OPTOMUX PROTOCOL GUIDE

Form 1572-180302—March 2018

43044 Business Park Drive • Temecula • CA 92590-3614
Phone: 800-321-OPTO (6786) or 951-695-3000
Fax: 800-832-OPTO (6786) or 951-695-2712
www.opto22.com

Product Support Services
800-TEK-OPTO (835-6786) or 951-695-3080
Fax: 951-695-3017
Email: support@opto22.com
Web: support.opto22.com

Optomux Protocol Guide
Form 1572-180302—March 2018

Opto 22
Automation Made Simple.

Table of Contents

1: Introduction

OVERVIEW

Opto 22's Optomux system provides low-cost, distributed access to both digital and analog input/output (I/O) points. Depending on your Opto 22 hardware, the Optomux protocol can be used over either serial (RS-422/485) or Ethernet networks. See the comparison chart on page 4.

This guide describes the Optomux protocol and the Optomux Driver. All commands for both digital and analog I/O are described in detail and examples included for both the Optomux protocol and the driver. Use the driver for easier programming, or write your own Optomux driver or application for custom hardware using the protocol.

A Little History

On Drivers

If you've used an Opto 22 driver for Optomux in the past, it was either the original Optoware (16 bit) or the more recent Optomwd.dll (32 bit).

The new Optomux Protocol Driver documented in this guide, however, is recommended for any new development. It offers the ability to use Ethernet as well as serial with brain boards that are Ethernet capable, such as the E1 and E2. This new driver works with all Optomux brain boards (B1, B2, serial B3000, E1, and E2) and can be used for both Ethernet and serial communications. To migrate applications to the new protocol, see the migration information starting on page 12.

If you are only making changes to an existing system and do not need the Ethernet capability, however, you can still use the older drivers. Optomwd does not have specific documentation; to use it, simply look at the header files and the examples provided. If you need Optoware documentation, the *Optoware Manual* (form 0092) is archived on our website.

On the Optomux Protocol

The previous manual for the Optomux Protocol (form 203), included hardware information for B1 and B2 brain boards as well as protocol commands. This *Optomux Protocol Guide* includes all the protocol information that used to be in form 203. Since Optomux is used with other brain boards as well (serial B3000, E1, and E2), the B1/B2 hardware information has been separated into its own manual, the *B1 and B2 User's Guide* (form 1574).

Migration Paths

For information on migrating your Optomux system to Ethernet communications and integrating newer SNAP hardware, see the *E1 and E2 Architecture and Migration Overview* (form 1567). The overview is available on our website, www.opto22.com; to find it quickly, search on the form number.

ABOUT THIS GUIDE

This guide assumes that you already know how to program software applications in the language of your choice for the device that will be running them. Examples for the Optomux Driver are shown in Visual Basic.

This guide includes the following sections:

Chapter 1: Introduction—information about this guide and how to reach Opto 22 Product Support

Chapter 2: Using the Optomux Driver—how to use the driver to develop custom applications

Chapter 3: Using the Optomux Protocol—complete description of the Optomux protocol and how to use it.

Chapter 4: Command Directory—lists all commands for both the driver and the Optomux protocol and refers you to the page number for the command.

Chapter 5: Setup Commands

Chapter 6: I/O Configuration Commands

Chapter 7: Digital Read/Write Commands

Chapter 8: Digital Latch Commands

Chapter 9: Digital Counting Commands

Chapter 10: Digital Time Delay and Pulse Commands

Chapter 11: Digital Pulse Duration Measurement Commands

Chapter 12: Analog Read/Write Commands

Chapter 13: Analog Input Range Commands

Chapter 14: Analog Offset/Gain Commands

Chapter 15: Analog Waveform Commands

Chapter 16: Revision Identification Command

Appendix A: Troubleshooting Optomux

Appendix B: ASCII-Hex Tables

Appendix C: Reading Negative Numbers and Temperature

Document Conventions

- `Courier` typeface indicates text to be typed. Unless otherwise noted, such text may be entered in upper or lower case.
- Italic typeface indicates emphasis and is used for book titles. (Example: “See the *OptoControl User’s Guide* for details.”)
- Key press combinations are indicated by hyphens between two or more key names. For example, Shift+F1 is the result of holding down the Shift key, then pressing and releasing the F1 key. Similarly, Menu commands are sometimes referred to with the Menu > Command convention. For example, “Select File > Run” means to select the Run command from the File menu.

- Numbered lists indicate procedures to be followed sequentially. Bulleted lists (such as this one) provide general information.

Related Documents

Refer to the following documents for information on hardware used with Optomux. All documents are available for download from our website, www.opto22.com. The easiest way to locate one is to search on its form number.

For information on	See document	Form #
Moving from serial to Ethernet Optomux	*E1 and E2 Architecture and Migration Overview*	1567
E1 and E2 brain boards	*E1 and E2 User's Guide*	1563
B1 and B2 brain boards	*Optomux 16-Channel Digital and Analog Brain Boards Data Sheet*	0463
B3000 brain boards	*SNAP Analog/Digital Mistic/Optomux Brain Data Sheet*	0787
PCI-AC48 adapter card	*PCI-AC48 User's Guide*	1520
RS-232 to RS-485 conversion	*AC7 A/B User's Guide*	0233

FOR HELP

If you have problems using Optomux or the Optomux Protocol Driver and cannot find the help you need in this guide or on our website, contact Opto 22 Product Support.

Phone:	800-TEK-OPTO (800-835-6786 toll-free in the U.S. and Canada) 951-695-3080 Monday through Friday, 7 a.m. to 5 p.m. Pacific Time
Fax:	951-695-3017
Email:	support@opto22.com
Opto 22 website:	www.opto22.com

NOTE: Email messages and phone calls to Opto 22 Product Support are grouped together and answered in the order received.

OPTOMUX PROTOCOL BRAIN COMPARISON CHART

The following table compares Opto 22's Optomux-capable brains and brain boards: B1, E1, B2, E2, and serial B3000. Features shown are for the Optomux protocol. Features are different if the brain is used with another protocol. See the brain's data sheet for specifications.

Feature	B1	E1	B2	E2	B3000
Optomux Digital Features					
Read/write to point	●	●			●
Input latching	●	●			●
Counting	●	●			●
Pulse duration measurement	●	●			●
Pulse generation	●	●			●
Time delays (10 ms resolution)	●	●			●
Watchdog timer	●	●			●
Optomux Analog Features					
Read/write to point in Engineering units			●	●	●
Input averaging			●	●	●
Minimum/maximum values (peak/valley recording)			●	●	●
Out of range testing (high/low)			●	●	●
Offset and gain calculation			●	●	●
Waveform generation			●	●	●
Watchdog timer			●	●	●
Networks					
Serial (RS-422/485)	●	●	●	●	●
Ethernet		●		●	
Module families					
SNAP modules					●
G1 digital modules	●	●			
G1 analog modules			●	●	
G4 digital modules	●	●			
Quad Pak modules	●	●			
Integral I/O racks	●	●			
Additional Protocols Supported					
Modbus/TCP		●		●	
OptoMMP		●		●	
mistic I/O					●

2: Using the Optomux Driver

OVERVIEW

If you are using the Optomux Protocol Driver to develop custom applications, read this chapter to understand how to use the driver. The driver makes programming easier because it takes care of building command messages, calculating checksums, and processing responses. Note that the toolkit is not compatible with .NET programming languages.

This chapter assumes that you are already familiar with using extension DLLs and with Visual Basic or C++.

System Requirements

The following hardware/software requirements for using the Optomux Protocol Driver are suggested minimums. Application size and development may raise the minimum hardware requirements.

- Microsoft® Windows® 2000 and XP
- Microsoft Visual Studio® Version 6 (and with latest service packs) or Visual Basic® 6
- 1 GHz processor or higher (depends on application performance)
- 256 Megabytes of RAM or Microsoft's recommendation (whichever is larger)
- Driver requires approximately 5 megabytes (additional space required for client application).

Installation

The Optomux Protocol Driver is included on the CD that came with your E1 or E2 brain board, and it is automatically installed with the rest of the software on the CD. Windows examples and dlls are provided.

To get started with the driver, open QuickStart.htm, which is installed by default in C:\Program Files\Opto22\E1_E2.

DRIVER BASICS

All that is required is to open a handle to the Optomux I/O unit, send a power-up clear, send and receive data, and then close the handle. The handle is kept open during normal communications.

Always remember to check for errors.

The rest of this chapter details the driver functions and how to handle errors, and provides migration information for those moving from an earlier Opto 22 Optomux driver.

OPTOMUX PROTOCOL DRIVER FUNCTIONS

The following table lists the available functions. For details and examples, see the referenced pages.

