



PISO-CPM100-DT

PISO-CPM100U-DT

CANopen Master PCI Card

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 2008 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.....	5
1.1. CANopen Introduction.....	5
1.2. CANopen Applications	6
1.3. PISO-CPM100 Library Characteristics	7
2. Hardware Configuration	10
2.1. Board Layout.....	10
2.2. Jumper Selection	11
2.3. Connector Pin Assignment	12
2.3.1. 5-pin screw terminal connector.....	12
2.3.2. 9-pin D-sub female connector.....	13
2.3.3. Wire Connection	14
2.4. Green LED	15
2.5. Red LED	15
2.6. Hardware Installation	15
3. Software Installation.....	16
3.1. Software Structure	16
3.2. Installation Driver Step by Step	17
4. PISO-CPM100 Function Library.....	21
4.1. Function List	21
4.2. Function Return Code	23
4.3. CANopen Master Library Application Flowchart.....	24
4.4. Communication Services Introduction	26
4.5. Function Description	29
4.5.1. CPM100_GetVersion	29
4.5.2. CPM100_TotalBoard	30
4.5.3. CPM100_GetBoardSwitchNo.....	31
4.5.4. CPM100_GetBoardInf.....	32
4.5.5. CPM100_ActiveBoard.....	33
4.5.6. CPM100_BoardIsActive.....	34
4.5.7. CPM100_CloseBoard	35
4.5.8. CPM100_GetCANStatus	36
4.5.9. CPM100_InitMaster	38
4.5.10. CPM100_ShutdownMaster.....	39
4.5.11. CPM100_AddNode.....	40
4.5.12. CPM100_RemoveNode	41
4.5.13. CPM100_NMTChangeState	42



4.5.14.	CPM100_NMTGetState.....	43
4.5.15.	CPM100_NMTGuarding	44
4.5.16.	CPM100_SendSYNC	45
4.5.17.	CPM100_GetSYNCingID	46
4.5.18.	CPM100_ChangeEMCYID	47
4.5.19.	CPM100_ChangeSYNCID	48
4.5.20.	CPM100_ReadEMCYCount.....	49
4.5.21.	CPM100_ReadEMCY	50
4.5.22.	CPM100_SDOAobrtTransmit	51
4.5.23.	CPM100_SDOReadData	52
4.5.24.	CPM100_SDOWriteData.....	53
4.5.25.	CPM100_DynamicPDO	55
4.5.26.	CPM100_InstallPDO	57
4.5.27.	CPM100_RemovePDO.....	58
4.5.28.	CPM100_PDOTxType.....	59
4.5.29.	CPM100_SetEventTimer	60
4.5.30.	CPM100_ChangePDOCobID	61
4.5.31.	CPM100_PDOWrite	62
4.5.32.	CPM100_PDORemote.....	63
4.5.33.	CPM100_ReadPDOCCount	64
4.5.34.	CPM100_ReadPDOMessage.....	65
4.5.35.	CPM100_WriteDO	66
4.5.36.	CPM100_ReadDI.....	67
4.5.37.	CPM100_WriteAO	68
4.5.38.	CPM100_ReadAI.....	69
4.5.39.	CPM100_GetFirmwareVersion.....	70
4.5.40.	CPM100_COBIDInfo	71
4.5.41.	CPM100_PDOMappingInfo	72
4.5.42.	CPM100_GetNodeList	74
5.	Demo Programs	75
5.1.	Brief of the demo programs	75
6.	CPM_ Utility Introduction	90
6.1.	Board Configure.....	91
6.2.	Node Configuration.....	92
6.3.	Refresh slave parameter.....	101
6.4.	DI/DO control	102
6.5.	AI/AO control	103
6.6.	Save receive messages	104



6.7.	Save/Load CPM_Utility Setting	105
6.8.	About us	105

1. General Information

1.1. CANopen 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 an 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. CANopen is one kind of the network protocols based on the CAN bus and it is applied in a low level network that provides connections between simple industrial devices (sensors, actuators) and higher-level devices (controllers), as shown in Figure 1.1.

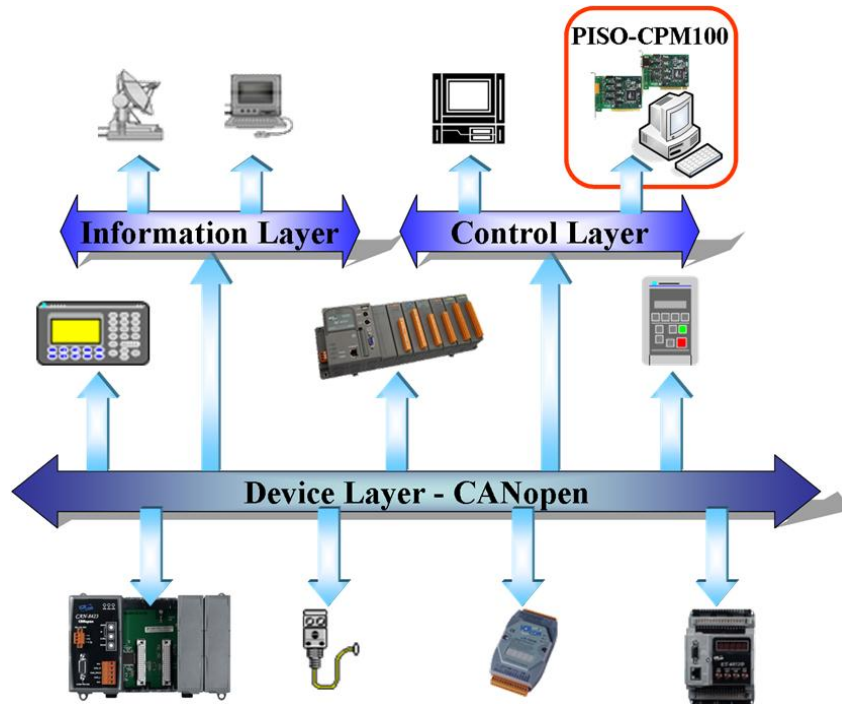


Figure 1.1 Example of the CANopen network

CANopen was developed as a standardized embedded network with highly flexible configuration capabilities. It provides standardized communication objects for real-time data (Process Data Objects, PDO), configuration data (Service Data Objects, SDO), network management data (NMT message, and Error Control), and special functions (Time Stamp, Sync message, and Emergency message). Nowadays, CANopen is used in many various application fields, such as medical equipment, off-road vehicles, maritime electronics, public transportation, building automation and so on.

1.2. CANopen Applications

CANopen is the standardized network application layer optimized for embedded networks. Its specifications cover the standardized application layer, frameworks for the various applications (e.g. general I/O, motion control system, maritime electronics and so forth) as well as device, interface, and application profiles.

The main CANopen protocol and products are generally applied in the low-volume and mid-volume embedded systems. The following examples show some parts of the CANopen application fields. (For more information, please refer to the web site, <http://www.can-cia.org>):

- Truck-based superstructure control systems
- Off-highway and off-road vehicles
- Passenger and cargo trains
- Maritime electronics
- Factory automation
- Industrial machine control
- Lifts and escalators
- Building automation
- Medical equipment and devices
- Non-industrial control
- Non-industrial equipment



1.3. PISO-CPM100(U) Library Characteristics

In order to use the PCI CAN board of PISO-CPM100(U), we provide CPM100 library for VC, VB and BCB development, and users can use it to establish the CANopen communication network rapidly. Most of the CANopen communication protocols, such as PDO, SDO and NMT, would be handled by the library function automatically. Therefore, it is helpful to reduce the complexity of developing a CANopen master interface, and let users ignore the detail CANopen protocol technology. This library mainly supports connection sets of master-slave architecture, which include some useful functions to control the CANopen slave device in the CANopen network. The following figure describes the general application architecture of PISO-CPM100(U).

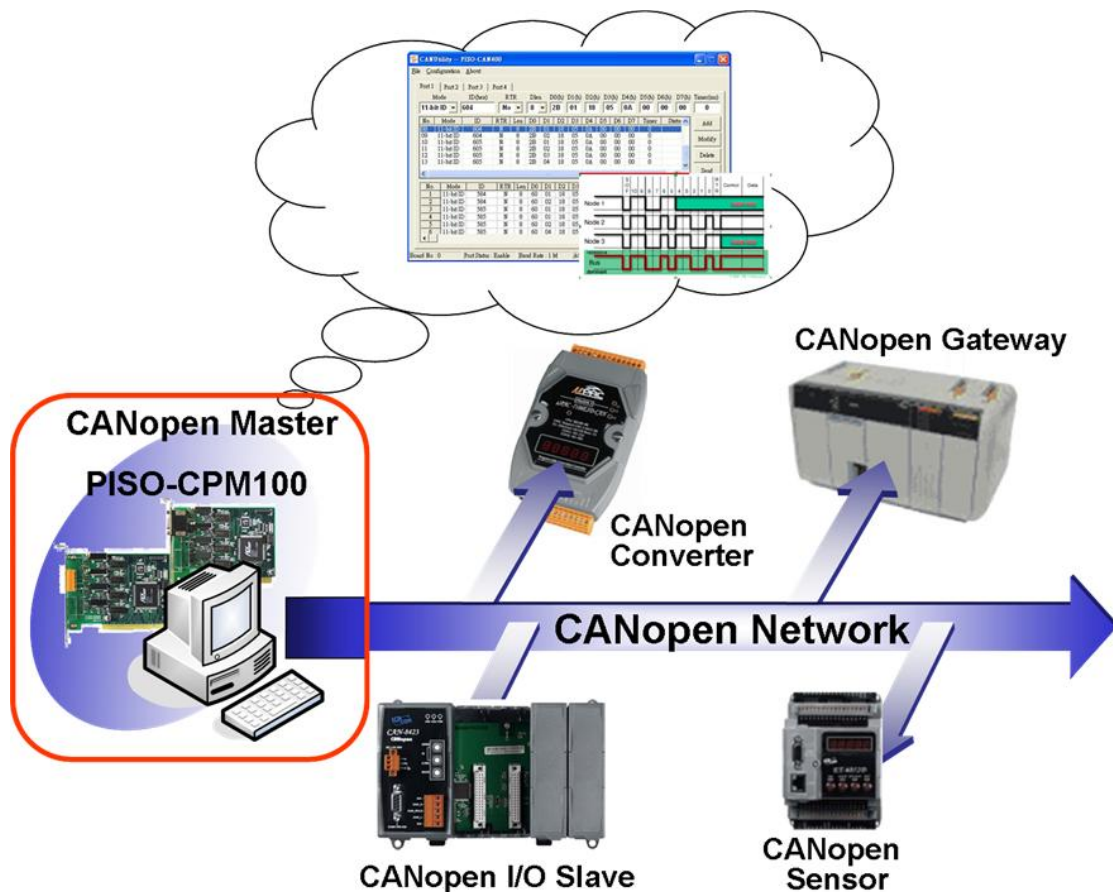


Figure 1.2 Example of application architecture

PISO-CPM100(U) follows the CiA CANopen specification DS-301 V4.01, and supports the several CANopen features. The CANopen communication general concept is shown as Figure 1.3.

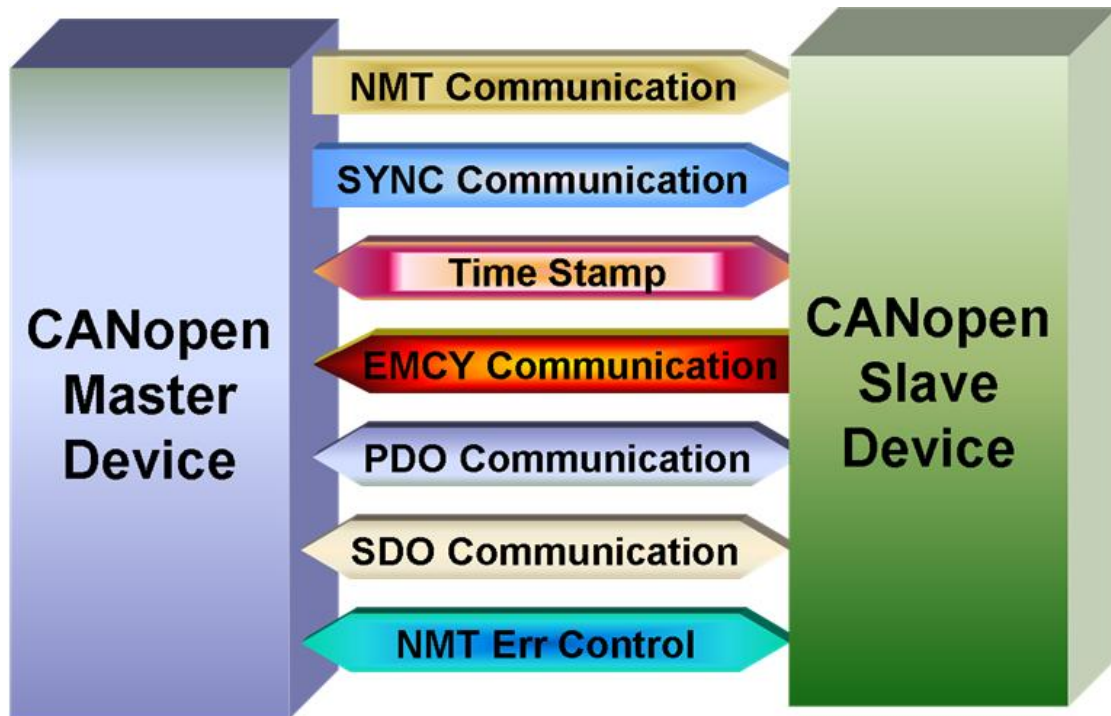


Figure 1.3 CANopen communication general concept

- **Node Manager (NMT Master)**
 - Functions for changing the slave device state
 - Node Guarding Protocol for error control
 - Support Emergency (EMCY) messages
- **SDO Manager**
 - Expedited, segmented and block methods for SDO download and upload
- **PDO Manager**
 - Support all transmission types and event timer
- **SYNC Manager**
 - SYNC message production
 - SYNC cycles of 0.1ms resolution
- **EMCY Manager**
 - EMCY message consumer

For more information about the CANopen functions described above, please refer to the function descriptions and demo programs shown in the chapter 3 and chapter 4.



Specifications

- PISO-CPM100-D/T:
 - 33 MHz 32bit 5 V PCI bus (V2.1) plug and play technology.
- PISO-CPM100U-D/T:
 - Universal PCI card supports both 5 V and 3.3 V PCI bus.
- CPU: 80186 compactable CPU, 80 MHz.
- CAN controller: NXP SJA1000T with 16 MHz.
- CAN transceiver: NXP 82C250.
- CAN bus interface: Follow ISO 11898-2 specification.
- Connector: 5-pin screw terminal or 9-pin D-sub female connector.
- 512 kbytes Flash memory, 512 kbytes SRAM, and 8 kbytes DPRAM.
- 2 kbytes EEPROM and 31 bytes NVRAM.
- Isolation voltage: 2500 Vrms photo-isolation protection on CAN side.
- Power requirements: 5 V@ 400 mA.
- Operating Temperature: 0 ~ +60 °C.
- Storage Temperature: -20 ~ +80 °C.
- Humidity: 0 ~ 90% non-condensing.
- Dimensions: please refer to section 2.1.

Features

- One CAN communication port.
- Follow CiA DS-301 V4.01
- 240 records CANopen PDO message receive buffer size
- 20 records CANopen EMCY message receive buffer size
- Support 8 kinds baud: 10 kbps, 20 kbps, 50 kbps, 125 kbps, 250 kbps, 500 kbps, 800 kbps, and 1 Mbps
- Each Port support maximum nodes up to 127
- Support upload and download SDO Segment
- Support Node Guarding protocol
- Provide 5 sets of SYNC cyclic transmission
- Support EMCY protocol
- Timestamp of CAN message with at least ± 1 ms precision
- Jumper select 120 Ω terminator resistor for CANopen network
- Support firmware update
- Two indication LEDs (Tx/Rx and Err LEDs)
- Provide VC++, VB, and BCB demos and function libraries

2. Hardware Configuration

This section would describe the hardware setting of the PISO-CPM100(U). This information includes the wire connection and terminal resistance configuration for the CAN network.

2.1. Board Layout

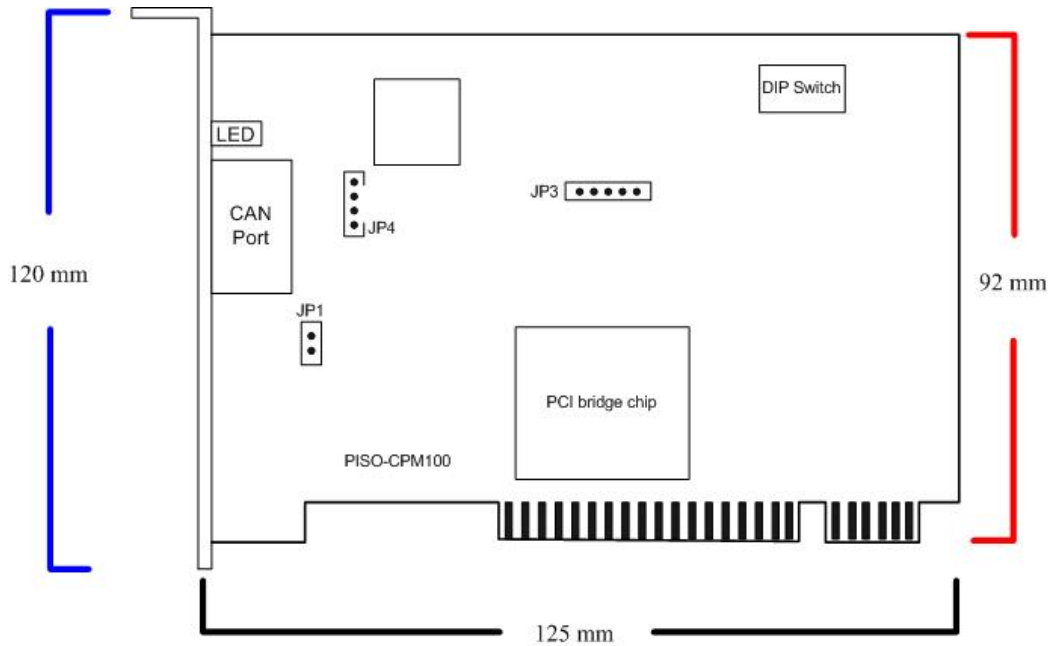


Figure 2.1 PISO-CPM100 board layout

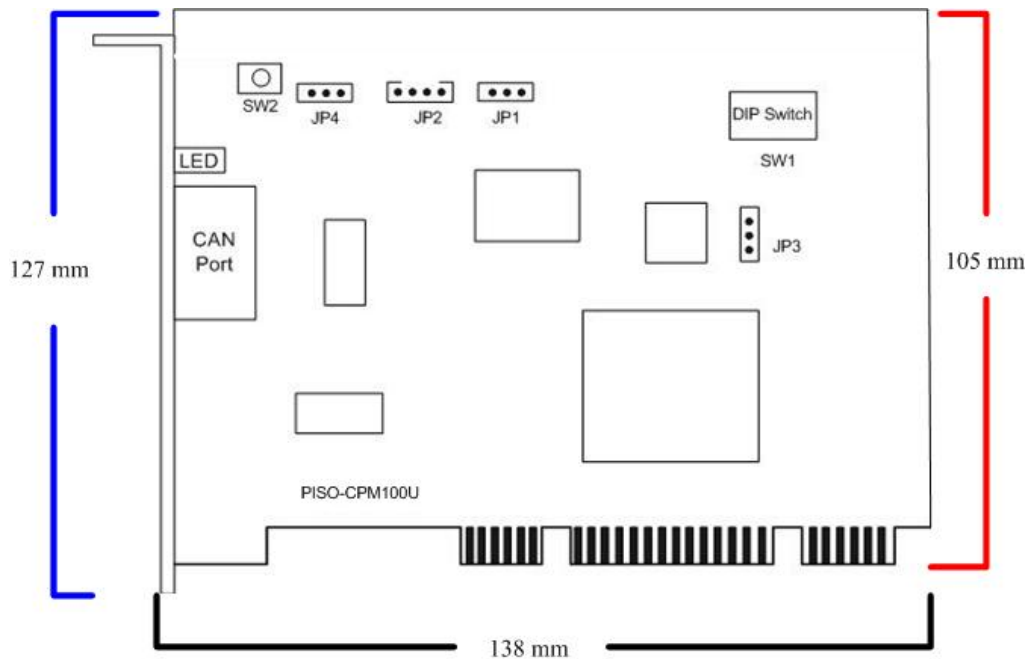


Figure 2.2 PISO-CPM100U board layout

Jumper Selection

The following table shows the definition of jumpers and DIP switch. Users need to refer to this table to configure the PISO-CPM100(U) hardware.





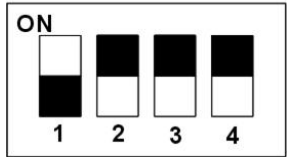
Jumper	Description	Status
JP1 (PISO-CPM100)	CAN Port 120 Ω terminal resistance	  Enable Disable
JP4 (PISO-CPM100U)	CAN Port 120 Ω terminal resistance	  Enable Disable
DIP switch	<p>DIP switch is used to set the PISO-CPM100(U) board No. Switch1 is for bit0, switch2 is for bit1 and so forth. For example, if the left-hand-side switch (switch1) is ON, the board No. is set to 1. The range of board No. is from 0 to 15. Be careful that the board No. for each board must be unique in the PC.</p>	<p style="text-align: center;">DIP Switch</p>  <p>The situation indicates the board No. 1.</p>

Table 2.1 Jumper or DIP switch selections

Connector Pin Assignment

The PISO-CPM100(U) has two kinds of connector. One is 5-pin screw terminal connector (PISO-CPM100(U)-T) and the other is 9-pin D-sub female connector (PISO-CPM100(U)-D) for wire connection of the CANopen network. The connector's pin assignment is specified as follows:

2.1.1. 5-pin screw terminal connector

The 5-pin screw terminal connector of the CAN bus interface is shown in figure 2.2. The details for the pin assignment are presented in the following table.

