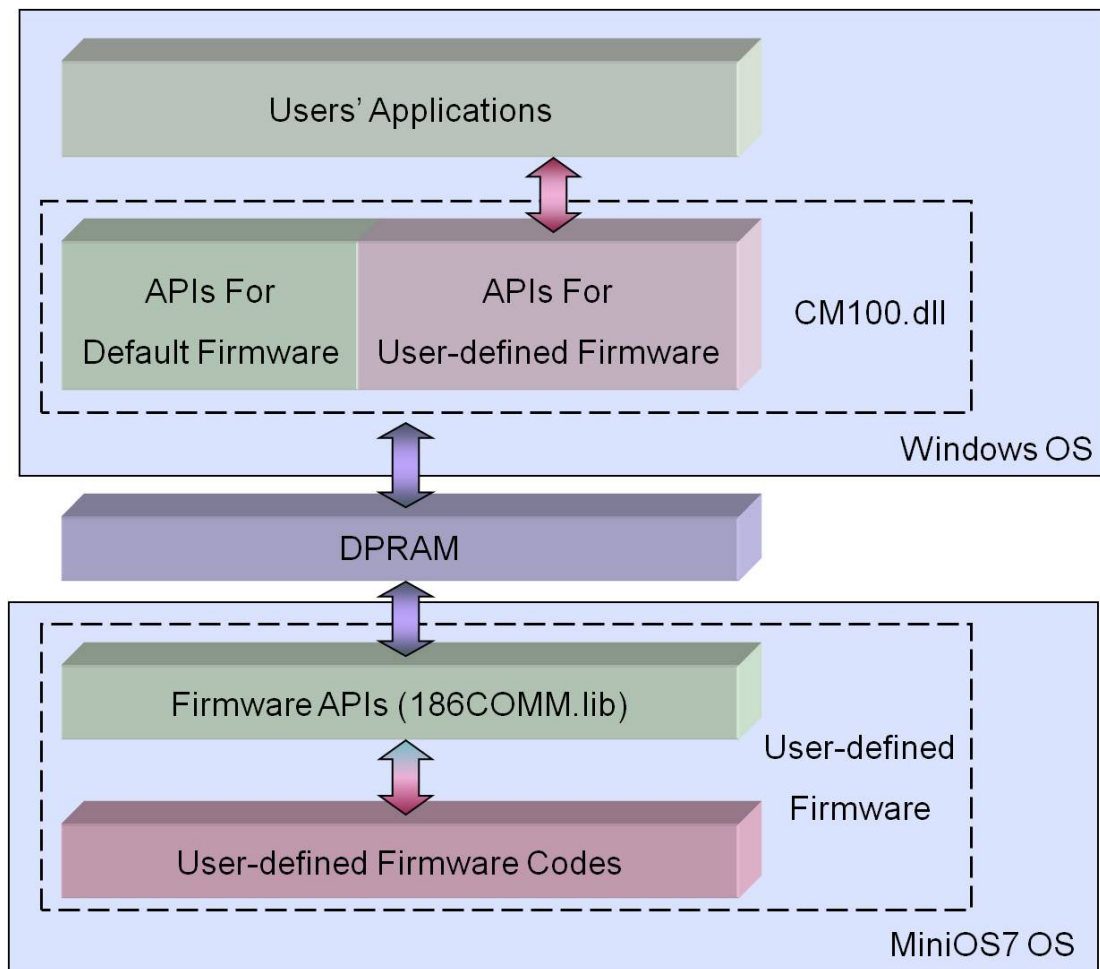


The CM100s board provides the flexibility to design the special firmware to fit the users' applications. Users can use the utility to download the firmware which will be run on the CM100s board. The default firmware offers the functions to access the CAN network as a normal CAN board. Through the firmware, users can configure the CAN controller, get the status of the CAN controller, send/receive the CAN messages to/from the CAN bus and send the CAN messages with cyclic transmission engine. These functions are helpful to reach the purposes of bus monitor, bus access, network debugging, basic network set up ... and etc. When using the default firmware, users can only

use the APIs which are for the default firmware. The information about what APIs can be used for the default firmware is described in the section 4.1. The software architecture is shown below.



If users' applications need to handle a lot of HMI interfaces or the complex CAN application protocols are applied, it is a good solution to design the special firmware to process the CAN messages. The raw CAN messages can be interpreted by the default firmware, and only the useful data are passed to the users' program of the PC. The CPU of the PC can concentrate on the HMI update, control command implementation, and data-exchange with other devices. Through this method, the CAN-related tasks are done by the user-defined firmware of the CM100s board, and the CPU loading of the PC can be effectively shared by the CPU of the CM100s board. This software architecture provides the features of the real-time execution, high performance and efficiently reducing the PC CPU loading. The software architecture is shown below.



4 APIs for the PC's Program

In this chapter, the APIs for both default firmware and user-defined firmware are described. The content includes the CM100.dll APIs introductions, error code description and the simple method for troubleshooting. It is helpful to develop the users' applications. The section 4.1 shows the list and information of all APIs supported by CM100.dll, and the section 4.2 shows the explication of the return codes of the API functions.

4.1 Windows API Definitions and Descriptions

All the functions provided by the CM100.dll are listed in the following table and the detailed information for each function presented in the following sub-section.

Function definition	Page	Note
WORD CM100_GetDllVersion(void)	31	○△
Int CM100_GetBoardInf(BYTE BoardNo, DWORD *dwVID, DWORD *dwDID, DWORD *dwSVID, DWORD *dwSDID, DWORD *dwSAuxID, DWORD *dwIrqNo)	32	○△
Int CM100_TotalBoard(void)	33	○△
Int CM100_TotalCM100Board(void)	33	○△
Int CM100_TotalDNM100Board(void)	34	○△
Int CM100_TotalCPM100Board(void)	34	○△
Int CM100_GetCM100BoardSwitchNo(BYTE BoardCntNo, BYTE *BoardSwitchNo)	35	○△
Int CM100_GetDNM100BoardSwitchNo(BYTE BoardCntNo, BYTE *BoardSwitchNo)	36	○△
Int CM100_GetCPM100BoardSwitchNo(BYTE BoardCntNo, BYTE *BoardSwitchNo)	37	○△
Int CM100_GetCardPortNum(BYTE BoardNo, BYTE *bGetPortNum)	38	○△
Int CM100_ActiveBoard(BYTE BoardNo)	39	○△
Int CM100_CloseBoard(BYTE BoardNo)	40	○△
int CM100_BoardIsActive(BYTE BoardNo)	41	○△
int CM100_AdujstDateTime(BYTE BoardNo)	42	○△
int CM100_Reset(BYTE BoardNo, BYTE Port)	43	○△

Function definition	Page	Note
int CM100_Init(BYTE BoardNo, BYTE Port)	44	○△
int CM100_HardwareReset(BYTE BoardNo, BYTE Port)	45	○△
int CM100_Check186Mode(BYTE BoardNo, BYTE *Mode)	46	○△
int CM100_Status(BYTE BoardNo, BYTE Port, BYTE *bStatus)	47	○△
int CM100_AddCyclicTxMsg(BYTE BoardNo, BYTE Port, BYTE Mode, DWORD MsgID, BYTE RTR, BYTE DataLen, BYTE *Data, DWORD TimePeriod, BYTE *Handle)	49	○△
int CM100_DeleteCyclicTxMsg(BYTE BoardNo, BYTE Port, BYTE Handle)	51	○△
int CM100_EnableCyclicTxMsg(BYTE BoardNo, BYTE Port, BYTE Handle)	52	○△
int CM100_DisableCyclicTxMsg(BYTE BoardNo, BYTE Port, BYTE Handle)	53	○△
void CM100_OutputByte(BYTE BoardNo, BYTE Port, WORD wOffset, BYTE bValue)	54	○△
BYTE CM100_InputByte(BYTE BoardNo, BYTE Port, WORD wOffset)	55	○△
int CM100_IsTxTimeout(BYTE BoardNo, BYTE Port, BYTE *Status)	56	○△
int CM100_SetSystemMsg(BYTE BoardNo, BYTE Port, BYTE Mode)	57	○△
int CM100_ClearSoftBuffer(BYTE BoardNo, BYTE Port)	58	○
int CM100_ClearBufferStatus(BYTE BoardNo, BYTE Port)	59	○
int CM100_ClearDataOverrun(BYTE BoardNo, BYTE Port)	60	○
int CM100_Config(BYTE BoardNo, BYTE Port, ConfigStruct *CanConfig)	61	○
int CM100_ConfigWithoutStruct(BYTE BoardNo, BYTE Port, DWORD AccCode, DWORD AccMask, BYTE BaudRate, BYTE BT0, BYTE BT1)	64	○
int CM100_RxMsgCount(BYTE BoardNo, BYTE Port)	65	○
int CM100_ReceiveMsg(BYTE BoardNo, BYTE Port, PacketStruct *CanPacket)	66	○
int CM100_ReceiveWithoutStruct(BYTE BoardNo, BYTE Port, BYTE *Mode, DWORD *MsgID, BYTE *RTR, BYTE *DataLen, BYTE *Data , DWORD *UpperTime , DWORD *LowerTime)	68	○
int CM100_SendMsg(BYTE BoardNo, BYTE Port, PacketStruct *CanPacket)	70	○
int CM100_SendWithoutStruct(BYTE BoardNo, BYTE Port, BYTE Mode, DWORD MsgID, BYTE RTR, BYTE DataLen, BYTE *Data)	71	○

Function definition	Page	Note
int CM100_SJA1000Config(BYTE BoardNo, BYTE Port, DWORD AccCode, DWORD AccMask, BYTE BaudRate, BYTE BT0, BYTE BT1)	73	△
int CM100_EnableSJA1000(BYTE BoardNo, BYTE Port)	74	△
int CM100_DisableSJA1000(BYTE BoardNo, BYTE Port)	75	△
int CM100_DPRAMInttToCM100(BYTE BoardNo, BYTE Port, BYTE Data)	76	△
int CM100_DPRAMWriteByte(BYTE BoardNo, BYTE Port, WORD Address, BYTE Data)	77	△
int CM100_DPRAMWriteWord(BYTE BoardNo, BYTE Port, WORD Address, WORD Data)	78	△
int CM100_DPRAMWriteDword(BYTE BoardNo, BYTE Port, WORD Address, DWORD Data)	79	△
int CM100_DPRAMWriteMultiByte(BYTE BoardNo, BYTE Port, WORD Address, BYTE *Data, WORD DataNum)	80	△
int CM100_DPRAMReadByte(BYTE BoardNo, BYTE Port, WORD Address, BYTE *Data)	81	△
int CM100_DPRAMReadWord(BYTE BoardNo, BYTE Port, WORD Address, WORD *Data)	82	△
int CM100_DPRAMReadDword(BYTE BoardNo, BYTE Port, WORD Address, DWORD *Data)	83	△
int CM100_DPRAMReadMultiByte(BYTE BoardNo, BYTE Port, WORD Address, BYTE *Data, WORD DataNum)	84	△
int CM100_DPRAMMemset(BYTE BoardNo, BYTE Port, WORD Address, BYTE Data, WORD DataNum)	85	△
int CM100_ReceiveCmd(BYTE BoardNo, BYTE Port, BYTE *Data, WORD *DataNum)	86	△
int CM100_SendCmd(BYTE BoardNo, BYTE Port, BYTE *Data, WORD DataNum)	88	△
int CM100_InstallUserISR(BYTE BoardNo, void (*UserISR)(BYTE BoardNo, BYTE InttValue))	89	△
int CM100_RemoveUserISR(BYTE BoardNo)	90	△

Note: In table 3.1, the mark ○ and △ indicate the use condition of the API functions. The function marked with ○ or △ presents that this function is useful while the default CM100 firmware or the user-defined firmware is running. If the default firmware is running, all of the functions marked with ○ could be applied. If users design their firmware by using the firmware library (firmware library is described in chapter 5), only the functions marked with △ is useful. The functions marked with ○△ can be used with default firmware or user-defined firmware.

In order to make the descriptions simpler and clearer, the attributes for the both of 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

4.1.1 CM100_GetDllVersion

- **Description:**

Obtains the version information of the CM100.dll driver.

- **Syntax:**

WORD CM100_GetDllVersion(void)

- **Parameter:**

None

- **Return:**

DLL version information. For example: If the value 100(hex) is return, it means the driver version is 1.00.

4.1.2 CM100_GetBoardInf

- **Description:**

This function is used to obtain the information of the PISO-CM100U, PCM-CM100, PISO-DNM100U or PISO-CPM100U, which includes the vender ID, the device ID and the interrupt number.

- **Syntax:**

```
int CM100_GetBoardInf(BYTE BoardNo, DWORD *dwVID,  
                      DWORD *dwDID, DWORD *dwSVID,  
                      DWORD *dwSDID, DWORD *dwSAuxID,  
                      DWORD *dwIrqNo)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or the rotary switch No. (0~9) of the PISO-CM100U, PCM-CM100, PISO-DNM100U or PISO-CPM100U.

*dwVID: [output] The address of a variable which is used to receive the vendor ID.

*dwDID: [output] The address of a variable used to receive device ID.

*dwSVID: [output] The address of a variable applied to receive sub-vendor ID.

*dwSDID: [output] The address of a variable applied to receive sub-device ID.

*dwSAuxID: [output] The address of a variable used to receive sub-auxiliary ID.

*dwIrqNo: [output] The address of a variable used to receive logical interrupt number.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

4.1.3 CM100_TotalBoard

- **Description:**

This function is used to obtain the total board number of the PISO-CM100U, PCM-CM100, PISO-DNM100U, and PISO-CPM100U boards installed in the PC or embedded system.

- **Syntax:**

Int CM100_TotalBoard(void)

- **Parameter:**

None

- **Return:**

The total scanned board number.

4.1.4 CM100_TotalCM100Board

- **Description:**

This function is used to obtain the total board number of the CM100s board installed in the PC or embedded system.

- **Syntax:**

Int CM100_TotalCM100Board(void)

- **Parameter:**

None

- **Return:**

The total scanned CM100s board number.

4.1.5 CM100_TotalDNM100Board

- **Description:**

This function is used to obtain the total board number of the PISO-DNM100U installed in the PC.

- **Syntax:**

Int CM100_TotalDNM100Board(void)

- **Parameter:**

None

- **Return:**

The total scanned PISO-DNM100U number.

4.1.6 CM100_TotalCPM100Board

- **Description:**

This function is used to obtain the total board number of the PISO-CPM100U plugged in the PC.

- **Syntax:**

Int CM100_TotalCPM100Board(void)

- **Parameter:**

None

- **Return:**

The total scanned PISO-CPM100U number.

4.1.7 CM100_GetCM100BoardSwitchNo

- **Description:**

Replies the DIP switch No. or rotary switch No. of the CM100s board.

- **Syntax:**

```
Int CM100_GetCM100BoardSwitchNo(BYTE BoardCntNo,  
                                BYTE *BoardSwitchNo)
```

- **Parameter:**

BoardCntNo: [input] The number of specified CM100s board. For example, if it is the first CM100s board in the PC, this value is 0. For second board, this value is 1.

* BoardSwitchNo: [output] The address of a variable used to get the DIP switch No. or rotary switch No. of the CM100s board.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: The parameter BoardCntNo exceeds the current total board numbers of the system.

4.1.8 CM100_GetDNM100BoardSwitchNo

- **Description:**

Replies the DIP switch No. of the PISO-DNM100U in the PC.

- **Syntax:**

```
Int CM100_GetDNM100BoardSwitchNo(BYTE BoardCntNo,  
                                   BYTE *BoardSwitchNo)
```

- **Parameter:**

BoardCntNo: [input] The number of specified PISO-DNM100U. For example, if it is the first PISO-DNM100U in the PC, this value is 0. For second board, this value is 1.

* BoardSwitchNo: [output] The address of a variable used to get the DIP switch No. of PISO-DNM100U.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: The parameter Board No. exceeds the current total scanned board number.

4.1.9 CM100_GetCPM100BoardSwitchNo

- **Description:**

Replies the DIP switch No. of the PISO-CPM100U in the PC.

- **Syntax:**

```
Int CM100_GetCPM100BoardSwitchNo(BYTE BoardCntNo,  
                                   BYTE *BoardSwitchNo)
```

- **Parameter:**

BoardCntNo: [input] The number of specified PISO-CPM100U. For example, if it is the first PISO-CPM100U in the PC, this value is 0. For second board, this value is 1.

* BoardSwitchNo: [output] The address of a variable used to get the DIP switch No. of PISO-CPM100U.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: The parameter Board No .exceeds the current total scanned board numbers.

4.1.10 CM100_GetCardPortNum

- **Description:**

Replies the port number of the PISO-CM100U, PCM-CM100, PISO-DNM100U, or PISO-CPM100U installed in the PC or embedded system.

- **Syntax:**

Int CM100_GetCardPortNum(BYTE BoardNo, BYTE *bGetPortNum)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the PISO-CM100U, PCM-CM100, PISO-DNM100U or PISO-CPM100U.

* bGetPortNum: [output] The address of a variable used to obtain the port number of the PISO-CM100U, PCM-CM100, PISO-DNM100U or PISO-CPM100U.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

4.1.11 CM100_ActiveBoard

- **Description:**

This function must be called once for activating the board before using the other functions of the APIs.

- **Syntax:**

int CM100_ActiveBoard(BYTE BoardNo)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board can not be activated or the driver file is lost.

4.1.12 CM100_CloseBoard

- **Description:**

This function must be called once for releasing the system resource before exiting the user's application program.

- **Syntax:**

int CM100_CloseBoard(BYTE BoardNo)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

4.1.13 CM100_BoardIsActive

- **Description:**

Checks if the specific board is active.

- **Syntax:**

int CM100_BoardIsActive(BYTE BoardNo)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

- **Return:**

0: the board is not active.

1: the board has been activated.

4.1.14 CM100_ AdujstDateTime

- **Description:**

Adjust the date and time of the CM100s board by using the system time.

- **Syntax:**

int CM100_AdujstDateTime(BYTE BoardNo)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_SetDateTimeFailure: Sets the date and time failure.

4.1.15 CM100_Reset

- **Description:**

Reset the CAN controller of the CM100s board.

- **Syntax:**

```
int CM100_Reset(BYTE BoardNo, BYTE Port)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

4.1.16 CM100_Init

- **Description:**

Initiates the CAN controller.

- **Syntax:**

int CM100_Init(BYTE BoardNo, BYTE Port)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

CM100_ModeError: This board is in download mode, and can't be changed to firmware mode.

4.1.17 CM100_HardwareReset

- **Description:**

Reset the CM100s board hardware, such as CAN controller, the firmware of the CM100s board and the DPRAM.

- **Syntax:**

```
int CM100_HardwareReset(BYTE BoardNo, BYTE Port)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

CM100_ModeError: This board is in download mode, and can't be changed to firmware mode.

4.1.18 CM100_Check186Mode

- **Description:**

Replies the operation mode of the specified board.

- **Syntax:**

```
int CM100_Check186Mode(BYTE BoardNo, BYTE *Mode)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

*Mode: [output] The address of a variable used to get the operation mode of the CM100s board. If this value is 0, the board is in download mode, and the value 1 is for firmware mode. If the board is in download mode, it can only update the firmware. The firmware will not work in this mode. Users can use the function CM100_Init() to change the mode of the board into the firmware mode.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_TimeOut: The CM100s board has no response.

CM100_InitError: The CM100s board replies erroneously.

CM100_ModeError: This board is in download mode, and can't be changed to firmware mode.

4.1.19 CM100_Status

- **Description:**

Replies the status of the CAN controller of the specified board.

- **Syntax:**

int CM100_Status(BYTE BoardNo, BYTE Port, BYTE *bStatus)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

*bStatus: [output] The address of a variable applied to get the status value of the CAN controller.

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	NAME	VALUE	STATUS
bit 0	Receive Buffer Status	1	full/not empty
		0	empty

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary
switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

4.1.20 CM100_AddCyclicTxMsg

- **Description:**

Adds a cyclic transmission message into the firmware of the board. Afterwards, users can use the functions CM100_EnableCyclicTxMsg() and CM100_DelectCyclicTxMsg() to enable or disable this cyclic transmission message. The maximum addable number of the transmission messages is 5. After adding a cyclic transmission message, the handle for this message will be returned. The less value of handle indicates the higher priority of the cyclic transmission messages. If there are two cycle transmission messages need to be sent at the same time, the higher priority message will be sent first.

- **Syntax:**

```
int CM100_AddCyclicTxMsg(BYTE BoardNo, BYTE Port, BYTE Mode,  
                        DWORD MsgID, BYTE RTR,  
                        BYTE DataLen, BYTE *Data,  
                        DWORD TimePeriod,  
                        DWORD TransmitTimes, BYTE *Handle)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Mode: [input] 0 for 11-bit message ID, 1 for 29-bit message ID.

MsgID: [input] The CAN message ID.

RTR: [input] 0 for transmitting the remote-transmit-request message, 1 for transmitting the normal message. As this bit is 1, the parameter *Data is useless.

DataLen: [input] The data length of the CAN message. The maximum value is 8.

*Data: [input] The start address of the data array of a CAN message. The maximum length of the Data array is 8 bytes.

TimePeriod: [input] The time period for the cyclic transmission. This parameter is formatted by 0.1ms. The minimum value is 5.

TransmitTimes: [input] The transmission times for the cyclic transmission. After enabling the transmission, the message will be sent for specified times. If the value of the parameter is 0, the transmission goes cyclically until disabling the cyclic transmission.

*Handle: [output] The address of a variable is used to get the handle of a cyclic transmission. When users would like to enable or disable the specified transmission, this value is needed.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

CM100_SetCyclicMsgFailure: The applied cyclic transmission message number is more than 5, the parameter time period is less than 5 (0.5ms). or the CM100s board replies erroneously.

4.1.21 CM100_DeleteCyclicTxMsg

- **Description:**

Removes the specified cyclic transmission message added by the function CM100_AddCyclicTxMsg().

- **Syntax:**

```
int CM100_DeleteCyclicTxMsg(BYTE BoardNo, BYTE Port,  
                             BYTE Handle)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Handle: [input] The handle of the cyclic transmission message, which is obtained by the function CM100_AddCyclicTxMsg().

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

CM100_SetCyclicMsgFailure: The CM100s board replies erroneously.

4.1.22 CM100_EnableCyclicTxMsg

- **Description:**

Enables the cyclic transmission message added by the function CM100_AddCyclicTxMsg(). After calling this function, the specified cyclic transmission message will be transmitted.

- **Syntax:**

```
int CM100_EnableCyclicTxMsg(BYTE BoardNo, BYTE Port,  
                             BYTE Handle)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Handle: [input] The handle of the cyclic transmission message, which is obtained by the function CM100_AddCyclicTxMsg().

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

CM100_SetCyclicMsgFailure: The CM100s board replies erroneously.

4.1.23 CM100_DisableCyclicTxMsg

- **Description:**

Disables the cyclic transmission message enabled by the function CM100_EnableCyclicTxMsg().

- **Syntax:**

```
int CM100_DisableCyclicTxMsg(BYTE BoardNo, BYTE Port,  
                             BYTE Handle)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Handle: [input] The handle of the cyclic transmission message, which is obtained by the function CM100_AddCyclicTxMsg().

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

CM100_SetCyclicMsgFailure: The CM100s board replies erroneously.

4.1.24 CM100_OutputByte

- **Description:**

Writes the data to the specified SJA1000 register of the CM100s board.

- **Syntax:**

```
void CM100_OutputByte(BYTE BoardNo, BYTE Port, WORD wOffset,  
                      BYTE bValue)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

wOffset: [input] The register address of the SJA1000.

bValue: [input] The value written to the specified register.

- **Return:**

None

4.1.25 CM100_InputByte

- **Description:**

Read the data from the specified SJA1000 register of the CM100s board.

- **Syntax:**

BYTE CM100_InputByte(BYTE BoardNo, BYTE Port, WORD wOffset)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

wOffset: [input] The register address of the SJA1000.

- **Return:**

The value read from the specified register.

4.1.26 CM100_IsTxTimeout

- **Description:**

Checks if the CAN controller successfully sends the CAN message.

- **Syntax:**

BYTE CM100_IsTxTimeout(BYTE BoardNo, BYTE Port, BYTE* Status)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

*Status: [output] The transmit status. The value 1 means that the CAN controller can't successfully send the CAN message to the network and the value 0 is for no error.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

4.1.27 CM100_SetSystemMsg

- **Description:**

This function is used to determine if the system information of the firmware of the CM100s board is shown.

- **Syntax:**

BYTE CM100_SetSystemMsg(BYTE BoardNo, BYTE Port,
BYTE Mode)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Mode: [input] The system information of the firmware is shown through the debug port of CM100s board as the value is 1, and the value 0 for disabling the display.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

4.1.28 CM100_ClearSoftBuffer <For default firmware>

- **Description:**

Clean the software buffer of the CM100s board. After calling this function, all the CAN messages which are not sent yet will be deleted.

- **Syntax:**

```
int CM100_ClearSoftBuffer(BYTE BoardNo, BYTE Port)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

4.1.29 CM100_ClearBufferStatus <For default firmware>

- **Description:**

Cleans the statuses of the CAN transmission and reception software buffers. When calling the functions CM100_SendMsg(), CM100_SendWithoutStruct(), CM100_ReceiveMsg() or CM100_ReceiveWithoutStruct(), users may get the error code due to the problems of the CAN message buffers. This function may be needed to clean the buffer and reset the buffer statuses.

- **Syntax:**

```
int CM100_ClearBufferStatus(BYTE BoardNo, BYTE Port)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

4.1.30 CM100_ClearDataOverrun <For default firmware>

- **Description:**

Cleans the data overrun status of the SJA1000.

- **Syntax:**

int CM100_ClearDataOverrun(BYTE BoardNo, BYTE Port)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

4.1.31 CM100_Config <For default firmware>

- **Description:**

Configures the baud, message filter of the SJA1000. After calling this function, the board can start to send/receive CAN messages to/from the CAN network.

- **Syntax:**

```
int CM100_Config(BYTE BoardNo, BYTE Port,  
                ConfigStruct *CanConfig)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

* CanConfig: [input] The address of a ConfigStruct structure variable used to configure the CM100s board. The ConfigStruct structure is defined below:

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

AccCode[4]: Acceptance code of CAN controller.

AccMask[4]: Acceptance mask of CAN controller.

The AccCode is used to decide which CAN IDs are accepted by the CAN controller. The AccMask is used to decide which bit of the CAN IDs are checked with the AccCode by the CAN controller. If the bit of AccMask is set to 0, it means that the bit in the same position of the CAN IDs need to be checked and the ID bit value needs to match the bit of AccCode in the same position.

AccCode and AccMask Definition For 11-bit ID:

AccCode and AccMask	Bit Position	Filter Target
high byte of the high word	bit7~bit0	bit10 ~ bit3 of ID
low byte of the high word	bit7~bit5	bit2 ~ bit0 of ID
low byte of the high word	bit4	RTR
low byte of the high word	bit3~bit0	no use
high byte of the low word	bit7~bit0	bit7 ~ bit0 of 1st byte data
low byte of the low word	bit7~bit0	bit7 ~ bit0 of 2nd byte data

AccCode and AccMask Definition For 29-bit ID:

AccCode and AccMask	Bit Position	Filter Target
high byte of the high word	bit7~bit0	bit28~ bit21 of ID
low byte of the high word	bit7~bit0	bit20 ~ bit13 of ID
high byte of the low word	bit7~bit0	bit12 ~ bit5 of ID
low byte of the low word	bit7~bit3	bit4 ~ bit0 of ID
low byte of the low word	bit2	RTR
low byte of the low word	bit1~bit0	no use

Example for 29 bit ID message:

	Array[0]	Array[1]	Array[2]	Array[3]	
AccCode :	00h	00h	00h	A0h	
AccMask :	FFh	FFh	FFh	1Fh	
ID bit :	bit28~bit21	bit20~bit13	bit12~bit5	bit4~bit0	
ID Value :	xxxx xxxx	xxxx xxxx	xxxx xxxx	101x x	will be accepted

(Note: The character “x” means the bit value doesn’t care. The character “h” behind the value means the value is in hex format.)

BaudRate:

Value	Description
0	User-defined baud (BT0,BT1 are needed)
1	10 k bps
2	20 k bps
3	50 k bps
4	125 k bps
5	250 k bps
6	500 k bps
7	800 k bps
8	1000 k bps

BT0, BT1: User-defined baud rate (used only if the BaudRate is 0). For example, sets the BT0 to 04h and the BT1 to 1Ch, then the CAN controller is running in 100 kbps baud rate. For more details about how to set the BT0 and BT1, please refer to the datasheet of the SJA1000 CAN controller.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

CM100_InitError: The CM100s board replies erroneously.

4.1.32 CM100_ConfigWithoutStruct <For default firmware>

- **Description:**

This function is similar to the function CM100_Config(). The difference is the input parameters. This function uses non-structure parameters so that it is easy to be applied in some program environment, such as VB. About the input parameters of this function, please refer to the function CM100_Config() for the details.

- **Syntax:**

```
int CM100_ConfigWithoutStruct(BYTE BoardNo, BYTE Port,  
                              DWORD AccCode, DWORD AccMask,  
                              BYTE BaudRate, BYTE BT0,  
                              BYTE BT1)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

AccCode: [input] Acceptance code of CAN controller.

AccMask: [input] Acceptance mask of CAN controller.

BaudRate: [input] The baud indicator of CAN controller.

BT0: [input] User-defined baud.

BT1: [input] User-defined baud.

For more information about these parameters, please refer to the description of the function CM100_Config().

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

CM100_InitError: The CM100s board replies erroneously.

4.1.33 CM100_RxMsgCount <For default firmware>

- **Description:**

Replies the number of CAN messages available in the reception software buffer.

- **Syntax:**

int CM100_RxMsgCount(BYTE BoardNo, BYTE Port)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

- **Return:**

The number of CAN messages stored in software buffer.

4.1.34 CM100_ReceiveMsg <For default firmware>

- **Description:**

Gets one message from the reception software buffer. Before using this function, the CAN controller must be configured by the functions CM100_Config() or CM100_ConfigWithoutStruct().

- **Syntax:**

```
int CM100_ReceiveMsg(BYTE BoardNo, BYTE Port,  
                    PacketStruct *CanPacket)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

*CanPacket: [output] The address of a PacketStruct structure variable used to get a CAN message. The PacketStruct structure is defined below:

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

} PacketStruct;

MsgTimeStamps: As the board gets a CAN message, this parameter records the time which is calculated from the boot time of the CPU of the CAN board. The unit is 0.1 ms.

mode: If the value is 0, the received CAN message is 11-bit-ID CAN message. The 29-bit-ID CAN message has the value 1

id: The CAN message ID.

rtr: 0 for the remote-transmit-request message, 1 for the

normal message. As this bit is 1, the parameter data[8] is useless.

len: The data length of a CAN message. The maximum length is 8 bytes.

data[8]: The data of a CAN message

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_SoftBufferIsEmpty: There is no CAN message in the reception software buffer.

CM100_SoftBufferIsFull: Users can still get CAN messages from the reception software buffer, but the software buffer is overflow.

CM100_TimeOut: The CM100s board has no response.

4.1.35 CM100_ReceiveWithoutStruct <For default firmware>

- **Description:**

Replies a received message from the software buffer. This function is similar to the function CM100_ReceiveMsg(). The difference is that this function doesn't use any structure parameter. It is easy to use in some program environment, such as VB.

- **Syntax:**

```
int CM100_ReceiveWithoutStruct(BYTE BoardNo, BYTE Port,  
                                BYTE *Mode, DWORD *MsgID,  
                                BYTE *RTR, BYTE *DataLen,  
                                BYTE *Data , DWORD *UpperTime ,  
                                DWORD *LowerTime)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

*Mode: [output] The address of a variable used to get the mode of a CAN message. If the value is 0, the received CAN message is 11-bit-ID CAN message. The 29-bit-ID CAN message will have the value 1.

*MsgID: [output] The address of a variable used to get the CAN message ID.

*RTR: [output] The address of a variable used to obtain the status of this CAN message. 0 for the remote-transmit-request message, 1 for

the normal message.

*DataLen: [output] The address of a variable used to obtain the data length of a CAN message. The range of this value is 0~8.

*Data: [output] The start address of a buffer used to get the data of a CAN message. Users need to put an 8-byte array in this filed.

*UpperTime: [output] The address of a variable used to obtain the higher 32-bit time stamp of a CAN message.

*LowerTime: [output] The address of a variable used to obtain the lower 32-bit time stamp of a CAN message. The unit of UpperTime and LowerTime are 0.1ms.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_SoftBufferIsEmpty: There is no CAN message in the reception software buffer.

CM100_SoftBufferIsFull: Users can still get CAN messages from the reception software buffer, but the software buffer is overflow.

CM100_TimeOut: The CM100s board has no response.

4.1.36 CM100_SendMsg <For default firmware>

- **Description:**

Sends a CAN message to the software transmission buffer. As the CAN bus is idle, the CAN message will be sent to the CAN network. Note that if there are something wrong in the CAN bus wiring and configuration, the CAN message may not be transmitted successfully. In this case, users can use the function API CM100_Status to check if any error is happen.

- **Syntax:**

```
int CM100_SendMsg(BYTE BoardNo, BYTE Port,  
                  PacketStruct *CanPacket)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

*CanPacket: [input] The address of a PacketStruct structure variable used to describe the sent CAN message. About the definition of PacketStruct, please refer to the description of the function CM100_ReceiveMsg().

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

CM100_SoftBufferIsFull: The transmission software buffer is overflow.

4.1.37 CM100_SendWithoutStruct <For default firmware>

- **Description:**

Sends a CAN message to the software transmission buffer. As the CAN bus is idle, the CAN message will be sent to the CAN network. This function is similar to the function CM100_SendMsg(). The difference is that this function doesn't use any structure parameter. It is easy to use in some program environment, such as VB. Note that if there are something wrong in the CAN bus wiring and configuration, the CAN message may not be transmitted successfully. In this case, users can use the function API CM100_Status to check if any error is happen.

- **Syntax:**

```
int CM100_SendWithoutStruct(BYTE BoardNo, BYTE Port, BYTE
                             Mode, DWORD MsgID, BYTE RTR,
                             BYTE DataLen, BYTE *Data)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Mode: [input] 0 for 11-bit message ID, 1 for 29-bit message ID.

MsgID: [input] CAN message ID.

RTR: [input] 0 for normal messages, 1 for remote-transmit-request messages. As the value is 1, the parameter *Data is useless.

DataLen: [input] The data length of a transmitted CAN message. The

maximum value is 8.

*Data: [input] The start address of a buffer used to store the transmitted data of a CAN message.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

CM100_SoftBufferIsFull: The transmission software buffer is overflow.

4.1.38 CM100_SJA1000Config <For user-defined firmware>

- **Description:**

Configures the message filter and baud of the SJA1000. About the input parameters of this function, please refer to the function CM100_Config() for the details. **Note that after using this function to configure the CAN controller, user must call the function CM100_EnableSJA1000 to enable the CAN controller and the configuration.**

- **Syntax:**

```
int CM100_SJA1000Config(BYTE BoardNo, BYTE Port,  
                        DWORD AccCode, DWORD AccMask,  
                        BYTE BaudRate, BYTE BT0, BYTE BT1)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

AccCode: [input] Acceptance code of CAN controller.

AccMask: [input] Acceptance mask of CAN controller.

BT0: [input] User-defined baud.

BT1: [input] User-defined baud.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

CM100_InitError: The CM100s board replies erroneously.

