Table of Contents

Introduction

Chapter Overview

This chapter serves as an introduction to the DIP052.

Introduction

The DIP052 provides a general purpose CAN bas interface for IBM PC/AT (ISA) bus compatible systems. The unit relies on the Signetics 82C200 CAN interface component to provide access to the CAN network. The hardware interface includes optical isolation for the CAN signals, reverse polarity protection for the CAN bus power and signal lines and a circuit to detect when the CAN bus power is removed.

The software interfaces to the DIP052 through an interrupt driven device driver. Commands are available to configure both the PC specific interface and the CAN network parameters.

Installing the DIP052

Chapter Overview

This chapter addresses the DIP052 Installation Requirements.

The DIP052 PC to CAN adapter may be installed in an ISA compatible Personal Computer. The module occupies 32 consecutive locations within the processor I/O space. Prior to installation the user must set both the base address and the interrupt level to be used by the adapter.

DIP052 Specifications

Size	8''X4.5'' (ISA Bus Compatible)
Power	5 Volt @ 100mA, 12 Volt @ 80mA bus power
Bus Interface	Optical Isolation, ISO/DIS 11898
Bus Speed	Up to 1 Mbit/sec, 120 nodes
Interrupt Levels	3,4,5,7,10,11,12,15 (user selectable)
I/O Addresses	200-21F, 280-29F, 300-31F, 380-39F

Port Addresses

The DIP052 may be configured to one of 4 separate base addresses using switch positions S9 and S10. Note that when setting the switch the down position is ON and the up position is OFF.

SWITCH	ADDRESS
S10 S9	
OFF OFF	200H
OFF ON	280H
ON OFF	300H
ON ON	380H

Interrupt Levels

The DIP052 may be used in either POLLED or INTERRUPT mode, depending on the support software used. POLLED mode provides lower performance. DIP supplies INTERRUPT driven drivers.

The DIP052 supports 8 different interrupts through switch positions S1 through S8. Only 1 of these switches should be in the ON (Down) position.

Interrupts are a very scarce resource on heavily populated personal computers. The following table shows the 'standard' use of interrupts for AT and 386/486 machines.

Switch (On)	Interrupt (Hex)	Common Usage
S1	IRQ15 (0F)	General I/O
S2	IRQ12 (0C)	General I/O
S3	IRQ11 (0B)	General I/O
S4	IRQ10 (0A)	General I/O
S5	IRQ7 (07)	LPT1
S6	IRQ5 (05)	LPT2, Sound, CD card
S7	IRQ4 (04)	Com1 and Com3
S8	IRQ3 (03)	Com2 and Com4

The hexadecimal values in parenthesis are the corresponding interrupt level to be coded in the MONITOR initialization files (DOS version).

DIP052 Connector Pin-Out

The DIP052 pin-out for the DB9 connector is as follows:

DB-9 Pin Number	Function
1	N.C.- do not connect to this pin
2	CANL
3	Bus --
4	N.C. – do not connect to this pin
5	PC GND – do not connect to this pin in isolated systems
6	Shield Ground
7	CANH
8	N.C. – do not connect to this pin
9	Bus +

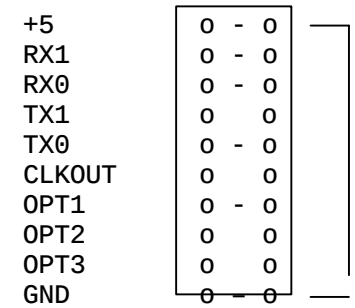
Network Adapter Jumpers

The DIP052 is provided with an optically coupled CAN transceiver. To support applications requiring alternate drivers two sets of dual row 10 position jumpers are provided. Jumper group P1 carries signals from the CAN controller to the CAN transceiver circuit. Jumper group P2 carries signals from the transceiver to the DB9 connector. Jumper P2-10 provides a convenient 120Ω network termination resistor.

Jumper P1

Unless a custom network driver is installed the jumper locations on P1 containing white silk screened lines should be installed.

CONNECTOR P1



Jumper P2

Jumper group P2 connects the CAN transceiver circuit to the DB9 connector. Unless a custom network driver is installed the jumper locations on P2 containing white silk screened lines should be installed. Additional jumpers will be required to select the power option for the transceiver.

To provide full isolation between the CAN network and the PC the transceiver circuit requires a separate power source. This is typically provided by a network wide power supply carried on the BUS+ and BUS_GND signals.

