Janus-MM Basic CAN Driver
For Linux 2.6.xx and Windows XP

User Manual

Revision A

Revision	Date	Comments
A	06/16/2011	Initial Version

FOR TECHNICAL SUPPORT
PLEASE CONTACT:

support@diamondsystems.com

Diamond Systems Corporation
555 Ellis Street
Mountain View, CA 94043 USA
Tel 1-650-810-2500
Fax 1-650-810-2525
www.diamondsystems.com

Table of Contents

1 Introduction 3

2 Scope 3

3 Setting up the hardware 4

3.1 Hardware Jumper settings 4

3.1.1 The CAN Termination, Slew Rate and Power Supply Selection 4

3.2 The CAN IRQ Selection 5

3.2.1 The CAN Base Address Selection 5

3.3 Loop Back Cable Setup 6

3.4 CAN Analyzer Setup (For Debugging) 6

4 Linux 2.6.xx Driver - Functions Exported 7

4.1 open() 8

4.2 close() 9

4.3 IOCTL – Write CAN Frame 10

4.4 IOCTL – Read CAN Frames 11

4.5 IOCTL – Set CAN Baud Rate 12

5 Linux Driver Installation and Running the application 13

5.1 Compiling and loading the driver 13

5.2 Compiling and running the Application 13

5.3 Stopping the application and unloading 15

5.4 Linux Driver Application examples 16

6 Windows XP Driver - Functions exported 17

6.1 OpenDevice () 18

6.2 CloseHandle () 19

6.3 IOCTL – Write CAN Frame 20

6.4 IOCTL – Read CAN Frames 21

6.5 IOCTL – Set CAN Baud Rate 22

6.6 IOCTL – Configure CAN base address 23

7 Windows Driver Installation and Running the application 24

7.1 Driver Installation 24

7.2 Driver Un-Installation 26

7.3 Driver Sample Application Usage 26

1 Introduction

This document provides complete instructions on using the device drivers for exercising the CAN bus on the Janus-MM CAN ports. The Janus-MM board has two CAN ports which are CAN 2.0 compatible using the SJA1000 CAN bus controllers.

In order to communicate with the CAN controllers, Diamond Systems provides a set of freely usable basic device drivers on Linux 2.6.xx and Windows XP operating systems so that the board can be quickly evaluated and a user can develop applications with the driver on the OS of choice.

This document is meant for…
1. Application developers who want to use the CAN driver can use this document to understand the driver calls and use in their application.
2. Test engineers who want to test the driver and application can use this document to set up the hardware test environment, connect a snooper, install driver and run the application.

2 Scope

The device driver distributed has the following specifications.

- Available on Linux kernel version 2.6.23 and Windows XP OS.
- Low level driver that can perform open, close, read, write on the device.
- Ability to change the baud rate of the CAN communication up to 1Mbps baud rate.
- Handle interrupt request on both the IRQs for each CAN port on the board.

3 Setting up the hardware

3.1 Hardware Jumper settings

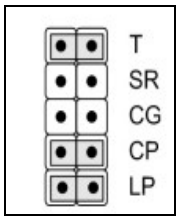
The jumper settings have to be made according to the configuration as mentioned in the Janus-MM user manual. It is required that the hardware base address and the IRQ settings used for the operation is available in the SBC that is being used to run the Janus-MM board. When using Athena-II SBC, please restore the Athena-II BIOS to default settings.

The Sample jumper settings are described in the sections below when Janus-MM is used with Athena-II SBC.

3.1.1 The CAN Termination, Slew Rate and Power Supply Selection

Jumpers J10 and J11 provide CAN termination, slew rate and power supply selection for both CAN ports; J10 is used to configure CAN port A and J11 is used to configure CAN port B.

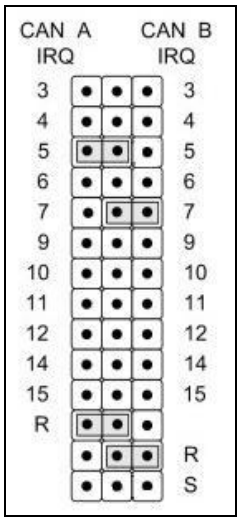
The following diagram shows the jumper pin layout and the recommended jumper setting.



