Series 2040 Test Systems

MSP User Manual

PN# 4200-0174
Version 2.6

Table of Contents

MSP User Manual 5
- MSP Block Diagram 6
- MULTIPLE SERIAL PROTOCOL BOARD 7
- FUNCTIONAL CALLS 8
- Sample Call 8

UART Functional Calls 9
- sendSerial 10
- recvSerial 11
- setUARTParams 12
- getUARTParams 14
- getMspClock 16
- mspDownloadBuffer 17
- mspException 18
- mspReset 20
- mspVersion 21

CAN Functional Calls 23
- canStat 24
- canrmsg 26
- canwmsg 28
- canRxStart 30
- canRxObject 31
- canRxSingle 32
- canTxSingle 34
- canTxRxSingle 36
- canwcontrol 38
- canwstatus 39
- canrstatus 40
- canwcpuinter 41
- canrcpuinter 42
- canwmaskshort 43
- canwmasklong 44
- canwmaskmsg15 45
- canwbittime0 46
- canwbittime1 47
- canrinterrupt 48
- canrmsgcon0 49

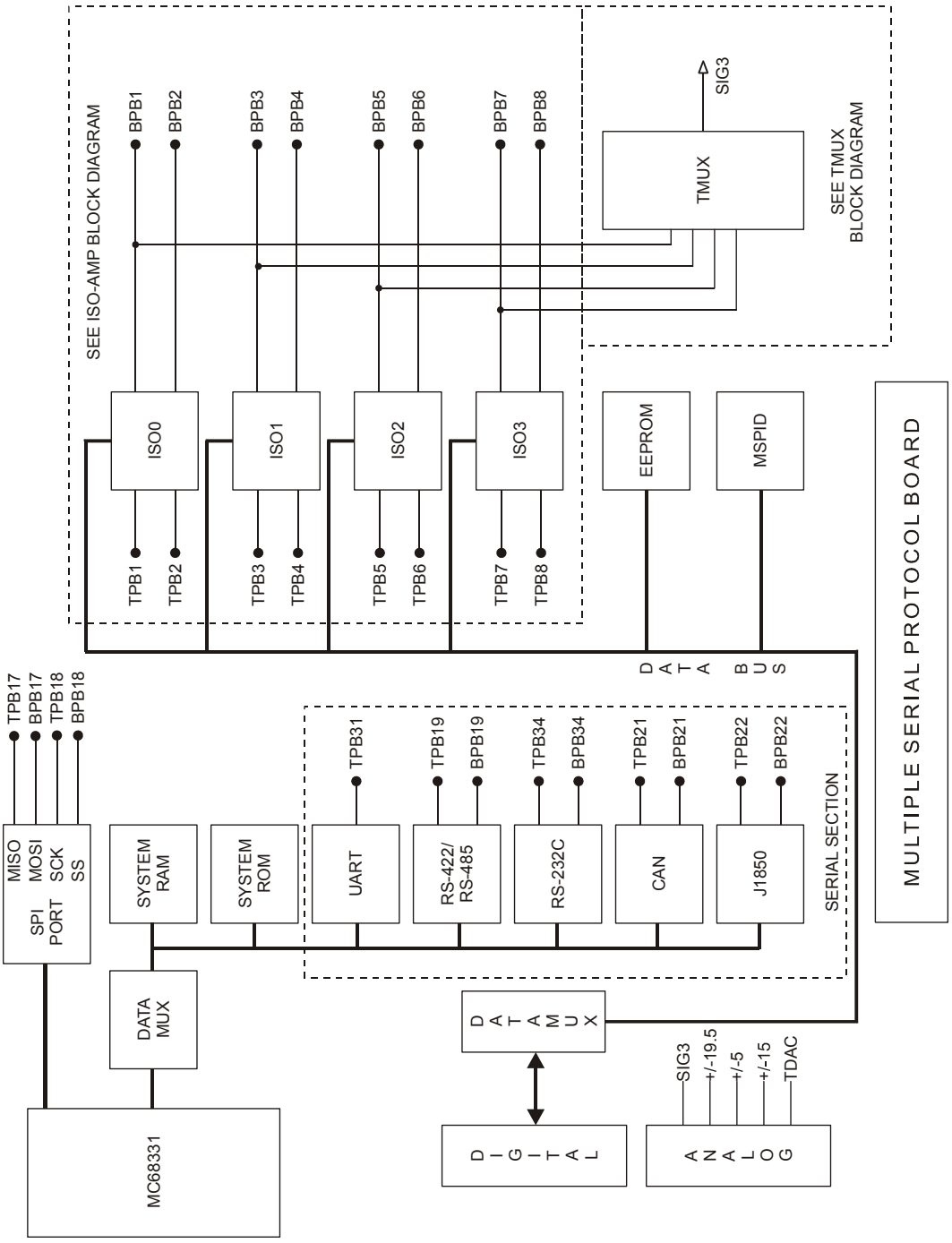
canrmsgcon1 50
canwmsgcon0 51
canwmsgcon1 52
canrmsgarb 53
canwmsgarb 54
canrmsgconfig 55
canwmsgconfig 56
canrmsgdata 57
canwmsgdata 58
canConfigMsgobj 59
canConfigMsgobjExt 60
canErrors 61
canResetErrors 63
canPurgeBuffer 64
canPurgeBufferExt 65
recvCanFrames 66
recvCanFramesExt 68
recvCanFrame15 70
recvCanFrame15Ext 71
sendCanFrames 72
sendCanFramesExt 74
recvCanMsgobj 76
sendCanDataobj 78
sendCanMsgobj 79
SPI Functional Calls 81
getSpiParams 82
setSpiParams 84
spiTransmit 86
spiReceive 87
spiTransceive 88
spiTransmitBuffer 89
J1850 VPW Functional Calls 91
enableJ18560VPWChannelPeriodic 92
getJ1850VPWClock 94
getJ1850VPWException 95
getJ1850VPWParams 97
getJ1850VPWPeriodicParams 100
getJ1850VPWRecvStatus 101
getJ1850VPWVersion 102
J1850VPWDownloadBuffer 103

recvJ1850VPW 104
resetJ1850VPW 105
sendJ1850VPW 106
sendJ1850VPWBuffer 107
sendJ1850VPWPeriodic 108
setJ1850VPWParams 109
setJ1850VPWPeriodicParams 112
KWP2000 Functional Calls 115
Keyword Protocol 2000 Overview 116
initKWP2000Fast 117
sendKWP2000 119
recvKWP2000 121
Additional Functional Calls 123
Isolation Amplifier 124
INST 125
Testhead Multiplexer 126
TMUX 127
Appendix - A 129
Appendix - B 133

MSP User Manual

Version 2.6

MSP Block Diagram



MULTIPLE SERIAL PROTOCOL BOARD

The serial communications section of the Multiple Serial Protocol (MSP) board is designed to communicate with Units Under Test (UUTs) via a variety of serial protocols. Included are RS-232C, asynchronous RS-422/RS-485, and Controller Area Network (CAN). Other protocols such as single wire UART lines can also be used with this card.

In addition to these serial protocols, the MC68331 processor has a Serial Peripheral Interface (SPI) port for simultaneous bi-directional communications with external peripherals through a synchronous serial bus. The features of the MSP-SPI port include:

- Programmable master bit rates
- Programmable clock polarity and phase
- Programmable chip-select
- Programmable transfer length
- Programmable transfer delay

The MSP-SPI port is programmed to operate as the bus master. In this mode, the MSP initiates all serial transfers. The MSP generates the clock output (SCK) used to control the serial transfer of data over the SPI port. SCK polarity, phase and baud rate may be programmed using the setSpiParams functional call. The peripheral chip select (SS) output state is also controlled by the MSP. The active state of the chip select output is programmable using the setSpiParams functional call. The delay between the peripheral chip select becoming active to SCK transition is also programmable. The MISO pin is configured as the serial input pin and the MOSI pin is configured as the serial output pin for SPI transfers. The SPI port transfers data in a burst mode with up to sixteen serial transfers occurring within each burst. Data is transferred with the most significant bit first. The length of each serial transfer is programmable from 8 to 16 bits, inclusive. The delay between transfers within a burst is also programmable. The SPI peripheral chip select output is always driven to an inactive state between each burst. The peripheral chip select state between transfers within a burst is programmable using the setSpiParams functional call.

The MSP board also contains the Selftest Multiplexer (TMUX). The TMUX provides readback of system signals using Sig3, which is returned to the AMS through the Analog Motherboard. It is used in calibrating the D/A's, ARB's and

the AMS using TDAC as a Reference. TDAC is calibrated to a secondary standard during the Digalog Certification Procedure. TMUX is available to the USER and may be used to readback Isolation Amplifier outputs.

The MSP card may optionally contain four Isolation Amplifiers (ISOAMPs). These amplifiers have differential inputs followed by a programmable gain stage. The output is fed through a programmable filter. The inputs of the amplifiers are “floating” and can measure small voltage differences in the presence of large common mode voltages. The Isolation Amplifiers share the functional call INST with the Instrumentation Amplifier card. This functional call is covered later in this manual.

FUNCTIONAL CALLS

The following section contains the functional calls for the MSP board. Visual BASIC declarations and parameters are shown for each call and follow this format:

Sample Call

Visual BASIC Declaration:

Public Sub Sample(ByVal Param1 As Long, ByVal Param2 As Single, ByVal Param3 As Double)

Call Sample(Param1, Param2, Param3)

WHERE:

Param1

= Min1 to Max1. Property (e.g., time, voltage, or current) given in the required units of measure (e.g., seconds, volts, or amps).

Param2

= Min2 to Max2. Property (e.g., time, voltage, or current) given in the required units of measure (e.g., seconds, volts, or amps).

Param3

= Min3 to Max3. Property (e.g., time, voltage, or current) given in the required units of measure (e.g., seconds, volts, or amps).

UART Functional Calls

sendSerial

The sendSerial functional call sends a message using the default UART port on the MSP board. The message must be fully assembled by the calling program, as the function transmits the message transparently. Communication is at the baud rate set up by the setUARTParams call.

Visual BASIC Declaration:

Public Sub sendSerial(ResultCode As Integer, smsg() As Integer, ByVal msgLen As Integer, ByVal timeout As Double)

Call sendSerial(ResultCode, smsg, msgLen, timeout)

WHERE:

ResultCode
= The returned error code.

smsg
= The message to send. An array of integers with the upper byte in each array being ignored.

msgLen
= 1 to 4096. Number of bytes to send.

timeout
= 0 to 65. The time to wait in seconds for the receive line to be idle before transmitting.

EXAMPLE:

```
Dim ResultCode As Integer
Dim smsg() As Integer
Dim msgLen As Integer
Dim timeout As Double
msgLen = 100

Call sendSerial(Resultcode,smsg(),msgLen,1) ..... Send smsg and wait
 ..... 1 second for an errorcode.
```

recvSerial

The recvSerial functional call receives a message using the default UART port on the MSP board. The message must be disassembled by the calling program as the function receives the message transparently. Communication is at the baud rate set up by the setUARTParams call.

Visual BASIC Declaration:

Public Sub recvSerial(ResultCode As Integer, rmsg() As Integer, msgLen As Integer, ByVal timeout As Double)

Call recvSerial(ResultCode, rmsg, msgLen, timeout)

WHERE:

ResultCode

= The returned error code.

rmsg

= The message to receive. An array of integers. Upon return, the upper byte of each integer contains a 0. The lower byte is a received byte.

msgLen

= 0 to 4096. The calling variable will contain the number of bytes to wait for. The returning variable will contain the number of bytes read.

timeout

= 0 to 65. The time to wait in seconds for a message.

EXAMPLES:

```
Dim ResultCode As Integer
Dim rmsg() As Integer
Dim msgLen As Integer
Dim timeout As Double
msgLen = 100

Call recvSerial(ResultCode,rmsg(),msgLen,1) ..... Receive message and wait
 ..... 1 second for an errorcode.
```

setUARTParams

The setUARTParams functional call sets up the default parameters used by the serial functions. The MSP board reverts back to the default UART parameter settings on Testhead power-up or reset. A Testhead reset occurs whenever the TClear function is called.

Visual BASIC Declaration:

Public Sub setUARTParams(ByVal Index As Integer, ByVal Value As Long)

Call setUARTParams(Index, Value)

WHERE:

Index

= Index of the parameter to set.

Value

= Value to set the parameter to.

INDEX#	PARAMETER NAME	PARAMETER VALUE
1	Port Code	0 = RS232 (Default) 1 = Single line SXR 2 = RS422
2	Baudrate	Baudrate (Default = 8192)
3	Echo Timeout (0 to 65000)	Milliseconds (Default = 10)
4	Receive Timeout (0 to 65000)	Milliseconds Default = 1000)
5	Gap Timeout (0 to 65000)	Milliseconds (Default = 15)
6	Idle Timeout (0 to 65000)	Milliseconds (Default = 1000)
8	Check Echo	0 = Do not check echo byte 1 = Check echo type (Default)
9	XDE Compliant	0 = Do not check for XDE compliance 1 = Check for XDE compliance (Default)
10	SCI Idle Detect	0 = Do not use SCI port idle detect 1 = Use SCI port idle detect (Def.)
11	SXR Bus Power	0 = +5VDC (Default) 1 = +15VDC
12	Inter Byte Delay	Milleseconds (Default = 0)

Port Code	=	The port to use on the MSP board
Baudrate	=	The baudrate to set the MSP board to
Echo Timeout	=	The time to wait for an echo byte. (Not currently implemented.)
Receive Timeout	=	The time to wait for a response from the sender
Gap Timeout	=	The maximum time allowed between characters
Idle Timeout	=	The maximum time to wait for an idle line before transmitting
Check Echo	=	Flag to compare the echoed byte with the transmitted byte
XDE Compliant	=	Flag to maintain strict XDE 5024 compliance (Max 170 data bytes)
SCI Idle Detect	=	Flag to use the SCI idle port detect. If this flag is set, the Gap Timeout will be ignored unless it is less than the SCI port idle detect.
SXR Bus Power	=	Selects the voltage level of the SXR bus.
Inter Byte Delay	=	The time delay between transmitted bytes.