Function Name	Description	See
Open_Optomux() [in] string pMethod [in] long Address [in] long Timeout_Ms	Open a handle to an Optomux unit. pMethod—communication method description Address—Optomux Protocol address of the brain Timeout_Ms— timeout of the connect cycle. Note: Timeouts may vary widely depending on the media and distance to the destination host. Return: Return values greater than or equal to zero indicate a valid handle. Return less than zero is a 32-bit error code.	page 7
Send_Receive_Optomux() [in] long iHandle [in] long icommand [out/in] l_array iPositions [out/in] l_array iModifiers [out/in] l_array iData_Array [in] long Timeout_Ms	Perform a complete message transaction to the Optomux brain with a traditional positions array. Create and send command, wait for and decode response. iHandle—a valid handle to an Optomux object (created through Open_Optomux) iCommand—command number (see Chapter 4: Command Directory) iPositions—a 16-position array of the points to be affected by the command (if applicable). If less than 16 elements, the list must be terminated by a -1. (The last element in the array must have a value of -1.) iModifiers—a 2-position array of the modifier arguments (if applicable) iDataArray—a 16-position array of the data arguments (if applicable) Timeout_Ms— timeout of the receive in milliseconds Return: 0 = success < 0 = a 32-bit error code	page 8
Send_Receive_Linear_Optomux() [in] long iHandle [in] long iCommand [out/in] l_array iPositions [out/in] l_array iModifiers [out/in] l_array iData_Array [in] long Timeout_Ms	Same as Send_Receive_Optomux except for the way positions are indicated. Rather than a list of points to be changed by the command, the positions array is filled with 0s and 1s. A 1 indicates that the corresponding point should be changed by this command.	page 9
Error_Optomux() [in] long Return_code [in/out] string Product_Message [in/out] string Error_Message [in/out] long Product_Message_Length [in/out] long Error_Message_Length	Retrieve descriptive error messages for error codes reported by the library. Also return an "old style" 16-bit error code for applications that used optomwd.dll. See page 13. Return_code— the 32-bit bitmask code returned from driver functions (except this one). The lower 16 bits indicate the error. (See page 13.) Product_Message—up to a 128-character string for which library reported the error Error_Message—up to a 256-character string for the error message the library reported Product_Message_Length—length of the product string (pass maximum characters, receive back the actual length of the message) Error_Message_Length—length of the error string (pass maximum characters, receive back the actual length of the message) Return: A 16-bit-like error code. Note: This function translates most of the error codes. However, since the libraries are different, some errors are lumped together. See "Handling Errors in Migrated Applications" on page 13.	page 10

Function Name	Description	See
Close_Optomux() [in] long iHandle	Close an open Optomux handle. iHandle—handle to an open Optomux brain Return: none	page 11

The Open_Optomux Function

The Optomux Protocol Driver uses a communication layer that relies on a string to describe the communication interface. (Prior versions of drivers assumed a serial communication method for Optomux.)

The communication string notation looks like this:
`(driver)|(driver arguments separated by colons)|(connection-specific arguments)`

The following sample method string opens a serial interface at COM 2 with a baud rate of 38,400.
`serial|com:2|-b 38400`

The Open_Optomux function looks like the following examples. The Optomux address used is 14 and the timeout is 1000 ms.

C/C++ Example

```
returncode = Open_Optomux("serial|com:2|-b 38400",14,1000);
// greater than zero values returned in the error code is a valid handle
if (returncode < 0)
{
    // handle the error
}
else
{
    // save the handle for use in subsequent calls to the driver
    My_Handle = returncode;
}
```

Visual Basic Example (serial)

```
returncode = Open_Optomux("serial|com:2|-b 38400",14,1000)
' greater than zero values returned in the return code indicate a valid
handle
if(returncode < 0) Then
' handle the error
Else
' save the handle for use in subsequent calls to the driver
  My_Handle = returncode
End If
```

Visual Basic Example (Ethernet)

```
returncode = Open_Optomux("ip|udp:10.192.54.15:5000",14,1000)
' greater than zero values returned in the return code indicate a valid
handle
if(returncode < 0) Then
' handle the error
Else
' save the handle for use in subsequent calls to the driver
  My_Handle = returncode
End If
```

The Send_Receive_Optomux Function

Commands and responses are processed through the Send_Receive_Optomux function. The first action after opening a handle is to initialize the unit by sending a Power-Up Clear (see page 33) and point configuration information (configure positions, watchdogs, timer resolution, temperature probe type, offset, and gain). Until the Power-Up Clear command is sent, the Optomux unit will not respond to any other command. (All available commands are listed in Chapter 4: Command Directory.)

NOTE: The handle, address, command, positions, modifier, and data array notations are preserved for ease of migration from the previous driver. The new member function appears very similar, as shown below.

C/C++ Power-Up Clear Example

```
returncode =
Send_Receive_Optomux(My_Handle,0,Positions,Modifiers,DataArray,1000);

// inspect the returncode
if(returncode < 0)
{
    // handle the error
}
```

Visual Basic Power Up Clear Example

```
returncode = _
Send_Receive_Optomux(My_Handle,0,Positions,Modifiers,DataArray,1000)

' inspect the returncode
if(returncode < 0) Then
' handle the error
End If
```

C/C++ Turn On Digital Outputs Example

```
// turn on outputs 2 and 9
Positions[0] = 2;
Positions[1] = 9;
Positions[2] = -1; //indicates the end of the list

// send-receive the command
returncode =
Send_Receive_Optomux(My_Handle,9,Positions,Modifiers,DataArray,1000);

// inspect the returncode
if(returncode < 0)
{
    // handle the error
}
```

Visual Basic Turn On Digital Outputs Example

```
' turn on outputs 2 and 9
Positions[0] = 2
Positions[1] = 9
Positions[2] = -1

' send-receive the command
returncode = _
Send_Receive_Optomux(My_Handle,9,Positions,Modifiers,DataArray,1000)

' inspect the returncode
```

```
if(returncode < 0) Then
' handle the error
End If
```

MyHandle is a handle of the Optomux object. This handle not only refers to the communication port but also the address of the brain.

The Send_Receive_Linear_Optomux Function

Provided for convenience as an alternative to Send_Receive_Optomux, the Send_Receive_Linear_Optomux() function uses a set of Boolean flags in the positions array to select points to be affected.

The following table compares equivalent positions array values between the original packed positions function and the linear expanded positions function. Note that Optomux indexes start from zero.

Index	Positions (Packed Method)	Positions (Linear Method)
0	2	0
1	5	0
2	3	1
3	9	1
4	15	0
5	-1	1
6	(don't care)	0
7	(don't care)	0
8	(don't care)	0
9	(don't care)	1
10	(don't care)	0
11	(don't care)	0
12	(don't care)	0
13	(don't care)	0
14	(don't care)	0
15	(don't care)	1

C/C++ Turn On Digital Outputs Example

```
// the positions array is initialized to zeroes
// turn on outputs 2 and 9 (starting from zero)
Positions[2] = 1;
Positions[9] = 1;

// send-receive the command
returncode =
Send_Receive_Linear_Optomux(My_Handle,9,Positions,Modifiers,DataArray,100
0);

// inspect the returncode
if(returncode < 0)
{
    // handle the error
}
```

Visual Basic Turn On Digital Outputs Example

```
' the positions array is initialized to zeroes
' turn on outputs 2 and 9 (starting from zero)
Positions[2] = 1
Positions[9] = 1

' send-receive the command
returncode = _
Send_Receive_Linear_Optomux(My_Handle,9,Positions,Modifiers,DataArray,100
0)

' inspect the returncode
if(returncode < 0) Then
' handle the error
End If
```

The Error_Optomux() Function

The Optomux Protocol Library uses a 32-bit return code, which includes the error code and a product identifier. The combination of these two helps identify which dll reported the error. (Read more in "Dealing with Errors" on page 11.)

Error_Optomux() has two purposes. The first is to return text messages for the 32-bit return code (a product message and an error message). The second is useful if you are migrating from the older optomwd.dll: the function also converts the error code to a similar 16-bit error code in order to minimize the need to modify existing applications. See "Handling Errors in Migrated Applications" on page 13 for additional information.

C/C++ Example

```
char  Product_String[128];
char  Error_String[256];
int   Returncode;
short Error16;
int   Product_String_Length;
int   Error_String_Length;

Returncode = Send_Receive_Optomux(MyHandle,9,Positions,Modifiers,
            DataArray,1000)
// a driver function generates a return code (named Returncode)

// inspect the returncode
if (Returncode < 0)
{
    // initialize the string lengths
    Product_String_Length = 128;
    Error_String_Length = 256;

    // retrieve product message, error message, and convert the error
    Error16 = Error_Optomux(Returncode,Product_String,Error_String,
              Product_String_Length,Error_String_Length);

    // display the messages
}
```

Visual Basic Example

```
Dim Product_String As String
Dim Error_String As String
Dim Returncode As Long
Dim Error16 As Integer
Dim Product_String_Length As Integer
Dim Error_String_Length As Integer
Returncode = Send_Receive_Optomux(MyHandle,9,Positions,Modifiers,
            DataArray,1000)

'a driver function generates a return code (named returncode)

If (Returncode < 0) Then
  ' initialize the string lengths and strings
  Product_String_Length = 128
  Error_String_Length = 256
  Product_String = String(128, Chr(0))
  Error_String = String(256, Chr(0))
  ' retrieve product message, error message, and convert the error
  Error16 = Error_Optomux(Returncode, _
              Product_String, _
              Error_String, _
              Product_String_Length, _
              Error_String_Length)

  ' display the messages
End If
```

The Close_Optomux() Function

The Close_Optomux () function closes an open Optomux handle.