4.1.39 CM100_EnableSJA1000 <For user-defined firmware>

- **Description:**

After calling the function CM100_SJA1000Config, the CAN controller is reset. So users must to use this function to enable the CAN controller.

- **Syntax:**

int CM100_EnableSJA1000 (BYTE BoardNo, BYTE Port)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

CM100_InitError: The CM100s board replies erroneously.

4.1.40 CM100_DisableSJA1000 <For user-defined firmware>

- **Description:**

If the function is used, the CAN controller will be disabled and can't receive or send any CAN message until calling the function CM100_EnableSJA1000.

- **Syntax:**

int CM100_DisableSJA1000(BYTE BoardNo, BYTE Port)

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_TimeOut: The CM100s board has no response.

CM100_InitError: The CM100s board replies erroneously.

4.1.41 CM100_DPRAMInttToCM100 <For user-defined firmware>

- **Description:**

Send an interrupt signal to the CM100s board. This interrupt signal is passed to the user-defined firmware, and triggers some action designed by users. Be careful about that too many interrupt signals at a short period of time will affect the normal procedure of the user-defined firmware.

- **Syntax:**

```
int CM100_DPRAMInttToCM100(BYTE BoardNo, BYTE Port,  
                             BYTE Data)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Data: [input] The interrupt indicator. The range is 0x00 ~ 0xdf. Users can define their own interrupt indicator and run the specified tasks depending on the corresponding interrupt indicators in the user-defined firmware.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_DpramOverRange: The parameters Data is out of range.

4.1.42 CM100_DPRAMWriteByte <For user-defined firmware>

- **Description:**

Write one byte data into the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int CM100_DPRAMWriteByte(BYTE BoardNo, BYTE Port,  
                           WORD Address, BYTE Data)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Address: [input] The specified address of the DPRAM where users want to write the data to. The legal address range for the DPRAM is 0 ~ 6999.

Data: [input] The byte data written to the DPRAM.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_DpramOverRange: The parameter Address is out of range.

4.1.43 CM100_DPRAMWriteWord <For user-defined firmware>

- **Description:**

Write one word data into the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int CM100_DPRAMWriteWord(BYTE BoardNo, BYTE Port,  
                           WORD Address, WORD Data)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Address: [input] The specified address of the DPRAM where users want to write the data to. The legal address range for the DPRAM is 0 ~ 6998.

Data: [input] The word data written to the DPRAM.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_DpramOverRange: The parameter Address is out of range.

4.1.44 **CM100_DPRAMWriteDword** <For user-defined firmware>

- **Description:**

Write one double-word data into the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int CM100_DPRAMWriteDword(BYTE BoardNo, BYTE Port,  
                           WORD Address, DWORD Data)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Address: [input] The specified address of the DPRAM where users want to write the data to. The legal address range for the DPRAM is 0 ~ 6996.

Data: [input] The double-word data written to the DPRAM.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_DpramOverRange: The parameter Address is out of range.

4.1.45 CM100_DPRAMWriteMultiByte <For user-defined firmware>

- **Description:**

Write multi-byte data into the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int CM100_DPRAMWriteMultiByte(BYTE BoardNo, BYTE Port,  
                                WORD Address, BYTE *Data,  
                                WORD DataNum)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Address: [input] The specified address of the DPRAM where users want to write the data to. The legal address range is depended on the parameter DataNum. The sum of the parameters Address and DataNum can't exceed the value 6999.

*Data: [input] The start address of a byte array written to the DPRAM.

DataNum: [input] The byte number of an data array written to the DPRAM.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_DpramOverRange: The sum of the parameters Address and DataNum is out of range.

4.1.46 CM100_DPRAMReadByte <For user-defined firmware>

- **Description:**

Read one byte data from the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int CM100_DPRAMReadByte(BYTE BoardNo, BYTE Port,  
                          WORD Address, BYTE *Data)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Address: [input] The specified address of the DPRAM where users want to read the data from. The legal address range for the DPRAM is 0 ~ 6999.

*Data: [output] The address of a variable used to receive the obtained data of the DPRAM.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_DpramOverRange: The parameter Address is out of range.

4.1.47 **CM100_DPRAMReadWord** <For user-defined firmware>

- **Description:**

Read one word data from the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int CM100_DPRAMReadWord(BYTE BoardNo, BYTE Port,  
                          WORD Address, WORD *Data)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Address: [input] The specified address of the DPRAM where users want to read the data from. The legal address range for the DPRAM is 0 ~ 6998.

*Data: [output] The address of a variable used to receive the obtained data of the DPRAM.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_DpramOverRange: The parameter Address is out of range.

4.1.48 **CM100_DPRAMReadDword** <For user-defined firmware>

- **Description:**

Read one double-word data from the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int CM100_DPRAMWriteDword(BYTE BoardNo, BYTE Port,  
                           WORD Address, DWORD *Data)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Address: [input] The specified address of the DPRAM where users want to read the data from. The legal address range for the DPRAM is 0 ~ 6996.

*Data: [output] The address of a variable used to receive the obtained data of the DPRAM.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_DpramOverRange: The parameter Address is out of range.

4.1.49 CM100_DPRAMReadMultiByte <For user-defined firmware>

- **Description:**

Read multi-byte data from the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int CM100_DPRAMReadMultiByte(BYTE BoardNo, BYTE Port,  
                               WORD Address, BYTE *Data,  
                               WORD DataNum)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Address: [input] The specified address of the DPRAM where users want to read the data from. The legal address range is depended on the parameter DataNum. The sum of the parameters Address and DataNum can't exceed the value 6999.

*Data: [output] The start address of a byte array applied to receive the obtained data of the DPRAM.

DataNum: [input] The byte number which users want to read from the DPRAM.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_DpramOverRange: The sum of the parameters Address and DataNum is out of range.

4.1.50 CM100_DPRAMMemset <For user-defined firmware>

- **Description:**

Sets multi-byte DPRAM data to be the specified value.

- **Syntax:**

```
int CM100_DPRAMMemset(BYTE BoardNo, BYTE Port,  
                        WORD Address, BYTE Data,  
                        WORD DataNum)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

Address: [input] The specified address of the DPRAM where users want to write the data to. The legal address range is depended on the parameter DataNum. The sum of the parameters Address and DataNum can't exceed the value 6999.

Data: [input] The data written to the DPRAM.

DataNum: [input] The byte number which users want to write to the DPRAM.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

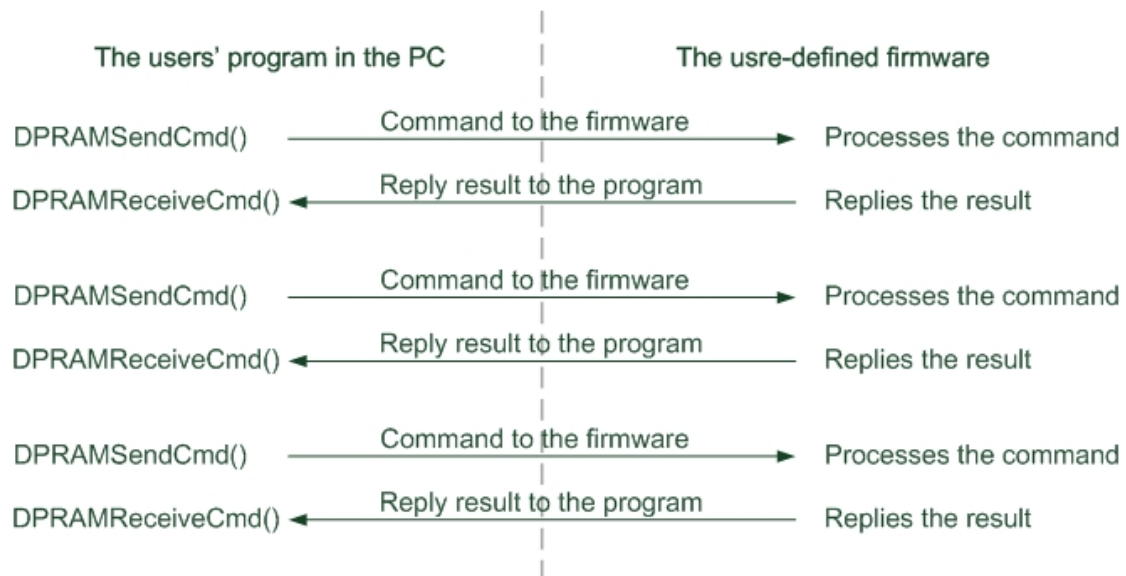
CM100_PortNumberError: The port number is not correct.

CM100_DpramOverRange: The sum of the parameters Address and DataNum is out of range.

4.1.51 CM100_ReceiveCmd <For user-defined firmware>

- **Description:**

Use this function to receive the command transmitted from the user-defined firmware. After calling the function DPRAMSendCmd() in the PC program to send a command to the user-defined firmware, call this function to receive the response from the user-defined firmware. If users do not receive the response until another response is given from the user-defined firmware, the former one will be covered by the latter one. It is strongly recommended to use the polling mechanism in the PC program and the user-defined firmware, as the following figure.



- **Syntax:**

```
int CM100_ReceiveCmd(BYTE BoardNo, BYTE Port, BYTE *Data,
                     WORD *DataNum)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

*Data: [output] The start address of a byte array applied to receive the command from DPRAM of CM100s board.

*DataNum: [output] The address of a variable applied to receive the command length.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_NoDpramCmd: There is no command transmitted from the user-defined firmware.

CM100_DpramOverRange: The command length is over 512 bytes.

4.1.52 CM100_SendCmd <For user-defined firmware>

- **Description:**

Call this function to send the command to the user-defined firmware. The maximum command length is 512 bytes. Afterwards, users can use the function DPRAMReceiveCmd() of the firmware library to get the command, and run the corresponding task in the firmware. About the function DPRAMReceiveCmd(), please refer to chapter 5 for the details.

- **Syntax:**

```
int CM100_SendCmd(BYTE BoardNo, BYTE Port, BYTE *Data,  
                  WORD DataNum)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

Port: [input] The CAN port No. of the CM100s board. This value is always 1.

*Data: [input] The start address of a byte array of the sent command.

DataNum: [input] The word value indicates the length of the command sent to the user-defined firmware.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

CM100_PortNumberError: The port number is not correct.

CM100_DpramOverRange: The command length is over 512 bytes.

4.1.53 CM100_InstallUserISR <For user-defined firmware>

- **Description:**

This function allows users to apply the ISR (interrupt service routine) to process the reply from the firmware. When users install the ISR by using this function, the interrupt indicators from the user-defined firmware can be read in the ISR, and users can immediately run the tasks in the users' program depending on the interrupt indicator. The interrupt indicator, CAN_COMM_CMD_FROM_CM100, defined in "cm100.h" is also passed to the users' ISR after calling the function DPRAMSendCmd in the user-defined firmware.

- **Syntax:**

```
int CM100_InstallUserISR(BYTE BoardNo,  
                        void (*UserISR)(BYTE BoardNo, BYTE InttValue))
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

(*UserISR)(BYTE BoardNo, BYTE InttValue)): [input] The pointer which points a function with format "void XXX(Byte BoardNo, Byte InttValue)". The characters XXX are the function name of the users' ISR. The parameter BoardNo of the ISR indicates which board receives the interrupt indicator, and the parameter InttValue is the interrupt indicator from the user-defined firmware.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

4.1.54 CM100_RemoveUserISR <For user-defined firmware>

- **Description:**

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

- **Syntax:**

```
int CM100_RemoveUserISR(BYTE BoardNo)
```

- **Parameter:**

BoardNo: [input] The DIP switch No. (0~15) or rotary switch No. (0~9) of the board.

- **Return:**

CM100_NoError: OK

CM100_DriverError: The driver is inactive or no board is in the system.

CM100_BoardNumberError: Can't find the DIP switch No. or rotary switch No. of the board to match with the BoardNo.

CM100_ActiveBoardError: This board is not activated.

4.2 Windows API Return Codes Troubleshooting

Return Code	Error ID (Error Description)	Troubleshooting
0	CM100_NoError	OK
1	CM100_DriverError	1. Reinstall the CM100s board driver correctly. 2. Unplug the CM100s board, and plug it again. Turn on the PC and check if the board can be found in the list of the hardware management of the Windows operation system.
2	CM100_ActiveBoardError	1. Set the parameter BoardNo of the function to match the DIP switch No. or rotary switch No. of the board. 2. Turn off all programs which may activate this board. 3. Check if each CAN board has the unique DIP switch No. or rotary switch No. 4. Reinstall the CM100s board driver correctly. 5. Unplug the CM100s board, and plug it again. Turn on the PC and check if the board can be found in the list of the hardware management of the Windows operation system.
3	CM100_BoardNumberError	1. Set the parameter BoardNo of the function to match the DIP switch No. or rotary switch No. of the board. 2. Check if each CAN board has the unique DIP switch No. or rotary switch No. 3. Unplug the CM100s board, and plug it again. Turn on the PC and check if the board can be found in the list of the hardware management of the Windows operation system.
4	CM100_PortNumberError	1. Set the Port parameter to 1 if users use the CM100s board.
7	CM100_InitError	1. Call the function CM100_Init() and configure the CM100s board again.
21	CM100_SoftBufferIsEmpty	1. Wait for a while and retry it again.
22	CM100_SoftBufferIsFull	1. Use the function CM100_ClearBufferStatus to clean the status of buffer overflow. 2. Reduce the CAN bus loading.
23	CM100_TimeOut	1. Check if the users' program and user-defined firmware follows the polling mechanism. 2. Confirm that only one API or thread can access the board at the same time. 3. Call the function CM100_Init() and configure the CM100s board again. 4. Push the SW1 or SW2 of the board to force the board into the download mode. Update the firmware by using the CAN utility.

Return Code	Error ID (Error Description)	Troubleshooting
		5. Confirm if the board is in operation mode, not in download mode.
24	CM100_SetCyclicMsgFailure	1. Check if 5 cyclic messages are already used. 2. Call the function CM100_Init() and configure the CM100s board again.
25	CM100_DpramOverRange	1. Check the value of the parameters Address or DataNum of the function.
26	CM100_NoDpramCmd	1. Wait for a while and retry it again.
27	CM100_ModeError	1. Push the SW1 or SW2 of the board to force the board into the download mode. Update the firmware by using the CAN utility.
30	CM100_NoFileInside	1. Push the SW1 or SW2 of the board to force the board into the download mode. Update the firmware by using the CAN utility.
31	CM100_DownloadFailure	1. Close the utility and try to update the firmware one minute later. 2. Call your distributor to solve this problem
32	CM100_EEPROMDamage	1. Call your distributor to solve this problem
33	CM100_NotEnoughSpace	1. The file size of the user-defined firmware is too large to put it into the CM100s board.
34	CM100_StillDownloading	1. Close the utility and try to update the firmware one minute later. 2. Call your distributor to solve this problem
35	CM100_BoardModeError	1. Close the utility and try to update the firmware one minute later.
36	CM100_SetDateTimeFailure	1. Call your distributor to solve this problem.
X	Transmit one CAN message successfully, but can't monitor this can messages on the CAN bus.	1. Use the function CM100_Status to check if any error is happen. 2. Check if the CAN bus line is correctly connected with the connector of the CM100s board. 3. Check if the baud rate of the CM100s board is correct.

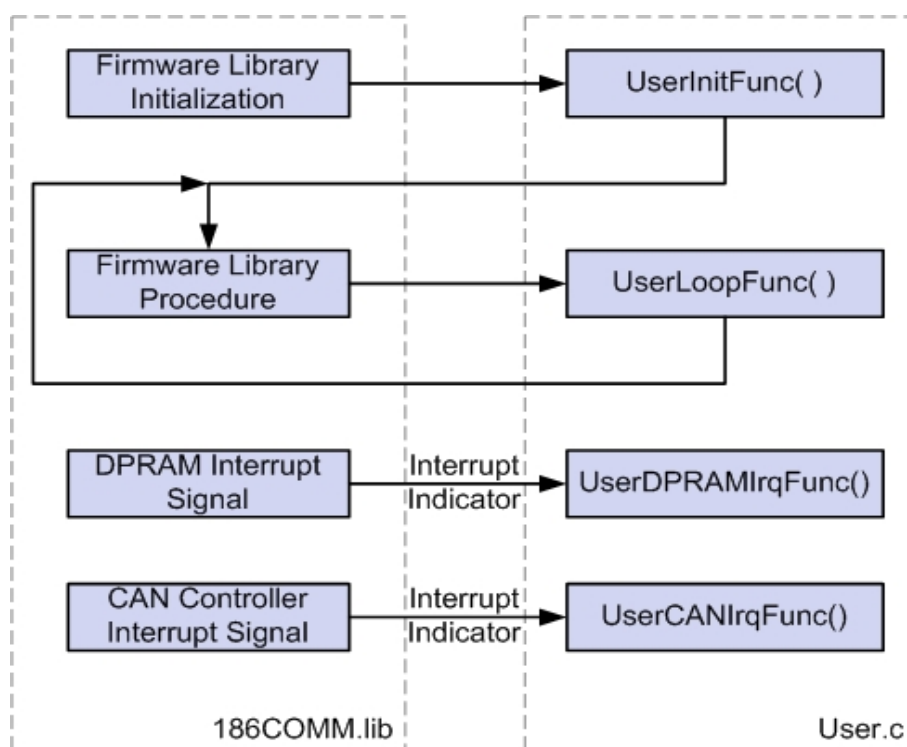
Note: If users' problem can't be fixed after following the recommended methods. Please contact your distributor or email to service@icpdas.com to solve the problem.

5 Functions of Firmware Library

If the default firmware is used, users do not need to read this chapter. This chapter introduces all the functions provided by the firmware library, 186COMM.lib. The content includes the function introduction, error code description, and simple method of troubleshooting. The section 5.1 shows the list and the information of all functions supported by the 186COMM.lib. The section 5.2 shows how to solve the problem if some error occurs.

5.1 Firmware Library Definitions and Descriptions

The firmware library is needed when designing the user-defined firmware. In order to reduce the development cycle, the firmware library provides 4 callback functions. If some initial task need to be run, put the codes into the function UserInitFunc(). The users' loop procedure can be put in the UserLoopFunc(). These callback functions are run as soon as possible while the user-defined firmware boots up. If users would like to process the interrupt indicators from the DPRAM or CAN controller, use callback functions, UserDPRAMIrqFunc() and UserCANIrqFunc(). These 4 callback functions must be applied once in users' .c file even the content of these functions are empty. The architecture is show as following figure.



The 186COMM.lib also supports the functions to handle the hardware of the CM100s board, such as the DPRAM, EEPROM, NVRAM, LEDs, real time clock, timer, debug port, and the CAN interface. Users can use Borland C/C++ or Turbo C/C++ to compile the user-defined firmware, and the Turbo C++ 1.01 or Turbo C 2.0 can be free downloaded in the website <http://dn.codegear.com/museum>. All the functions are listed in the table 5.1 and the details for every function is presented in the following sub-section.

Function definition	Page
void L1Off(void)	97
void L1On(void)	97
void L2Off(void)	98
void L2On(void)	98
void DPRAMInttToHost(char InttValue)	99
void UserDPRAMIrqFunc(unsigned char INTT)	100
int DPRAMWriteByte(unsigned int Address, unsigned char Data)	101
int DPRAMWriteWord(unsigned int Address, unsigned int Data)	102
int DPRAMWriteDword(unsigned int Address, unsigned long Data)	103
int DPRAMWriteMultiByte(unsigned int Address, char *Data, unsigned int DataNum)	104
int DPRAMReadByte(unsigned int Address, unsigned char *Data)	105
int DPRAMReadWord(unsigned int Address, unsigned int *Data)	106
int DPRAMReadDword(unsigned int Address, unsigned long *Data)	107
int DPRAMReadMultiByte(unsigned int Address, char *Data, unsigned int DataNum)	108
int DPRAMMemset(unsigned int Address, char data, unsigned int DataNum)	109
int DPRAMReceiveCmd(char *Data, unsigned int *DataNum)	110
int DPRAMSendCmd(char *Data, unsigned int DataNum)	111
	112
int GetKbhit(void)	113
int Print(const char *fmt, ...)	114
void GetTime(int *hour, int *minute, int *sec)	115
int SetTime(int hour, int minute, int sec)	116
void GetDate(int *year, int *month, int *day)	117
int SetDate(int year, int month, int day)	118
int GetWeekDay(void)	119
int ReadNVRAM(int Address)	120
int WriteNVRAM(int Address, int data)	121
unsigned long GetTimeTicks100us(void)	122

Function definition	Page
long GetTimeTicks(void)	123
void DelayMs(unsigned int DelayTime_ms)	124
void CM100_InstallUserTimer(void (*Fun)(void))	125
void T_StopWatchStart(STOPWATCH *sw)	126
unsigned long T_StopWatchGetTime(STOPWATCH *sw)	126
void T_StopWatchPause(STOPWATCH *sw)	126
void T_StopWatchContinue(STOPWATCH *sw)	126
void T_CountDownTimerStart(COUNTDOWNTIMER *cdt, unsigned long timems)	128
void T_CountDownTimerPause(COUNTDOWNTIMER *cdt)	128
void T_CountDownTimerContinue(COUNTDOWNTIMER *cdt)	128
int T_CountDownTimerIsTimeUp(COUNTDOWNTIMER *cdt)	128
unsigned long T_CountDownTimerGetTimeLeft(COUNTDOWNTIMER *cdt)	128
int CM100_EEPROMReadByte(unsigned int Block, unsigned int Address, unsigned char *Data)	130
int CM100_EEPROMReadMultiByte(unsigned int Block, unsigned int Address, char *Data, unsigned int DataNum)	131
int CM100_EEPROMWriteByte(unsigned int Block, unsigned int Address, unsigned char Data)	132
int CM100_EEPROMWriteMultiByte(unsigned int Block, unsigned int Address, char *Data, unsigned int DataNum)	133
void UserCANIrqFunc(unsigned char INTT)	134
void SJA1000HardwareReset(void)	135
int SetCANBaud(unsigned long Baud, char BT0, char BT1)	136
void GetCANBaud(unsigned long *Baud, char *BT0, char *BT1)	137
int SetCANMask(long AccCode, long AccMask)	138
void GetCANMask(long *AccCode, long *AccMask)	140
int CANConfig(unsigned long Baud, char BT0, char BT1, long AccMask, long AccCode)	141
int CANConfigBySJA1000Reg(char BaudType, char BT0, char BT1, char *AccCode, char *AccMask);	142
void EnableSJA1000(void)	143
void DisableSJA1000(void)	143
int GetCANStatus(void)	144
void ClearDataOverrunStatus(void)	145
int SendCANMsg(char Mode, unsigned long MsgID, char RTR, char DataLen, char *Data)	146
void ClearTxSoftBuffer(void)	147
int GetCANMsg(char *Mode, unsigned long *MsgID, char *RTR, char *DataLen, char *Data, unsigned long *UpperTime, unsigned long *LowerTime)	148
void ClearRxSoftBuffer(void)	150
int RxMsgCount(void)	150

Function definition	Page
int CheckTxStatus(void)	151
int AddCyclicTxMsg(char Mode, unsigned long MsgID, char RTR, char DataLen, char *Data, unsigned long TimePeriod, unsigned long TransmitTimes, unsigned char *Handle)	152
int DeleteCyclicTxMsg(unsigned char Handle)	154
int EnableCyclicTxMsg(unsigned char Handle)	155
int EnableCyclicTxMsgWithTimes(unsigned char Handle, unsigned long TransmitTimes)	156
int GetRestCyclicTxCnt(unsigned char Handle, unsigned long *RestTimes)	157
int DisableCyclicTxMsg(unsigned char Handle)	158
void ResetCyclicTxBuf(void)	158
void SystemHardwareReset(void)	159
void SystemInit(void)	159
int GetLibVer(void)	160
void RefreshWDT(void)	160
void UserInitFunc(void)	161
void UserLoopFunc(void)	162

5.1.1 L1Off

- **Description:**

Turn off the red LED of the CM100s board.

- **Syntax:**

void L1Off(void)

- **Parameter:**

None

- **Return:**

None

5.1.2 L1On

- **Description:**

Turn on the red LED of the CM100s board.

- **Syntax:**

void L1On(void)

- **Parameter:**

None

- **Return:**

None

5.1.3 L2Off

- **Description:**

Turn off the green LED of the CM100s board.

- **Syntax:**

void L2Off(void)

- **Parameter:**

None

- **Return:**

None

5.1.4 L2On

- **Description:**

Turn on the green LED of the CM100s board.

- **Syntax:**

void L1Off(void)

- **Parameter:**

None

- **Return:**

None

5.1.5 DPRAMInttToHost

- **Description:**

Call this function to signal the users' program of the PC by an interrupt. When users' program of the PC receives the interrupt indicator from the user-defined firmware, check the value of the indicator to know the meaning of the interrupt. The user-defined firmware can communicate with the program of the PC by the pre-definition of the interrupt indicators. Because of the interrupt mechanism, calling this function frequently increases the PC CPU loading and disturbs the normal procedure of the program of the PCs.

- **Syntax:**

```
void DPRAMInttToHost(char InttValue)
```

- **Parameter:**

InttValue: [input] The interrupt indicator sent to the users' program of the PC. The range is 0x00 ~ 0xdf.

- **Return:**

None

5.1.6 UserDPRAMIrqFunc *<must be called once >*

- **Description:**

This is a callback function, and must be call once in the user-defined firmware. When the firmware library receives an interrupt indicator from the users' program of the PC, the interrupt indicator will be passed to this function. Users can run the proper procedures in this function to process each interrupt indicator. It is not allowed to put an infinite loop into this function. Users must keep the program of this function as short as possible.

- **Syntax:**

```
void UserDPRAMIrqFunc(unsigned char INTT)
```

- **Parameter:**

INTT: [input] The interrupt indicator from the users' program of the PC.

- **Return:**

None

5.1.7 DPRAMWriteByte

- **Description:**

Write one byte data into the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int DPRAMWriteByte(unsigned int Address, unsigned char Data)
```

- **Parameter:**

Address: [input] The specified address of DPRAM where users want to write data. The legal address range for the DPRAM is 0 ~ 6999.

Data: [input] The byte data written to the DPRAM.

- **Return:**

_NO_ERR: OK

_DPRAM_OVER_RANGE: The parameter Address is out of range.

5.1.8 *DPRAMWriteWord*

- **Description:**

Write one word data into the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int DPRAMWriteWord(unsigned int Address, unsigned int Data)
```

- **Parameter:**

Address: [input] The specified address of DPRAM where users want to write data. The legal address range for the DPRAM is 0 ~ 6998.

Data: [input] The word data written to the DPRAM.

- **Return:**

_NO_ERR: OK

_DPRAM_OVER_RANGE: The parameter Address is out of range.

5.1.9 *DPRAMWriteDword*

- **Description:**

Write one double-word data into the specified address of the DPRAM of the CM100s board.

- **Syntax:**

int DPRAMWriteDword(unsigned int Address, unsigned long Data)

- **Parameter:**

Address: [input] The specified address of DPRAM where users want to write data. The legal address range for the DPRAM is 0 ~ 6996.

Data: [input] The double-word data written to the DPRAM.

- **Return:**

_NO_ERR: OK

_DPRAM_OVER_RANGE: The parameter Address is out of range.

5.1.10 *DPRAMWriteMultiByte*

- **Description:**

Write multi-byte data into the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int DPRAMWriteMultiByte(unsigned int Address, char *Data,  
                        unsigned int DataNum)
```

- **Parameter:**

Address: [input] The specified start address of DPRAM where users want to write data. The sum of the parameters Address and DataNum can't exceed the value 6999.

*Data: [input] The start address of a byte array written to the DPRAM.

DataNum: [input] The byte numbers of an data array written to the DPRAM.

- **Return:**

_NO_ERR: OK

_DPRAM_OVER_RANGE: The parameter Address is out of range.

5.1.11 *DPRAMReadByte*

- **Description:**

Read one byte data from specified the address of the DPRAM of the CM100s board.

- **Syntax:**

```
int DPRAMReadByte(unsigned int Address, unsigned char *Data)
```

- **Parameter:**

Address: [input] The specified address of DPRAM where users want to read data. The legal address range for the DPRAM is 0 ~ 6999.

*Data: [output] The address of a variable used to receive the data from the DPRAM.

- **Return:**

_NO_ERR: OK

_DPRAM_OVER_RANGE: The parameter Address is out of range.

5.1.12 *DPRAMReadWord*

- **Description:**

Read one word data from the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int DPRAMReadWord(unsigned int Address, unsigned int *Data)
```

- **Parameter:**

Address: [input] The specified address of DPRAM where users want to read data. The legal address range for the DPRAM is 0 ~ 6998.

*Data: [output] The address of a variable applied to receive the data from the DPRAM.

- **Return:**

_NO_ERR: OK

_DPRAM_OVER_RANGE: The parameter Address is out of range.

5.1.13 *DPRAMReadDword*

- **Description:**

Read one double-word data from the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int DPRAMReadDword(unsigned int Address, unsigned long *Data)
```

- **Parameter:**

Address: [input] The specified address of DPRAM where users want to read data. The legal address range for the DPRAM is 0 ~ 6996.

*Data: [output] The address of a variable applied to receive the data from the DPRAM.

- **Return:**

_NO_ERR: OK

_DPRAM_OVER_RANGE: The parameter Address is out of range.

5.1.14 *DPRAMReadMultiByte*

- **Description:**

Write the multi-byte data into the specified address of the DPRAM of the CM100s board.

- **Syntax:**

```
int DPRAMReadMultiByte(unsigned int Address, char *Data,  
                        unsigned int DataNum)
```

- **Parameter:**

Address: [input] The specified start address of DPRAM where users want to read data. The sum of the parameters Address and DataNum can't exceed the value 6999.

*Data: [output] The start address of a byte array applied to receive the data from the DPRAM.

DataNum: [input] The byte numbers which users want to read from the DPRAM.

- **Return:**

_NO_ERR: OK

_DPRAM_OVER_RANGE: The parameter Address is out of range.

5.1.15 *DPRAMMemset*

- **Description:**

Sets the multi-byte DPRAM data to be the specified value.

- **Syntax:**

```
int DPRAMMemset(unsigned int Address, char data,  
                unsigned int DataNum)
```

- **Parameter:**

Address: [input] The specified start address of DPRAM where users want to write data. The sum of the parameters Address and DataNum can't exceed the value 6999.

Data: [input] The data written to the DPRAM.

DataNum: [input] The byte numbers which users want to write to the DPRAM.

- **Return:**

_NO_ERR: OK

_DPRAM_OVER_RANGE: The parameter Address is out of range.

5.1.16 *DPRAMReceiveCmd*

- **Description:**

Use this function to receive the command from the program of the PC. When users use the function `CM100_SendCmd()` to send a command in the program of the PC, call this function to receive the command from the users' program. If this function is not called until another new command comes from the users' program, the former one will be covered by the latter one.

- **Syntax:**

```
int DPRAMReceiveCmd(char *Data, unsigned int *DataNum)
```

- **Parameter:**

***Data:** [output] The start address of a byte array is applied to receive the command data from the DPRAM.

***DataNum:** [output] The address of a variable is applied to receive the command length.

- **Return:**

`_NO_ERR`: OK

`_NO_DPRAM_CMD`: There is no command transmitted from user-defined firmware.

`_DPRAM_OVER_RANGE`: The command length is over 512 bytes.

5.1.17 *DPRAMSendCmd*

- **Description:**

Call this function to send the command to the program of the PC. The maximum command length is 512 bytes. Afterwards, users can use the function `CM100_ReceiveCmd()` of `CM100.dll` to get this command.

- **Syntax:**

`int DPRAMSendCmd(char *Data, unsigned int DataNum)`

- **Parameter:**

`*Data`: [input] The start address of a byte array of a sent command.

`DataNum`: [input] The word value indicates the length of the command from the user-defined firmware. The maximum value is 512 bytes.

- **Return:**

`_NO_ERR`: OK

`_DPRAM_OVER_RANGE`: The command length is out of range.

5.1.18 **DebugPrint** <assist with CM100_DEBUG_MONITOR.EXE>

- **Description:**

This function is used to debug the user-defined firmware. Call this function to send debug the messages from the user-defined firmware to the CM100_DEBUG_MONITOR.exe. The use method of this function is similar with the standard C function printf(). When users use this function, it is necessary to execute the CM100_DEBUG_MONITOR program and active the CM100s board. If the CM100s board has been activated by other Windows programs, users don't need to activate the CM100s board in the CM100_DEBUG_MONITOR.exe again. For the more information about the CM100_DEBUG_MONITOR, please refer to chapter 6. The maximum string length can't be more than 100 bytes.

- **Syntax:**

```
int DebugPrint(const char *fmt,...)
```

- **Parameter:**

* fmt: [input] The string of the debug information. The maximum length of *fmt string is 100 bytes. Please refer to the standard C function printf() to know how to use this parameters. If users need print in the new line, add “\r\n” in the end of the string of the debug information.

- **Return:**

_NO_ERR: OK

_DPRAM_OVER_RANGE: The string of the debug information length is over 100 bytes.

5.1.19 **GetKbhit** <assist with debug cable and 7188xw.exe>

- **Description:**

This function is used to debug the user-defined firmware. Call this function to get a character keyed from the keyboard. The function GetKbhit() is similar with the standard C function kbhit(). When users connect the debug port of the CM100s board with the available PC RS-232 port via the debug cable shown in the chapter 2 and execute the program 7188xw.exe, any key action to the 7188xw.exe can be caught by this function.

- **Syntax:**

int GetKbhit(void)

- **Parameter:**

None

- **Return:**

The return code is the received character which is keyed in the 7188xw.exe.

5.1.20 **Print** <assist with debug cable and 7188xw.exe>

- **Description:**

This function is used to debug the user-defined firmware. Call this function to send the debug information to the 7188xw.exe. The function Print() is similar with standard C function printf(). When users connect the debug port of the CM100s board with the available PC RS-232 port via the debug cable shown in the chapter 2 and execute the program 7188xw.exe, the debug information sent by this function will be put to the 7188xw.exe.

- **Syntax:**

```
int Print(const char *fmt, ...)
```

- **Parameter:**

* fmt: [input] The data format of keyboard input. Please refer to the standard C function printf() to know how to use this parameters.

- **Return:**

If it is successful, the return code is a non-zero value.

5.1.21 *GetTime*

- **Description:**

Use this function to get the current time from the real time clock.

- **Syntax:**

```
void GetTime(int *hour, int *minute, int *sec)
```

- **Parameter:**

*hour: [output] The address of a variable used to receive the hour value of the current time.

*minute: [output] The address of a variable used to receive the minute value of the current time.

*sec: [output] The address of a variable used to receive the second value of the current time.

- **Return:**

None.

5.1.22 *SetTime*

- **Description:**

Use this function to modify the time of the real time clock.

- **Syntax:**

int SetTime(int hour, int minute, int sec)

- **Parameter:**

hour: [input] The hour value set to the real time clock.

minute: [input] The minute value set to the real time clock.

sec: [input] The second value set to the real time clock.

- **Return:**

_NO_ERR: OK

_SET_TIME_ERROR: The input value of hour, minute or sec is invalid.

5.1.23 *GetDate*

- **Description:**

Use this function to get the current date from the real time clock.

- **Syntax:**

```
void GetDate(int *year, int *month, int *day)
```

- **Parameter:**

*year: [output] The address of a variable used to receive the year value of the current date.

*month: [output] The address of a variable used to receive the month value of the current date.

*day: [output] The address of a variable used to receive the day value of the current date.

- **Return:**

None.

5.1.24 *SetDate*

- **Description:**

Use this function to modify the date of the real time clock.

- **Syntax:**

int SetDate(int year, int month, int day)

- **Parameter:**

year: [input] The year value set to the real time clock.

month: [input] The month value set to the real time clock.

day: [input] The day value set to the real time clock.