Note: The gap timeout also determines how long the MSP board will wait before determining that a generic serial message has ended.

Note: The SXR Bus Power option is only available on certain versions of the MSP board. A 100:021 UART Unknown Parameter error will be returned if the MSP hardware doesn't support this option.

EXAMPLES:

Call setUARTParams(2,16384) ..... Set baudrate to 16384.

Call setUartParams(5, 1).....Set the Gap Timeout to 1 millesecond.

getUARTParams

The getUARTParams functional call retrieves the default parameters used by the serial functions.

Visual BASIC Declaration:

Public Sub getUARTParams(ByVal Index As Integer, Value As Long)

Call getUARTParams(Index, Value)

WHERE:

Index

= Index of the parameter to get.

Value

= Value the parameter is set to.

INDEX#	PARAMETER NAME	PARAMETER VALUE
1	Port Code	0 = RS232 1 = Single line SXR 2 = RS422
2	Baudrate	Baudrate
3	Echo Timeout (0 to 65000)	Milliseconds
4	Receive Timeout (0 to 65000)	Milliseconds
5	Gap Timeout (0 to 65000)	Milliseconds
6	Idle Timeout (0 t0 65000)	Milliseconds
8	Check Echo	0 = Do not check echo byte 1 = Check echo type
9	XDE Compliant	0 = Do not check for XDE compliance 1 = Check for XDE compliance
10	SCI Idle Detect	0 = Do not use SCI port idle detect 1 = Use SCI port idle detect
11	SXR Bus Power	0 = +5VDC 1 = +15VDC
12	Inter Byte Delay	Milleseconds (Default = 0)

Port Code	=	The port used on the MSP board
Baudrate	=	The baudrate the MSP board is set to
Echo Timeout	=	The time to wait for an echo byte. (Not currently implemented.)
Receive Timeout	=	The time to wait for a response from the sender
Gap Timeout	=	The maximum time allowed between characters
Idle Timeout	=	The maximum time to wait for an idle line before transmitting
Check Echo	=	Flag to compare the echoed byte with the transmitted byte
XDE Compliant	=	Flag to maintain strict XDE 5024 compliance (Max 170 data bytes)
SCI Idle Detect	=	Flag to use the SCI idle port detect. If this flag is set, the Gap Timeout will be ignored unless it is less than the SCI port idle detect.
SXR Bus Power	=	Selects the voltage level of the SXR bus.
Inter Byte Delay	=	The time delay between transmitted bytes.

Note: The gap timeout also determines how long the MSP board will wait before determining that a generic serial message has ended.

Note: The SXR Bus Power option is only available on certain versions of the MSP board. A 100:021 UART Unknown Parameter error will be returned if the MSP hardware doesn't support this option.

EXAMPLES:

```
Dim Value As Long

Call getUARTParams(2,Value) ..... Get baudrate.
Call getUARTParams(5,Value) ..... Get gap timeout.
```

getMspClock

The getMspClock functional call gets the MSP system clock frequency in Hertz from the MSP board.

Visual Basic Declaration

Public Cub getMspClock(ByRef frequency As Long)

Call getMspClock(frequency)

WHERE:

frequency

= The MSP clock frequency in Hertz.

EXAMPLES:

Dim frequency As Long

Call getMspClock(frequency) ..... Get the MSP system clock frequency.

WHERE:

frame

= The array to write the exception frame to.

frameSize

= 1 to 92. (maximum MSP exception frame size). The size of the frame in bytes. This parameter will be updated with the actual MSP exception frame size in bytes.

EXAMPLES:

```
Dim frame(1 to 92) As Byte
Dim frameSize As Integer
frameSize = 92

Call mspException(frame, frameSize)..... Get the MSP exception frame.
```

mspReset

The mspReset functional call resets the microcontroller on the MSP board.

Visual Basic Declaration:

Public Sub mspReset()

Call mspReset

EXAMPLES:

Call mspReset ..... Resets the microcontroller on the MSP board.

mspVersion

The mspVersion functional call gets the firmware version string from the MSP board.

Visual Basic Declaration:

Public Sub mspVersion(Version As String)

Call mspVersion(Version)

WHERE:

Version

= String containing the MSP firmware version string.

EXAMPLES:

Dim Version As String

Call mspVersion(Version) ..... Get the MSP firmware version string.

CAN Functional Calls

canStat

The canstat call reads all of the following CAN controller registers:

- Control
- Status
- CPU Interface
- Global Mask - Standard
- Global Mask - Extended
- Message 15 Mask
- Clockout
- Bus Configuration
- Bit Timing Register 0
- Bit Timing Register 1
- Interrupt

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canstat(ByRef control As Byte, ByRef status As Byte, ByRef cpuinter As Byte, ByRef maskshort As Integer, ByRef masklong As Long, ByRef maskmsg15 As Long, ByRef clkout As Byte, ByRef busconfig As Byte, ByRef bittime0 As Byte, ByRef bittime1 As Byte, ByRef intreg As Byte)

Call canstat(control , status, cpuinter, maskshort, masklong, maskmsg15, clkout, busconfig, bittime0, bittime1, intreg)

WHERE:

control
= Data from the CAN Control Register.

status
= Data from the CAN Status Register.

cpuinter
= Data from the CAN CPU Interface Register.

maskshort
= Data from the CAN Global Mask - Standard Registers, 2 bytes.

masklong
= Data from the CAN Global Mask - Extended Registers, 4 bytes.

maskmsg15
= Data from the CAN Message 15 Mask Registers, 4 bytes.

clkout
= Data from the CAN Clockout Register.

busconfig
= Data from the CAN Bus Configuration Register.

bittime0
= Data from the CAN Bit Timing Register 0.

bittime1
= Data from the CAN Bit Timing Register 1.

intreg
= Data from the CAN Interrupt Register.

EXAMPLES:

```
Dim control As Byte
Dim status As Byte
Dim cpuinter As Byte
Dim maskshort As Integer
Dim masklong As Long
Dim maskmsg15 As Long
Dim clkout As Byte
Dim busconfig As Byte
Dim bittime0 As Byte
Dim bittime1 As Byte
Dim intreg As Byte

Call canstat (control, status, cpuinter, maskshort, masklong, maskmsg15, clkout,
busconfig, bittime0, bittime1, intreg)
```

canrmsg

The canrmsg call reads all of the following CAN controller message registers:

- Message Control 0
- Message Control 1
- Arbitration Registers
- Message Configuration
- Message Data

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canrmsg(ByVal msgobj As Byte, ByRef msgcon0 As Byte, ByRef msgcon1 As Byte, ByRef msgarb As Long, ByRef msgconfig As Byte, msgdata() As Byte)

Call canrmsg(msgobj, msgcon0, msgcon1, msgarb, msgconfig, msgdata)

WHERE:

msgobj
= 1 to 15. CAN Message object number to read.

msgcon0
= Data from the CAN Message Control 0 Register.

msgcon1
= Data from the CAN Message Control 1 Register.

msgarb
= Data from the CAN Message Arbitration Registers, 4 bytes.

msgconfig
= Data from the CAN Message Configuration Register.

msgdata
= Data array from the CAN Message Data Registers, 8 bytes.

EXAMPLES:

```
Dim msgcon0 As Byte
Dim msgcon1 As Byte
Dim msgarb As Long
Dim msgconfig As Byte
Dim msgdata(7) As Byte
Call canrmsg (1, msgcon0, msgcon1, msgarb, msgconfig, msgdata)
```

canwmsg

The canwmsg call writes data to the following CAN controller message registers:

- Message Control 0
- Message Control 1
- Arbitration Registers
- Message Configuration
- Message Data

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwmsg(ByVal msgobj As Byte, ByVal msgcon0 As Byte, ByVal msgcon1 As Byte, ByVal msgarb As Long, ByVal msgconfig As Byte, msgdata() As Byte)

Call canwmsg(msgobj, msgcon0, msgcon1, msgarb, msgconfig, msgdata)

WHERE:

msgobj
= 1 to 15. CAN Message object number to write.

msgcon0
= Data to be written to the CAN Message Control 0 Register.

msgcon1
= Data to be written to the CAN Message Control 1 Register.

msgarb
= Data to be written to the CAN Message Arbitration Registers, 4 bytes.

msgconfig
= Data to be written to the CAN Message Configuration Register.

msgdata
= Data array to be written to the CAN Message Data Registers, 8 bytes

EXAMPLES:

```
Dim msgdata(7) As Byte

msgdata(0) = 0
msgdata(1) = 1
msgdata(2) = 2
msgdata(3) = 3
msgdata(4) = 4
msgdata(5) = 5
msgdata(6) = 6
msgdata(7) = 7

Call canwmsg (1, &HA5, &H55, 0, &H8C, msgdata)
```

canRxStart

The canStartRx functional call is a register-based function that configures the passed CAN controller Message Object to receive a single message.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canStartRx(ByVal rxMsgObj As Byte, ByVal rxMsgId As Long, ByVal msgType As Byte)

Call canStartRx(rxMsgObj, rxMsgId, msgType)

WHERE:

rxMsgObj =

= 1 to 15. CAN Controller Message Object to configure.

rxMsgId

= Receive Message ID.

msgType

= 0 Standard CAN message (11 bit ID).
= 1 Extended CAN message (29 bit ID).

EXAMPLES:

Call canStartRx(1, &H12345678, 1) ..... Configure message object #1
..... to receive any extended CAN message
..... with a message ID of &H12345678.

canRxObject

The canRxObject functional call is a register-based function that retrieves a single received CAN message. The selected CAN Controller Message Object must first be configured using the canStartRx function before any messages can be received. A timeout error will be returned if a message hasn't been received before the timeout period elapses.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canRxObject(ByVal rxMsgObj As Byte, rxArray() As Byte, ByRef numBytes As Integer, ByVal timeout As Double)

Call canRxObject(rxMsgObj, rxArray, numBytes, timeout)

WHERE:

rxMsgObj

= 1 to 15. CAN Controller Message Object.

rxArray

= Array to write the received data to.

numBytes

= 0 to 8. Number of bytes to receive. This variable will be updated with the actual number of data bytes received. The size of rxArray buffer must be greater than or equal to the number of bytes to receive.

timeout

= 0 to 65.5 seconds. The time in seconds to wait for message to be received.

EXAMPLES:

```
Dim rxArray(0 to 7) As Byte
Dim numBytes As Integer
numBytes = 8
Call canRxObject(1, rxArray, numBytes, 1#) ..... Read message
 ..... from message object #1.
```

canRxSingle

The canRxSingle functional call is a register-based function that configures a message object as a receiver and then receives a single CAN message. A timeout error will be returned if a message hasn't been received before the timeout period elapses.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canRxSingle(ByVal rxMsgObj As Byte, ByVal rxMsgId As Long, ByVal msgType As Byte, rxArray() As Byte, ByRef numBytes As Integer, ByVal timeout As Double)

Call canRxSingle(rxMsgObj, rxMsgId, msgType, rxArray, numBytes, timeout)

WHERE:

rxMsgObj

= 1 to 15. CAN Controller Message Object.

rxMsgId

= Receive Message ID.

msgType

= 0 Standard CAN message (11 bit ID).
= 1 Extended CAN message (29 bit ID)

rxArray

= Array to write the received data to.

numBytes

= 0 to 8. Number of bytes to receive. This variable will be updated with the actual number of data bytes received. The size of rxArray buffer must be greater than or equal to the number of bytes to receive.

timeout

= 0 to 65.5 seconds. The time in seconds to wait for message to be received.

EXAMPLES:

```
Dim rxArray(0 to 7) As Byte
Dim numBytes As Integer
numBytes = 8

Call canRxSingle(1, &H12345678, 1, rxArray, numBytes, 1#).....
 .....Configure and receive an extended CAN message
 ..... with a message ID of &H12345678 using message object #1.
```

canTxSingle

The canTxSingle functional call is a register-based function that configures a message object as a transmitter and then transmits a single CAN message.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canTxSingle(ByVal txMsgObj As Byte, ByVal txMsgId As Long, ByVal msgType As Byte, txArray() As Byte, ByVal numBytes As Integer, ByVal timeout As Double)

Call canTxSingle(txMsgObj, txMsgId, msgType, txArray, numBytes, timeout)

WHERE:

txMsgObj

= 1 to 14. CAN Controller Message Object.

txMsgId

= Transmit Message ID.

msgType

= 0 Standard CAN message (11 bit ID).
= 1 Extended CAN message (29 bit ID).

txArray

= Array containing the data to transmit.

numBytes

= 0 to 8. Number of bytes to transmit. The size of txArray array must be greater than or equal to the number of bytes to transmit.

timeout

= 0 to 65.5 seconds. The time in seconds to wait for an idle bus.

EXAMPLES:

Dim txArray(0 to 7) As Byte

Call canTxSingle(1, &H12345678, 1, txArray, 8, 1#) ..... Configure and transmit
..... an eight byte extended CAN message with a message ID
..... of &H12345678 using message object #1.

canTxRxSingle

The canTxRxSingle functional call is a register-based function that first transmits a single CAN message and then receives a single response message. This function configures both the transmit and receive message objects before transmitting and receiving a message. A timeout error will be returned if the transmit object has not detected a bus idle or the receive object has not received a message before the timeout period elapses.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canTxRxSingle(ByVal txMsgObj As Byte, ByVal txMsgId As Long, txArray() As Byte, ByVal numTxBytes As Integer, ByVal rxMsgObj As Byte, ByVal rxMsgId As Long, ByVal msgType As Byte, rxArray() As Byte, ByRef numRxBytes As Integer, ByVal timeout As Double)

Call canwcontrol(control)Call canTxRxSingle(txMsgObj, txMsgId, txArray, numTxBytes, rxMsgObj, rxMsgId, msgType, rxArray, numRxBytes, timeout)

WHERE:

txMsgObj

= 1 to 14. Transmitter CAN Controller Message Object.

txMsgId

= Transmit Message ID.

txArray

= Array containing the data to transmit.

numTxBytes

= 0 to 8. Number of bytes to transmit. The size of txArray array must be greater than or equal to the number of bytes to transmit.

rxMsgObj

= 1 to 15. Receiver CAN Controller Message Object.

rxMsgId

= Receive Message ID.

msgType

= 0 Standard CAN message (11 bit ID).
= 1 Extended CAN message (29 bit ID).

rxArray

= Array to write the received data to.

numRxBytes

= 0 to 8. Number of bytes to receive. This variable will be updated with the actual number of data bytes received. The size of rxArray buffer must be greater than or equal to the number of bytes to receive.

timeout

= 0 to 65.5 seconds. The time in seconds to wait for either an idle bus (Tx) or a message to be received (Rx).

