CANUSB-I/II Intelligent CAN-bus Interface

V6.4

Contents

Chapter 1 Product Overview

1.1 Product Overview

CANUSB-I/II interface module is an intelligent CAN-bus communication interface that compatible with USB2.0 bus and supports one/two CAN channels. Using this module will enable PC to connect to CAN-bus network via USB bus, forming the CAN-bus network control nodes for the data processing and data collection for the CAN-bus networks such as bus laboratory, industrial control, intelligent residential zone, auto electronics network, and etc.

CANUSB-I/II interface comes with an electrical isolation module, which could be used to avoid the damage caused by the ground loop and enhance the system reliabilitywhen working under a tough environment.

CANUSB-I/II interface module can use CANUSB Tester software provided by us to directly finish CAN Bus configuration, message sending, and receiving.

CANUSB-I/II interface module supports **Win9X/Me、Win2000/XP 、Server 2003、 Vista** operation systems, as well as **Windows CE**. And CANUSB-I/II provides DLL dynamic library and the complete demonstration code, including **VB, VB2003, VC, C++Builder, Delphi, Labview, eMbedded Visual C++ 4.0(for Wince)**, which make it convenient for user to develop programs.

1.2 Parameters

PC interface supports USB1.1 protocol and is USB 2.0 compliant;
Max data flow 6500 fps (extend frame);
Supports CAN2.0A and CAN2.0B protocols, conforms to ISO/DIS11898 specification;
Integrates 1/2-channels CAN-bus interface, each channel can be operated independently;
Programmable CAN-bus communication Baud rates from 5Kbps to 1Mbps;
Adopts electrical isolation, the isolation voltage is : 2500Vrms;
Small size, plug and play;
Directly powered by USB port, no need for external power supply;
Operating temperature: -20 to +70 ℃;

1.3 Typical applications

CAN-bus network diagnosis and test
Auto electronic applications
Electric power communication network
Industrial control devices
High-speed and large data communications

1.4 Ordering Information

Part Number	Operating temperature	Interface
CANUSB-I	-20℃～+70℃	OPEN5
CANUSB-II	-20℃～+70℃	OPEN5

1.5 Product Sales list

[1] CANUSB-I/II interface module；
[2] USB cable；
[3] CD-ROM.(Datasheet, Drivers, Dll, CANUSB-I/II Tester software, **VB, VB2003, VC, C++Builder, Delphi, Labview, eMbedded Visual C++ 4.0(for Wince)**)

1.6 Support Information

Technical Support Mail: embededperfect@163.com
Web site: http://www.embedded-soc.com

Chapter 2 Product Installation

2.1 CAN-bus connector

CANUSB-I/II module integrates two CAN-bus channels, while CANUSB-I integrates one. The pin signal definitions see Figure 2-1 and Table 2-1.