- **Return:**

_NO_ERR: OK

_SET_DATE_ERROR: The input value of year, month or day is invalid.

5.1.25 *GetWeekDay*

- **Description:**

Use this function to obtain what day is today.

- **Syntax:**

int GetWeekDay(void)

- **Parameter:**

None.

- **Return:**

Return Code	Meaning
0	Sunday
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday

5.1.26 *ReadNVRAM*

- **Description:**

Use this function to get one-byte data of the NVRAM.

- **Syntax:**

int ReadNVRAM(int Address)

- **Parameter:**

Address: [input] The NVRAM address where users want to read the data. The range of this parameter is 0 ~ 30.

- **Return:**

_ACCESS_NVRAM_FAILE: The address of NVRAM is invalid.

Others: The value obtained from the NVRAM. The range of return value is 0 ~ 255.

5.1.27 WriteNVRAM

- **Description:**

Use this function to write one-byte data to specified address of the NVRAM. If the system has no power, the data stored in the NVRAM will not disappear until the Li battery is dead.

- **Syntax:**

int WriteNVRAM(int Address, int data)

- **Parameter:**

Address: [input] The NVRAM address where users want to write the data to. The range of this parameter is 0 ~ 30.

data: [input] The data written to the NVRAM. The range of this parameter is 0 ~ 255. If value is over 255, only low byte of data will be written to the NVRAM.

- **Return:**

_NO_ERR: OK.

_ACCESS_NVRAM_FAILE: The address of the NVRAM is invalid.

5.1.28 *GetTimeTicks100us*

- **Description:**

Read the time ticks of the CM100s board by using this function. While the board is powered, the time ticks start to be counted. Firmware reset will clean the accumulated counters of this value. If the accumulated counter is over 0xFFFFFFFF, it starts from the value 0.

- **Syntax:**

unsigned long GetTimeTicks100us(void)

- **Parameter:**

None.

- **Return:**

The time ticks value calculated as the firmware starts. The unit is 0.1 ms.

5.1.29 *GetTimeTicks*

- **Description:**

Call this function to read CM100s board time ticks. While the board is powered, the time ticks start to be counted. This function can't be called in the interrupt service routine. If the accumulated counter is over 0xFFFFFFFF, it starts from the value 0.

- **Syntax:**

long GetTimeTicks(void)

- **Parameter:**

None

- **Return:**

The time ticks value calculated as the firmware starts. The unit is 1 ms.

5.1.30 DelayMs

- **Description:**

Use this function to delay some time for the procedure of the user-defined firmware. Because of the watchdog mechanism, users can't delay for a long time. The watchdog is triggered every 800 ms. It is recommended that if users want to delay the procedure of the user-defined firmware more than 500 ms. The function RefreshWDT() must be applied to avoid the watchdog timeout. This function is not allowed to put in the interrupt service routine. If users want to use delay functions in interrupt service routine, it is strongly recommended to move this part of the codes from the interrupt service routine into the UserLoopFunc().

- **Syntax:**

```
void DelayMs(unsigned int DelayTime_ms)
```

- **Parameter:**

DelayTime_ms: [input] The delay time. The unit is 1 ms.

- **Return:**

None

5.1.31 CM100_InstallUserTimer

- **Description:**

This function offers the timer interrupt. When users put their timer interrupt service routine in this function, this interrupt service routine will be executed every predefined period. Be careful that too many codes in the interrupt service routine will disturb the normal procedure of the user-defined firmware.

- **Syntax:**

```
void CM100_InstallUserTimer(void (*Fun)(void))
```

- **Parameter:**

(*Fun)(void): [input] The pointer which points a function with format “void XXX(void)”. The characters XXX are the name of the users' interrupt service routine.

- **Return:**

None

5.1.32 *T_StopWatchXXX series functions*

- **Description:**

Call this function to use a stopwatch. There are 4 functions for the stopwatch operations. When users want to start a stopwatch, T_StopWatchStart() must be applied. Afterwards, users can use the function T_StopWatchGetTime() to get the current time counted by this stopwatch. If users want to stop time counting, use the function T_StopWatchPause(). Calling the function T_StopWatchContinue() can enable this timer again. If users want to use more than one stopwatch, just input the different structure variable STOPWATCH to these 4 functions. One structure variable is mapped to one stopwatch. The time unit of these 4 functions and the members of the STOPWATCH structure are millisecond.

- **Syntax:**

```
void T_StopWatchStart(STOPWATCH *sw)

unsigned long T_StopWatchGetTime(STOPWATCH *sw)

void T_StopWatchPause(STOPWATCH *sw)

void T_StopWatchContinue(STOPWATCH *sw)
```

- **Parameter:**

*sw: [output] The address of a STOPWATCH structure variable applied to describe the stopwatch. The member of the STOPWATCH structure is shown as following:

```
typedef struct {

    unsigned long ulStart;

    unsigned long ulPauseTime;

    unsigned int uMode;

}STOPWATCH;
```

The parameter `ulStart` shows the start time of stopwatch. The parameter `ulPauseTime` returns the time as the stopwatch is paused. The parameter `uMode` returns the status of the stopwatch. If `uMode` is 0, the stopwatch is paused. If `uMode` is 1, the stopwatch is running.

- **Return:**

The return code of the function `T_StopWatchGetTime()` is the current time count after the stopwatch started.

5.1.33 *T_CountDownTimerXXX series functions*

- **Description:**

Call this function to use a countdown timer. There are 5 functions for the countdown timer operations. When users want to start a countdown timer, the function `T_CountDownTimerStart()` must be applied. Afterwards, If users want to stop the countdown timer, use the function `T_CountDownTimerPause()`. Calling the function `T_CountDownTimerContinue()` can enable this countdown timer again. Users can use the function `T_CountDownTimerIsTimeUp()` to check if the countdown timer is timeout or not. Or, use the function `T_CountDownTimerGetTimeLeft()` to obtain the rest time of the countdown timer. If users want to use more than one countdown timer, just input the different structure variable `COUNTDOWNTIMER` to these 5 functions. One structure variable is mapped to one countdown timer. The time unit of these 5 functions and the members of the `COUNTDOWNTIMER` structure are millisecond.

- **Syntax:**

```
void T_CountDownTimerStart(COUNTDOWNTIMER *cdt,
                           unsigned long timems)
void T_CountDownTimerPause(COUNTDOWNTIMER *cdt)
void T_CountDownTimerContinue(COUNTDOWNTIMER *cdt)
int T_CountDownTimerIsTimeUp(COUNTDOWNTIMER *cdt)
unsigned long T_CountDownTimerGetTimeLeft(
                           COUNTDOWNTIMER *cdt)
```

- **Parameter:**

`timems`: [input] The time interval which indicates that how much time the countdown timer to count down.

`*cdt`: [output] The address of a `COUNTDOWNTIMER` structure variable used to describe the countdown timer. The member of the `COUNTDOWNTIMER` structure is shown as following:

```
typedef struct {
    unsigned long ulTime;
    unsigned long ulStartTime;
```

```
        unsigned long ulPauseTime;  
        unsigned int uMode;  
    } COUNTDOWNTIMER;
```

The parameter `ulTime` is the time interval of the countdown timer. The parameter `ulStartTime` returns the start time of the countdown timer. The parameter `ulPauseTime` obtains the time as the countdown timer is paused. The parameter `uMode` returns the status of the countdown timer. If `uMode` is 0, it means that the countdown timer is stopped. If `uMode` is 1, the countdown timer is running.

- **Return:**

The return code of the `T_CountDownTimerIsTimeUp()` is `_NO_ERR` or `_COUNT_DOWN_TIMER_TIME_UP`. If the countdown timer is timeout, the return code is `_COUNT_DOWN_TIMER_TIME_UP`. If not, the return code is `_NO_ERR`. The return code of the `T_CountDownTimerGetTimeLeft()` is the rest time of the countdown timer.

5.1.34 CM100_EEPROMReadByte

- **Description:**

Use this function to read the data of the specified address of the EEPROM.

- **Syntax:**

```
int CM100_EEPROMReadByte(unsigned int Block,  
                           unsigned int Address,  
                           unsigned char *Data)
```

- **Parameter:**

Block: [input] The EEPROM block No.. The range is 0 ~ 6.

Address: [input] The EEPROM address where users want to read the data from. Each block has 256 bytes. Therefore, the range of this parameter is 0 ~ 255.

*data: [output] The address of a variable used to obtain the data of specified address of the EEPROM

- **Return:**

_NO_ERR: OK.

_EEPROM_OVER_RANGE: The parameter Block is over 6, or the parameter Address is over 255.

5.1.35 CM100_EEPROMReadMultiByte

- **Description:**

Use this function to read some data from the EEPROM.

- **Syntax:**

```
int CM100_EEPROMReadMultiByte(unsigned int Block,  
                                unsigned int Address,  
                                char *Data,  
                                unsigned int DataNum)
```

- **Parameter:**

Block: [input] The EEPROM block No.. The range is 0 ~ 6.

Address: [input] The start EEPROM address where users want to write the data to. Each block has 256 bytes. Therefore, the range of this parameter is 0 ~ 255.

*data: [output] The start address of a byte array used to receive the data from the EEPROM

DataNum: [input] The parameter indicates that how many data users want to obtain.

- **Return:**

_NO_ERR: OK.

_EEPROM_OVER_RANGE: The parameter Block is over 6, the parameter Address is over 256, or the specified range of reading data is over the block 6 and address 255.

5.1.36 CM100_EEPROMWriteByte

- **Description:**

Use this function to write the data to the specified address of the EEPROM. If the system has no power, the data stored in EEPROM will not disappear, but the EEPROM has the limitation for erase or write cycles. The data which need to frequently write to memory is not proper to be used in the EEPROM.

- **Syntax:**

```
int CM100_EEPROMWriteByte(unsigned int Block,  
                           unsigned int Address,  
                           unsigned char Data)
```

- **Parameter:**

Block: [input] The EEPROM block No.. The range is from 0 to 6.

Address: [input] The EEPROM address where users want to write the data to. Each block has 256 bytes. Therefore, the range of this parameter is 0 ~ 255.

data: [input] The data written to the EEPROM

- **Return:**

_NO_ERR: OK.

_EEPROM_ACCESS_ERROR: Can't write data to the specified EEPROM address. The EEPROM may be damaged.

_EEPROM_OVER_RANGE: The block No. is over 6, or the address is over 256.

5.1.37 CM100_EEPROMWriteMultiByte

- **Description:**

Use this function to write some data to specified address of EEPROM. If the system has no power, the data stored in EEPROM will not disappear, but the EEPROM has the limitation for erase or write cycles. The data which need to frequently write to memory is not proper to be used in the EEPROM.

- **Syntax:**

```
int CM100_EEPROMWriteMultiByte(unsigned int Block,  
                                unsigned int Address,  
                                char *Data,  
                                unsigned int DataNum)
```

- **Parameter:**

Block: [input] The EEPROM block No.. The range is from 0 to 6.

Address: [input] The EEPROM address where users want to write the data to. Each block has 256 bytes. Therefore, the range of this parameter is 0 ~ 255.

*data: [output] The start address of a byte array used to store the data written to the EEPROM.

DataNum: [input] The parameter indicates that how many data users want to write.

- **Return:**

_NO_ERR: OK.

_EEPROM_ACCESS_ERROR: Can't write data to specified EEPROM address. The EEPROM may be damaged.

_EEPROM_OVER_RANGE: The parameter Block is over 6, the parameter Address is over 256, or the specified range of writing data is over the block 6 and address 255.

5.1.38 UserCANIrqFunc <must be called once>

- **Description:**

This is a callback function, and must be call once in the user-defined firmware. When the firmware library receives an interrupt signal from the CAN controller, this function will be called. The interrupt indicator which shows what kind of the CAN controller interrupt is activated is passed to this function. Users are able to design their interrupt routine according to the corresponding interrupt indicators. It is not allowed to put an infinite loop in to this function, and users must keep the codes of this function as short as possible.

- **Syntax:**

```
void UserCANIrqFunc(unsigned char INTT)
```

- **Parameter:**

INTT: [input] The interrupt indicator from the CAN controller. The meanings of the indicators are shown below.

Indicator (Hex)	Meaning
0x01	Receive a message successfully
0x02	Transmit a message successfully
0x04	Error warring
0x08	Data Overrun
0x10	CAN controller wake-up
0x20	Bus Passive
0x40	Arbitration Lost
0x80	Bus Error

- **Return:**

None

5.1.39 SJA1000HardwareReset

- **Description:**

Resets the CAN controller by the reset pin of SJA1000. After calling this function, users must configure the baud and message filter of the CAN controller. Then, use EnableSJA1000() to activate the SJA1000 to send and receive CAN messages.

- **Syntax:**

```
void SJA1000HardwareReset(void)
```

- **Parameter:**

None

- **Return:**

None

5.1.40 SetCANBaud

- **Description:**

Sets the CAN baud of the CAN controller.

- **Syntax:**

int SetCANBaud(unsigned long Baud, char BT0, char BT1)

- **Parameter:**

Baud: [input] The baud of the CAN controller. There are 12 kinds of predefined baud, 5 k, 10 k, 20 k, 25 k, 50 k, 100 k, 125 k, 200 k, 250 k, 500 k, 800 k, and 1 M bps. If these bauds cannot satisfy users, set this parameter 12 and define the BT0 and BT1 of the SJA1000.

Value	Description
0	5 k bps
1	10 k bps
2	20 k bps
3	25 k bps
4	50 k bps
5	100 k bps
6	125 k bps
7	200 k bps
8	250 k bps
9	500 k bps
10	800 k bps
11	1000 k bps
others	The user-defined baud (The BT0 and BT1 are needed)

BT0: [input] The user-defined baud.

BT1: [input] The user-defined baud. For the more information about how to set the BT0 and BT1, please refer to the datasheet of the SJA1000.

- **Return:**

_NO_ERR: OK.

_CAN_CHIP_SOFT_RESET_ERR: The SJA1000 can't be reset by the software. The CAN controller may be damaged.

5.1.41 *GetCANBaud*

- **Description:**

Gets the current CAN baud of the CAN controller.

- **Syntax:**

```
void GetCANBaud(unsigned long *Baud, char *BT0, char *BT1)
```

- **Parameter:**

*Baud: [output] The address of a variable used to obtain the baud of the CAN controller. If this parameter is more than 12, the BT0 and BT1 are useful. Please refer to the function SetCANBaud() for the details.

*BT0: [output] The address of a variable used to get the BT0 value obtained from the SJA1000.

*BT1: [output] The address of a variable used to get the BT1 value obtained from the SJA1000. For more information about how to use the BT0 and BT1, please refer to the datasheet of the SJA1000.

- **Return:**

None

5.1.42 SetCANMask

- **Description:**

Sets the message filter of the CAN controller.

- **Syntax:**

```
int SetCANMask(long AccCode, long AccMask)
```

- **Parameter:**

AccCode: [input] Acceptance code of CAN controller

AccMask: [input] Acceptance mask of CAN controller.

The AccCode is used to decide which CAN IDs are accepted by the CAN controller. The AccMask is used to decide which bit of the CAN IDs are checked with the AccCode by the CAN controller. If the bit of AccMask is set to 0, it means that the bit in the same position of the CAN IDs need to be checked and the ID bit value needs to match the bit of AccCode in the same position.

AccCode and AccMask Definition For 11-bit ID:

AccCode and AccMask	Bit Position	Filter Target
high byte of the high word	bit7~bit0	bit10 ~ bit3 of ID
low byte of the high word	bit7~bit5	bit2 ~ bit0 of ID
low byte of the high word	bit4	RTR
low byte of the high word	bit3~bit0	no use
high byte of the low word	bit7~bit0	bit7 ~ bit0 of 1st byte data
low byte of the low word	bit7~bit0	bit7 ~ bit0 of 2nd byte data

AccCode and AccMask Definition For 29-bit ID:

AccCode and AccMask	Bit Position	Filter Target
high byte of the high word	bit7~bit0	bit28~ bit21 of ID
low byte of the high word	bit7~bit0	bit20 ~ bit13 of ID

high byte of the low word	bit7~bit0	bit12 ~ bit5 of ID
low byte of the low word	bit7~bit3	bit4 ~ bit0 of ID
low byte of the low word	bit2	RTR
low byte of the low word	bit1~bit0	no use

Example for 29 bit ID message:

	Array[0]	Array[1]	Array[2]	Array[3]
AccCode :	00h	00h	00h	A0h
AccMask :	FFh	FFh	FFh	1Fh
ID bit :	bit28~bit21	bit20~bit13	bit12~bit5	bit4~bit0
ID Value :	xxxx xxxx	xxxx xxxx	xxxx xxxx	101x x will be accepted

(Note: The character “x” means the bit value doesn’t care. The character “h” behind the value means the value is in hex format.)

- **Return:**

_NO_ERR: OK.

_CAN_CHIP_SOFT_RESET_ERR: The SJA1000 can’t be reset by the software. The CAN controller may be damaged.

5.1.43 *GetCANMask*

- **Description:**

Gets the current message filter configuration of the CAN controller.

- **Syntax:**

```
void GetCANMask(long *AccCode, long *AccMask)
```

- **Parameter:**

- * AccCode: [output] The address of a variable used to obtain the acceptance code of SJA1000.

- * AccMask: [output] The address of a variable used to obtain the acceptance mask of SJA1000.

- **Return:**

None

5.1.44 CANConfig

- **Description:**

Configures the baud, message filter of the CAN controller. After calling this function, users need to call the EnableSJA1000() to active the CAN controller.

- **Syntax:**

```
int CANConfig(unsigned long Baud, char BT0, char BT1, long AccMask,  
              long AccCode)
```

- **Parameter:**

Baud: [input] The baud of CAN controller.

BT0: [input] The user-defined baud.

BT1: [input] The user-defined baud.

AccCode: [input] The acceptance code of the CAN controller.

AccMask: [input] The acceptance mask of the CAN controller.

For the more information about these parameters, please refer to the functions SetCANBaud() and SetCANMask().

- **Return:**

_NO_ERR: OK.

_CAN_CHIP_SOFT_RESET_ERR: The SJA1000 can't be reset by the software. The CAN controller may be damaged.

5.1.45 CANConfigBySJA1000Reg

- **Description:**

This function is similar to the function CANConfig. The difference is the input parameters. This function uses non-structure parameters.

- **Syntax:**

```
int CANConfigBySJA1000Reg(char BaudType, char BT0,  
                           char BT1, char *AccCode,  
                           char *AccMask)
```

- **Parameter:**

BaudType: [input] The baud of CAN controller.

BT0: [input] The user-defined baud.

BT1: [input] The user-defined baud.

* AccCode: [output] The address of a variable used to obtain the acceptance code of SJA1000. The first element is the LSB of the AccCode and the forth is the MSB of the AccCode.

* AccMask: [output] The address of a variable used to obtain the acceptance mask of SJA1000. The first element is the LSB of the AccMask and the forth is the MSB of the AccMask.

For the more information about these parameters, please refer to the functions SetCANBaud() and SetCANMask().

- **Return:**

_NO_ERR: OK.

_CAN_CHIP_SOFT_RESET_ERR: The SJA1000 can't be reset by the software. The CAN controller may be damaged.

5.1.46 *EnableSJA1000*

- **Description:**

Use this function to activate the SJA1000. Afterwards, users can send/receive CAN messages by other functions.

- **Syntax:**

`void EnableSJA1000(void)`

- **Parameter:**

None

- **Return:**

None

5.1.47 *DisableSJA1000*

- **Description:**

Call the function `DisableSJA1000()` to disable the SJA1000. If users want to enable the SJA1000 again, the configuration of the SJA1000 must be don first.

- **Syntax:**

`void DisableSJA1000(void)`

- **Parameter:**

None

- **Return:**

None

5.1.48 GetCANStatus

- **Description:**

Obtains the status register of the SJA1000.

- **Syntax:**

```
int GetCANStatus(void)
```

- **Parameter:**

None

- **Return:**

The return codes are the value of the status register of the SJA1000, and are described below.

Bit NO.	Description
7 (MSB)	Bus status. 1 for bus off, 0 for bus on.
6	Error status. 1 for at least one error, 0 for OK.
5	SJA1000 Transmit status. 1 for transmitting, 0 for idle.
4	SJA1000Receive status. 1 for receiving, 0 for idle.
3	SJA1000 Transmit complete status. 1 for complete, 0 for incomplete.
2	SJA1000 Transmit buffer status. 1 for released, 0 for locked
1	Data overrun status. 1 for SJA1000 reception buffer overrun, 0 for OK.
0 (LSB)	Receive buffer status. 1 for at least one message stored in the SJA1000 reception buffer, 0 for empty.

5.1.49 *ClearDataOverrunStatus*

- **Description:**

When the data overrun status is obtained by using the function GetCANStatus(), call this function to clean this status.

- **Syntax:**

void ClearDataOverrunStatus(void)

- **Parameter:**

None

- **Return:**

None

5.1.50 *SendCANMsg*

- **Description:**

Sends a CAN message to the software transmission buffer. When the CAN bus is idle, this CAN message will be send to the CAN network.

- **Syntax:**

```
int SendCANMsg(char Mode, unsigned long MsgID, char RTR,  
               char DataLen, char *Data)
```

- **Parameter:**

Mode: [input] 0 for 11-bit message ID, 1 for 29-bit message ID.

MsgID: [input] CAN message ID.

RTR: [input] 0 for normal messages, 1 for remote-transmit-request messages. As the value is 1, the parameter *Data is useless.

DataLen: [input] The data length of a transmitted CAN message. The maximum value is 8.

*Data: [input] The start address of a buffer used to store the transmitted data of a CAN message.

- **Return:**

_NO_ERR: OK.

_SOFT_BUF_FULL: The transmission software buffer is full. Users need to transmit the CAN message later, or use the function ClearTxSoftBuffer() to clean the CAN transmission buffer.

5.1.51 *ClearTxSoftBuffer*

- **Description:**

Call this function to clean the transmission software buffer of the CAN messages.

- **Syntax:**

void ClearTxSoftBuffer(void)

- **Parameter:**

None

- **Return:**

None

5.1.52 GetCANMsg

- **Description:**

Obtain a received CAN message from the software buffer.

- **Syntax:**

```
int GetCANMsg(char *Mode, unsigned long *MsgID, char *RTR,  
              char *DataLen, char *Data, unsigned long *UpperTime,  
              unsigned long *LowerTime)
```

- **Parameter:**

*Mode: [output] The address of a variable used to get the mode of a CAN message. If the value is 0, the received CAN message is 11-bit-ID CAN message. The 29-bit-ID CAN message will have the value 1.

*MsgID: [output] The address of a variable used to get the CAN message ID.

*RTR: [output] The address of a variable used to obtain the status of this CAN message. 0 for the remote-transmit-request message, 1 for the normal message.

*DataLen: [output] The address of a variable used to obtain the data length of a CAN message. The range of this value is 0~8.

*Data: [output] The start address of a buffer used to get the data of a CAN message. Users need to put an 8-byte array in this filed.

*UpperTime: [output] The address of a variable used to obtain the higher 32-bit time stamp of a CAN message.

*LowerTime: [output] The address of a variable used to obtain the lower

32-bit time stamp of a CAN message. The unit of UpperTime and LowerTime are 0.1ms.

- **Return:**

_NO_ERR: OK.

_RX_SOFT_BUF_EMPTY: There is no message stored in the reception software buffer.

_SOFT_BUF_FULL: The reception software buffer of the CAN messages is full. Use the function ClearRxSoftBuffer() to clean the status when receiving the return code.

5.1.53 ClearRxSoftBuffer

- **Description:**

Call this function to clean the reception software buffer of the CAN messages.

- **Syntax:**

void ClearRxSoftBuffer(void)

- **Parameter:**

None

- **Return:**

None

5.1.54 RxMsgCount

- **Description:**

Call this function to know how many available CAN messages stored in the reception software buffer.

- **Syntax:**

int RxMsgCount(void)

- **Parameter:**

None

- **Return:**

The return code is the number of the CAN messages stored in the reception software buffer.

5.1.55 *ChcekTxStatus*

- **Description:**

Checks if the CAN controller successfully sends the CAN message.

- **Syntax:**

int CheckTxStatus(void)

- **Parameter:**

None

- **Return:**

The value 1 means that the CAN controller can't successfully send the CAN message to the network and the value 0 is for no error.

5.1.56 AddCyclicTxMsg

- **Description:**

Adds a cyclic transmission message into the firmware of the board. Afterwards, users can use the functions EnableCyclicTxMsg() and DelectCyclicTxMsg() to enable or disable this cyclic transmission message. The maximum addable number of the transmission messages is 5. After adding a cyclic transmission message, the handle for this message will be returned. The less value of handle indicates the higher priority of the cyclic transmission messages. If there are two cycle transmission messages need to be sent at the same time, the higher priority message will be sent first.

- **Syntax:**

```
int AddCyclicTxMsg(char Mode, unsigned long MsgID, char RTR,  
                  char DataLen, char *Data,  
                  unsigned long TimePeriod,  
                  unsigned long TransmitTimes,  
                  unsigned char *Handle)
```

- **Parameter:**

Mode: [input] 0 for 11-bit message ID, 1 for 29-bit message ID.

MsgID: [input] The CAN message ID.

RTR: [input] 0 for transmitting the remote-transmit-request message, 1 for transmitting the normal message. As this bit is 1, the parameter *Data is useless.

DataLen: [input] The data length of the CAN message. The maximum value is 8.

*Data: [input] The start address of the data array of a CAN message. The maximum length of the Data array is 8 bytes.

TimePeriod: [input] The time period for the cyclic transmission. This parameter is formatted by 0.1ms. The minimum value is 5.

TransmitTimes: [input] The transmission times for the cyclic transmission. After enabling the transmission, the message will be sent for specified times. If the value of the parameter is 0, the transmission goes cyclically until disabling the cyclic transmission.

*Handle: [output] The address of a variable is used to get the handle of a cyclic transmission. When users would like to enable or disable the specified transmission, this value is needed.

- **Return:**

_NO_ERR: OK

_CYCLIC_CONFIG_ERR: The applied cyclic transmission message number is more than 5, or the parameter time period is less than 5 (0.5ms).

5.1.57 DeleteCyclicTxMsg

- **Description:**

Removes the specified cyclic transmission message added by the function AddCyclicTxMsg().

- **Syntax:**

int DeleteCyclicTxMsg(unsigned char Handle)

- **Parameter:**

Handle: [input] The handle of the cyclic transmission message which is obtained by AddCyclicTxMsg() function.

- **Return:**

_NO_ERR: OK

_CYCLIC_HANDLE_ERR: The handle value can't be found in the cyclic transmission engine.

5.1.58 *EnableCyclicTxMsg*

- **Description:**

Enables the cyclic transmission message added by the function `AddCyclicTxMsg()`. After calling this function, the specified cyclic transmission message will be transmitted.

- **Syntax:**

`int EnableCyclicTxMsg(unsigned char Handle)`

- **Parameter:**

Handle: [input] The handle of the cyclic transmission message which is obtained by the function `AddCyclicTxMsg()`.

- **Return:**

`_NO_ERR`: OK

`_CYCLIC_HANDLE_ERR`: The handle value can't be found in the cyclic transmission engine.

5.1.59 *EnableCyclicTxMsgWithTimes*

- **Description:**

Enables the cyclic transmission message added by the function `AddCyclicTxMsg()` with the specified times. After calling this function, the specified cyclic transmission message will be transmitted.

- **Syntax:**

```
int EnableCyclicTxMsgWithTimes(unsigned char Handle,  
                               unsigned long TransmitTimes)
```

- **Parameter:**

Handle: [input] The handle of the cyclic transmission message which is obtained by the function `AddCyclicTxMsg()`.

TransmitTimes: [input] The cyclic message number which will be transmitted. After enabling the transmission, the message will be sent for specified times. If the value of the parameter is 0, the transmission goes cyclically until disabling the cyclic transmission.

- **Return:**

`_NO_ERR`: OK

`_CYCLIC_HANDLE_ERR`: The handle value can't be found in the cyclic transmission engine.

5.1.60 *GetRestCyclicTxCnt*

- **Description:**

This function returns the rest number of the cyclic messages which will not be transmitted. If users configure the cyclic message with the specified times or use the function `EnableCyclicTxMsgWithTimes()` to start the cyclic transmission. Calling this function can obtain how many cyclic messages are not sent to the CAN bus.

- **Syntax:**

```
int GetRestCyclicTxCnt(unsigned char Handle,  
                      unsigned long *RestTimes)
```

- **Parameter:**

Handle: [input] The handle of the cyclic transmission message which is obtained by the function `AddCyclicTxMsg()`.

*RestTimes: [output] The rest number of the cyclic messages which will not be transmitted.

- **Return:**

_NO_ERR: OK

_CYCLIC_HANDLE_ERR: The handle value can't be found in the cyclic transmission engine.

5.1.61 *DisableCyclicTxMsg*

- **Description:**

Disable a cyclic transmission message which is enabled by EnableCyclicTxMsg() function before.

- **Syntax:**

int DisableCyclicTxMsg(unsigned char Handle)

- **Parameter:**

Handle: [input] The handle of the cyclic transmission message which is obtained by the function AddCyclicTxMsg().

- **Return:**

_NO_ERR: OK

_CYCLIC_HANDLE_ERR: The handle value can't be found in the cyclic transmission engine.

5.1.62 *ResetCyclicTxBuf*

- **Description:**

Cleans the software buffer of the cyclic transmission engine. After calling this function, the transmission of all cyclic messages are stopped, and all of the cyclic messages are removed from the cyclic transmission engine.

- **Syntax:**

void ResetCyclicTxBuf(void)

- **Parameter:**

None

- **Return:**None

5.1.63 *SystemHardwareReset*

- **Description:**

Use this function to reset the hardware of the CM100s board.

- **Syntax:**

void SystemHardwareReset(void)

- **Parameter:**

None

- **Return:**

None

5.1.64 *SystemInit*

- **Description:**

Use this function to initiate the DPRAM, LEDs, cyclic transmission engine, CAN transmission software buffer, and CAN controller.

- **Syntax:**

void SystemInit(void)

- **Parameter:**

None

- **Return:**

None

5.1.65 *GetLibVer*

- **Description:**

Gets the version of the firmware library.

- **Syntax:**

int GetLibVer(void)

- **Parameter:**

None

- **Return:**

The return code is the version of the firmware library. For example: If 100(hex) is return, it means driver version is 1.00.

5.1.66 *RefreshWDT*

- **Description:**

Call this function to refresh the watchdog timer of the CM100s board. When users design the user-defined firmware, this function must be called where the users' procedure may have a processed period more than 500ms. If the function RefreshWDT() is not called in 800ms, the CPU of the board will be reset.

- **Syntax:**

void RefreshWDT(void)

- **Parameter:**

None

- **Return:**

None

5.1.67 UserInitFunc *<must be called once>*

- **Description:**

When users design the user-defined firmware, this callback function must be used. Users can put some procedures into this function. These procedures are those which will be executed only one time in the user-defined firmware. When the CPU of the CM100s board is booting up, the firmware library will call this callback function once.

- **Syntax:**

void UserInitFunc(void)

- **Parameter:**

None

- **Return:**

None

5.1.68 UserLoopFunc *<must be called once>*

- **Description:**

When users design the user-defined firmware, this callback function must be used. Users can put their main procedures into this function. The main procedure will be cyclically executed as soon as possible. The time period is correlated with the complexity of the users' main procedure. When the CPU of the CM100s board boots up, the firmware library will call the function UserInitFunc() once and then call the function UserLoopFunc() cyclically until the CPU of the CM100s board is turned off. It is not allowed to put an infinite loop in this function.

- **Syntax:**

void UserLoopFunc(void)

- **Parameter:**

None

- **Return:**

None

5.2 Firmware Library Return Codes Troubleshooting

If the default firmware is used, users do not need to read this section.

Return Code	Error ID	Troubleshooting
-19	_SET_TIME_ERROR	1. Check the time format of the input parameters.
-18	_SET_DATE_ERROR	1. Check the date format of the input parameters.
-9	_ACCESS_NVRAM_FAILE	1. Try it again. 2. Call your distributor to solve this problem.
0	_NO_ERR	OK
1	_COUNT_DOWN_TIMER_TIME_UP	1. The countdown timer started by users is timeout.
101	_CAN_CHIP_SOFT_RESET_ERR	1. Call SJA1000HardwareReset(), and try it again. 2. Call your distributor to solve this problem.
102	_CAN_CHIP_CONFIG_ERR	1. Check the parameters of baud, BT0, BT1, acceptance code, and acceptance mask.
103	_RX_SOFT_BUF_EMPTY	1. Wait for a while and call the function again.
104	_SOFT_BUF_FULL	1. Use the function ClearTxSoftBuffer() or the function ClearRxSoftBuffer() to clear the status of the buffer overflow. 2. Reduce the bus loading of the CAN network.
105	_DPRAM_WRITE_ERR	1. Wait for a while and call the function again. 2. Call your distributor to solve this problem.
106	_DPRAM_READ_ERR	1. Wait for a while and call the function again. 2. Call your distributor to solve this problem.
107	_DPRAM_OVER_RANGE	1. Check the address or space range of the written DPRAM.
108	_NO_DPRAM_CMD	1. Wait for a while and call the function again.
109	_CYCLIC_CONFIG_ERR	1. Check if users already use 5 cyclic messages. 2. Set the parameters TimePeriod to be more than 5.
110	_CYCLIC_HANDLE_ERR	1. Check the parameter Handle.
111	_EEPROM_OVER_RANGE	1. Check the address or space range of written EEPROM.
112	_EEPROM_ACCESS_ERROR	1. Wait for a while and call the function again. 2. Call your distributor to solve this problem.

Table 5.6 Return Code Troubleshooting

Note: If users' problems can't be fixed after following the recommended methods. Please contact your distributor or email to service@icpdas.com to solve the problem.

6 Application Programming

In this chapter, the program skills about how to use the default firmware and the design user-defined firmware are shown. Section 6.1 describes the program basic architecture of an application and briefs some demo programs. Section 6.2 introduces the CANUtility tool. It is a useful free tool for monitoring and accessing the CAN network. Furthermore, when users want to update the default firmware or download user-defined firmware into the CM100s board. This tool must be used. Section 6.3 gives a profile about how to design the user-defined firmware, and the corresponding application on the Windows platform. Some demo programs for the user-defined firmware are also shown. Section 6.4 provides two ways to debug the user-defined firmware. If users just use the default firmware, the Section 6.3 and 6.4 can be ignored.

6.1 *Windows Programming With Default Firmware*

This section is only for the default firmware. It is useless if users want to design the user-defined firmware. The figure 6.1 presents the “Send CAN Message” procedure. When users want to design their application by using the APIs of CM100.dll on the Windows platform, this flowchart may be a good reference. The figure 6.2 is a standard procedure for receiving a CAN messages. This procedure let users obtain the CAN messages from the CAN bus easily. If users need to send some specified CAN messages every period of time, the flowchart shown in the figure 6.3 may give a good example. The figure 6.4 shows the flowchart to send and receive CAN messages in one process or thread. Because it is impossible for the CAN board to be accessed by more than one APIs of the CM100.dll at the same time, the flowchart of the figure 6.4 can be a good reference. Owing to these 4 flowcharts, it may satisfy most application of the users’ program of the PC with the default firmware of the CM100s board. Following these principles can help users to build their application easier and faster. When users want to design the Windows program, the functions, CM100_ActiveBoard(), CM100_Init(), and CM100_Config(), are only called once when the program starts. If the program needs to be terminated, call the function CM100_CloseBoard() once to release the system resource.