C/C++ Example

```
Close_Optomux(My_Handle)
```

Visual Basic Example

```
Close_Optomux(My_Handle)
```

DEALING WITH ERRORS

At some point, one of the driver functions will report an error. All of the member functions except the Error_Optomux function return a 32-bit return code that includes the error. Error_Optomux translates the error into an older (Optomwd.dll-compatible) 16-bit error code for backwards compatibility and converts the return code to two readable ASCII messages, Product_Message and Error_Message.

Product_Message shows the "product" that reported the error, for example, "Optomux Protocol Library." Error_Message describes the error, for example, "Timeout. No response from device. Check hardware connection, address, power, and jumpers."

The most common errors are listed in "Common Optomux Errors" on page 140.

Of the 32 bits returned by functions like Send_Receive_Optomux, only the lower 16 bits include error code information. The other bits include a product code and a severity code. See `iolib_error_codes.h`, specifically the #defines beginning with `O22_PRODUCT_` for a list of possible product codes. These codes indicate which dll reported the error. The same file includes #defines beginning with `O22_RESULT_` for errors. Each error code is followed by a comment describing the error.

If you want to use these error codes for error handling, you can extract the lower 16 bits from the value returned by Send_Receive_Optomux and other functions.

In C++, use the macro in `iolib_error_codes.h` like this:

```
error_result = IOLIB_ERROR_ERROR_CODE(result);
```

or simply:

```
error_result = 0xFFFF & result;
```

In Visual Basic, you can extract the lower 16 bits can by converting them to a hex string, dropping all but the right-most four characters, and converting back to a number:

```
error_result = Val("&h" + Right(Hex(result), 4))
```

Example:

```
result = Send_Receive_Optomux(m_Handle, iCommand&, PositionArray&, _
         ModifierArray&, m_Values(0), 1000 * m_Timeout)
' at this point, if no PUC has been sent to the Optomux brain, and
' iCommand was something other than 0 (PUC), the result will be
-2126197482
' To extract just the error number, get the least-significant 4 nibbles:
error_result = Val("&h" + Right(Hex(result), 4))

' now process the error code
' O22_RESULT_OPTOMUX_PROTO_PUC_EXPECTED is -13034 (see
iolib_error_codes.h)
if (error_result = -13034) then
   ' Optomux brain has cycle power since last command was sent
   ' send a Power-up Clear command
   result = Send_Receive_Optomux(m_Handle, 0, PositionArray&, _
                     ModifierArray&, m_Values(0), 1000 * m_Timeout)

   ' log PUC
endif
```

MIGRATING APPLICATIONS

Identifying the Application's Toolkit

Opto 22 has published two previous Optomux Protocol toolkits. When migrating applications to the new version described in this guide (the Optomux Protocol Driver), you'll need to identify which toolkit was used to build your application.

- Optoware, the first toolkit, was for 16-bit DOS/Windows 3.1 environments. Optoware's signature is that it had a single function named "optoware." Searching the source code for this function name will determine if the application was based on this library.
- OptoDriver Toolkit provided an update for 32-bit Windows operating systems (such as Windows 95, Windows 98, Windows Me, Windows NT, and Windows 2000). Signatures of this toolkit are a function named "SendOptoMux" and its reliance on a dynamic link library (dll) named OptoMWD.dll.

Because the Optomux protocol is a simple ASCII protocol, many developers created their own protocol engines. These applications may also be migrated, but care must be taken to evaluate how the application communicated to the I/O.

NOTE: Because the Optomux Protocol Driver assumes a processor architecture of 32 bits or larger, all 16-bit applications (such as those based on Optoware) must first migrate to the new 32-bit target platform.

Modifying the Application

Port and Addressing Changes

Addresses are handled differently in the new driver due to the addition of Ethernet capability. Optoware had special command numbers (100 and above) to open and configure a serial port; these commands are now obsolete. The older Opto22MwdPortOpenApi() function opened just the serial port, not a connection to the Optomux device.

Since the newer Open_Optomux() function was designed for both serial and Ethernet, it includes both the port and the address as parameters in the function. Make sure to call Open_Optomux() for each address you are using, so each address will have a unique handle. (See page 7.)

For the same reason, the older functions such as SendOptoMux() used the address as a parameter. However, since the address has already been passed as part of the open command, the newer functions such as Send_Receive_Optomux do not require the address as a parameter, only the handle.

Changes When Migrating from Optoware (16-bit Windows)

If your existing application used Optoware, replace the following functions as shown.

Optoware Function	Change to . . .

optoware(…,SET_SERIAL_PORT_NUMBER,…)
optoware(….CONFIGURE_SERIAL_PORT,…)
	Open_Optomux()

optoware(…,(communication commands),…)	Send_Receive_Optomux()
Application Shutdown	Close_Optomux()

Changes When Migrating from OptoMWD.dll (32-bit Windows)

If your existing application used OptoMWD.dll, replace the following functions:

OptoDriver Toolkit Function	Change to . . .

Opto22MwdPortOpenApi()
Opto22MwdPortOpenAC37()
	Open_Optomux()

SendOptoMux()	Send_Receive_Optomux()
Opto22MwdPortClose()	Close_Optomux()
O22ErrorAsString()	Error_Optomux()

In addition, values are returned differently for the following commands:

- 64—Read Binary On/Off Status
- 66—Read Binary Latches
- 67—Read and Clear Binary Latches

For these commands, OptoMWD.dll returned a 32-bit mask with sign extension in the upper 16 bits. In other words, if the 15th bit was set, bits 16–31 were also set.

Optomux Protocol Driver also returns a 32-bit mask. The lower 16 bits contain values for the associated points, but the upper 16 bits are always zero-filled. If your application evaluates the entire 32-bit number, you'll need to change it for the new driver.

Handling Errors in Migrated Applications

The Optomux Protocol Library uses a 32-bit return code, which includes the error code and a product identifier. The combination of these two helps identify which dll reported the error.

Error_Optomux() returns text messages for the 32-bit return code and also converts the error code to a similar 16-bit error code. Use this returned 16-bit error code with existing error handling routines that formerly handled errors returned by optomwd.dll. Note that since additional error codes have been added for clarity, mappings are not necessarily one-to-one.

For more information, see "Dealing with Errors" on page 11 and "Common Optomux Errors" on page 140.

Old error value	Old optoerr.rh constant (optomwd.dll compatible)	iolib_error_codes.h constant	Value of constant
Errors Originating from the Brain			
-1	OPTOMUX_PUC_EXPECTED	O22_RESULT_OPTOMUX_PROTO_PUC_EXPECTED	-13034
-2	OPTOMUX_UNDEFINED_COMMAND	O22_RESULT_OPTOMUX_PROTO_UNDEF_COMMAND	-13035
-3	OPTOMUX_CHECKSUM_ERROR	O22_RESULT_OPTOMUX_PROTO_CHECKSUM_ERROR	-13036
-4	OPTOMUX_INPUT_BUFFER_OVERRUN	O22_RESULT_OPTOMUX_PROTO_INPUT_BUFFER_OVER	-13037
-5	OPTOMUX_NON_PRINTABLE	O22_RESULT_OPTOMUX_PROTO_NON_PRINTABLE_ASCII	-13038
-6	OPTOMUX_DATA_FIELD_ERROR	O22_RESULT_OPTOMUX_PROTO_DATA_FIELD_ERROR	-13039
-7	OPTOMUX_WATCHDOG_TIMEOUT	O22_RESULT_OPTOMUX_PROTO_WATCHDOG_TIMEOUT	-13040
-8	OPTOMUX_INVALID_LIMITS	O22_RESULT_OPTOMUX_PROTO_INVALID_LIMITS_SET	-13041
Errors Reported by the Driver			
-20	INVALID_CMD_ERROR	O22_RESULT_INVALID_DRIVER_FUNCTION	-13005
-20	INVALID_CMD_ERROR	O22_RESULT_UNDEFINED_COMMAND	-1
-21	INVALID_MODULE_ERROR	O22_RESULT_MISCONFIGURED_POINT	-13020
-22	DATA_RANGE_ERROR	O22_RESULT_INVALID_DATA_FIELD	-6
-25	INVALID_ADDR_ERROR	O22_RESULT_INVALID_ADDRESS	-56
-29	TIMEOUT_ERROR	O22_RESULT_TIMEOUT	-9
-31	RET_CHECKSUM_ERROR	O22_RESULT_DVF_MISMATCH	-2
-33	SEND_ERROR	O22_RESULT_DATA_SEND_ERROR	-13004
-34	EVENT_RANGE_ERROR	O22_RESULT_COMMAND_VALIDATION_FAILED	-13033
-102	PORT_INITIALIZE_ERROR	O22_RESULT_DRIVER_NOT_FOUND	-203
-102	PORT_INITIALIZE_ERROR	O22_RESULT_BAD_STREAM_METHOD_STRING	-13002
-102	PORT_INITIALIZE_ERROR	O22_RESULT_CANNOT_CREATE_FILE	-415
-102	PORT_INITIALIZE_ERROR	O22_RESULT_ERROR_ON_SOCKET_ACCEPT	-442
-102	PORT_INITIALIZE_ERROR	O22_RESULT_ERROR_ON_SOCKET_BIND	-440
-102	PORT_INITIALIZE_ERROR	O22_RESULT_ERROR_ON_SOCKET_OPT_BROADCAST	-13018
-102	PORT_INITIALIZE_ERROR	O22_RESULT_INVALID_METHOD_ARGUMENT	-13003
-102	PORT_INITIALIZE_ERROR	O22_RESULT_INVALID_SOCKET	-411
-102	PORT_INITIALIZE_ERROR	O22_RESULT_SERIAL_EOM_NOT_SET	-13007
-102	PORT_INITIALIZE_ERROR	O22_RESULT_SOCKET_CONNECT_ERROR	-412
-121	PORT_OPEN_ERROR	O22_RESULT_ALREADY_OPEN	-47
-123	HANDLE_INVALID	O22_RESULT_INVALID_HANDLE	-420
-123	HANDLE_INVALID	O22_RESULT_INVALID_STREAM_HANDLE	-53
-125	RECEIVE_ERROR	O22_RESULT_RECEIVE_PROBLEM	-59
-135	OUT_OF_HANDLE_ERROR	O22_RESULT_NOT_ENOUGH_DATA_RETURNED	-40
-135	OUT_OF_HANDLE_ERROR	O22_RESULT_OBJECT_NOT_OPEN	-13028
-135	OUT_OF_HANDLE_ERROR	O22_RESULT_OUT_OF_HANDLES	-414
Any unknown or unsupported old errors are translated to:			
-32768		O22_RESULT_UNSUPPORTED_LIBRARY_FUNCTION	-13001