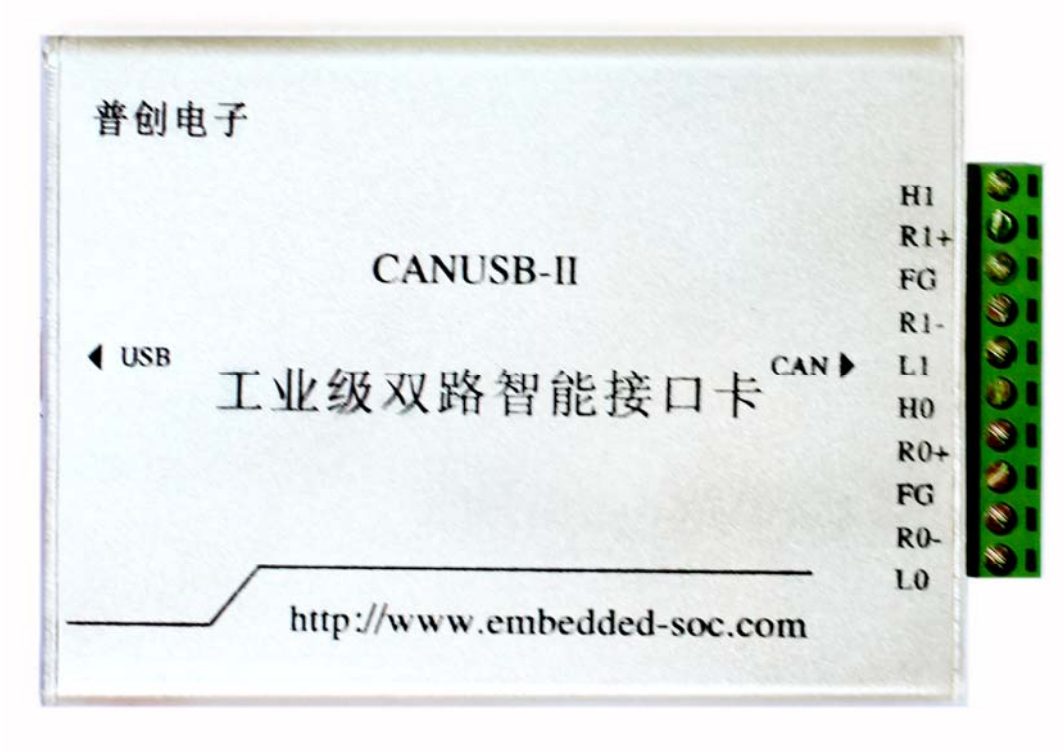

Figure 2-1 : CANUSB-I/II CAN interface module

Table 2-1 : Pin Description(Note:CANUSB-II integrates two CAN-bus channels, CANUSB-I integrates one CAN-bus channels)

Pin	Channel	Name	功能
1	CAN0	CANL0	CAN bus Signal L
2		R0－	Terminal resistor R-
3		FG	Shield cable (FG)
4		R0＋	Terminal resistor R+
5		CANH0	CAN bus Signal H
6	CAN1	CANL1	CAN bus Signal L
7		R1－	Terminal resistor R-
8		FG	Shield cable (FG)
9		R1＋	Terminal resistor R+
10		CANH1	CAN bus Signal H

2.2 CAN bus connections

To connect CANUSB-I/II module to the CAN-bus, user only need to connect CAN_L and CAN_L, CAN-bus network adopts straight-line topology, and two terminal 120Ω resistances need to be installed on the two bus terminals. If the number of nodes larger than 2, the 120Ω resistance is not necessary to be installed on the middle node. The length of branch connection should not be longer than 3 meters. The connections for the CAN-bus are shown in Figure 2-2.

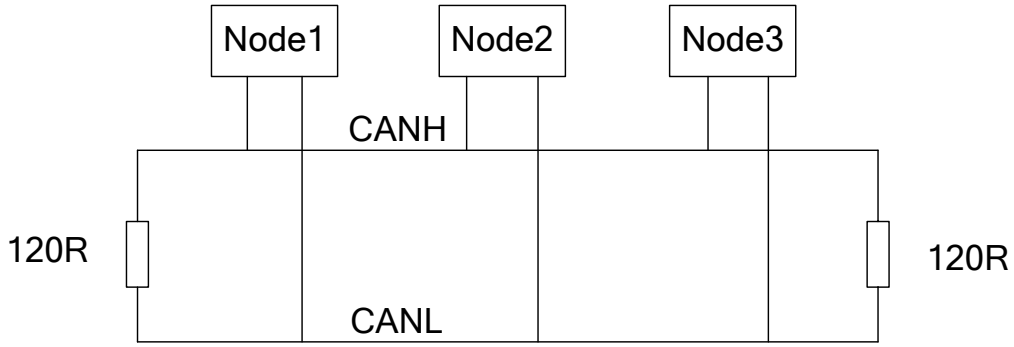

Figure 2-2: The topology for CAN-bus Network

Chapter 3 Software development

If users intend to make a program for their own application, they need to read following descriptions very carefully, and refer the demo source code.

Develop files include CAN_TO_USB.h，CAN_TO_USB.lib（For VC），CAN_TO_USBbc.lib(For BC), SiUSBXp.dll，CAN_TO_USB.dll.

We provides examples for **VB, VB2003, VC, C++Builder, Delphi, Labview, eMbedded Visual C++ 4.0(for Wince)**, which make it convenient for user to develop programs.

3.1 Data Structure of Library

3.1.1 VCI_INIT_CONFIG

```
typedef struct _INIT_CONFIG{
    DWORD AccCode;
    DWORD AccMask;
    DWORD Reserved;
    UCHAR  Filter;
    UCHAR  Timing0;
    UCHAR  Timing1;
    UCHAR  Mode;
}VCI_INIT_CONFIG,*P_VCI_INIT_CONFIG;
```

AccCode acceptance code for filter

AccMask mask code for filter

Reserved not used

Filter filter mode ,single or double

Timing0 timer0 (BTR0)

Timing1 timer1 (BTR1）

Mode work mode 0: normal work, 1: listen only

Timing0 and remark Timing1 is used for setting CAN baud rate. The following table is about setting of 15 kinds of common baud rates