Note: It is impossible to access the same CAN board with more than two procedure or threads. Users need to confirm the completion of the API of the CM100.dll before calling a next API of the CM100.dll.

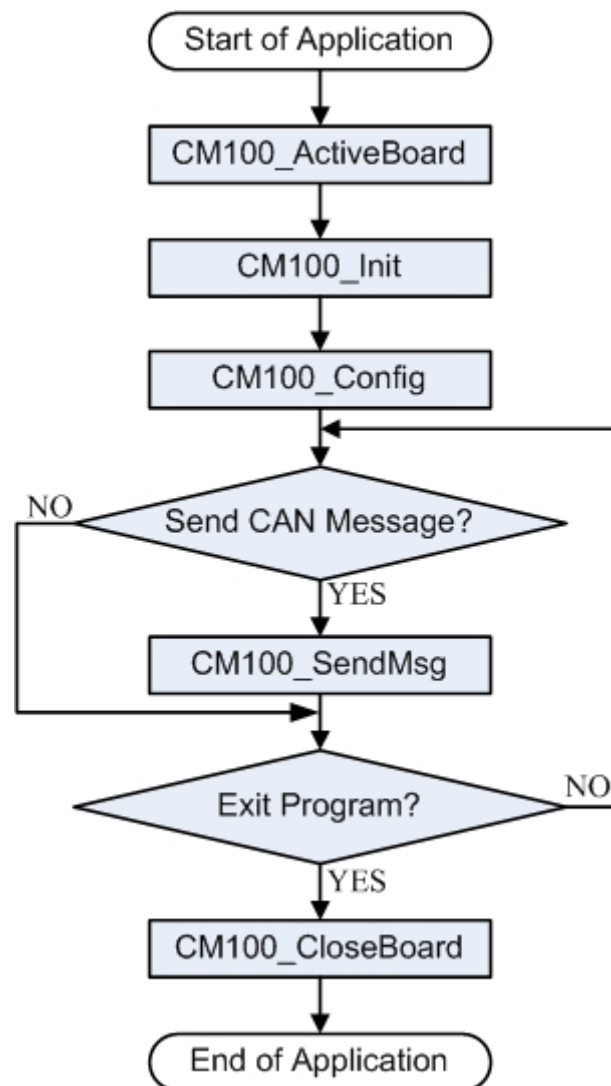


Figure 6.1 Flowchart of Sending CAN Messages

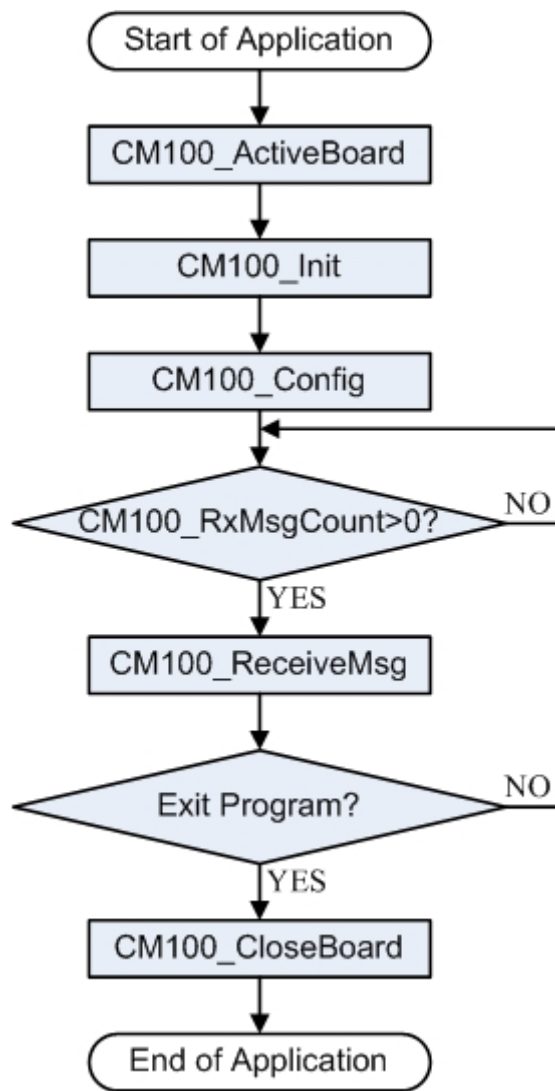


Figure 6.2 Flowchart of Receiving CAN Messages

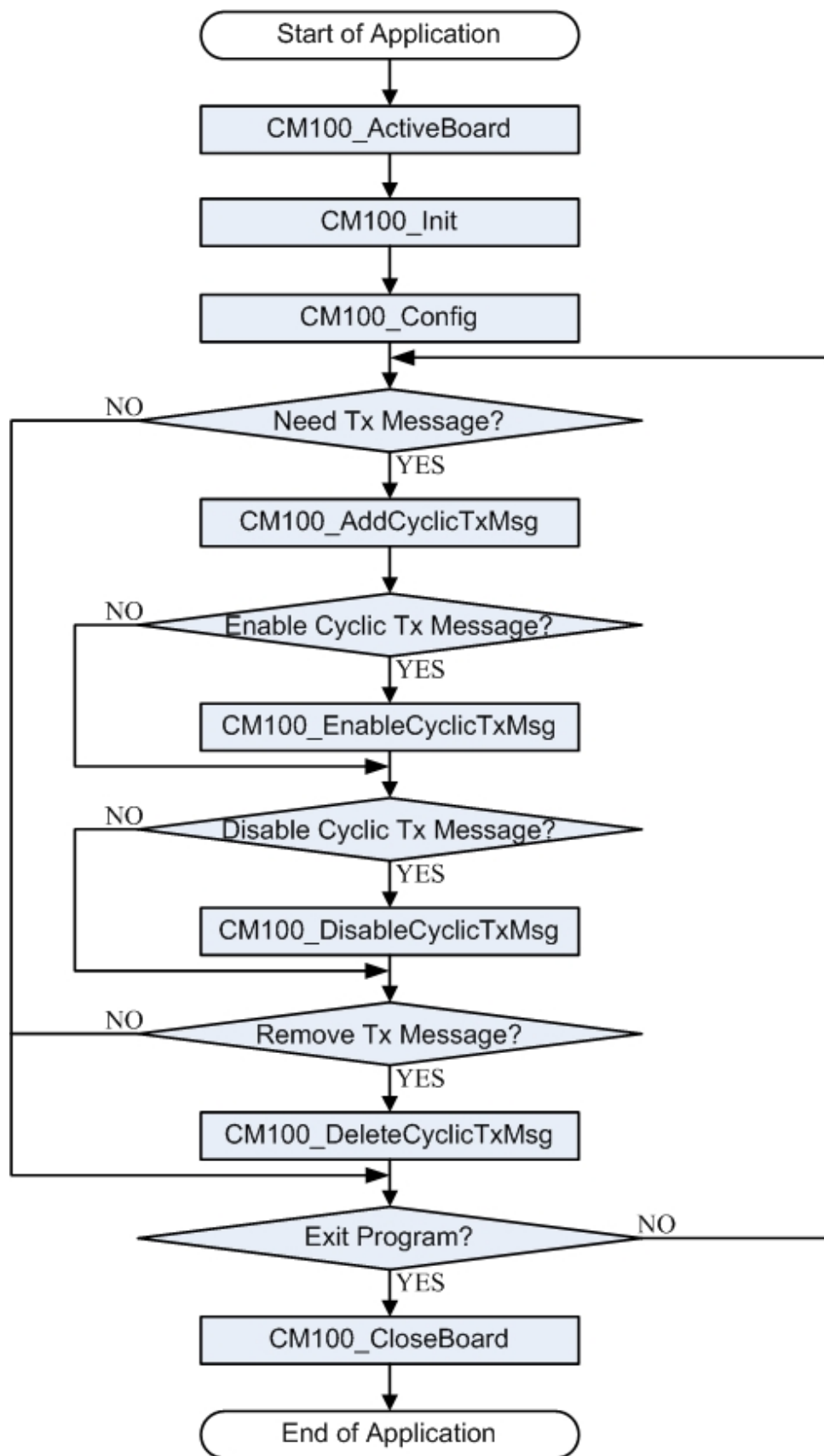


Figure 6.3 Flowchart of Cyclic Transmitting CAN Messages

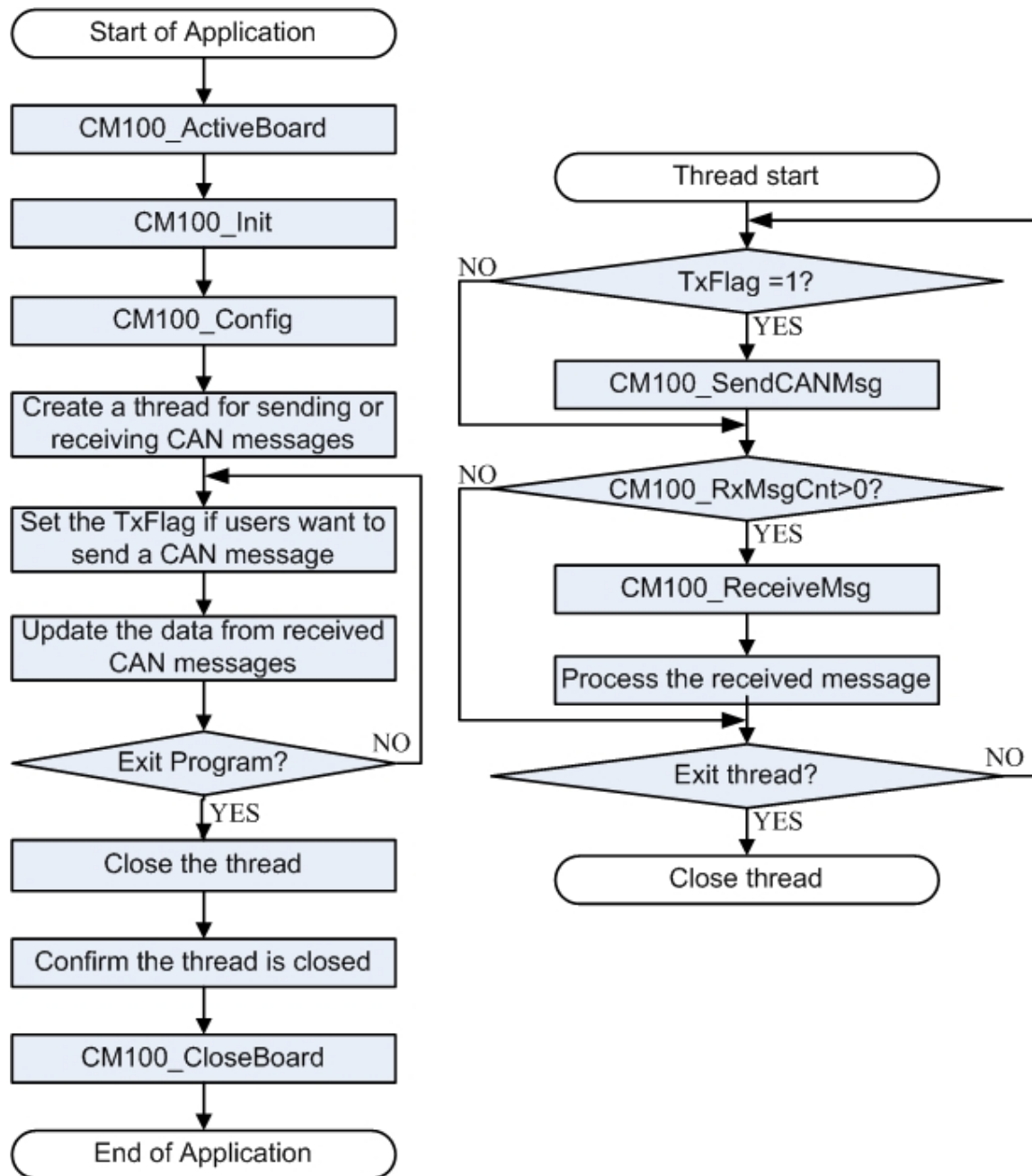


Figure 6.4 Flowchart of sending and receiving CAN Messages

Briefs of the demo programs:

All of demo programs described here need to assist with the default firmware of the CM100s board. Each demo can't work normally if the DLL driver would not be installed correctly. During the installation process of the DLL driver, the installation program also copy the demo programs to the proper position which is based on the path selected before. After installing the driver installation, the related demo programs, development library and declaration header files for different development environments are presented as follows.

--\Demos	→ CM100s board demo programs
--\For_Default_Firmware	→ For default firmware
--\BCB	→ For Borland C++ Builder 3
--\Library	→ Folder for library
--\ReceiveMsg	→ Demo for getting CAN messages
--\TransmitMsg	→ Demo for sending CAN messages
--\TransmitMsgCyclically	→ Demo for sending CAN messages
--\VC++	cyclically
--\Library	→ for Visual C++ 6.0
--\ReceiveMsg	→ Folder for library
--\TransmitMsg	→ Demo for getting CAN messages
--\TransmitMsgCyclically	→ Demo for sending CAN messages
--\VB	→ Demo for sending CAN messages
--\Module	cyclically
--\ReceiveMsg	→ For Visual Basic 6.0
--\TransmitMsg	→ Folder for library
--\TransmitMsgCyclically	→ Demo for getting CAN messages
--\Default_Firmware	→ Demo for sending CAN messages
	→ Demo for sending CAN messages
	cyclically
	→ Default firmware copy

ReceiveMsg:

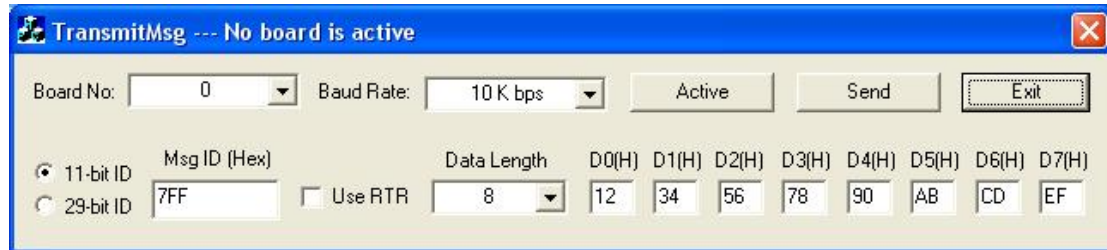
The ReceiveMsg demo is a sample example for demonstrating about how to receive CAN messages from CAN network by using the APIs of the CM100.dll with the default firmware. The dialog of this demo is shown in the follow figure.



Select the CAN baud and board No. of the specified CM100s board. Click the “Active Board” button to start this demo. Afterwards, the title of the dialog will display the name of the activated board. The SJA1000 status shown in the status filed will be updated every 500ms. Click the “Clear Status” button when the buffer of the SJA1000 is overflow. If there is any CAN message received by the CM100s board, users need to click “Receive” button to get these CAN messages from the reception software buffer. Of cause, users can put this part of the demo codes into the timer function or thread. The action of the receiving messages will always be checked by the program instead of the manual operation. If users need to clean the message list in the bottom of this dialog, click the “Clear List” button to do this.

TransmitMsg:

This demo is very useful if users want to send CAN messages. The dialog of the TransmitMsg demo is shown in the following figure.



As the description above, select the CAN baud and board No. of the specified CM100s board firstly. Then, click the “Active Board” button to start this demo. The title of the dialog will display the name of the activated board. After filling all parameters of a CAN message, users can click the “Send” button to send it out.

TransmitMsgCyclically:

Board No: 0 Baud: 10 K bps Active Exit

☒ 11-bit ID Msg ID (Hex) Data Length D0 (H) D1 (H) D2 (H) D3 (H) D4 (H) D5 (H) D6 (H) D7 (H) Period(0.1ms)
☐ 29-bit ID 7FF ☐ Use RTR 8 12 34 56 78 90 AB CD EF 10000

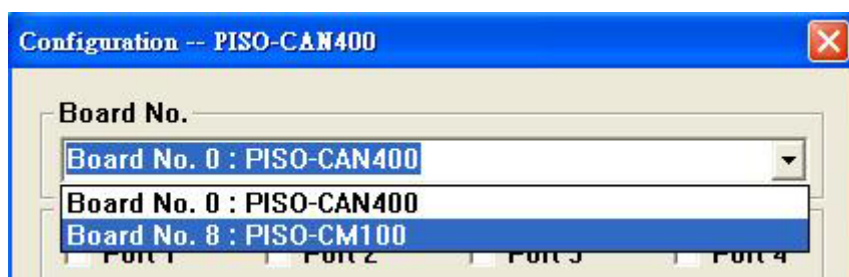
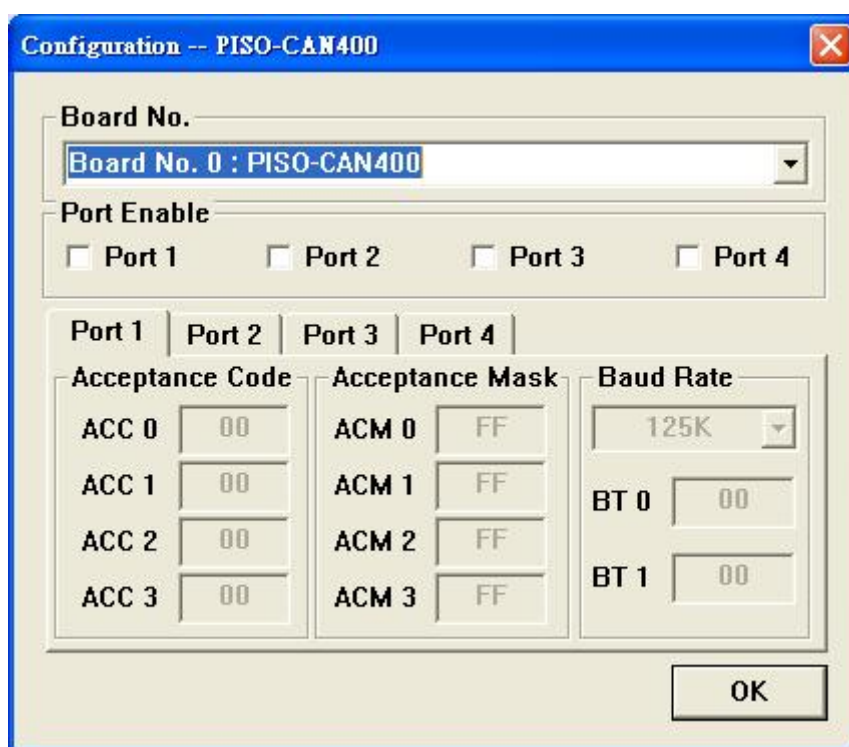
Handle No.	Mode	ID	RTR	DLen	D0	D1	D2	D3	D4	D5	D6	D7	Time (0.1ms)	Status

Add Delete Enable Disable

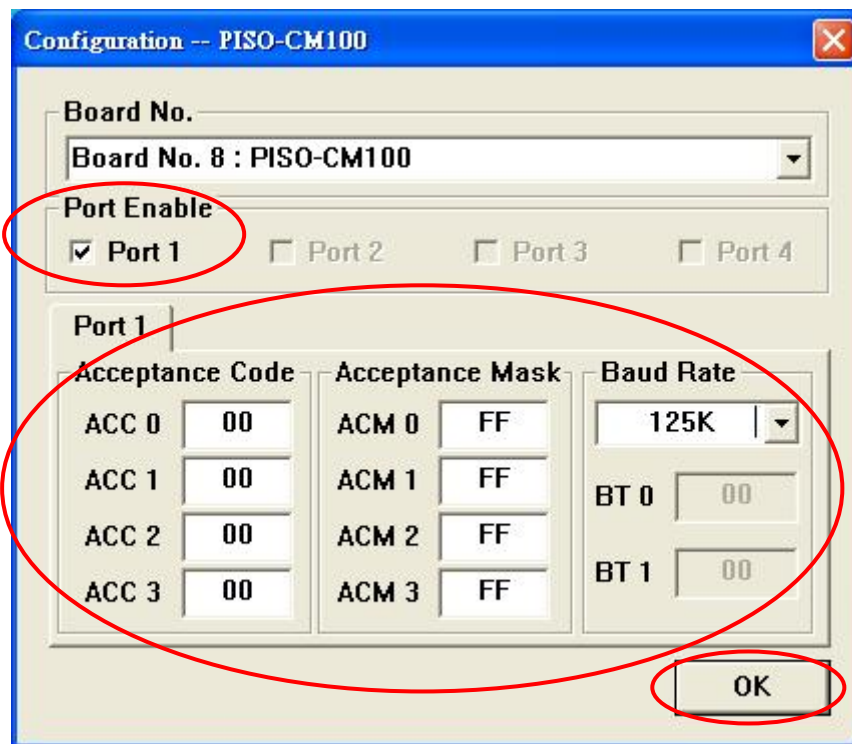
The dialog is shown as above figure. Firstly, select board No., baud and click the “Active” button to activate the specified CM100s board. Secondly, configure all parameters of the cyclic CAN message. Be careful that the parameter Period is the unit of 0.1ms. The value of this parameter must be more than 5. Click the “Add” button to add this message into the cyclic transmission engine. This message will also be shown in the message list in the bottom of the dialog. Afterwards, users can select a cyclic message listed in the message list, and click “Enable” button to start the message transmission. If users want to stop it, select it from the message list, and click the “Disable” button. The action of deleting a cyclic message from cyclic transmission engine is similar with the action of disable a cyclic transmission. Just select the cyclic message from the message list, and click the “Delete” button.

6.2 Introduction of CANUtility Tool

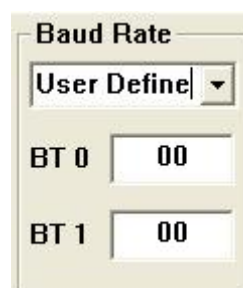
The CANUtility is designed for the CAN boards of the ICP DAS. It provides useful functions when users want to debug users' CAN application, monitor CAN devices and access a CAN network. Users can find it in the folder of PISO-CM100U where you installed the driver before. The default path is "c:\ICPDAS\PISO-CM100U". When you execute the CANUtility.exe, the Configuration dialog is popped up below. All CAN board searched by the CANUtility.exe will be listed in the Board No. filed. If users do not want to do the configuration, click the button "  " to skip this procedure. Here, select PISO-CM100 for the demonstration.



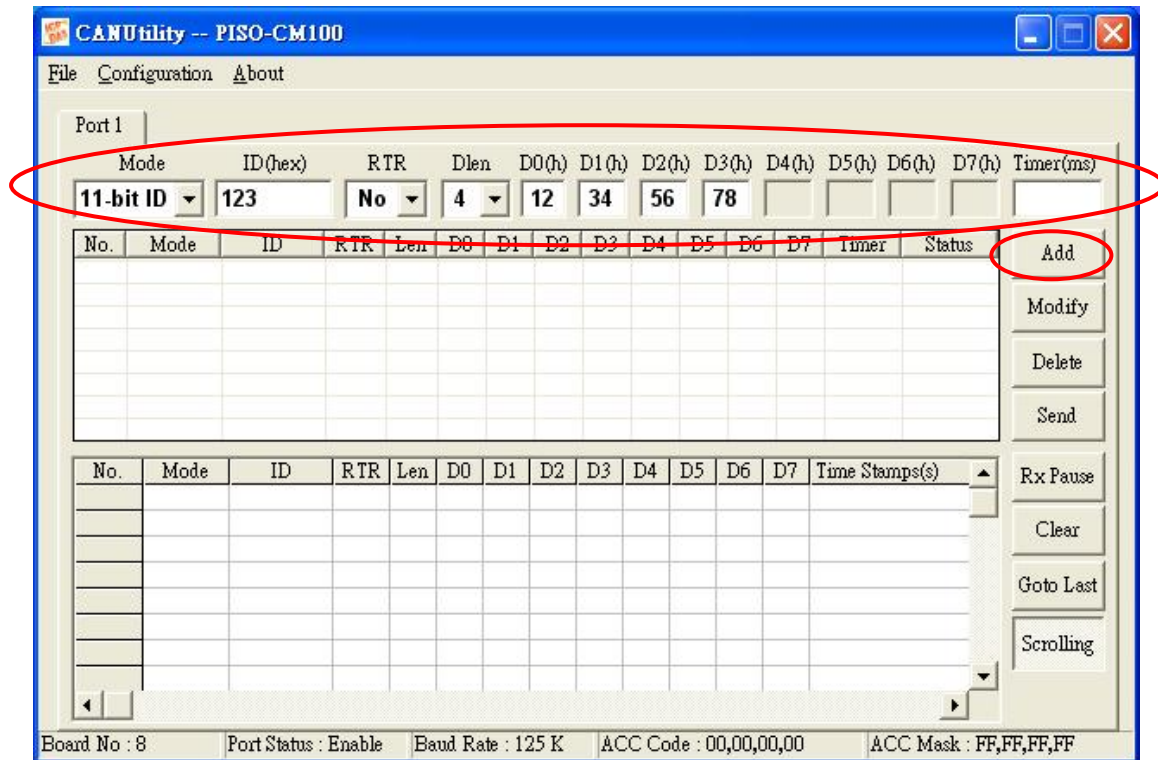
Because the PISO-CM100 has only one CAN port, the Port2, Port3 and Port4 are disabled. Check the checkbox Port1 to enable it. Afterwards, you can modify the parameters of the acceptance code, acceptance mask and baud. The description of the function CM100_Config in chapter 4 can give a good reference about how to set the acceptance code and acceptance mask.



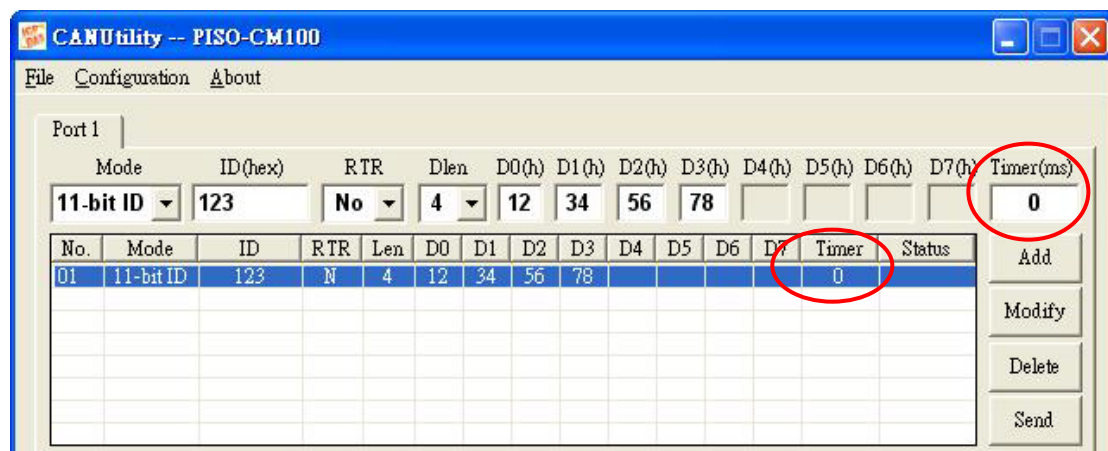
If the proper baud can't be found in the Baud list, select the "User Define" to define special baud by using the BT0 and BT1 of the SJA1000. In this case, users need to study the datasheet of the SJA1000 to know how to use the registers BT0 and BT1 to configure the baud.



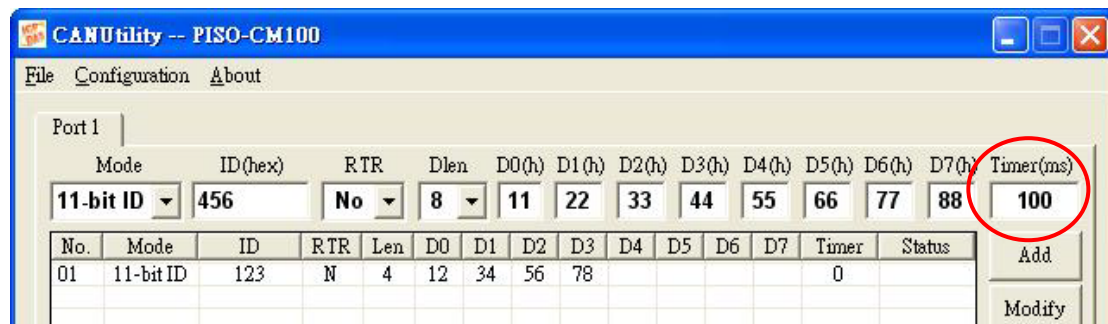
After finishing the configuration and clicking the button “OK”, the main screen of the CANUtility is displayed. The title of the CANUtility shows the activated board name. The status of this board is shown on the status bar in the bottom of the window.



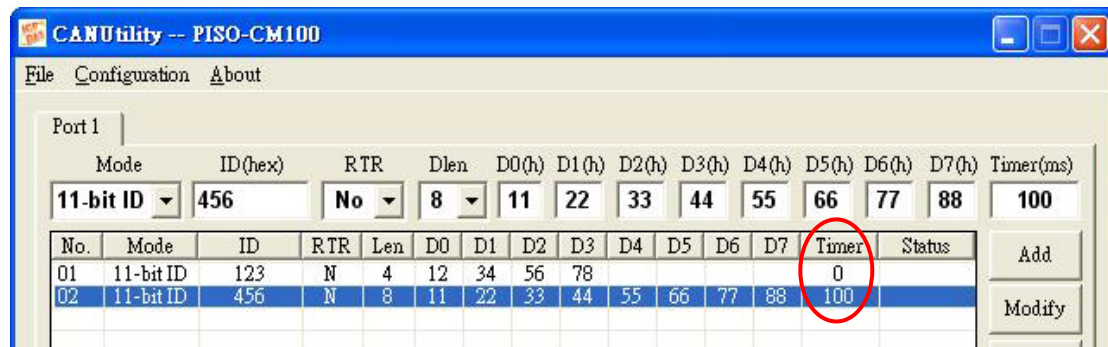
Users can set the parameters of a transmitted CAN message, and click Add button to put it into the list. If users just need to send one CAN message, please let the Timer filed to be 0 or empty.



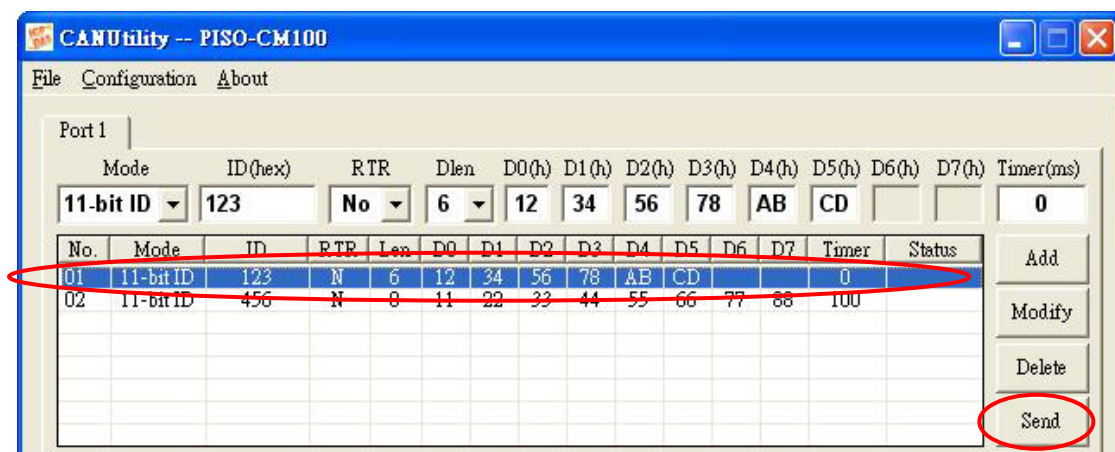
If the cyclic transmission messages are demanded, users can configure the message parameters with the timer filed.



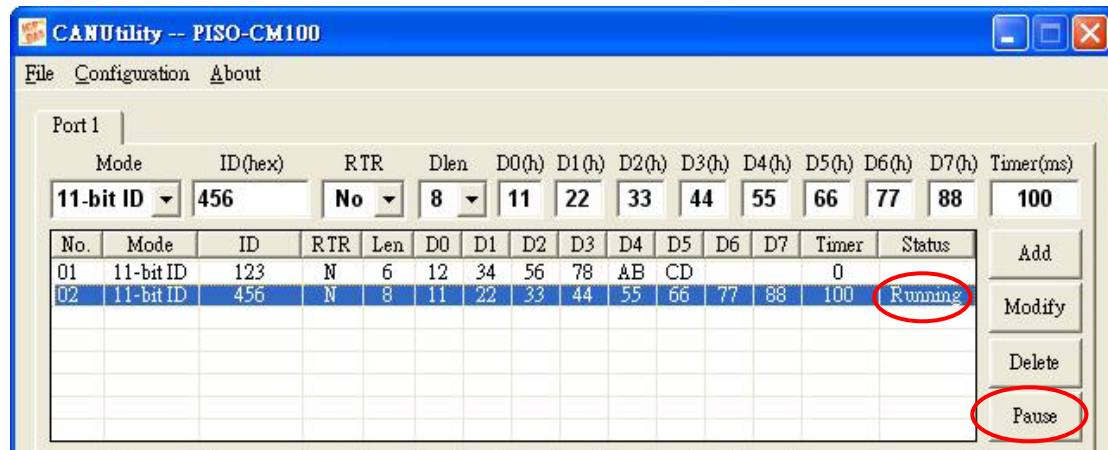
After finishing the configuration, click the button “Add” to add the cyclic transmission message into the list.



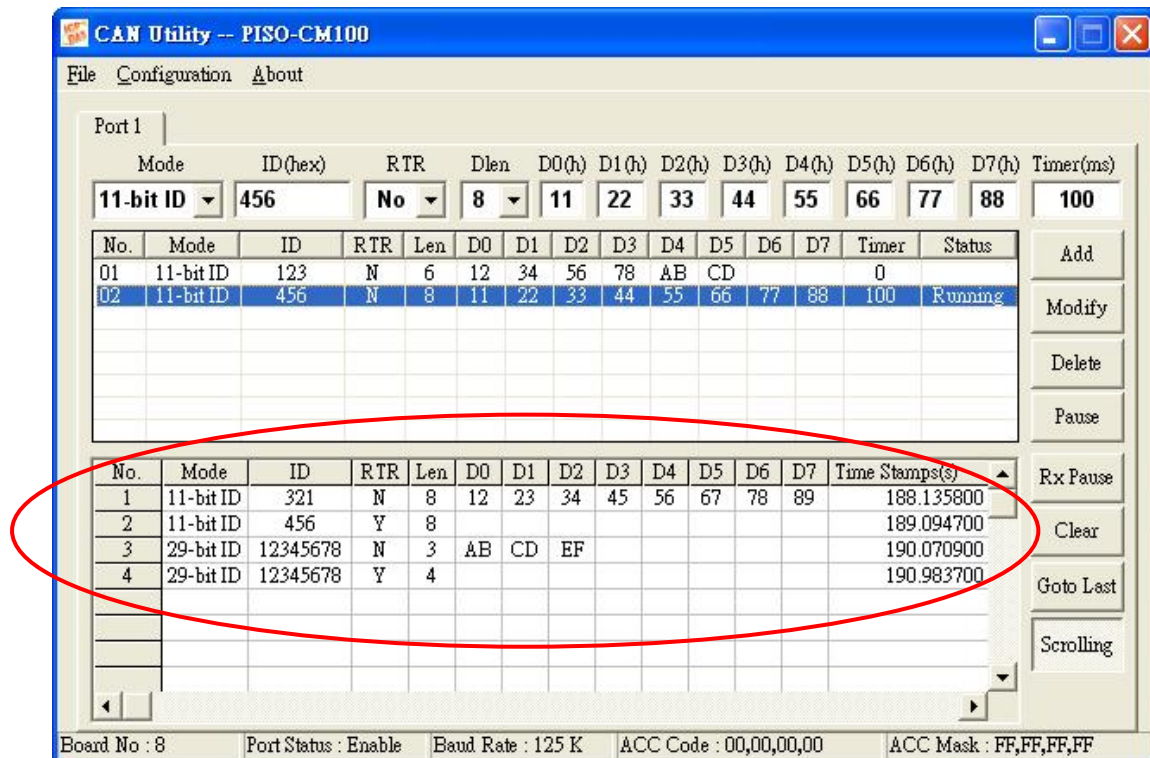
When the CAN messages are added into the list, the PISO-CM100 will not send them to the CAN network until users click the button “Send”. Therefore, select the CAN messages which you want to send from the list, and click the button “Send” to send it.