Fully Isolated Interface

The following jumper options should be used when the BUS+ and BUS_GND signals are to be used to power the transceiver.

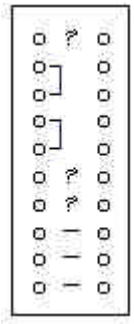
Isolated P2

Internal Function	DB-9	(Pin)	Function
Xcvr Gnd	o — o	(3)	Bus –
Unreg Pwr +	o o	(1)	N.C.
PC +12	o o	(8)	N.C.
Xcvr GND	o o	(4)	N.C.
PC GND	o o	(5)	N.C.
Xcvr GND	o — o	(6)	Bus –
Unreg Pwr +	o — o	(9)	Bus +
Xcvr CAN H	o — o	(7)	Can H
Xcvr CAN L	o — o	(2)	Can L
120 Ohm Resistor	o — o	(-)	120 Ohm Terminating Resistor

Non-Isolated Interface

The following jumper options should be used when the PC +12 power is to be used to power the transceiver. Note that this configuration does not provide galvanic isolation between the CAN network and the PC. Installing the jumpers marked '**?**' will connect the PC +12 volt and GND supply to the DB9 connector. CARE SHOULD BE EXERCISED.

Non-Isolated P2

Internal Function	DB-9	(Pin)	Function
Xcvr GND		(3)	Bus –
Unreg Pwr +		(1)	N.C.
PC +12		(8)	N.C.
Xcvr GND		(4)	N.C.
PC GND		(5)	N.C.
Xcvr GND		(6)	Bus –
Unreg Pwr +		(9)	Bus +
Xcvr CAN H		(7)	Can H
Xcvr CAN L		(2)	Can L
120 Ohm Resistor		(-)	120 Ohm Terminating Resistor

DRV051 Functions

Chapter Overview

This chapter addresses the DRV051 functions (DOS).

DOS Device Driver

The DIP052 uses the Signetics 82C200 Can controller. The controller is I/O mapped and requires 32 consecutive addresses. The base address is defined by the CanPort variable set during initialization. Until the port is configured all functions will return an E_NOTCONFIG error code.

A DOS based interrupt handler is provided in Microsoft compatible C SOURCE code to provide an example of using the DIP052. The driver code provided consists of the following files:

DRV051.C	Source code for interface routines
DRV051.H	Header file

HEADER FILES

DRV051.H defines the function prototypes and various constants. It should be included in any module making access to the DRV051 interface.

ERROR CODES

All user interface functions will return status information in the form of an unsigned int. The following are possible error codes.

E_OK	0x00	- No error detected
E_TIMEOUT	0xffff	- Timed out due to lack of response
E_NOTCONFIG	0xfffe	- DIP052 has not been configured
E_BUSY	0xfffd	- DIP052 controller not available
E_EMPTY	0xfffc	- No messages in receive queue
E_FULL	0xfffb	- Transmit queue is full
E_PRESENT	0xfffa	- DIP052 not present at specified port
E_LENGTH	0xfff9	- length parameter incorrect

PACKET MANAGEMENT

CAN bus traffic is buffered in memory queues. The receive queue is 128 bytes in length. The transmit queue is 32 bytes. Packets received are transferred to the receive queue by the internal interrupt routine and the user Receive function transfers the packet information from the queue to user buffers.

Messages to be transmitted are transferred directly to the DIP052 CAN adapter if possible. If the CAN controller is busy the user message(s) will be queued in the transmit buffer.
Auxiliary functions allow retrieving the entire DIP052 register set as well as reading the DIP052 status register and setting the control and command registers.

DRV051 FUNCTIONS

The following functions are available to application DOS programs.

CanConfig	- configure CAN network interface
CanRecv	- accept incoming CAN messages
CanXmit	- send CAN messages
CanStatus	- return status register (80C200)
CanControl	- send control byte to 80C200
CanCommand	- send command byte to 80C200
CanDump	- return all 80C200 registers
GetAll	- return all registers and FIFO content

CanConfig

sts = CanConfig(port_address, interrupt_level, config)

```
unsigned int  sts;
int port_address;
int interrupt_level;
unsigned char config[6];
```

The DRV051 interface must be advised to the port, interrupt and internal controller conditions to be used.

The port address must not conflict with other I/O devices. The recommended value is 0x300.