J10 and J11 Jumpers with Default settings

3.2 The CAN IRQ Selection

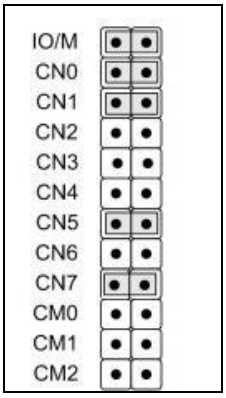
Use jumper J5 to specify the IRQ for both CAN ports. The following diagram shows the jumper pin layout. The IRQ jumper setting on the board must match the software settings in the driver configuration or command line arguments when running the applications as described in the later sections of this document.



J5 Jumpers with Default Settings (IRQ 5 for CAN A and IRQ7 for CAN B)

3.2.1 The CAN Base Address Selection

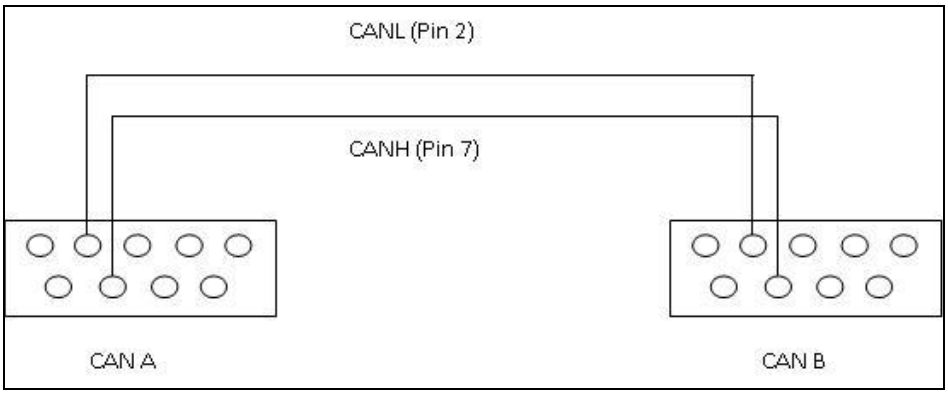
Use pin sets CN0-CN7 of jumper J4 to set the CAN base address using Memory address spaces. The diagram below shows the jumper setting for Memory address 0xD7000 for CANA. The port CANB automatically is offset by 0x200 so the address for CANB would be 0xD7200. Please refer to the Janus-MM user manual for more information on the jumper settings and board configuration.



J4 Jumper setting for using memory address of 0xD7000

3.3 Loop Back Cable Setup

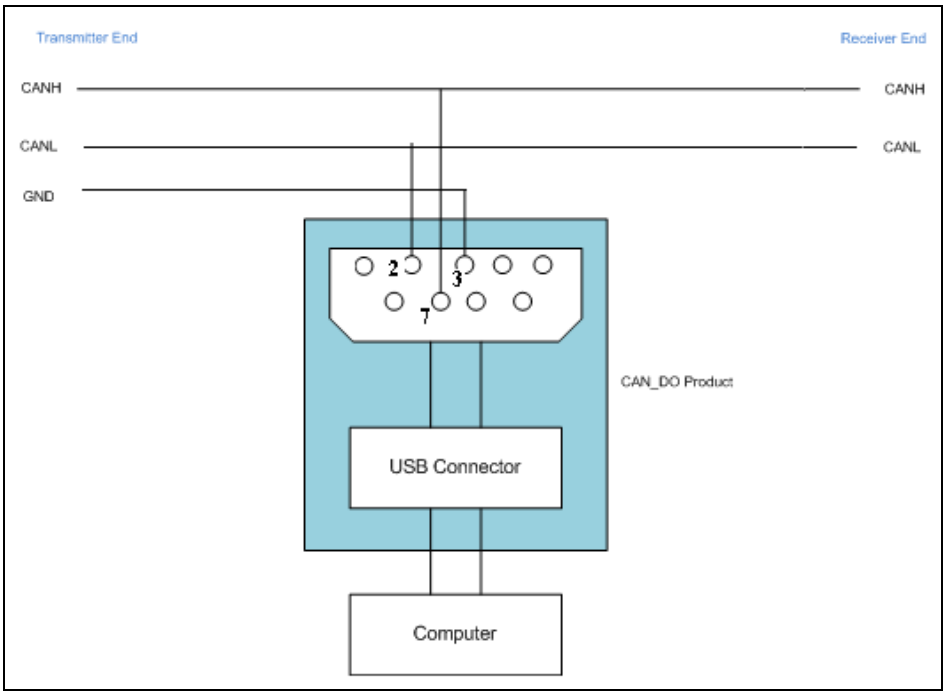
The CAN H and CAN L lines should be connected as below for stand-along testing on a Janus-MM CANA and CANB ports. In this configuration, the data sent on one port will be received by the other port to verify the connectivity as well as the operation of the driver.