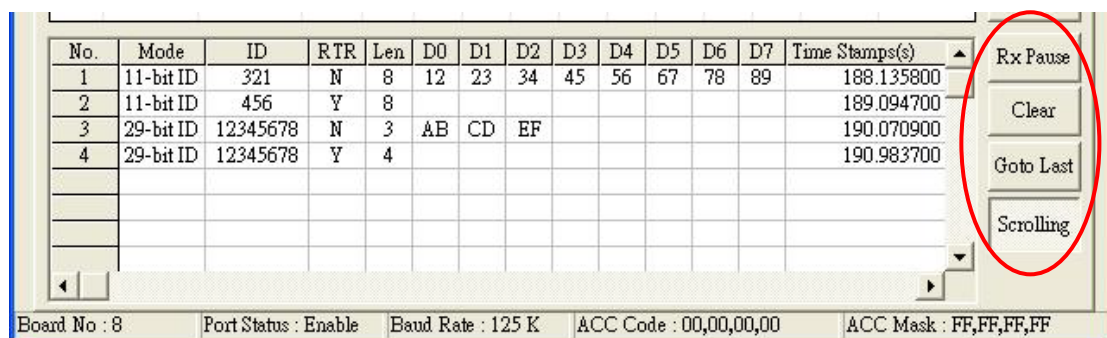
Similarly, select the cyclic transmission messages from the list, and click button “Send” to send the cyclic transmission. Afterwards, the status of the sent cyclic transmission message is changed to “Running”, and the button “Send” is also changed to the button “Pause”. If users want to stop the transmission, select the sent cyclic transmission message from the list, and click the button “Pause” to stop it.



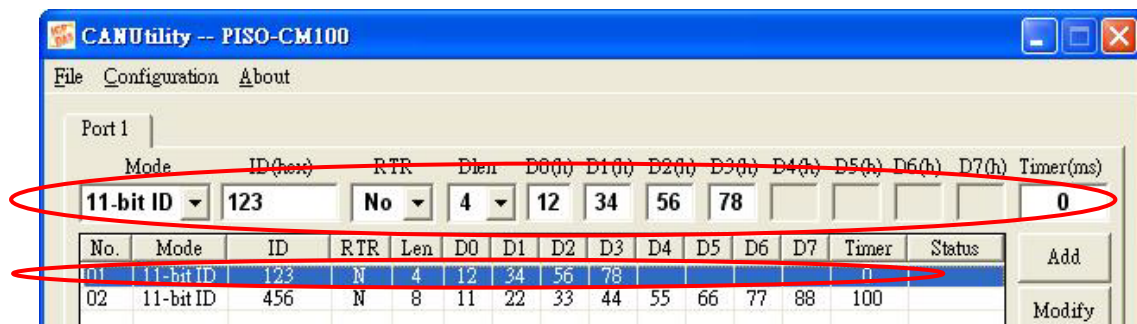
If any CAN message is obtained by the CANUtility, it will be put into the reception list in the bottom of the window. The filed “Time Stamps” shows the time when a message is got. The time base is the boot up time of the CPU of the PISO-CM100U (Take a note that the board which has no CPU uses the system time to be the time stamp while the kernel driver gets this message).



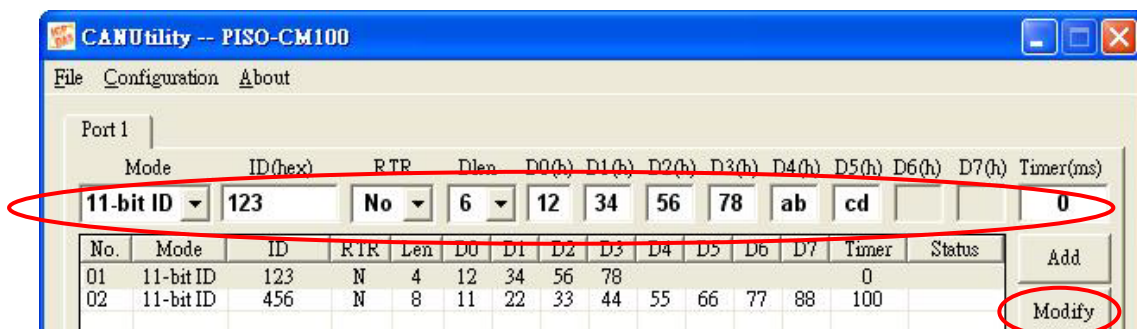
Users can click the button “Rx Pause” to pause the reception of the CAN messages. Or click this button again to continue the reception. Click the button “Clear” to empty the reception list. The button “Goto Last” is used to move the scroll bar of the reception list to the last record of the received CAN message. If the button “Scrolling” is activated as the following figure, the reception list will be scrolled automatically when any CAN message is received. If the button “Scrolling” is inactivated, the auto-scrolling stops, but the received CAN messages are still put into the reception list.



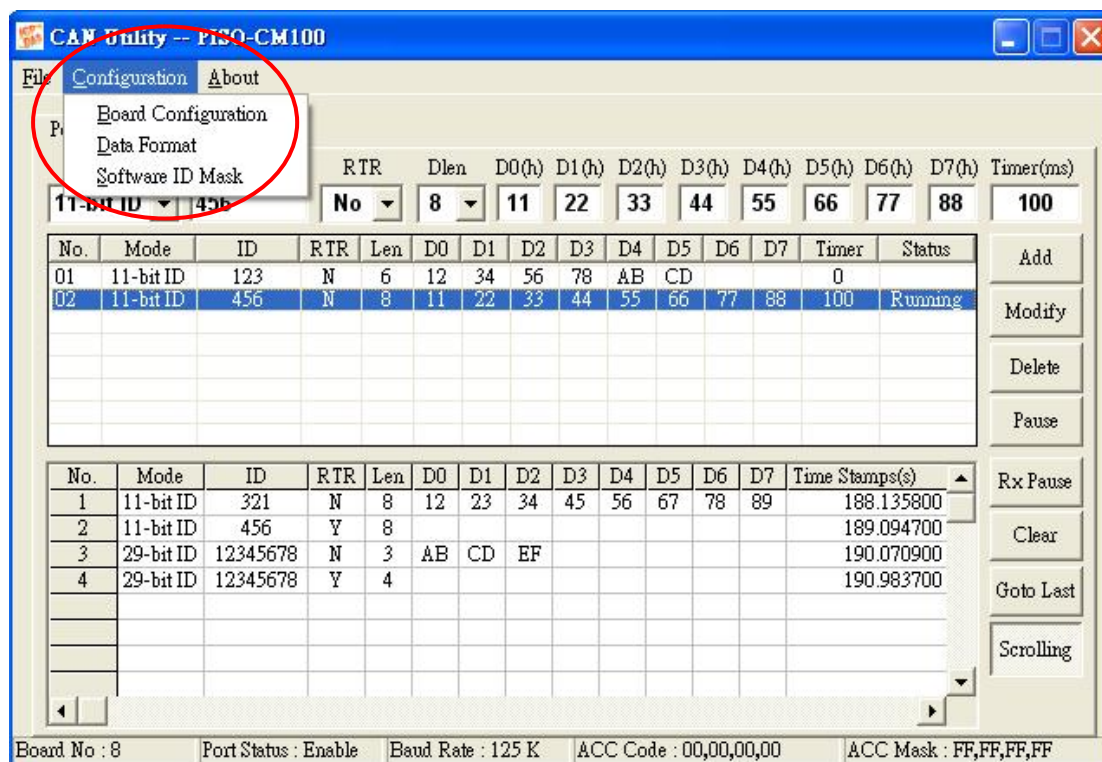
If users want to modify the parameters of the CAN message added before, select the specified CAN message from the list. Then the configuration fields will be filled with the parameters of the specified CAN message.



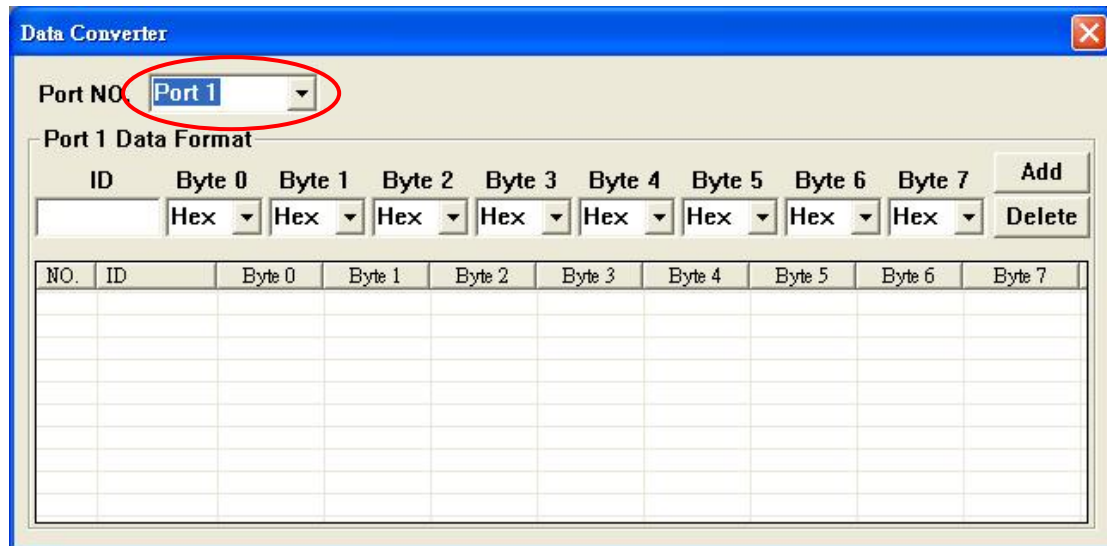
Users can modify the parameters in these configuration fields. Then, click the button “Modify” to modify it.



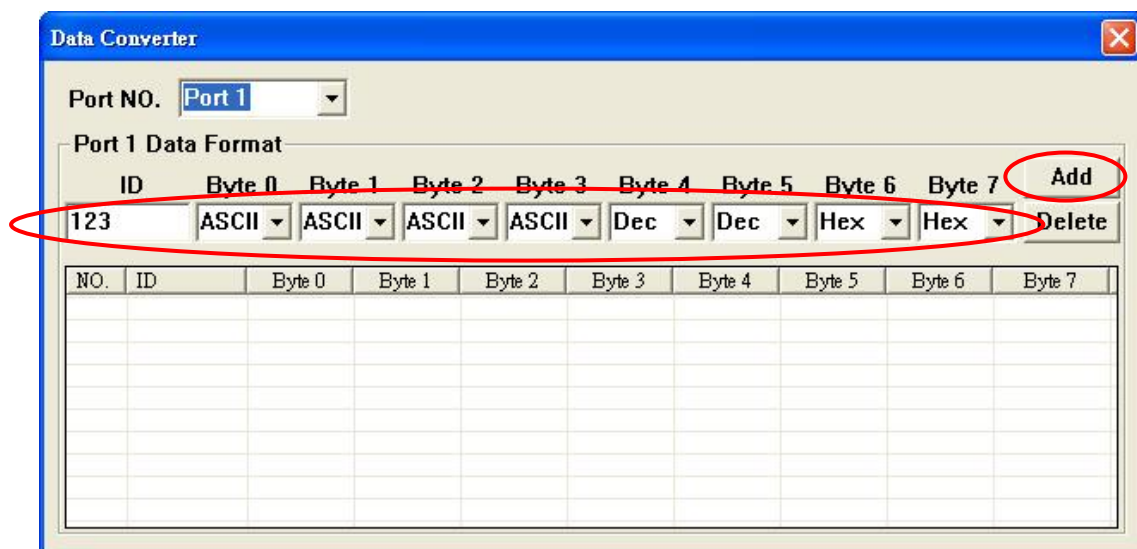
The CAN utility also provides the special functions when reading the CAN data. For example, some CAN messages with specified message ID need to be notified, or some bytes of the data filed of a CAN message need to be transferred to the ASCII characters. These demands can be achieved by using the functions of the Configuration item of the menu. The Board Configuration function let users modify the configuration of the specified board. It is the same as the dialog popped up when the CANUtility.exe is starting. The Data Format function provides a human interface to set the data format for each byte of the data filed of a specified CAN message. The Software ID Mask function is similar with the functions of the acceptance code and acceptance mask of Configuration dialog popped up in the start of the CANUtility.exe. The former uses software method to filter the useless CAN messages, the latter uses hardware method to do this. The feature of the Software ID Mask is that it allows users to filter any CAN messages which you wouldn't need by select the "Un Pass" and the interesting messages by select the "To Pass". It is more flexible than setting the acceptance code and acceptance mask. But its performance is not good enough as the hardware message filter.



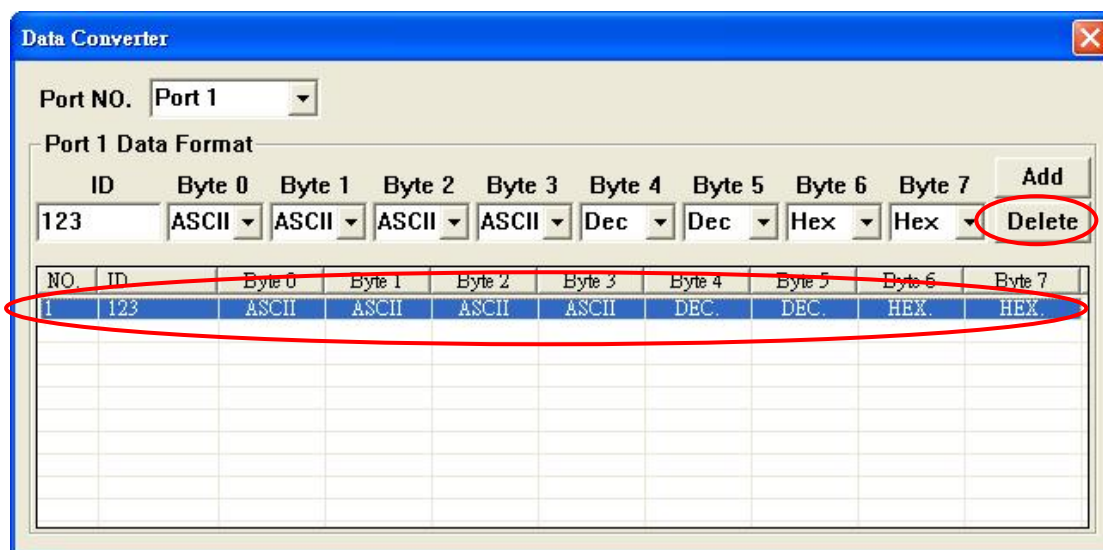
After clicking the item of the Data Format, the Data Converter dialog is popped up. Users can select the port No. to set the port which will transfer the received messages to the specified data format.



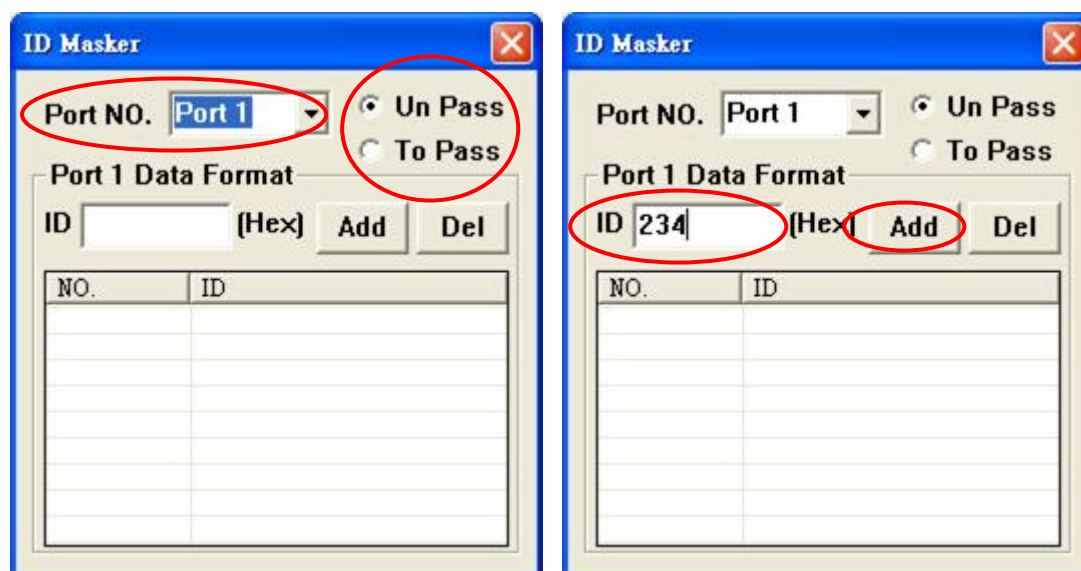
When finishing the settings of the data format for the specified message ID, click Add button to save the configuration. Here provides three kinds of data format, Hex., Dec. and ASCII. The default setting for received CAN messages is hexadecimal format.



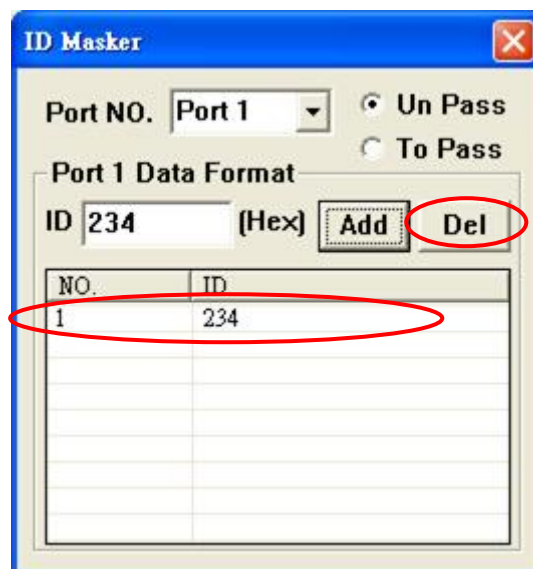
If users want to cancel the configuration which is set before, select the record from the list firstly, then click “Delete” button to remove it.



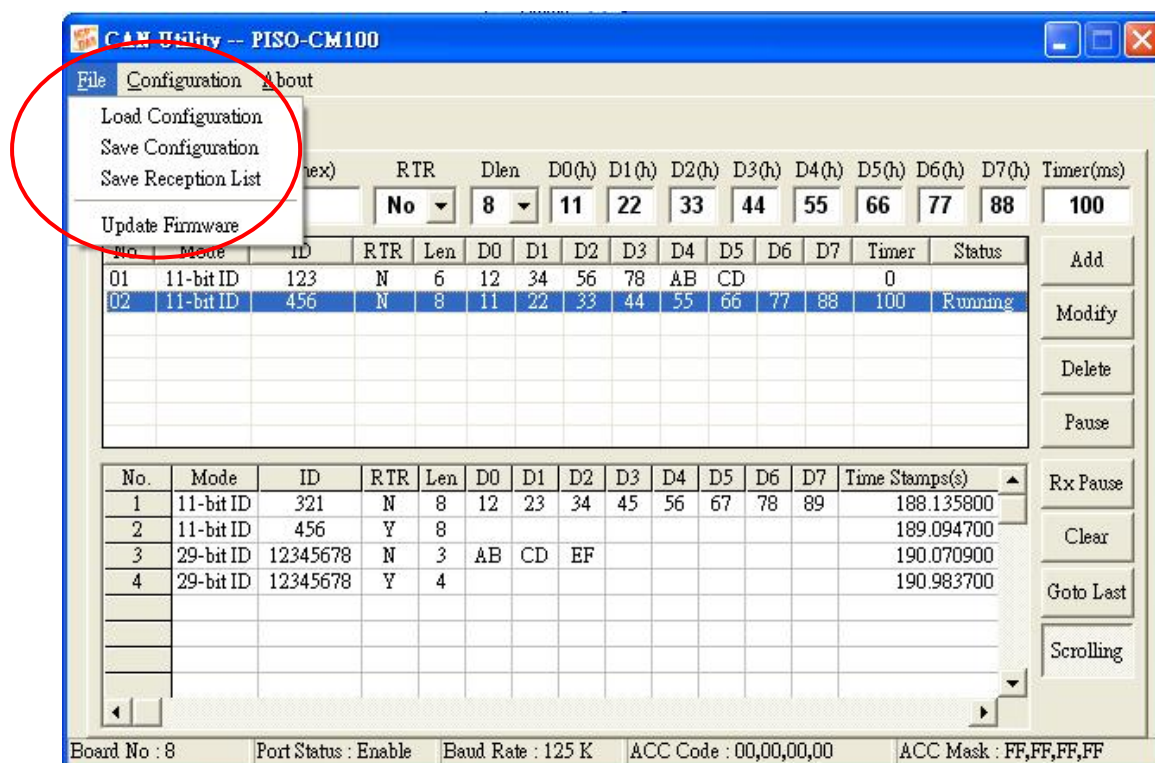
The “Software ID Mask” function is executed in the “ID Masker” dialog. Select the port No. and the “Pass” type firstly. Fill the message IDs of the CAN messages which you want to drop. Finally, click “Add” button to store the result.



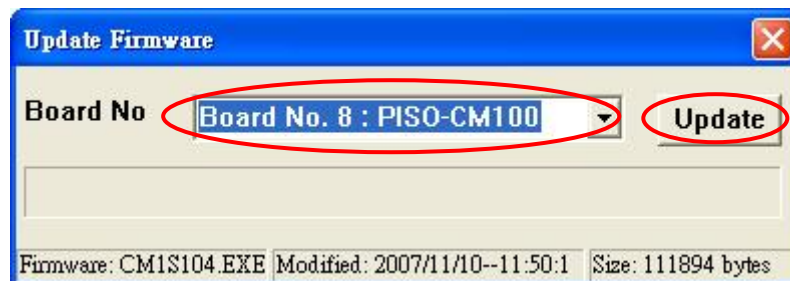
Select the record from the list and click Del button to remove the record added before. The maximum content of the list is 20 records.



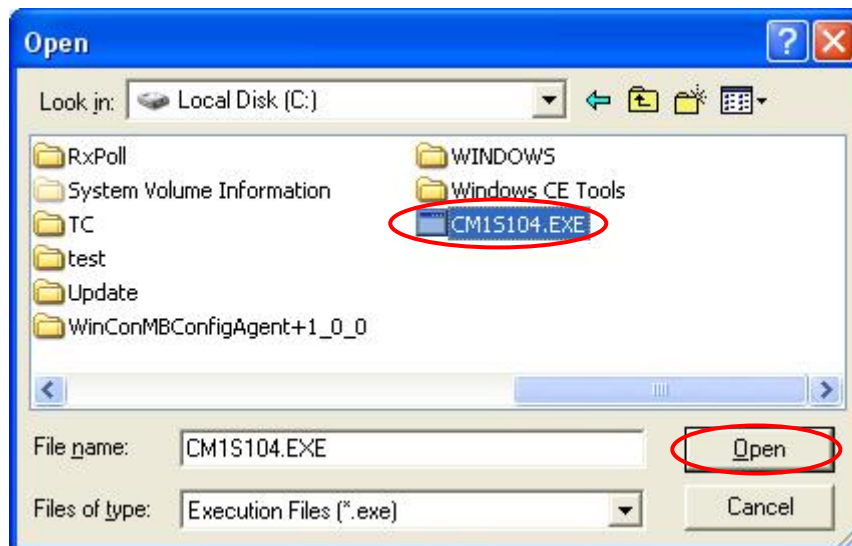
Besides the functions described above, the CANUtility allows users to save and load the configuration parameters by applying the Load Configuration or Save Configuration of the “File” item in the menu. The Save Reception List function helps users to store the records of the received CAN messages into .txt file. The Update Firmware function let users update the default firmware or download the user-defined firmware. This function is only for CM100 series cards.



When users apply the Update Firmware function, select the specified board firstly. Only CM100, CPM100 and DNM100 series cards are listed in the Combo box. Click the Update button to select the proper firmware for the specified board.



Only the .exe file can be downloaded into the CM100s board.

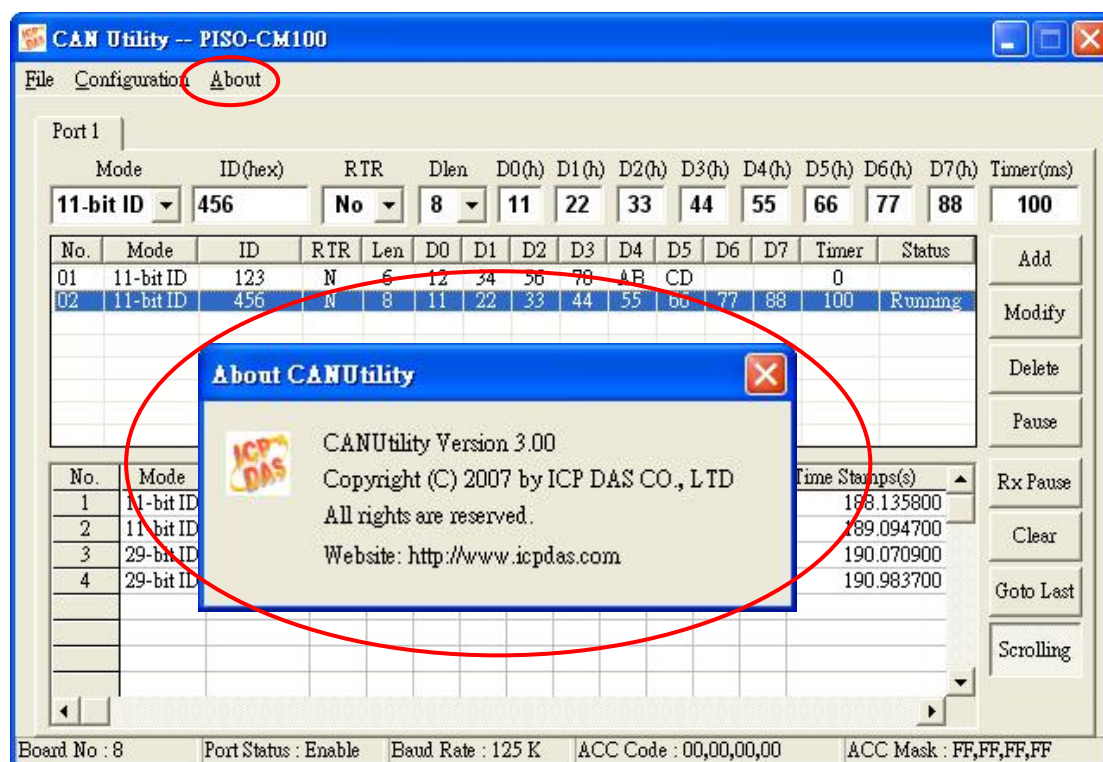


When finishing the download procedure, the Download OK dialog is popped up. Click the OK button to continue.



If you want to check the version of the CAN Utility, please click the "About"

item of the menu to get the information. The version No. of the CAN Utility you use may be different with the following picture.

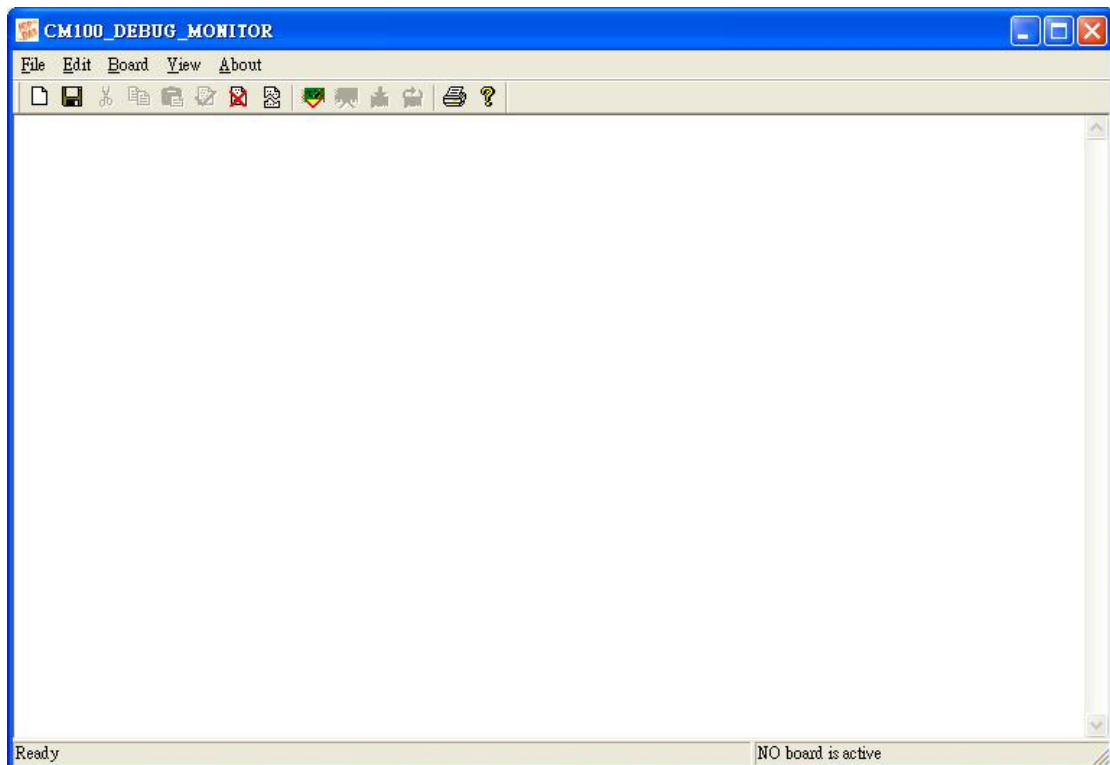


6.3 Debug Tools for User-defined Firmware Programming

If users just apply the default firmware for their application, this section can be ignored. This section introduces the debug methods when users design their firmware. Basically, when users develop the user-defined firmware, the debug message can be put into the code section of the user-defined firmware which may have bugs inside. Then, compile the user-defined firmware, and download it into the CM100s board. Owing to check the debug message, the bugs could be found. The debug methods are shown below.

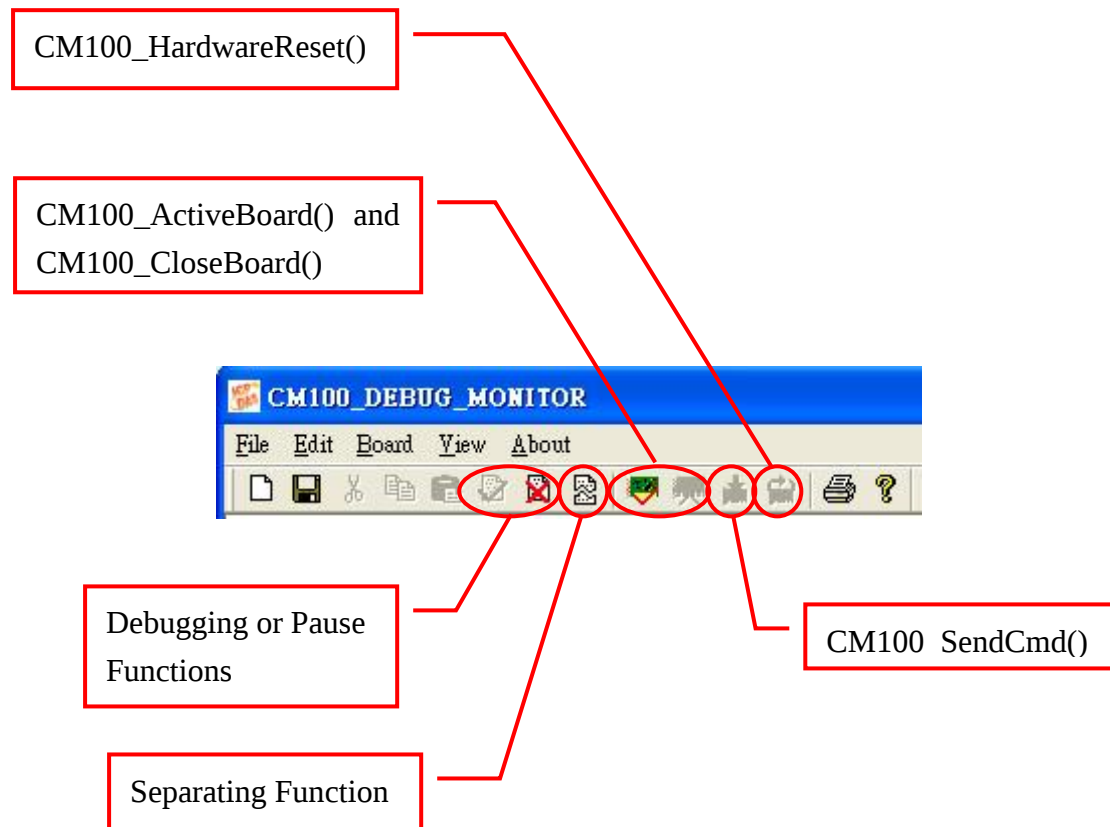
CM100_DEBUG_MONITOR.exe:

If the functions `DebugPrint()` is applied in the user-defined firmware. Users need to check the debug message by using the `CM100_Debug_Monitor.exe`. It is displayed as following figure. Users can find it in the Fieldbus CD. The path is `CAN\PC\PISO-CM100\`.

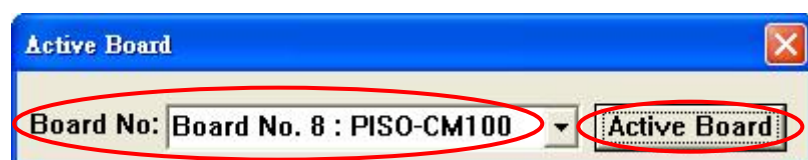


Because of the software architecture of the CM100s board, the `CM100_Debug_Monitor.exe` is useful only if the CM100s board is activated. Therefore, this debug program provides the `CM100_ActiveBoard()` function and `CM100_CloseBoard()` function. If users want to send the commands to

user-defined firmware or restart the user-defined firmware, they are also provided by CM100_SendCmd() and CM100_HardwareReset() functions in the CM100_DEBUG_MONITOR.exe. These functions are built in the tool bar as below.

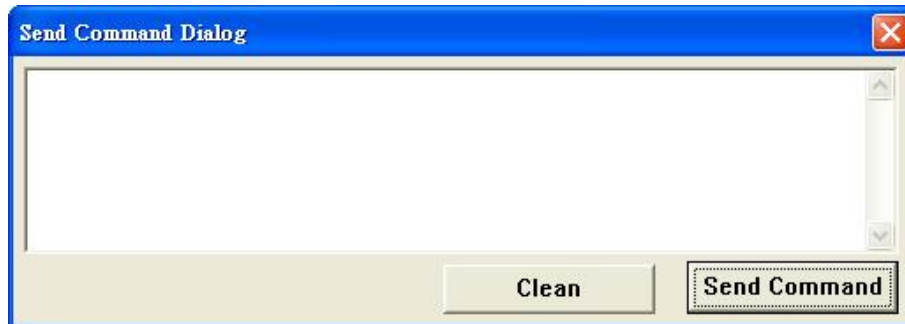


If any other program has activated the specified CM100s board, the functions CM100_ActiveBoard() and CM100_CloseBoard() of the CM100_DEBUG_MONITOR.exe are not needed because one CM100s board can be activated by only one program at the same time. After clicking the CM100_ActiveBoard() function, the dialog is popped up as below.

