Series 2040 Test Systems

# MSP
# UART/Class II
# User Manual

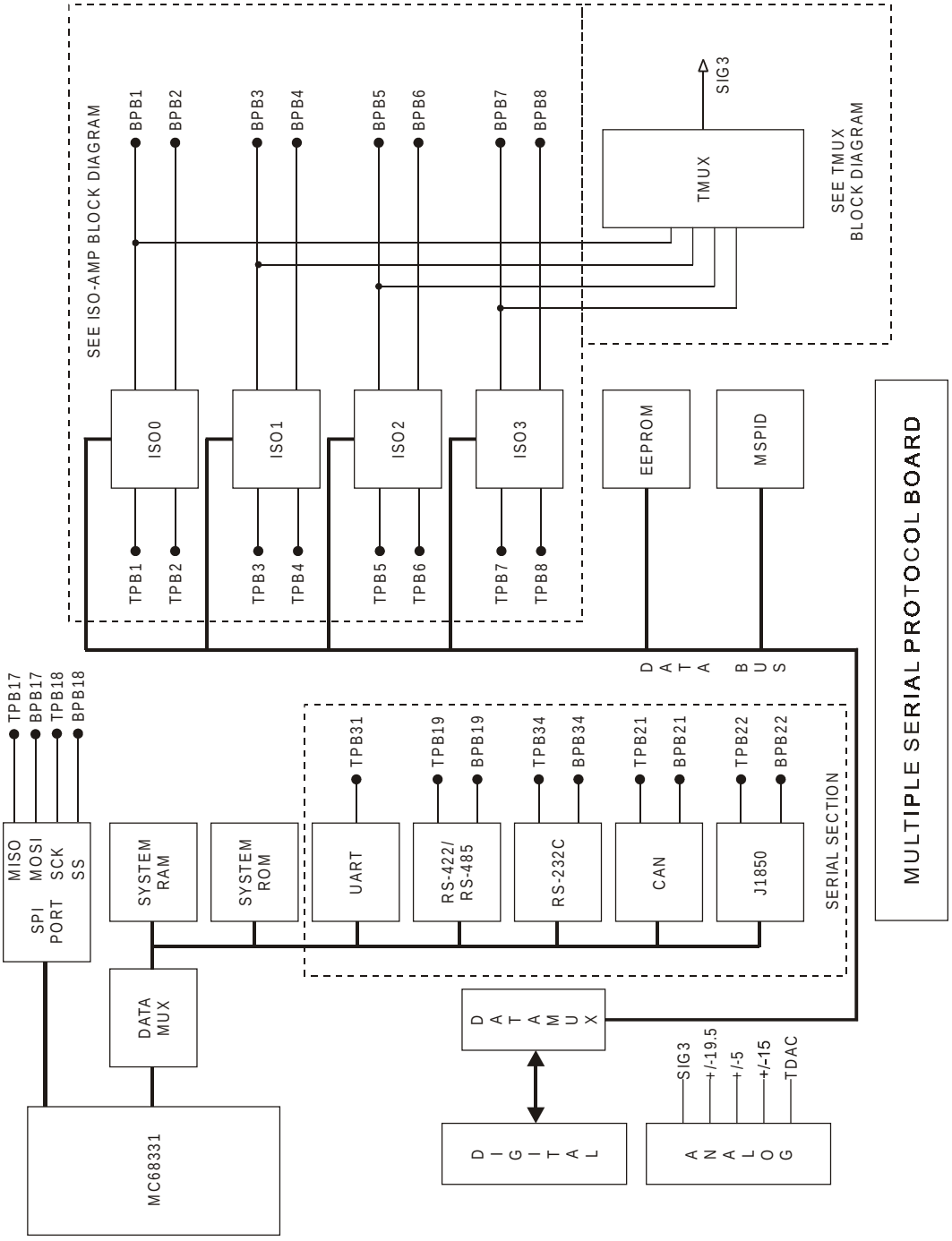
**Part Number 4200-0178**
**Version 2.2**

# Table Of Contents

# *MSP UART/Class II User Manual*

***Version 2.2***

# 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 and Class II can also be used with this card.

The MSP card has four Isolation Amplifiers (ISOAMPs). These amplifiers have differential inputs followed by a programmable gain stage, the output of which 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.