The intlevel specifies what hardware interrupt level to use. The interrupt driver does not allow sharing interrupts between devices. A recommended level is 5 (printer LPT2).

The 6 byte configuration array consists of the following UNSIGNED CHAR fields:

accept_code — Message IDENTIFIER(s) to be recognized by this node. Defines which message packets received by the controller will be accepted, subject to mask_code operation.

mask_code — MASK value which will be applied to accept_code and Message IDENTIFIER when qualifying message acceptance. The mask_code value is 'AND'ed with both the incoming message IDENTIFIER and the accept_code. Setting a bit within the mask_code informs the controller to ignore the corresponding bit in the accept_code. A mask_code of 0xFF will allow the controller to receive all packets.

Bus Time 0 — Baud rate multiplier and jitter correction control bits. (Refer to 80C200 specific register information).

Bus Time 1 — Data bit sampling control. (Refer to 80C200 specific register information).

Output cfg — The control register used to determine drive levels for the 80C200 output drivers. Refer to the 80C200 specific register information. For standard hardware configurations use a value of DEF_NORMAL_CFG.

Clock control — The 80C200 generates a separate auxiliary clock output which may be used in specialized hardware configurations. For standard hardware use a value of 0.

The Bus Time 0 and Bus Time 1 parameters are specific to the 80C200 controller and defines both the transmission speed and bit jittering adjustment capability. The drv051.h file includes predefined parameters for 125, 250, 500 and 1000 kbit/second networks.

DEF_SPD125_0	0x03
DEF_SPD125_1	0x1c
DEF_SPD250_0	0x01
DEF_SPD250_1	0x1c
DEF_SPD500_0	0x00
DEF_SPD500_1	0x1c
DEF_SPD1000_0	0x00
DEF_SPD1000_1	0x14

The routine will return E_OK (0) if a controller is found. It will return E_TIMEOUT if the DIP052 fails to respond in .1 second and E_PRESENT if the controller is not found at the specified address.

CanRecv

```
sts = CanRecv(Iptr, Lptr, Bptr)

unsigned int      sts;
unsigned int     *Iptr;
int              *Lptr;
unsigned char    *Bptr;
```

The CanRecv() function is used to receive messages from the CAN network.

If the DIP052 has not been configured the routine will return an E_NOTCONFIG error. If no message is available the routine will return an E_EMPTY error.

The Iptr must be a pointer to an integer which will contain the packet identifier of the received message.

The Lptr points to an integer, which will contain the number of message bytes received. If the packet is a Remote Frame (RTR bit set high) then the length variable is set to 0x10 and no message bytes will be transferred. If the RTR bit is cleared the length variable will be set to the actual number of bytes received in the packet (0x00 - 0x08).

The Bptr points to an unsigned char array to receive the packet data. The CAN specification limits the packet data length to 8 bytes and it is recommended that all receive buffers be at least 8 bytes since it is not possible to pre-determine the length of incoming packets.

CanXmit

```
sts = CanXmit(Id, Len, Bptr)

unsigned int  sts;
unsigned int  Id;
int           Len;
unsigned char *Bptr;
```

The CanXmit() function will transmit a packet on the CAN network.

If the DIP052 has not been configured the routine will return an E_NOTCONFIG error. If no space is available in the message queue the routine will return an E_FULL error. If an invalid Len parameter is specified the routine returns an E_LENGTH error status.

The Id must contain the packet identifier (11 bits).

The Len integer contains the number of message bytes to be transmitted. If the packet is a Remote Frame (RTR bit set high) then the length variable must be set to 0x10 and no message bytes will be transferred.

If the RTR bit is be cleared the length variable must be set to the actual number of bytes to be transmitted the packet (0x00 - 0x08).

The Bptr points to an unsigned char array which contains the packet data. The CAN specification limits the packet data length to 8 bytes.

CanDump

sts = CanDump(Bptr)

unsigned int sts;
unsigned char *Bptr;

The CanDump() function will read all 32 registers from the 80C200 into the user supplied buffer. When accessing the internal registers of the 80C200 it is necessary to reset the device. Any pending messages will be lost.

Refer to the 80C200 data sheet for the register map.

Register 31 is not used by the 80C200 and instead contains the DIP052 status word which reflects the 2 user assigned status bits, the current interrupt status bit and the cable present status bit.

CanStatus

sts = CanStatus(void)

unsigned int sts;

The CanStatus() function will return the status word of the 80C200. If the DIP052 port has not been configured the routine returns an E_NOTCONFIG.