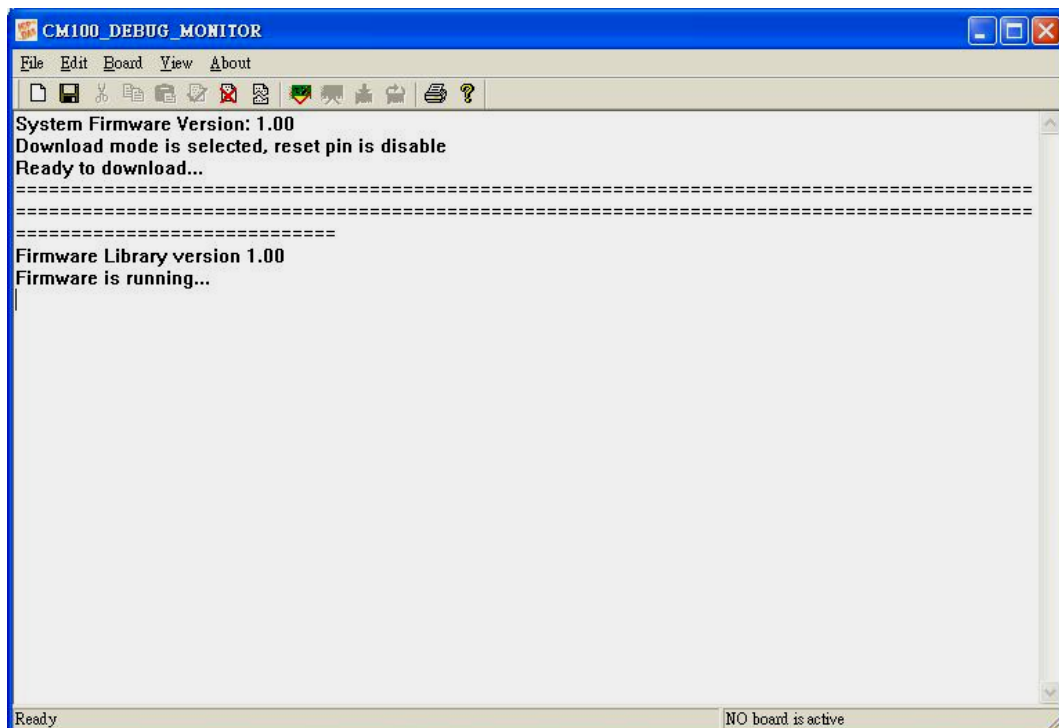


Users can select the proper board name and click “Active Board” button to activate this board. When board is activated, users can use CM100_SendCmd() function to send command to the user-defined firmware. The sending command dialog is shown below. Users can key the ASCII string

in the edit box and click the button “Send Command” to the user-defined firmware. If users need to clean the edit box, use the “Clean” button to do this.

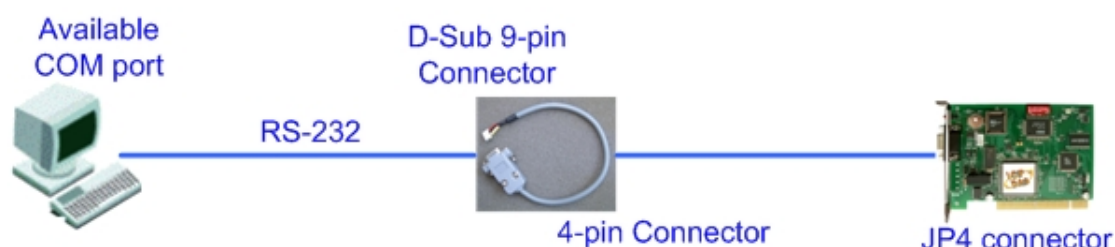


The debugging and pause functions are used to decide if the CM100_DEBUG_MONITOR.exe shows the received debug messages or not. If not, the debug messages will be dropped. The separating function is applied when users want to separate the debug messages. After using this function, the screen of the CM100_DEBUG_MONITOR.exe is shown below. The end of content of debug messages will be separated by the equal marks. When newer debug messages are received by the CM100_DEBUG_MONITOR.exe, they are put in the end of these equal marks.

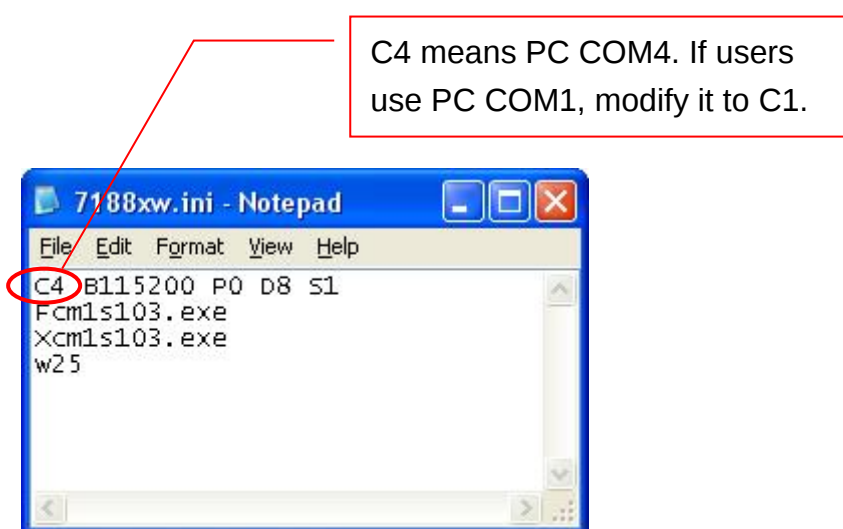


7188xw.exe:

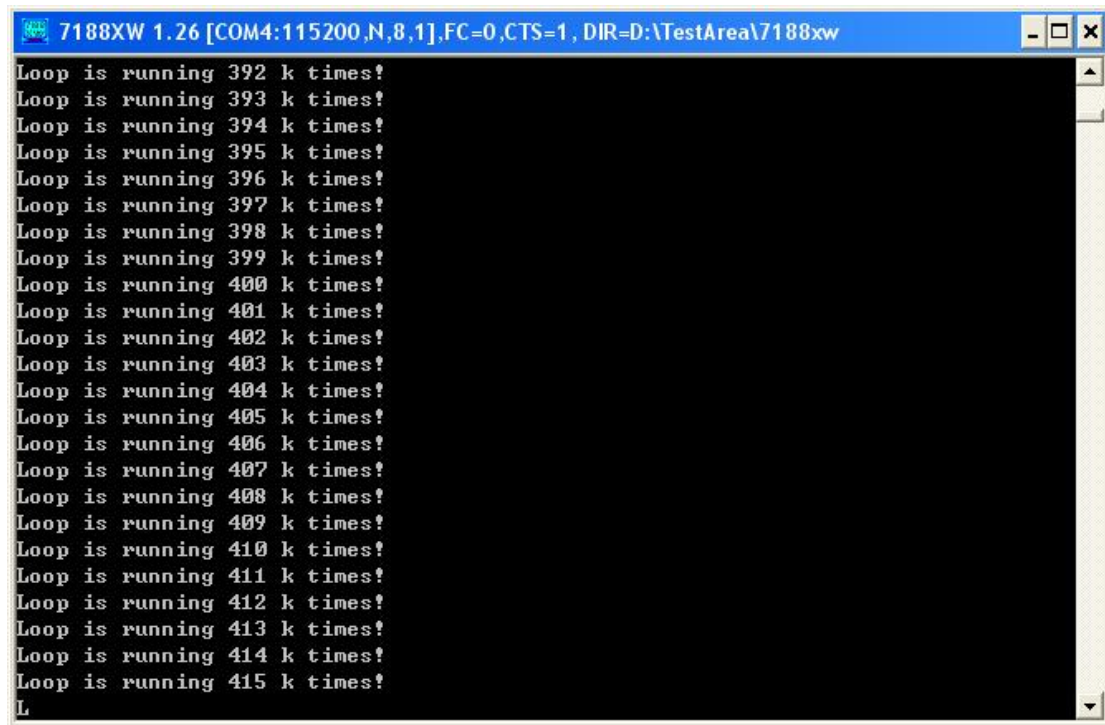
The firmware library provides two functions for debugging. The function GetKbhit() allows users to receive a inputted character from 7188xw.exe. Therefore, users can use this feature to trigger some specified event for debugging. The function Print() allows users to send debug messages to 7188xw.exe. Then, 7188xw.exe will put these debug messages on the screen of 7188xw.exe. Before implementing this method, users need to use the debug cable. Plug the debug cable to the debug connector of the CM100s board described in the chapter 2. Connect an available PC COM port with the D-Sub 9-pin connector of debug cable. The situation is shown as following figure.



Then, use Notepad.exe to configure the 7188xw.ini to set the number of the specified PC COM port which is connected with the debug cable, and execute the 7188xw.exe. The configuration screen is displayed as following figure. Users can find the 7188xw.ini and 7188xw.exe in the Field Bus CD. The path is CAN\PC\ISO-CM100U\.



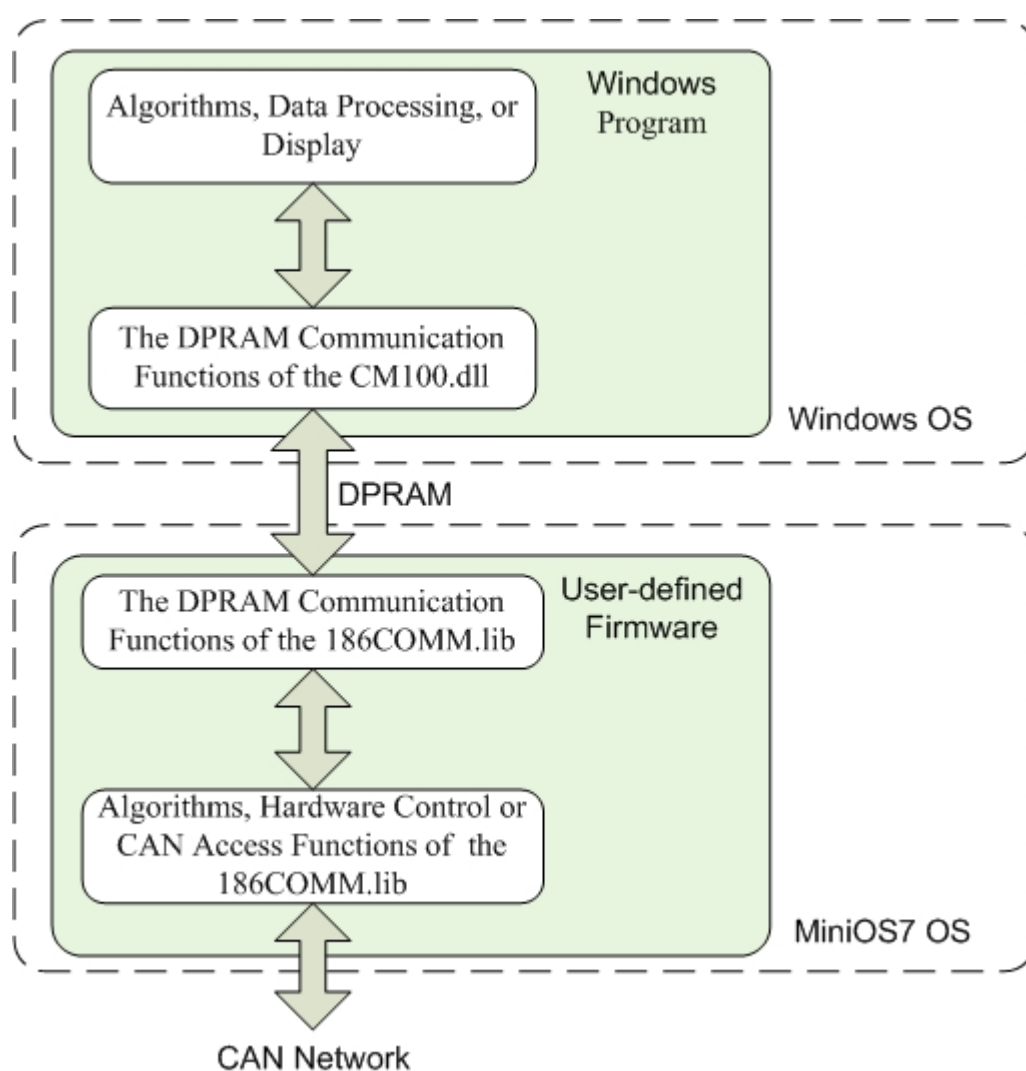
Any keyboard input will be caught by the user-defined firmware via the GetKbhit() function. The debug messages sent by the function Print() will also be displayed on the screen of the 7188xw.exe.

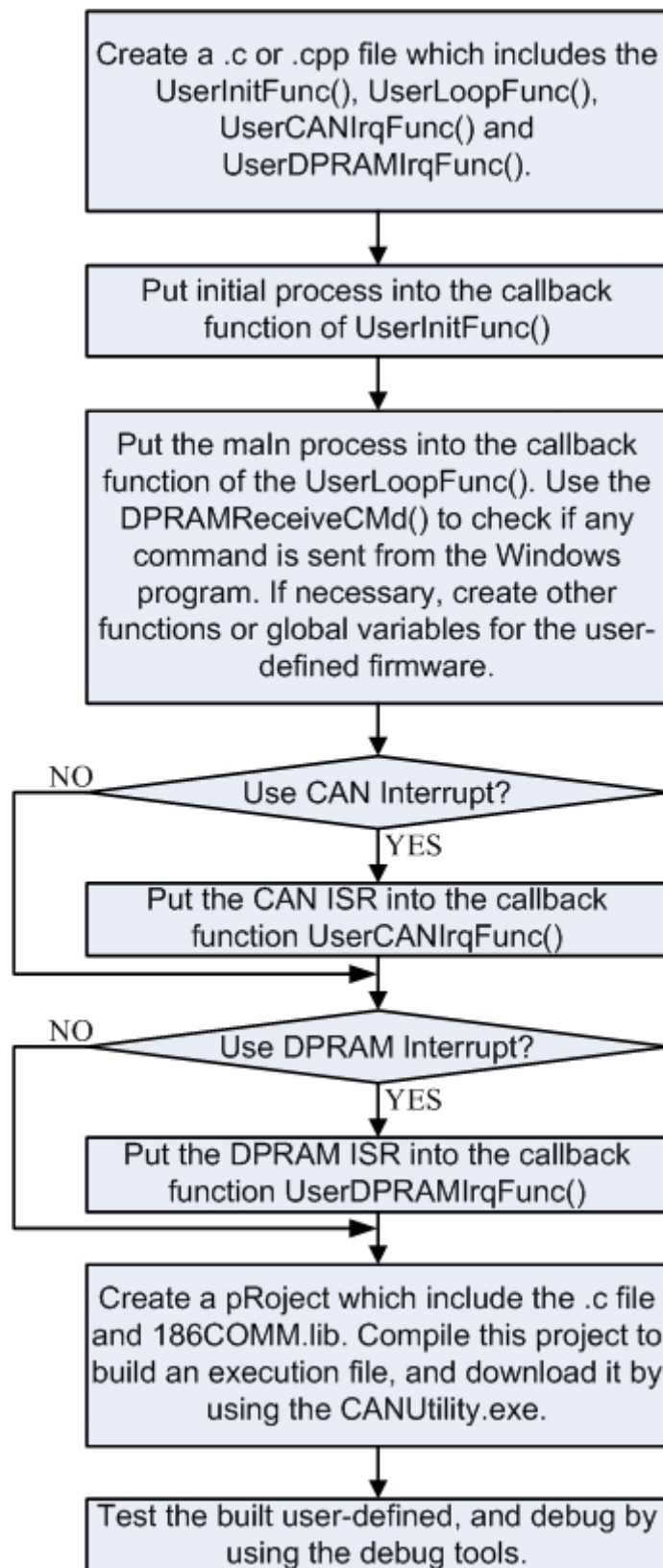


```
7188XW 1.26 [COM4:115200,N,8,1],FC=0,CTS=1, DIR=D:\TestArea\7188xw
Loop is running 392 k times!
Loop is running 393 k times!
Loop is running 394 k times!
Loop is running 395 k times!
Loop is running 396 k times!
Loop is running 397 k times!
Loop is running 398 k times!
Loop is running 399 k times!
Loop is running 400 k times!
Loop is running 401 k times!
Loop is running 402 k times!
Loop is running 403 k times!
Loop is running 404 k times!
Loop is running 405 k times!
Loop is running 406 k times!
Loop is running 407 k times!
Loop is running 408 k times!
Loop is running 409 k times!
Loop is running 410 k times!
Loop is running 411 k times!
Loop is running 412 k times!
Loop is running 413 k times!
Loop is running 414 k times!
Loop is running 415 k times!
L
```

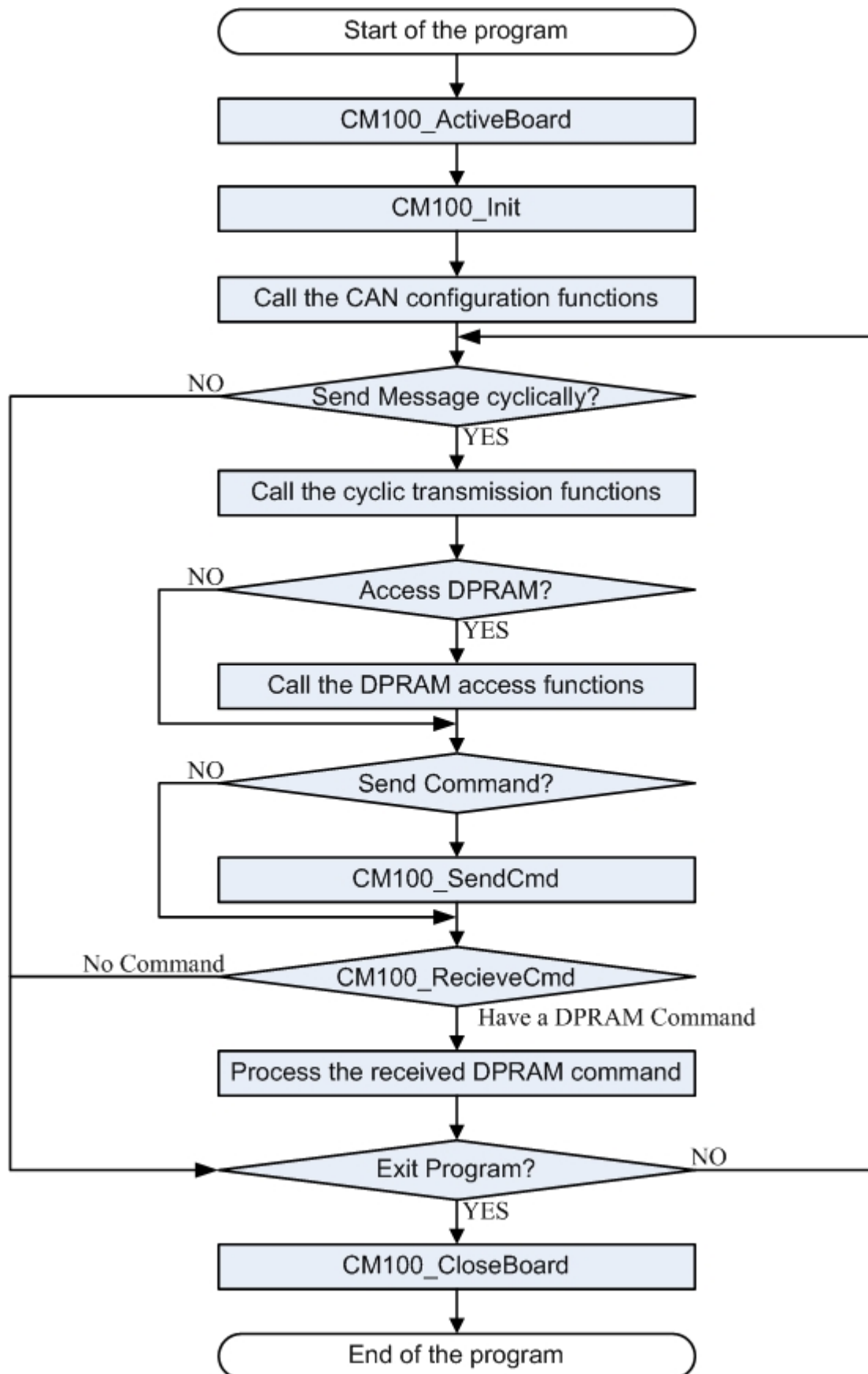
6.4 User-defined Firmware Programming

If users just apply the default firmware for their application, this section can be ignored. This section describes about how to build a user-defined firmware. A CAN application can be implemented corresponding to the good cooperation of the Windows application and the user-defined firmware. Generally speaking, the user-defined firmware processes the interpretation of the CAN communication protocol and the algorithms of sending the required CAN messages. The Windows program can use the processed data from the user-defined firmware to implement the applications or shows these data on the HMI. The windows program can also give a command to the user-defined firmware for sending the data to the CAN network. The relationship between Windows applications and the user-defined firmware is shown as the following figure.





The development procedure of the user-defined firmware



The development procedure of the corresponding Windows program

The above two figures shows the basic flowchart of developing the user-defined firmware and corresponding Windows program. For the user-defined firmware development, users can create a C/C++ project, and include the .c files and 186COMM.lib. Put the 4 callback functions in one of these .c file. Program the codes into these 4 callback functions. If necessary, build your functions and global variables. Then, compile this project, and you can get the user-defined firmware. Download it by using the CANUtility.exe and test it. Afterwards, according to the user-defined firmware, design your Windows program to communicate with the firmware via the DPRAM access functions of the CM100.lib and the 186COMM.lib. Besides the DPRAM access functions, the firmware library supports most APIs for the hardware on the CM100s board, such as the EEPROM accessing, the RTC access, the timer functions... and so forth.

When you want to design a Windows program, the BCB, VC++, or VB development environment are needed. Users can refer to the textbook of the BCB, VC++ or VB for more information about how to use the APIs of the .dll library in these development environments. The user-defined firmware can be programmed by the BC/BC++/TC/TC++ development environments. Here, it is considered that how to build an execution file with the 186COMM.lib by using the TC++1.01 compiler. Before starting the step-by-step procedures, users need to install the TC++1.01 compiler and the CM100s board Windows driver. Users can free download the TC++1.01 compiler on the following website.

TC++1.01: <https://winworldpc.com/product/turbo-c/1x>
BC++3.1: <https://winworldpc.com/product/borland-c/30>

The CM100s board Windows driver can be found in the Fieldbus CD or our website. Please refer to the chapter 3 for more details. The following paragraph is a step-by-step description about how to build the user-defined firmware.

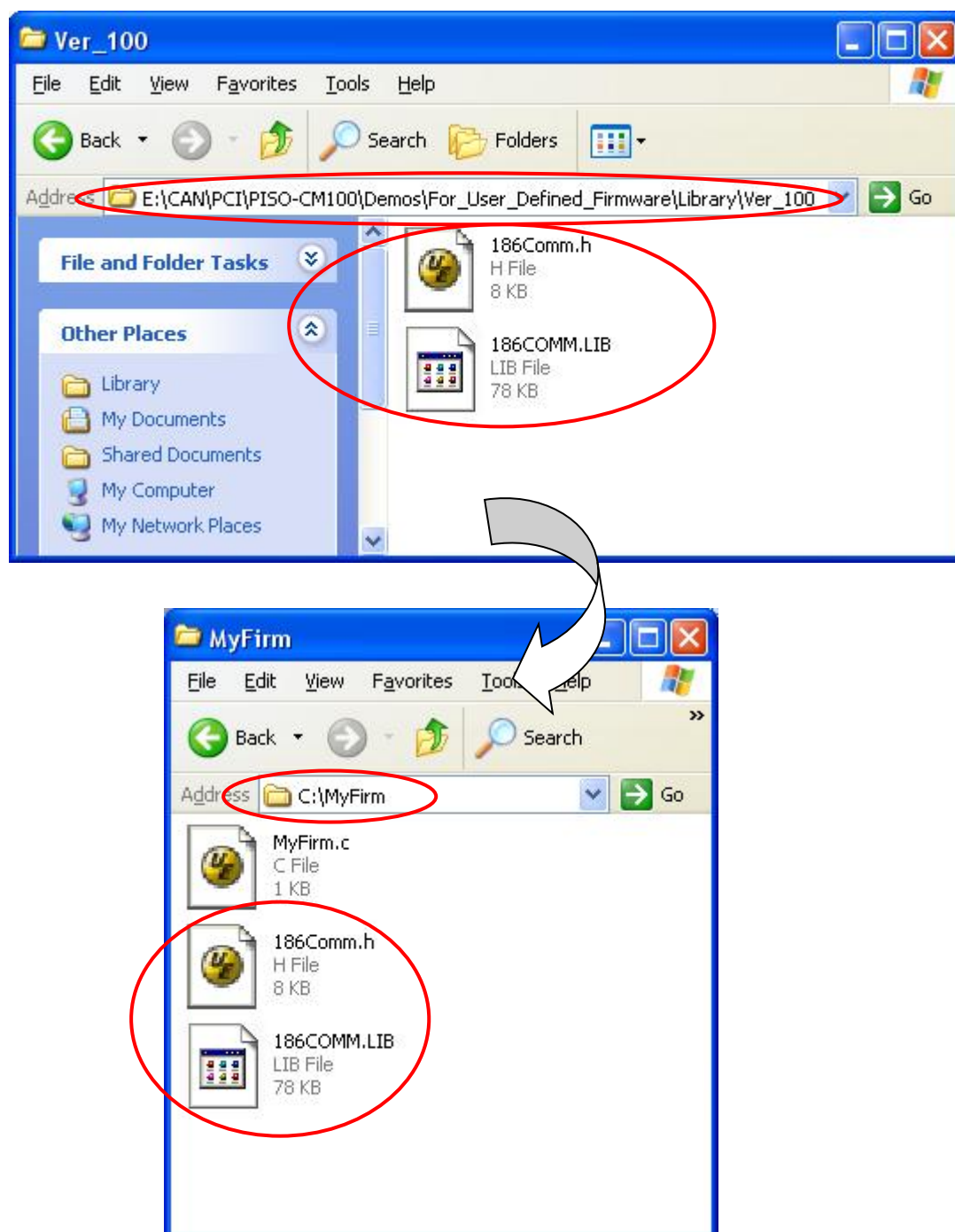
Step1: Create a folder named “MyFirm” in the C disk.



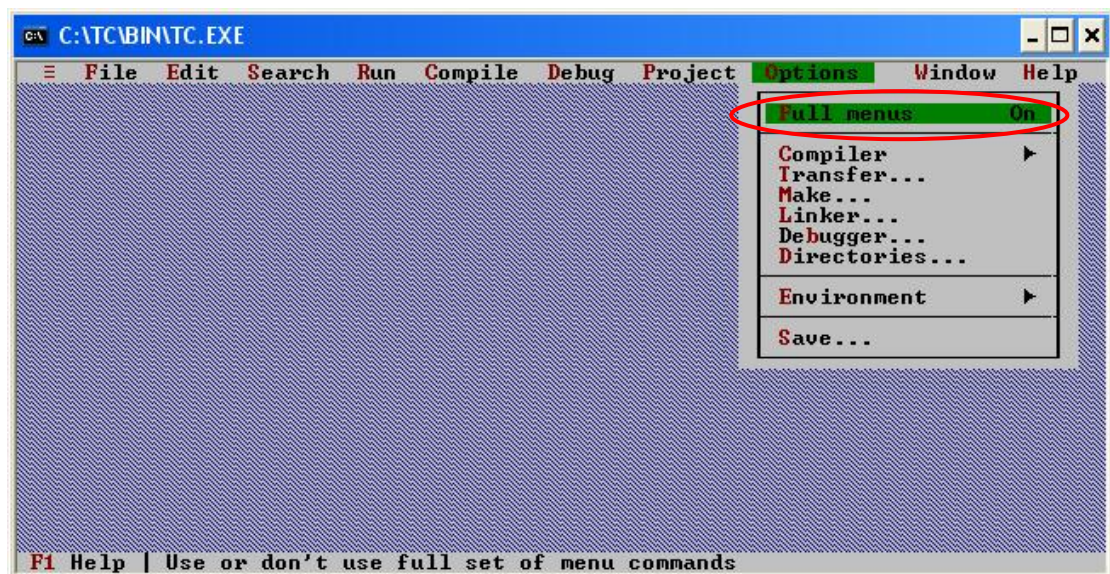
Step2: In the folder MyFirm, create a .c file and name it as “MyFirm.c”. Design the MyFirm.c file as follows. The 4 callback functions must be used in user-defined function.

```
MYFIRM - Notepad
File Edit Format View Help
#include "186COMM.h"
char CmdData[512];
unsigned int StartToRunFlag=0, CmdLen; //Global variable
void UserDPRAMIrqFunc(unsigned char INTT) //must be called
{
    //Do nothing
}
void UserCANIrqFunc(unsigned char INTT) //must be called
{
    //Do nothing
}
void UserInitFunc(void) //must be called
{
    Print("The user-defined firmware is initialized. \r\n");
}
void UserLoopFunc(void) //must be called
{
    if (StartToRunFlag==0){
        Print("The user-defined firmware is running.\r\n");
        StartToRunFlag=1;
    }
    if (DPRAMReceiveCmd(CmdData,&CmdLen)==_NO_ERR){
        Print("The first byte of the Cmd is %c\r\n",CmdData[0]);
    }
}
```

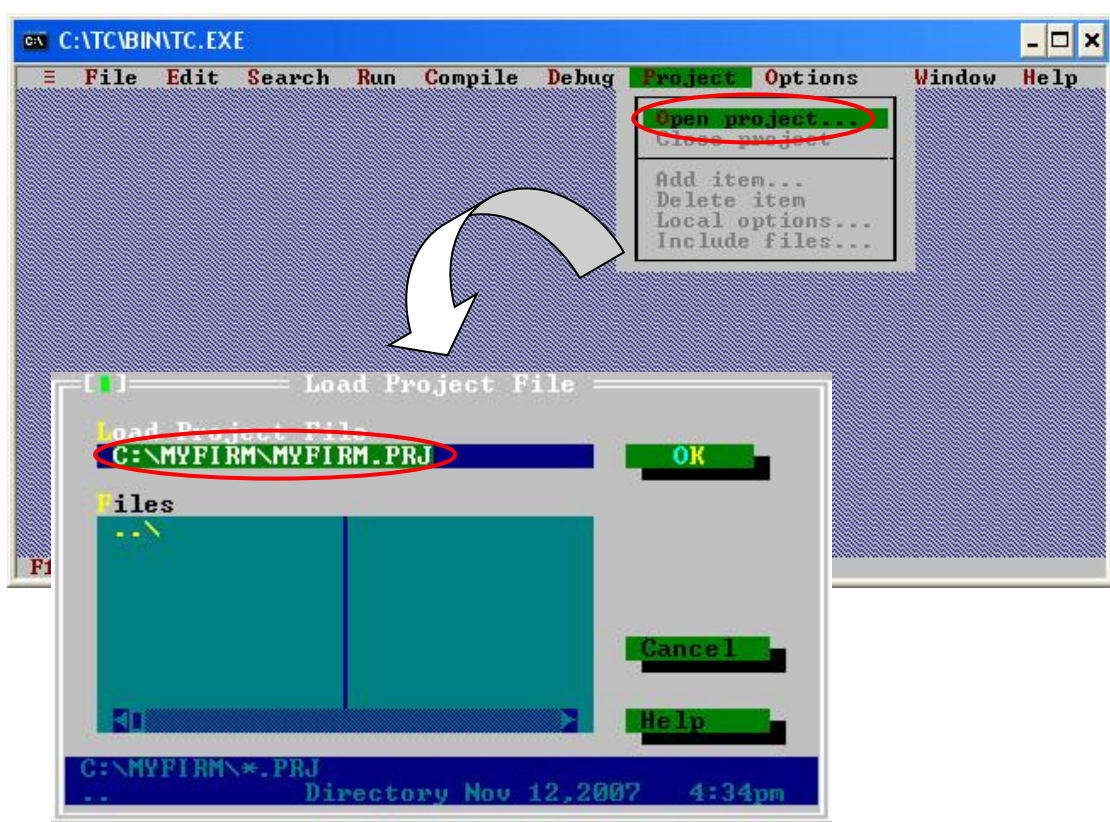
Step3: Copy the 186COMM.lib file and the 186COMM.h file into the MyFirm folder. Users can find them with version 1.00 in the path CAN\PCI\PISO-CM100U\Demos\For_User_Defined_Firmware\ver_100 in the Fieldbus CD. If there is newest version library, use it for the user-defined firmware.



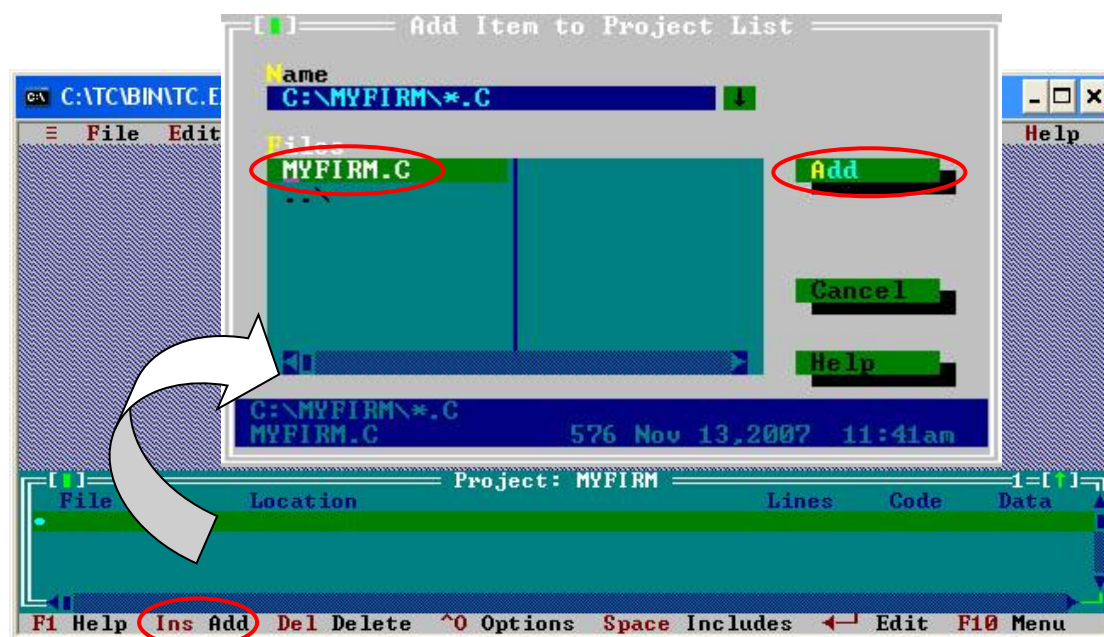
Step4: Run the TC++1.01 development environment. Click the “Options\Full menus” to expand the all functions listed in the menus.