3: Using the Optomux Protocol

INTRODUCTION

If you are using the Optomux protocol but NOT using an Opto 22 Optomux driver, read this chapter to understand how the protocol works and how to develop custom applications using the protocol.

If you are using an Opto 22 Optomux driver, you do not need to read this chapter. However, you may find the information useful if you want to understand what packets are generated by the driver—for example, if you need to sniff communications on the wire.

OVERVIEW

An Optomux brain (or I/O unit) is an intelligent device that acts as a slave device to a host computer. The host computer issues instructions to Optomux units by sending command messages over the serial communications link. The unit responds to the host by sending response messages.

NOTE: The Optomux protocol used over an Ethernet network works in the same way; the Optomux packet is simply wrapped in a UDP/IP packet to be sent over Ethernet.

All messages between the host computer and Optomux units are made up of ASCII (American Standard Code for Information Interchange) characters. Appendix B: ASCII-Hex Tables shows the ASCII character table. These are the only characters Optomux recognizes.

Checksum Basics

In normal operation, Optomux uses a two-pass protocol with a checksum for message transactions. The host computer initiates the transaction by sending a command to one of the Optomux units in the network. On a serial network, although all serially connected Optomux I/O units receive the message, only the addressed unit will verify that the command is valid and then execute the command. On an Ethernet network, the packet is received and acted upon only by the unit with the IP address specified in the packet. Upon command completion, the unit returns an acknowledgment along with any requested data.

Command messages and response messages containing data from an Optomux unit (connected either serially or via Ethernet) contain a message checksum to ensure secure communications. Each ASCII character has an associated number value. (The letter A, for example, is represented by the decimal number 65.) The message checksum is calculated by adding up the number values that represent all the characters in a message. This sum is converted into two ASCII characters and is appended to the end of the message. A simple "Acknowledge" response does not use a checksum.

The application or driver on the host computer must calculate the checksum and send it along as part of the Command Message to the Optomux unit. When the unit receives the message, it calculates its own checksum and compares that value with the transmitted checksum. If they match, the unit can be sure the message was received correctly.

The same procedure is repeated whenever an Optomux unit returns data to the host computer. When a message is received from the unit, the application on the host computer must calculate the message checksum and compare against the checksum that was transmitted as part of the message.

How Optomux Represents Numbers

Optomux uses the hexadecimal (hex) numbering system to represent numbers in commands and responses. Appendix B: ASCII-Hex Tables includes a hex-to-binary conversion table for reference.

Optomux command messages are transmitted as a series of ASCII characters. Numbers are transmitted as the ASCII number characters 0 through 9 and the uppercase ASCII characters A through F. We will refer to an ASCII character representing a hex number as "ASCII-hex."

EXAMPLE: 15 (decimal) = F (hex) and is transmitted as the ASCII "F" character.

BUILDING COMMAND MESSAGES

All Optomux command messages consist of three main parts: beginning, middle, and end. Command contents are illustrated below and described in the following sections.

	Beginning		Middle				End	
Contents:	SOM	ADDR	CMD	[POS]	[MOD]	[DATA]	CHK	CR
# Characters:	1	2	1	1–4	1	1–64	2	1
Example:	>	CC	i	0040	32	0064	E2	CR

Beginning

The beginning of each message always contains the Start Of Message (SOM) character (>) followed by two characters representing the address of the intended Optomux I/O unit. (00–FF).

If devices other than an Optomux I/O unit are to communicate on the same serial communications link, the beginning of message character `>` cannot appear in any communication between the host computer and the other device(s).

If you are sending Optomux packets over Ethernet rather than serial, the destination of the message is already identified by the IP address, so the address field within the Optomux command is ignored.

Middle

The middle of an Optomux message contains one to four fields. The fields shown in square brackets may or may not be included, depending on the command.

```
cmd[positions][modifier][data]
```

cmd

Cmd always contains a single character that specifies the command to be executed.

[positions]

[Positions] is required by some commands to specify which I/O points are to be affected. The positions field contains from one to four ASCII-hex digits. Each of these digits affects a group of four points.

Since a single hexadecimal digit can represent a 4-bit binary number (0000 through 1111; see page 148), you can specify all the possible combinations of 0s and 1s for four points with one hex digit.

Optomux converts each hex character appearing in the positions field to its 4-bit binary equivalent and uses each bit to read or write to one particular point. The least significant bit corresponds to the least significant point; that is, bit 0 corresponds to point 0.

Example #1: [positions] field contains A5F0. With four characters in the Positions field, the entire range of 16 points is specified.

Point:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data in Binary	1	0	1	0	0	1	0	1	1	1	1	1	0	0	0	0
Data in Hex:	A				5				F				0			

Example #2: [positions] field contains 1E. With only two characters in the positions field, only points in the range 0–7 will be affected by the command.

Point:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data in Binary	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
Data in Hex:									1				E			

Note: When the positions field immediately precedes the end of the Optomux message (that is, there are no other fields between the positions field and the checksum), then leading zeros in the positions field can be omitted.

[modifier]

[Modifier] is required by commands that have more than one possible execution feature. For example, the Set Time Delay command requires a modifier field to indicate what type of time delay is to be used. If required, this field contains one or two ASCII characters.

[data]

[Data] consists of 1–64 characters (depending on the command) and is required by commands that transmit data (for example, to specify analog output values). Data may be a value or a list of values.

End

The end of an Optomux message always contains three characters: two characters representing the checksum followed by a carriage return. The "." character can be used in place of the carriage return (cr).

For debugging purposes, when using a terminal, two "?" characters can be used in place of the checksum characters. The ?? is a wildcard checksum and should not be used in the final application, because it defeats the purpose of checksum verification for message integrity.

Compute the message checksum by adding the decimal values of all the ASCII characters in the message EXCLUDING the start of command character (>). Convert this sum to hex and use the last two digits as the checksum.

Checksum Example

`>08KC01289cr` is a valid command that includes a checksum.

`08KC012` is the part used to calculate the checksum.

NOTE: The start of command character is NOT part of the checksum calculation.

Add the decimal values of the ASCII characters that make up the command, convert the sum to hex, and use the last two digits as the checksum:

ASCII characters:	0	8	K	C	0	1	2
Value of characters:	48	56	75	67	48	49	50
Add the values:	48 + 56 + 75 + 67 + 48 + 49 + 50 = 393						
Convert to hex:	393 decimal = 189 hex						
Checksum is last two digits:	89 hex						

The complete command then becomes `>08KC01289cr`

Alternate forms of this message include `>08KC01289.` or `>08KC012??cr`

CARRYING OUT MESSAGE TRANSACTIONS

The type of message transaction used between the host computer and Optomux is a 2-pass transaction.

For proper 2-pass operation of a B1 or B2 Optomux unit, make sure jumper B10 is installed on the brain board. For E1 and E2 brain boards, and for B3000 brains used in Optomux mode, 2-pass is the only mode supported.

2-Pass Mode

In 2-pass mode, the host computer transmits a command (the first pass) and the Optomux unit returns a complete response on the second pass:

1. The host sends a command message to Optomux (such as `>86K1004Acr`).
2. If the command was executed successfully, the Optomux unit responds with an A followed by a carriage return, or if data is to be returned, an A followed by the data characters, a checksum, and ending with a carriage return. (In the example above, a typical response may be `Acr`.) If the command was not executed for some reason, the Optomux unit responds with an N followed by a 2-digit error code and a carriage return (such as N03cr.) Note that no checksum is returned if the response is an error condition.