Note that the status value will be in the range 0-0xff. The error codes will be in the range 0xff00 - 0xffff. Refer to the 80C200 data sheet for status register bit assignments.

CanCommand

sts = CanCommand(Cmd)

unsigned int sts;
unsigned int Cmd;

The CanCommand() function will write the user specified Cmd byte to the 80C200 command register. Refer to the 80C200 data sheet for the possible command values.

The routine will return E_NOTCONFIG if the DIP052 port has not been configured.

CanControl

sts = CanControl(Cntr)

unsigned int sts;
unsigned int Cntr;

The CanControl() function will write the user specified Cntr byte to the 80C200 control register. Refer to the 80C200 data sheet for the possible control values.

The routine will return E_NOTCONFIG if the DIP052 port has not been configured.

If the interrupt is due to a Transmit Complete the function checks if additional messages are available from the transmit queue. If so, the oldest message is copied from the buffer to the 80C200 transmission registers.

An ERROR or OVERRUN condition is cleared.

DRV052 Functions

Chapter Overview

This chapter addresses the DRV052 functions (Windows).

DN Functions

These functions allow the user to send DeviceNet commands:

- DNAllocate
- DNFree
- DNReset
- DNGetAttribute
- DNSetAttribute

DNAllocate

This function allows the user to create a M/S connection with a node within the DeviceNet network.

Function Prototype:

long DNAllocate (unsigned short int node, unsigned short int conn,
unsigned char *buf)

Parameters:

node DeviceNet node that the user wants to allocate. The value ranges from 0 to 63.

conn Connection to be established with the node. (Explicit =1, Poll= 2, Strobe= 4, etc).

buf Pointer to an array of bytes for a response from DNAllocate. The size of the array must be 150.

C Declaration:

```
long rts;
int node;
int conn;
unsigned char buf[150];

rts = DNAllocate(node,conn,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                    ' return value
Dim node As Integer
Dim conn As Integer
Dim buf(150) as Byte

rts =DNAllocate(node,conn,buf(0))
```

Return Data:

*buf returns the following data

buf[0],[1]	Error code (0 for successful response)

Ignore the rest of the packet if an Error code is received.

buf[2],[3]	Receive Id from node
buf[4],[5]	Size of CAN message
buf[6],....	Message from node

Comments:

This function returns a non-zero value for Error. See Error Codes for details. The function waits 100ms for a response. If callback is implemented then the response is made available to the user as soon as it is received.

DNFree

This function allows the user to free M/S connection with a node within the DeviceNet network.

Function Prototype:

long DNFree (unsigned short int node, unsigned short int conn, unsigned char *buf)

Parameters:

node DeviceNet node that the user wants to free. The value ranges from 0 to 63.

conn Connection to be established with the node. (Explicit =1, Poll= 2, Strobe= 4).

buf Pointer to an array of bytes for function DNFree. The size of the array must be 150.

C Declaration:

```
long rts;
int node;
int conn;
unsigned char buf[150];

rts = DNFree(node,conn,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                    ' return value
Dim node As Integer
Dim conn As Integer
Dim buf(150) as byte

rts =DNFree(node,conn,buf(0))
```

Return Data:

*buf returns the following data

buf[0],[1] Error code (0 for successful response)

Ignore the rest of the packet if an Error code is received.

buf[2],[3] Receive Id from node

buf[4],[5]	Size of CAN message
buf[6],....	Message from node

Comments:

The function returns a non-zero value for Error. See Error Codes for details. The function waits 100ms for a response. If callback is implemented then the response is made available to the user as soon it is received.

DNReset

This function allows the user to reset the node.

Function Prototype:

long DNReset (unsigned short int node, unsigned short int clss, unsigned short int inst, unsigned short int rlen, unsigned char *buf)

Parameters:

node DeviceNet node that the user wants to reset. The value ranges from 0 to 63.

clss Class to be accessed.

inst Instance to be accessed.

rlen Number of characters send in *buf. Set to 0 if no data is to be sent.

buf On entry, buf has data to be sent to the node. On exit, buf has data response from DNReset. The size of the array must be 150.