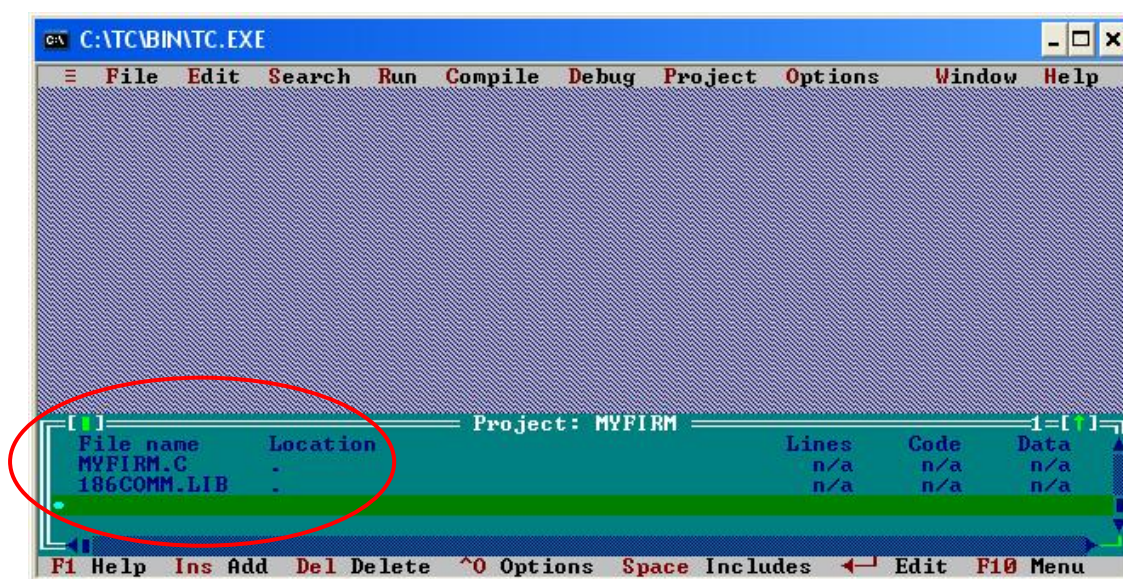
Step5: Click the “Project\Open project...” to create a new project. Input the project name “MyFirm.PRJ”, and click the OK button to continue.



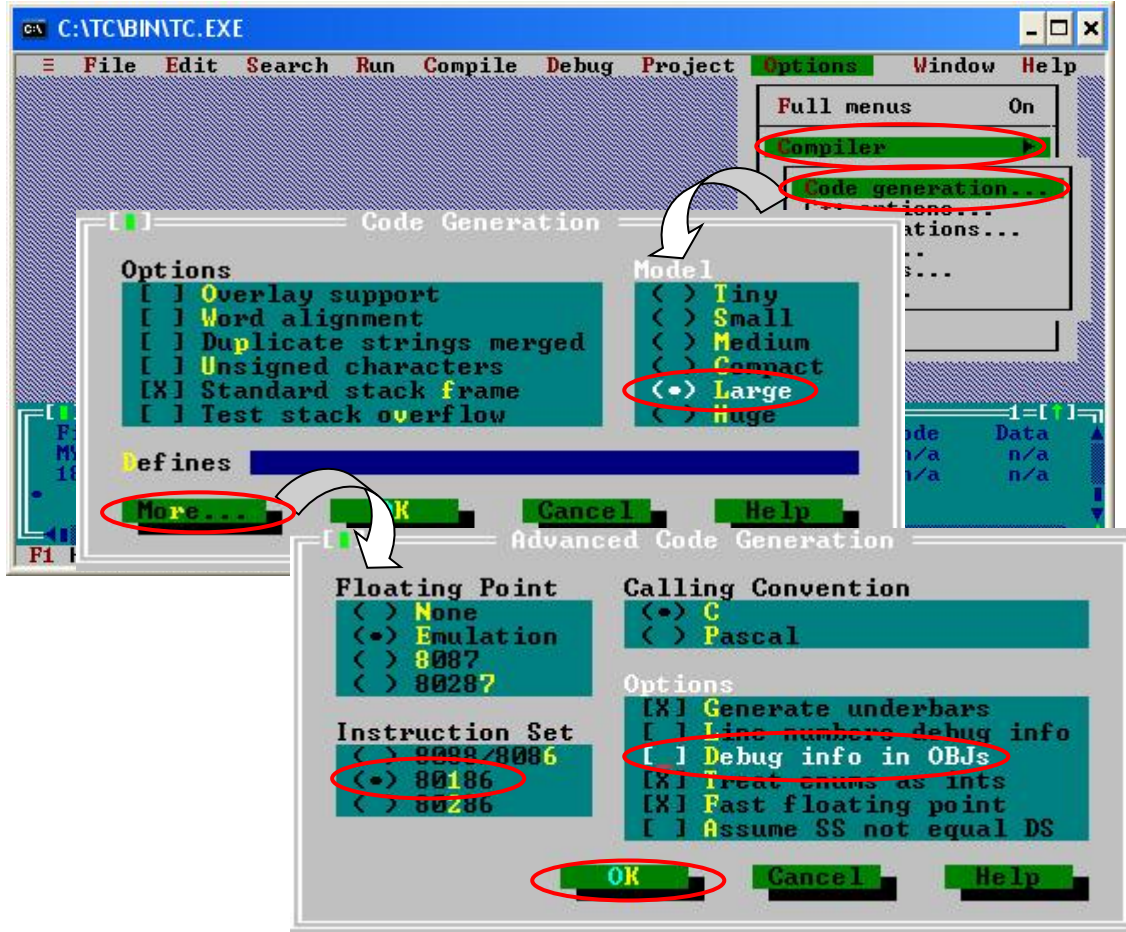
Step6: Click the “Add” item on the bottom of the TC++1.01 window. Search all .c file by setting “c:\MyFirm*.c” in the Name field of the popup window. Use the “Add” button to add the MyFirm’ .c file in to MyFirm project. Then, change the search command from “c:\MyFirm*.c” to “c:\MyFirm*.lib” in the Name field. Add the library file 186COMM.lib into the MyFirm project as the same way.



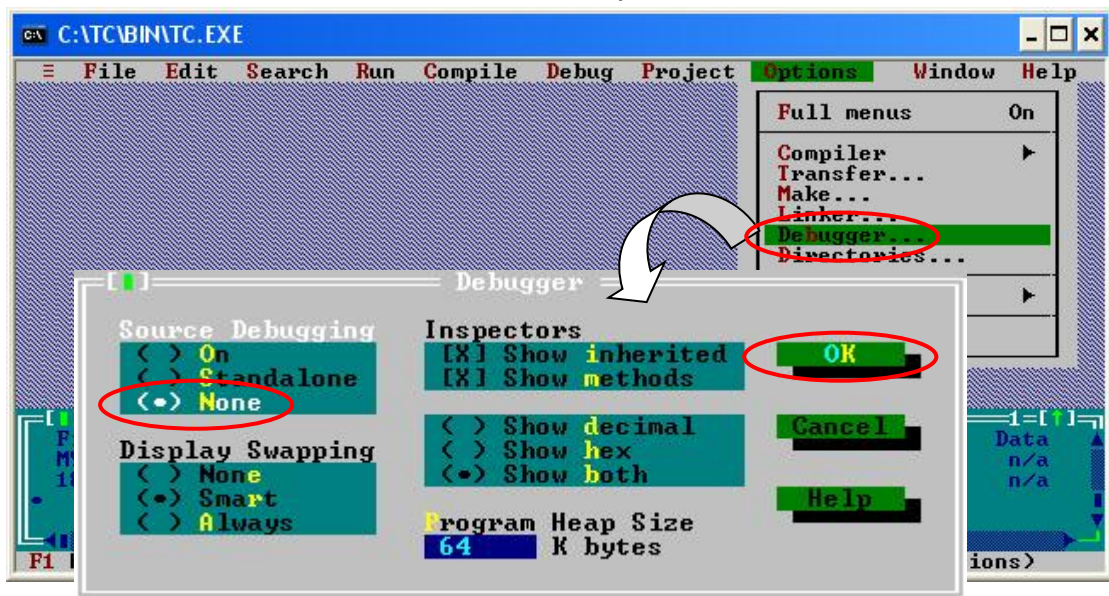
Step7: After finishing the Step6, the TC++1.01 window will look like as follows.



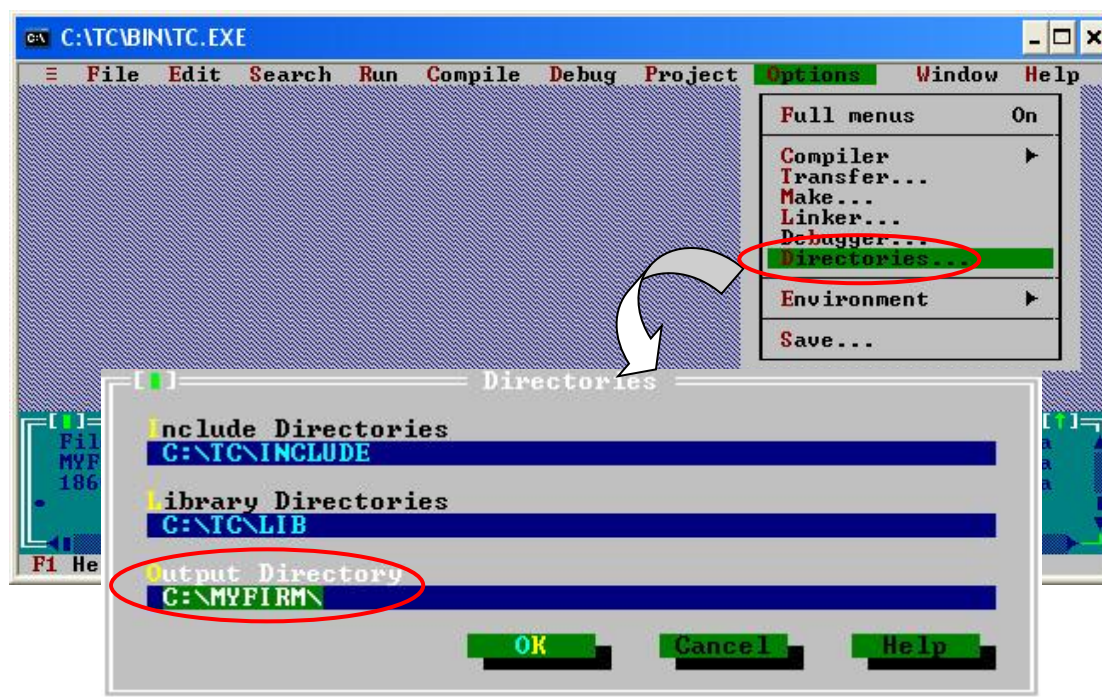
Step8: Click the “Options/Compiler/Code generation...” to set the compiler model to the large mode. Click “More...” to set the “Floating point” and “Instruction Set” parameters, the Emulation and 80186 item will be used respectively. Then, click the OK button to save the configuration.



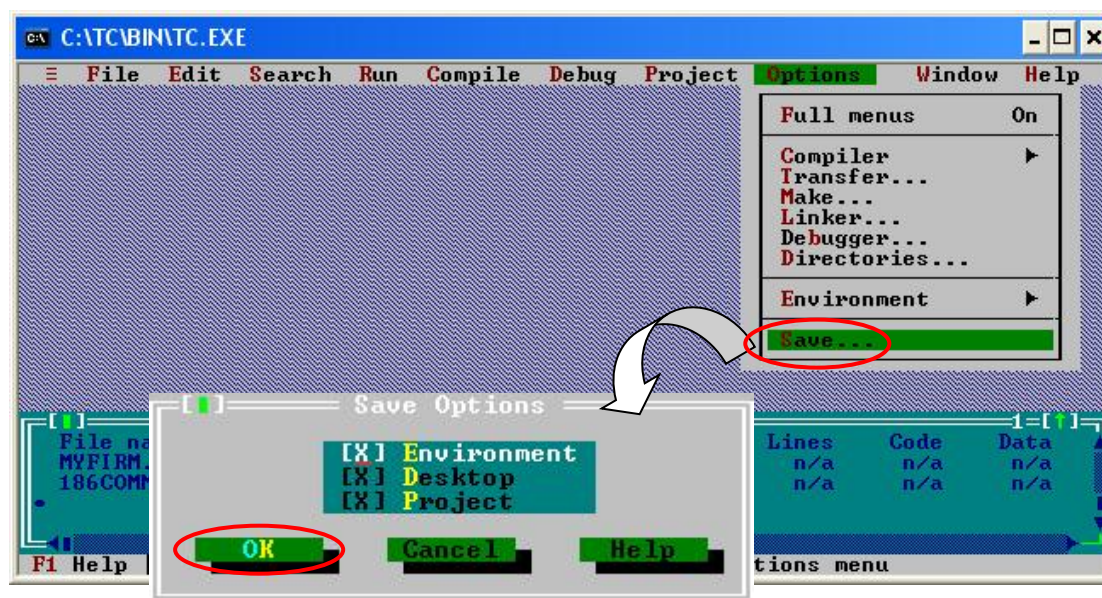
Step9: Click the “Option/Debugger...” to set the “Source Debugging” parameter. Here, select the “None” item for this parameter.



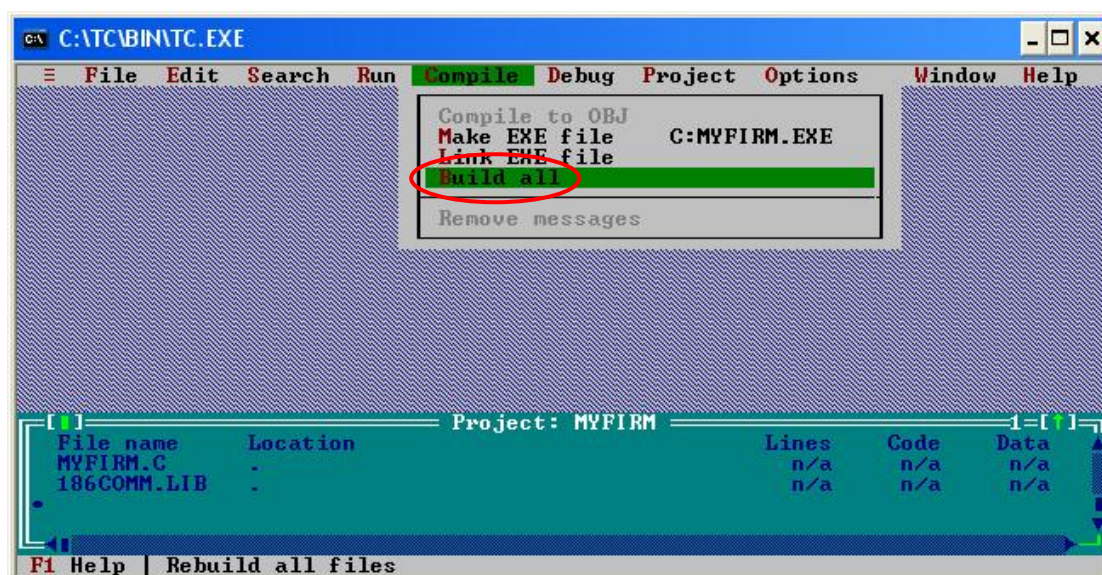
Step10: Click the “Option/Directories...” to set the “Output Directory” parameter.
Here, set the “C:\MyFirm” for the “Output Directory” parameter.



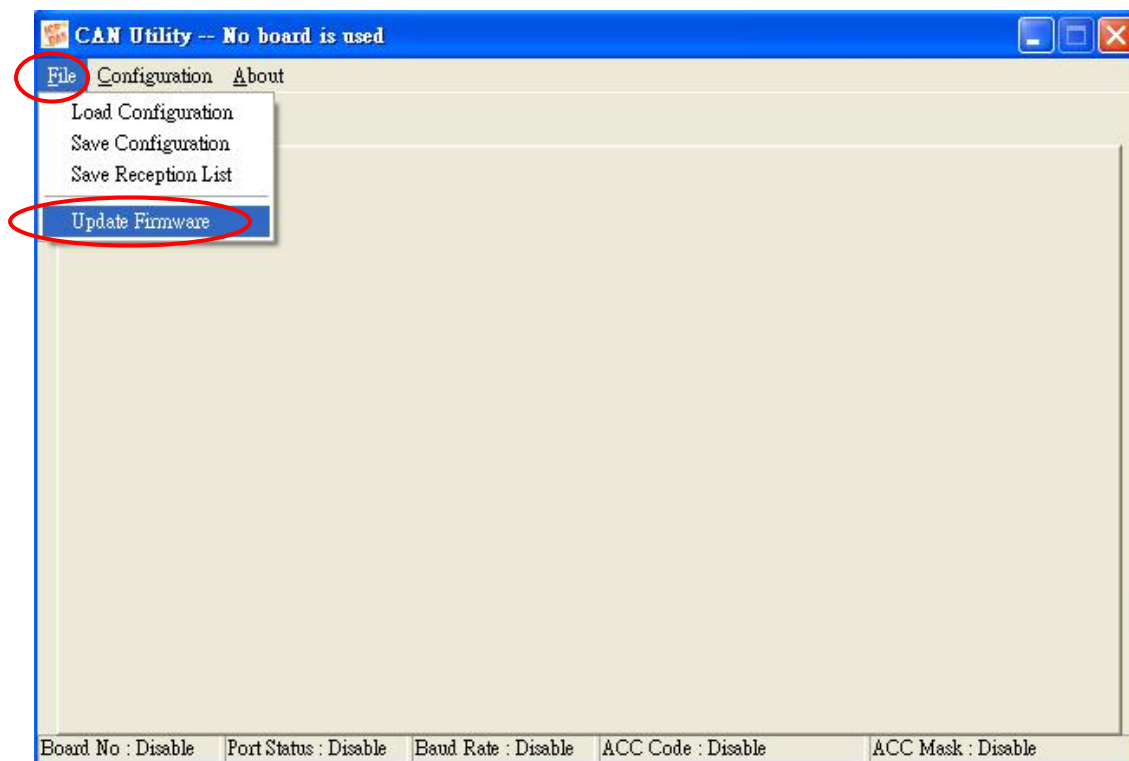
Step11: After finishing the parameters setting, click the “Options/save” to save this project.



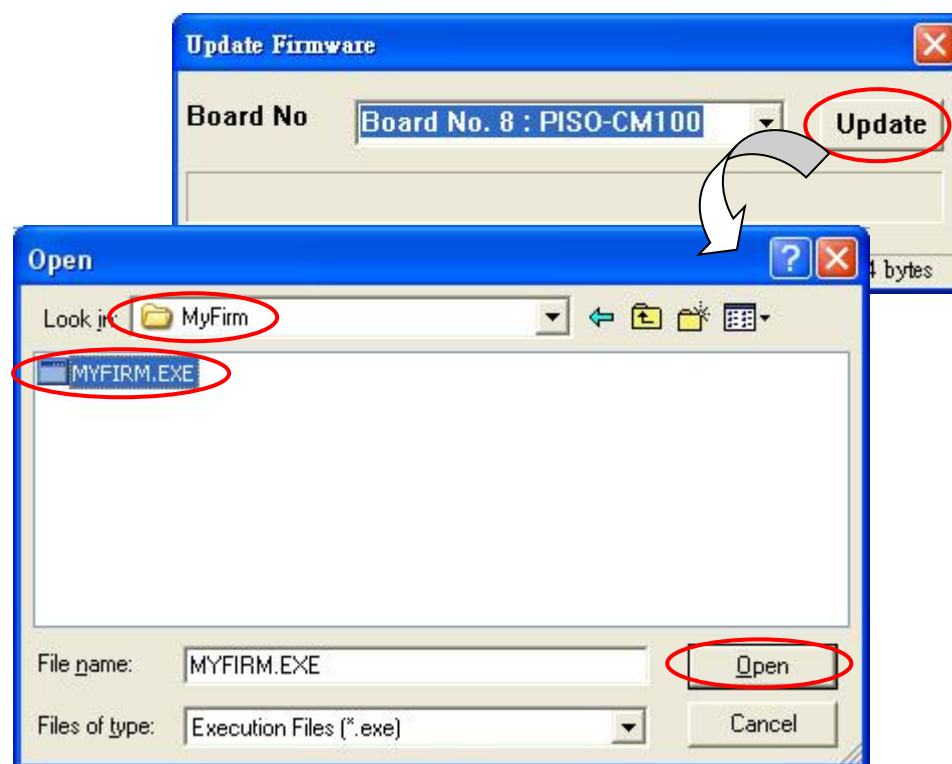
Step12: After finishing the parameters configuration, click the “Compile/build all” to produce the execution file. Users can find the execution file named as MyDemo.exe in the MyFirm folder. The warning messages may occur during the compiling procedure because the INTT parameters of the UserCANIrqFunc() and UserDPRAMIrqFunc() are not used. These warnings will not have any affection.



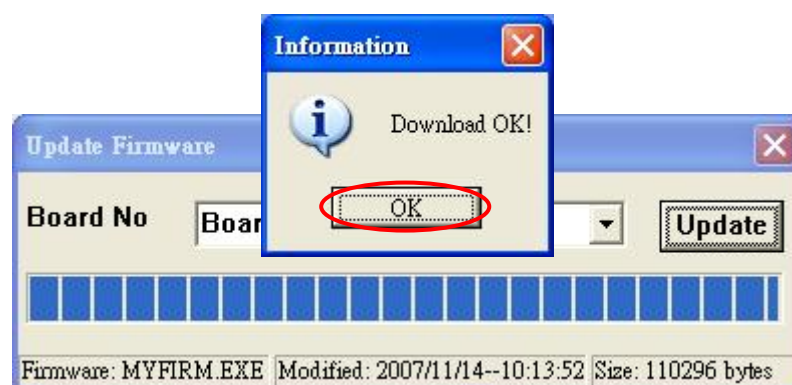
Step13: Execute the CANUtility.exe, and select the “File\Update Firmware” to download the user-defined firmware.



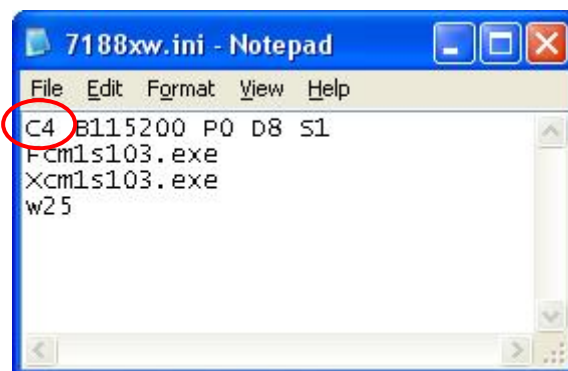
Step14: Select the board name, click the “Update” button, and find the MyFirm.exe file from the dialog.



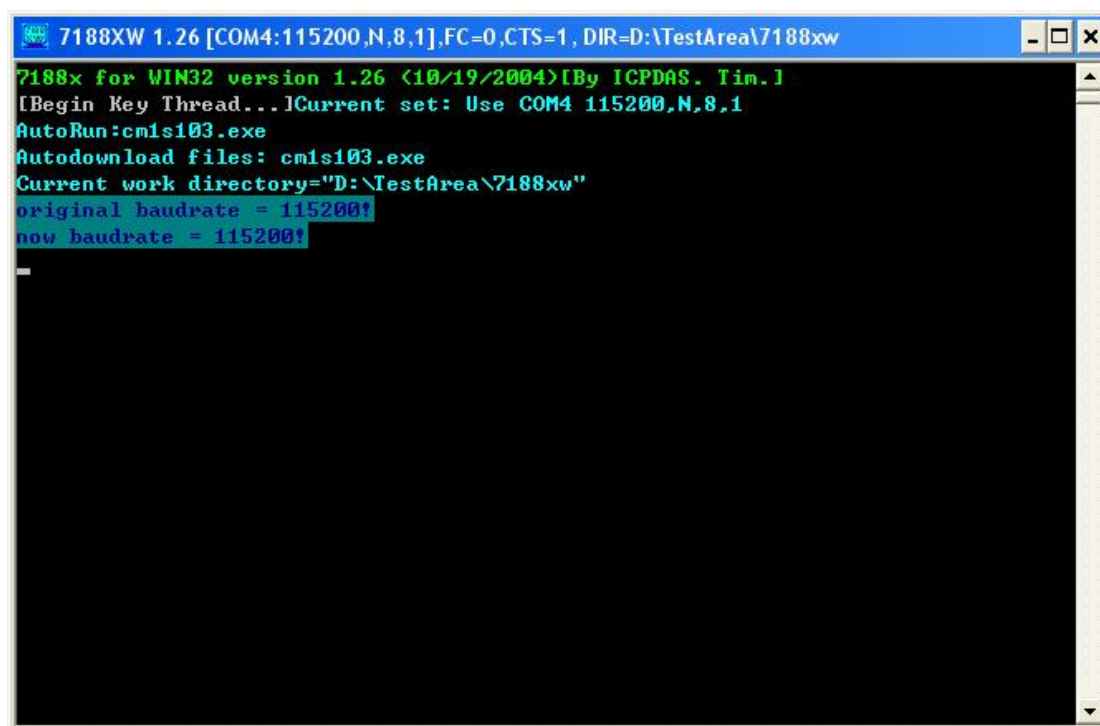
Step15: When finishing, the Download OK messages is shown.



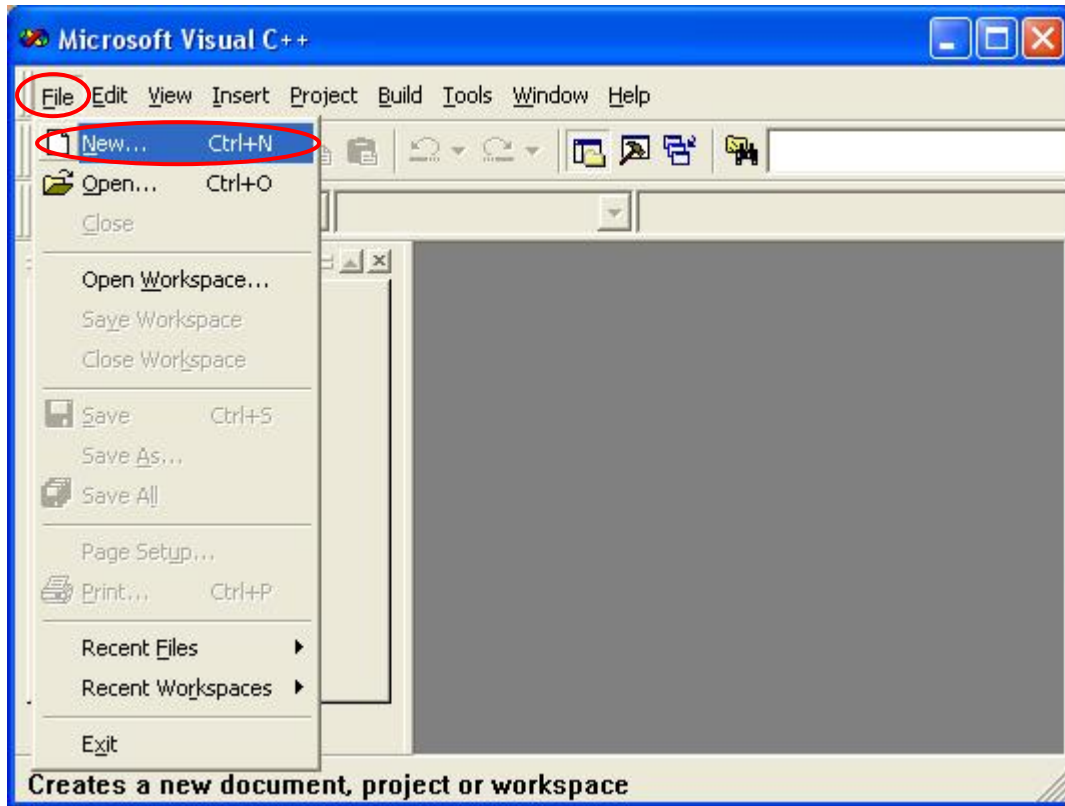
Step16: Check the 7188xw.ini. Here, use the PC COM4 to connect the debug cable of the CM100s board. Set the COM No. value to "C4" for the PC COM4. If users use COM1, set the value to "C1". The 7188xw.ini and 7188xw.exe can be found in the driver installation path. The default is "C:\ICPDAS\PISO-CM100U\".



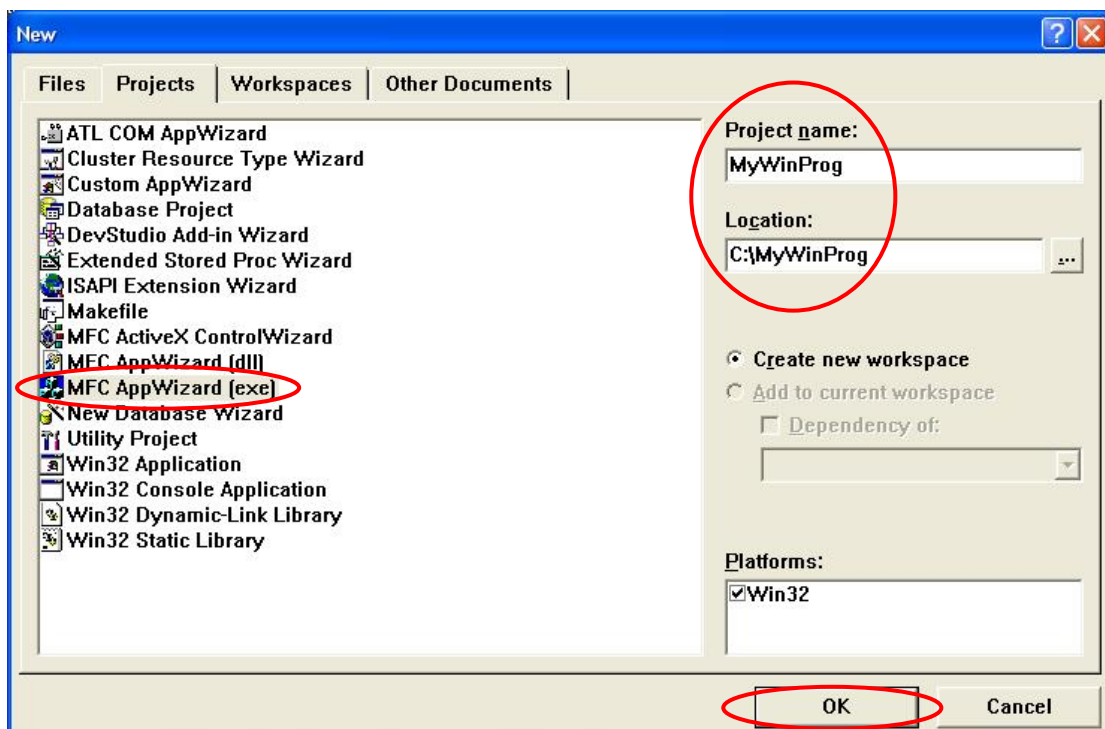
Step17: Connect the debug cable introduced in the chapter 2 with the CAN board. Run the 7188xw.exe.



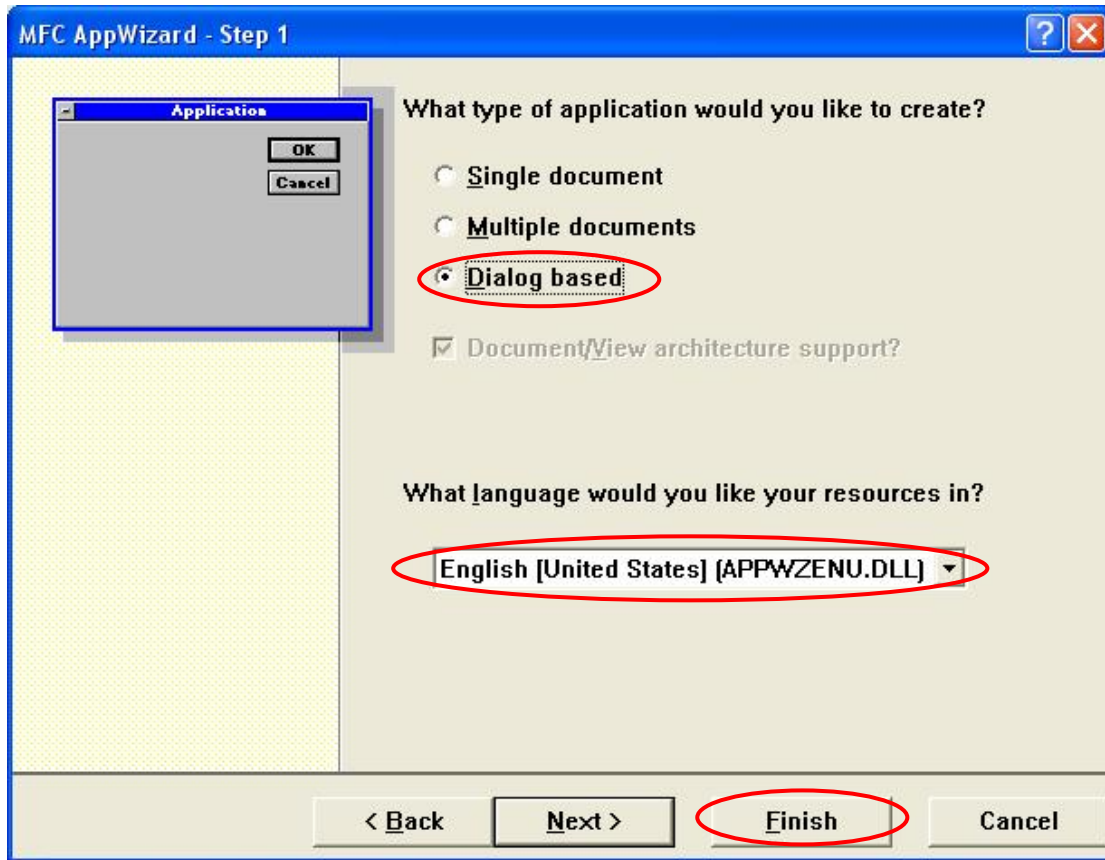
Step18: After finishing the user-defined firmware, run the VC++ 6.0 to set up a corresponding Windows program. Select the “File/New...” to create a project.



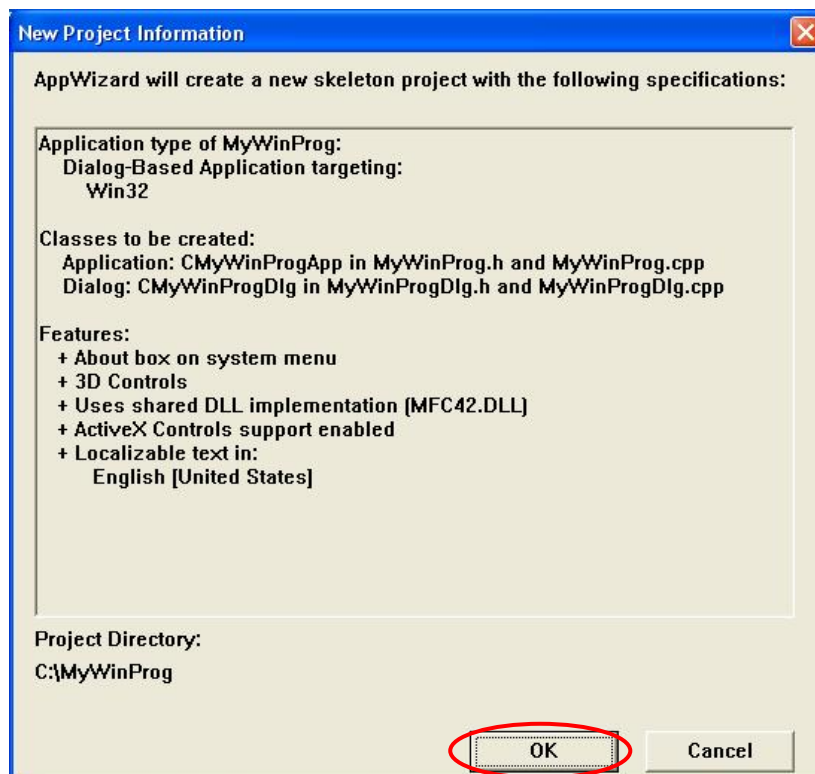
Setp19: Use the “MFC AppWizard (exe) for the project. The project name and location are given as following figure. Click the OK button to continue.