EXAMPLES:

```
Dim txArray(0 to 7) As Byte
Dim rxArray(0 to 7) As Byte
Dim numRxBytes As Integer
numRxBytes = 8

Call canTxSingle(1, &H1234, txArray, 8, 2, &H5678, 1, rxArray, numBytes, 1#)
 ..... Configure both the transmit and receive message objects.
 ..... Transmit an eight byte extended CAN message
 ..... with a message ID of &H1234 using message object #1,
 ..... and receive an extended CAN message
 ..... with a message ID of &H5678 using message object #2.
```

canwcontrol

The canwcontrol functional call writes data to the CAN Control register.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwcontrol(ByVal control As Byte)

Call canwcontrol(control)

WHERE:

control

= Data to be written to the CAN Control Register

EXAMPLES:

Call canwcontrol(&H4A)

canwstatus

The canwstatus functional call writes data to the CAN Status register.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwstatus(ByVal status As Byte)

Call canwstatus(status)

WHERE:

status

= Data to be written to the CAN Status Register.

EXAMPLES:

Call canwstatus(&H07)

canrstatus

The canrstatus functional call reads data from the CAN Status register.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual BasicDeclaration:

Public Sub canrstatus(ByRef status As Byte)

Call canrstatus(status)

WHERE:

status

= Data read from the CAN Status Register.

EXAMPLES:

```
Dim status As Byte

Call canrstatus(status)
```

canwcpuinter

The canwcpuinter functional call writes data to the CAN CPU Interface register.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwcpuinter(ByVal cpuinter As Byte)

Call canwcpuinter(cpuinter)

WHERE:

cpuinter

= Data to be written to the CAN CPU Interface Register.

EXAMPLES:

Call canwcpuinter(&H40)

canrcpuinter

The canrcpuinter functional call reads data from the CAN CPU Interface register.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canrcpuinter(ByRef cpuinter As Byte)

Call canrcpuinter(cpuinter)

WHERE:

cpuinter

= Data read from the CAN CPU Interface Register.

EXAMPLES:

```
Dim cpuinter As Byte

Call canrcpuinter(cpuinter)
```

canwmaskshort

The canwmaskshort functional call writes data to the CAN Global Mask - Standard registers.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwmaskshort(ByVal maskshort As Integer)

Call canwmaskshort(maskshort)

WHERE:

maskshort

= Data to be written to the CAN Global Mask - Standard Registers, 2 bytes.

EXAMPLES:

Call canwmaskshort(&H1234)

canwmasklong

The canwmasklong functional call writes data to the CAN Global Mask - Extended registers.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwmasklong(ByVal masklong As Long)

Call canwmasklong(masklong)

WHERE:

masklong

= Data to be written to the CAN Global Mask - Extended Registers, 4 bytes.

EXAMPLES:

Call canwmasklong(&H12345678)

canwmaskmsg15

The canwmaskmsg15 functional call writes data to the CAN Message 15 Mask registers.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwmaskmsg15(ByVal maskmsg15 As Long)

Call canwmaskmsg15(maskmsg15)

WHERE:

maskmsg15

= Data to be written to the CAN Message 15 Mask Registers, 4 bytes.

EXAMPLES:

Call canwmaskmsg15(&H12345678)

canwbittime0

The canwbittime0 functional call writes data to the CAN Bit Timing register 0.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwbittime0 Lib(ByVal bittime0 As Byte)

Call canwbittime0(bittime0)

WHERE:

bittime0

= Data to be written to the CAN Bit Timing Register 0.

EXAMPLES:

Call canwbittime0(&H44)

canwbittime1

The canwbittime1 functional call writes data to the CAN Bit Timing register 1.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwbittime1(ByVal bittime1 As Byte)

Call canwbittime1(bittime1)

WHERE:

bittime1

= Data to be written to the CAN Bit Timing Register 1.

EXAMPLES:

Call canwbittime1(&H94)

canrinterrupt

The canrinterrupt functional call reads data from the CAN Interrupt register.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canrinterrupt(ByRef intreg As Byte)

Call canrinterrupt(intreg)

WHERE:

intreg = Data read from the CAN Interrupt Register.

EXAMPLES:

```
Dim intreg As Byte

Call canrinterrupt(intreg)
```

canrmsgcon0

The canrmsgcon0 functional call reads data from the CAN Message Control 0 register.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canrmsgcon0(ByVal msgobj As Byte, ByRef msgcon0 As Byte)

Call canrmsgcon0 (msgobj, msgcon0)

WHERE:

msgobj

= 1 to 15. CAN Message object number to read.

msgcon0

= Data read from the CAN Message Control 0 Register.

EXAMPLES:

Dim msgcon0 As Byte

Call canrmsgcon0(1, msgcon0) ..... Read message object #1 control 0 register.

canrmsgcon1

The canrmsgcon1 functional call reads data from the CAN Message Control 1 register.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canrmsgcon1(ByVal msgobj As Byte, ByRef msgcon1 As Byte)

Call canrmsgcon1(msgobj, msgcon1)

WHERE:

msgobj

= 1 to 15. CAN Message object number to read.

msgcon1

= Data read from the CAN Message Control 1 Register.

EXAMPLES:

```
Dim msgcon1 As Byte

Call canrmsgcon1(1, msgcon1) ..... Read message object #1 control 1 register.
```

canwmsgcon0

The canwmsgcon0 functional call writes data to the CAN Message Control 0 register.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwmsgcon0(ByVal msgobj As Byte, ByVal msgcon0 As Byte)

Call canwmsgcon0 (msgobj, msgcon0)

WHERE:

msgobj

= 1 to 15. CAN Message object number to write.

msgcon0

= Data to write to the CAN Message Control 0 Register.

EXAMPLES:

Call canwmsgcon0(1, &HA5)

canwmsgcon1

The canwmsgcon1 functional call writes data to the CAN Message Control 1 register.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwmsgcon1(ByVal msgobj As Byte, ByVal msgcon1 As Byte)

Call canwmsgcon1 (msgobj, msgcon1)

WHERE:

msgobj

= 1 to 15. CAN Message object number to write.

msgcon1

= Data to write to the CAN Message Control 1 Register.

EXAMPLES:

Call canwmsgcon1(1, &H55)

canrmsgarb

The canrmsgarb functional call reads data from the CAN Message Arbitration registers.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canrmsgarb(ByVal msgobj As Byte, ByRef msgarb As Long)

Call canrmsgarb(msgobj, msgarb)

WHERE:

msgobj

= 1 to 15. CAN Message object number to read.

msgarb

= Data read from the CAN Message Arbitration Registers, 4 bytes.

EXAMPLES:

Dim msgarb As Long

Call canrmsgarb(1, msgarb) ..... Read message object #1 arbitration registers.

canwmsgarb

The canwmsgarb functional call writes data to the CAN Message Arbitration registers.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwmsgarb(ByVal msgobj As Byte, ByVal msgarb As Long)

Call canwmsgarb (msgobj, msgarb)

WHERE:

msgobj

= 1 to 15. CAN Message object number to write.

msgarb

= Data to be written to the CAN Message Arbitration Registers, 4 bytes.

EXAMPLES:

Call canwmsgarb(1, &H12345678)

canrmsgconfig

The canrmsgconfig functional call reads data from the CAN Message Configuration register.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canrmsgconfig(ByVal msgobj As Byte, ByRef msgconfig As Byte)

Call canrmsgconfig (msgobj, msgconfig)

WHERE:

msgobj

= 1 to 15. CAN Message object number to read.

msgconfig

= Data read from the CAN Message Configuration Register.

EXAMPLES:

```
Dim msgconfig As Byte

Call canrmsgconfig(1, msgconfig) .....Read message object #1
 ..... configuration register.
```

canwmsgconfig

The canwmsgconfig functional call writes data to the CAN Message Configuration register.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwmsgconfig(ByVal msgobj As Byte, ByVal msgconfig As Byte)

Call canwmsgconfig (msgobj, msgconfig)

WHERE:

msgobj

= 1 to 15. CAN Message object number to write.

msgconfig

= Data to write to the CAN Message Configuration Register.

EXAMPLES:

Call canwmsgconfig(1, &H8C)

canrmsgdata

The canrmsgdata functional call reads data from the CAN Message Data registers.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canrmsgdata(ByVal msgobj As Byte, msgdata() As Byte)

Call canrmsgdata (msgobj, msgdata)

WHERE:

msgobj

= 1 to 15. CAN Message object number to read.

msgdata

= Data array read from the CAN Message Data Registers, 8 bytes.

EXAMPLES:

Dim msgdata(7) As Byte

Call canrmsgdata(1, msgdata) ..... Read message object #1 data registers.

canwmsgdata

The canwmsgdata functional call writes data to the CAN Message Data registers.

Refer to the Intel 82527 Serial Communications Controller documentation for Can Controller register information.

Visual Basic Declaration:

Public Sub canwmsgdata(ByVal msgobj As Byte, msgdata() As Byte)

Call canwmsgdata (msgobj, msgdata)

WHERE:

msgobj

= 1 to 15. CAN Message object number to write.

msgdata

= Data array to be written to the CAN Message Data Registers, 8 bytes.

EXAMPLES:

```
Dim msgdata(7) As Byte

msgdata(0) = &H01
msgdata(1) = &H23
msgdata(2) = &H45
msgdata(3) = &H67
msgdata(4) = &H89
msgdata(5) = &HAB
msgdata(6) = &HCD
msgdata(7) = &HEF

Call canwmsgdata(1, msgdata)
```

canConfigMsgobj

The canConfigMsgobj call is a frame based function which configures a message object with the passed parameters. An error message will be returned if the Message Object is busy transmitting or receiving messages.

Visual Basic Declaration:

Public Sub canConfigMsgobj(ByVal msgobj As Byte, ByVal msgarb As Long, ByVal direction As Byte, ByVal valid As Byte)

Call canConfigMsgobj(msgobj, msgarb, direction, valid)

WHERE:

msgobj

=	1	to 15. The message object to configure

msgarb

=		The 32-bit message object arbitration ID register value.

direction

=	0	Configure as a receive message object
=	1	Configure as a transmit message object

valid

=	0	Unconfigure message object by marking it invalid
=	1	Make message object active by marking it valid

EXAMPLES:

Call canConfigMsgobj(1, &H12345678, 1, 1) .....Configure message object #1 ..... as a transmitter with an arbitration ID register value of &H12345678.

canConfigMsgobjExt

The canConfigMsgobjExt call is an extended message format, frame based function which configures a message object with the passed parameters. An error message will be returned if the Message Object is busy transmitting or receiving messages.

Visual Basic Declaration:

Public Sub canConfigMsgobjExt(ByVal msgobj As Byte, ByVal msgid As Long, ByVal direction As Byte, ByVal valid As Byte)

Call canConfigMsgobjExt(msgobj, msgid, direction, valid)

WHERE:

msgobj

= 1 to 15. The message object to configure.

msgid

= 29-bit message ID.

direction

= 0 Configure as a receive message object.
= 1 Configure as transmit message object.

valid

= 0 Unconfigure message object by marking it invalid.
= 1 Make message object active by marking it valid.

EXAMPLES:

Call canConfigMsgobjExt(1, &H12345678, 1, 1) ..... Configure message object #1 ..... as a transmitter with a message ID of &H12345678.

canErrors

The canErrors call retrieves the value of each CAN Controller error counter and the total number of errors.

Visual Basic Declaration:

Public Sub canErrors(ByRef errcount As Long, ByRef spurint As Integer, ByRef msglost As Integer, ByRef msgundefined As Integer, ByRef stuff As Integer, ByRef form As Integer, ByRef ack As Integer, ByRef bit1 As Integer, ByRef bit0 As Integer, ByRef crc As Integer)

Call canErrors(errcount, spurint, msglost, msgundefined, stuff, form, ack, bit1, bit0, crc)

WHERE:

errcount
= Total number of errors.

spurint
= Spurious interrupt error counter.

msglost
= Message lost error counter.

msgundefined
= Message undefined error counter.

stuff
= Stuff error counter.

form
= Form error counter.

ack
= Acknowledgment error counter.

bit1
= Bit 1 error counter.

bit0
= Bit 0 error counter.

crc
= CRC error counter.

EXAMPLES:

```
Dim errcopunt As Long
Dim spurint As Integer
Dim msglost As Integer
Dim msgundefined As Integer
Dim stuff As Integer
Dim form As Integer
Dim ack As Integer
Dim bit1 As Integer
Dim bit0 As Integer
Dim crc As Integer

Call canErrors(errcount, spurint, msglost, msgundefined, stuff, form, ack, bit1,
bit0, crc)
```

canResetErrors

The canResetErrors call resets the CAN controller error counters.

Visual Basic Declaration:

Public Sub canResetErrors()

Call canResetErrors

EXAMPLES:

Call canResetErrors

canPurgeBuffer

The canPurgeBuffer call is a frame based function which purges CAN Controller message object buffers. The message object buffer to purge will be selected based on the supplied mode and 32-bit message object Arbitration ID register value.

Visual Basic Declaration:

Public Sub canPurgeBuffer(ByVal mode As Byte, ByVal msgarb As Long)

Call canPurgeBuffer(mode, msgarb)

WHERE:

mode

=		The method to use to determine which message object buffer to purge.
=	0	Purge all message object buffers.
=	1	Purge a transmit message object buffer whose arbitration id matches the passed parameter.
=	2	Purge a receive message object buffer whose arbitration id matches the passed parameter.
=	3	Purge all transmit message object buffers.
=	4	Purge all receive message object buffers.

msgarb

=		The 32-bit message object arbitration ID register value. Used to determine which message object buffer to purge.