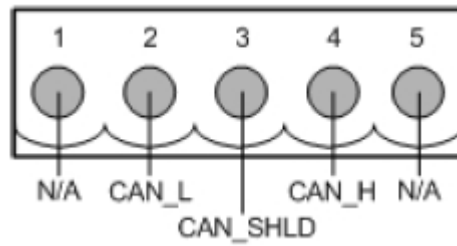


Figure 2.2 5-pin screw terminal connector

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

Table 2.2 Pin assignment of 5-pin screw terminal connector

2.1.2. 9-pin D-sub female connector

The 9-pin D-sub female connector of the CAN bus interface is shown in figure 2.3 and the corresponding pin assignments are given in following table.

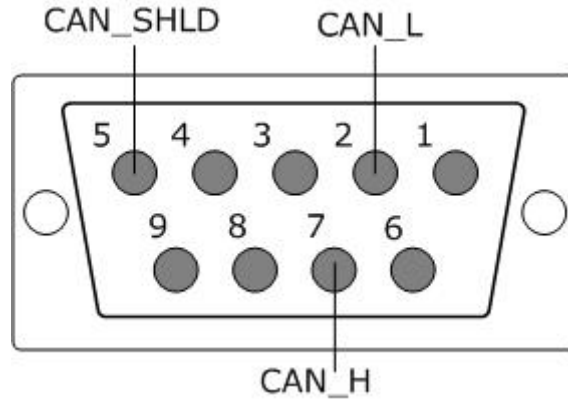


Figure 2.3 9-pin D-sub female connector

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

Table 2.3 Pin assignment of the 9-pin D-sub female connector

2.1.3. 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 $108\Omega\sim 132\Omega$). The length related resistance should have $70m\Omega/m$. Users should check the resistances of the CAN bus, before they install a new CAN network.

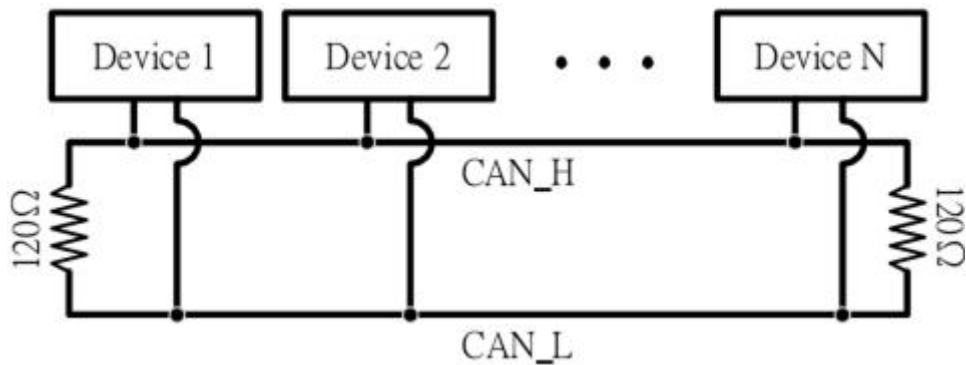


Figure 2.4 CANopen network topology

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 ($m\Omega/m$)	Cross Section (Type)	
0~40	70	0.25(23AWG)~0.34 mm^2 (22AWG)	124 (0.1%)
40~300	<60	0.34(22AWG)~0.6 mm^2 (20AWG)	127 (0.1%)
300~600	<40	0.5~0.6 mm^2 (20AWG)	150~300
600~1K	<20	0.75~0.8 mm^2 (18AWG)	150~300

Table 2.4 Relationship between cable feature and terminal resistance



2.2. Green LED

After PISO-CPM100(U) has been activated, the green LED would be flashed once when PISO-CPM100 receives or transmits one message to CAN bus successfully. If the bus loading is heavy, the green LED would turn on always.

2.3. Red LED

When some error occurs, the red LED would turn on until the error has been solved. Users can use CPM100_GetCANStatus function to get the situation except buffer status.

2.4. Hardware Installation

When users want to use PISO-CPM100(U), the hardware installation needs to be finished as following steps.

1. Shutdown your personal computer.
2. Configure the DIP switch and JP1 of PISO-CPM100(U) for the board No. and the terminal resistance. The more detail information could be found on the section 2.1.
3. Find an empty PCI slot for the PISO-CPM100(U) on the mother board of the personal computer. Plug the configured PISO-CPM100(U) into this empty PCI slot.
4. Plug the CAN bus cable(s) into the 5-pin screw terminal connector or the 9-pin D-sub connector.

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

3. Software Installation

3.1. Software Structure

The CPM DLL driver is the CANopen specification function collections for the PISO-CPM100(U) cards used in Windows 98/Me/NT/2000/XP systems. The application structure is presented in the following figure. The users' CANopen master application programs can be developed by the following program development tools: Borland C++ Builder, VB, and visual C++. The driver architecture is shown in the following Figure.

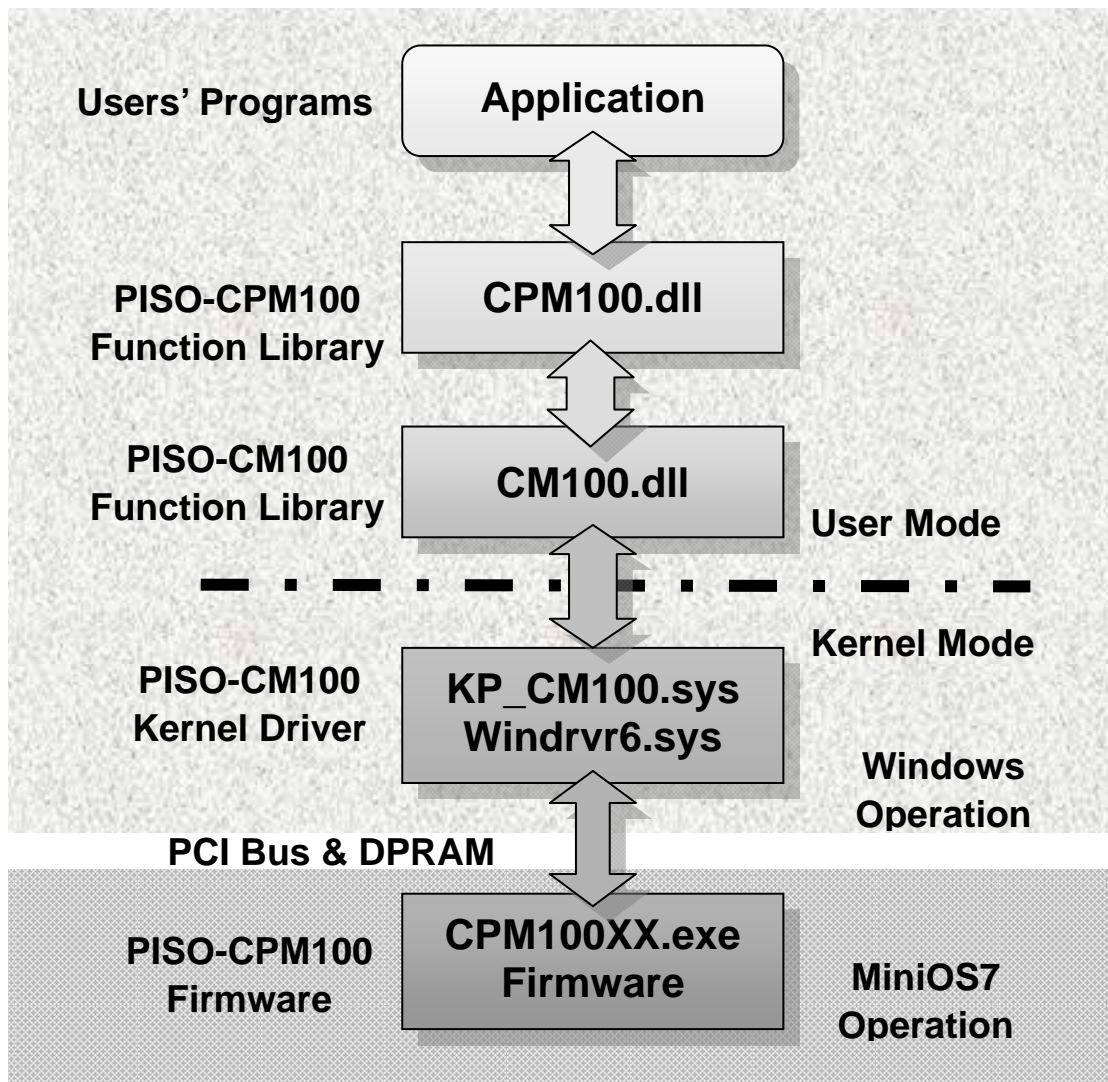


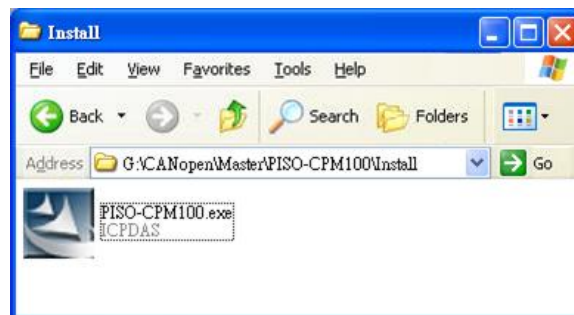
Figure 3.1 Driver concept of PISO-CPM100(U)

3.2. Installation Driver Step by Step

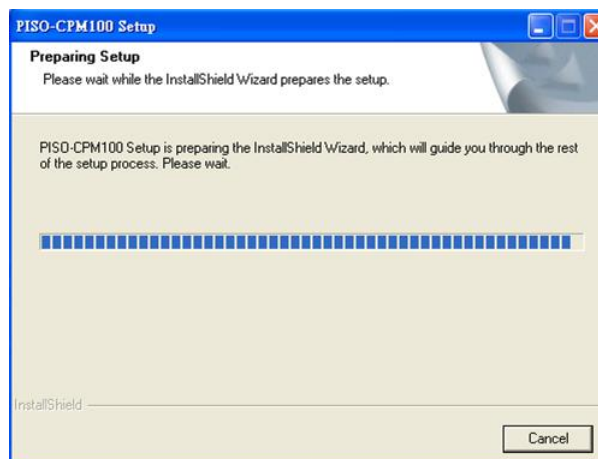
When users want to use the PISO-CPM100(U) CAN card, the PISO-CPM100(U) driver must be installed firstly. After finishing the installation process, the CPM_Utility and the demo programs would also be installed to the PC. The demo programs may be a good reference for users to build their CANopen master interface by using VC++, BCB and VB. The demo programs also give a simple interface to show the basic functions of master/slave connection and CANopen master program architectures. It is very helpful for users to understand how to use these functions and develop their CANopen master application. If users do not want to develop this application by themselves, the CPM_Utility can be used to be an easily CANopen master program. The following description displays the step-by-step procedures about how to install the PISO-CPM100(U) driver.

Install the PISO-CPM100(U) CAN card driver

Step 1: Insert the product CD into the CD-ROM and find the path \CANopen\Master\PISO-CPM100\Install\. Then execute the PISO-CPM100.exe to install the PISO-CPM100(U) CAN card driver.

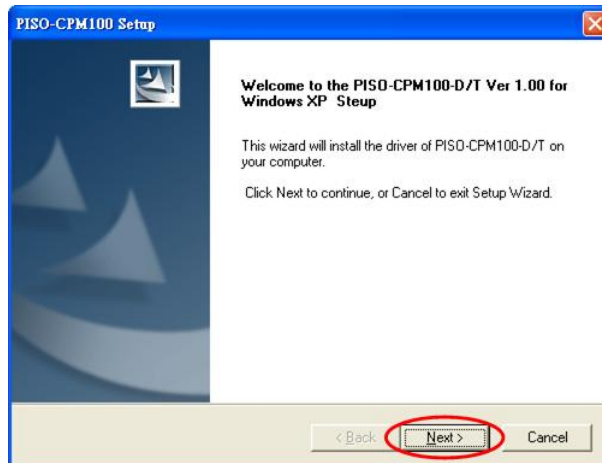


Step 2: Wait until the install wizard has prepared.

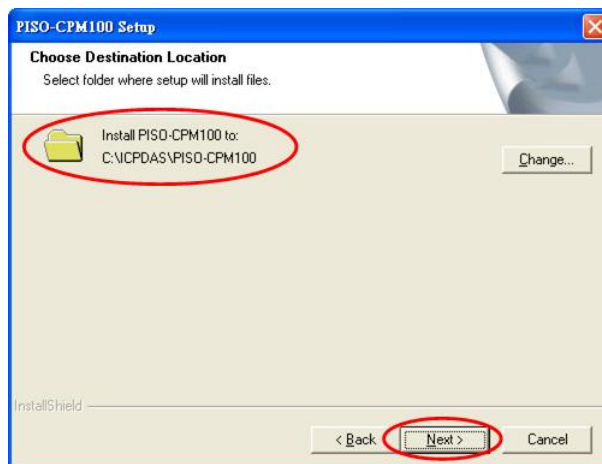




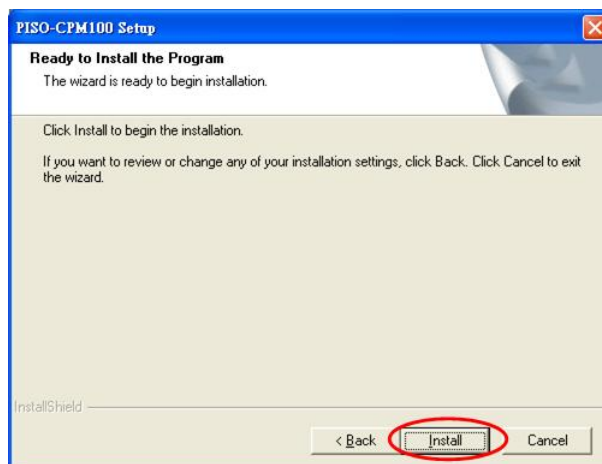
Step 3: Click “Next” to start the PISO-CPM100(U) installation.



Step 4: Select the folder where the PISO-CPM100(U) setup would be installed and click “Next” button to continue.

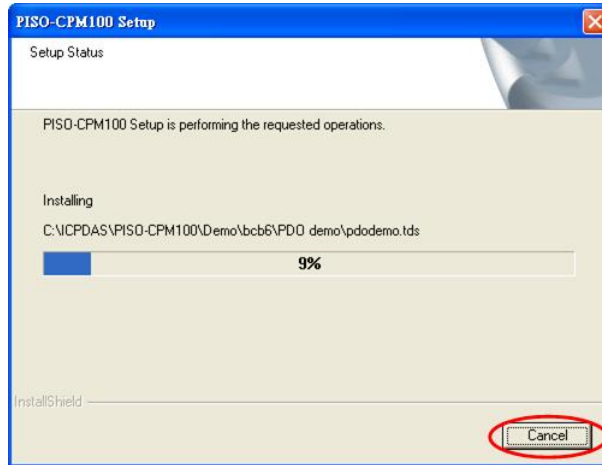


Step 5: Click the button “Install” to continue.





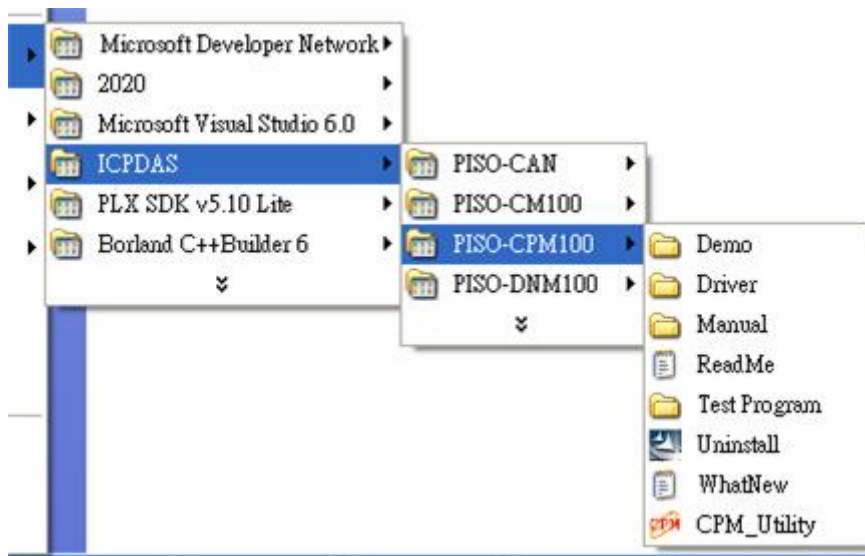
Step 6: Wait for finishing the PISO-CPM100(U) installation.



Step 7: Finally, restart the computer to complete the installation.



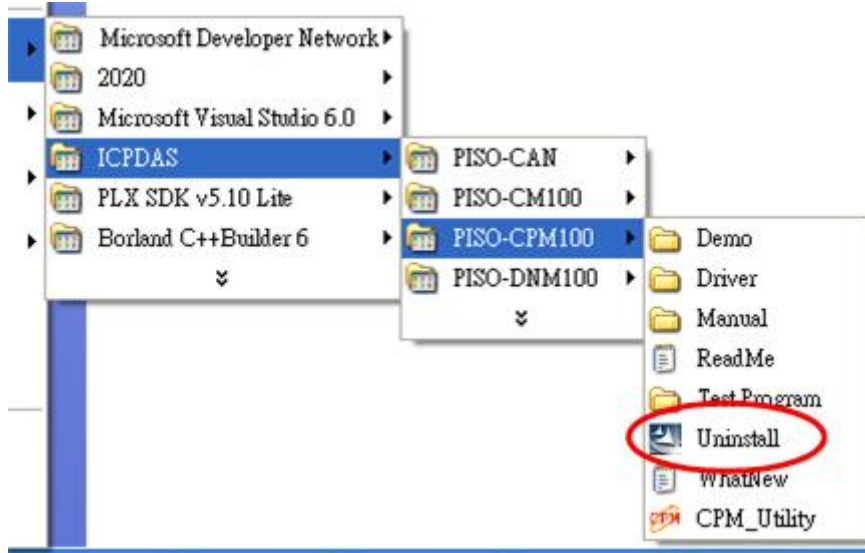
Step 8: When finishing the installation. The PISO-CPM100(U) folder would be found at the Start menu shown as below.





Remove the PISO-CPM100(U) driver

If the PISO-CPM100(U) driver is not used any more, users can click the “Uninstall” to remove the PISO-CPM100(U) driver below.





4. PISO-CPM100(U) Function Library

4.1. Function List

In order to use the PISO-CPM100(U) more easily, we provide some useful and easy-to-use functions in CPM100 library. There are three function libraries for different compiler, such as VC, VB and BCB. Users can use these functions to control the PISO-CPM100(U) by using the functions. The following table shows all functions provided by the CPM100 library.

Function Name	Description
CPM100_GetVersion	Get the version of the CPM100 library
CPM100_TotalBoard	Get the total number of PISO-CPM100(U)
CPM100_GetBoardSwitchNo	Get the board number from the dip switch number of the PISO-CPM100(U)
CPM100_GetBoardInf	Get board information of the PISO-CPM100(U)
CPM100_ActiveBoard	Activate the PISO-CPM100(U) CAN board
CPM100_BoardIsActive	Obtain the active status of the PISO-CPM100(U)
CPM100_CloseBoard	Close the PISO-CPM100(U) CAN board
CPM100_GetCANStatus	Obtain the status of the CAN controller
CPM100_InitMaster	Initialize CPM100 library
CPM100_ShutdownMaster	Remove all nodes and stop master
CPM100_AddNode	Add a node into CPM100 master manager
CPM100_RemoveNode	Remove a node from CPM100 master manager
CPM100_NMTChangeState	Change the specific CANopen node state
CPM100_NMTGetState	Get the specific CANopen node state
CPM100_NMTGuarding	Start the specific node guarding function
CPM100_SendSYNC	Send SYNC message from the specific channel
CPM100_GetSYNCingID	Get the total SYNC ID of sending
CPM100_ChangeSYNCID	Change SYNC COB-ID
CPM100_ChangeEMCYID	Change EMCY COB-ID
CPM100_ReadEMCYCount	Get the number of EMCY message of the specific channel
CPM100_ReadEMCY	Read EMCY message from the specific channel
CPM100_SDOAbortTransmit	Send SDO abort message from the specific channel
CPM100_SDOReadData	Read data by uploading SDO protocol
CPM100_SDOWriteData	Write data by downloading SDO protocol

CPM100_DynamicPDO	Map-a new PDO object dynamically
CPM100_InstallPDO	Enable a specific PDO object of the specific Nodes
CPM100_RemovePDO	Remove a specific PDO object of the specific Nodes
CPM100_PDOTxType	Set the transmission type of TxPDO of the specific Nodes
CPM100_SetEventTimer	Set the event timer of the specific TxPDO
CPM100_ChangePDOCobID	Change the PDO COB-ID of the specific Node
CPM100_PDOWrite	Use PDO to write data to the CANopen node via the COB-ID
CPM100_PDORemote	Use PDO to get data from CANopen node via the COB-ID
CPM100_ReadPDOCount	Get the number of not RTR PDO message
CPM100_ReadPDOMessage	Read the not RTR PDO message data of the specific COB-ID
CPM100_WriteDO	Output 8 bits DO value to the specific Node
CPM100_ReadDI	Read 8 bits DI value from the specific Node
CPM100_WriteAO	Output one AO channel to the specific Node
CPM100_ReadAI	Read one AI channel from the specific Node
CPM100_COBIDInfo	Get SYNC, EMCY, and all PDO COB-ID of the specific node
CPM100_PDOMappingInfo	Get mapping data information of the PDO via COB-ID connection

Table 4.1 Description of functions

4.2. Function Return Code

The following table interprets all the return code returned by the CANopen Master Library functions.