**EXAMPLES:**

```
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 error code.
```

## 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 error code.
```

## 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)<br>1 = Single line SXR<br>2 = RS422 |
| 2 | Baudrate (1 to 65000) | 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<br>1 = Check echo type (Default) |

| | | |
|---|---|---|
| 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 |

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

**EXAMPLES:**

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

## 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<br>1 = Single line SXR<br>2 = RS422 |
| 2 | Baudrate (1 to 65000) | 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<br>1 = Check echo type |

| | | |
|---|---|---|
| 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 |

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

**EXAMPLES:**

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

## loadTestMod

The loadTestMod functional call loads a binary test module into memory for later access by functional calls such as uart01, uart02, and c2dnld. The module contains test routines that get downloaded via a serial link from the tester to the DUT by the functional calls. The individual tests in the module are parsed and indexed after the file is loaded. The size of the index must be specified since programs like uart01 rely on an 8-bit index, and programs that could download programs larger than 170 bytes like c2dnld rely on a 16-bit index.

**Visual BASIC Declaration:**

Public Sub loadTestMod(ByVal ModuleName As String, ByVal IndexSize As Integer)

**Call loadTestMod(ModuleName, IndexSize)**

**WHERE:**

**ModuleName**

= The name of the binary test module (including path).

**IndexSize**

= 8 or 16. The size of index at the beginning of each test (8-bit, 16-bit).

**EXAMPLES:**

Call loadTestMod("temp.bin", 8) ..... Load a file called temp.bin
.....that has an 8-bit index.
Call loadTestMod("c:\Digalog\Projects\MyProj\C2Tests.Bin", 16) ..... Load a file
..... called C2Tests.Bin that has a 16-bit index.

## aldl

As defined in the Delco Corporate Serial Data Communications Specification, a request may be made via the serial data link to transmit product diagnostic codes and any other information specified in the product's XDE. The ALDL request is termed MODE 1. Communication is at the baud rate set up by SetUARTParams.

**Visual BASIC Declaration:**

Public Sub aldl(rid As Byte, rmsg() As Byte, ByVal sid As Byte)

**Call aldl(rid, rmsg, sid)**

**WHERE:**

**rid**
= DUT ID if operation is error free.

**rmsg**
= Message received from the DUT.

**sid**
= ID of DUT transmitted to to the tester.

**EXAMPLES:**

```
Dim rmsg(1 To 20) As Byte
Dim rid As Byte

Call aldl(rid, rmsg, &HFA) ..... Send ID and wait for a response or error code.
```

## dnreq

For products that do not have illegal input conditions specified for entering the factory test mode, the Delco Corporate Serial Data Communications Specification and XDE-5024 provides a means to enter the factory test mode. This is typically referred to as a MODE 5. Once the request has been granted, RAM download messages via MODE 6 (RAM download and execute) may commence.

The dnreq call sends a MODE 5 message to request the product enter download and execute mode. If the request is granted, a $AA is returned in rmsg(1). Anything else signifies that the product is busy. Communication is at the baud rate set up by SetUARTParams.

**Visual BASIC Declaration:**

Public Sub dnreq(rid As Byte, rmsg() As Byte, ByVal sid As Byte)

**Call dnreq(rid, rmsg, sid)**

**WHERE:**

**rid**
= Received DUT ID if operation error free, error code if not.

**rmsg**
= Data array received from DUT.

**sid**
= ID of DUT transmitted to the DUT.

**EXAMPLES:**

```
Dim rmsg(1 To 20) As Byte
Dim rid As Byte

Call dnreq(rid, rmsg, &HFA).....Send download request to DUT
 ..... and wait for response or error code.
```

## uart01

The UART01 functional call sends a MODE 6 message to the product. When executed, the function links to the product RAM download module (loaded by 'loadTstMod') to form a complete serial communications message. The function follows the serial protocol outlined in XDE-5024. This function downloads to product RAM starting at $0010 and communicates at the baud rate set up by SetUARTParams.

**Visual BASIC Declaration:**

Public Sub uart01(rid As Byte , rmsg() As Byte , ByVal sid As Byte, smsg() As Byte)

**Call uart01(rid, rmsg, sid, smsg,)**

**WHERE:**

**rid**
= Received DUT ID if operation error free, error code if not.

**rmsg**
= Data received from DUT.

**sid**
= ID of DUT transmitted to DUT.

**smsg**
= RAM download module test number and parameters.

**smsg(1)**
= Test number.

**smsg(2)-(n)**
= Parameters for test.

**EXAMPLES:**

```
Dim rmsg(1 To 20) As Byte
Dim smsg(1 To 20) As Byte
Dim rid As Byte
Dim i As Integer
For i = 1 to 20  ' Data to send
Smsg(i) = i
Next i

Call uart01(rid, rmsg, &HFA, smsg) ..... Send ID and message to DUT,
 ..... wait for response or error code.
```

## uart02

The uart02 functional call sends a MODE 6 message to the product. When executed, the function links to the product RAM download module (loaded by 'loadTstMod') to form a complete serial communication message. The function follows the serial protocol outlined in XDE-5024. Communication is at the baud rate set up by 'SetUARTParams'.

**Visual BASIC Declaration:**

Public Sub uart02(rid As Byte , rmsg() As Byte , ByVal sid As Byte, smsg() As Byte, ByVal raddr as Long)

**Call uart02(rid, rmsg, sid, smsg, raddr)**

**WHERE:**

**rid**
= Received DUT ID if operation error free, error code if not.

**rmsg**
= Data received from DUT.

**sid**
= ID of DUT transmitted to DUT.

**smsg**
= RAM download module test number and parameters.

**smsg(1)**
= Test number.

**smsg(2)-(n)**
= Parameters for test.

**raddr**
= Address to download the test and parameters to.

**EXAMPLES:**

```
Dim rmsg(1 To 20) As Byte
Dim smsg(1 To 3) As Byte
Dim rid As Byte
smsg(1) = 0          'Test number
smsg(2) = &HAB 'Test parameters
smsg(3) = &HCD