EXAMPLES:

Call canPurgeBuffer(3, 0) ..... Purge all transmit message object buffers.
Call canPurgeBuffer(2, &H12345678) ..... Purge the message object buffer whose
..... arbitration ID register value is &H12345678.

canPurgeBufferExt

The canPurgeBufferExt call is an extended message format, frame based function which purges CAN Controller message object buffers. The message object buffer to purge will be selected based on the supplied mode and 29-bit message ID parameters.

Visual Basic Declaration:

Public Sub canPurgeBufferExt (ByVal mode As Byte, ByVal msgid As Long)

Call canPurgeBufferExt(mode, msgid)

WHERE:

mode

=		The method to use to determine which message object buffer to purge.
=	0	Purge all message object buffers.
=	1	Purge a transmit message object buffer whose 29-bit message ID matches the passed parameter.
=	2	Purge a receive message object buffer whose 29-bit message ID matches the passed parameter.
=	3	Purge all transmit message object buffers.
=	4	Purge all receive message object buffers.

msgid

=	29-bit message ID.

EXAMPLES:

Call canPurgeBufferExt(3, 0) ..... Purge all transmit message objects.
Call canPurgeBufferExt(2, &H12345678) ..... Purge the message object buffer
..... whose 29-bit message id is &H12345678.

recvCanFrames

The recvCanFrames call is a frame based function which receives the specified number of frames from message objects 1 - 14 buffers. The message object which receives the frames will be selected based on the supplied 32-bit message object Arbitration ID register value. An error will be returned if a receive message object with a matching Arbitration ID is not found or could not be dynamically assigned. If the number of frames specified hasn't been received before the timeout period expires, a error will be returned. The number of frames received will also be returned in the numframes variable.

A message object 1 - 14 frame is 9 bytes long and contains the following fields:

Message configuration - 1 Byte
Data - 8 Bytes

Visual Basic Declaration:

Public Sub recvCanFrames(ByVal msgarb As Long, ByRef numframes As Integer, frames() As Byte, ByVal timeout As Double)

Call recvCanFrames(msgarb, numframes, frames, timeout)

WHERE:

msgarb
= The 32-bit message object arbitration ID register value. Used to determine which message object to receive from.

numframes
= The number of frames to receive. This variable will contain the number of frames received after the call has been made.

frames
= The frame data array to placed received frames into. The array size must be at least the number of frames times 9 bytes long. Array size cannot exceed 32767 bytes.

timeout
= 0 to 65 seconds. The time to wait in seconds for the specified number of frames to be received.

EXAMPLES:

Dim numframes As Integer
Dim frames(18) As Byte
Call recvCanFrames(&H12345678, numframes, frames, 1.0) . Receive two frames ..... from a message object whose arbitration ID register value is &H12345678.

recvCanFramesExt

The recvCanFramesExt call is an extended message format, frame based function which receives the specified number of frames from message objects 1 - 14 buffers. The message object which receives the frames will be selected based on the supplied 29-bit message ID. An error will be returned if a receive message object with matching message ID is not found or could not be dynamically assigned. An error will also be returned if the number of frames specified hasn't been received before the timeout period expires. The actual number of frames received will be returned in the numframes variable.

Each message object 1- 14 frame is 9 bytes long and contains the following fields:

Number of data bytes in frame - 1 byte
Data - 8 bytes

Visual Basic Declaration:

Public Sub recvCanFramesExt(ByVal msgid As Long, ByRef numframes As Integer, frames() As Byte, ByVal timeout As Double)

Call recvCanFramesExt(msgid, numframes, frames, timeout)

WHERE:

msgid

= 29-bit message ID. Used to determine which message object to receive from.

numframes

= The number of frames to receive. This variable will contain the number of frames received after the call has been made.

frames

= The frame data array to write the received frames to. The array size must be at least the number of frames times 9 bytes long. The array size cannot exceed 32767 bytes.

timeout

= 0 to 65 seconds. The time to wait in seconds for the specified number of frames to be received.

EXAMPLES:

```
Dim numframes As Integer
Dim frames(0 to 17) As Byte
numframes = 2

Call recvCanFramesExt(&H12345678, numframes, frames, 1.0) ..... Receive
 ..... two frames from a message object
 ..... whose 29-bit message ID is &H12345678.
```

recvCanFrame15

The recvCanFrame15 call is a frame based function which receives the specified number of frames from Message Object 15. If the number of frames specified hasn't been received before the timeout period expires, a timeout error will be returned. The number of frames received will also be returned in the numframes variable.

A message object 15 frame is 13 bytes long and contains the following fields:

32-bit Arbitration ID register value - 4 Bytes
Message configuration register value - 1 Byte
Data - 8 Bytes

Visual Basic Declaration:

Public Sub recvCanFrame15(ByRef numframes As Integer, frames() As Byte, ByVal timeout As Double)

Call recvCanFrame15(numframes, frames, timeout)

WHERE:

numframes

= The number of frames to receive. This variable will contain the number of frames received after the call has been made.

frames

= The frame data array to place received frames into. The frame array size must be at least the number of frames times 13 bytes long. Array size cannot exceed 32767 bytes.

timeout

= 0 to 65 seconds. The time to wait in seconds for the specified number of frames to be received

EXAMPLES:

```
Dim numframes As Integer
Dim frames(26) As Byte

numframes = 2
Call recvCanFrame15(numframes, frames, 1.0)..... Receive two frames
 ..... from message object 15.
```

recvCanFrame15Ext

The recvCanFrames15Ext call is an extended message format, frame based function which receives the specified number of frames from message object 15 buffer. A timeout error will be returned if the number of frames specified hasn't been received before the timeout period expires. The actual number of frames received will be returned in the numframes variable.

Each message object 15 frame is 13 bytes long and contains the following fields:

29-bit Message ID - 4 bytes
Number of data bytes in frame - 1 byte
Data - 8 bytes

Visual Basic Declaration:

Public Sub recvCanFrame15Ext(ByRef numframes As Integer, frames() As Byte, ByVal timeout As Double)

Call recvCanFrame15Ext(numframes, frames, timeout)

WHERE:

numframes

= The number of frames to receive. This variable will contain the number of frames received after the call has been made.

frames

= The frame data array to write the received frames to. The array size must be at least the number of frames times 13 bytes long. The array size cannot exceed 32767 bytes.

timeout

= 0 to 65 seconds. The time to wait in seconds for the specified number of frames to be received.

EXAMPLES:

```
Dim numframes As Integer
Dim frames(0 to 25) As Byte
numframes = 2

Call recvCanFrame15Ext(numframes, frames, 1.0) ..... Receive two
 ..... frames from message object 15.
```

sendCanFrames

The sendCanFrames call is a frame based function which transmits the specified number of frames from message objects 1 -14. The message object which transmits the frames will be selected based on the supplied 32-bit message object Arbitration ID register value. An error will be returned if a transmit message object with a matching Arbitration ID is not found or could not be dynamically assigned. If the number of frames specified haven't been loaded into the message object's buffer before the timeout period expires, a error will be returned. The number of frames loaded into the message object buffer will also be returned in the numframes variable.

A message object 1 - 14 frame is 9 bytes long and contains the following fields:

Message configuration register value - 1 Byte
Data - 8 Bytes

Visual Basic Declaration:

Public Sub sendCanFrames(ByVal msgarb As Long, ByRef numframes As Integer, frames() As Byte, ByVal timeout As Double)

Call sendCanFrames(msgarb, numframes, frames, timeout)

WHERE:

msgarb
= The 32-bit message object arbitration ID register value. Used to determine which message object to transmit from.

numframes
= The number of frames to transmit. This variable will contain the number of frames transferred after the call has been made.

frames
= The frame data array containing the frames to transmit. The array must be at least the number of frames times 9 bytes long. Array size cannot exceed 32767 bytes.

timeout
= 0 to 65 seconds. The time to wait in seconds for the specified number of frames to be loaded into the message object buffer.

EXAMPLES:

```
Dim numframes As Integer
Dim frames(9) As Byte

numframes = 1
frames(0) = &H8C - ‘message object configuration register value
frames(1) = &H01
frames(2) = &H23
frames(3) = &H45
frames(4) = &H67
frames(5) = &H89
frmaes(6) = &HAB
frames(7) = &HCD
frames(8) = &HEF

Call sendCanFrames(&H12345678, numframes, frames, 1.0) ..... Send 1 frame
 ..... using the object whose arbitration ID registration value is &H12345678.
```

sendCanFramesExt

The sendCanFramesExt call is an extended message format, frame based function which transmits the specified number of frames from message objects 1 - 14 buffers. The message object which transmits the frames will be selected based on the supplied 29-bit message ID. An error will be returned if a transmit message object with matching message ID is not found or could not be dynamically assigned. An error will also be returned if the number of frames specified hasn't been loaded into the message object's buffer before the timeout period expires. The actual number of frames loaded into the message object buffer will be returned in the numframes variable.

Each message object 1- 14 frame is 9 bytes long and contains the following fields:

Number of data bytes in frame - 1 byte
Data - 8 bytes

Visual Basic Declaration:

Public Sub sendCanFramesExt(ByVal msgid As Long, ByRef numframes As Integer, frames() As Byte, ByVal timeout As Double)

Call sendCanFramesExt(msgid, numframes, frames, timeout)

WHERE:

msgid

= 29-bit message ID. Used to determine which message object to transmit from.

numframes

= The number of frames to transmit. This variable will contain the number of frames transferred after the call has been made.

frames

= The frame data array containing the frames to transmit. The array size must be at least the number of frames times 9 bytes long. The array size cannot exceed 32767 bytes.

timeout

= 0 to 65 seconds. The time to wait in seconds for the specified number of frames to be loaded into the message object's buffer.

EXAMPLES:

```
Dim numframes As Integer
Dim frames(0 to 8) As Byte
numframes = 1
frames(0) = 8 ‘Number of data bytes in frame
frames(1) = &H01
frames(2) = &H23
frames(3) = &H45
frames(4) = &H67
frames(5) = &H89
frames(6) = &HAB
frames(7) = &HCD
frames(8) = &HEF

Call sendCanFramesExt(&H12345678, numframes, frames, 1.0) ..... Send
 ..... one frame from a message object
 ..... whose 29-bit message ID is &H12345678.
```

recvCanMsgobj

The recvCanMsgobj functional call is a register based function which receives one complete message from the selected message object. If a message hasn't been received after the timeout period elapses, a timeout error will be returned.

The selected message object needs to be properly configured before using this call. The receive interrupt enable in the Message Object's Control 0 register should be disabled. Also, the direction bit in the Message Object's Configuration register needs to be set to zero to receive.

Visual Basic Declaration:

Public Sub recvCanMsgobj(ByVal msgobj As Byte, ByRef msgcon0 As Byte, ByRef msgcon1As Byte, ByRef msgarb As Long, ByRef msgconfig As Byte, msgdata() As Byte, ByVal timeout As Double)

Call recvCanMsgobj(msgobj, msgcon0, msgcon1, msgarb, msgconfig, msgdata, timeout)

WHERE:

msgobj
= 1 to 15. The CAN message object number to read.

msgcon0
= Data from the CAN Message Control 0 Register.

msgcon1
= Data from the CAN Message Control 1 Register.

msgarb
= Data from the CAN Message Arbitration Registers, 4 bytes.

msgconfig
= Data from the Can Message Configuraton Register.

msgdata
= Data array from the CAN Message Data Registers, 8 bytes.

timeout
= 0 to 65 seconds. The time to wait in seconds for a message.

EXAMPLES:

```
Dim msgcon0 As Byte
Dim msgcon1 As Byte
Dim msgarb As Long
Dim msgconfig As Byte
Dim msgdata(7) As Byte

Call recvCanMsgobj(1, msgcon0, msgcon1, msgarb, msgconfig, msgdata, 1.0) ....
 ..... Read message from message object #1.
```

sendCanDataobj

The sendCanDataobj functional call is a register based function call which loads new data values into the selected CAN message object's data registers and then transmits that message object. The selected message object needs to be properly configured before using this call. The transmit interrupt enable in the Message Object's Control 0 register should be disabled. Also, the direction bit in the Message Object's Configuration register needs to be set to one to transmit.

Visual Basic Declaration:

Public Sub sendCanDataobj(ByVal msgobj As Byte, smsg() As Byte, ByVal timeout As Double)

Call sendCanDataobj(msgobj, smsg(), timeout)

WHERE:

msgobj

= 1 to 14. The CAN Controller Message Object Number.

smsg()

= The data array to transmit. Valid data array size - 1 to 8 bytes.

timeout

= 0 to 65 seconds to wait for an idle bus.

EXAMPLES:

```
Dim smsg(3) As Byte
smsg(0)=0
smsg(1)=1
smsg(2)=2
smsg(3)=3

Call sendCanDataobj(1, smsg, 1.0) ..... Transmit message object #1.
```

sendCanMsgobj

The sendCanMsgobj functional call is a register based function which transmits the selected CAN message object previously set up with the CAN register functions. The selected message object needs to be properly configured before using this call. The transmit interrupt enable in the Message Object's Control 0 register should be disabled. Also, the direction bit in the Message Object's Configuration register needs to be set to one to transmit.

Visual Basic Declaration:

Public Sub sendCanMsgobj(ByVal msgobj As Byte, ByVal timeout As Double)

Call sendCanMsgobj(msgobj, timeout)

WHERE:

msgobj

= 1 to 14. The CAN Controller Message Object Number.

timeout

= 0 to 65 seconds to wait for an idle bus.

EXAMPLES:

Call sendCanMsgobj(1, 1.0) ..... Transmit message object #1.

SPI Functional Calls

getSpiParams

The getSpiParams functional call retrieves the current Serial Peripheral Interface (SPI) port settings.

Visual BASIC Declaration:

Public Sub getSpiParams(ByVal Index As Integer, ParamValue As Long)

Call getSpiParams(Index, ParamValue)

WHERE:

Index

= Index of the parameter to get.

ParamValue

= Value the parameter is set to.