4-Pass Mode (B1, E1, B2, and E2 Brain Boards Only)

The 4-pass mode is activated by either removing the B10 jumper before powering up the brain board or by using the "E" command to instruct the unit to use either 2-pass or 4-pass protocol. For more efficient and faster communications, it is highly recommended that B1s, E1s, B2s, and E2s be used in 2-pass mode. The 4-pass mode is useful for troubleshooting a network, however. It works as follows:

1. The host sends a command message (such as `>FFACDcr`).
2. The Optomux unit echoes the message, substituting an A for the `>`, if there was no error (such as `AFFCDcr`). If an error occurred, the unit responds with an N followed by an error code and carriage return (such as `N02cr`). In either case, the command is not executed.
3. The host must now send the unit an E character followed by a carriage return to instruct the unit to execute the command. Example: >FFEcr.
4. If the command was executed successfully, the Optomux unit responds with an A followed by a carriage return, or if data is to be returned, an A followed by the data characters, a checksum, and ending with a carriage return. If the command was not executed, the unit responds with an N followed by an error code and a carriage return (such as `N03cr`).

INTERPRETING THE RESPONSE

Optomux responses can be divided into three types:

- Error response
- Acknowledgment response
- Acknowledgment with data response

Error Response

The error response is a message consisting of the letter N followed by a two-digit error code and a carriage return (example: `N06cr`). This type of response does not return a checksum. (See "Errors Originating from the Brain" on page 140 for suggestions to resolve errors. The table column titled "Protocol" contains the error code.)

Error Codes

Code	Meaning
00	Power-Up Clear Expected — Command Ignored. A command other than "A" (Power-Up Clear) was attempted after power-up or power failure. Once the error is received, it is not necessary to execute a Power-Up Clear command. The next command will be executed normally. **IMPORTANT:** If this error message is received, it means that the Optomux unit has gone through its power-up sequence and has reset all characteristics to defaults. The unit needs to be reinitialized (see page 20).
01	Undefined Command. The command character was not a legal command character. This error may also be received if the unit is an older model that cannot recognize a newer command.*
02	Checksum Error. The checksum received by the brain board did not match the sum of the characters in the command. The command was ignored.
03	Input Buffer Overrun. The command received by the brain board contained more than 71 characters for analog or 16 characters for digital brain boards. The command was ignored.
04	Non-printable ASCII Character Received — Command Ignored. Only characters from 21 hex to 7F hex are permitted in Optomux messages.
05	Data Field Error. Not enough characters received.
06	Communications Link Watchdog Timeout Error
07	Specified Limits Invalid

* Older B1s and B2s were manufactured before July 1987 and do not have jumper B11.
Newer commands are:

SET ENHANCED DIGITAL WATCHDOG
SET TIMER RESOLUTION
READ MODULE CONFIGURATION
HIGH RESOLUTION SQUARE WAVE
RETRIGGER TIME DELAY
GENERATE n PULSES
START ON PULSE
START OFF PULSE

SET ANALOG WATCHDOG USER-DEFINED VALUE
SET ANALOG WATCHDOG
READ MODULE CONFIGURATION
SET TEMPERATURE PROBE TYPE
READ TEMPERATURE INPUTS
CHECKSUM ON ROM
DATE OF FIRMWARE

Acknowledgment Response

An acknowledgment response is a message that consists of the ASCII letter A followed by a carriage return (such as `Acr`). This response is typical of commands that instruct an Optomux I/O unit to perform a function that returns no data. This type of response does not return a checksum.

Acknowledgment with Data Response

The third type of response is the acknowledgment message followed by data, a checksum, and a carriage return (such as `A130110010FFA100D59cr`). The data component of the message can be of variable length. It is composed of fields that are three characters each when using the Read Analog Outputs command or four characters each when using all other commands that return data. The data is returned with the most significant position's field first in sequence; the least significant field is last.

For instance, suppose a command was sent to a digital Optomux unit at address 23 hex to read the counter values for points 0, 2, 4, 6, 8, and 10. The host message would look like the following:
`>23W5555Bcr`

Notice that the positions field sent to the unit is only three characters (555) and the checksum is always two characters (5B). If the response is as follows:
`A123405671111????ABCD000127cr`

The return message can be interpreted as follows:

Point	Hex Data	Decimal Value
0	`0001`	1
2	ABCD	43,981
4	`????`	Point 4 is an output.
6	`1111`	4,369
8	`0567`	1,383
10	`1234`	4,660

With commands that look at status bits of points, the data component of the message is a 4-character data field representing a 16-bit integer value with a one-to-one correspondence between bit and point (bit 5 corresponds to point 5).

INITIALIZING AN OPTOMUX NETWORK

Many of the operating characteristics of an Optomux unit must be set by the host computer using a series of commands. This process is called initializing the Optomux network.

This initialization usually involves sending a Power-Up Clear and a Configure Positions command to each Optomux unit. Commands to set the characteristics for watchdog, timer resolution, and temperature probe type (if used) should also be sent at this time. If using the Offset and Gain capability on the analog Optomux, commands to set gain coefficients and offsets should be sent during initialization for each input point.

When an Optomux unit is powered up or after a Reset command, any prior initialization is lost and the unit initializes itself to a set of default characteristics. The Reset defaults are as follows:

- Turnaround delay is set to 0.
- On a B1, E1, B2, or E2, message protocol (2-pass or 4-pass) is set according to jumper B10.
- Watchdog timer is disabled.
- Analog Optomux - All points are configured to function as inputs with gain set to 1.000 and offsets set to zero.
- Digital Optomux - All points are configured to function as inputs.
- All counters are set to 0.
- All time delays are set to 0.
- All latches are cleared and trigger set for OFF-to-ON.
- Pulse width measurement is set for ON pulses.

An Optomux unit will respond with an `N00cr` error whenever power was cycled. The purpose of this error is to warn you that the unit has lost its configuration and needs to be initialized.

Always reinitialize the Optomux network when an Optomux unit goes through a reset condition. The reset condition can occur whenever a Reset command is sent OR when power is lost and returns. Note that a power-up condition can also be caused by a momentary dip on the + 5 VDC line, which causes the Optomux unit's processor to reset.

CONFIGURING SERIAL PORTS

Before the host computer can communicate with Optomux units, the port characteristics must be set to match those of the unit. Optomux units require a 10-bit data word with the following format:

```
1 Start Bit 8 Data Bits (no parity) 1 Stop Bit
```

Example: ASCII "3" (Hex 33)

Start bit	Data bits								Stop bit
0	1	1	0	0	1	1	0	0	1
LSB								MSB (always 0)	

NOTE: The MSB is used by some equipment as a parity bit; however, Optomux units ignore this bit in host-to-unit transmissions.

Baud Rates

Optomux units can communicate at several different baud rates. The baud rate at the host computer's serial port must match the baud rate on the brain board. Possible baud rates and how to set them for each brain board are shown in the following table.

Baud Rate	B1	B2	B3000	E1	E2
Baud rate set by:	jumpers*	jumpers*	jumpers*	jumpers*	jumpers*
115,200			X		
57,600			X		
38,400	X	X	X	X	X
19,200	X	X	X	X	X
9600	X	X	X	X	X
4800	X	X	X	X	X
2400	X	X	X	X	X
1200	X	X	X	X	X
600	X	X	X	X	X
300	X	X	X	X	X
* See the brain board user's guide for jumper settings.					

CONFIGURING AN ETHERNET CONNECTION (E1 AND E2)

If you are using E1 and E2 brain boards over an Ethernet connection, you can send an Optomux command within a UDP packet. Send only one command per packet. The port used is port 5000.

INTERPRETING ANALOG DATA

This section applies only to analog B2 and E2 brain boards and B3000 brains using analog modules.

Analog Outputs

All analog output values exchanged between the host computer and the Optomux unit are represented by three ASCII hex digits. Analog output modules are offset so that when the host computer instructs the unit to write a value of zero scale (000 hex) to the point, the point goes to its most negative output. When the host computer instructs the unit to write a value of full-scale to the point (FFF hex), the point goes to its most positive output.

```
zero scale  000 Hex = 0 decimal
full-scale  FFF Hex = 4095 decimal
```

When the host computer instructs the Optomux unit to return the current value of an analog output point, the unit returns three hex digits representing the value. No offset is added to the value, and it is returned in the same form that it was sent to the unit. Therefore, there is no need to store the current values of analog outputs in your host application, because they can be retrieved from the unit at any time.

Example

If you want to output 2.32 volts from a DA4 (0 to 5 volt output), do the following:

```
Divide desired value by 5 volts
2.32-volts / 5-volts = 0.464
Multiply result by 4,095 (full-scale in decimal)
1900 = .464 * 4,095
Convert to hex
1900 decimal = 76C hex
```

Therefore, you would instruct Optomux to write the value 76C hex to the output module.

NOTE: Refer to the module's data sheet for detailed information on each module.