C Declaration:

```
long rts;
int clss;
int inst;
int rlen;
unsigned char buf[150];

rts = DNReset(node,clss,inst,rlen,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                    ' return value
Dim clss As Integer
Dim inst as Integer
Dim rlen as Integer
Dim buf(150) as byte

rts =DNReset(node,clss,inst,rlen,buf(0))
```

Return Data:
*buf returns the following data

buf[0],[1]	Error code (0 for successful response)

Ignore the rest of the packet if an Error code is received.

buf[2],[3]	Receive Id from node
buf[4],[5]	Size of message
buf[6],....	Message from node

Comments:

The function returns a non-zero value for Error. See Error Codes for details. The function waits 100ms for a response. If callback is implemented then the response is made available to the user as soon it is received.

DNGetAttribute

This function supports DeviceNet Service GET_SINGLE.

Function Prototype:

long DNGetAttribute (unsigned short int node, unsigned short int clss, unsigned short int inst, unsigned short int attr, unsigned char *buf)

Parameters:

node	DeviceNet node that the user wants to reset. The value ranges from 0 to 63.
clss	Class to be accessed.
inst	Instance to be accessed.

attr Attribute to be accessed.

buf On exit, buf has data response from DNGetAttribute. The size of the array must be 150.

C Declaration:

```
long rts;
int clss;
int inst;
int attr;
unsigned char buf[150];

rts = DNGetAttribute(node,clss,inst,attr,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                    ' return value
Dim clss As Integer
Dim inst as Integer
Dim attr as Integer
Dim buf(150) as byte

rts =DNGetAttribute(node,clss,inst,attr,buf(0))
```

Return Data:

*buf returns the following data

buf[0],[1] Error code (0 for successful response)

Ignore the rest of the packet if an Error code is received.

buf[2],[3] Receive Id from node
buf[4],[5] Size of message
buf[6],.... Message from node

Comments:

The function returns a non-zero value for Error. See Error Codes for details. The function waits 100ms for a response. If callback is implemented then the response is made available to the user as soon it is received.

DNSetAttribute

This function supports DeviceNet Service SET_SINGLE.

Function Prototype:

long DNSetAttribute(unsigned short int node, unsigned short int clss, unsigned short int inst, unsigned short int attr, unsigned short int rlen, unsigned char *buf)

Parameters:

node DeviceNet node that the user wants to reset. The value ranges from 0 to 63.

clss Class to be accessed.

inst Instance to be accessed.

attr Attribute to be accessed.

rlen Number of characters send in *buf. Set to 0 if no data is to be sent.

buf On entry, buf has the data to be sent to the node. On exit, buf has data response from DNSetAttribute. The size of the array must be 150.

C Declaration:

```
long rts;
int clss;
int inst;
int attr;
int rlen;
unsigned char buf[150];

rts = DNSetAttribute(node,clss,inst,attr,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                    ' return value
Dim clss As Integer
Dim inst as Integer
Dim attr as Integer
Dim int as Integer
Dim buf(150) as byte
```

rts =DNSetAttribute(node,clss,inst,attr,rlen,buf(0))

Return Data:

*buf returns the following data

buf[0],[1] Error code (0 for successful response)

Ignore the rest of the packet if an Error code is received.

buf[2],[3] Receive Id from node
buf[4],[5] Size of message
buf[6],.... Message from node

Comments:

The function returns a non-zero value for Error. See Error Codes for details. The function waits 100ms for a response. If callback is implemented then the response is made available to the user as soon it is received.

CAN functions

These functions allow the user to receive and transmit generic CAN messages.

CANRcv

This function will read a message from the VXD. If no messages are available an Error code is generated.

Function Prototype:

long CANRcv (unsigned short int *radd, unsigned short int *rlen, unsigned char *buf)

Parameters:

radd 11-bit identifier.

rlen Number of characters received.

buf On exit, data response from CANRcv. The size of the array must be 150.

C Declaration:

long rts;
int radd;

```
int rlen;
unsigned char buf[150];

rts = CANRcv(radd,rlen,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                    ' return value
Dim radd As Integer
Dim rlen as Integer
Dim buf(150) as byte

rts =CANRcv(radd,rlen,buf(0))
```

Return Data:

*buf returns the following data

buf[0],[1] Error code (0 for successful response)

Ignore the rest of the packet if an Error code is received.

buf[2],[3] Receive Id from node
buf[4],[5] Size of message
buf[6],.... Message from node

Comment:

The function returns a non-zero value for Error. See Error Codes for details.

CANXmit

This function will write a message to the VXD.