Call uart02(rid, rmsg, &HFA, smsg, &H12345678) ..... Set up download.
```

## uart02nr

The uart02nr functional call sends a MODE 6 message to the product. When executed, the function links to the product RAM download module (loaded by 'loadTstMod') to form a complete serial communication message. The function follows the serial protocol outlined in XDE-5024. This function does not get a response from the product. Communication is at the baud rate set up by the SetUARTParams call.

**Visual BASIC Declaration:**

Public Sub uart02nr(rid As Byte, rmsg() As Byte, ByVal sid As Byte, smsg() As Byte, ByVal raddr as Long)

**Call uart02nr(rid, rmsg, sid, smsg, raddr)**

**WHERE**

**rid** = Received DUT ID if operation error free, error code if not.

**rmsg** = Data received from DUT.

**sid** = ID of transmitted to DUT.

**smsg** = RAM download module test number and parameters.

**smsg(1)** = Test number.

**smsg(2)-(n)** = Parameters for test.

**raddr** = Address to download the test and parameters to.

**EXAMPLES:**

```
Dim rmsg(1 To 20) As Byte
Dim smsg(1 To 3) As Byte
Dim rid As Byte
smsg(1) = 0          'Test number
smsg(2) = &HAB 'Test parameters
smsg(3) = &HCD
Call uart02nr(rid, rmsg, &HFA, smsg, &H12345678) ..... Set up download.
```

## xde9067_1

The xde9067_1 functional call is used to receive any type of legal XDE-5024 message. This function receives and checks a message, but only returns the RID to the calling program. This function will return an error if a message is not received within the default receive timeout period set up by the SetUARTParams call. Communication is at the baud rate set up by the SetUARTParams call.

**Visual BASIC Declaration:**

Public Sub xde9067_1(rid As Byte)

**Call xde9067_1(rid)**

**WHERE:**

**rid**

= Received DUT ID if operation error free, error code if not.

**EXAMPLES:**

```
Dim rid As Byte

Call xde9067_1(rid).....Get DUT ID or error code.
```

## xde9067_2

The xde9067_2 functional call is used to receive any type of legal XDE-5024 message. This function will return an error if a message is not received within the default receive timeout period set up by the SetUARTParams call. Communication is at the baud rate set up by the SetUARTParams call.

**Visual BASIC Declaration:**

Public Sub xde9067_2(rid As Byte, rmsg() As Byte)

**Call xde9067_2(Rid, Rmsg)**

**WHERE:**

**rid**
= Received DUT ID if operation error free, error code if not.

**rmsg**
= Message received from DUT.

**EXAMPLES:**

```
Dim rmsg(1 To 20) As Byte
Dim rid As Byte

Call xde9067_2(rid, rmsg)..... Get DUT ID and data or error code.
```

## xde9067_3

The xde9067_3 functional call is used to receive any type of legal XDE-5024 message. Communication is at the baud rate set up by the SetUARTParams call.

**Visual BASIC Declaration:**

Public Sub xde9067_3(rid As Byte, rmsg() As Byte, ByVal timeout As Double)

**Call xde9067_3(rid, rmsg, timeout)**

**WHERE:**

**rid**
= Received DUT ID if operation error free, error code if not.

**rmsg**
= Message received from DUT.

**timeout**
= 0 to 65. Time in seconds to wait for response from DUT.

**EXAMPLES:**

```
Dim rmsg(1 To 20) As Byte
Dim rid As Byte

Call xde9067_3(rid, rmsg, 1)..... Get ID and message from DUT,
 ..... wait 1 second for response or error code.
```

## xde9067_4

The xde9067_4 functional call is used to send and receive any type of legal XDE-5024 message. The transmitted message will consist of SID, message length ($55) and checksum. Communication is at the baud rate set up by the SetUARTParams call.

**Visual BASIC Declaration:**

Public Sub xde9067_4(rid As Byte, rmsg() As Byte, ByVal timeout As Double, ByVal sid As Byte)

### Call xde9067_4(rid, rmsg, timeout, sid)

**WHERE:**

**rid**
= Received DUT ID if operation error free, error code if not.

**rmsg**
= Message received from DUT.

**timeout**
= 0 to 65. Time in seconds to wait for response from DUT.

**sid**
= ID of DUT transmitted to.

**EXAMPLES:**

```
Dim rmsg(1 To 20) As Byte
Dim rid As Byte

Call xde9067_4(rid, rmsg, 1, &HFA) ..... Send ID to DUT, wait 1 second for ID
 ..... and message from DUT or error code.
```

## xde9067_5

The xde9067_5 functional call is used to send and receive any type of legal XDE-5024 message. Communication is at the baud rate set up by the SetUARTParams call.

**Visual BASIC Declaration:**

Public Sub xde9067_5(rid As Byte, rmsg() As Byte, ByVal timeout As Double, ByVal sid As Byte, smsg() As Byte)

**Call xde9067_5(rid, rmsg, timeout, sid, smsg)**

**WHERE:**

**rid**
= Received DUT ID if operation error free, error code if not.

**rmsg**
= Message received from DUT.

**timeout**
= 0 to 65. Time in seconds to wait for response from DUT.

**sid**
= ID of DUT transmitted to.

**smsg**
= Message to send to DUT.

**EXAMPLES:**