Return Code	Error ID	Description
0	CPM100_NoError	OK
1	CPM100_DriverError	Kernel driver is not opened
2	CPM100_ActiveBoardError	PISO-CPM100(U) can't be activated successfully
3	CPM100_BoardNumberError	The board number of PISO-CPM100(U) is error
102	CPM100_ConfigureErr	Library hasn't been configured successfully
103	CPM100_DataLenErr	Data length is erroneous
104	CPM100_NodeAddErr	Adding a CANopen node is a failure
106	CPM100_StatusErr	CANopen slave NMT status is error
107	CPM100_SetGuardErr	Setting the guarding parameters of CANopen slave node is a failure
109	CPM100_NodeNumberErr	Set the node number is error
110	CPM100_CobIdErr	Set COB-ID error
112	CPM100_SDOSEndErr	Send SDO expedition message error
115	CPM100_PDOSendErr	Send PDO message error
116	CPM100_PDOTypeErr	TxPDO or RxPDO type is error
118	CPM100_PDOEntryErr	Set subindex content of PDO error
120	CPM100_PDORemoveErr	Remove PDO content error
121	CPM100_TimeOut	Message response timeout
127	CPM100_ChannelErr	This I/O channel isn't exist
130	CPM100_SYNCSEndErr	Send SYNC message error
131	CPM100_SYNCSEtErr	Cyclic SYNC is over 5 or cyclic timer is less than 5
140	CPM100_SDODataLose	SDO buffer is overflow
147	CPM100_PDOFIFOisEmpty	The PDO buffer doesn't have any message
148	CPM100_EMCFIFOisEmpty	The EMCY buffer doesn't have any message
150	CPM100_SendCmdErr	The command send to PISO-CPM100(U) is error
160	CPM100_FirmwareErr	The firmware of PISO-CPM100(U) isn't running
162	CPM100_MasterInitErr	Master initializes error
163	CPM100_MasterShutdownErr	Master shut down error
165	CPM100_CobIdChangeErr	Can't change the COB ID
167	CPM100_SetEventTimerErr	Set event timer of TxPDO error

Table 4.2 Description of return code



4.3. CANopen Master Library Application Flowchart

In this section, it describes the operation procedure about how to use the CANopen Master Library to build users applications. This information is helpful for users to apply the CANopen Master Library easily. Besides, the CANopen operation principles must be obeyed when build a CANopen master application. For example, if the CANopen node is in the pre-operational status, the PDO communication object is not allowed to use. For more detail information, please refer to the demo programs in section 5.

When users' programs apply the CANopen Master Library functions, the functions CPM100_ActiveBoard and CPM100_InitMaster must be call first. The functions are used to activate PISO-CPM100(U), configure the CAN port, and initialize CPM100 library.

After activate the CAN interface card and initialize library successfully, users need to use the CPM100_AddNode function to install at least one CANopen device into the node list.

If the functions CPM100_ActiveBoard, CPM100_InitMaster and CPM100_AddNode have been executed, the communication services (NMT, SYNC, EMCY, SDO, and PDO services) can be used at any time before calling the functions CPM100_ShutdownMaster and CPM100_CloseBoard. That is because the CPM100_ShutdownMaster would stop all process created by the function CPM100_InitMaster, and the CPM100_CloseBoard would stop the hardware of PISO-CPM100(U).

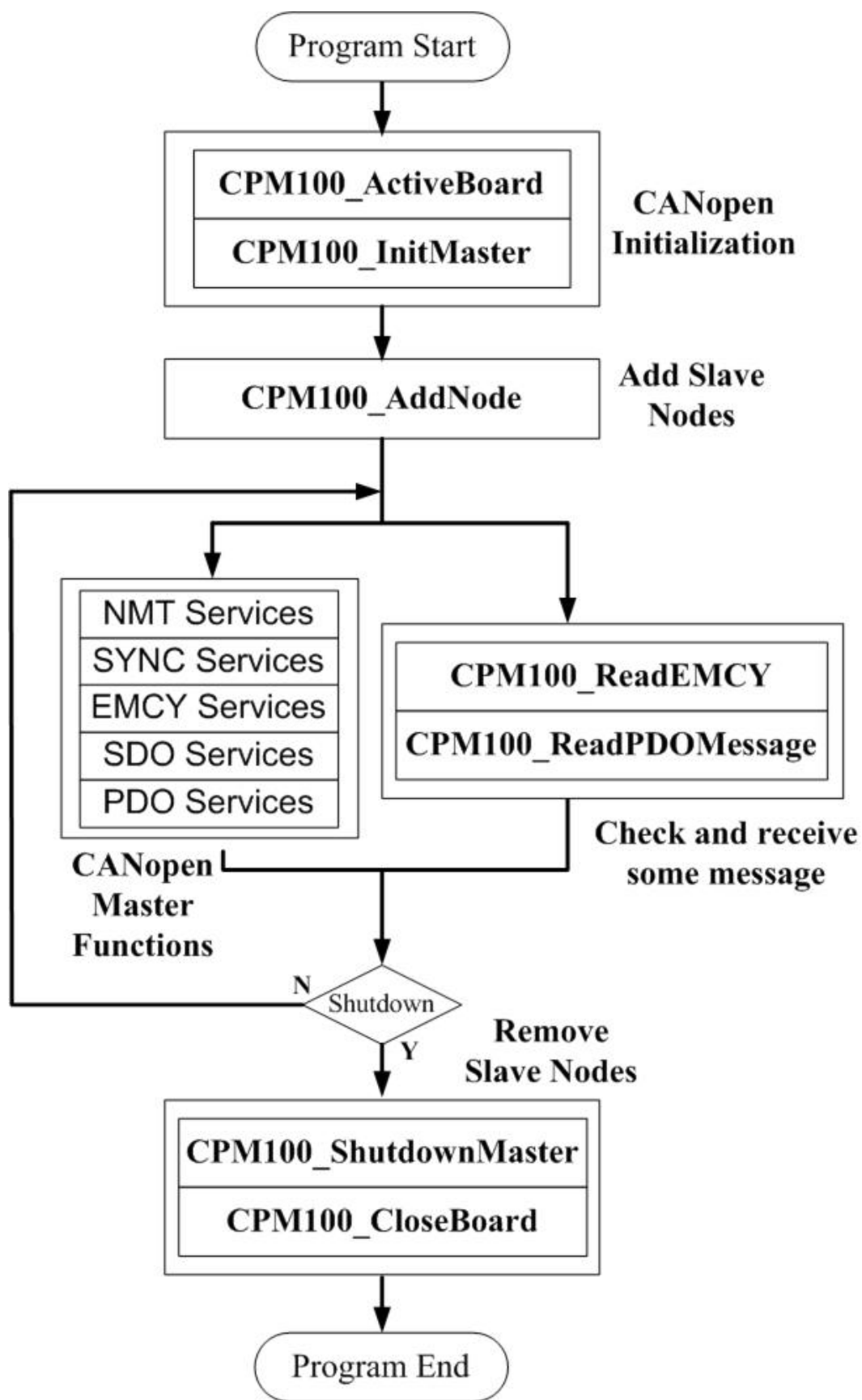


Figure 4.1 Main programming sequences

4.4. Communication Services Introduction

NMT Services

The CANopen Master Library provides several NMT services functions, such as the CPM100_AddNode, CPM100_RemoveNode, CPM100_NMTChangeState, CPM100_NMTGetState, and CPM100_NMTGuarding functions. As the prerequisite for the master, the slave nodes have to be registered by using CPM100_AddNode function with the Node-ID. The registered slave nodes can be individually removed from the node list by using the CPM100_RemoveNode function. Through NMT services, the NMT Master controls the state of the slave. Table 4.3 lists the command value and corresponding NMT command for the input parameters of the CPM100_NMTChangeState function. When using the CPM100_NMTGetState function, the slave status value and their descriptions are shown in the table 4.4. The Node Guarding protocol is implemented via the CPM100_NMTGuarding function. If the slave nodes are in the node list, users can change the node guarding parameters defined in the slave nodes by calling the CPM100_NMTGuarding function.

Command Value	Description
1 (0x01)	Enter Operational
2 (0x02)	Stop
128 (0x80)	Enter Pre-Operational
129 (0x81)	Reset_Node
130 (0x82)	Reset_Communication

Table 4.3 NMT Command Specifier

State of Slave	Description
4 (0x04)	STOPPED
5 (0x05)	OPERATIONAL
127 (0x7F)	PRE-OPERATIONAL

Table 4.4 State of the Slave



SDO Services

Initiating SDO download or SDO upload protocol is used when SDO data length ≤ 4 bytes. If the SDO message data length > 4 bytes, segment SDO download or upload protocol would be used. To call these two CPM100_SDOReadData and CPM100_SDOWriteData functions, the initial protocol and segment protocol would be selected automatically according to data length.

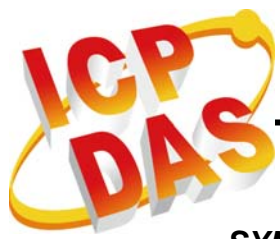
CPM100_SDOAbortTransmit function can abort a pending SDO transfer at any time. Applying the aborting transmitting service, that would not have any confirmation from the slave device.

PDO Services

The CPM100_DynamicPDO function is used for setting TPDOs or RPDOs mapping object. Each PDO object supports 0~8 application objects. These application objects defined in the CANopen specification DS401 are mapped to the DI/DO/AI/AO channels. After calling the function CPM100_DynamicPDO, the PDO communication object will be mapped and activated. If the PDO communication object is not needed no more, use the CPM100_RemovePDO function to remove it.

The PDOs data are written to the PDO buffer by using the CPM100_WritePDO function. This function can write all 8-byte PDO data or write some part of 8-byte PDO data to PDO buffer. If users write some part of the PDO data to the buffer, the other part of the PDO data will not be changed. Users can use the CPM100_SetEventTimer and CPM100_PDOTxType functions to change the response type of TPDO. When devices reply PDO data with data event or timer event, users can use the CPM100_ReadPDOMessage function to read these data stored in the PDO buffer. It also can change the output type of RPDO with the function CPM100_PDOTxType.

In CANopen specification, users can get the TxPDOs data by applying the remote transmit request CAN frame. The CPM100_PDORemote function is needed in this case.



SYNC Services

Calling the CPM100_SendSYNC function can start the SYNC object transmission. This function supports to send single SYNC message and cyclic SYNC message. The SyncCycle parameter of the CPM100_SendSYNC function can adjust the cyclic period of SYNC COB-ID sent by master if the SyncCycle parameter is not 0. This unit of SyncCycle parameter range is 0.1ms. When the SyncCycle parameter is set to 0, the SYNC object transmission will send single SYNC message, or the function will be stopped if the SYNC COB-ID is running. And the maxima number of SYNC cyclic message is 5.

EMCY Services

Emergency objects are triggered by the occurrence of a device internal error situation. Users can call the function CPM100_ReadEMCY to receive EMCY message if any CAN slaves send EMCY messages.

For example, if users have used CPM100_NMTGuarding to guard some slave and the guarding failure event of node 1 of the slave has occurred. The PISO-CPM100(U) and the node 1 of the slave will both produce EMCY messages as below, and users can use CPM100_ReadEMCY function to read the EMCY message sent by node 1.

Master	ID	Len	D0	D1	D2	D3	D4	D5	D6	D7
	0x81	8	0x30	0x81	0x11	0x07	0x23	0x00	0x00	0x00
Slave	ID	Len	D0	D1	D2	D3	D4	D5	D6	D7
	0x81	8	0x30	0x81	0x11	0x07	0x00	0x00	0x00	0x00

If users have only received the EMCY produced by Master, it means the connection between the Master and the Slave may be opened. If users have received both the two EMCY, it means the guard time may be too short, or the bus loading may be too heavy.



4.5. Function Description

4.5.1. CPM100_GetVersion

- **Description:**

This function is used to obtain the version information of CPM100.lib library.

- **Syntax:**

`float CPM100_GetVersion(void)`

- **Parameter:**

None

- **Return:**

LIB library version information.



4.5.2. CPM100_TotalBoard

- **Description:**

Obtain the total board number of PISO-CPM100(U) plugged in the PCI bus.

- **Syntax:**

```
int CPM100_TotalBoard(void)
```

- **Parameter:**

None

- **Return:**

Return the scanned total PISO-CPM100(U) number.



4.5.3. CPM100_GetBoardSwitchNo

- **Description:**

Obtain the DIP switch No. of PISO-CPM100(U).

- **Syntax:**

```
int CPM100_GetBoardSwitchNo(BYTE BoardCntNo,  
                             BYTE *BoardSwitchNo)
```

- **Parameter:**

BoardCntNo: [input] The number of specified PISO-CPM100(U). For example, if the first PISO-CPM100(U) is applied, this value is 0. If the second board is applied, this value is 1.

***BoardSwitchNo:** [output] The address of a variable used to get the DIP switch No. of PISO-CPM100(U).

- **Return:**

CPM100_NoError

CPM100_DriverError

CPM100_BoardNumberError



4.5.4. CPM100_GetBoardInf

- **Description:**

Obtain the information of PISO-CPM100(U), which includes vender ID, device ID and the interrupt number.

- **Syntax:**

```
int CPM100_GetBoardInf(BYTE bBoardNo, DWORD *dwVID,  
                        DWORD *dwDID, DWORD *dwSVID,  
                        DWORD *dwSDID, DWORD *dwSAuxID,  
                        DWORD *dwIrqNo)
```

- **Parameter:**

bBoardNo: [input] Switch No. of PISO-CPM100(U) DIP. The value is from 0 to 15.

***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 variable applied to receive sub-vendor ID.

***dwSDID:** [output] The address of 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:**

CPM100_NoError

CPM100_DriverError

CPM100_BoardNumberError



4.5.5. CPM100_ActiveBoard

- **Description:**

Activate the PISO-CPM100(U) board. It must be called once before using the other functions of PISO-CPM100(U) APIs.

- **Syntax:**

```
int CPM100_ActiveBoard(BYTE bBoardNo)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

- **Return:**

CPM100_NoError
CPM100_DriverError
CPM100_BoardNumberError
CPM100_ActiveBoardError
CPM100_FirmwareErr
CPM100_TimeOut



4.5.6. CPM100_BoardIsActive

- **Description:**

Obtain the active status of the specific board.

- **Syntax:**

int CPM100_BoardIsActive(**BYTE** bBoardNo)

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

- **Return:**

0: means the board is inactive.

1: means the board is active.



4.5.7. CPM100_CloseBoard

- **Description:**

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

- **Syntax:**

`int CPM100_CloseBoard(BYTE bBoardNo)`

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

- **Return:**

CPM100_NoError
CPM100_DriverError
CPM100_BoardNumberError



4.5.8. CPM100_GetCANStatus

- **Description:**

Obtain the status of the CAN controller of the specific PISO-CPM100(U) board.

- **Syntax:**

```
int CPM100_GetCANStatus(BYTE bBoardNo, BYTE *bStatus)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

***bStatus:** [output] The address of a variable is applied to get the status value of CAN controller. The bit interpretation of the bStatus parameter is as below.

Bit	Name	Value	Status
Bit 7	Bus Status	1	Bus-off
		0	Bus-on
Bit 6	Error Status	1	Error
		0	OK
Bit 5	Transmit Status	1	Transmit
		0	Idle
Bit 4	Receive Status	1	Receive
		0	Idle
Bit 3	Transmission Complete Status	1	Complete
		0	Incomplete
Bit 2	Transmit Buffer Status	1	Release
		0	Locked
Bit 1	Data Overrun Status	1	Overrun
		0	Absent
Bit 0	Receive Buffer Status	1	Not Empty
		0	Empty

- **Return:**

CPM100_NoError



CPM100_DriverError
CPM100_BoardNumberError
CPM100_ActiveBoardError



4.5.9. CPM100_InitMaster

- **Description:**

The function must be applied when configuring the CAN controller and initialize the CPM100 library. It must be called once before using other functions of CPM100.lib.

- **Syntax:**

`int CPM100_InitMaster(BYTE bBoardNo, BYTE bBaudrate)`

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bBaudrate: [input] The baudrate of the PISO-CPM100(U)

Value	Baud rate
0	10Kbps
1	20Kbps
2	50Kbps
3	125Kbps
4	250Kbps
5	500Kbps
6	800Kbps
7	1Mbps

- **Return:**

CPM100_NoError
CPM100_ActiveBoardError
CPM100_ConfigureErr
CPM100_SendCmdErr
CPM100_MasterInitErr
CPM100_TimeOut



4.5.10. CPM100_ShutdownMaster

- **Description:**

The CPM100_ShutdownMaster function removes all the slaves added to master, and stop all the functions of CPM100. The function must be called before exit the users' application programs.

- **Syntax:**

int CPM100_ ShutdownMaster (**BYTE** bBoardNo)

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15)

- **Return:**

CPM100_OK
CPM100_MasterShutdownerr
CPM100_TimeOut



4.5.11. CPM100_AddNode

- **Description:**

The CPM100_AddNode function can add a CANopen slave with the specified Node ID into the node list of CPM100. After calling this function to add a slave by the program, the slave would into the operational state directly and the default TxPDO COB ID(0x180 + node ID, 0x280 + node ID, 0x380 + node ID, 0x480 + node ID)and RxPDO COB ID(0x200 + node ID, 0x300 + node ID, 0x400 + node ID, 0x500 + node ID) would also be installed if the slave supports these default PDO COB ID. The added node can be removed from the node list by the CPM100_RemoveNode function.

- **Syntax:**

`int CPM100_AddNode(BYTE bBoardNo, BYTE bNodeID)`

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

- **Return:**

CPM100_NoError
CPM100_ActiveBoardError
CPM100_NodeAddErr
CPM100_NodeNumberErr
CPM100_SendCmdErr
CPM100_MasterInitErr
CPM100_TimeOut



4.5.12. CPM100_RemoveNode

- **Description:**

The CPM100_RemoveNode function removes the slave with the specified Node-ID from the node list of the node manager. It requires a valid Node-ID, which has installed by the function CPM100_AddNode before.

- **Syntax:**

`int CPM100_RemoveNode(BYTE bBoardNo, BYTE bNodeID)`

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

- **Return:**

CPM100_NoError

CPM100_NodeNumberErr

CPM100_TimeOut



4.5.13. CPM100_NMTChangeState

- **Description:**

The function CPM100_NMTChangeState is used to change the NMT state of a slave. If the node parameter of this function is set to 0, the state of all nodes on the same CAN network will be changed.

- **Syntax:**

```
int CPM100_NMTChangeState(BYTE bBoardNo, BYTE bNodeID,  
                           BYTE bState)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

bState: [input] NMT command specifier.

1: enter OPERATIONAL

2: stop

128: enter PRE-OPERATIONAL

129: Reset_Node

130: Reset_Communication

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_NodeNumberErr

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut



4.5.14. CPM100_NMTGetState

- **Description:**

The function CPM100_GetState can get the NMT state from the specific slaves.

- **Syntax:**

```
int CPM100_NMTGetState(BYTE bBoardNo, BYTE bNodeID,  
                       BYTE *bState)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

***bState:** [output] The NMT state of the slave.

4: STOPPED

5: OPERATIONAL

127: PRE-OPERATIONAL

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_NodeNumberErr

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut



4.5.15. CPM100_NMTGuarding

- **Description:**

Use the CPM100_Guarding function to set Guard Time and Life Time Factor of the specified slave with the specific node ID. Then, CPM100 would automatically send the Guarding message to the slave device according to the wGuardTime parameters. If the CPM100 doesn't receive the confirmation of Guarding message from the slave, PISO_CPM100 would send EMCY message to the bus. Users need to use CPM100_ReadEMCY to get the EMCY message. However, if the slave doesn't receive the Guarding message during the Node Life time period (Node Life time = wGuardTime * bLifeTimeFactor), it will be triggered to send the EMCY message of. It is recommended that bLifeTimeFactor value is set to more than 1.

- **Syntax:**

```
int CPM100_Guarding(BYTE bBoardNo, BYTE bNodeID,  
                   WORD wGuardTime,  
                   BYTE bLifeTimeFactor)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

wGuardTime: [input] Guard Time (1 ~ 65535).

bLifeTimeFactor: [input] Life Time Factor (1 ~ 255).

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_StatusErr

CPM100_SetGuardErr

CPM100_NodeNumberErr

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut



4.5.16. CPM100_SendSYNC

- **Description:**

Use the CPM100_SendSYNC function to send a SYNC message with the specified COB-ID cyclically. The maxima 5 cyclic SYNC message can be added.

- **Syntax:**

```
int CPM100_SendSYNC(BYTE bBoardNo, WORD wCobid,  
                   DWORD dwSyncCycle)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

wCobid: [input] COB-ID used by the SYNC object.

dwSyncCycle: [input] The time period of cyclic SYNC transmission.

This parameter is formatted by 0.1ms and the minimum time period is 0.5ms. If the parameter is 0, the running SYNC message will be stopped or the sanded SYNC message will be send once.

- **Return:**

CPM100_NoError
CPM100_ActiveBoardError
CPM100_NodeNumberErr
CPM100_CobldErr
CPM100_SYNCSEndErr
CPM100_SYNCSEtErr
CPM100_SendCmdErr
CPM100_MasterInitErr
CPM100_TimeOut



4.5.17. CPM100_GetSYNCingID

- **Description:**

Use this function to get the total SYNC IDs which have been sending.