Function Prototype:

long CANXmit (unsigned short int xadd, unsigned short int xlen, unsigned char *buf);

Parameters:

radd 11-bit identifier.

xlen Number of characters received. This value must be less or equal to 8.

buf On entry, data to be sent to the node.

C Declaration:

```
long rts;
int xadd;
int xlen;
unsigned char buf[150];

rts = CANXmit(xadd,xlen,&buf);
```

Visual Basic Declaration:

```
Dim rts As Long                    ' return value
Dim xadd As Integer
Dim xlen as Integer
Dim buf(150) as byte

rts =CANXmit(xadd,xlen,buf(0))
```

Return Value:
NONE

VXD functions

LoadVXD

This function loads the proper driver for the DIP052 and DIP065. The DRV052.DLL can be used on Windows 95 and NT operating system.

Function Prototype:

```
long  LoadVXD  (unsigned short int Port, unsigned short int IRQ,
                unsigned char *Config);
```

The port_address must not conflict with other I/O devices. The recommended value is 0x300H. The port_address is set by using the DIPswitch on the DIP052.

SWITCH	ADDRESS
S10 S9	Port
OFF OFF	200H
OFF ON	280H
ON OFF	300H
ON ON	380H

The interrupt_level specifies what hardware interrupt level to use. The interrupt driver does not allow sharing interrupts between devices. A recommended level is 5 (printer LPT2). The interrupt_level is set by using the DIPswitch on the DIP052.

Switch (On)	Interrupt (Hex)	Common Usage
S1	IRQ15 (0F)	General I/O
S2	IRQ12 (0C)	General I/O
S3	IRQ11 (OB)	General I/O
S4	IRQ10 (0A)	General I/O
S5	IRQ7 (07)	LPT1
S6	IRQ5 (05)	LPT2, Sound, CD card
S7	IRQ4 (04)	Com1 and Com3
S8	IRQ3 (03)	Com2 and Com4

The 10-byte configuration array consists of the following UNSIGNED CHAR fields:

Config[0] Do not use. Internal use.
Config[1] Do not use. Internal use.
Config[2] Do not use. Internal use.
Config[3] Do not use. Internal use.

Config[4] accept_code.
accept_code Message IDENTIFIER(s) to be recognized by this node. Defines which message packets received by the controller will be accepted, subject to mask_code operation. (Basically, node address of DIP052)

Config[5] mask_code.
mask_code MASK value which will be applied to accept_code and Message IDENTIFIER when qualifying message acceptance. The mask_code value is 'AND'ed with both the incoming message IDENTIFIER and the accept_code. Setting a bit within the mask_code informs the controller to ignore the corresponding bit in the accept_code. A mask_code of 0xFF will allow the controller to receive all packets.

Config[6] Bus Time 0
Bus Time 0 Baud rate multiplier and jitter correction control bits. (Refer to 80C200 specific register information).

Config[7] Bus Time 1
Bus Time 1 Data bit sampling control. (Refer to 80C200 specific register information).

To set up the data rate to 125kb use:

Bus Time 0	DEF_SPD125_0	0x03
Bus Time 1	DEF_SPD125_1	0x1c

To set up the data rate to 250kb use:

Bus Time 0	DEF_SPD250_0	0x01
Bus Time 1	DEF_SPD250_1	0x1c

To set up the data rate to 500kb use:

Bus Time 0	DEF_SPD500_0	0x00
Bus Time 1	DEF_SPD500_1	0x1c

To set up the data rate to 1000kb use:

Bus Time 0	DEF_SPD1000_0	0x00
Bus Time 1	DEF_SPD1000_1	0x14

Config[8] Output cfg
Output cfg The control register used to determine drive levels for the 80C200 output drivers. Refer to the 80C200 specific register information. For standard hardware configurations use a value of DEF_NORMAL_CFG = 0xfa.

Config[9] Counter reg
Counter reg 0x4e.

Port 0x200, 0x280, 0x300, 0x380 for DIP052.
IRQ 5, 7, 9, 10, 11, 12, 14, 15. Ignored for DIP052.
Config As described above.

Note: On NT Port and IRQ are ignored, Config is processed.

UnloadVXD

This function unloads the proper driver for the DIP052 and DIP065. The DRV052.DLL can be used on Windows 95 and NT operating system.

Function Prototype:

```
long UnloadVXD();
```

RegisterCB

This function registers a callback function within the DLL. This allows the user to be notified when a CAN transaction has occurred.