INDEX#	PARAMETER NAME	PARAMETER VALUE
1	Bits per transfer	8 thru 16 bits
2	Clock Polarity	0 = Inactive state of SCK is low 1 = Inactive state of SCK is high
3	Clock Phase	0 = Data is captured on the leading edge of SCK and changed on the following edge of SCK. 1 = Data is changed on the leading edge of SCK and captured on the following edge of SCK.
4	Serial Clock Baud Rate	SCK frequency in Hz.
5	Delay before SCK	SCK delay in nanoseconds.
6	Delay after Transfer	Transfer delay in nanoseconds.
7	Peripheral Chip Select State	0 = Active Low. 1 = Active High.
8	Peripheral Chip Select Mode	0 = Chip select deselected between transfers. 1 = Chip select asserted between transfers.
9	Default Transmit Data	Data transmitted during the spiReceive function.

Bits Per Transfer	=	The number of data bits transferred for each serial transfer.
Clock Polarity	=	Determines the inactive state of SCK.
Clock Phase	=	Determines which edge of SCK causes data to change and which edge causes data to be captured.
Serial Clock Baud Rate	=	Serial clock (SCK) frequency.
Delay Before SCK	=	Delay from the peripheral chip select valid to SCK transition for serial transfers.
Delay After Transfer	=	SPI delay after each serial transfer within a burst.
Peripheral Chip Select State	=	Active state of the SPI peripheral chip select output.
Peripheral Chip Select Mode	=	Determines if the SPI peripheral chip select is asserted or deselected between serial transfers within a burst.
Default Transmit Data	=	Default data transmitted during the spiReceive function.

EXAMPLES:

```
Dim ParamValue As Long

Call getSpiParams(1,ParamValue) ..... Get Bits per Transfer value.
```

setSpiParams

The setSpiParams functional call sets the Serial Peripheral Interface (SPI) port parameters. The MSP board reverts back to the default SPI parameter settings on Testhead power-up. A TClear or MSPReset functional call will also set the SPI parameters to their default values.

Visual BASIC Declaration:

Public Sub setSpiParams(ByVal Index As Integer, ByVal ParamValue As Long)

Call setSpiParams(Index, ParamValue)

WHERE:

Index

= Index of the parameter to set.

ParamValue

= Value to set the parameter to.

INDEX#	PARAMETER NAME	PARAMETER VALUE
1	Bits per Transfer	8 thru 16 (Default = 8).
2	Clock Parity	0 = Inactive state of SCK is low (Default). 1 = Inactive state of SCK is high.
3	Clock Phase	0 = Data is captured on the leading edge of SCK and changed on the following edge of SCK. 1 = Data is changed on the leading edge of SCK and captured on the following edge of SCK (Default).
4	Serial Clock Baud Rate	SCK clock frequency in Hz. (Default = 2097152 Hz). 33026 Hz <= SCK => 4194304 Hz with a 16.78 MHz system clock.
5	Delay before SCK	SCK Delay in nanoseconds. (Default = 0 : 1/2 SCK period). SCK Clock Range: 0 = 1/2 SCK period -or- 59 nS <= SCK Delay => 7629 nS with a 16.78 MHz system clock.

6	Delay after Transfer	Transfer Delay in nanoseconds (Default = 0 : 1000 nS, referred to as the Standard Delay). Transfer Delay Range: 0 = 1000 nS -or- 1907nS <= Transfer Delay => 488281 nS with a 16.78 MHz system clock.
7	Peripheral Chip Select State	0 = Active Low (Default). 1 = Active High.
8	Peripheral Chip Select Mode	0 = Deselected between transfers (Default). 1 = Asserted between transfers.
9	Default Transmit Data	Data transmitted during the spiReceive function. (Default = 0) = &H0000 to &HFFFF.

Bits Per Transfer	=	The number of data bits transferred for each serial transfer.
Clock Polarity	=	Determines the inactive state of SCK.
Clock Phase	=	Determines which edge of SCK causes data to change and which edge causes data to be captured.
Serial Clock Baud Rate	=	Serial clock (SCK) frequency.
Delay Before SCK	=	Delay from the peripheral chip select valid to SCK transition for serial transfers.
Delay After Transfer	=	SPI delay after each serial transfer within a burst.
Peripheral Chip Select State	=	Active state of the SPI peripheral chip select output.
Peripheral Chip Select Mode	=	Determines if the SPI peripheral chip select is asserted or deselected between serial transfers within a burst.
Default Transmit Data	=	Default data transmitted during the spiReceive function.

EXAMPLES:

Call setSpiParams(1,8) ..... Set Bits per Transfer to 8 bits.

spiTransmit

The spiTransmit functional call transmits messages using the Serial Peripheral Interface (SPI) port on the MSP board.

Visual Basic Declaration:

Public Sub spiTransmit(smsg() As Integer, ByVal smsgSize As Long)

Call spiTransmit(smsg, smsgSize)

WHERE:

smsg
= Array holding the message to send

smsgSize
= Number of words to send

EXAMPLES:

```
Dim smsg(1 To 4) As Integer
smsg(1) = &H0123
smsg(2) = &H4567
smsg(3) = &H89AB
smsg(4) = &HCDEF

Call spiTransmit(smsg, 4) ..... Transmit 4 words of information.
```

spiReceive

The spiReceive functional call receives messages using the Serial Peripheral Interface (SPI) port on the MSP board.

Visual Basic Declaration:

Public Sub spiReceive(rmsg() As Integer, ByRef rmsgSize As Long)

Call spiReceive(rmsg, rmsgSize)

WHERE:

rmsg
= Buffer containing the received message

rmsgSize
= The number of words to receive. This parameter will be updated with the number of actual words received.

EXAMPLES:

```
Dim rmsg(1 To 4) As Integer
Dim rmsgSize As Long
rmsgSize = 4

Call spiReceive(rmsg, rmsgSize) ..... Receive 4 words of information.
```

spiTransceive

The spiTransceive functional call sends and receives messages using the Serial Peripheral Interface (SPI) port on the MSP board.

Visual Basic Declaration:

Public Sub spiTransceive(smsg() As Integer, rmsg() As Integer, ByRef msgSize As Long)

Call spiTransceive(smsg, rmsg, msgSize)

WHERE:

smsg
= Buffer holding the message to send.

rmsg
= Buffer containing the received message.

msgSize
= Pointer to the variable containing the number of words to transmit and receive (must be <= size of both smsg and rmsg buffers). This parameter will be updated with the number of actual words received.

EXAMPLES:

```
Dim smsg(1 To 4) As Integer
Dim rmsg(1 To 4) As Integer
Dim msgSize As Long
msgSize = 4
smsg(1) = &H0123
smsg(2) = &H4567
smsg(3) = &H89AB
smsg(4) = &HCDEF

Call spiTransceive(smsg, rmsg, msgSize) ..... Transmit and receive
 ..... 4 words of information.
```

spiTransmitBuffer

The spiTransmitBuffer functional call transmits the contents of a previously loaded MSP buffer using the Serial Peripheral Interface (SPI) port on the MSP board. Messages may be downloaded to a MSP buffer using the mspDownloadBuffer functional call.

Visual Basic Declaration:

Public Sub spiTransmitBuffer(ByVal bufferNumber As Integer)

Call spiTransmitBuffer(bufferNumber)

WHERE:

bufferNumber

= 0 to 19. MSP buffer number holding the message to send.

EXAMPLES:

Call spiTransmitBuffer(0) ..... Transmit the contents of MSP buffer #0.

J1850 VPW Functional Calls

enableJ18560VPWChannelPeriodic

The enableJ1850VPWChannelPeriodic functional call enables or disables periodic message transmission on a VPW-J1850 channel. The periodic message transmission for a channel may be enabled or disabled at any time. Periodic messages will only be transmitted on enabled channels after the periodic timer has been started using the sendJ1850VPWPeriodic functional call. It is up to the user to make sure that the periodic message has been downloaded using the J1850VPWDownloadBuffer functional call before the periodic timer is started and the periodic message transmission for a channel is enabled. Disabling, then re-enabling the periodic message transmission for a channel causes its message counters and period timer counters to be reset to the values programmed by the setJ1850VPWPeriodicParams functional call. Period message transmissions for all channels are disabled on Testhead power-up or whenever a Tclear or resetJ1850VPW functional call occurs.

Visual Basic Declaration:

Public Sub enableJ1850VPWChannelPeriodic(ByVal channel As Integer, ByVal messageNum As Integer, ByVal state As Integer)

Call enableJ1850VPWChannelPeriodic(channel, messageNum, state)

WHERE:

channel

=	0	Channel 0 on VPW-J1850 board 0
=	1	Channel 1 on VPW-J1850 board 0
.	.	.
.	.	.
=	15	Channel 15 on VPW-J1850 board 0
=	16	Channel 0 on VPW-J1850 board 1, etc. to 4 boards/system

msgNum

=		Channels periodic message number to enable/disable
=	-1	both channel's periodic message numbers.
=	0	or 1, channel's periodic message number.

state

=		Enable or disable period message transmission.
=	0	Disable periodic message transmission.
=	1	Enable periodic messages transmission.

EXAMPLES:

Call enableJ1850VPWChannelPeriodic (0, -1, 1)..... .....
..... Enable message transmissions for both
.....channel #0's periodic messages.

getJ1850VPWClock

The getJ1850VPWClock function gets the system clock frequency of the MC68331 microprocessor on the VPW-J1850 board.

Visual Basic Declaration:

Public Sub getJ1850VPWClock(ByVal board As Integer, ByRef frequency As Long)

Call getJ1850VPWClock(board, frequency)

WHERE:

board

=		VPW-J1850 board number.
=	0	VPW-J1850 board 0.
=	1	VPW-J1850 board 1.
=	2	VPW-J1850 board 2.
=	3	VPW-J1850 board 3.

frequency

=		The VPW-J1850 clock frequency in Hz.

EXAMPLES:

```
Dim frequency As Long

Call getJ1850VPWClock (0, frequency) ..... Get the system clock
 ..... frequency from VPW-J1850 board #0.
```

getJ1850VPWException

The getJ1850VPWException function gets the MC68331 microprocessor exception frame data from the VPW-J1850 board. An exception occurs when a special condition occurs that pre-empts normal processing on the VPW-J1850 board. The 109:030 (VPW-J1850 Exception Frame) error message will be generated whenever an exception occurs on a VPW-J1850 board. The exception frame data will be saved until a board reset occurs. The MC68331 will resume normal program execution after an exception occurs. This function is normally not used and is provided only as a debugging aid. The exception frame contains the following MC68331 microprocessor information:

CPU Register Contents:

D0 Register Contents (first 4 bytes of frame)
D1 Register Contents (4 bytes)
D2 Register Contents (4 bytes)
D3 Register Contents (4 bytes)
D4 Register Contents (4 bytes)
D5 Register Contents (4 bytes)
D6 Register Contents (4 bytes)
D7 Register Contents (4 bytes)
A0 Register Contents (4 bytes)
A1 Register Contents (4 bytes)
A2 Register Contents (4 bytes)
A3 Register Contents (4 bytes)
A4 Register Contents (4 bytes)
A5 Register Contents (4 bytes)
A6 Register Contents (4 bytes)
A7 Register Contents (4 bytes)
Status Register (2 bytes)
Exception Vector Offset (4 bytes)

CPU Exception Stack frame:

Status Register (2 bytes)
Program Counter High (2 bytes)
Program Counter Low (2 bytes)
Format(upper nibble)/Vector Offset (2 bytes)

Other Processor State Information (0, 4 or 16 bytes)

Visual Basic Declaration:

Public Sub getJ1850VPWException(ByVal board As Integer, frame() As Byte, ByRef frameSize As Integer)

Call getJ1850VPWException(board, frame, frameSize)

WHERE:

board

=		VPW-J1850 board number
=	0	VPW-J1850 board 0
=	1	VPW-J1850 board 1
=	2	VPW-J1850 board 2
=	3	VPW-J1850 board 3

frame

=		The array to write the exception frame to

frameSize

=		The size of the frame array in bytes. This parameter will be updated with the actual VPW-J1850 exception frame size in bytes.
=	94	(minimum frame array size needed)

EXAMPLES:

```
Dim frame(0 To 93) As Byte
Dim frameSize As Integer
frameSize = 94

Call getJ1850VPWException (0, frame, frameSize) ..... Get the exception
 ..... frame from VPW-J1850 board #0.
```

getJ1850VPWParams

The getJ1850VPWParams functional call is used to get the value of a parameter used by the VPW-J1850 functions for the specified channel.

Visual BASIC Declaration:

Public Sub getJ1850VPWParams(ByVal channel As Integer, ByVal index As Integer, ByRef value As Integer)

Call getJ1850VPWParams(channel, index, value)

WHERE:

Channel

=	0	Channel 0 on VPW-J1850 board 0
=	1	Channel 1 on VPW-J1850 board 0
.	.	. . . .
.	.	. . . .
=	15	Channel 15 on VPW-J1850 board 0
=	16	Channel 0 on VPW-J1850 board 1, etc. to 4 boards/system

Index

= Index of the parameter to get.

Value

= Current parameter setting.

Index #	Parameter name	Parameter value
1	DLC Config	Configuration byte
2	Set Speed	0 = Normal speed (default) 1 = High speed
4	Receive Timeout	0 to 32767 mS (default = 100)
6	Idle Timeout	0 to 32767 mS (default = 10)
10	Disable Retry	0 = Retry enabled 1 = Retry disabled (default)
11	Reconfigure	0 = Do not configure DLC (default) 1 = Configure DLC before xmit
12	Issue Reset	0 = Do not reset DLC before xmit (default) 1 = Reset DLC before xmit

13	Flush Previous	0 = Do not clear receive buffer before xmit 1 = Clear receive buffer before xmit (default)
14	Issue Flush	0 = Do not issue flush message 1 = Issue flush message before xmit (def)
15	Check Code	0 = Do not check completion code 1 = Check completion code after xmit, flush RFIFO. (def) 2 = Check completion code, do not flush RFIFO.
17	XMIT Completion Code	= Number of times to attempt to receive the transmit completion code.

Dlcp Config = Current MC68HC58 DLC configuration byte.