3.4 CAN Analyzer Setup (For Debugging)

To debug the connectivity, a CAN analyzer can be used to tap into the CAN connections of the Janus MM CAN ports. There are two CAN ports, CANA and CANB, available in the JANUSMM board. These ports are looped-back for testing. The CAN analyzer tapping the CAN packet is also shown in the picture below. Using this connectivity, the CAN traffic going between the two ports can be monitored on the CAN analyzer.

NOTE: This is provided as a suggestion and Diamond Systems does not recommend any specific CAN tools. It is up to you to use any available CAN hardware which are CAN 2.0 compatible.



4 Linux 2.6.xx Driver - Functions Exported

This section discusses all the functions exported by the Linux driver, their prototypes, their usage and an example segment of code. The Linux driver is supported on kernel version 2.6.xx and it is recommended to use the kernel version 2.6.23.

The following table provides a list of functions exported by the Linux driver.

<table>
<tr><th>Function Name</th><th colspan="2">Description</th></tr>
<tr><td>open ()</td><td colspan="2">This function is used to open the CAN device and get the handle for that device. The handle returned by this function MUST be used in all sub-sequent calls in the application.</td></tr>
<tr><td>close ()</td><td colspan="2">This function is used to release the device and to remove the handle returned by the call to OpenDevice function.</td></tr>
<tr><td rowspan="4">IOCTL</td><td colspan="2">For all other functions, the driver provides a set of IOCTL commands to perform various functions on the driver like read from CAN bus, write to CAN bus, configure memory address and set baud rate. All the various IOCTL commands should be executed via the “ioctl” function which is prototyped in “sys/ioctl.h”.</td></tr>
<tr><td>IOCTL CODE</td><td>Operation</td></tr>
<tr><td>SJA1000_IOCTRANS</td><td>Transmit ioctl request code. This IOCTL code should be used for transmitting data on the CAN bus.</td></tr>
<tr><td>SJA1000_IOCRECV</td><td>Receive ioctl request code. This IOCTL should be used for reading data from the CAN bus.</td></tr>
<tr><td>SJA1000_IOCBTR</td><td>Baud-rate ioctl request code. This IOCTL should be used to set the baud rate of the CAN bus port.</td></tr>
</table>

4.1 open()

DESCRIPTION

This function is used to open the CAN device and get the handle for that device.

PROTOTYPE

int open(char *Devicename, int flag)

RETURN VALUE

Returns nonnegative value on successful execution and it will act as handler to access the device. Negative value means failure to open the device.

PARAMETERS

Devicename

The device name to open the device. This file name should be any of the following
dev/CANA, /dev/CANB

flag

O_RDWR

appDevHandle

This parameter is the handle for the device.

REMARKS

This function should be called before calling any other driver functions. This function will give the handle for the CAN device.

EXAMPLE

```
#include "can.h"
#include "sja1000_ioctl.h"
……
int appDevHandle;
char appDeviceName[15] = /dev/CANA;

appDevHandle = open(appDeviceName , O_RDWR);
if(appDevHandle < 0) {
        printf("Device Open Error");
        exit(0);
}
……
```

4.2 close()

DESCRIPTION

This function is used to release the device and to remove the handle.

PROTOTYPE

int close(int appDevHandle);

RETURN VALUE

Returns nonnegative value on successful execution.

PARAMETERS

appDevHandle

This parameter is the handle for the device.

REMARKS

This should be the last function call in the application.

EXAMPLE

```
#include "can.h"
#include "sja1000_ioctl.h"
……
int appDevHandle;

appRetVal = close(appDevHandle);
if(appRetVal == 0) {
    printf("Device Closed\n");
}
……
```

4.3 IOCTL – Write CAN Frame

DESCRIPTION

This function is used to transmit the CAN frames to the hardware.

PROTOTYPE

```
#include <sys/ioctl.h>
int ioctl(int appDevHandle, int request, unsigned long *in_frame);
```

RETURN VALUE

On success zero is returned. On error, -1 is returned, and errno is set appropriately.

PARAMETERS

appDevHandle
This parameter is the handle for the device.

request
A long word integer that specifies the Transmit ioctl request code (SJA1000_IOCTRANS)

in_frame
The CAN Frame that has to be transmitted.