```
Dim rmsg(1 To 20) As Byte
Dim smsg(1 To 2) As Byte
Dim rid As Byte

smsg(1) = 1     ‘mode number
smsg(2) = 0     ‘message number

Call xde9067_5(rid, rmsg, 1, &HFA, smsg)..... Send ID and message to DUT, wait
 ..... 1 second for response from DUT or error code.
```

# CLASS II Functional Calls

## MSP - Class II Load Jumper Selections

| MSP Board<br>Digalog Part Number | Terminating Node<br>C = 3300pF, R = 1.5KOhm | Standard "Unit" Node<br>C = 470pF, R = 10.6KOhm |
|---|---|---|
| 0000-5030 | JP7: 2-3, JP8: 2-3 | JP7: 1-2, JP8: 1-2 |
| 0000-5330 | JP5: 2-3, JP6: 2-3 | JP5: 1-2, JP6: 1-2 |
| 0000-5332 | JP1: 2-3, JP2: 2-3 | JP1: 1-2, JP2: 1-2 |
| 0000-6030 | JP1: 2-3, JP2: 2-3 | JP1: 1-2, JP2: 1-2 |

***NOTE: Jumper selection 2-3 is the jumper position closest to the board stiffener.***

## c2alive

The c2alive functional call sends a tool present message to the DUT to keep the 5 second activity timer reset. If the message is transmitted without error, rid will equal sid. If there was an error, the error code will be in rid. For more information, see XDE-3005.

**Visual BASIC Declaration:**

Public Sub c2alive(rid As Byte, ByVal sid As Byte)

**Call c2alive(rid, sid)**

**WHERE:**

**rid**

= DUT ID if operation is error free.

**sid**

= ID of DUT transmitted to DUT.

**EXAMPLES:**

```
Dim rid As Integer
Dim sid As Integer

sid = &H55
Call c2alive(rid, sid) ..... Reset timer.
```

## c2dnrqst

The functional call c2dnrqst sends a request for block transfer message to the DUT. For more information see XDE-3005.

**Visual BASIC Declaration:**

Public Sub c2dnrqst(rid As Byte, rmsg() As Byte, ByVal raddr As Long, ByVal sid As Byte, ByVal length As Integer)

**Call c2dnrqst(rid, rmsg, raddr, sid, length)**

**WHERE:**

**rid** = Received DUT ID if operation error free, error code if not.

**rmsg** = Data array received from DUT.

**raddr** = DUT address for block transfer.

**sid** = ID of DUT transmitted to DUT.

**length** = Maximum data bytes for each block transfer.

**EXAMPLES:**

```
Dim rmsg(1 To 20) As Byte
Dim rid As Byte

Call c2dnrqst(rid, rmsg, &H1111, &HFA, 20) ..... Send starting ram address
 ..... and ID to transfer 20 bytes of data,
 ..... receive DUT ID and requested data or error code.
```

## c2dnld

The c2dnld functional call allows RAM based test modules to be downloaded to the DUT. It will also receive the response message from the DUT. For more information see XDE-3005.

**Visual BASIC Declaration:**

Public Sub c2dnld(rid As Byte, rmsg() As Byte, ByVal raddr As Long, ByVal sid As Byte, smsg() As Byte, ByVal itimeout As Integer, status As Byte, ByVal mode As Byte )

**Call c2dnld(rid, rmsg, raddr, sid, Smsg, itimeout, status, mode)**

**WHERE:**

| | | | |
|---|---|---|---|
| **rid** | | | |
| | = | | Received DUT ID if operation error free, error code if not. |
| **rmsg** | | | |
| | = | | Data received from DUT. |
| **raddr** | | | |
| | = | | DUT address to load and/or execute program. |
| **sid** | | | |
| | = | | ID of DUT transmitted to DUT. |
| **smsg** | | | |
| | = | | TEMP.BIN test number and parameters. |
| **smsg(1)** | | | |
| | = | | Test number. |
| **smsg(2)-(n)** | | | |
| | = | | Parameters for test. |
| **itimeout** | | | |
| | = | 0 | to 32767. Maximum wait time for a 'TOUTPUT' response (milliseconds). |
| **status** | | | |
| | = | | DUT response to block transfer. |
| **mode** | | | |
| | = | | Mode parameter. |
| | = | 0 | Execute program at raddr. |
| | = | 1 | Download program at raddr. |
| | = | 2 | Download and execute at raddr. |

## EXAMPLES:

```
Dim rmsg(1 To 20) As Byte
Dim smsg(1 To 3) As Byte
Dim rid As Byte

Smsg(1) = 0          'Test number
Smsg(2) = &HAB 'Test parameters
Smsg(3) = &HCD

Call c2dnld(rid, rmsg, &H123456, &HFA, smsg, 1000, status, &H1).....
 ..... Download program
```

## c2nstd

The functional call c2nstd allows transmission and reception of nonstandard class 2 messages between the DUT and the functional tester. For more information see XDE-3005.

**Visual BASIC Declaration:**

Public Sub c2nstd(rid As Byte, rmsg() As Byte, ByVal sid As Byte, smsg() As Byte, ByVal itimeout as Integer, ByVal numbytes As Integer)

**Call c2nstd(rid, rmsg, sid, smsg, itimeout, numbytes)**

**WHERE:**

| | | | |
|---|---|---|---|
| **rid** | | | |
| | = | | Received DUT ID if operation error free, error code if not. |
| **rmsg** | | | |
| | = | | Data received from DUT. |
| **sid** | | | |
| | = | | ID of DUT transmitted to DUT. |
| **smsg** | | | |
| | = | | Array containing data to be sent to the DUT. |
| **itimeout** | | | |
| | = | 0 | to 32767. Maximum wait time for a 'TOUTPUT' response (milliseconds). |
| **numbytes** | | | |
| | = | | Number of data bytes to transmit (from the smsg array). |

**EXAMPLES:**