Set Speed = The MC68HC58 DLC configuration byte register high speed bit setting.

Receive Timeout = The time to wait for a message to be received from the product.

Idle Timeout = The time to wait for an idle line before transmitting.

Disable Retry = Terminate auto-retry command setting which is written to the MC68HC58 DLC command register before each transmission.

Reconfigure = Reconfigure the DLC command setting. The MC68HC58 DLC is reconfigured by writing the DLC Config parameter to the DLC configuration register before each transmission.

Issue Reset = Reset command setting. The MC68HC58 DLC transmitter is reset when the reset command is written to the DLC command register before each transmission. This causes the transmitter to be reset. Any transmission in progress is terminated and the TxFIFO buffer is flushed.

Flush Previous = Flush circular receive buffer command. The J1850 channel's circular receive buffer is flushed before each transmission.

Issue Flush = RxFIFO command setting which is written to the MC68HC58 DLC command register before each transmission. This command flushes the current frame except for the completion code from the MC68HC58 DLC RxFIFO.

Check Code = Firmware waits for a completion code after a transmit and returns an error based on the completion code. If the parameter value = 1, the J1850 channel's circular receive buffer is flushed before each transmission.

XMIT Comp. Code = The number of times to attempt to receive the transmit completion code when transmitting.

EXAMPLES:

```
Dim value As Integer

Call getJ1850VPWParams(0, 12, value) ..........................................
 ..... Get the issue reset setting for channel 0.
```

getJ1850VPWPeriodicParams

The getJ1850VPWPeriodicParams function gets the VPW-J1850 channel's currently programmed periodic message transmission parameters.

Visual Basic Declaration:

Public Sub getJ1850VPWPeriodicParams(ByVal channel As Integer, ByVal messageNum As Integer, ByRef bufferNum As Integer, ByRef numTimes As Integer, ByRef period As Integer)

Call getJ1850VPWPeriodicParams(channel, messageNum, bufferNum, numTimes, period)

WHERE:

channel

=		VPW-J1850 channel number
=	0	Channel 0 on VPW-J1850 board 0
=	1	Channel 1 on VPW-J1850 board 0
.	.	. . . .
.	.	. . . .
=	15	Channel 15 on VPW-J1850 board 0
=	16	Channel 0 on VPW-J1850 board 1, etc. to four boards per system

msgNum

=		Channels periodic message number parameters to get
=	0	Get the channel's periodic message number #0 parameters
=	1	Get the channel's periodic message number #1 parameters

bufferNum

=		VPW-J1850 buffer number to transmit

numTimes

=		Number of times to transmit the message

period

=		Time between buffer transmissions

EXAMPLES:

```
Dim bufferNum As Integer
Dim numTimes As Integer
Dim period As Integer
Call getJ1850VPWPeriodicParams (0, 0, bufferNum, numTimes, period) .....
 ..... Get the periodic message parameters
 ..... for channel #0 - message#0.
```

getJ1850VPWRecvStatus

The getJ1850VPWRecvStatus function gets the receiver buffer status for all VPW-J1850 channels on the passed VPW-J1850 board. If a message is available in a channel's buffer, the corresponding bit in the status parameter is set. Channel 15 is the MSB and Channel 0 is the LSB of the status parameter.

Visual Basic Declaration:

Public Sub getJ1850VPWRecvStatus(ByVal board As Integer, ByRef status As Integer)

Call getJ1850VPWRecvStatus(board, status)

WHERE:

board

=		VPW-J1850 board number to reset
=	0	VPW-J1850 board 0
=	1	VPW-J1850 board 1
=	2	VPW-J1850 board 2
=	3	VPW-J1850 board 3

status

=		Receiver status

EXAMPLES:

```
Dim status As Integer

Call getJ1850VPWRecvStatus (0, status) .....
 ..... Get the receiver status for all
 ..... VPW-J1850 board #0 channels.
```

getJ1850VPWVersion

The getJ1850VPWVersion function gets the firmware revision number from the VPW-J1850 board.

Visual Basic Declaration:

Public Sub getJ1850VPWVersion(ByVal board As Integer, version As String)

Call getJ1850VPWVersion(board, version)

WHERE:

board

=		VPW-J1850 board number
=	0	VPW-J1850 board 0
=	1	VPW-J1850 board 1
=	2	VPW-J1850 board 2
=	3	VPW-J1850 board 3

version

=		String containing the VPW-J1850 firmware version string

EXAMPLES:

```
Dim version As String

Call getJ1850VPWVersion (0, version) ..... Get the firmware version
 ..... of VPW-J1850 board #0.
```

J1850VPWDownloadBuffer

The J1850VPWDownloadBuffer function downloads data into the selected VPW-J1850 buffer which may be transmitted at a future time. Thirty-two buffers are available on each VPW-J1850 board. A 109:007 (VPW-J1850 Out of Memory) error message will be returned if the number of bytes of download data exceeds the amount of free VPW-J1850 memory.

Visual Basic Declaration:

Public Sub J1850VPWDownloadBuffer(ByVal board As Integer, ByVal bufferNumber As Integer, data() As Byte, ByVal dataSize As Long)

Call J1850VPWDownloadBuffer(board, bufferNumber, data, dataSize)

WHERE:

board

=		VPW-J1850 board number
=	0	VPW-J1850 board 0
=	1	VPW-J1850 board 1
=	2	VPW-J1850 board 2
=	3	VPW-J1850 board 3

bufferNumber

=		The VPW-J1850 board buffer number to set load
=	0	to 31

data

= Array containing the data to download

dataSize

= The number of bytes to download

EXAMPLES:

```
Dim data(0 To 3) As Byte
data(0) = &H6C
data(1) = &HA6
data(2) = &HF0
data(3) = &H3F

Call J1850VPWDownloadBuffer (0, 0, data, 4) ..... Download data
 ..... array to buffer #0 on VPW-J1850 board #0.
```

recvJ1850VPW

The recvJ1850VPW functional call receives a message using the specified J1850 channel on the VPW-J1850 board. The message must be disassembled by the caller as the function receives the message transparently.

Visual Basic Declaration:

Public Sub recvJ1850VPW(ByVal channel As Integer, rmsg() As Byte, ByRef rmsgSize As Integer, ByVal timeout As Double)

Call recvJ1850VPW(channel, rmsg, rmsgSize, timeout)

WHERE:

channel

=		VPW-J1850 channel number
=	0	Channel 0 on VPW-J1850 board 0
=	1	Channel 1 on VPW-J1850 board 0
.	.	. . . .
.	.	. . . .
=	15	Channel 15 on VPW-J1850 board 0
=	16	Channel 0 on VPW-J1850 board 1, etc. to four boards per system

rmsg

=		Array containing the received message

rmsgSize

=		The number of bytes to receive and must be <= the size of the rmsg array. This parameter will be updated with the actual number of bytes received.

timeout

=		The time to wait in seconds for a message to be received.
=	-1	Check for message immediately
=	0	Use default receive timeout
=	>0	and <= 60, the time to wait in seconds for a message to be received.

EXAMPLES:

```
Dim rmsg(0 To 14) As Byte
Dim rmsgSize As Integer
rmsgSize = 15

Call recvJ1850VPW(0, rmsg, rmsgSize, 0) ..... Receive a message on VPW-J1850 channel #0.
```

resetJ1850VPW

The resetJ1850VPW function resets the VPW-J1850 board. All of the J1850 channel parameters are reset to their default values and all download message buffers are cleared.

Visual Basic Declaration:

Public Sub resetJ1850VPW(ByVal board As Integer)

Call resetJ1850VPW(board)

WHERE:

board

=		VPW-J1850 board number to reset.
=	-1	All VPW-J1850 boards.
=	0	VPW-J1850 board 0.
=	1	VPW-J1850 board 1.
=	2	VPW-J1850 board 2.
=	3	VPW-J1850 board 3.

EXAMPLES:

Call resetJ1850VPW(0) ..... Reset VPW-J1850 board #0.

sendJ1850VPW

The sendJ1850VPW functional call sends a message using the specified J1850 channel on the VPW-J1850 board. The message must be fully assembled by the caller as the function transmits the message transparently.

Visual Basic Declaration:

Public Sub sendJ1850VPW(ByVal channel As Integer, smsg() As Byte, ByVal smsgSize As Integer, ByVal timeout As Double)

Call sendJ1850VPW(channel, smsg, smsgSize, timeout)

WHERE:

channel

=		VPW-J1850 channel number.
=	0	Channel 0 on VPW-J1850 board 0.
=	1	Channel 1 on VPW-J1850 board 0.
.	.	. . . .
.	.	. . . .
=	15	Channel 15 on VPW-J1850 board 0.
=	16	Channel 0 on VPW-J1850 board 1, etc. to four boards per system.

smsg

=		The array holding the message to send.

smsgSize

=		Size of the message buffer in bytes.

timeout

=		The time to wait in seconds for the serial line to be idle.
=	-1	Try to send message immediately - return if line not idle.
=	0	Use default idle timeout.
=	>0	and <= 60, the time to wait in seconds for the serial line to be idle.

EXAMPLES:

```
Dim smsg(0 To 3) As Byte
smsg (0) = &H6C
smsg (1) = &HA6
smsg (2) = &HF0
smsg (3) = &H3F

Call sendJ1850VPW (0, smsg, 4, 0) ..... Transmit smsg array
 ..... on VPW-J1850 channel #0.
```

sendJ1850VPWBuffer

The sendJ1850VPWBuffer function sends a previously downloaded message over the specified J1850 channel on the VPW-J1850 board.

Visual Basic Declaration:

Public Sub sendJ1850VPWBuffer(ByVal channel As Integer, ByVal timeout As Double, ByVal bufferNum As Integer)

Call sendJ1850VPWBuffer(channel, timeout, bufferNum)

WHERE:

channel

=		VPW-J1850 channel number.
=	0	Channel 0 on VPW-J1850 board 0.
=	1	Channel 1 on VPW-J1850 board 0.
.	.	. . . .
.	.	. . . .
=	15	Channel 15 on VPW-J1850 board 0.
=	16	Channel 0 on VPW-J1850 board 1, etc. to four boards per system.

timeout

=		The time to wait in seconds for the serial line to be idle.
=	-1	Try to send message immediately - return if line not idle.
=	0	Use default idle timeout.
=	>0	and <= 60, the time to wait in seconds for the serial line to be idle.

bufferNum

=		The buffer number of the previously downloaded message to send.
=	0	to 31.

EXAMPLES:

Call sendJ1850VPWBuffer (0, 0, 0) ..... Transmit the contents of
..... VPW-J1850 buffer #0 on channel #0.

sendJ1850VPWPeriodic

The sendJ1850VPWPeriodic function starts or stops a VPW-J1850 board from periodically transmitting previously downloaded messages. The periodic timer may be started or stopped at any time. Periodic messages will only be transmitted on the channels that have been enabled using the enableJ1850VPWChannelPeriodic functional call. It is up to the user to make sure that the periodic message has been downloaded using the J1850VPWDownloadBuffer functional call before the periodic timer is started and the periodic message transmission for a channel is enabled. Stopping, then restarting the periodic timer causes the message counters and period timer counters to be reset to the values programmed by the setJ1850VPWPeriodicParams functional call. The period timer is turned off on Testhead power-up or whenever a Tclear or resetJ1850VPW functional call occurs.

Visual Basic Declaration:

Public Sub sendJ1850VPWPeriodic(ByVal board As Integer, ByVal state As Integer)

Call sendJ1850VPWPeriodic(board, state)

WHERE:

board

=		VPW-J1850 board number to reset.
=	0	VPW-J1850 board 0.
=	1	VPW-J1850 board 1.
=	2	VPW-J1850 board 2.
=	3	VPW-J1850 board 3.

state

=		Start or stop sending periodic messages.
=	0	Stop board from sending periodic messages.
=	1	Start sending periodic messages from board.

EXAMPLES:

Call sendJ1850VPWPeriodic (0, 1) ..... Start the periodic timer
..... on VPW-J1850 board #0.

setJ1850VPWParams

The setJ1850VPWParams functional call is used to set up the default parameters used by the VPW-J1850 functions for the specified channel. The VPW-J1850 board reverts back to the default parameter settings on Testhead power-up. A Tclear or resetJ1850VPW functional call will also reset the VPW-J1850 parameters to their default settings.

Visual BASIC Declaration:

Public Sub setJ1850VPWParams(ByVal channel As Integer, ByVal index As Integer, ByVal value As Integer)

Call setJ1850VPWParams(channel, index, value)

WHERE:

Channel

=	0	Channel 0 on VPW-J1850 board 0
=	1	Channel 1 on VPW-J1850 board 0
.	.	. . . .
.	.	. . . .
=	15	Channel 15 on VPW-J1850 board 0
=	16	Channel 0 on VPW-J1850 board 1, etc. to 4 boards/system

Index

= Index of the parameter to set.

Value

= Value to set the parameter to.

Index #	Parameter name	Parameter value
1	DLC Config	Configuration byte
2	Set Speed	0 = Normal speed (default) 1 = High speed
4	Receive Timeout	0 to 32767 mS (default = 100)
6	Idle Timeout	0 to 32767 mS (default = 10)
9	Direct Command	DLC Command and data byte
10	Disable Retry	0 = Retry enabled 1 = Retry disabled (default)

11	Reconfigure	0 = Do not configure DLC (default) 1 = Configure DLC before xmit
12	Issue Reset	0 = Do not reset DLC before xmit (default) 1 = Reset DLC before xmit
13	Flush Previous	0 = Do not clear receive buffer before xmit 1 = Clear receive buffer before xmit (default)
14	Issue Flush	0 = Do not issue flush message 1 = Issue flush message before xmit (def)
15	Check Code	0 = Do not check completion code 1 = Check completion code after xmit (default) 2 = Check completion code, do not flush RFIFO.
16	Command	1 = Reset transmitter. 2 = Flush the firmware receiver buffer
17	XMIT Completion Code	1 to 255. Number of times to attempt to receive the transmit completion code.

DlC Config = Writes the parameter value to the MC68HC58 DLC configuration byte register. (Should not be used under normal conditions).