Analog Inputs

When the host computer instructs an Optomux unit to return the value of an analog input point, the unit returns four ASCII-hex digits representing the value. Analog input modules are scaled so that when the point is receiving its zero scale input, the unit returns 1000 hex, and when the point is receiving its full-scale input, the unit returns 1FFF hex. (This information does not apply to the Read Temperature Inputs command. See "Read Temperature Inputs" on page 99 for information on handling direct temperature data. Also see "Appendix C: Reading Negative Numbers and Temperature.")

```
zero scale   1000 hex = 4096 decimal
full-scale   1FFF hex = 8191 decimal
```

The first of the four hex digits is normally 1, which represents a 1000 hex (4,096 decimal) offset that the module adds to the value of the input (zero scale is 4,096, not 0). If we subtract this offset (1000 hex) from the value returned by the unit, we end up with the following zero and full-scale readings:

Zero Scale

```
(1000 hex - 1000 hex) = 0 hex = 0 decimal
```

Full-Scale

```
(1FFF Hex - 1000 hex) = FFF hex = 4095 decimal
```

By subtracting 1000 hex (4,096 decimal) from the value returned by the unit, zero scale works out to be 0 decimal, and full-scale is 4,095 decimal. If an input point is receiving an input slightly less than zero scale, the Optomux unit returns a value less than 1000 hex; if you convert the value to decimal and subtract 4,096; you get a negative number, indicating that the input is less than zero scale. If the point is receiving an input more negative than 2.5 percent below zero scale, the unit returns a value of 0000 hex, so subtraction of the offset yields -4,096. This 2.5 percent (approximately) under-range limit is determined by the hardware and cannot be altered. The Optomux unit may return values up to 100 percent over range with some modules.

Therefore, by always subtracting 4,096 from analog input values, you have the following:

no module installed	value is -4,096
input is more negative than 2.5% below zero-scale	value is -4,096
module is below zero-scale	value is less than 0
module is at zero-scale	value is 0
module is at full-scale	value is 4,095
module is above full-scale	value is greater than 4,095

NOTE: In the following examples dealing with analog inputs, it is assumed that the value returned by the Optomux unit has been converted to decimal, and the 4,096 offset has been subtracted. This value is referred to as the decimal value.

Example #1

If you read the value of a 0 to 5 volt input module (for example, AD6) that is receiving -0.122 volts, the Optomux unit returns the value 0F9C. To interpret the return data, do the following:

```
Convert the value to decimal
0F9C hex = 3,996 decimal
Subtract 4,096 offset
3,996 - 4,096 = -100
```

The value -100 indicates that the input is under scale. To determine the value in volts, do the following:

```
5 volts/4,096 * (-100) = -0.122 volts
```

Example #2

If you read the value of a 0 to 5 volt input module (for example, AD6) that is receiving 3 volts, the unit returns the value 199A. To interpret the return data, do the following:

```
Convert the value to decimal
199A hex = 6,554 decimal
Subtract 4,096 offset
6,554-4,096 = 2,458
```

To convert 2,458 to volts, do the following:

```
5 volts/4,096 * 2,458 = 3 volts
```

Calibration: Offset and Gain

Offset

The Optomux unit can be instructed to set offsets for input points. Each time the unit is instructed to return a value for a particular input point, the offset for that point is subtracted from the value of the input and the result is returned to the host. This is an easy method of correcting minor variations. For example, if you are using a 0 to 5 volt input module (for example, AD6), and the lowest possible input to the point is 0.01 volts, it may be desirable to consider this 0.01 volts as zero scale in your host program. This can be accomplished by instructing the Optomux unit to calculate and set the offset for the point while it is receiving 0.01 volts. After this has been done, the unit will return zero scale (1000 hex) when the point is receiving 0.01 volts.

Gain

The Optomux unit can also be instructed to set gain coefficients for input points. Each time the unit is instructed to return a value for a particular input point, the gain coefficient for that point is multiplied by the value of the input and the result is returned to the host. For example, if you are using a 0 to 5 volt input module (such as AD6), and the highest possible input to the point is 4.5 volts, it may be desirable to consider this 4.5 volts as full-scale in your host program. This can be accomplished by instructing the Optomux unit to calculate and set the gain for the point when it is at 4.5 volts. After this has been done, the unit will return full-scale (1FFF hex) when the module is receiving 4.5 volts.

NOTE: The Offset and Gain features should be used only for calibration and to compensate for small deviations in sensor range. Do not use these commands to scale a module with a wide range to a scale with a smaller range (for example, using a 0 to 10 VDC module as a 3 to 5 VDC module), because the values will tend to make large jumps for small increments of input.

4: Command Directory

OPTOMUX COMMAND SUMMARY

This section provides a quick reference to all the analog and digital Optomux commands in both the driver and the protocol.

The remaining chapters cover the commands in detail. The description of each command in the following chapters is formatted as follows:

Command Name Driver Command Number

Versions (Optomux units that support the command) Optomux Command Letter

Purpose: Tells briefly what the command does.

Remarks: Describes in detail how the command is constructed and used.

Driver

Parameters: Lists parameters required with the driver command.

Example: Shows how you might use the command in a Visual Basic application.

Optomux Protocol

Format: Illustrates the correct format for the middle part of the Optomux command message. Brackets "[ ]" are used for clarity to separate fields in the guide. Do not include these brackets in the message.

Example: Shows sample command messages and responses that demonstrate the use of the command.

LIST OF COMMANDS BY FUNCTION

The following tables list all the commands by function, their number (if you are using the driver) and their letter (if you are using the Optomux protocol). Version indicates whether the command is an A(nalog) or

D(igital) command. Digital commands are used with the E1, the B1, and the SNAP serial B3000 brain. Analog commands are used with the E2, the B2, and the B3000. (The SNAP serial B3000 brain recognizes both digital and analog commands in its Optomux mode.)

For a list of Optomux driver commands by number, see page 29.

For a list of Optomux protocol commands by letter, see page 31.

Chapter 5: Setup Commands

Command Name	Driver Command	Optomux Format	Version	See
Power-Up Clear	0	A	A, D	page 33
Reset	1	B	A, D	page 34
Set Turnaround Delay	2	C[data]	A, D	page 35
Set Digital Watchdog	3	D[data]	D	page 36
Set Analog Watchdog	45	D[positions][data]	A	page 37
Set Enhanced Digital Watchdog	71	m[positions][data]	D	page 41
Set Analog Watchdog Timeout	78	m[positions][data]	A	page 42
Set Protocol	4	E[data]	A, D	page 39
Identify Optomux Type	5	F	A, D	page 40
Set Timer Resolution	75	n[data]	D	page 43
Set Temperature Probe Type	76	k[positions][data]	A	page 44

Chapter 6: I/O Configuration Commands

Command Name	Driver Command	Optomux Format	Version	See
Configure Positions	6	G[positions]	A, D	page 47
Configure as Inputs	7	H[positions]	A, D	page 48
Configure as Outputs	8	I[positions]	A, D	page 49
Read Configuration	70	j	A, D	page 50

Chapter 7: Digital Read/Write Commands

Command Name	Driver Command	Optomux Format	Version	See
Write Digital Outputs	9	J[positions]	D	page 51
Write Binary Outputs	65	J[positions]	D	page 52
Activate Digital Outputs	10	K[positions]	D	page 53
Deactivate Digital Outputs	11	L[positions]	D	page 54
Read On/Off Status	12	M	D	page 55
Read Binary On/Off Status	64	M[positions]	D	page 56

Chapter 8: Digital Latch Commands

Command Name	Driver Command	Optomux Format	Version	See
Set Latch Edges	13	N[positions]	D	page 57
Set Off-to-On Latches	14	O[positions]	D	page 58
Set On-to-Off Latches	15	P[positions]	D	page 59
Read Latches	16	Q	D	page 60
Read and Clear Latches	17	R[positions]	D	page 61
Clear Latches	18	S[positions]	D	page 62
Read Binary Latches	66	Q	D	page 63
Read and Clear Binary Latches	67	R[positions]	D	page 64

Chapter 9: Digital Counting Commands

Command Name	Driver Command	Optomux Format	Version	See
Start and Stop Counters	19	T[positions]	D	page 65
Start Counters	20	U[positions]	D	page 66
Stop Counters	21	V[positions]	D	page 67
Read Counters	22	W[positions]	D	page 68
Read and Clear Counters	23	X[positions]	D	page 70
Clear Counters	24	Y[positions]	D	page 71

Chapter 10: Digital Time Delay and Pulse Commands

Command Name	Driver Command	Optomux Format	Version	See
Set Time Delay	25	Z[positions][modifier][data]	D	page 73
Initiate Square Wave	26	Z[positions]L[data]	D	page 75
Turn Off Time Delay/Square Wave	27	Z[positions]G	D	page 76
High Resolution Square Wave	68	Z[positions]M[data]	D	page 77
Retrigger Time Delay	69	h[positions]	D	page 78
Generate N Pulses	72	i[positions][modifier][data]	D	page 79
Start On Pulse	73	k[positions][data]	D	page 80
Start Off Pulse	74	l[positions][data]	D	page 81

Chapter 11: Digital Pulse Duration Measurement Commands

Command Name	Driver Command	Optomux Format	Version	See
Set Pulse Trigger Polarity	28	a[positions]	D	page 83
Trigger On Positive Pulse	29	b[positions]	D	page 85
Trigger On Negative Pulse	30	c[positions]	D	page 86
Read Pulse Complete Bits	31	d	D	page 87
Read Pulse Duration Counters	32	e[positions]	D	page 88
Read and Clear Duration Counters	33	f[positions]	D	page 89
Clear Duration Counters	34	g[positions]	D	page 90