Function Prototype:

long RegisterCB(unsigned short int mode);

mode 1 = A CAN transaction has been sent.
2 = A CAN transaction has been received.
0xffff = Notified everything.

Utility functions

These functions allow the user to make some simple conversion between data types.

The first set of functions converts bytes into integers, longs or floats by pointing to an element of the array.

unsigned short int Byte2Int (unsigned char *bData);
long Byte2Long (unsigned char *bData);
float Byte2Float (unsigned char *bData);

The second set of functions converts integers, longs or floats into bytes. These functions return 0.

long Int2Byte (unsigned short int *Param1, unsigned char *bData);
long Long2Byte (unsigned long *Param1, unsigned char *bData);
long Float2Byte (float *Param1, unsigned char *bData);

Error Codes

All user interface functions will return status information in the form of an unsigned long. The following are possible error codes.

E_OK	0x00	- No error detected.
E_TIMEOUT	0xffff	- Timed out due to lack of response.
E_NOTCONFIG	0xfffe	- DIP052 has not been configured.
E_BUSY	0xfffd	- DIP052 controller not available.
E_EMPTY	0xfffc	- No messages in receive queue.
E_FULL	0xfffb	- Transmit queue is full.
E_PRESENT	0xfffa	- DIP052 not present at specified port.
E_LENGTH	0xfff9	- length parameter incorrect.
E_PRESENT	0xfff8	- Unable to determine OS.
E_LENGTH	0xfff7	- Generic error.
E_LENGTH	0xfff5	- COM port is in used by another device.

Visual Basic Function Prototypes

The following section describes the declaration under Visual Basic.

Function Prototypes and Declaration:

Declare Function DNAllocate Lib "drv052.dll" (ByVal node As Integer, ByVal conn As Integer, rbuf As Any) As Long

Declare Function DNFree Lib "drv052.dll" (ByVal node As Integer, ByVal conn As Integer, rbuf As Any) As Long

Declare Function DNReset Lib "drv052.dll" (ByVal node As Integer, ByVal cls As Integer, ByVal inst As Integer, ByVal rlen As Integer, rbuf As Any) As Long

Declare Function DNGetAttribute Lib "drv052.dll" (ByVal node As Integer, ByVal cls As Integer, ByVal inst As Integer, ByVal attr As Integer, rbuf As Any) As Long

Declare Function DNSetAttribute Lib "drv052.dll" (ByVal node As Integer, ByVal cls As Integer, ByVal inst As Integer, ByVal attr As Integer, ByVal rlen As Integer, rbuf As Any) As Long

Declare Function CANRcv Lib "drv052.dll" (radd As Integer, rlen As Integer, rbuf As Any) As Long

Declare Function CANXmit Lib "drv052.dll" (ByVal radd As Integer, ByVal rlen As Integer, rbuf As Any) As Long

Declare Function LoadVXD Lib "drv052.dll" (ByVal port As Integer, ByVal Irq As Integer, config As Any) As Long

Declare Function UnloadVXD Lib "drv052.dll" () As Long

Declare Function RegisterCB Lib "drv052.dll" (ByVal mode As Integer) As Long

Declare Function Byte2Int Lib "drv052.dll" (xbuf As Any) As Integer

Declare Function Byte2Long Lib "drv052.dll" (xbuf As Any) As Long

Declare Function Byte2Float Lib "drv052.dll" (xbuf As Any) As Single

```
Declare Function Int2Byte Lib "drv052.dll" (par1 As Integer, xbuf
As Any) As Long

Declare Function Long2Byte Lib "drv052.dll" (par1 As Long, xbuf
As Any) As Long

Declare Function Float2Byte Lib "drv052.dll" (par1 As Single,
xbuf As Any) As Long
```

Obtaining Help

Chapter Overview

This chapter will focus on obtaining help with the product.

Sources for Help

Sources for obtaining help are listed below.

- Visit the DIP Web Site at **http://www.dipinc.com**. The newest updates and revisions to the software as well as the documentation will be posted there.
- Send a request for information through e-mail to **info@dipinc.com**. If the question is related to sales or marketing, send your e-mail to **sales@dipinc.com**.
- Reach us by telephone at **(909) 686-4211**.
- Fax us at **(909) 686-4122**.
- Send us Postal Mail at:

DIP, Inc.
1860 Chicago Ave. Suite I-5
Riverside, CA 92507
USA