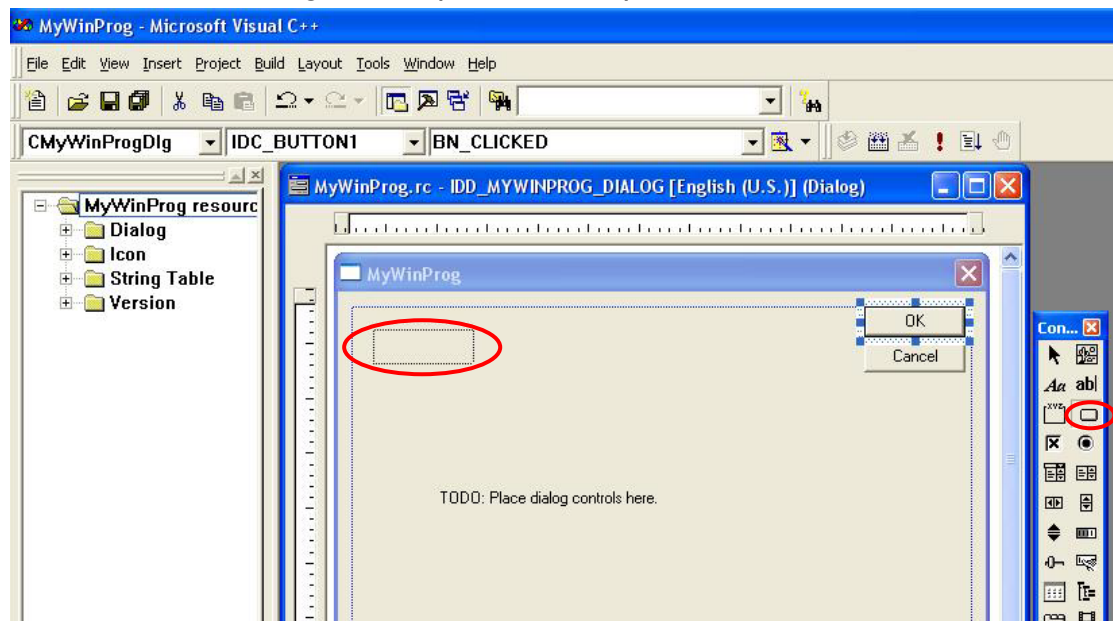
Step20: Set the project by using the following parameters, and click the Finish button to continue.



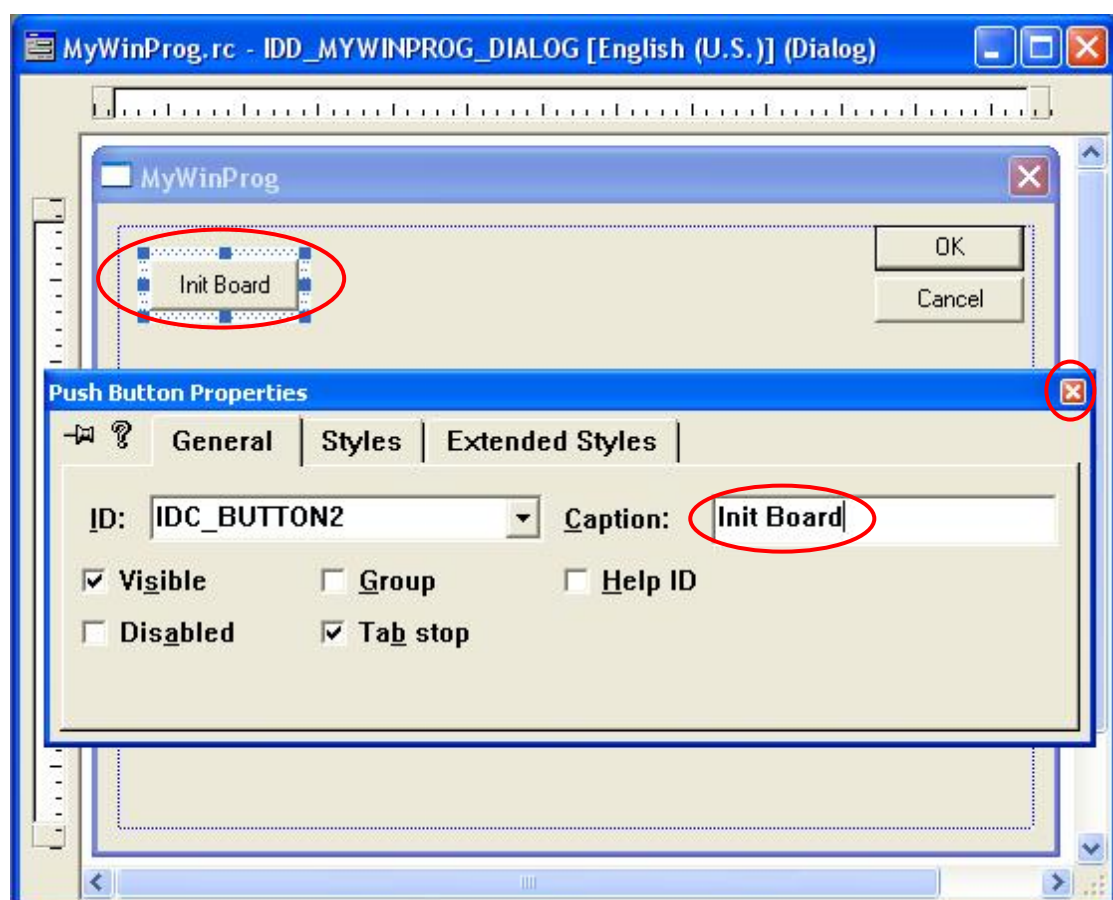
Step21: The relative project information is shown in the popped up dialog.



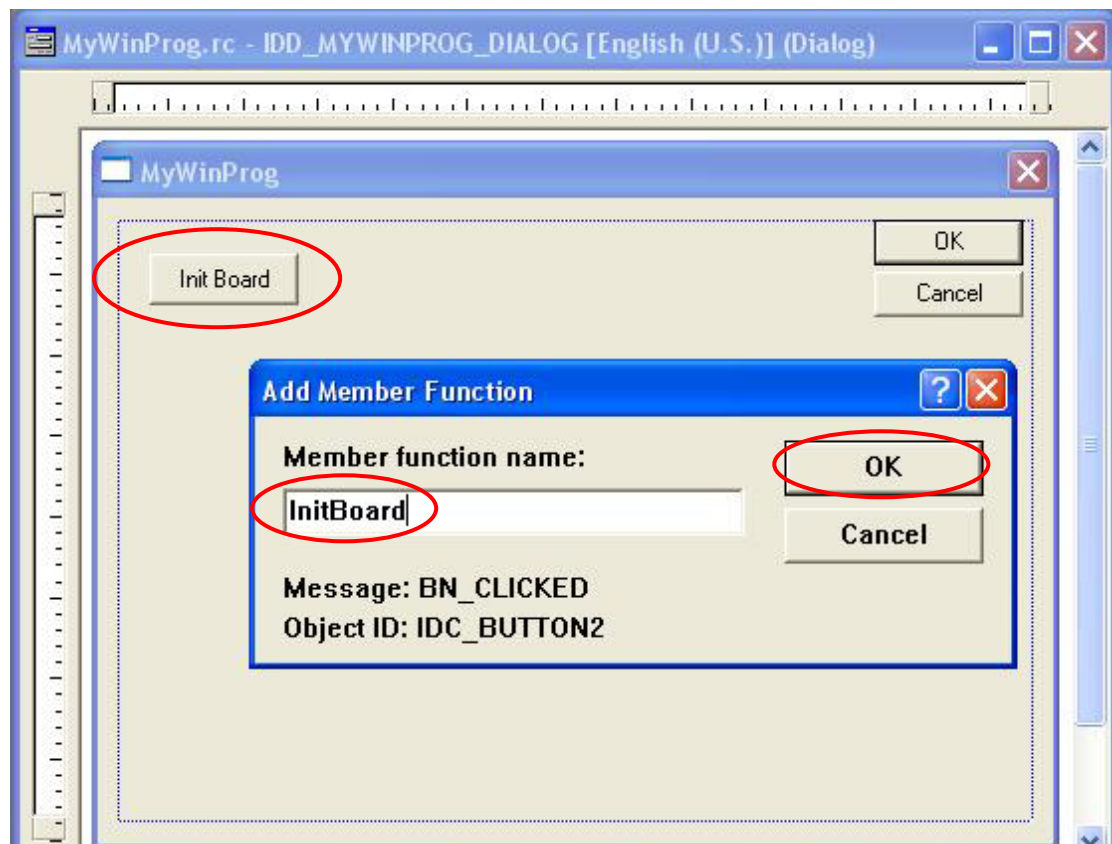
Step22: Add a button to the Windows program. Select the button icon from the tool box, drag and drop the mouse pointer to build a button.



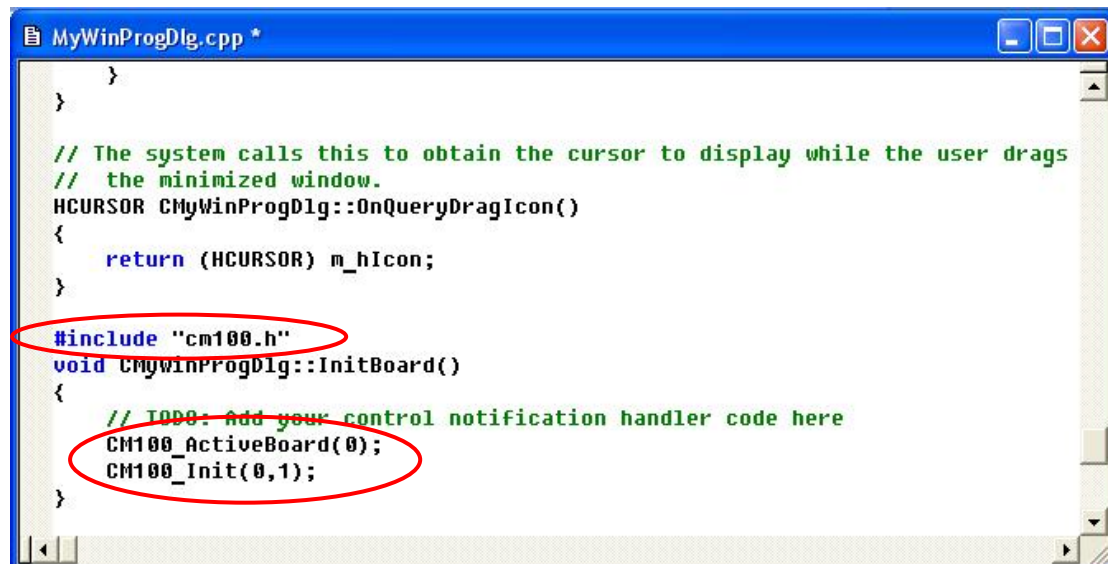
Step23: Select the button, and key the characters "Init Board" directly. Then the property dialog is popped up. Afterwards, close the property dialog.



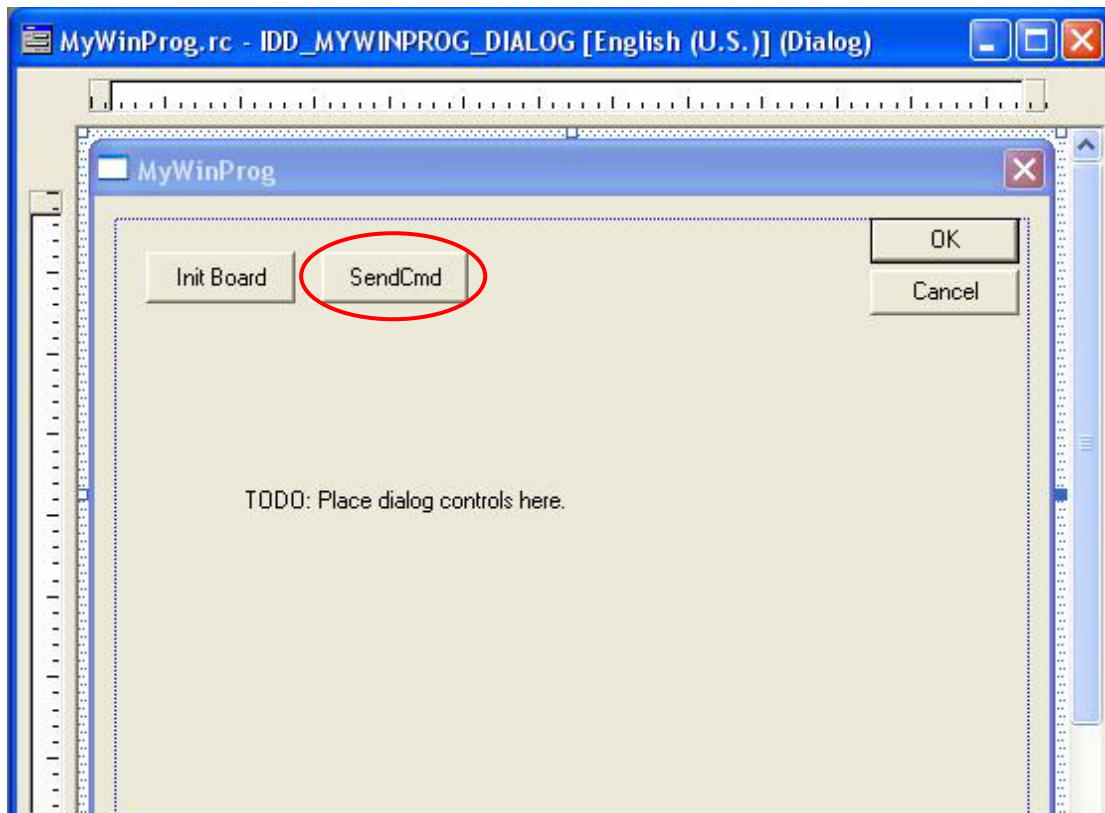
Step24: Double click the button “Init Board”, and key the characters “InitBoard” for creating a member function. Click the OK button to continue.



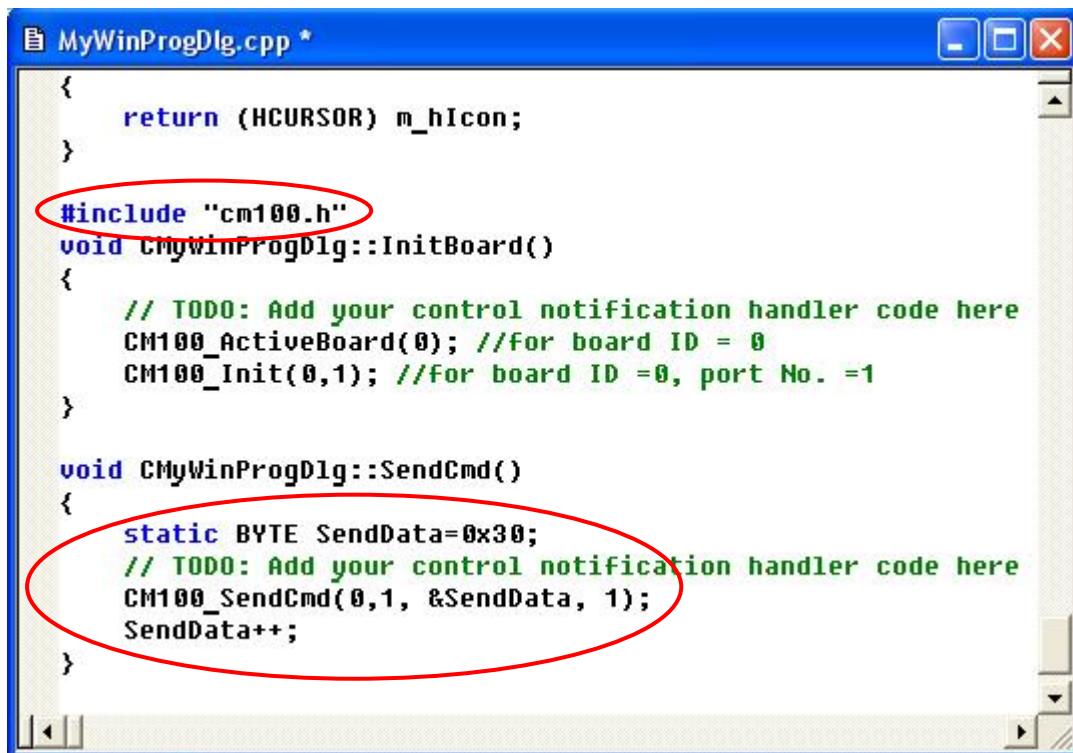
Step25: Add the codes in the member function.



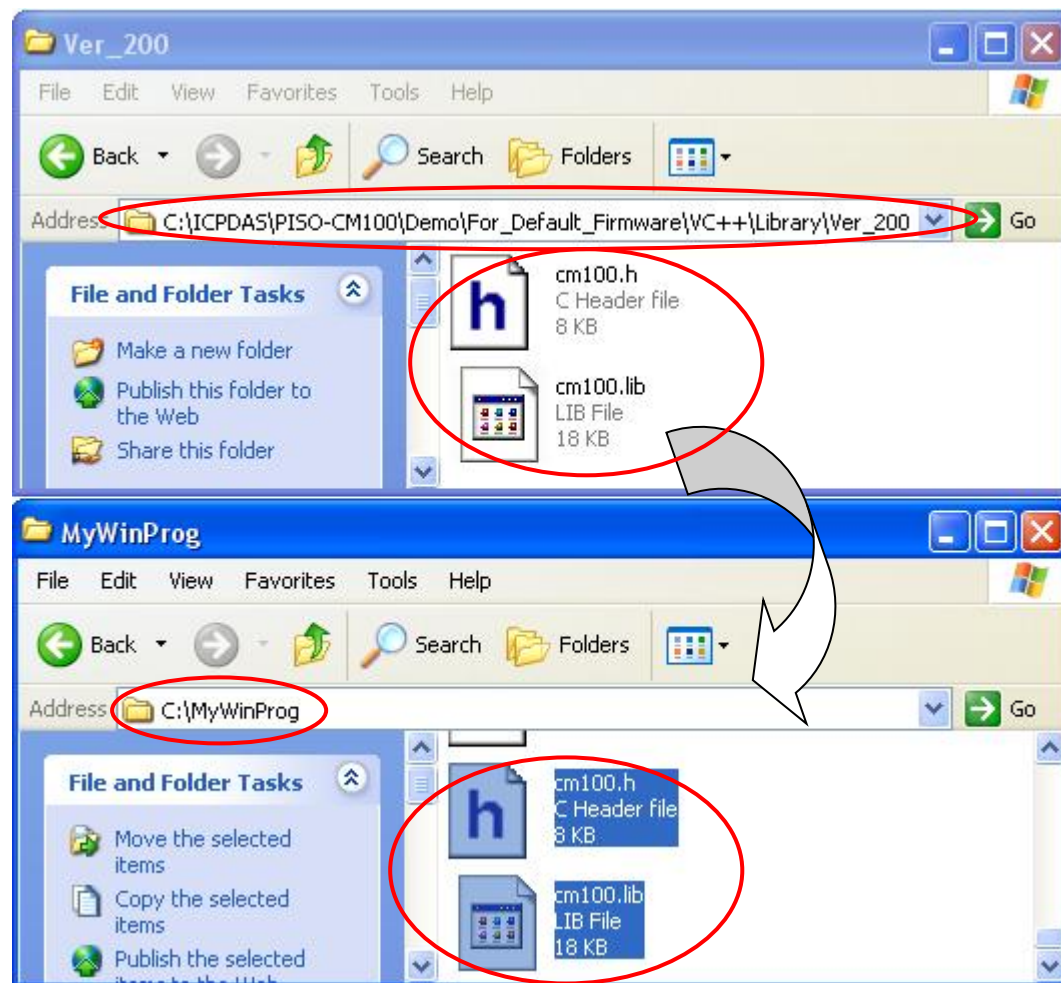
Step26: Repeater Step 22 to Step 25 to create another button named as “SendCmd”, and add a member function named as “SendCmd” for this button.



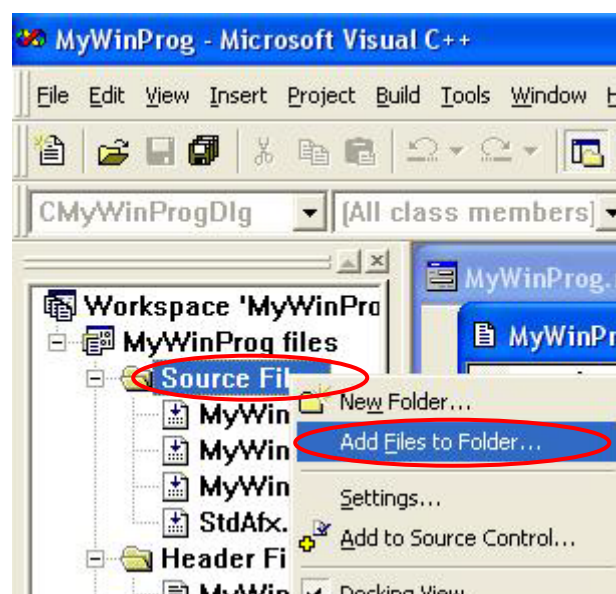
Step27: The codes of the member function are as follows.



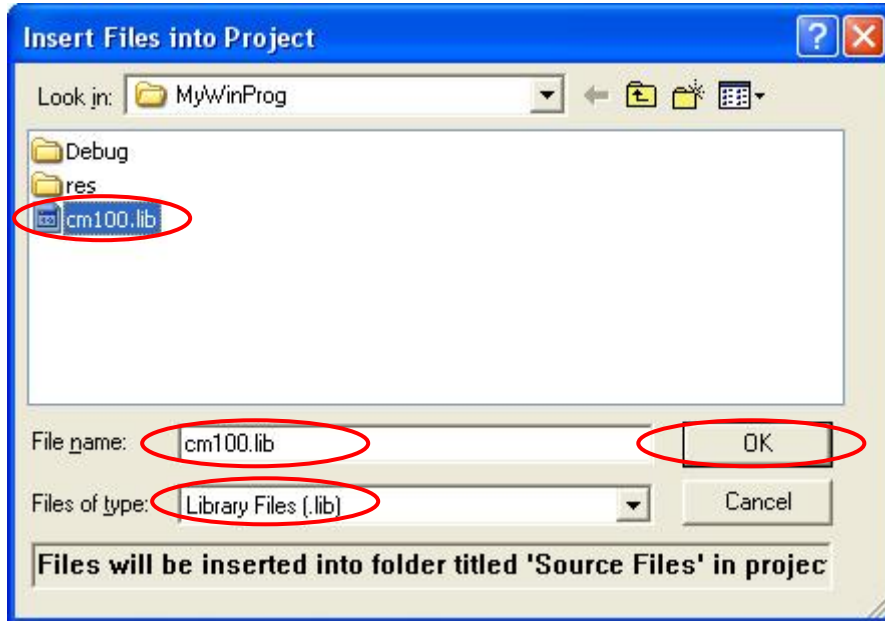
Step28: Copy the cm100.h and cm100.lib files to the folder of the project. You can find the two files in the PISO-CM100U driver installation path.



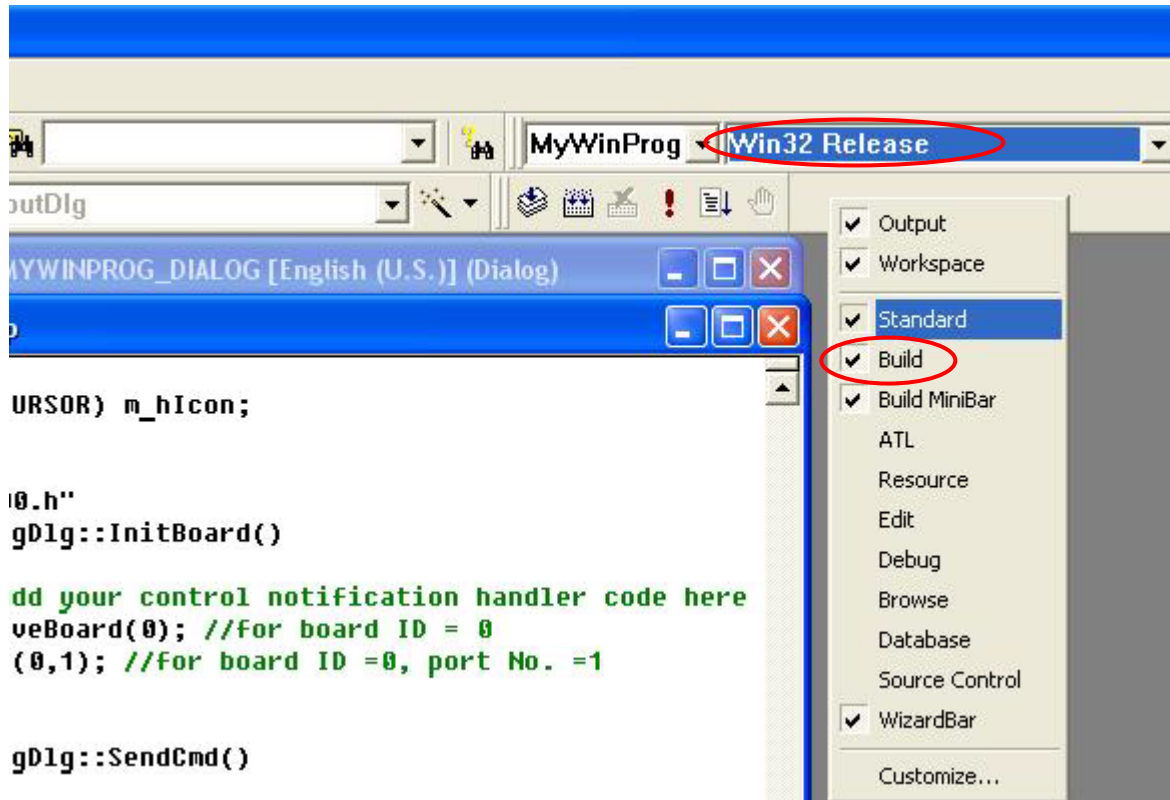
Step29: In order to add the CM100.lib into your project. Right click on the "Source File" item. Select the "Add Files to Folder" item.



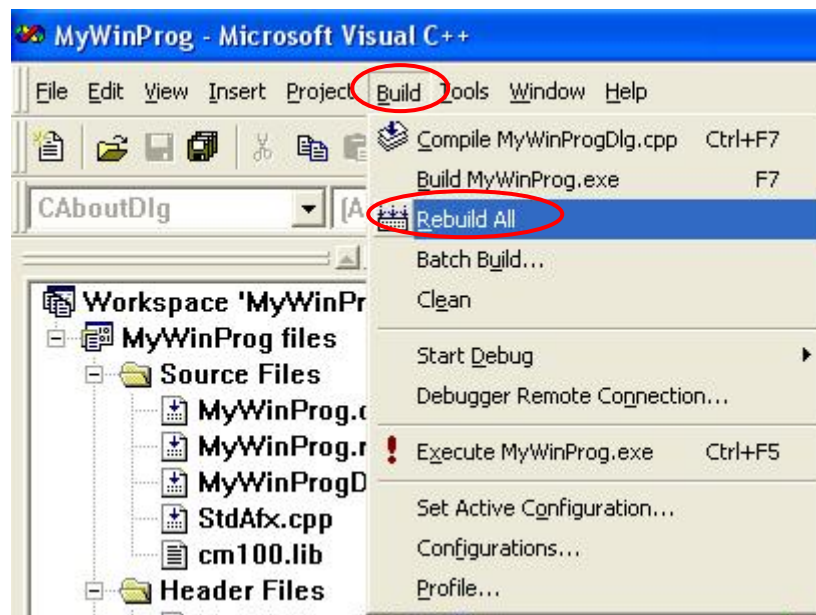
Step30: Select the “Files of type” to the “Library Files (.lib)” first. Add the cm100.lib into your project. Click the OK button to continue.



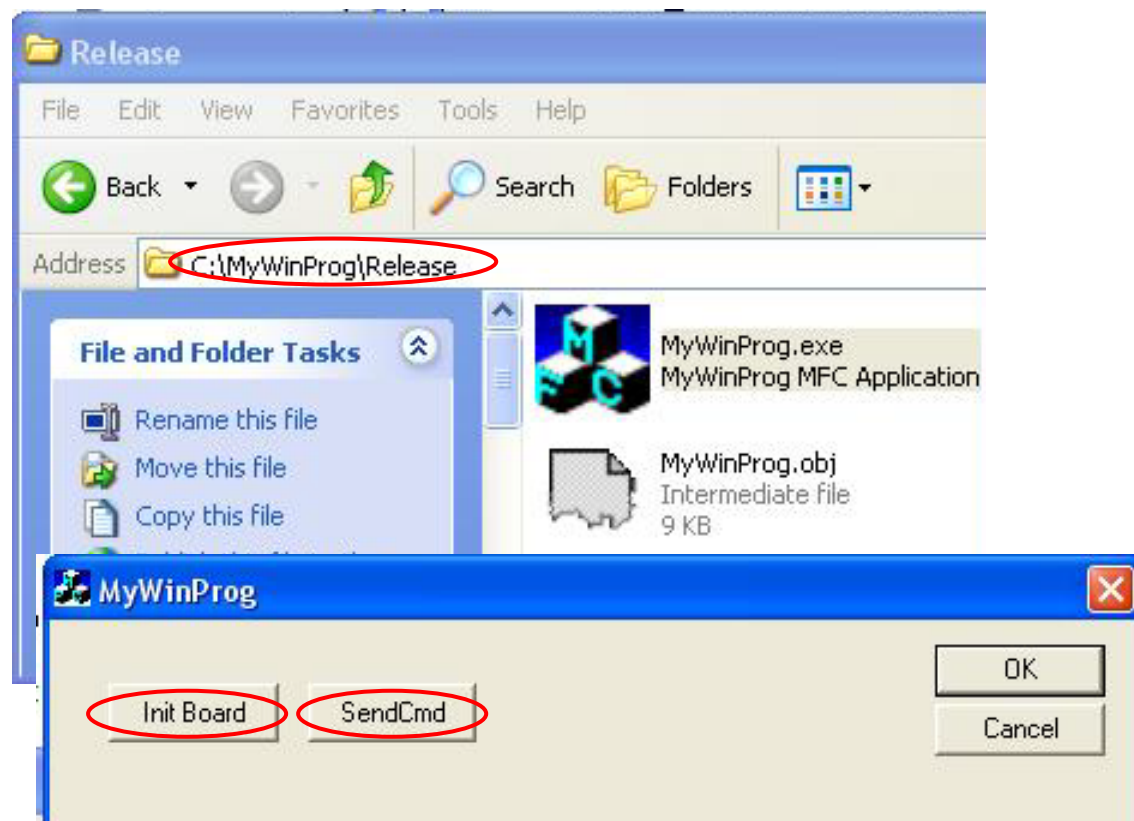
Step31: In the space of the toolbar, right click to add the “Build” toolbar. Select the “Win32 Release”.



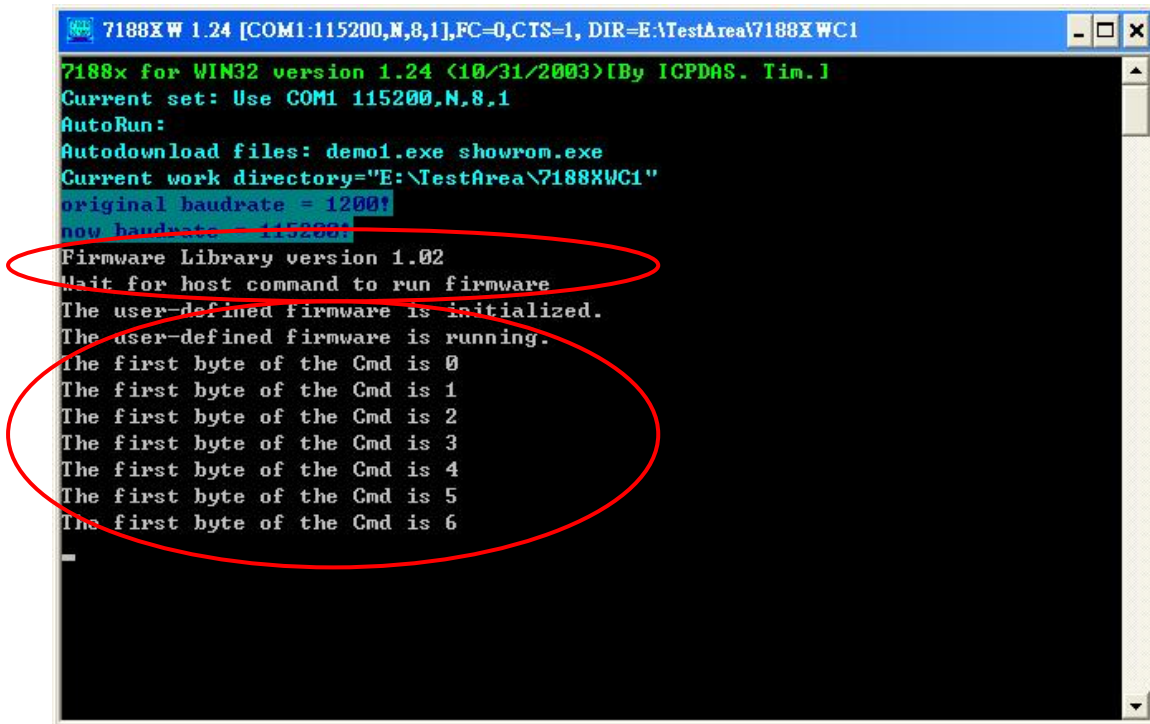
Step32: Select the “Build/Rebuild All” in the menu to compile the project.



Step33: Set the CAN board ID to 0, and run the MyWinProg.exe in the Release folder of the project. Click the “Init Board” button first. Then, use the “SendCmd” button to send the command to the user-defined firmware.



Step34: Each click on the “SendCmd” button of the Windows program sends a command to the user-defined firmware, and the command will be caught by the user-defined firmware. The first two lines of the debug information are shown by the CAN board system. The other debug messages are shown by the user-defined firmware.



```
7188XW 1.24 [COM1:115200,N,8,1],FC=0,CTS=1, DIR=E:\TestArea\7188XWC1
7188x for WIN32 version 1.24 (10/31/2003)[By ICPDAS. Tim.]
Current set: Use COM1 115200,N,8,1
AutoRun:
Autodownload files: demo1.exe showrom.exe
Current work directory="E:\TestArea\7188XWC1"
original baudrate = 1200!
now baudrate = 115200!
Firmware Library version 1.02
Wait for host command to run firmware
The user-defined firmware is initialized.
The user-defined firmware is running.
The first byte of the Cmd is 0
The first byte of the Cmd is 1
The first byte of the Cmd is 2
The first byte of the Cmd is 3
The first byte of the Cmd is 4
The first byte of the Cmd is 5
The first byte of the Cmd is 6
```