Chapter 12: Analog Read/Write Commands

Command Name	Driver Command	Optomux Format	Version	See
Write Analog Outputs	35	J[positions][data]	A	page 91
Read Analog Outputs	36	K[positions]	A	page 92
Update Analog Outputs	46	S[positions][data]	A	page 93
Read Analog Inputs	37	L[positions]	A	page 94
Average and Read Input	38	M[positions][data]	A	page 95
Start Averaging Inputs	47	T[positions][data]	A	page 96
Read Average Complete Bits	48	i	A	page 97
Read Averaged Inputs	49	U[positions]	A	page 98
Read Temperature Inputs	77	l[positions]	A	page 99
Read Average Temperature Inputs	79	o[positions]	A	page 101

Chapter 13: Analog Input Range Commands

Command Name	Driver Command	Optomux Format	Version	See
Set Input Range	39	N[positions][data]	A	page 103
Read Out-of-Range Latches	40	O	A	page 104
Read and Clear Out-of-Range Latches	41	P[positions]	A	page 105
Clear Out-of-Range Latches	42	Q[positions]	A	page 107
Read Lowest Values	58	a[positions]	A	page 108
Clear Lowest Values	59	b[positions]	A	page 109
Read and Clear Lowest Values	60	c[positions]	A	page 110
Read Peak Values	61	d[positions]	A	page 111
Clear Peak Values	62	e[positions]	A	page 112
Read and Clear Peak Values	63	f[positions]	A	page 113

Chapter 14: Analog Offset/Gain Commands

Command Name	Driver Command	Optomux Format	Version	See
Calculate Offsets	52	g[positions]	A	page 115
Set Offsets	53	W[positions][data]	A	page 117
Calculate and Set Offsets	54	h[positions]	A	page 119
Calculate Gain Coefficients	55	X[positions]	A	page 121
Set Gain Coefficients	56	Y[positions][data]	A	page 123
Calculate and Set Gain Coefficients	57	Z[positions]	A	page 125

Chapter 15: Analog Waveform Commands

Command Name	Driver Command	Optomux Format	Version	See
Set Output Waveform	43	R[positions][modifiers][data]	A	page 127
Turn Off Existing Waveforms	44	R[positions][modifier]	A	page 130
Enhanced Output Waveform	50	V[positions][modifier][data]	A	page 131
Cancel Enhanced Waveforms	51	V[positions][modifier]	A	page 133

Chapter 16: Revision Identification Command

Command Name	Driver Command	Optomux Format	Version	See
Date of Firmware	80	ʹ	A, D	page 135

LIST OF DRIVER COMMANDS BY NUMBER

The following list shows all Optomux Driver Commands by their command number.

Command Number	Command Name	See
0	Power-Up Clear	page 33
1	Reset	page 34
2	Set Turnaround Delay	page 35
3	Set Digital Watchdog	page 36
4	Set Protocol	page 39
5	Identify Optomux Type	page 40
6	Configure Positions	page 47
7	Configure as Inputs	page 48
8	Configure as Outputs	page 49
9	Write Digital Outputs	page 51
10	Activate Digital Outputs	page 53
11	Deactivate Digital Outputs	page 54
12	Read On/Off Status	page 55
13	Set Latch Edges	page 57
14	Set Off-to-On Latches	page 58
15	Set On-to-Off Latches	page 59
16	Read Latches	page 60
17	Read and Clear Latches	page 61
18	Clear Latches	page 62
19	Start and Stop Counters	page 65
20	Start Counters	page 66
21	Stop Counters	page 67
22	Read Counters	page 68
23	Read and Clear Counters	page 70
24	Clear Counters	page 71
25	Set Time Delay	page 73
26	Initiate Square Wave	page 75
27	Turn Off Time Delay/Square Wave	page 76
28	Set Pulse Trigger Polarity	page 83
29	Trigger On Positive Pulse	page 85
30	Trigger On Negative Pulse	page 86
31	Read Pulse Complete Bits	page 87
32	Read Pulse Duration Counters	page 88

Command Number	Command Name	See
33	Read and Clear Duration Counters	page 89
34	Clear Duration Counters	page 90
35	Write Analog Outputs	page 91
36	Read Analog Outputs	page 92
37	Read Analog Inputs	page 94
38	Average and Read Input	page 95
39	Set Input Range	page 103
40	Read Out-of-Range Latches	page 104
41	Read and Clear Out-of-Range Latches	page 105
42	Clear Out-of-Range Latches	page 107
43	Set Output Waveform	page 127
44	Turn Off Existing Waveforms	page 130
45	Set Analog Watchdog	page 37
46	Update Analog Outputs	page 93
47	Start Averaging Inputs	page 96
48	Read Average Complete Bits	page 97
49	Read Averaged Inputs	page 98
50	Enhanced Output Waveform	page 131
51	Cancel Enhanced Waveforms	page 133
52	Calculate Offsets	page 115
53	Set Offsets	page 117
54	Calculate and Set Offsets	page 119
55	Calculate Gain Coefficients	page 121
56	Set Gain Coefficients	page 123
57	Calculate and Set Gain Coefficients	page 125
58	Read Lowest Values	page 108
59	Clear Lowest Values	page 109
60	Read and Clear Lowest Values	page 110
61	Read Peak Values	page 111
62	Clear Peak Values	page 112
63	Read and Clear Peak Values	page 113
64	Read Binary On/Off Status	page 56
65	Write Binary Outputs	page 52
66	Read Binary Latches	page 63
67	Read and Clear Binary Latches	page 64
68	High Resolution Square Wave	page 77
69	Retrigger Time Delay	page 78
70	Read Configuration	page 50
71	Set Enhanced Watchdog Timeout	page 41
72	Generate N Pulses	page 79

Command Number	Command Name	See
73	Start On Pulse	page 80
74	Start Off Pulse	page 81
75	Set Timer Resolution	page 43
76	Set Temperature Probe Type	page 44
77	Read Temperature Inputs	page 99
78	Set Analog Watchdog Timeout	page 42
79	Read Average Temperature Inputs	page 101
80	Date of Firmware	page 135

LIST OF OPTOMUX PROTOCOL COMMANDS BY LETTER

The following list shows all Optomux Protocol Commands by command letter.

Command Letter	Command Name	Version	See
A	POWER-UP CLEAR	D, A	page 33
B	RESET	D, A	page 34
C	SET TURNAROUND DELAY	D, A	page 35

D	SET DIGITAL WATCHDOG
	SET ANALOG WATCHDOG
		D
		A
			page 36
			page 37

E	SET PROTOCOL	D, A	page 39
F	IDENTIFY OPTOMUX TYPE	D, A	page 40
G	CONFIGURE POSITIONS	D, A	page 47
H	CONFIGURE AS INPUTS	D, A	page 48
I	CONFIGURE AS OUTPUTS	D, A	page 49
J	WRITE OUTPUTS WRITE ANALOG OUTPUTS	D A	page 51 page 91
K	ACTIVATE OUTPUTS READ ANALOG OUTPUTS	D A	page 53 page 92
L	DEACTIVATE OUTPUTS READ ANALOG INPUTS	D A	page 54 page 94
M	READ ON/OFF STATUS AVERAGE AND READ INPUTS	D A	page 55 page 95

N	SET LATCH EDGES
	SET INPUT RANGE
		D
		A
			page 57
			page 103

O	SET OFF-TO-ON-LATCHES READ OUT-OF-RANGE LATCHES	D A	page 58 page 104
P	SET ON-TO-OFF LATCHES READ AND CLEAR OUT-OF-RANGE LATCHES	D A	page 59 page 105
Q	READ LATCHES CLEAR OUT-OF-RANGE LATCHES	D A	page 60 page 107
R	READ AND CLEAR LATCHES SET OUTPUT WAVEFORM	D A	page 61 page 127

Command Letter	Command Name	Version	See
S	CLEAR LATCHES UPDATE ANALOG OUTPUTS	D A	page 62 page 93
T	START/STOP COUNTERS START AVERAGING INPUTS	D A	page 65 page 96
U	START COUNTERS READ AVERAGED INPUTS	D A	page 66 page 98
V	STOP COUNTERS ENHANCED OUTPUT WAVEFORMS	D A	page 67 page 131
W	READ COUNTERS SET OFFSETS	D A	page 68 page 117
X	READ AND CLEAR COUNTERS CALCULATE GAIN COEFFICIENTS	D A	page 70 page 121
Y	CLEAR COUNTERS SET GAIN COEFFICIENTS	D A	page 71 page 123
Z	SET TIME DELAY INITIATE SQUARE WAVE TURN OFF TIME DELAY/SQUARE WAVE HIGH RESOLUTION SQUARE WAVE CALCULATE AND SET GAIN COEFFICIENTS	D D D D A	page 73 page 75 page 76 page 77 page 125
a	SET PULSE TRIGGER POLARITY READ LOWEST VALUES	D A	page 83 page 108
b	TRIGGER ON POSITIVE PULSE CLEAR LOWEST VALUES	D A	page 85 page 109
c	TRIGGER ON NEGATIVE PULSE READ AND CLEAR LOWEST VALUES	D A	page 86 page 110
d	READ PULSE COMPLETE BITS READ PEAK VALUES	D A	page 87 page 111
e	READ PULSE DURATION COUNTERS CLEAR PEAK VALUES	D A	page 88 page 112
f	READ AND CLEAR DURATION COUNTERS READ AND CLEAR PEAK VALUES	D A	page 89 page 113
g	CLEAR DURATION COUNTERS CALCULATE OFFSETS	D A	page 90 page 115
h	RETRIGGER TIME DELAY CALCULATE AND SET OFFSETS	D A	page 78 page 119
i	GENERATE N PULSES READ AVERAGE COMPLETE BITS	D A	page 79 page 97
j	READ MODULE CONFIGURATION	D, A	page 50
k	START ON PULSE SET TEMPERATURE PROBE TYPE	D A	page 80 page 44
l	START OFF PULSE READ TEMPERATURE INPUTS	D A	page 81 page 99

m	SET ENHANCED DIGITAL WATCHDOG
	SET ANALOG WATCHDOG USER-DEFINED VALUE
		D
		A
			page 41
			page 42

n	SET TIMER RESOLUTION	D	page 43
o	READ AVERAGE TEMPERATURE INPUTS	A	page 101
‘	DATE OF FIRMWARE	D	page 135

5: Setup Commands

Power-Up Clear — Driver Command 0

Digital, Analog — Optomux Command A

Purpose: Prevents the Optomux unit from returning a Power-Up Clear Expected error message in response to the first command after the unit is turned on.