```
Dim rmsg(1 To 20) As Byte
Dim smsg(1 To 20) As Byte
Dim rid As Byte
Dim i As Integer
For i = 1 to 20  ' Data to send
smsg(i) = i
Next i

Call c2nstd(rid, rmsg, &HFA, smsg, 100, 20) ..... Send the DUT ID and 20 bytes
 .....of data from smsg, wait 100 mS for a response
 ..... and DUT data or error code from DUT.
```

## c2sak

The functional call c2sak sends the seed and key messages to the DUT. For more information see XDE-3005.

**Visual BASIC Declaration:**

Public Sub c2sak(rid As Byte, rmsg() As Byte, ByVal sid As Byte, smsg() As Byte, ByVal mode As Byte)

### Call c2sak(rid, rmsg, sid, smsg, mode)

**WHERE:**

**rid**
= Received DUT ID if operation error free, error code if not.

**rmsg**
= Data received from DUT.

**rmsg(1)**
= Response code.

**rmsg(2)**
= SEED (MSB).

**rmsg(3)**
= SEED (LSB).

**sid**
= ID of DUT transmitted to DUT [DIM BYTE].

**smsg**
= Data to send to DUT [DIM (n) BYTE].

**smsg(1)**
= KEY (MSB).

**smsg(2)**
= KEY (LSB).

**mode**
= Desired SEED/KEY mode.
$00 = Get SEED from DUT
$FF = Send KEY to DUT

| | |
|---|---|
| Response codes: | SEED codes |
| $00 | = Valid acceptance of SEED request |
| $D8 | = DUT did not accept SEED request |
| | |
| | KEY codes |
| $D8 | = DUT did not accept KEY request |
| $33 | = Product secured |
| $AA | = Access allowed |
| $CC | = Invalid key |
| $XX | = Any other - access denied |

## EXAMPLES:

```
Dim rmsg(1 To 20) As Byte
Dim smsg(1 To 2) As Byte
Dim rid As Byte

smsg(1) = &H12
smsg(2) = &H34

Call c2sak(rid, rmsg, &HFA, smsg, &H0) ..... Get SEED from DUT
Call c2sak(rid, rmsg, &HFA, smsg, &HFF) ..... Send KEY to DUT
```

## c2speed

The c2speed functional call changes the speed of the class2 bus to either normal or high speed. The call sends and receives the messages required and instructs the MSP board to change its DLCP speed. For more information see XDE-3005.

**Visual BASIC Declaration:**

Public Sub c2speed(rid As Byte, rmsg() As Byte, ByVal sid As Byte, ByVal mode As Byte)

### Call c2speed(rid, rmsg, sid, mode)

**WHERE:**

**rid**
= Received DUT ID if operation error free, error code if not.

**rmsg**
= Data received from DUT.

**rmsg(1)**
= Response code.

**sid**
= ID of DUT transmitted to DUT.

**mode**
= Desired speed mode.
$01 = Normal mode
$04 = High speed mode

Response codes:
$00 = Successful speed change
$D0 = MSP board did not respond to speed change
$D1 = DUT did not accept disable normal communication request
$D2 = DUT did not accept request to enter high speed mode

**EXAMPLES:**

```
Dim rmsg(1 To 20) As Byte
Dim rid As Byte

Call c2speed(rid, rmsg, &HFA, &H4) ..... Request high speed mode.
```

## c2upld

The functional call c2upld requests a block of data from the DUT. For more information see XDE-3005.

**Visual BASIC Declaration:**

Public Sub c2upld(rid As Byte, rmsg() As Byte, ByVal raddr As Long, ByVal sid As Byte, ByVal numbytes As Integer, upstatus As Byte)

**Call c2upld(rid, rmsg, raddr, sid, numbytes, upstatus)**

**WHERE:**

**rid**
= Received DUT ID if operation error free, error code if not,

**rmsg**
= Data received from DUT.

**raddr**
= DUT address for block transfer.

**sid**
= ID of DUT transmitted to DUT.

**numbytes**
= Number of data bytes to be transmitted.

**upstatus**
= Response code of transfer request.
$FF = No response (generated by the MSP board)
$XX = DUT response

**EXAMPLES:**

```
Dim rmsg(1 To 1024) As Byte
Dim rid As Byte
Dim status As Byte

Call c2upld(rid, rmsg, &H123456, &HFA, 1024, status) ..... Request 1024 bytes.
```

## c2transmit

The functional call c2transmit allows for free form CLASS2 data transmission to a product. Essentially, the caller's buffer is transmitted as is to the product. For more information see XDE-3005.

**Visual BASIC Declaration:**

Public Sub c2transmit(xmitbuff() As Byte, ByVal xmitsize As Integer, xmiterr As Integer)

**Call c2transmit(xmitbuff, xmitsize, xmiterr)**

**WHERE:**

**xmitbuff**
= The buffer containing data to to be transmitted to the DUT.

**xmitsize**
= 0 to 4096. The transmit data size (number of bytes to transmit from the buffer).

**xmiterr**
= The results of the communication with the MSP and product (0 = NO ERROR)

**EXAMPLES:**