Set Speed = Changes the value of the high speed control bit in the MC68HC58 DLC configuration byte register.

Receive Timeout = The time to wait for a message to be received from the product.

Idle Timeout = The time to wait for an idle line before transmitting.

Direct Command = Writes directly to the command and data registers of the MC68HC58 DLC. The upper byte of the parameter is written to the command register, and the lower byte is written to the data register.

Disable Retry = Writes terminate auto-retry command to the MC68HC58 DLC command register before each transmission.

Reconfigure = Reconfigures the MC68HC58 DLC by writing the DLC config parameter value to the MC68HC58 DLC configuration register before each transmission.

Issue Reset	=	Writes a reset command to the MC68HC58 DLC command register before each transmission. This causes the transmitter to be reset. Any transmission in progress is terminated and the TxFIFO buffer is flushed.
Flush Previous	=	Flushed the J1850 channel's circular receive buffer before each transmission.
Issue Flush	=	Writes a flush RxFIFO command to the MC68HC58 DLC command register before each transmission. This command flushes the current frame except for the completion code from the MC68HC58 DLC RxFIFO.
Check Code	=	Firmware waits for a completion code after a transmit and returns an error based on the completion code value. If the parameter value = 1, the J1850 channel's circular receive buffer is flushed before each transmission.
Command	=	Commands to either issue a transmitter reset command to the MC68HC58 DLC or flush the firmware circular receive buffer.
XMIT Comp. Code	=	The number of times to attempt to receive the transmit completion code when transmitting.

EXAMPLES:

Call setJ1850VPWParams(0, 12, 1) ..... Configure channel 0 to reset
..... the J1850 controller before each transmission.

setJ1850VPWPeriodicParams

The setJ1850VPWPeriodicParams function sets the VPW-J1850 channel's parameters for periodic message transmission. Each channel may be programmed to transmit two different messages periodically. Any parameter changes occur immediately if the periodic message transmissions are enabled and running. The VPW-J1850 board reverts back to the default parameter settings on Testhead power-up. A Tclear or resetJ1850VPW functional call will also reset the VPW-J1850 parameters to their default settings.

Visual Basic Declaration:

Public Sub setJ1850VPWPeriodicParams(ByVal channel As Integer, ByVal messageNum As Integer, ByVal bufferNum As Integer, ByVal numTimes As Integer, ByVal period As Integer)

Call setJ1850VPWPeriodicParams(channel, messageNum, bufferNum, numTimes, period)

WHERE:

channel

=		VPW-J1850 channel number.
=	0	Channel 0 on VPW-J1850 board 0.
=	1	Channel 1 on VPW-J1850 board 0.
.	.	. . . .
.	.	. . . .
=	15	Channel 15 on VPW-J1850 board 0.
=	16	Channel 0 on VPW-J1850 board 1, etc. to four boards per system.

msgNum

=		Channels periodic message number parameters to set.
=	0	Set up channel's periodic message number #0 parameters.
=	1	Set up channel's periodic message number #1 parameters.

bufferNum

=		J1850VPW buffer number to transmit.
=	0	to 31 (default = 0).

numTimes

=		Number of times to transmit the message.
=	-1	Transmit message continuously.
=	0	Don't transmit message (default).
=	>0	Number of times to transmit message.

period

= Time between buffer transmissions
= 125 to 15900 milliseconds (125 ms resolution) (default = 125)

EXAMPLES:

Call setJ1850VPWPeriodicParams (0, 0, 0, -1, 2000)
.....Setup channel #0 - message#0 to transmit
..... the contents of buffer #0 once every 2 seconds.

KWP2000 Functional Calls

Keyword Protocol 2000 Overview

Keyword Protocol 2000 is implemented on the MSP using the SXR serial port. An external interface circuit needs to be used if the SXR bus voltage on the MSP board doesn't meet the applications bus voltage requirements. The initKWP2000Fast function is provided to meet the Keyword Protocol 2000 fast initialization requirements. Keyword Protocol 2000 messages may be sent and received using either the MSP sendKWP2000 and recvKWP2000 functional calls or the sendSerial and recvSerial functional calls. If the sendSerial and recvSerial functions are used, the user must fully assemble/disassemble Keyword Protocol 2000 messages since they are sent and received transparently by these two functions. It is left up to the user to comply with the end of product response and start of new request timing requirements. Refer to ISO 14230-2 for timing requirements and message construction. The following is a list of UART parameters that need to be set (see setUartParams functional call) to meet the basic Keyword Protocol 2000 timing requirements:

Portcode	- Set to 1 to select the SXR port
Baudrate	- Use to set the baud rate.
Gap Timeout	- Set to a value greater than the inter byte delay
SCI Idle Detect	- Set to 0, turns off SCI idle line detection
SXR Bus Power	- Use to select the SXR bus voltage, 0 = +5VDC, 1 = +15VDC
Inter Byte Delay	- Use to set the inter byte delay for transmitted messages

Note: The SXR Bus Power option is only available on certain versions of the MSP board. A 100:021 UART Unknown Parameter error will be returned if the MSP hardware doesn't support this option.

initKWP2000Fast

The initKWP2000Fast functional call assembles the Keyword Protocol 2000 message, transmits a Keyword Protocol 2000 wakeup pattern, then transmits the assembled Keyword Protocol 2000 message using the UART parameter settings. The message is assemble (including checksum) based on the format byte provided by the user. Refer to ISO 14230-2 for more information about Keyword Protocol 2000 message formats.

Visual BASIC Declaration:

Public Sub initKWP2000Fast(ByVal format As Byte, ByVal target As Byte, ByVal source As Byte, ByVal length As Byte, dataArray() As Byte, ByVal timeout As Double)

Call initKWP2000Fast(format, target, source, length, dataArray(), timeout)

WHERE:

format
= Format byte.

target
= Target address for the message.

source
= Address of the transmitting device.

length
= The number of bytes of the data array to send (must be <= size of the data array). If any of the length bits (d5-d0) in the format byte are non-zero, this value must be the same value as that of the format byte length bits

dataArray
= Data array to transmit, byte #0 is always the service identification byte (Sid)

timeout
= The maximum time to wait in seconds for an idle bus. If this value is zero, the default idle timeout value set by setUartParams will be used.

EXAMPLES:

```
Dim dataArray(5) As Byte
dataArray (0) = &H81

Call initKWP2000Fast(&H80, &H21, &HF3, 1, dataArray (), 1#)..... ...
 .....Send KWP2000 fast initialization request
```

sendKWP2000

The sendKWP2000 functional call transmits a Keyword Protocol 2000 message using the UART parameter settings. The message is assembled (including checksum) based on the format byte provided by the user. Refer to ISO 14230-2 for more information about Keyword Protocol 2000 message formats.

Visual BASIC Declaration:

Public Sub sendKWP2000(ByVal format As Byte, ByVal target As Byte, ByVal source As Byte, ByVal length As Byte, dataArray () As Byte, ByVal timeout As Double)

Call sendKWP2000(format, target, source, length, dataArray(), timeout)

WHERE:

format
= Format byte.

target
= Target address for the message.

source
= Address of the transmitting device.

length
= The number of bytes of the data array to send (must be <= size of the data array). If any of the length bits (d5-d0) in the format byte are non-zero, this value must be the same value as that of the format byte length bits.

dataArray
= Data array to transmit, byte #0 is always the service identification byte (Sid).

timeout
= The maximum time to wait in seconds for an idle bus. If this value is zero, the default idle timeout value set by setUartParams will be used.

EXAMPLES:

```
Dim dataArray (2) As Byte
dataArray (0) = &H83
dataArray (1) = &H02

Call sendKWP2000(&H80, &H21, &HF3, 2, dataArray (), 1#).....
 ..... Send KWP2000 message
```

recvKWP2000

The recvKWP2000 functional call receives and disassembles a Keyword Protocol 2000 message using the UART parameter settings. A checksum error message will be generated if the received checksum is different from the calculated checksum. Refer to ISO 14230-2 for more information about Keyword Protocol 2000 message formats.

Visual BASIC Declaration:

Public Sub recvKWP2000(ByRef format As Byte, ByRef target As Byte, ByRef source As Byte, ByRef length As Byte, dataArray () As Byte, ByRef checksum As Byte, ByVal timeout As Double)

Call recvKWP2000(format, target, source, length, dataArray(), checksum, timeout)

WHERE:

format
= Format byte.

target
= Target address for the received message.

source
= Address of the transmitting device.

length
= The length in bytes of the receive data array. This parameter will be updated with the total number of data bytes received.

dataArray
= Receive data array, byte #0 contains the service identification byte (Sid).

checksum
= The checksum of the received message.

timeout

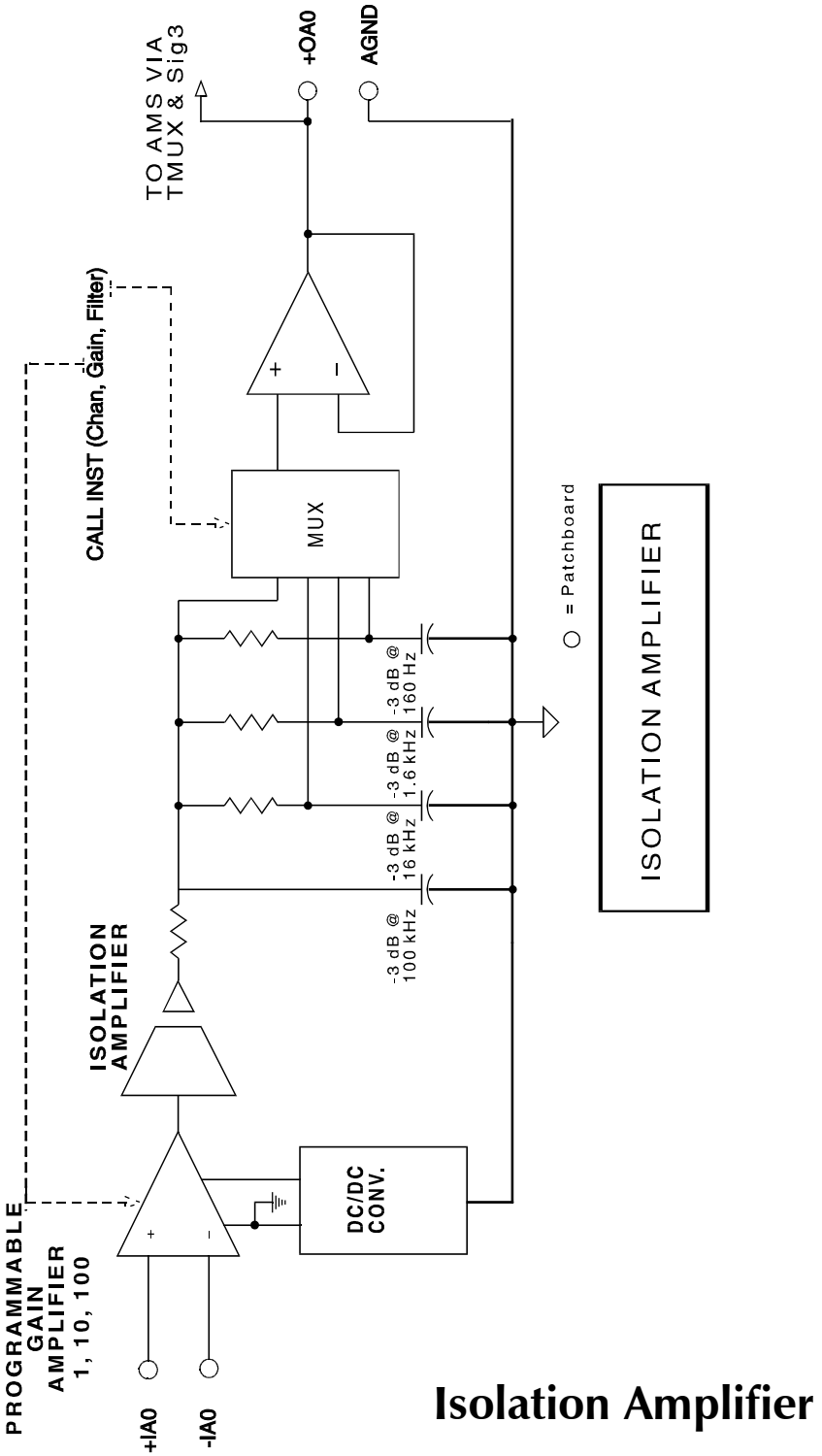
= The maximum time to wait in seconds for a message to be received. If this value is zero, the default receive timeout value set by setUartParams will be used.

EXAMPLES:

```
Dim format, target, source, length, checksum As Byte
Dim dataArray (10) As Byte
length = 10

Call recvKWP2000(format, target, source, length, dataArray (), checksum, 1#) ....
 ..... Receive KWP2000 message
```

Additional Functional Calls



Isolation Amplifier

INST

This functional call is used to set up the four differential isolation amplifiers (If your version of the MSP board has Isolation Amplifiers installed) on the MSP board. Each amplifier has programmable gain and programmable filters, and can be readback with the TMUX call.