- **Syntax:**

```
int CPM100_GetSYNCingID(BYTE bBoardNo, BYTE *IdNum,  
                        WORD *wSYNCIdList,  
                        DWORD *dwSYNCCycleList)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

***IdNum:** [output] The total SYNC ID number.

***wSYNCIdList:** [output] This parameter is a maximum 5 array. Per array is save a SYNC ID that had been sending.

***dwSYNCCycleList:** [output] This parameter is a maximum 5 array. Per array is save a cyclic timer of SYNC ID.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut



4.5.18. CPM100_ChangeEMCYID

- **Description:**

The function is used to change the EMCY COB-ID of a slave device.

- **Syntax:**

int CPM100_ChangeEMCYID (**BYTE** bBoardNo, **BYTE** bNodeID,
WORD wCobid)

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

wCobid: [input] COB-ID used by the EMCY object.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_StatusErr

CPM100_NodeNumberErr

CPM100_SDOSEndErr

CPM100_SDODataLose

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_CobIdChangeErr

CPM100_TimeOut



4.5.19. CPM100_ChangeSYNCID

- **Description:**

Users can apply the function to change the SYNC COB-ID of a slave device.

- **Syntax:**

```
int CPM100_ChangeSYNCID (BYTE bBoardNo, BYTE bNodeID,  
                          WORD wCobid)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

wCobid: [input] COB-ID used by the SYNC object.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_StatusErr

CPM100_NodeNumberErr

CPM100_SDOSEndErr

CPM100_SDODataLose

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_CobIdChangeErr

CPM100_TimeOut



4.5.20. CPM100_ReadEMCYCount

- **Description:**

Obtain the message count of EMCY message in EMCY buffer of PISO-CPM100(U).

- **Syntax:**

`int CPM100_ReadEMCYCount (BYTE bBoardNo)`

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

- **Return:**

EMCY message count



4.5.21. CPM100_ReadEMCY

- **Description:**

This function is used to read the EMCY message from EMCY buffer.

- **Syntax:**

```
int CPM100_ReadEMCY (BYTE bBoardNo, WORD *wCobid,  
                    BYTE *EMCY_Data, WORD *wYear,  
                    WORD *wMonth, WORD *wDayOfWeek,  
                    WORD *wDay, WORD *wHour,  
                    WORD *wMinute, WORD *wSecond,  
                    WORD *wMilliseconds)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

wCobid: [output] COB-ID used by the EMCY object.

***EMCY_Data:** [output] EMCY data.

***wYear:** [output] Specifies the current year.

***wMonth:** [output] Specifies the current month.

January = 1, February = 2, and so on.

***wDayOfWeek:** [output] Specifies the current day of the week.

Sunday = 0, Monday = 1, and so on.

***wDay:** [output] Specifies the current day of the month.

***wHour:** [output] Specifies the current hour.

***wMinute:** [output] Specifies the current minute.

***wSecond:** [output] Specifies the current second.

***wMilliseconds:** [output] Specifies the current millisecond.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_MasterInitErr

CPM100_EMCFIFOisEmpty

4.5.22. CPM100_SDOAbortTransmit

- **Description:**

This function is used to cancel the SDO transmission to the specific node. The node parameter is used to specify which SDO communication for terminating communication between the master and the specified slave device.

- **Syntax:**

```
int CPM100_SDOAbortTransmit (BYTE bBoardNo, BYTE bNodeID,  
                             WORD wIndex, BYTE bSubIndex)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

wIndex: [input] The object index value of the object dictionary.

bSubIndex: [input] The object subindex value of the object dictionary.

- **Return:**

CPM100_NoError
CPM100_ActiveBoardError
CPM100_StatusErr
CPM100_NodeNumberErr
CPM100_SDOSendErr
CPM100_SDODataLose
CPM100_SendCmdErr
CPM100_MasterInitErr
CPM100_TimeOut



4.5.23. CPM100_SDOReadData

- **Description:**

The function is useful to upload the SDO data from a specified slave. This function supports both expedition mode and segment mode.

- **Syntax:**

```
int CPM100_SDOReadData (BYTE bBoardNo, BYTE bNodeID,  
                        WORD wIndex, BYTE bSubIndex,  
                        WORD *pRDatalen, BYTE *pRData)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

wIndex: [input] The object index value of the object dictionary.

bSubIndex: [input] The object subindex value of the object dictionary.

***pRDatalen:** [output] Total data length.

***pRData:** [output] SDO data respond from the specified slave device.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_StatusErr

CPM100_NodeNumberErr

CPM100_SDOSEndErr

CPM100_SDODataLose

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut



4.5.24. CPM100_SDOWriteData

- **Description:**

The CPM100_SDOWriteData function can send out a SDO message to the specified slave device. This procedure is also called download SDO protocol. Because the data length of each object stored in object dictionary of the node is different, users need to know the data length when writing the data to the object of object dictionary of specified slave devices. This function supports both expedition mode and segment mode.

- **Syntax:**

```
int CPM100_SDOWriteData (BYTE bBoardNo, BYTE bNodeID,  
                          WORD wIndex, BYTE bSubIndex,  
                          WORD wWDataLen, BYTE *pWData,  
                          WORD *pRDataLen, BYTE *pRData)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

wIndex: [input] The object index value of the object dictionary.

bSubIndex: [input] The object subindex value of the object dictionary.

wWDataLen: [input] Total data size to be written.

* **pWData:** [input] The SDO data written to slave device.

***pRDataLen:** [output] Total data length.

***pRData:** [output] SDO data respond from the specified slave device.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_DataLenErr

CPM100_StatusErr

CPM100_NodeNumberErr

CPM100_SDOSEndErr

CPM100_SDODataLose



CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut

4.5.25. CPM100_DynamicPDO

- **Description:**

After calling the function, a new PDO COB ID would be added or an old PDO COB ID would be modified in the PDO object list. If the slave device has defined the default PDO object (defined by DS301), these default PDO would be installed automatically when function CPM100_AddNode is called.

- **Syntax:**

```
int CPM100_DynamicPDO(BYTE bBoardNo, BYTE bNodeID,
                     WORD wCobid, BYTE bRxTxType,
                     BYTE bPDOEntry, BYTE *pbMappingData)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

wCobid: [input] COB-ID used by the PDO object.

bRxTxType: [input] PDO type (0: RxPDO, 1: TxPDO).

bPDOEntry: [input] Entry number of the PDO (1~8).

***pbMappingData:** [input] 4-byte information of mapped application object. Users need to look up the user manual of CAN slave device to find the information of application object, and obey the following format to fill this parameter.

pbMappingData [0] : The numbers of bit of specified application object.

pbMappingData [1] : The subindex of specified application object.

pbMappingData [2] : The low byte of index of specified application object.

pbMappingData [3] : The high byte of index of specified application object.

For example, there is an application object built in the CAN slave device. This AI application object uses index 0x6401 and subindex 0x06. It is used to store a 16-bit data. When users add this specified application object in the 3rd entry of PDO object list of Cob-ID 0x183, the bPDOEntry is set to 3 (indicating



the PDO entry), pbMappingData[0] is set to 0x10 (for indicating the stored data is 16-bit), pbMappingData [1] is 0x06 (for indicating the subindex is 0x06), pbMappingData [2] is 0x01 (for indicating the low byte of the index 0x6401), and pbMappingData [3] is 0x64 (for indicating the high byte of the index 0x6401). And the mapping result is as below.

Byte	0	1	2	3	4	5	6	7
Original	DI 0	DI 1	X	X	X	X	X	X
After	DI 0	DI 1	AI_6_L	AI_6_H	X	X	X	X

After mapping the object successfully, the PDO object would have total 4 bytes data and 3 entries (per DI entry has one byte data and per AI entry has two bytes data). If users want to re-mapping a AI application object with index 0x6401 and subindex 0x02 in the 2nd entry of the PDO object. The bPDOEntry is set to 2, pbMappingData[0] is set to 0x10, pbMappingData [1] is 0x05, pbMappingData [2] is 0x01, and pbMappingData [3] is 0x64. The result is as below.

Byte	0	1	2	3	4	5	6	7
Original	DI 0	DI 1	AI_6_L	AI_6_H	X	X	X	X
After	DI 0	AI_2_L	AI_2_H	AI_6_L	AI_6_H	X	X	X

After mapping successfully, the DI 1 would be recovered by AI 2, and the PDO object will have total 5 bytes data and 3 entries.

- **Return:**

- CPM100_NoError
- CPM100_ActiveBoardError
- CPM100_DataLenErr
- CPM100_StatusErr
- CPM100_NodeNumberErr
- CPM100_CobIdErr
- CPM100_SDOSendErr
- CPM100_PDOTypeErr
- CPM100_ChannelErr
- CPM100_SDODataLose
- CPM100_SendCmdErr
- CPM100_MasterInitErr
- CPM100_TimeOut



4.5.26. CPM100_InstallPDO

- **Description:**

If there is a PDO object of a CANopen slave which has been mapping default, but the PDO object is not the default one in DS-301. Users can use the function to install the PDO object into the CANopen node list.

- **Syntax:**

```
int CPM100_InstallPDO(BYTE bBoardNo, BYTE bNodeID,  
                     WORD wCobid, WORD bPDOIndex)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

wCobid: [input] COB-ID used by the PDO object.

bPDOIndex: [input] The index of object dictionary of the PDO object.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_StatusErr

CPM100_NodeNumberErr

CPM100_CobIdErr

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut

4.5.27. CPM100_RemovePDO

- **Description:**

The function can remove a TxPDO or RxPDO which has been installed by the CPM100_DynamicPDO or CPM100_InstallPDO. This function also can remove single entry mapped in TxPDO or RxPDO.

- **Syntax:**

```
int CPM100_RemovePDO(BYTE bBoardNo, BYTE bNodeID,  
                    WORD wCobid, BYTE bRxTxType,  
                    BYTE bPDOEntry)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

wCobid: [input] COB-ID used by the PDO object.

bRxTxType: [input] PDO type (0: RxPDO, 1: TxPDO).

bPDOEntry: [input] PDO mapping object entry value (0 ~ 8). If

bPDOEntry parameter is 0, the specified PDO object will be removed. If others (1 ~ 8), the specified subindex content of the PDO will be removed.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_NodeNumberErr

CPM100_CobIdErr

CPM100_PDOPDOEntryErr

CPM100_PDORemoveErr

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut



4.5.28. CPM100_PDOTxType

- **Description:**

Call this function to set transmission type of the PDO to the specific node.

- **Syntax:**

```
int CPM100_PDOTxType(BYTE bBoardNo, BYTE bNodeID,  
                     WORD wCobid, BYTE bTxType)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

wCobid: [input] COB-ID used by the PDO object.

bTxType: [input] Transmission type of the TxPDO or RxPDO object.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_StatusErr

CPM100_NodeNumberErr

CPM100_CobIdErr

CPM100_SDOSEndErr

CPM100_SDODataLose

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut



4.5.29. CPM100_SetEventTimer

- **Description:**

Call this function to set event timer of the PDO of the specific node. It is important that the function is only useful for TxPDO.

- **Syntax:**

```
int CPM100_SetEventTimer(BYTE bBoardNo, BYTE bNodeID,  
                        WORD wCobid, WORD wEventTimer)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

wCobid: [input] COB-ID used by the PDO object.

wEventTimer: [input] Event timer of the TxPDO object. This parameter is formatted by 1ms. If the parameter is 0, the event timer of the PDO will be stopped.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_StatusErr

CPM100_NodeNumberErr

CPM100_CobIdErr

CPM100_SDOSEndErr

CPM100_SDODataLose

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut



4.5.30. CPM100_ChangePDOcobID

- **Description:**

If users want to change the PDO COB ID, this function would be useful. When using the function to change PDO COB ID, the mapping data of the PDO would not need to re-map.

- **Syntax:**

```
int CPM100_ChangePDOcobID(BYTE bBoardNo, BYTE bNodeID,  
                           WORD old_CobID, WORD new_CobID)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

old_Cobid: [input] Original COB-ID used by the PDO object.

new_Cobid: [input] The new COB-ID will be used by the PDO object.

- **Return:**

CPM100_NoError
CPM100_ActiveBoardError
CPM100_StatusErr
CPM100_NodeNumberErr
CPM100_CobIdErr
CPM100_SDOSEndErr
CPM100_SDODataLose
CPM100_SendCmdErr
CPM100_MasterInitErr
CPM100_TimeOut



4.5.31. CPM100_PDOWrite

- **Description:**

The function is used to send out a PDO message to the specified slave device. Before using this function, users need to use the CPM100_InstallPDO function to install the PDO object if users want to use non-default PDO. Then, change the NMT state of target slave device to operational mode by using the CPM100_NMTChangeState function if the slave is not in the operational mode. Use the parameter offset to set the start byte position of PDO data which need to be modified, and use the parameters * pTData and bTDataLen to point the data and data length which users want to fill to the PDO data.

- **Syntax:**

int CPM100_PDOWrite (**BYTE** bBoardNo, **WORD** wCobid,
BYTE bOffset, **BYTE** bTDataLen,
BYTE *pTData)

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

wCobid: [input] COB-ID used by the PDO object.

bOffset: [input] The start byte position of PDO data (0 ~ 7).

bTDataLen: [input] data size (bTDataLen + bOffset ≤ 8).

***pTData:** [output] The data pointer point to the PDO data.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_DataLenErr

CPM100_StatusErr

CPM100_CobIdErr

CPM100_PDOSendErr

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut

4.5.32. CPM100_PDORemote

- **Description:**

Use the function to send a RTR (remote-transmit-request) PDO message to the slave device.

- **Syntax:**

```
int CPM100_PDORemote (BYTE bBoardNo, WORD wCobid,
                     BYTE *pRDataLen, BYTE *pRData,
                     DWORD *UpperTime, DWORD *LowerTime)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

wCobid: [input] COB-ID used by the PDO object.

***pRDataLen:** [output] The data length of the RTR PDO message.

***pRData:** [output] The PDO message received from the slave device.

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

***LowerTime:** [output] The address of a variable used to obtain the lower double-word of time stamp of a CAN message. The time stamp is “(**UpperTime** << 32)+ **LowerTime**”. And the unit of the time stamp is 0.1ms.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_StatusErr

CPM100_CobIdErr

CPM100_PDOSendErr

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut



4.5.33. CPM100_ReadPDOCount

- **Description:**

Obtain the message count of non RTR PDO message in PDO buffer of the specific board.

- **Syntax:**

```
int CPM100_ReadPDOCount(BYTE bBoardNo)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

- **Return:**

PDO message count



4.5.34. CPM100_ReadPDOMessage

- **Description:**

Read the non RTR PDO message from the PDO buffer of the specific board.

- **Syntax:**

```
int CPM100_ReadPDOMessage(BYTE bBoardNo, WORD wCobid,  
                           BYTE *pRDatalen, BYTE *pRData,  
                           DWORD *UpperTime, DWORD *LowerTime)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

wCobid: [input] COB-ID used by the PDO object.

***pRDatalen:** [output] The data length of the RTR PDO message.

***pRData:** [output] The PDO message received from the slave device.

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

***LowerTime:** [output] The address of a variable used to obtain the lower double-word of time stamp of a CAN message. The time stamp is “(**UpperTime** << 32)+ **LowerTime**”. And the unit of the time stamp is 0.1ms.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_MasterInitErr

CPM100_PDOFIFOisEmpty



4.5.35. CPM100_WriteDO

- **Description:**

This function is used to output one byte (8 channels) DO data to the specific node.

- **Syntax:**

```
int CPM100_WriteDO(BYTE bBoardNo, BYTE bNodeID,  
                  BYTE bDOChannel, BYTE bValue)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

bDOChannel: [input] The subindex (>0) of index 0x6200 of specified application object. Please refer to slave device user manual for more detail information.

bValue: [input] The value for 8-channel digital output which is used 1 byte for presentation.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_StatusErr

CPM100_NodeNumberErr

CPM100_SDOSendErr

CPM100_SDODataLose

CPM100_ChannelErr

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut



4.5.36. CPM100_ReadDI

- **Description:**

Use this function to read one byte (8 channels) DI data from the specific node.

- **Syntax:**

```
int CPM100_ReadDI(BYTE bBoardNo, BYTE bNodeID,  
                 BYTE bDIChannel, BYTE *bValue)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

bDIchannel: [input] The subindex (>0) of index 0x6000 of specified application object. Please refer to slave device user manual for more detail information.

***bValue:** [output] The value for 8-channel digital input which is used 1 byte for presentation.

- **Return:**

CPM100_NoError
CPM100_ActiveBoardError
CPM100_StatusErr
CPM100_NodeNumberErr
CPM100_SDOSendErr
CPM100_SDODataLose
CPM100_ChannelErr
CPM100_SendCmdErr
CPM100_MasterInitErr
CPM100_TimeOut



4.5.37. CPM100_WriteAO

- **Description:**

Use this function to output one channel AO data to the specific node.

- **Syntax:**

```
int CPM100_WriteAO(BYTE bBoardNo, BYTE bNodeID,  
                  BYTE bAOChannel, WORD wValue)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

bAOchannel: [input] The subindex (>0) of index 0x6411 of specified application object. Please refer to slave device user manual for more detail information.

wValue: [input] One AO channel value which is used two bytes for presentation.

- **Return:**

CPM100_NoError
CPM100_ActiveBoardError
CPM100_StatusErr
CPM100_NodeNumberErr
CPM100_SDOSendErr
CPM100_SDODataLose
CPM100_ChannelErr
CPM100_SendCmdErr
CPM100_MasterInitErr
CPM100_TimeOut

4.5.38. CPM100_ReadAI

- **Description:**

Use this function to read one channel AI data from the specific node.

- **Syntax:**

```
int CPM100_ReadAI(BYTE bBoardNo, BYTE bNodeID,  
                 BYTE bAIChannel, WORD *wValue)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

bAIchannel: [input] The subindex (>0) of index 0x6401 of specified application object. Please refer to slave device user manual for more detail information.

***wValue:** [output] Read one AI channel value which is used two bytes for presentation.

- **Return:**

CPM100_NoError
CPM100_ActiveBoardError
CPM100_StatusErr
CPM100_NodeNumberErr
CPM100_SDOSendErr
CPM100_SDODataLose
CPM100_ChannelErr
CPM100_SendCmdErr
CPM100_MasterInitErr
CPM100_TimeOut



4.5.39. CPM100_GetFirmwareVersion

- **Description:**

Use this function to get the firmware version of the PISO-CPM100(U).

- **Syntax:**

int CPM100_GetFirmwareVersion(**BYTE** bBoardNo, **WORD** *wVersion)

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

***wVersion:** [output] Firmware version of the PISO-CPM100(U).

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut



4.5.40. CPM100_COBIDInfo

- **Description:**

Use this function to get EMCY COB ID, SYNC COB ID, all RxPDO, and all TxPDO COB ID of the specific slave.

- **Syntax:**

```
int CPM100_COBIDInfo(BYTE bBoardNo, BYTE bNodeID,  
                    WORD *wSYNCID, WORD *wEMCYID,  
                    BYTE *bRxPDONum, WORD *wRxPDOID,  
                    BYTE *bTxPDONum, WORD *wTxPDOID)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

bNodeID: [input] Slave device Node-ID (1~127).

***wSYNCID:** [output] To Get the SYNC COB ID.

***wEMCYID:** [output] To Get the EMCY COB ID.

***bRxPDONum:** [output] The number of RxPDO of the slave.

***wRxPDOID:** [output] This is an array of total RxPDO COB ID.

***bTxPDONum:** [output] The number of TxPDO of the slave.

***wTxPDOID:** [output] This is an array of total TxPDO COB ID.

- **Return:**

CPM100_NoError

CPM100_ActiveBoardError

CPM100_NodeNumberErr

CPM100_SendCmdErr

CPM100_MasterInitErr

CPM100_TimeOut

4.5.41. CPM100_PDOMappingInfo

- **Description:**

Use this function to get the mapping status of the PDO of the specific slave.