```
Dim smsg(1 To 20) As Byte
Dim xmiterr As Integer
Dim i As Integer

For i = 1 to 20 ‘ Data to transmit
smsg(i) = i
Next i

Call c2transmit(smsg, 20, xmiterr) ..... Send buffer to product.
```

## c2receive

The functional call c2receive allows for free form CLASS2 data reception from a product. The routine will await the message initiation for a time specified by the caller after which time the attempt is aborted and an error is returned. For more information see XDE-3005.

**Visual BASIC Declaration:**

Public Sub c2receive(recvbuff() As Byte, ByVal timeout As Integer, recverr As Integer)

**Call c2receive(recvbuff, timeout, recverr)**

**WHERE:**

**recvbuff**
= The buffer containing data to to be received from the DUT.

**timeout**
= 0 to 32767. The time to wait (in milliseconds) for a message initiation from the DUT.

**recverr**
= The results of the communication with the MSP and product (0 = NO ERROR).

**EXAMPLES:**

```
Dim rmsg(1 To 20) As Byte
Dim recverr As Integer

Call c2receive(rmsg, 1000, recverr) ..... Wait 1000 mS for DUT data.
```

## c2transceive

The c2transceive functional call allows for free form CLASS2 data transmission and subsequent reception from a product. The routine will await the message initiation for a time specified by the caller after which time the attempt is aborted and an error is returned. For more information see XDE-3005.

**Visual BASIC Declaration:**

Public Sub c2transceive(xmitbuff() As Byte, ByVal xmitsize As Integer, recvbuff() As Byte, ByVal timeout As Integer, commerr As Integer)

**Call c2transceive(xmitbuff, xmitsize, recvbuff, timeout, commerr)**

**WHERE:**

| | | |
|---|---|---|
| **xmitbuff** | | |
| = | | The buffer containing data to to be transmitted to the DUT. |
| **xmitsize** | | |
| = | 0 | to 4096. The transmit data size (number of bytes to transmit from the buffer). |
| **recvbuff** | | |
| = | | The buffer containing data to to be received from the DUT. |
| **timeout** | | |
| = | 0 | to 32767. The time to wait (in milliseconds) for a message initiation from the DUT. |
| **commerr** | | |
| = | | The results of the communication with the MSP and product (0 = NO ERROR). |

**EXAMPLES:**

```
Dim smsg(1 To 20) As Byte
Dim rmsg(1 To 20) As Byte
Dim commerror As Integer
Dim i As Integer
For i = 1 to 20  ‘ Data to transmit
smsg(i) = i
Next i

Call c2transceive(smsg, 20, rmsg, 1000, commerror) ..... Wait 1000 mS for
 ..... communication.
```

## c2disnormal

The functional call c2disnormal sends a MODE 28 (DISABLE NORMAL COMMUNICATIONS) message to the product via the MSP card as well as receiving the response to the request. The routine will await the response intitiation for a time specified after which time the attempt is aborted and an error is returned. For more information see XDE-3005.

**Visual BASIC Declaration:**

Public Sub c2disnormal(modaddr As Byte, ByVal timeout As Integer, status As Byte, commerr As Integer)

**Call c2disnormal(modaddr, timeout, status, commerr)**

**WHERE:**

**modaddr**
= The MODULE ADDRESS as specified in the return message in XDE-3001B.

**timeout**
= 0 to 32767. The time to wait (in milliseconds) for a message initiation from the DUT.

**status**
= The status of the disable request (0 = NO ERROR).

**commerr**
= The results of the communication with the MSP and product (0 = NO ERROR).

Status Codes:
$00 = Normal communications disabled
$11 = Mode not supported
$12 = Sub function not supported
$22 = Conditions not correct for request

**EXAMPLES:**

```
Dim modAddr As Byte
Dim status As Byte
Dim commerror As Integer

Call c2disnormal(modAddr, 1000, status, commerror) ..... Send message and
 .....wait  1000 mS for a response.
```

## dlcpParams

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

**Visual BASIC Declaration:**

Public Sub dlcpParams(ByVal paramNumber As Integer,ByVal paramValue As Integer)

**Call dlcpParams(paramNumber, paramValue)**

**WHERE:**

**paramNumber**
= Index of the parameter to set.

**paramValue**
= Value to set the parameter to.

| Index # | Parameter name | Parameter value |
|---|---|---|
| 1 | Dlcp Config | Config byte |
| 2 | Set Speed | 0 = Normal speed (default)<br>1 = High speed |
| 4 | Receive Timeout | 0 to 32767 mS (default = 100) |
| 6 | Idle Timeout | 0 to 32767 mS (default = 10) |
| 9 | Direct Command | DLCP command and data byte |
| 10 | Disable Retry | 0 = Retry enabled<br>1 = Retry disabled (default) |
| 11 | Reconfigure | 0 = Do not configure DLCP (default)<br>1 = Configure DLCP before xmit |
| 12 | Issue Reset | 0 = Do not reset DLCP before xmit (default)<br>1 = Reset DLCP before xmit |
| 13 | Flush Previous | 0 = Do not clear RFIFO before xmit<br>1 = Clear RFIFO before xmit (default) |
| 14 | Issue Flush | 0 = Do not issue flush message<br>1 = Issue flush message before xmit (def) |
| 15 | Check Code | 0 = Do not check completion code<br>1 = Check completion code after xmit (def) |
| 16 | Command | 1 = Reset Transmitter<br>2 = Flush the firmware receive buffer |

| | | |
|---|---|---|
| Dlcp Config | = | The config byte to write to the DLCP chip (Should not be used under normal conditions). |
| Set Speed | = | Command to change the speed of the DLCP chip. |
| Receive Timeout | = | The time to wait for a message 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 DLCP chip. |
| Disable Retry | = | Writes terminate auto-retry command to the DLCP before each transmission. |
| Reconfigure | = | Write the config register of the DLCP before each transmission. |
| Reset | = | Writes a reset command to the DLCP before each transmission. |
| Flush Previous | = | Clears the RFIFO with flush byte commands before each transmission. |
| Issue Flush | = | Writes a flush message command to the DLCP before each transmission. |
| Check Comp. Code | = | Firmware waits for a completion code after a xmit and returns an error based on the completion code. This also causes the RFIFO to be flushed and an flush message command to be issued. |

Notes: The Class 2 firmware is interrupt driven and interrupts should not be masked off in the dlcp chip.

A Tclear will set the parameters back to their default values.

## EXAMPLES:

Call dlcpParams(13, 1) ..... Causes the MSP firmware to clear the RFIFO ..... before each transmission.
Call dlcpParams(14, 1) ..... Causes the MSP to issue a flush message command ..... before each transmission.

## recvDLCP

The recvDlcp functional call receives a message using the DLCP chip on the MSP board. The message must be disassembled by the caller as the function receives the message transparently.

**Visual BASIC Declaration:**

Public Sub recvDlcp(ErrCode As Integer, rmsg() As Byte, rmsgSize As Integer, ByVal timeout As Double)

**Call recvDLCP(ErrCode, rmsg, rmsgSize, timeout)**

**WHERE:**

**ErrCode**
= The returned error code.

**rmsg**
= The returned message array.

**rmsgSize**
= 0 to 4096. The size of the Rmsg array.

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

**EXAMPLES:**