CAN Baud rate	Timer0	Timer1
5Kbps	0xBF	0xFF
10Kbps	0x31	0x1C
20Kbps	0x18	0x1C
40Kbps	0x87	0xFF
50Kbps	0x09	0x1C
80Kbps	0x83	0Xff
100Kbps	0x04	0x1C
125Kbps	0x03	0x1C
200Kbps	0x81	0xFA

250Kbps	0x01	0x1C
400Kbps	0x80	0xFA
500Kbps	0x00	0x1C
666Kbps	0x80	0xB6
800Kbps	0x00	0x16
1000Kbps	0x00	0x14

3.1.2 VCI_CAN_OBJ

```
typedef  struct  _VCI_CAN_OBJ{
    DWORD ID;
    BYTE    SendType;
    BYTE    ExternFlag;
    BYTE    RemoteFlag;
    BYTE    DataLen;
    BYTE    Data[8];
}VCI_CAN_OBJ,*P_VCI_CAN_OBJ;
```

ID packet ID, 4 bytes

SendType 0: Normal Send,1 :self reception

RemoteFlag remote frame or not

ExternFlag extended frame or not

DataLen data length(<=8)，it is the length of data

Data data of packet

3.2 Function description

[1] Open the device.

BOOL __stdcall VCI_OpenDevice(DWORD DevIndex);

DevIndex Device index.0: The first device.1: The second device.

Return value 1: Success, 0: Fail

[2] Close the device

BOOL __stdcall VCI_CloseDevice(DWORD DevIndex);

DevIndex Device index.0: The first device.1: The second device.

Return value 1: Success, 0: Fail

[3] Initialize can.

BOOL _stdcall VCI_InitCAN(DWORD DevIndex,DWORD CANIndex, P_VCI_INIT_CONFIG InitConfig);

DevIndex Device index.0: The first device.1: The second device.

CANIndex Can channel.0: The first channel.1: The second channel

InitConfig Init parameters structure.

InitConfig->AccCode	AccCode corresponds to four registers in SJA1000.
InitConfig->AccMask	ACR0=((InitConfig->AccCode)>>24)&0xFF AMR0=((InitConfig->AccMask)>>24)&0xFF
InitConfig->Reserved	reserved
InitConfig->Filter	Filter mode，0- single filter，1-dual filter
InitConfig-> timer0	Baud rate timer 0
InitConfig-> timer1	Baud rate timer 1
InitConfig->Mode	0:Normal mode.1:Listen only

Return value 1: Success, 0: Fail

[4] Reset can.

BOOL __stdcall VCI_ResetCAN(DWORD DevIndex ， DWORD CANIndex);

DevIndex Device index.0: The first device.1: The second device.

CANIndex Can channel.0: The first channel.1: The second channel

Return value 1: Success, 0: Fail

[5] Send can packet.

BOOL __stdcall VCI_Transmit(DWORD DevIndex ， DWORD CANIndex, P_VCI_CAN_OBJ *pSend);

DevIndex Device index.0: The first device.1: The second device.

CANIndex Can channel.0: The first channel.1: The second channel

pSend Packet to Send.

Return value 1: Success, 0: Fail

[6] Receive can packet.

DWORD __stdcall VCI_Receive(DWORD DevIndex ， DWORD CANIndex，PVCI_CAN_OBJ pReceive , DWORD Len , DWORD WaitTime);

DevIndex Device index.0: The first device.1: The second device.

CANIndex Can channel.0: The first channel.1: The second channel

pReceive Receive packet.

Len Number of packets you need to read.

WaitTime Time out.

Return value Actually packets number returned。

[7] Get the number packet in the internal buffer

DWORD __stdcall VCI_GetReceiveNum(DWORD DevIndex,DWORD CANIndex);

DevIndex Device index.0: The first device.1: The second device.

CANIndex Can channel.0: The first channel.1: The second channel

Return value number of the packets in the internal buffer

[8] Clear the internal buffer.

BOOL __stdcall VCI_ClearBuffer(DWORD DevIndex,DWORD CANIndex);

DevIndex Device index.0: The first device.1: The second device.

CANIndex Can channel.0: The first channel.1: The second channel

Return value 1: Success, 0: Fail

[9] Read the serial number of the device.

BOOL _stdcall VCI_ReadDevSn(DWORD DevIndex, PCHAR DevSn);

DevIndex Device index.0: The first device.1: The second device.

DevSn Serial Number of the device

Return value 1: Success, 0: Fail

3.4 Interface library function using flow