EXAMPLE

```
#include "can.h"
#include "sja1000_ioctl.h"
……
int appDevHandle;
struct can_frame frame;
……

// Data is Transmitted as can frames

appWRetVal = ioctl(appDevHandle, SJA1000_IOCTRANS,
(unsigned long)&frame);
……
```

4.4 IOCTL – Read CAN Frames

DESCRIPTION

This function is used by the driver to receive the CAN frames from the hardware when data is available.

PROTOTYPE

```
#include <sys/ioctl.h>
int ioctl(int appDevHandle, int request, unsigned long
*out_frame);
```

RETURN VALUE

Return count of the data received. On error, -1 is returned, and errno is set appropriately.

PARAMETERS

appDevHandle
This parameter is the handle for the device.

request
A long word integer that specifies the Receive ioctl request code (SJA1000_IOCRECV)

out_frame
The received CAN Frame.

EXAMPLE

```
#include "can.h"
#include "sja1000_ioctl.h"
……
int appDevHandle;
unsigned long data[10];
……

// Data is received as unsigned long type // array which can be
packed back as CAN frame

appWRetVal = ioctl(appDevHandle, SJA1000_IOCRECV,
(unsigned long) data);
……
```

4.5 IOCTL – Set CAN Baud Rate

DESCRIPTION

This function is used to set the bitrate for the specified device.

PROTOTYPE

```
#include <sys/ioctl.h>
int ioctl(int appDevHandle, int request, unsigned long *in_btr);
```

RETURN VALUE

On success zero is returned. On error, -1 is returned, and errno is set appropriately.

PARAMETERS

appDevHandle

This parameter is the handle for the device.

request

A long word integer that specifies the Bit-timing ioctl request code (SJA1000_IOCBTR)

in_btr

Bitrate for the specified CAN device.

EXAMPLE

```
#include "can.h"
#include "sja1000_ioctl.h"
……
int appDevHandle;
struct can_btr btr;
……

// Bitrate value ranges upto 1000000

appWRetVal = ioctl(appDevHandle, SJA1000_IOCBTR,
(unsigned long)&btr);
……
```

5 Linux Driver Installation and Running the application

This section discusses how to install the driver and run the application.

Untar can_beta0.2.tar.bz2 with the following command,

```
# tar xfvj can_beta0.2.tar.bz2
```

5.1 Compiling and loading the driver

From the release directory issue change directory to enter into the driver folder and issue make command,

```
# cd driver
# make
```

Run install.sh to load the kernel,

```
# ./install.sh
```

Current install.sh is a shell script that uses the following configuration,

canA : Address = 0xD7000 and IRQ = 5

canB : Address = 0xD7200 and IRQ = 7

This can be edited to use different address and IRQ number for both CAN devices.

5.2 Compiling and running the Application

From the release directory issue change directory to enter into the application folder,

```
# cd application
```

Issue the following command. This will create three binaries for transmitting, receiving and setting bitrate.

```
# ./compile.sh
```

Run the following command for setting bitrate.

```
# ./bitrate –h
```

Usage: ./bitrate [options] [<CAN-Device-Name>]

Options:

-b <bitrate>	: bit-rate in bits/sec
-s <samp_pt>	: sample-point in one-tenth of a percent or 0 for CIA recommended sample points
-c <clock>	: real CAN system clock in Hz
-h	: help

Ex 1: configure bitrate for both canA and canB

```
# ./bitrate –b 1000000
```

Ex 2: configure biterate for any one device

```
# ./bitrate –b 500000 canA
```

Open two new terminals, and run `canrecv` and `cansend` from different terminals. `Canrecv` has to be run first. It will wait for data.

```
# ./canrecv canB
```

`Cansend` has to be run next, the command is as shown below.

```
# ./cansend canA
```

The help menu for `canrecv` and `cansend` is shown below:

a. cansend

```
#  ./cansend –help
```

Usage: `cansend [<can-interface>] [Options] <can-msg>`

`<can-msg>` can consist of up to 8 bytes given as a space separated list

Options:

-i, --identifier=ID	CAN Identifier (default = 1)
-r --rtr	send remote request
-e --extended	send extended frame
-l	send message infinite times
--loop=COUNT	send message COUNT times
-p --pattern	send message data in incremental pattern
-o <filename>	Input filename
-v, --version	be verbose
-h, --help	this help