- **Syntax:**

```
int CPM100_PDOMappingInfo(BYTE bBoardNo, WORD wCobid,
                           BYTE *bRxTxType, BYTE *bPDOEntry,
                           BYTE *bLen, BYTE *bRxData,
                           BYTE *bMappingData)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

wCobid: [input] Slave device Node-ID (1~127).

***bRxTxType:** [output] PDO type (0: RxPDO, 1: TxPDO).

***bPDOEntry:** [output] The total useful entry of the PDO.

***bLen:** [output] The byte number of the PDO.

***bRxData:** [output] The last output data of the PDO. The parameter is only for RxPDO.

***bMappingData:** [output] 4-byte information of mapped application object per entry. So, the size of the bMappingData array must more than (bPDOEntry * 4). The following information is for first PDO entry. The second entry is from bMappingData[4] to bMappingData[7] and so on.

bMappingData [0] : The numbers of bit of specified application object.

bMappingData [1] : The subindex of specified application object.

bMappingData [2] : The low byte of index of specified application object.

bMappingData [3] : The high byte of index of specified application object.



- **Return:**

- CPM100_NoError
- CPM100_ActiveBoardError
- CPM100_SendCmdErr
- CPM100_MasterInitErr
- CPM100_CobldErr
- CPM100_TimeOut



4.5.42. CPM100_GetNodeList

- **Description:**

Use this function to get the total node ID list that has been added into the master.

- **Syntax:**