```
Dim ErrCode As Integer
Dim rmsg(12) As Byte
Dim rmsgSize As Integer

rmsgSize = 12
Call recvDlcp(Errcode, rmsg, rmsgSize, 1) ..... Receive message.
```

## sendDLCP

The sendDlcp functional call sends a message using the DLCP chip on the MSP board. The message must be fully assembled by the caller as the function transmits the message transparently.

**Visual BASIC Declaration:**

Public Sub sendDlcp(ErrCode As Integer, smsg() As Byte, ByVal smsgSize As Integer, ByVal timeout As Double)

**Call sendDLCP(ErrCode, smsg, smsgSize, timeout)**

**WHERE:**

**ErrCode**
= The returned error code.

**smsg**
= The message array to send.

**smsgSize**
= 0 to 4096. The size of the Smsg array.

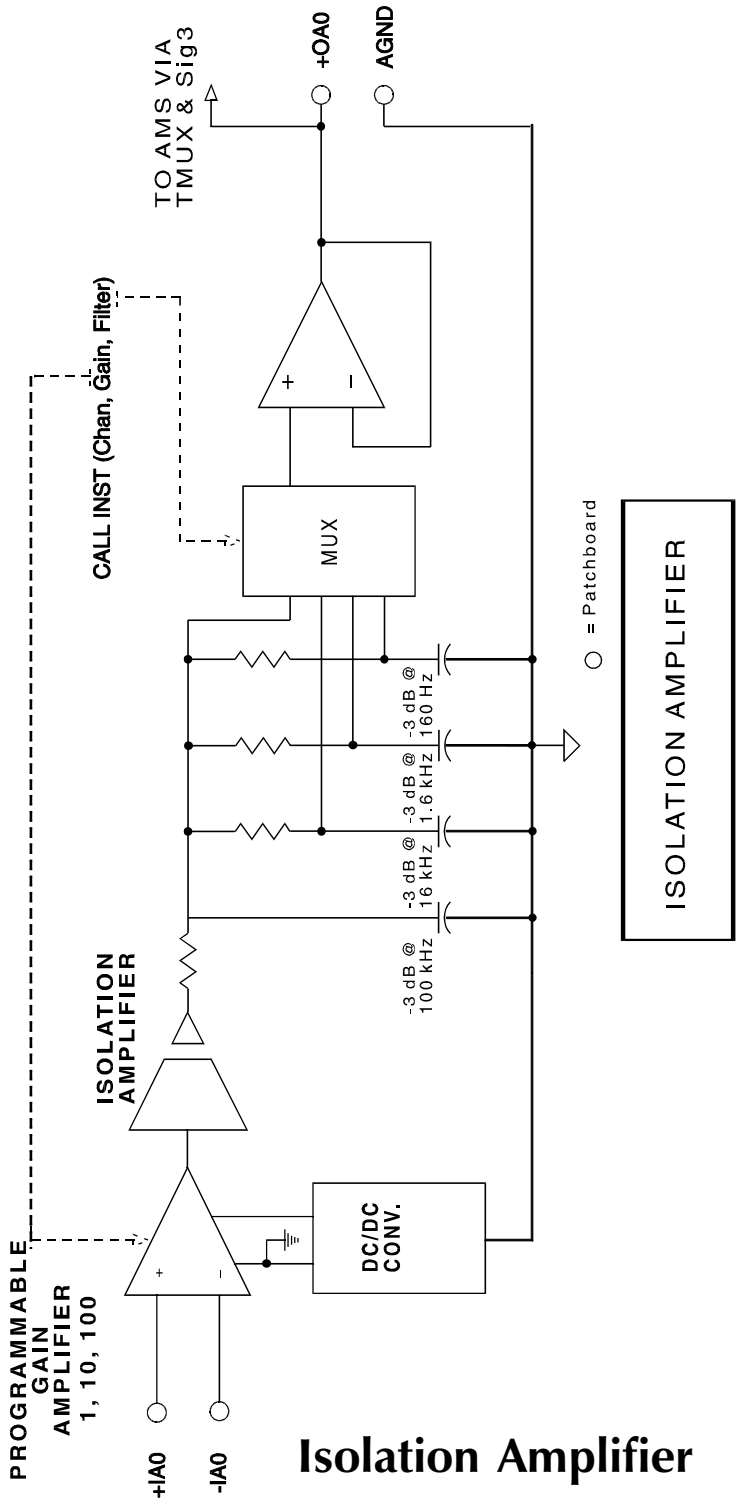
**timeout**
= 0 to 65. The time to wait in seconds for an idle line.

**EXAMPLES:**