Ex 1: Transmit CAN frames

```
# ./cansend canA 0x1 0x2 0x3 0x4 0x5 0x6 0x7 0x8 –l --pattern
```

Ex 2: Transmit a file continuously

```
# ./cansend canA –o Transmit.txt
```

b. canrecv

```
#  ./canrecv –help
```

Usage: `canrecv [<can-interface>] [Options]`

Options:

-o <filename>	Output filename
-h, --help	this help

EX 1: Receive CAN Frames

```
# ./canrecv canB
```

EX 2: Receive Files and store the file in incremental order (1recv.txt, recv.txt, 3recv.txt,…….,)

```
# ./canrecv canB –o recv.txt
```

Note: For 10 minutes the application can send 37 files each of size 1024 bytes.

5.3 Stopping the application and unloading

The application can be stopped by pressing Ctrl+C.

The driver is unloaded using the following command.

```
# ./uninstall.sh
```

5.4 Linux Driver Application examples

Example 1:

The delivery package can be tested to transmit data patterns as follows,

a) Open a command shell and type the following.

```
# ./canrecv canB > rx.txt
```

The application will be waiting for data from ‘cansend’ application.

b) Open another command shell and type the following. The ‘cansend’ application will start sending the data.

```
# ./cansend canA 0x1 0x2 0x3 0x4 0x5 0x6 0x7 > tx.txt
```

c) Leave the applications running for few hours. First stop the ‘cansend’ application by pressing Ctrl-C. Then stop ‘canrecv’.

d) Now you have the log files generated at both the ends. And they can be compared using the linux command ‘diff’ as follows,

```
# diff tx.txt rx.txt
```

Example 2:

The delivery package can be tested to transmit file as follows,

a) Open a command shell and type the following.

```
# ./canrecv canB -o rx.txt
```

The application will be waiting for file from ‘cansend’ application.

b) Open another command shell and type one of the following command. The ‘cansend’ application will start sending the file.

```
# ./cansend canA -o tx.txt --loop=0x5  //will send the file for 5 times
# ./cansend canA -o tx.txt -l          //will send the file continuously
# ./cansend canA -o tx.txt             //will send the file once
```

c) Leave the applications running for few hours. First stop the ‘cansend’ application by pressing Ctrl-C. Then stop ‘canrecv’.

d) Now you have the multiple instances of the transmitted file generated by the receiving node as *1rx.txt*, *2rx.txt*, *3rx.txt* and so on. And they can be compared using the Linux command ‘diff’.

6 Windows XP Driver - Functions exported

This section discusses all the functions exported by the Windows driver, their prototypes, their usage and an example segment of code.

The following table lists the functions supported by the driver.

Function Name	Description	
OpenDevice ()	This function is used to open the CAN device and get the handle for that device. The handle returned by this function MUST be used in all sub-sequent calls in the application.	
CloseHandle ()	This function is used to release the device and to remove the handle returned by the call to OpenDevice function.	
IOCTL	For all other functions, the driver provides a set of IOCTL commands to perform various functions on the driver like read from CAN bus, write to CAN bus, configure memory address and set baud rate. All the various IOCTL commands should be executed via the DeviceIoControl function.	
	IOCTL CODE	**Operation**
	IOCTL_WRITE	This macro specifies the Transmit ioctl request code. This IOCTL code should be used for transmitting data on the CAN bus.
	IOCTL_READ	This macro specifies the Receive ioctl request code. This IOCTL should be used for reading data from the CAN bus.
	IOCTL_SET_BAUD_RATE	This macro specifies the Baud-rate ioctl request code. This IOCTL should be used to set the baud rate of the CAN bus port.
	IOCTL_SET_MEM	This macro specifies the memory base address configuration request code. The value passed when setting the base address should match the base address jumper configuration in the hardware.

The section below provides a list of functions exported by the Windows driver.

6.1 OpenDevice ()

DESCRIPTION

This function is used to open the CAN device and get the handle for that device.

PROTOTYPE

OpenDevice(IN CONST GUID * InterfaceGuid,
IN ULONG FileFlagOptions)

RETURN VALUE

Returns Handle value.

PARAMETERS

InterfaceGuid

The device interface GUID to be opens the port. This GUID name should be unique and declared in public.h file

FileFlagOptions

This parameter should be set to FILE_FLAG_OVERLAPPED.

REMARKS

This function should be called before calling any other driver functions. This function will give the handle for the CAN device.