```
int CPM100_GetNodeList(BYTE bBoardNo, BYTE *bNodeList)
```

- **Parameter:**

bBoardNo: [input] PISO-CPM100(U) DIP switch No.(0~15).

***bNodeList:** [output] The node list parameter is a 16-bytes array. Per byte means un-added/added of 8 node id (0: un-added, 1: added). For example, the data of bNodeList[0] is 0x2A (binary data: 0010 1100), this means node id 2,3,and 5 have been added into the master. And the data of bNodeList[1] is 0x49 (binary data: 0100 1001), this means node id 8,11,and 14 have added into the master.

- **Return:**

CPM100_NoError
CPM100_ActiveBoardError
CPM100_SendCmdErr
CPM100_MasterInitErr
CPM100_TimeOut



5. Demo Programs

There are SDO, PDO, TxType demos and one test program, TestProg, for VB6.0/VC++/BCB6. Users can refer these source codes to develop various applications. There is also a tool, CPM_Utility, to control/monitor CANopen slaves with PISO-CPM100(U) easily and quickly. Users can find these demos and utility tool in the fieldbus CD or on the web site.

The path of CAN CD

fieldbus_cd://canopen/master/piso-cpm100

The address of the web site is

http://www.icpdas.com/products/Remote_IO/can_bus/piso-cpm100.htm

5.1. Brief of the demo programs

These demo programs are developed for demonstrating how to use the CANopen master library to apply the general CANopen communication protocol, SDO, PDO, NMT, SYNC, and EMCY. Each communication protocol is achieved by using different functions of CANopen master library. The relationship between CANopen master library functions and CANopen communication protocols are displayed in the following description.

NMT Services: CPM100_NMTChangeState, CPM100_NMTGetState,
NMTCPM100_Guarding

SDO Services: CPM100_SDOReadData, CPM100_SDOWriteData,
CPM100_SDOAbortTransmit

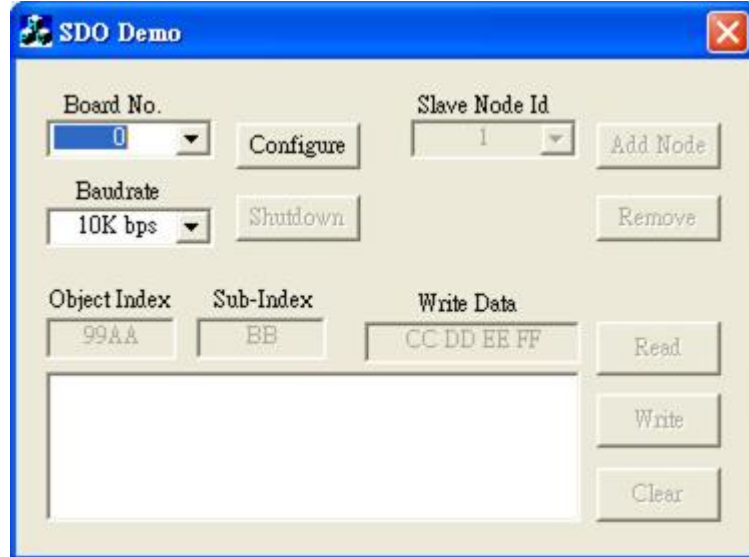
PDO Services: CPM100_InstallPDO, CPM100_DynamicPDO,
CPM100_RemovePDO, CPM100_PDOTxType,
CPM100_SetEventTimer, CPM100_PDOWrite,
CPM100_PDORemote, CPM100_ReadPDOCount,
CPM100_ReadPDOMessage, CPM100_ChangePDOCobID

SYNC Services: CPM100_ChangeSYNCID, CPM100_SendSYNC

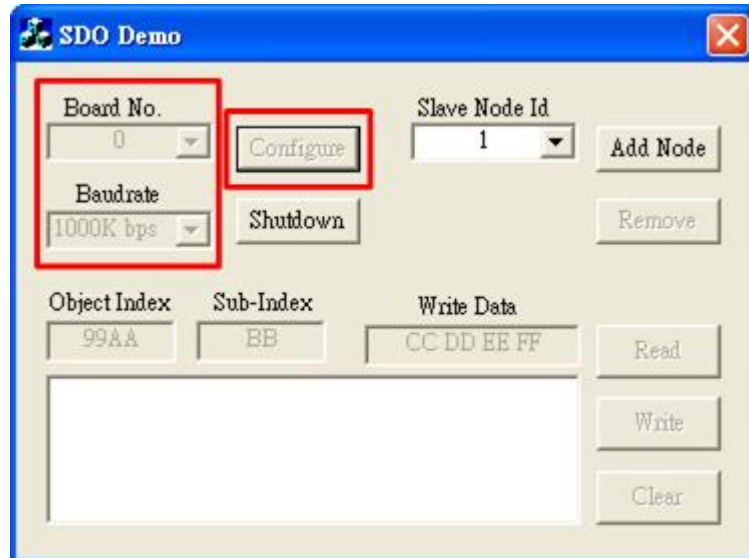
EMCY Services: CPM100_ChangeEMCYID, CPM100_ReadEMCYCount,
CPM100_ReadEMCY

SDO demo

When the demo runs, the user interface of the demo is shown below.



Before click the “Configure” button, Users must select “Board No.” and “Baudrate” firstly. The “Board No.” is the DIP Switch number of PISO-CPM100(U), and the “Baudrate” is the CANopen communication speed.



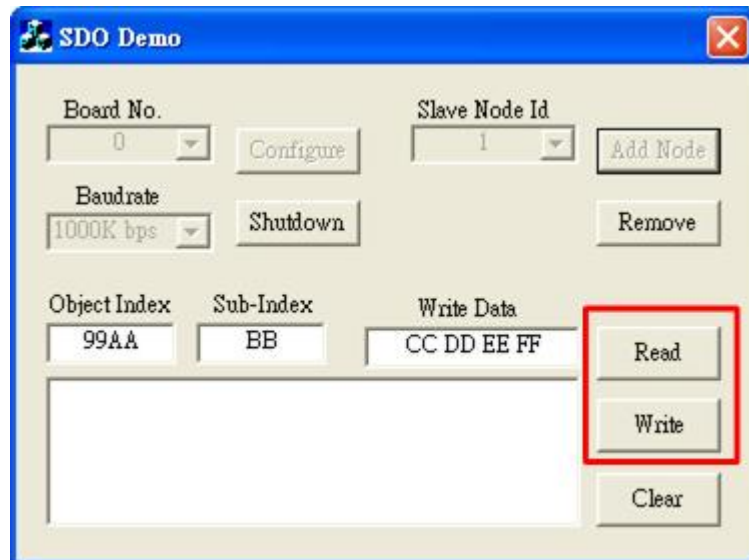


Select the “Slave Node Id” and click “Add Node” button to add the slave into the control list of PISO-CPM100(U).

Attention, the slave must be connected on the CANopen network really before click “Add Node” button.

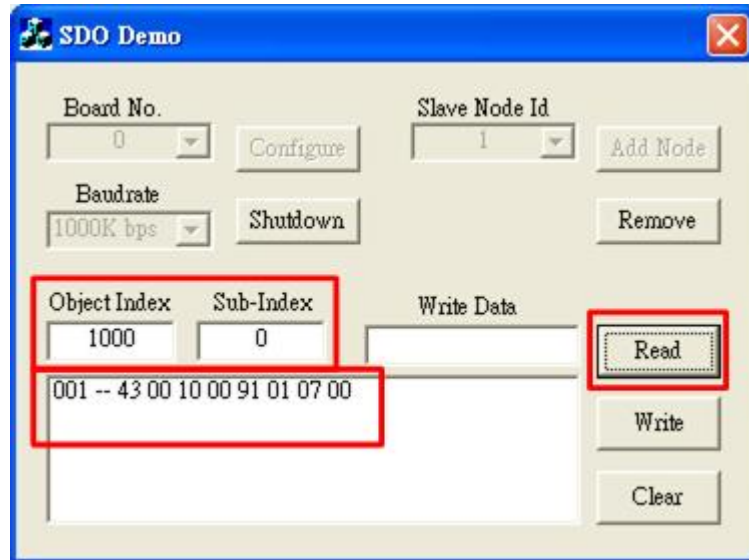


There are two SDO functions to use, “Read” button is the CPM100_SDOReadData, and “Write” button is the CPM100_SDOWriteData.

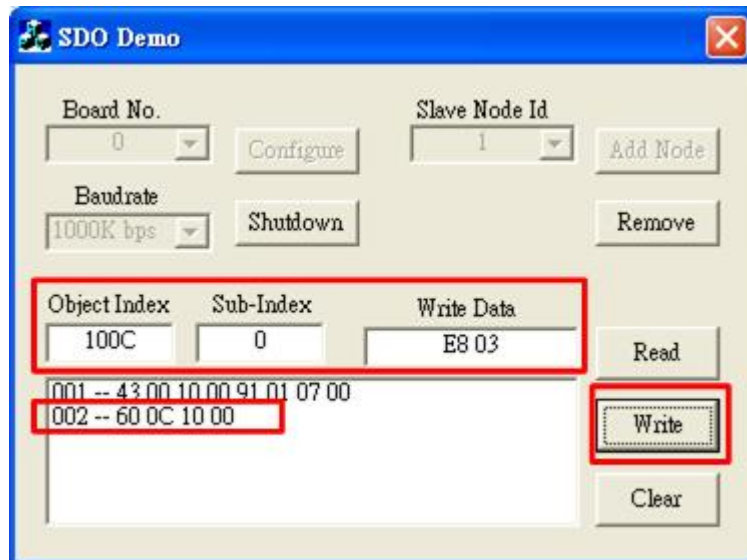




Input “Object Index” and “Sub-Index” of the object dictionary of the slave, and click “Read” button to read the object data. The response data would show on the list box as follows.

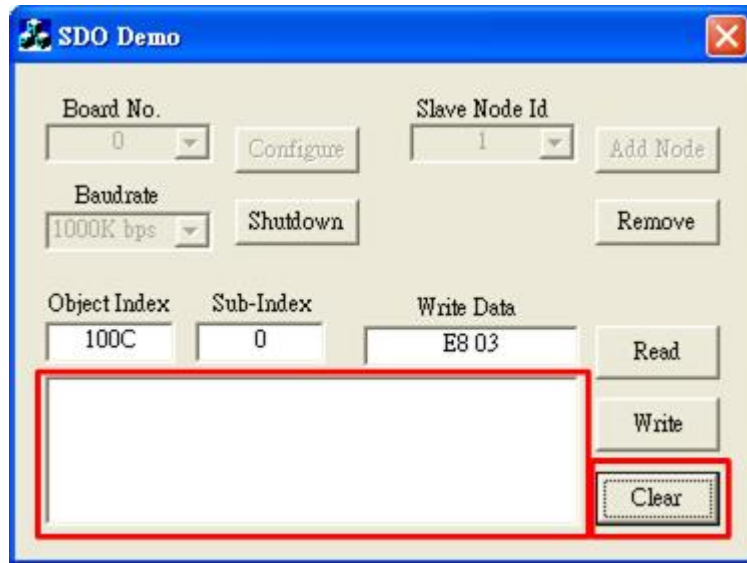


Set these values in “Object Index” field, “Sub Index” field, and “Write Data” field. Then click “Write” button to write the object data. The response message would show on the list box as follows.

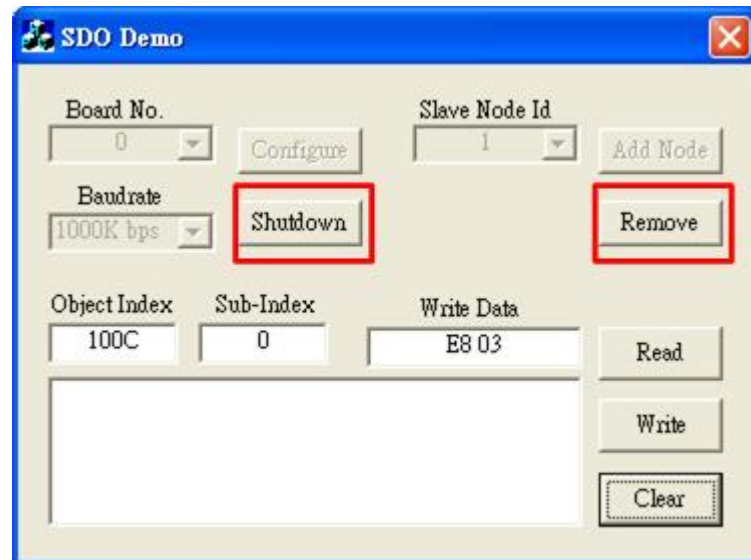




Clicking the “Clear” button would clear all the messages shown on the list box.



If users want to control another slave, users can click “Remove” button to remove the slave and then select another slave to add. Or if users want to change another board, the “Shutdown” button would execute CPM100_ShutdownMaster function to close the board. And then users can select another board to control.

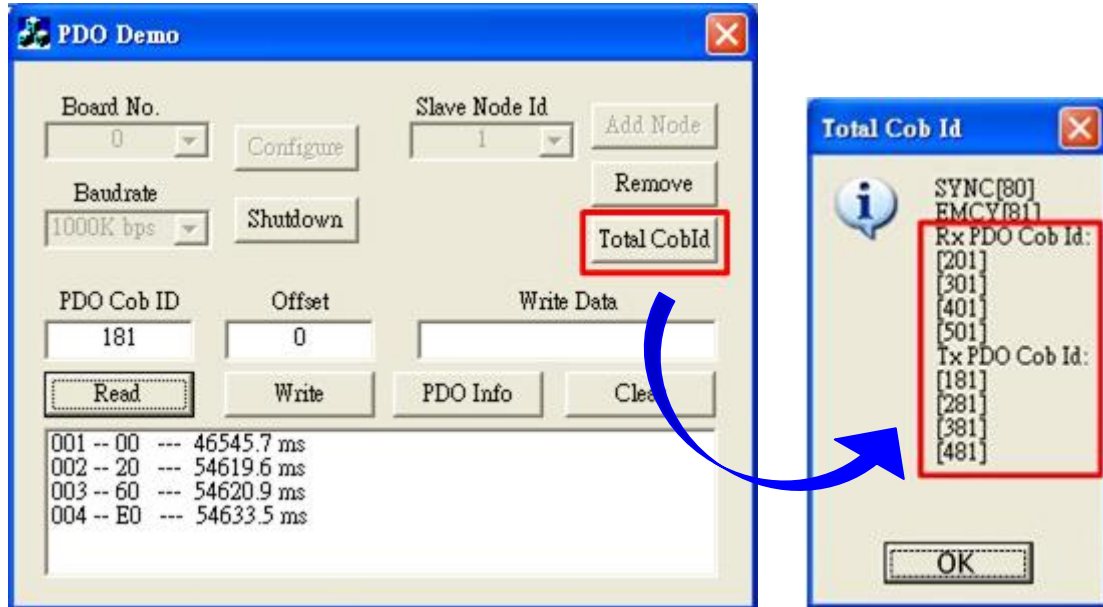


PDO demo

When the demo runs, the user interface of the demo is shown below.

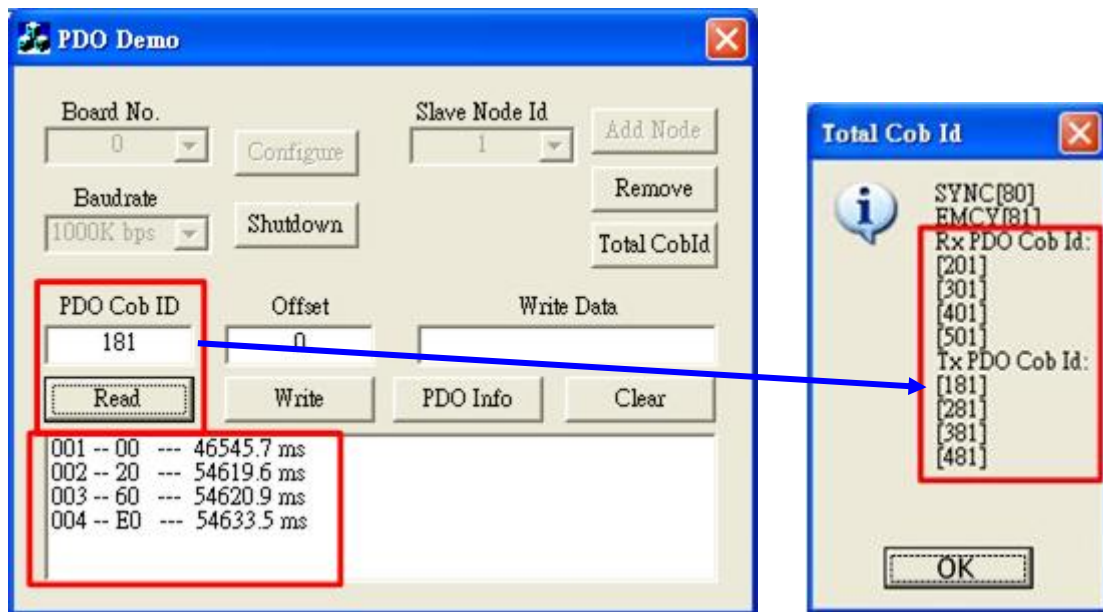


It is the same as the SDO demo. Firstly, users must select the board number and baud rate to configure, and select the slave ID to control. After finishing these steps, users can click the “Total CobId” button to know which PDO COB ID on the slave can be controlled.

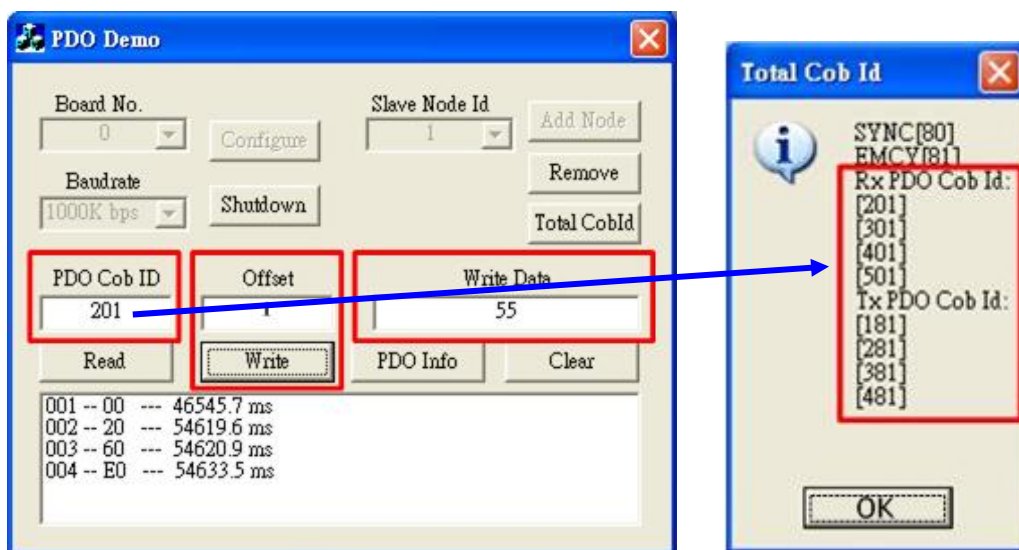




To click “Read” button, users can read the PDO data with “PDO Cob ID”. And the “PDO Cob ID” can be shown in the “Tx PDO Cob Id:” list.

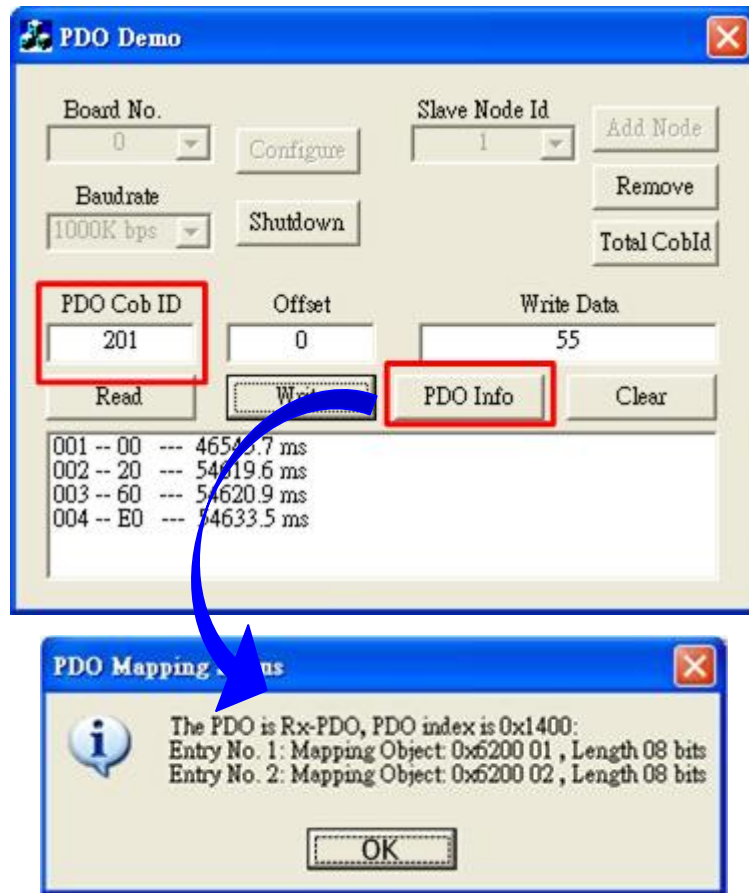


If users want to output the PDO data to the slave, users must set the values of “PDO Cob ID”, “Offset”, and “Write Data” in three edit boxes firstly. The “PDO Cob ID” must be list in the “Rx PDO Cob Id:” list. And the “Offset” is meaning that PDO data would be output from the specific byte. So, the output data would not be changed before the “Offset” byte. For example, firstly, if the “Offset” is “0” and the “Write Data” is “FF FF”. The PDO would output data “FF FF”. Second, if the “Offset” is changed to “1” and “Write Data” is changed to “55”. The PDO would output data “FF 55”, the first byte “FF” would not be changed.



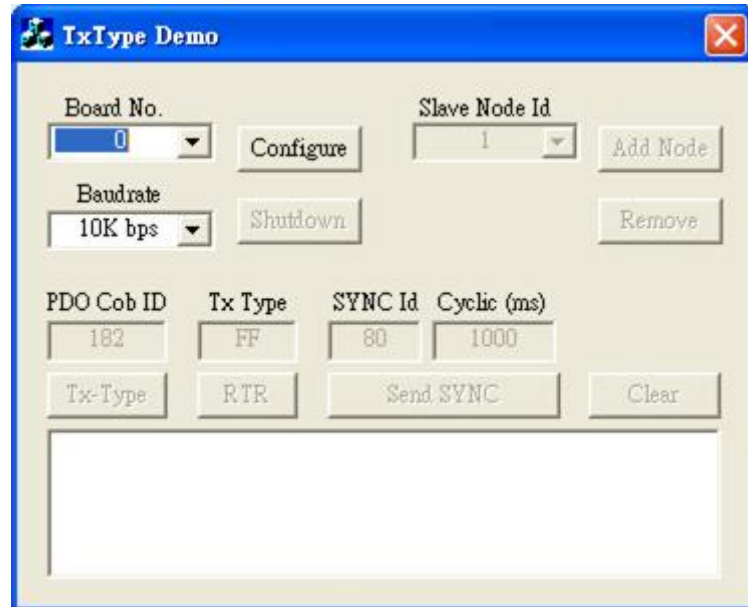


By clicking “PDO Info” button, users can check the information of PDO mapping status below.

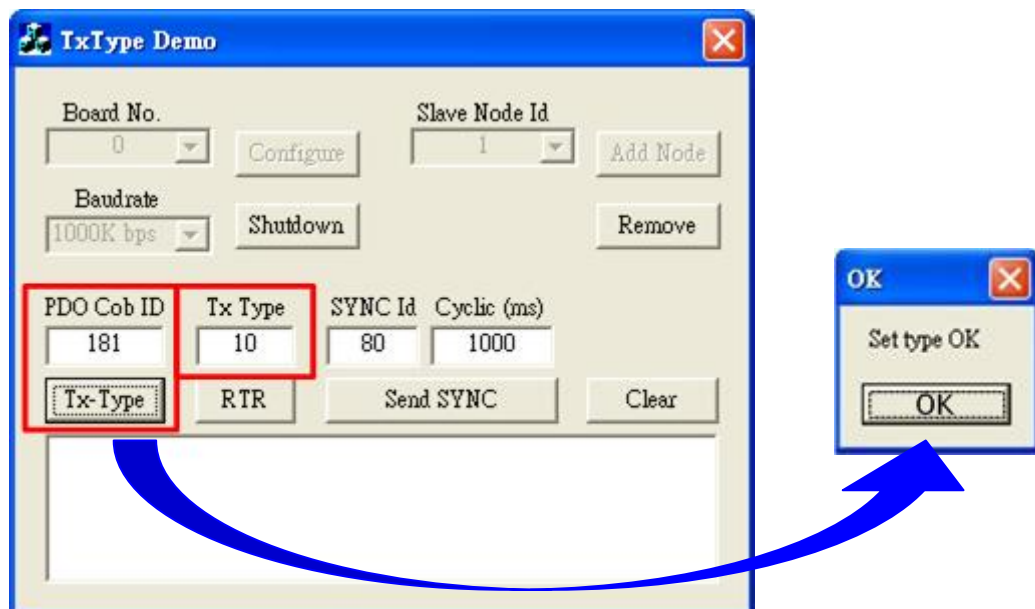


Tx Type demo

When the demo runs, the user interface of the demo is shown below.



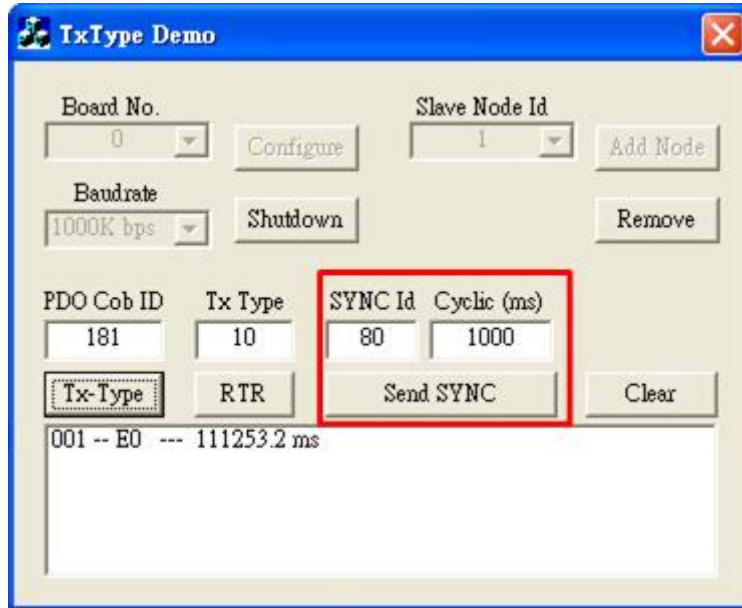
This demo can help users to test the TxPDO transmission type of the slave. For example, input “181” in “PDO Cob ID” and “10” in “Tx Type”, and then click the “Tx-Type” button to set the transmission type.



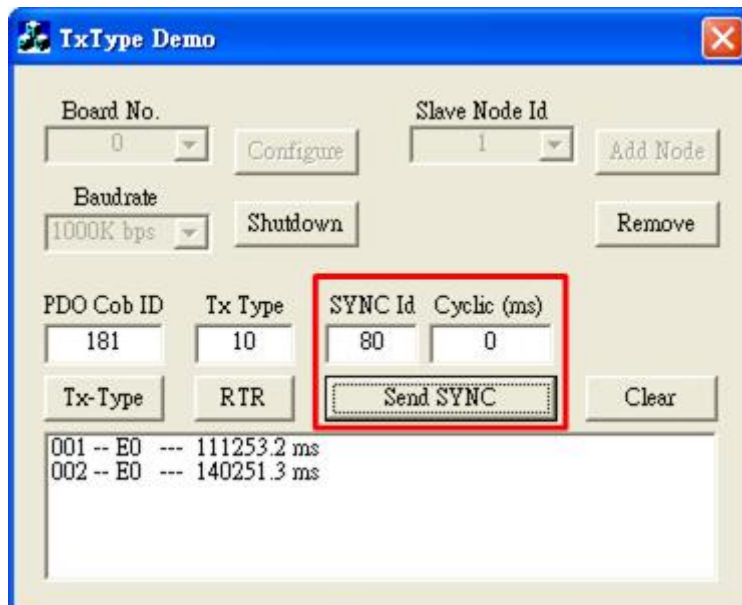


When the slave receives 16 (0x10) times SYNC message, the slave would response the 0x181 PDO message.

To send SYNC message, users have two kinds, there are automatic and manual modes to uses. For sending SYNC message automatically, users must set the timer in “Cyclic” liking “1000”. After clicking the “Send SYNC” button, the PISO-CPM100(U) would send SYNC message per 1000 million seconds. Anyway, after 16 seconds, the slave would response a PDO message.



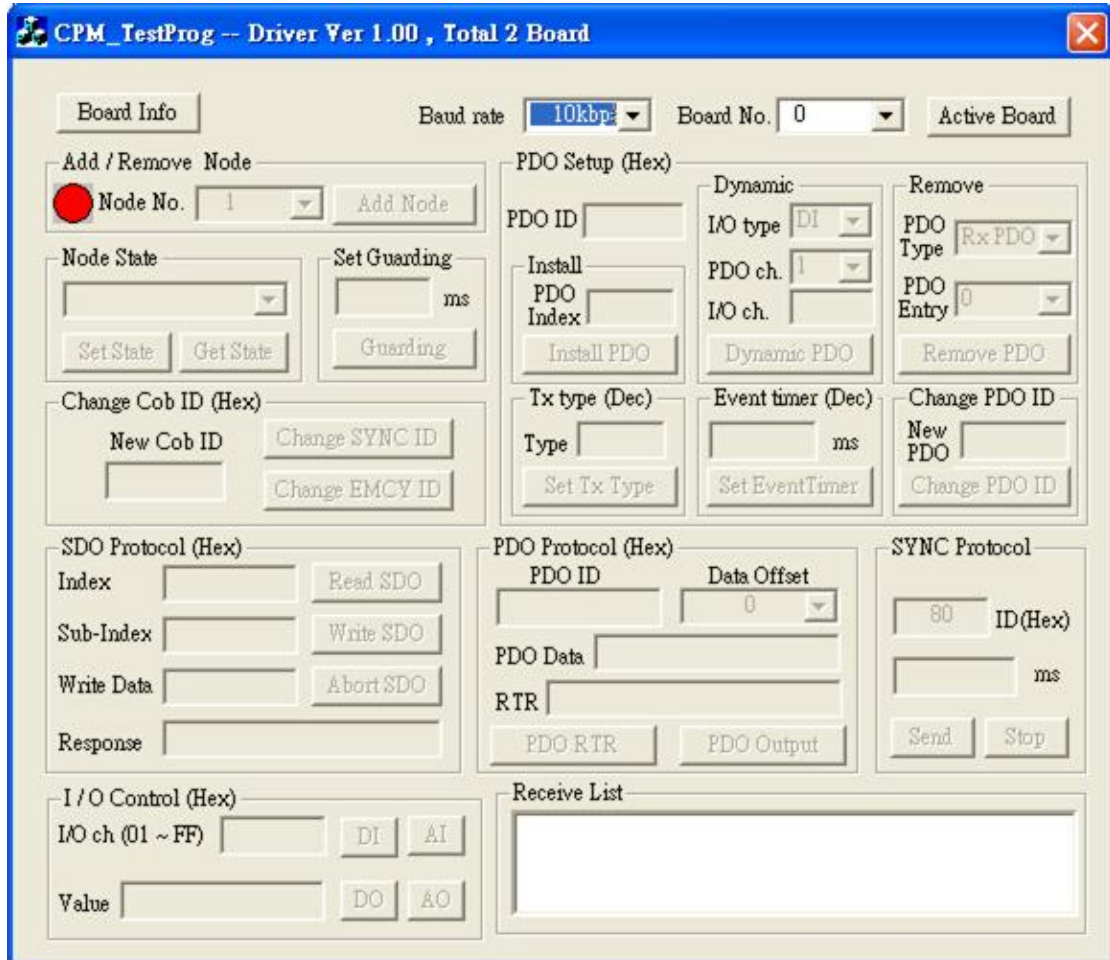
For sending SYNC message manually, users must input “0” in “Cyclic”. By clicking the “Send SYNC” button, the PISO-CPM100(U) would send one SYNC message per time. After 16 times, the slave would response the PDO message.





TestProg

When the TestProg runs, the user interface of this program is shown below.



The first step to use this program is to select the baud rate and board number. Then click “Active Board” button to control this PISO-CPM100(U).

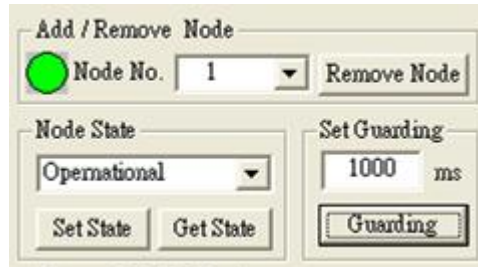
Except the acting board and adding node, the program has 8 parts to use. They are Node State, Guarding, Change Cob ID, PDO Setup, SDO Protocol, PDO Protocol, SYNC Protocol, I/O Control, and Receive List. However, about the SDO Protocol, PDO Protocol, and the SYNC Protocol can refer to the SDO demo, PDO demo, and Tx Type demo.



Node State and Guarding:

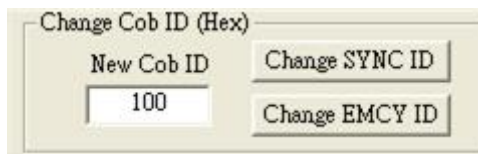
The part of Node State can get/set the slave working status. There are four kinds of status; Pre-Operational, Operational, Stop, and Reset can set to the slave.

The Guarding part can set guarding time to guard the slave. If the bus between PISO-CPM100(U) and slave is broken, the PISO-CPM100(U) can detect this condition.



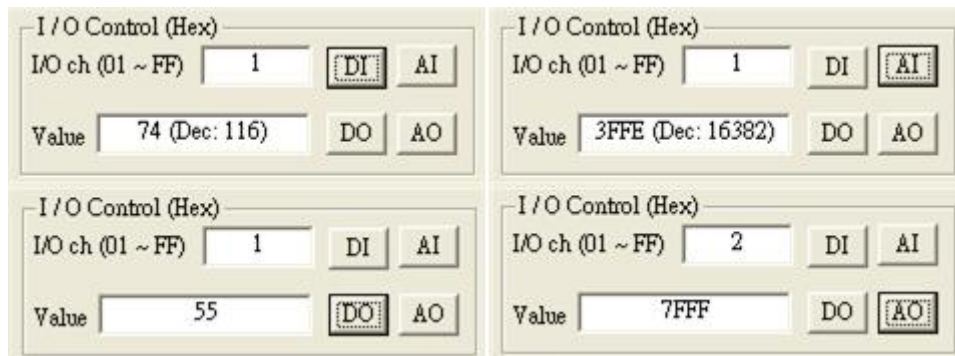
Change Cob ID:

If users want to change COB ID of SYNC or EMCY, this part provides the function to do that easily.



I/O Control:

Users can read DI/AI and write DO/AO in this I/O Control part. Firstly, users must set the value for which I/O channel to read/write. And then click “DI”/“AI” button to read data or input the “Value” with hex format. Also click “DO”/“AO” button to write data to the slave below.



PDO Setup:

The PDO Setup part has 6 functions to setup PDO parameters. The following procedures would introduce how to use these 6 functions.

Install PDO:

Select the slave firstly and input the values of PDO ID and PDO Index. Then click “Install PDO” button to add PDO to the slave and the PDO is under control. Please refer to the section 4.5.25 for the more detail.

The screenshot shows the 'PDO Setup (Hex)' dialog box. It is divided into several sections:

- Dynamic:** I/O type (DI), PDO ch. (2), I/O ch. (3). A 'Dynamic PDO' button is present.
- Remove:** PDO Type (Tx PDO), PDO Entry (3), and a 'Remove PDO' button.
- Change PDO ID:** New PDO (168) and a 'Change PDO ID' button.
- Event timer (Dec):** 1000 ms, with a 'Set EventTimer' button.
- Tx type (Dec):** Type (0), with a 'Set Tx Type' button.
- Install:** PDO Index (1605) and an 'Install PDO' button. This section is highlighted with a red box.
- Dynamic:** PDO ID (181). This field is also highlighted with a red box.

Installing Dynamic PDO:

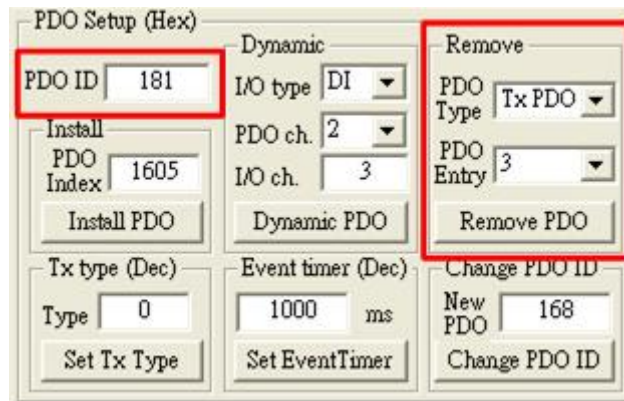
Select the slave and input the PDO ID firstly. Then users must select the I/O type for the PDO and “PDO ch.” for which PDO entry would be mapped in. Finally, set the “I/O ch.” for which I/O would be mapped and click the “Dynamic PDO” button to complete the dynamic mapping. In this part, the “I/O ch.” is the same as the “I/O ch.” in the I/O Control part. Please refer to the section 4.5.24 for the detail.

This screenshot is identical to the previous one, but the 'Dynamic' section is highlighted with a red box. This section includes:

- Dynamic:** I/O type (DI), PDO ch. (2), I/O ch. (3), and a 'Dynamic PDO' button.

Removing PDO:

When users want to remove the PDO object, this function would be useful. Firstly, select slave and input the PDO ID. Secondly, select the PDO type of RxPDO or TxPDO which your need. Finally, select the PDO entry which users want to remove and click “Remove PDO” button. If the PDO entry is 0, this function would remove this PDO object completely. Please refer to the section 4.5.26 for the more detail.

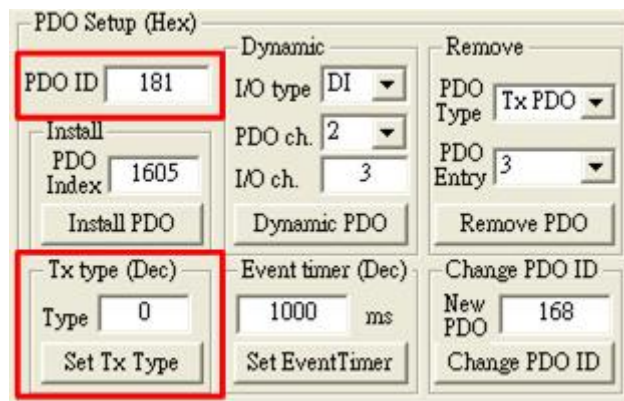


The screenshot shows the 'PDO Setup (Hex)' dialog box. The 'Remove' section is highlighted with a red box. It contains the following fields and buttons:

- Dynamic** (radio button, selected)
- I/O type**: DI (dropdown)
- PDO ch.**: 2 (dropdown)
- I/O ch.**: 3 (text input)
- PDO ID**: 181 (text input, highlighted in red)
- PDO Type**: Tx PDO (dropdown)
- PDO Entry**: 3 (dropdown)
- Remove PDO** (button)
- Install PDO Index**: 1605 (text input)
- Install PDO** (button)
- Dynamic PDO** (button)
- Tx type (Dec) Type**: 0 (text input)
- Set Tx Type** (button)
- Event timer (Dec)**: 1000 ms (text input)
- Set EventTimer** (button)
- Change PDO ID** (button)
- New PDO**: 168 (text input)
- Change PDO ID** (button)

Setting Transmission Type:

Select the slave firstly. Set the PDO ID and the type value for decimal. And then click the “Set Tx Type” button to set transmission type. Please refer to the section 4.5.27 for the more detail.

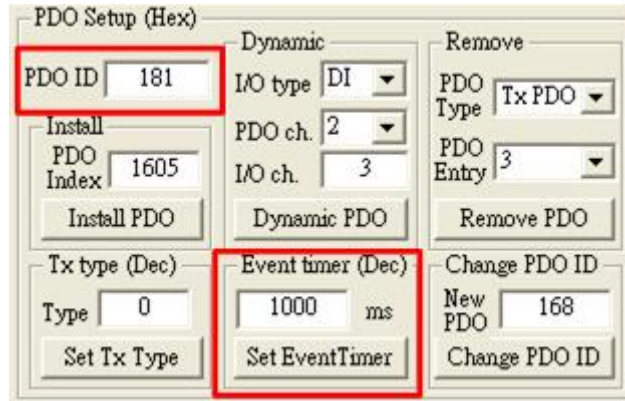


The screenshot shows the 'PDO Setup (Hex)' dialog box. The 'Tx type (Dec)' section is highlighted with a red box. It contains the following fields and buttons:

- Dynamic** (radio button, selected)
- I/O type**: DI (dropdown)
- PDO ch.**: 2 (dropdown)
- I/O ch.**: 3 (text input)
- PDO ID**: 181 (text input, highlighted in red)
- PDO Type**: Tx PDO (dropdown)
- PDO Entry**: 3 (dropdown)
- Remove PDO** (button)
- Install PDO Index**: 1605 (text input)
- Install PDO** (button)
- Dynamic PDO** (button)
- Tx type (Dec) Type**: 0 (text input, highlighted in red)
- Set Tx Type** (button)
- Event timer (Dec)**: 1000 ms (text input)
- Set EventTimer** (button)
- Change PDO ID** (button)
- New PDO**: 168 (text input)
- Change PDO ID** (button)

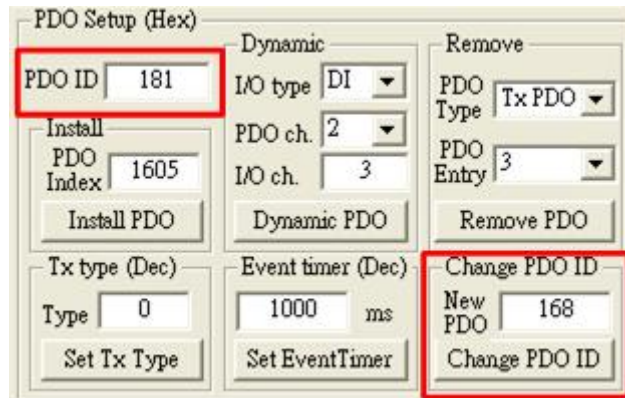
Setting Event Timer:

Select the slave firstly. Set the PDO ID and the timer for decimal. And click the “Set EventTimer” button to set event timer to the PDO object of the slave. The timer unit is million second. For more detail, please refer to the section 4.5.28.



Changing PDO ID:

Select the slave firstly. Set the PDO ID and the “New PDO” for hex. And click the “Change PDO ID” button to change PDO COB ID. For more detail, please refer to the section 4.5.29.



Receive List:

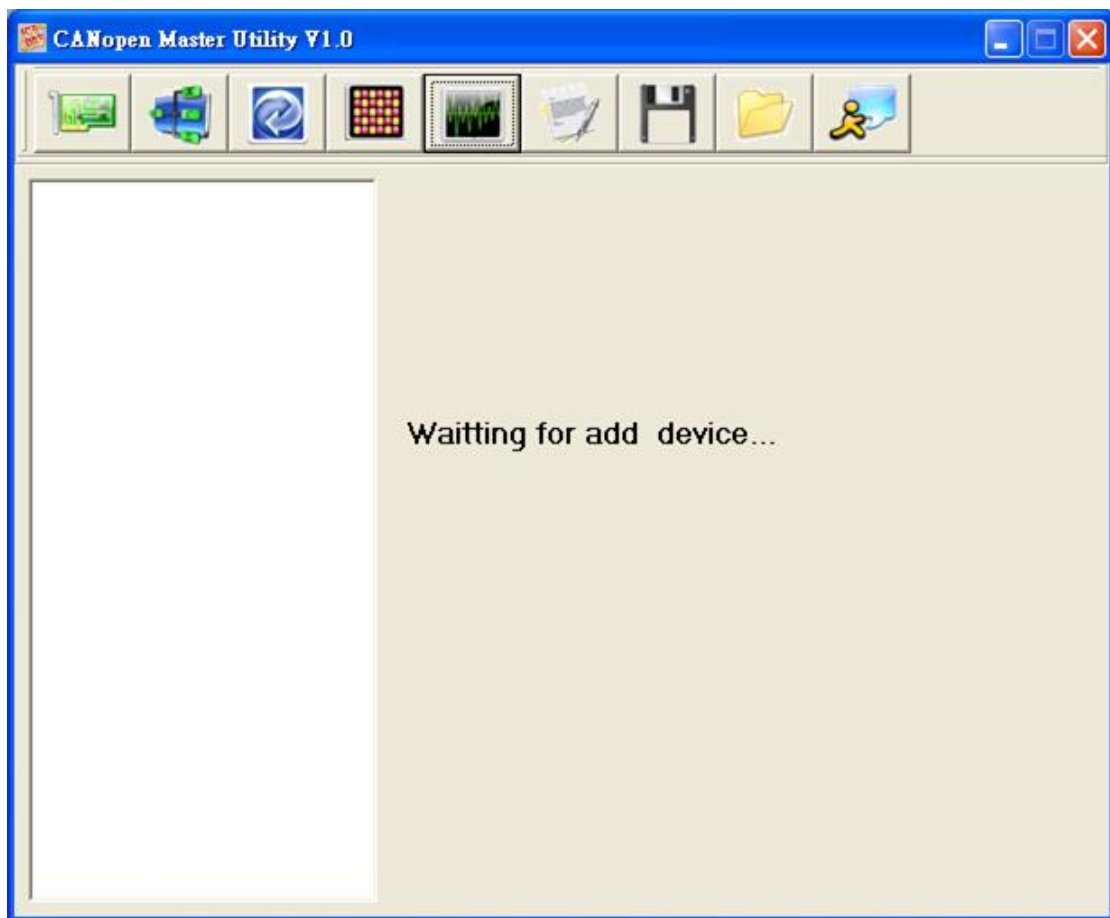
The Receive list can receive POD messages with event trigger and EMCY messages. The Receive list can show maximum 100 messages.



6. CPM_Utility Introduction

The CANopen Master Utility is a master tool of CANopen for PISO-CPM100(U). CPM_Utility can control the CANopen slave that follow DS301 and DSP401. If users would not develop CANopen application by themselves, CPM_Utility is a good choice.

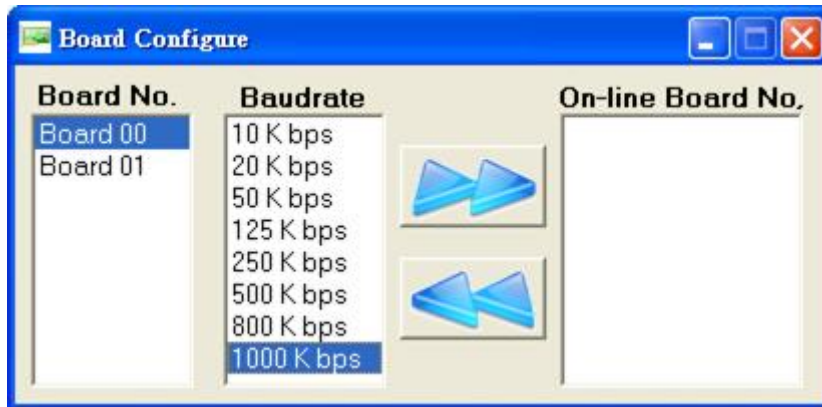
When the CPM_Utility runs, the user interface of this program is shown below.



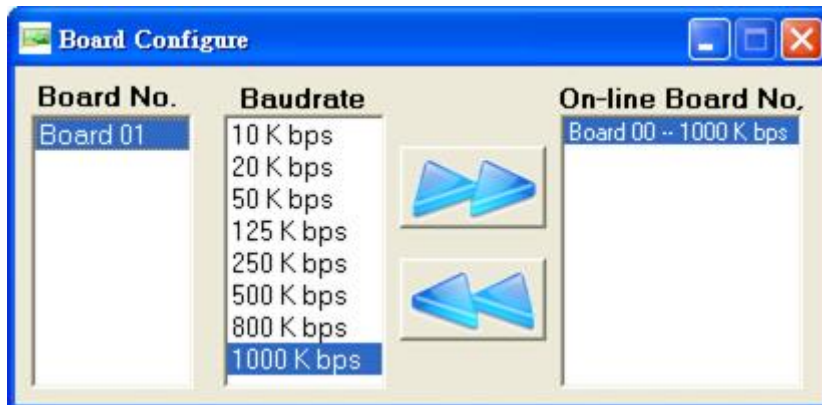
There are 9 buttons on the tool bar, from left to right. They are “Board Configure”, “Node Configure”, “Refresh slave parameter”, “DI/DO control”, “AI/AO control”, “Save receive messages”, “Save CPM_Utility Setting”, “Load CPM_Utility Setting”, and “About us”. Following, we would introduce how to use this tool below.

6.1. Board Configure

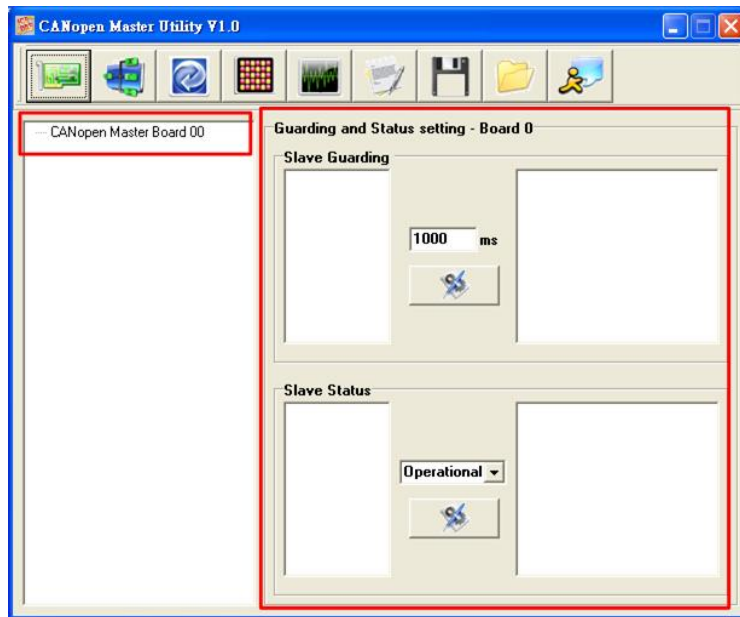
The button needs to be executed firstly after the CPM_UTILITY is running. The following picture is the form of “Board Configure”.



The “Board No” list would show total PISO-CPM100(U) boards of the PC. And the “On-line Board No” list would show which PISO-CPM100(U) has been activated. When users select the “Board No” and “Baudrate”, and then click the “Active Board” button, the selected board would be activated shown in the “On-line Board No” list as follows.



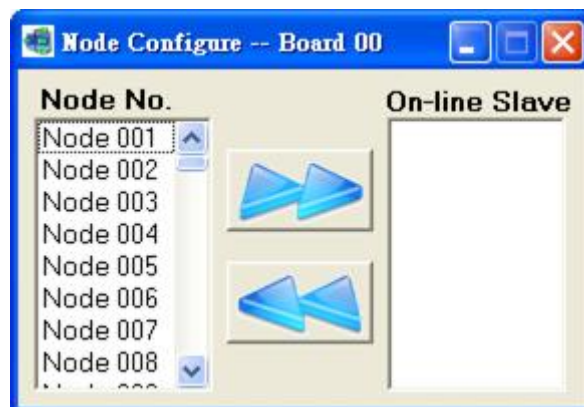
The following picture is the main form after active board 0. Because there is no slave been added, the tree view list, the “Guarding, and Status setting” group have not any slave list there.



If users want to close the activated board, only need to select the “On-line Board No” and click the “Close Board” button.

6.2. Node Configuration

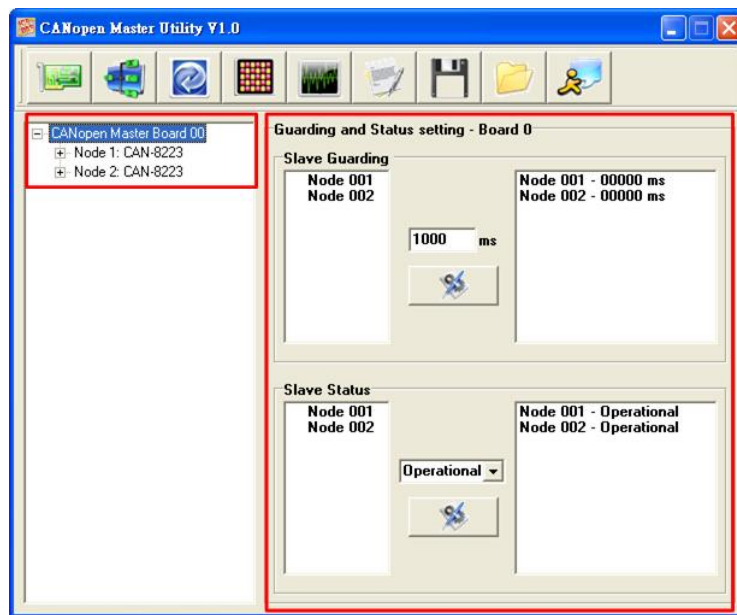
The node Configuration can add slaves to the control list of master. The button needs to be executed after the board has been activated. The following picture is the form of “Node Configure”.



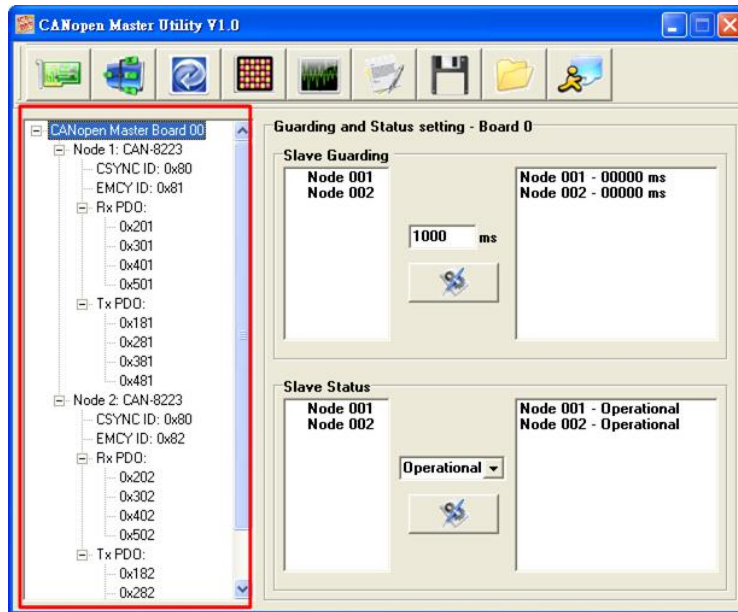
The “Node No” list shows the total slave IDs from 001 to 127. And the “On-line Slave” list would show which slave has been added. When users select the “Node No” and click the “Add slave” button, the selected slave would be added to the “On-line Slave” list as follows.



The following picture is the main form after adding slaves from the left selection list. In this example, there are two slaves shown on the tree view list and the “Guarding and Status setting” group. Users can set guarding function and work status to slaves in this group.

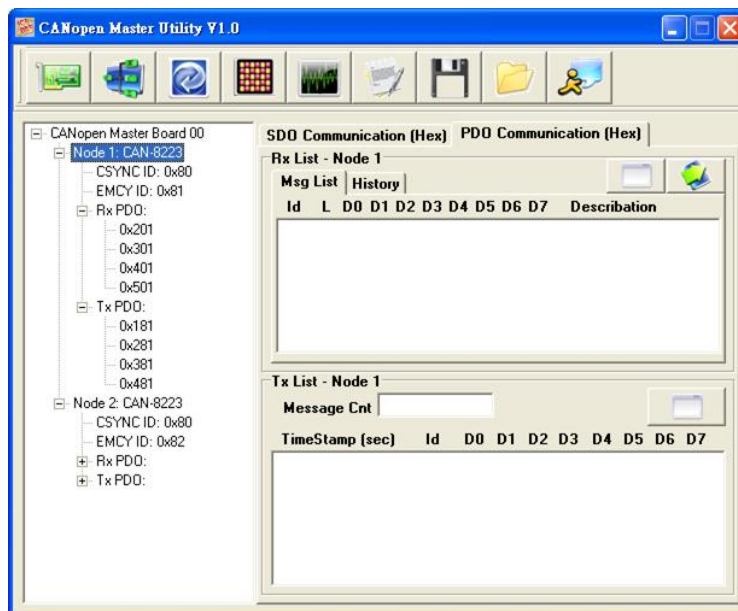


Expand the Node 1 and Node 2 tree list; users can see the Node, SYNC, EMCY, RxPDO, and TxPDO five parts below.

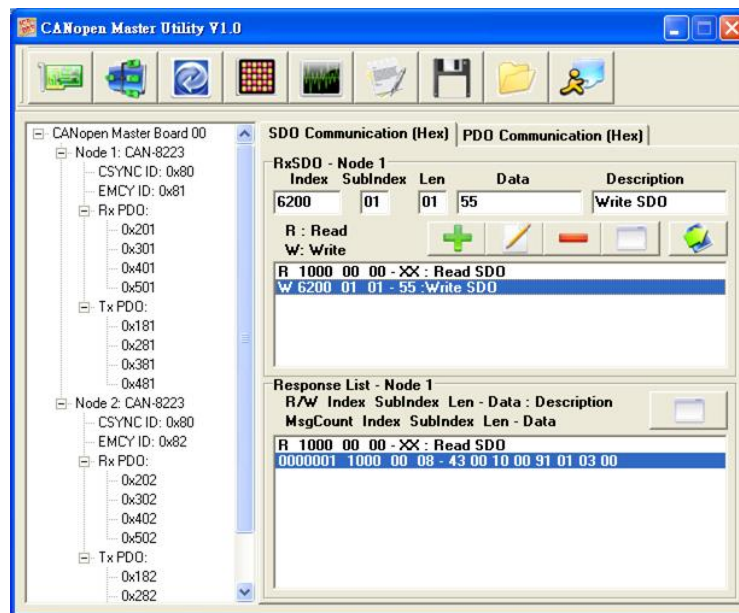


Node Part:

The Node part has two pages which are PDO Communication and SDO Communication. In the PDO Communication page, there are two groups which are Rx List and Tx List. The Rx List lists the sent PDO messages, and the Tx List lists the received PDO messages. More detail about these would be introduced at **RxPDO Part** and **TxPDO Part**.



The SDO Communication page also has two groups which are RxSDO List and Response List. In RxSDO list, there are five buttons to set the list.