```
Dim Errcode As Integer
Dim smsg(4) As Byte
Dim timeout As Double

smsg(1)=0
smsg(2)=1
smsg(3)=2
smsg(4)=3
Call sendDlcp(Errcode, smsg, 4, 1) ..... Send 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**

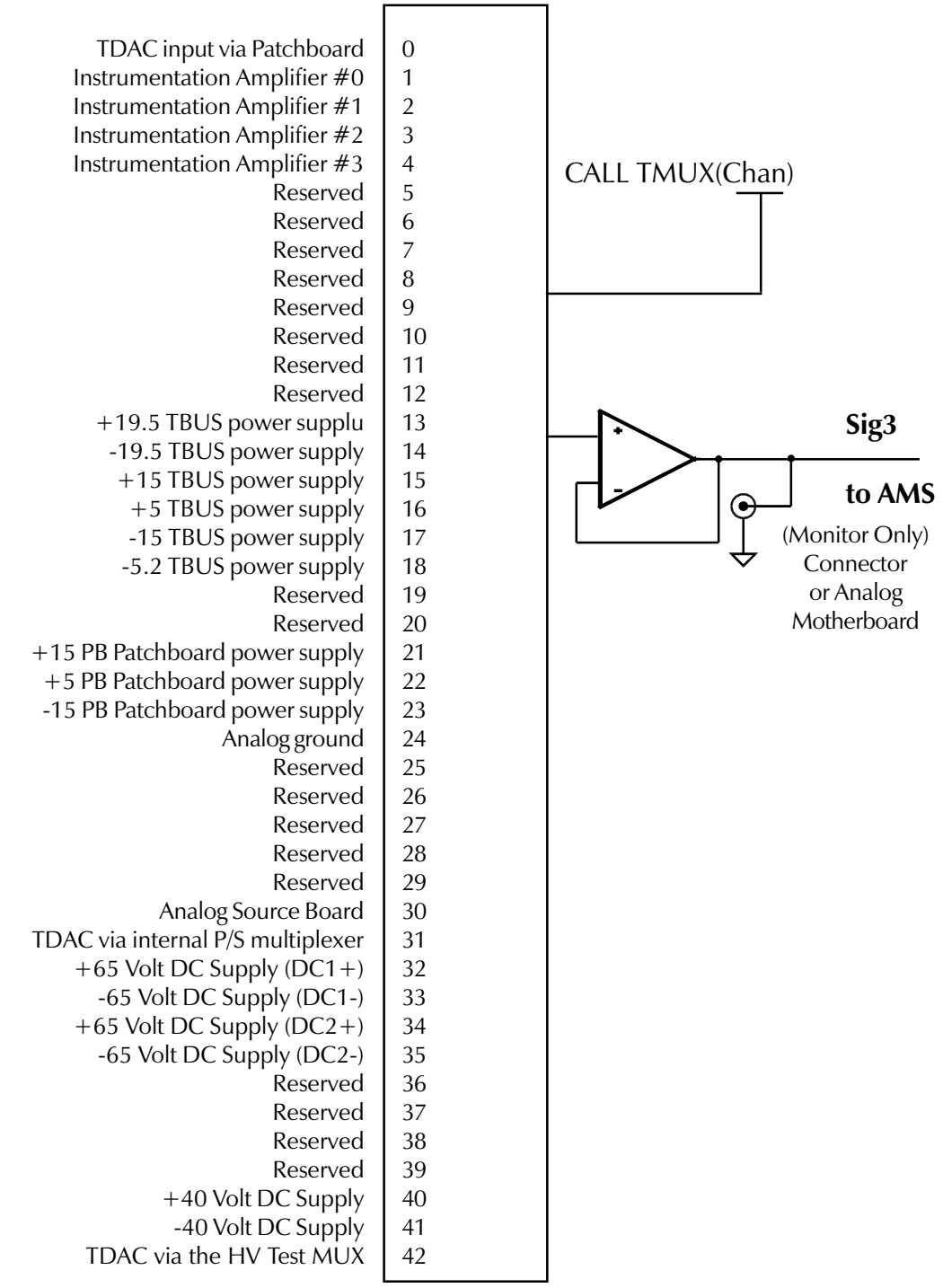
| | | |
|---|---|---|
| = | 0 | 1. |
| = | 1 | 10. |
| = | 2 | 100. |

**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 - 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.
100:020 (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.
100:100 (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. |
| 100:230 (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. |