EXAMPLE

```
#include “public.h"
……
HANDLE      hISAdevice;

hISAdevice = OpenDevice(
        &GUID_DEVINTERFACE_CAN,
        FILE_FLAG_OVERLAPPED );
```

6.2 CloseHandle ()

DESCRIPTION This function is used to release the device and to remove the handle.

PROTOTYPE CloseHandle(hISAdevice);

RETURN VALUE If the function succeeds, the return value is nonzero.

PARAMETERS

DevHandle
This parameter is the handle for the device obtained from a previous call to OpenDevice function.

REMARKS This should be the last function call in the application.

EXAMPLE

```
#include “public.h"
……
HANDLE      hISAdevice;

hISAdevice = CloseHandle(hISAdevice);
```

6.3 IOCTL – Write CAN Frame

DESCRIPTION This function is used to transmit the CAN frames to the hardware.

PROTOTYPE

```
#include <winioctl.h>

okay = DeviceIoControl( hISAdevice,
        IOCTL_WRITE, outbuf, outsize,
        inbuf, insize, &numread, lpOverlapped);
```

RETURN VALUE If the operation completes successfully, the return Value is non zero.

PARAMETERS

hISAdevice
This parameter is the handle for the device.

IOCTL_WRITE
This macro specifies the Transmit ioctl request code

Outbuf
The CAN Frame that has to be transmitted.

Ex: outbuf[0] =FrameFormat
outbuf[1] =Identifier
outbuf[n]= data.

Outsize
The size of data to be transmitted.

Inbuf
Not needed.

Insize
Not needed.

lpBytesReturned
Actual data size. (not needed.)

lpOverlapped
This should be always Null.

EXAMPLE

```
#include "public.h"
……
HANDLE      hISAdevice;
……
okay = DeviceIoControl( hISAdevice,
            IOCTL_WRITE, outbuf,
                outcount,inbuf,insize,
                &numread,NULL);
```

6.4 IOCTL – Read CAN Frames

DESCRIPTION This function is used to receive the CAN frames from the hardware.

PROTOTYPE

```
#include <winioctl.h>

okay = DeviceIoControl( hISAdevice,
        IOCTL_READ, outbuf, outsize,
        inbuf, insize, &numread, lpOverlapped);
```

RETURN VALUE If the operation completes successfully, the return Value is non zero.

PARAMETERS

hISAdevice
This parameter is the handle for the device.

IOCTL_READ
This macro specifies the Receive ioctl request code

Outbuf
Not needed

Outsize
Not needed.

Inbuf
The structure contains the frame datas in the following format,

Fi -> frame format
iVer->identifier
rData-> CAN Receive data.

Insize
Size of inbuf.

lpBytesReturned
Actual data size.

lpOverlapped
This should be always Null.

EXAMPLE

```
#include "public.h"
……
HANDLE      hISAdevice;
……
okay = DeviceIoControl( hISAdevice,
        IOCTL_READ, NULL,
                0,inbuf,insize,
                &numread,NULL);
```

6.5 IOCTL – Set CAN Baud Rate

DESCRIPTION This function is used to transmit the baud rate value to the driver

PROTOTYPE

```
#include <winioctl.h>

okay = DeviceIoControl( hISAdevice,
        IOCTL_SET_BAUD_RATE, outbuf,   outsize,inbuf,
insize,
 &numread, lpOverlapped);
```

RETURN VALUE If the operation completes successfully, the return Value is non zero.

PARAMETERS

hISAdevice
This parameter is the handle for the device.

IOCTL_SET_BAUD_RATE
This macro specifies the Baud-rate ioctl request code

Outbuf
Bit rate value.

Outsize
The size of bit rate value.

Inbuf
Not needed.

Insize
Not needed.

lpBytesReturned
Actual data size. (not needed.)

lpOverlapped
This should be always Null.

EXAMPLE

```
#include "public.h"
……
HANDLE     hISAdevice;
……
okay = DeviceIoControl( hISAdevice,
IOCTL_SET_BAUD_RATE, outbuf,
                    outcount,inbuf,insize,
                    &numread,NULL);
```

6.6 IOCTL – Configure CAN base address

DESCRIPTION This function is used to configure address of the CAN driver

PROTOTYPE

```
#include <winioctl.h>

okay = DeviceIoControl( hISAdevice,
        IOCTL_SET_MEM, outbuf, outsize,
        inbuf, insize, &numread, lpOverlapped);
```

RETURN VALUE If the operation completes successfully, the return Value is non zero.