The “Add” button can add the messages from edit box to the list. For example, set “6200” to “Index”, “01” to “SubIndex”, “01” to “Len”, “55” to “Data”, and “Write SDO” to “Description”, and then click “Add” button. The message “W 6200 01 01 – 55: Write SDO” would be added to the list. The “W” of the message means writing data to the specific slave because “Len” is not 0. If “Len” is 0, the header of the message would be “R” (Read data from specific slave).

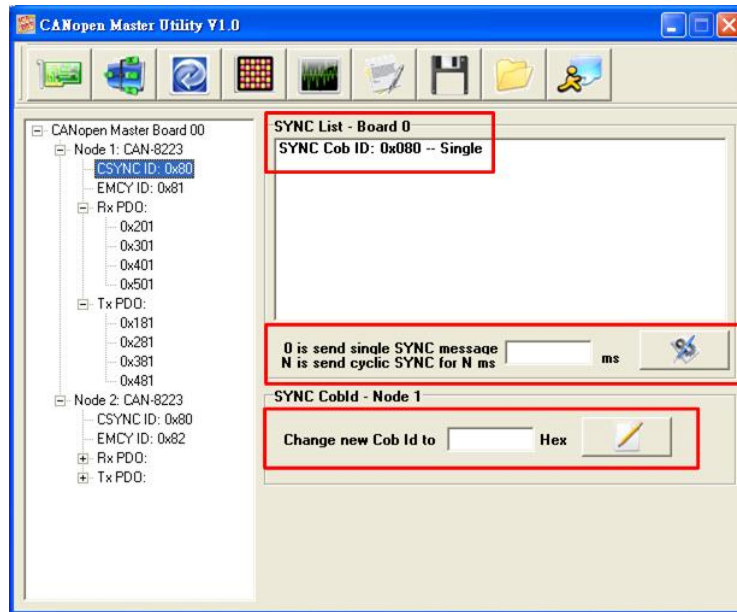
If users want to change some values of the messages in the list, key in the new value in the edit box firstly and click “Change” button to update this message.

“Delete” button can delete one of the messages shown in the list. And “Clear” button can delete all of them.

Select the messages and click “Send” button to send this message. To double click this message shows on the list also can send this message. When a message has been sent, the “Response List” would list what to send. If the utility has received the response message, the “Response List” would also list what to receive.

SYNC Part:

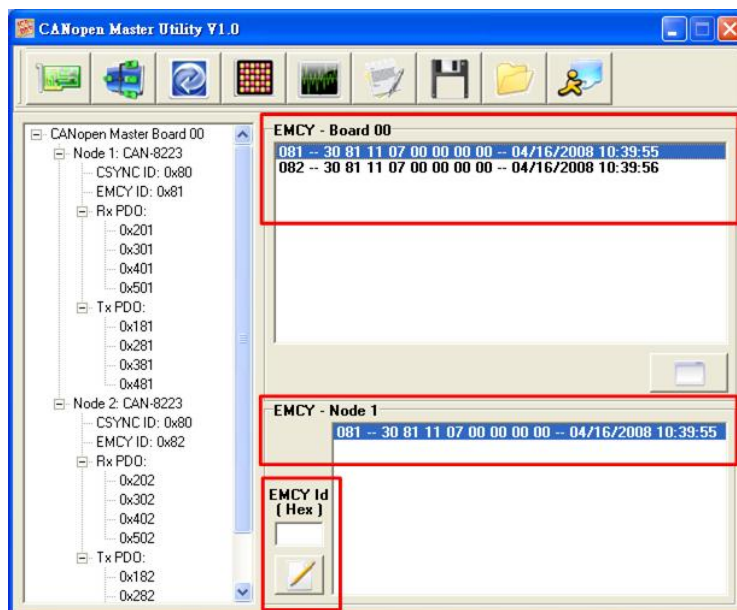
In this part, users can send SYNC message and change SYNC ID. Select which SYNC ID to send firstly, key in the cyclic timer in the timer edit box, and click “Send SYNC”. If the timer is 0, the SYNC message would be sent once per click. But if the timer is not 0, for example 100, the SYNC message would be sent cyclic with 100 million seconds.



Key in the new SYNC ID in the ID edit box and click “Change SYNC ID” button to change SYNC ID.

EMCY Part:

The EMCY part can list the EMCY messages and change EMCY ID. There are two EMCY lists, the Board EMCY and the Node EMCY. The Board EMCY list is the upper list box; it can show all the EMCY received by the PISO-CPM100(U). And the Node EMCY is the lower list box; it can only show the EMCY messages that produced from the specific slave. For example, the Board EMCY list has two messages but the Node EMCY list has only the node 1 EMCY message in the following picture.



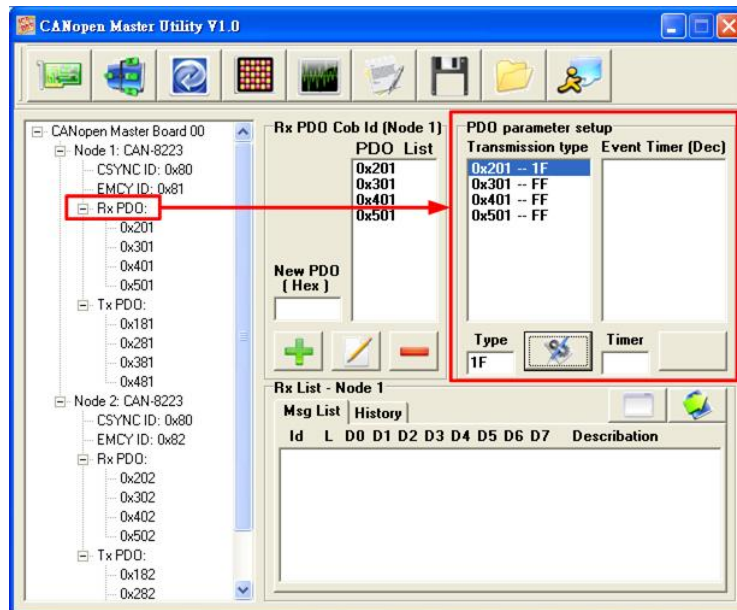


If users want to change the EMCY ID of the slave, users can key in the new ID in the EMCY ID edit box and click the “Change EMCY ID” button to change it.

RxPDO Part:

In this part, users can set the transmission type, install a new PDO, change the PDO ID, map a PDO dynamically, and send the PDO message.

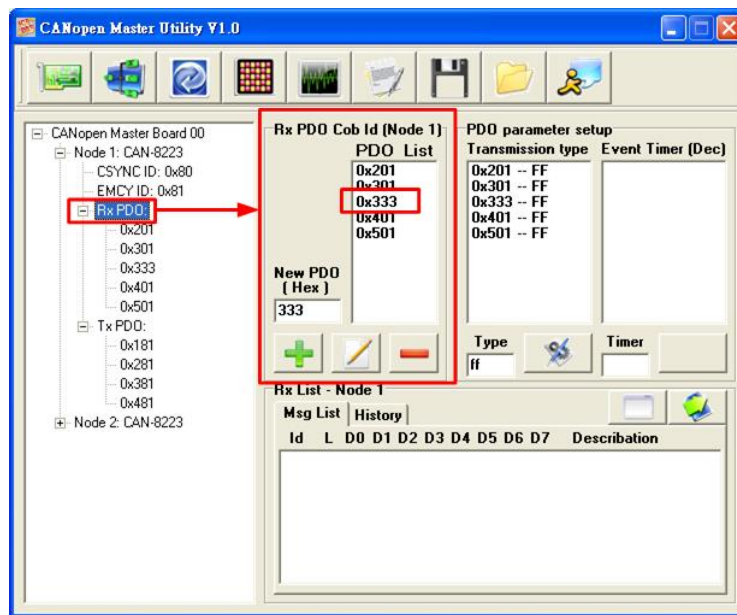
For setting the transmission type, users can select the “RxPDO” label to see “PDO parameter setup” group below.



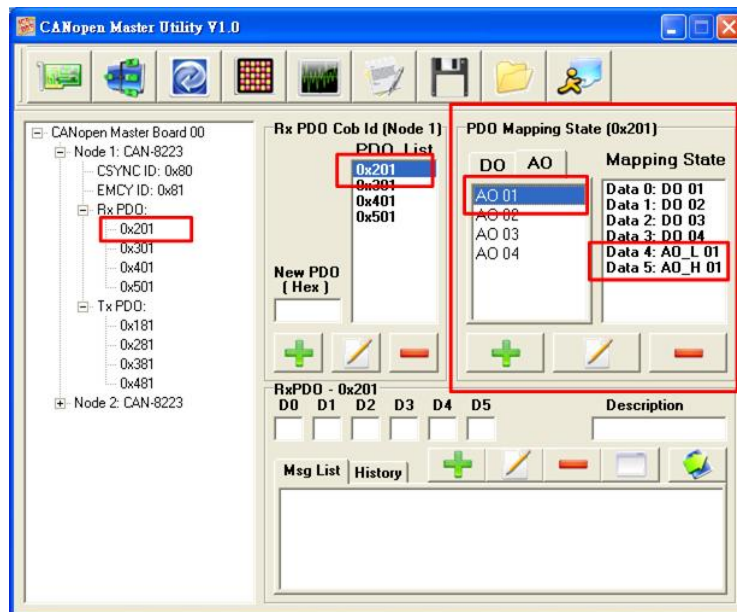
Select the PDO needed to be set the transmission type on the “Transmission type” list, key in the type value to the “Type” edit box, and then click the “Set Transmission Type” button to complete this setting.

Note that, the “Event Timer” is useless for RxPDO.

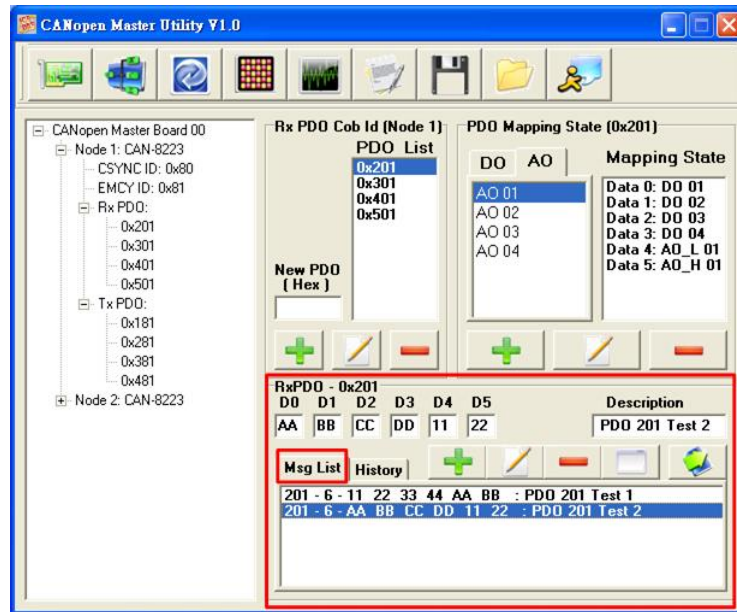
For installing a new PDO, users can input a new PDO ID in “New PDO” edit box and then click the “Add” button to add a new PDO. If this PDO is needed no longer, users can click “Delete” button to remove this PDO. Or to click “Change PDO ID” button to change the PDO ID to “New PDO”.



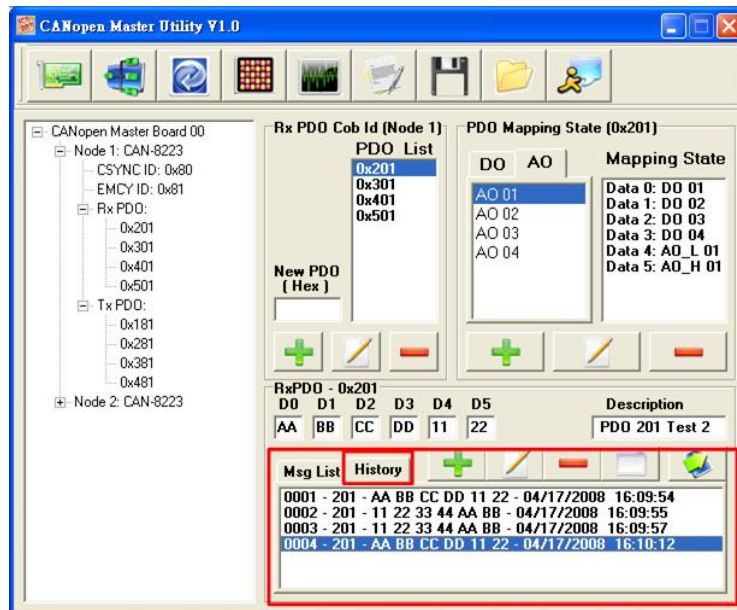
If users want to map the PDO data, they must click PDO ID label in the “PDO List” or in the “Tree List” to turn to “PDO Mapping State” page firstly. For example, there are four DO data in the 0x201 PDO ID originally. Users can select the DO/AO in the DO/AO page and click “Add” button to add the mapping data to the PDO or click “Change” button to change the original mapping data. Or click “Remove” button to delete the mapping data from the PDO that is not needed to use any more.



To send RxPDO, users can click “Add” button to add the PDO data then click “Send” to send the message or click “Send” button to send directly but not add to “Msg List”.

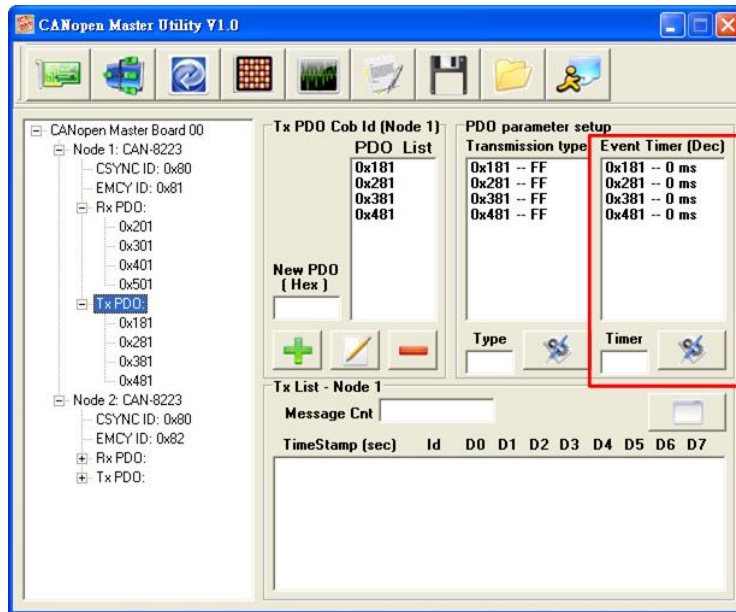


All the PDO messages that users send out would be list in the “History” page as follows. And these messages will also be listed at the “RxPDO” and “Node 1” page.

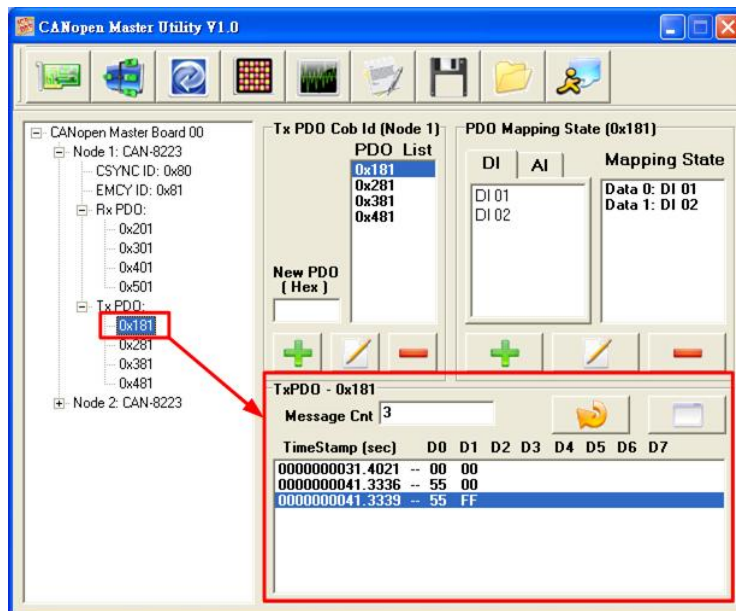


TxPDO Part:

TxPDO Part and RxPDO Part are the same at the most part. Only the “Event Timer” and “Remote PDO” are not the same. For “Event Timer”, users can select the “TxPDO” label at the tree list to open the “Event Timer” page. In this page, users can select the POD to change the event timer and click “Set” button to set it to the “Timer” value.

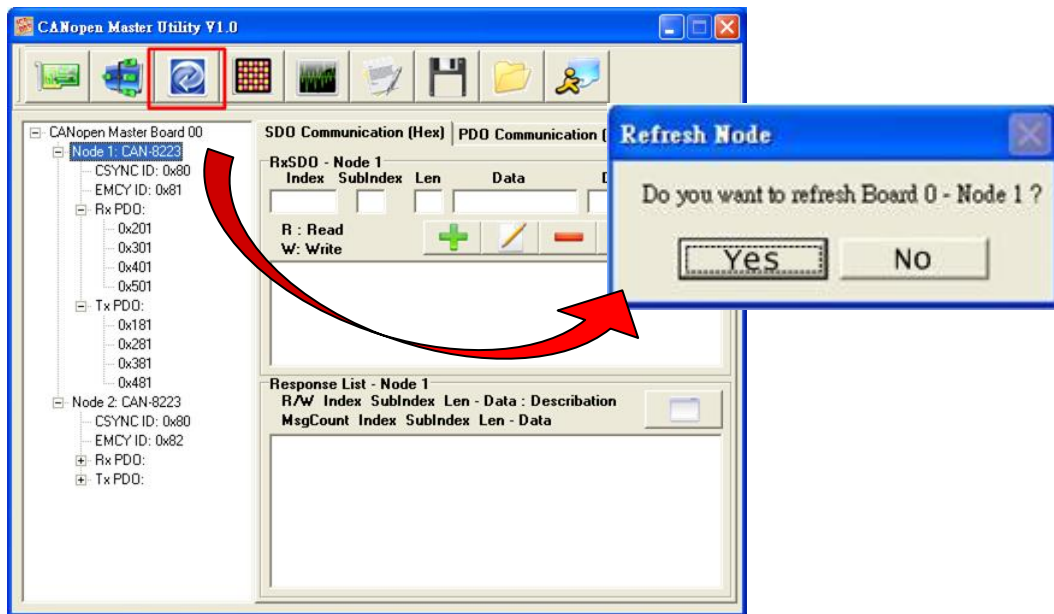


For the remote data, the function is under the PDD label page. For example “0x181” label page, click the “Remote” button to read the PDD data, the response data would be shown on the message list, and the “Message Cnt” box would let users to know how many messages have been received. And these messages would also be listed at the “TxPDO” and “Node 1” page.



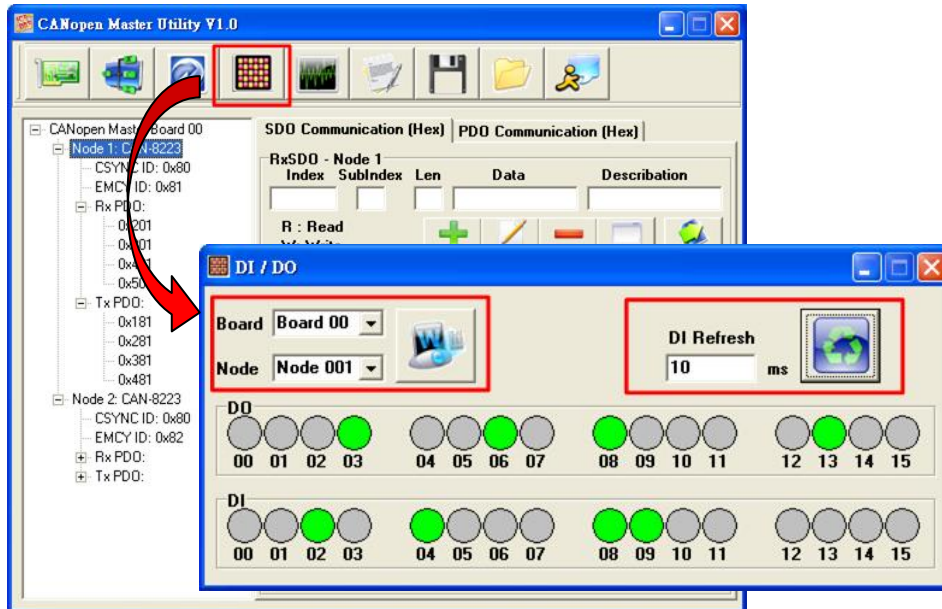
6.3.Refresh slave parameter

When some settings of the slaves on the CPM_Utility are not correctly (usually in Multi-Master to Single-Slave structure), users can select the slave on the tree list and click the “Refresh slave parameter” button to refresh the setting parameter. The CPM_Utility would get the correct setting from the slave and show on the CPM_Utility.



6.4. DI/DO control

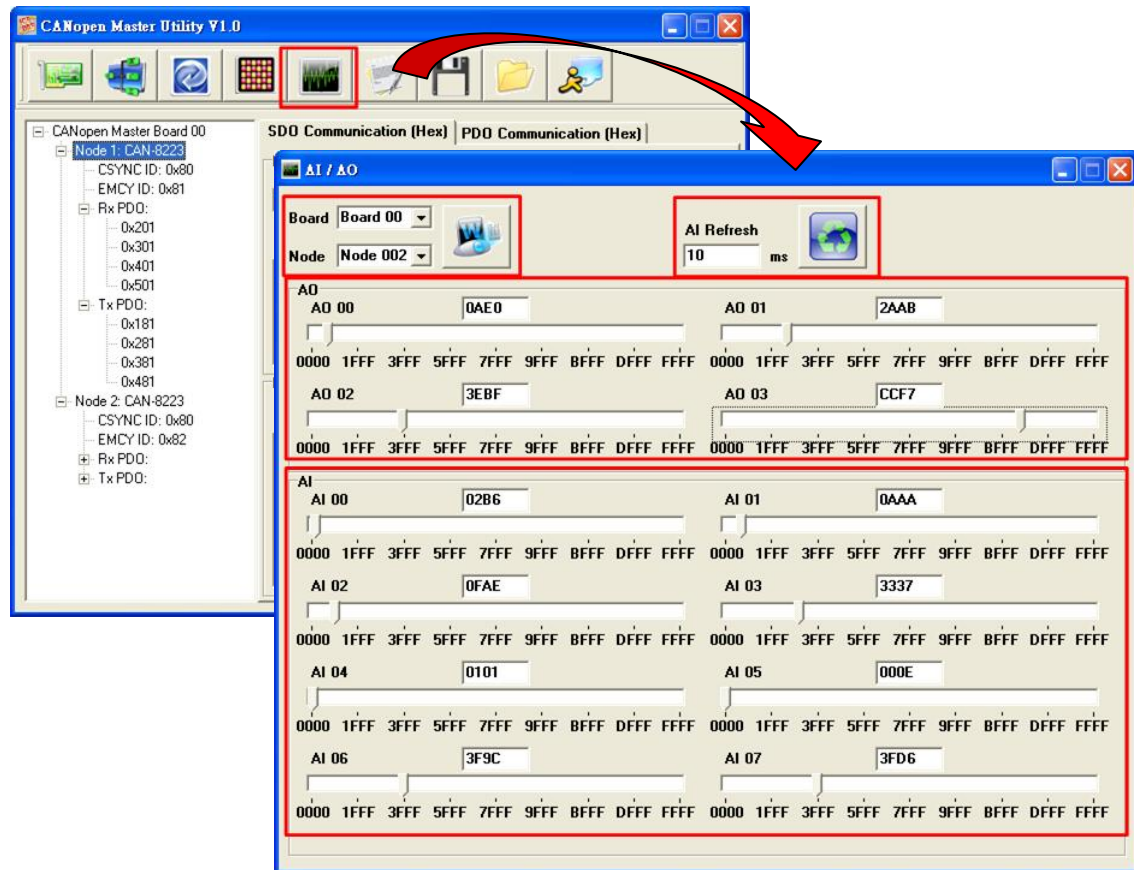
Users can use the DI/DO control to control the DI/DO directly. The DI/DO control form is shown below.



Users must select the board number and the slave firstly. Click the “Show I/O status” button to show the DI and DO control page which users select. And then the “DI Refresh” timer must be set if there are some DI data needed to be read. Also, users can click the DO LED to output DO data. Anyway, the DI data would be refreshed automatically in every DI Refresh time.

6.5. AI/AO control

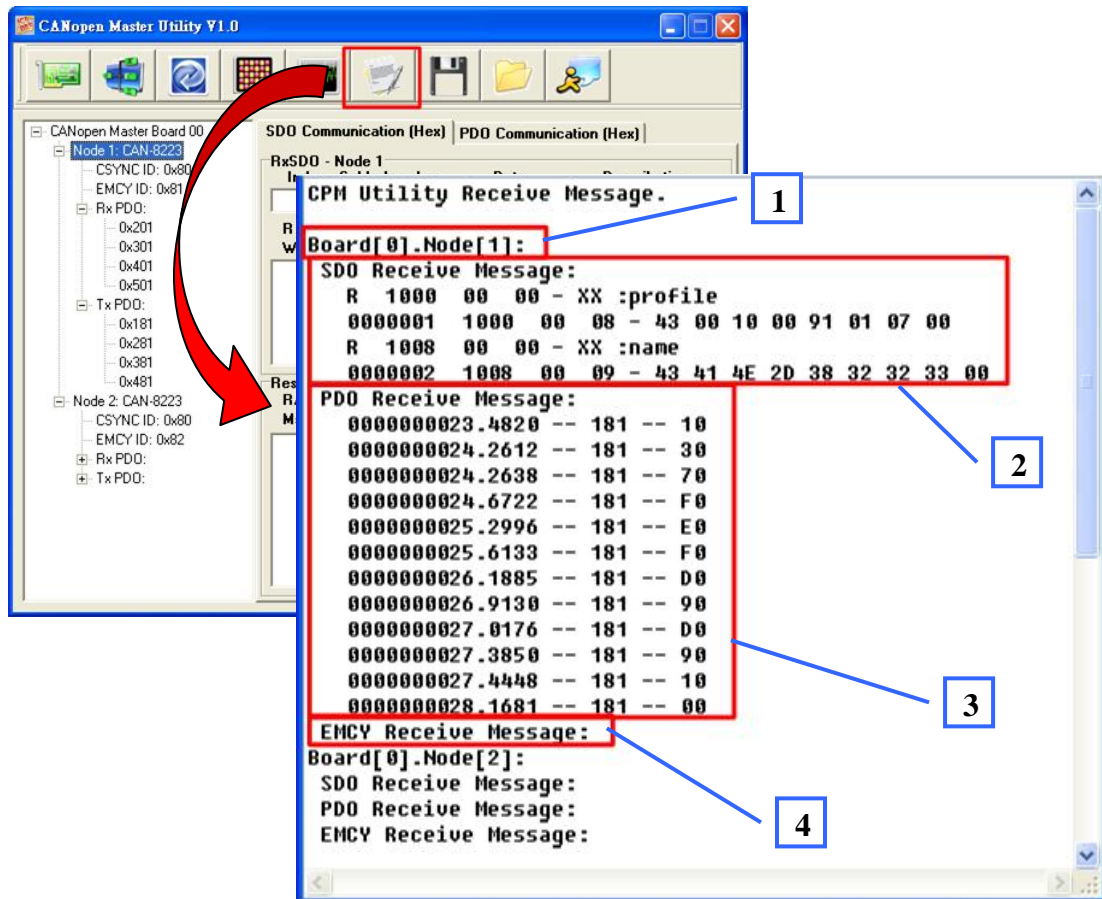
Users can use the AI/AO control to control the AI/AO directly. The AI/AO control form is shown below.



Users must select the board number and the slave firstly. Click the “Show I/O status” button to show the AI and AO control page which users select. And then the “AI Refresh” timer would be set if there are some AI data needed to be read. Also, users can drag the AO bar or key in the AO value to output AO data. Anyway, the AI data will be refreshed automatically in every “AI Refresh” time.

6.6. Save receive messages

Users can save the total CANopen messages received by the PISO-CPM100(U) by clicking the button as the following figure. These messages include SDO messages, PDO messages, and EMCY messages as the following text file.



- 1 · The following messages are received by the Node 1 of Board 0.
- 2 · The block 2 is SDO Received Messages. Every SDO message has two parts. The part 1 is sent from the board, and the part 2 is received from the board. For example, the part 1 is “R 1000 00 00 – XX: profile”. The message is meaning that “R” is the read SDO (and if “W” is write SDO), the “1000 00” is index and sub-index, “00” is data length (because this message is read message, so the data length is 0), the “XX” is meaning no data to output, and the “profile” is the user-define message for describe the SDO message. If the part 2 is “0000001 1000 00 08 – 43 00 10 00 ...”, the “0000001” is the message count, the “1000 00” is index and sub-index, “08” is data length, and the “43 00 10 ...” is the response data.



- 3 - The block 3 is the TxPDO Received Message. For example, the “0000000023.4820 – 181 -- 10” is to mean “time stamp – PDO ID -- data”.
- 4 - Block 4 is the EMCY Received Message.

6.7. Save/Load CPM_ Utility Setting

To click this “Save” button can save the total setting of the CPM_Utility to a *.cpm file. If users want to use these setting at last time, just click the “Load” button and select the cpm file to load.



6.8. About us

In about us, users can see the software version of the CPM_Utility, Copyright of our company, e-mail of our service, and the website of our homepage.