Visual BASIC Declaration:

Public sub Inst(ByVal chan As Integer, ByVal gain As Integer, ByVal Filter As Integer)

Call Inst(chan, gain, Filter)

WHERE:

chan

= 0 to 3.

gain

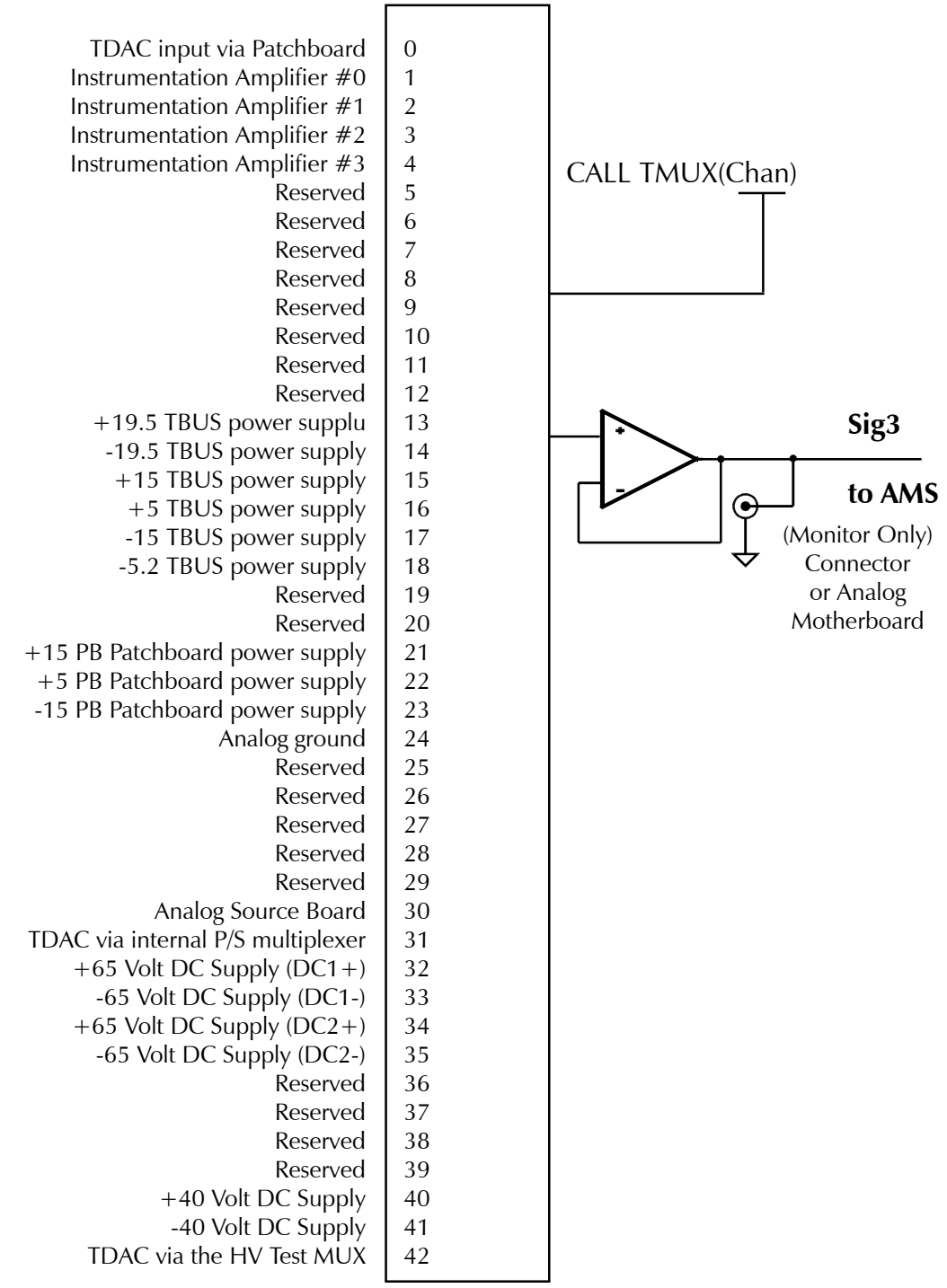
= 0 1.
= 1 10.
= 2 100.

Filter

= 0 No filter.
= 1 16,000 Hertz, -3db (-20db/decade)
= 2 1600 Hertz, (-20db/decade)
= 3 160 Hertz, (-20db/decade)

EXAMPLES:

Call Inst(1, 2, 0) ..... Amplifier 1 set to gain = 100 with no filter.
Call Inst(2, 1, 3) ..... Amplifier 2 set to gain = 10 with a 160 Hz filter.



Testhead Multiplexer

TMUX

The Selftest Multiplexer provides readback of system signals via Sig3, which is returned to the AMS via the Analog Motherboard. It is used in calibrating the D/A's, ARB's and the AMS using TDAC as a Reference. TDAC is calibrated to a secondary standard during the Digalog Certification Procedure. TMUX is available to the USER and may be used to readback Instrumentation Amplifier outputs.

Visual BASIC Declaration:

Public Sub TMux(ByVal Chan As Integer)

Call TMux(Chan)

WHERE:

Chan

=	0	TDAC input via Patchboard.
=	1	Instrumentation amplifier #0.
=	2	Instrumentation amplifier #1.
=	3	Instrumentation amplifier #2.
=	4	Instrumentation amplifier #3.
=	5	Reserved.
=	6	Reserved.
=	7	Reserved.
=	8	Reserved.
=	9	Reserved.
=	10	Reserved.
=	11	Reserved.
=	12	Reserved.
=	13	+19.5 TBUS power supply.
=	14	-19.5 TBUS power supply.
=	15	+15 TBUS power supply.
=	16	+5 TBUS power supply.
=	17	-15 TBUS power supply.
=	18	-5.2 TBUS power supply.
=	19	Reserved.
=	20	Reserved.
=	21	+15PB Patchboard power supply.
=	22	+5PB Patchboard power supply.
=	23	-15PB Patchboard power supply.
=	24	Analog ground.
=	25	Reserved.
=	26	Reserved.
=	27	Reserved.

=	28	Reserved.
=	29	Reserved.
=	30	Analog Source Board.
=	31	TDAC via internal P/S mux.
=	32	+65 Volt DC Supply (DC1+).
=	33	-65 Volt DC Supply (DC1-).
=	34	+65 Volt DC Supply (DC2+).
=	35	-65 Volt DC Supply (DC2-).
=	36	Reserved.
=	37	Reserved.
=	38	Reserved.
=	39	Reserved.
=	40	+40 Volt DC Supply.
=	41	-40 Volt DC Supply.
=	42	TDAC via the HV Test MUX.

EXAMPLES:

Dim Chan As Integer

Call TMux(1) ..... Multiplexes Instrumentation Amp #0 to Sig3 on the AMS.

Appendix - A

APPENDIX A - MSP ERROR CODES

100:001	(MSP)	No MSP board.
100:002	(MSP)	SCI port not ready.
100:003	(MSP)	SCI port overrun.
100:004	(MSP)	SCI port framing error.
100:005	(MSP)	SCI port noise error.
100:006	(MSP)	Invalid function number.
100:007	(MSP)	Out of memory.
100:008	(MSP)	MSP board is not responding to commands.
100:009	(MSP)	TBUS transmit timeout.
100:010	(MSP)	Invalid message number.
100:011	(MSP)	Invalid message size.
100:012	(MSP)	Bus error on board.
100:014	(MSP)	Unknown command.
100:015	(MSP)	MSP is already executing a command.
100:016	(MSP)	Differential SXR not supported.
100:020	(MSP)	UART unknown message.
100:021	(MSP)	UART unknown parameter.
100:022	(MSP)	UART buffer overflow.
100:025	(MSP)	DLCP unknown parameter.
100:030	(MSP)	Exception on board.
100:031	(MSP)	Exception frame is smaller than the MSP exception frame.
100:032	(MSP)	String storage size is smaller than the MSP Version string.
100:033	(MSP)	Bad message che3cksum.
100:100	(MSP)	C2DNLD Bad mode.
100:101	(MSP)	C2DNLD Bad test index.
100:113	(MSP)	C2DNLD Transfer suspended.
100:114	(MSP)	C2DNLD Transfer aborted.
100:116	(MSP)	C2DNLD Illegal address.
100:117	(MSP)	C2DNLD Illegal byte count.
100:118	(MSP)	C2DNLD Illegal block type.
100:119	(MSP)	C2DNLD CS error.
100:120	(MSP)	C2DNLD Incorrect byte count.
100:190	(MSP)	Mismatched echo.
100:191	(MSP)	Bad message length from product.
100:192	(MSP)	Bad checksum from product.
100:193	(MSP)	Timed out while waiting for response from product.
100:194	(MSP)	Framing, overrun, or noise error.
100:195	(MSP)	Timed out while waiting for an idle line.
100:196	(MSP)	Timed out while waiting for an echo byte.
100:200	(MSP)	DLCP receive FIFO invalid.
100:201	(MSP)	DLCP bus shorted.
100:202	(MSP)	DLCP timed out while waiting for an idle line.
100:203	(MSP)	DLCP invalid message size.
100:204	(MSP)	DLCP timed out while waiting for message.
100:205	(MSP)	DLCP timed out while waiting to transmit.

100:206	(MSP)	DLCP missing completion code.
100:207	(MSP)	DLCP completion code indicated no transmit message.
100:208	(MSP)	DLCP transmitter overrun.
100:209	(MSP)	DLCP transmitter lost arbitration.
100:210	(MSP)	DLCP early completion code received.
100:211	(MSP)	DLCP circular buffer overflow.
100:212	(MSP)	DLCP transmit FIFO not empty.
100:213	(MSP)	DLCP receive FIFO overrun.
100:214	(MSP)	DLCP CRC error.
100:215	(MSP)	DLCP incomplete byte received.
100:216	(MSP)	DLCP bit timing error.
100:217	(MSP)	DLCP break error.
100:220	(MSP)	SPI unknown parameter.
100:221	(MSP)	SPI invalid parameter.
100:225	(MSP)	BDM invalid parameter.
100:226	(MSP)	BDM write buffer too large.
100:228	(MSP)	Illegal message arbitration ID value - must be either 29-bits max(ext) or 11-bits max(std).
100:229	(MSP)	Number of data bytes parameter in transmit frame buffer too large.
100:230	(MSP)	CAN controller in reset, sleep, or power down.
100:231	(MSP)	CAN controller configuration not enabled.
100:232	(MSP)	CAN register write error.
100:233	(MSP)	CAN illegal controller message object.
100:234	(MSP)	MSP timed out while trying to send a CAN message.
100:235	(MSP)	CAN no message received.
100:236	(MSP)	CAN bad parameter error.
100:237	(MSP)	CAN bad array byte size.
100:238	(MSP)	CAN message object's config register bit improperly set for function called.
100:239	(MSP)	CAN message object's control register 0 message valid bit not set.
100:240	(MSP)	CAN Timeout parameter is either negative or exceeds the maximum value.
100:241	(MSP)	CAN controller went busoff due to errors on the CAN bus.
100:242	(MSP)	CAN message object is busy transmitting messages.
100:243	(MSP)	CAN message object is busy receiving messages.
100:244	(MSP)	Frame size not equal to the message object buffer's frame size.
100:245	(MSP)	Message object buffer is full.
100:246	(MSP)	Matching CAN message object not found.
100:247	(MSP)	Invalid direction parameter.
100:248	(MSP)	Invalid Message Valid parameter value.
100:249	(MSP)	Timeout expired before the desired number of messages were received.
100:250	(MSP)	Invalid number of frames parameter - too large or equal to zero.

Appendix - B

ISO AMP #0 + Input	TPB1	BPB1	ISO AMP #0 Output
ISO AMP #0 - Input	TPB2	BPB2	ISO AMP #0 AGND
ISO AMP #1 + Input	TPB3	BPB3	ISO AMP #1 Output
ISO AMP #1 - Input	TPB4	BPB4	ISO AMP #1 AGND
ISO AMP #2 + Input	TPB5	BPB5	ISO AMP #2 Output
ISO AMP #2 - Input	TPB6	BPB6	ISO AMP #2 AGND
ISO AMP #3 + Input	TPB7	BPB7	ISO AMP #3 Output
ISO AMP #3 - Input	TPB8	BPB8	ISO AMP #3 AGND
	TPB9	BPB9	
	TPB10	BPB10	
	TPB11	BPB11	
	TPB12	BPB12	
	TPB13	BPB13	
	TPB14	BPB14	
	TPB15	BPB15	
VBAT	TPB16	BPB16	VBGD
MISO	TPB17	BPB17	MOSI
SCK	TPB18	BPB18	SS
RS-422/RS-485 +	TPB19	BPB19	RS-422/RS-485 -
IC1	TPB20	BPB20	IC2
CAN High	TPB21	BPB21	CAN Low
J1850 Load	TPB22	BPB22	J1850 Bus
TDI	TPB23	BPB23	TDO
TMS	TPB24	BPB24	TCK
TRST	TPB25	BPB25	PAI
D0	TPB26	BPB26	D1
D2	TPB27	BPB27	D3
D4	TPB28	BPB28	D5
D6	TPB29	BPB29	D7
IDGD	TPB30	BPB30	AGND
UART +	TPB31	BPB31	UART -
PCS3	TPB32	BPB32	PWMA
DGND	TPB33	BPB33	DGND
RS-232C XMT	TPB34	BPB34	RS-232C RCV

Patchboard Pin Mnemonics

ISO AMP #0 Output	Signal output from Isolation Amplifier #0.
ISO AMP #0 AGND	Analog ground associated with Isolation Amplifier #0.
ISO AMP #0 + Input	The higher of the two differential inputs to Isolation Amplifier #0.
ISO AMP #0 - Input	The lower of the two differential inputs to Isolation Amplifier #0.
VBAT	External voltage applied to the board to run the DLCP bus instead of the +12VDC on the MSP board.
VBGD	VBAT Return pin.
MISO	Serial input pin for SPI (Serial Perepheral Interface) communications.
MOSI	Serial output pin for SPI communications.
SCK	Clock signal generated on the MSP board to control SPI communications.
SS	Peripheral chip select line.
RS-422/485 +	Serial interface positive communication line.
RS-422/485 -	Serial interface negative communication line.
IC1	Input Compare #1 - Not currently used.
IC2	Input Compare #2 - Not currently used.
CAN High	CAN (Control Area Network) high side.
CAN Low	CAN low side.
J1850 Load	J1850 return line.
J1850 Bus	J1850 signal line.
TDI	MSP Boundary Scan TDI input.
TDO, TMS, TCK, TRST	MSP Boundary Scan TDI outputs.
PAI	Input Compare line - Not currently used.
D0 - D7	Patchboard ID pins.
IDGD	Patchboard ID ground.
AGND	Analog ground.
UART+	UART signal line (Single line UART)
UART-	UART negative signal line (Differential UART).
PCS3, PWMA	MSP Input Compare/PWM Output signals.
DGND	Digital ground.
RS-232C XMT	RS-232C tramsmit line.
RS-232C RCV	RS-232C receive line.
TPB9 through 15	Unused.
BPB9 through 15	Unused.