PARAMETERS

hISAdevice
This parameter is the handle for the device.

IOCTL_SET_MEM
This macro specifies the memory address configuration request code

Outbuf
Memory address value.

Outsize
The size of memory address variable.

Inbuf
Not needed.

Insize
Not needed.

lpBytesReturned
Actual data size. (not needed.)

lpOverlapped
This should be always Null.

EXAMPLE

```
#include "public.h"
……
HANDLE      hISAdevice;
……
okay = DeviceIoControl( hISAdevice,
IOCTL_SET_MEM, outbuf,
                outcount,inbuf,insize,
                &numread,NULL);
```

7 Windows Driver Installation and Running the application

This section discusses how to install the driver on Windows XP OS and run the sample application(s) called “canSend” and “canRecv”.

The Windows driver is available in binary format and is distributed as the following files.

Filename	Description
SJA1000_CAN.INF	INF File for usage when the device is detected by windows
SJA1000_CAN.SYS	The CAN driver (installs in C:\Windows\System32\Drivers folder)
INSTALL.BAT	Driver installation batch file.
REMOVE.BAT	Driver un-installation batch file.
DEVCON.EXE	Alternative to Device Manager. Used by INSTALL.BAT and REMOVE.BAT
WDFCOINSTALLER01009.DLL	Support DLL required along with the INF file.

NOTE: The driver is supported on Windows XP OS only and will not work with Windows 7.

7.1 Driver Installation

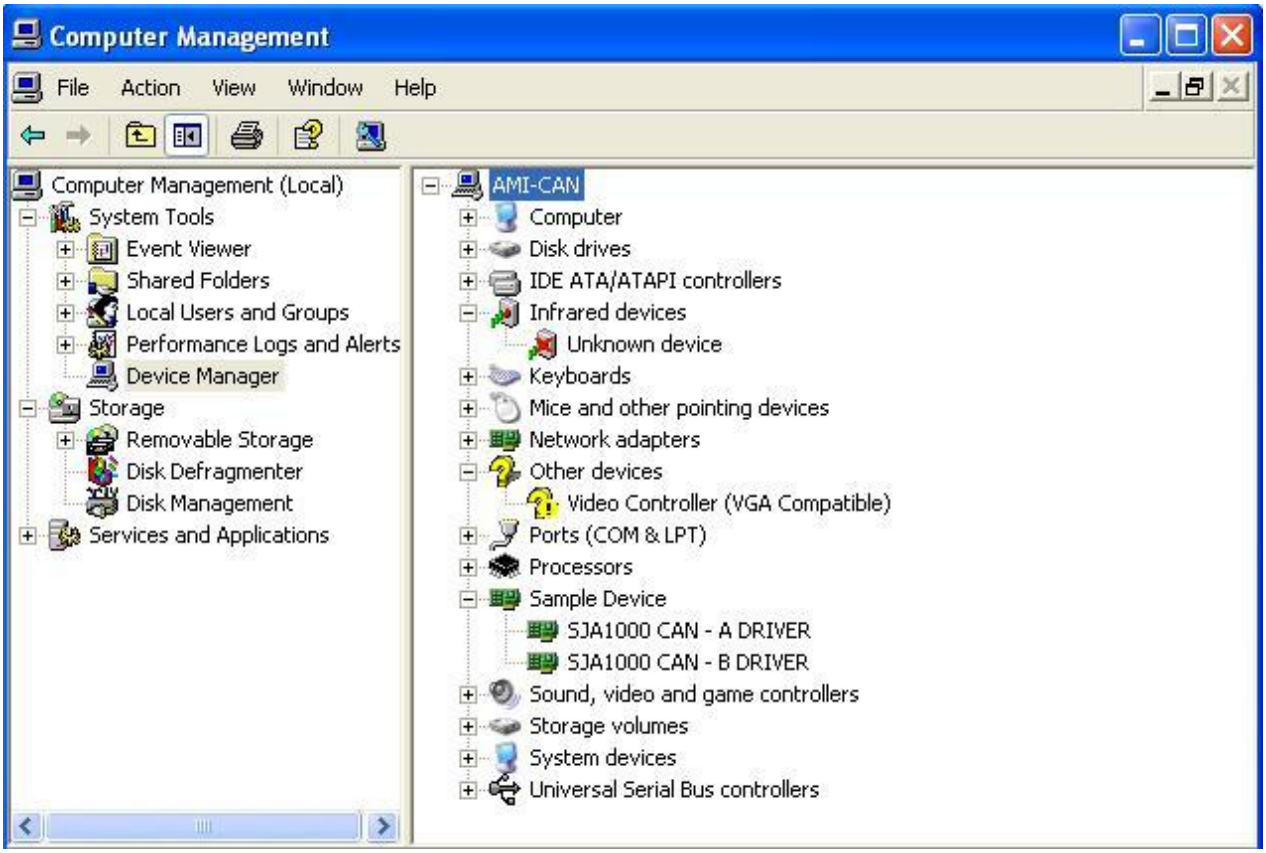
a. Double-click the **“install.bat”** file to install the driver.

b. Check for the driver entries (**canSend** & **canRecv)** in **“Control Panel -> system-> Hardware -> Device Manager -> Sample Device”.**

c. To configure the IRQ value, change the “IRQConfig” value in .inf file for both controllers as follows,

[S5933DK1_LogConfig]
IRQConfig=5

[S5933DK2_LogConfig]
IRQConfig=7


Computer Management
File Action View Window Help
Computer Management (Local)
System Tools
Event Viewer
Shared Folders
Local Users and Groups
Performance Logs and Alerts
Device Manager
Storage
Removable Storage
Disk Defragmenter
Disk Management
Services and Applications
AMI-CAN
Computer
Disk drives
IDE ATA/ATAPI controllers
Infrared devices
Unknown device
Keyboards
Mice and other pointing devices
Network adapters
Other devices
Video Controller (VGA Compatible)
Ports (COM & LPT)
Processors
Sample Device
SJA1000 CAN - A DRIVER
SJA1000 CAN - B DRIVER
Sound, video and game controllers
Storage volumes
System devices
Universal Serial Bus controllers

7.2 Driver Un-Installation

d. Double-click the **"remove.bat"** file to un-install the driver.
e. Check for the driver removal (**canSend** & **canRecv)** in **"Control Panel -> system -> Hardware -> Device Manager -> Sample Device".**

7.3 Driver Sample Application Usage

1) Steps for CAN frame transmission and reception

1. Open two command windows
2. Run "canSend.exe" in the first command window for transmission.
3. Run "canRecv.exe" in the second command window for reception.

NOTE: The same steps can be followed Vice-versa to make "canSend.exe" as master and "canRecv.exe" as slave.

Command description for canSend:

"Usage: canSend.exe [<can-interface>] [Options] <can-msg>"

"<can-msg> can consist of up to 8 bytes given as a space separated list"

"Options:"

" -i [ex: -i ID] CAN Identifier (default = 1)"

" -e send extended frame"

" -l [ex : -l COUNT] send message COUNT times"

" -p send message data in incremental pattern"

" -b [ex : -b br_value] baud-rate (default = 1000000)"

" -a [ex : -a address] address (default = 0xD7000) "

" -o <filename>" "Input filename"

" -h, --help this help"

Example for Data Transmission (TX) on device CANA:

Pattern : canSend.exe canA –b 1000000 –a d7200 –p –i 2 1 2 3 4 5 6 7 8
File : canSend.exe canA –b 1000000 –a d7200 –p –i 2 –o input.txt

Command description for canRecv:

"Usage: canRecv.exe [<can-interface>] [Options] <can-msg>"
"Options:"

" -b [ex : -b br_value] baud-rate (default = 1000000) "

" -a [ex : -a address] address (default = 0xD7200)"

" -o <filename>" "output filename"

" -h, --help this help"

Example of data reception (RX) on device CANB:

Pattern : canRecv.exe canB –b 1000000 –a d7000
File : canRecv.exe canB –b 1000000 –a d7000 –o output.txt

2) Logging the data transferred

Example:

TX:

canSend.exe canA –b 1000000 –a d7200 –p –i 2 1 2 3 4 5 6 7 8 > tx.txt

RX:

canRecv.exe canB –b 1000000 –a d7000 > rx.txt

Both the log files can be compared using any file comparing utility.

3) Compiling the Application

a. Install WDK (Windows Driver Kit) in development PC.

b. Go to "Programs->Windows Driver Kits -> WDK7600
->Build Environments ->Windows XP
-> x86 Free Build Environment

c. Go to Application source code directory.

d. Build the program into an EXE using the BLD command.