Remarks: Only functions if it is the first command sent after power-up or reset; a power-up clear expected error is returned if any other command is sent first, and that command is NOT executed. After a power-up clear expected error is returned, this command does not need to be sent; the next command will be executed normally.

This command has NO effect on the Optomux unit's operation or setup—the Power-up Clear Expected error provides an indication to the host that there has been a power failure and that Optomux has been reset to power-up conditions (see page 34).

Driver

Parameters:

Command:	0

Example: Perform Power-up Clear on OMUX_Handle:

```
result = Send_Receive_Optomux(OMUX_Handle, 0, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: A

Example: Perform power-up clear at Optomux address 79 (hex):

```
>79AB1cr
```

Reset — Driver Command 1

Digital, Analog — Optomux Command B

Purpose: Resets the Optomux unit to power-up conditions.

Remarks: This command sets all of the operating characteristics of the addressed Optomux unit to power-up conditions. Power-up conditions for digital and analog are:

Digital	Analog
All outputs turned off All points then configured as inputs Protocol as set by jumper B10 (not on B3000) Watchdog timer disabled Turnaround delay = 0 Counters/duration timers cancelled Latches cleared Timer resolution = 10 ms.	0 scale written to all output points All points then configured as inputs Protocol as set by jumper B10 (not on B3000) Watchdog timer disabled Turnaround delay = 0 All offsets set to 0 All gain coefficients set to 1 All averaging cancelled All temperature probe types cancelled

NOTE: If you are using a B3000 brain, a Reset command affects all four virtual addresses within the brain; you cannot reset just one. After sending a Reset command to a B3000 brain, you must send a Power-Up Clear command. Otherwise, the brain responds with an error.

After using a Reset command, the Optomux application should wait about 100 ms before trying to communicate with a B1, E1, B2, or E2 brain board. For a B3000, it should wait about 800 ms.

Driver

Parameters:

Command:	1

Example: Reset Optomux unit to power-up conditions:

```
result = Send_Receive_Optomux(OMUX_Handle, 1, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: B

Example: Reset Optomux unit at address 22 (hex) to power-up conditions
`>22BA6cr`

Set Turnaround Delay

Digital, Analog

Purpose: Tells the Optomux unit to wait for a specified time before responding to commands sent from host. This command is helpful in some half-duplex radio modem applications. (It does not apply to wireless Ethernet modems.)

Remarks: Valid delays are:

0	No Delay
1	10 ms
2	100 ms
3	500 ms

If no delay is specified, delay = 0 is assumed. On power-up, delay = 0.

Driver

Parameters:

Command:	2
Info Array	First element is delay (see Valid delays, above)

Example: Set turnaround delay to 100 ms:

```
OMUX_Info(0) = 2


result = Send_Receive_Optomux(OMUX_Handle, 2, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `C[data]`

Example: Set unit 92 (hex) turnaround delay to 100 ms

```
>92C2E0cr
```

Set Digital Watchdog — Driver Command 3

Digital — Optomux Command D

Purpose: Instructs a digital Optomux unit to monitor activity on the communications link and to take a predetermined action if there is no activity within a specified time. *No activity* means no activity of any kind on the serial link, or no communication with this brain board on the Ethernet link.

Remarks: Valid time and actions are:

INFO or [data]	Time	Action
0	--	Watchdog disabled
1	10 seconds	Turn all outputs OFF
2	1 minute	Turn all outputs OFF
3	10 minutes	Turn all outputs OFF
4	--	Watchdog disabled
5	10 seconds	Turn output 0 on, all other outputs OFF
6	1 minute	Turn output 0 on, all other outputs OFF
7	10 minutes	Turn output 0 on, all other outputs OFF

If no data is specified, 0 (watchdog disabled) is assumed. Watchdog is disabled on power-up. The Optomux unit will respond to the first command after a watchdog timeout with an error -7 for the driver or N06cr for the protocol, and the command will NOT be executed (except for a PUC). This error code is sent as a warning to let the host know a watchdog timeout occurred.

Driver

Parameters:

Command:	3
Info Array	First element is time and action (see above)

Example: Deactivate all outputs if there is no activity for one minute:

```
OMUX_Info(0) = 2


result = Send_Receive_Optomux(OMUX_Handle, 3, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `D[data]`

Example: Instruct Optomux unit at address 92 (hex) to deactivate all outputs if there is no activity for one minute.

```
>92D2E1cr
```

Set Analog Watchdog — Driver Command 45

Analog — Optomux Command D

Purpose: Instructs an analog Optomux unit to monitor activity on the communications link and to take a predetermined action if there is no activity within a specified time. No *activity* means no activity of any kind on the serial link, or no communication with this brain board on the Ethernet link.

Remarks: Valid time and actions are:

INFO or [data]	Time	Action
0	--	Watchdog disabled
1	10 seconds	Write zero-scale
2	1 minute	Write zero-scale
3	10 minutes	Write zero-scale
4	--	Watchdog disabled
5	10 seconds	Write full-scale
6	1 minute	Write full-scale
7	10 minutes	Write full-scale
8–19	--	Generates a limit error; command not executed.
20–65,535	Value x 10 ms	Sets timeout value. Output data is determined by Driver command 78 (Optomux command m). See page 42.

If no data is specified, 0 (watchdog disabled) is assumed. For values 0 - 7, times and actions are fixed and are not affected by the settings of command 78 (m).

The Optomux unit will respond to the first command after a watchdog timeout with an error -7 for the driver or N06cr for the protocol, and the command will NOT be executed (unless it is a PUC). The error message is a warning to let the host know a watchdog timeout occurred.

Driver

Parameters:

Command:	45
Positions Array	Points to be affected by the command.
Info Array	First element is time and action (see above). If using values 20–65,535, must also use command 78.

Example: Output zero-scale at points 0, 1, and 2 if there is no activity for one minute:

```
OMUX_Pos(0) = 0
OMUX_Pos(1) = 1
OMUX_Pos(2) = 2
OMUX_Pos(3) = -1
OMUX_Info(0) = 2
```

```
result = Send_Receive_Optomux(OMUX_Handle, 45, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `D[positions][data]`

Example: Instruct Optomux unit at address 1A (hex) to output zero-scale at points 0, 1, 2, and 3 if there is no activity for one minute.

```
>1AD000F2BEcr
```

Set Protocol — Driver Command 4

Digital, Analog — Optomux Command E

Purpose: Instructs Optomux unit to use either 2-pass or 4-pass protocol.

NOTE: Does not apply to B3000 Optomux units.

Remarks: Valid protocols are:

0	2-pass protocol
1	4-pass protocol

If no protocol is specified, protocol = 0 (2-pass) is assumed. On power-up, protocol is set according to jumper B10.

4-pass is used for diagnostics only.

Driver

Parameters:

Command:	4
Info Array	First element sets protocol (see above).

Example: Use 4-pass protocol:

```
OMUX_Info(0) = 1

result = Send_Receive_Optomux(OMUX_Handle, 4, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `E[data]`

Example: Instruct Optomux unit at address 9B (hex) to use 4-pass protocol.

```
>9BE1F1cr
```

Identify Optomux Type — Driver Command 5

Digital, Analog — **Optomux Command F**

Purpose: Instructs the Optomux unit to identify itself as either a digital or analog I/O unit.

Remarks: The unit responds with a return message containing an identification code:

Driver	Optomux	Identification
0	00	Digital Optomux
1	01	Analog Optomux

Driver

Parameters:

Command:	5
Info Array	(Response) First element of info array parameter in response message is ID code (see above).

Example: Request for Optomux unit to identify itself:

```
result = Send_Receive_Optomux(OMUX_Handle, 5, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
if (OMUX_Info(0) = 1) then 'it's analog
  else 'it's digital
endif
```

Optomux Protocol

Format: F

Example: Request for unit at address D0 (hex) to identify itself.

```
>D0FBAcr
```

Message returned by unit identifying itself as an analog Optomux I/O unit

```
A0161cr
```

Set Enhanced Digital Watchdog — Driver Command 71

Digital — Optomux Command m

Purpose: Instructs a digital Optomux unit to monitor activity on the communications link and to take a specified action if there is no activity within a specified time.

Remarks: After this command is issued, if a character is not received within the time specified by Info(0) or [data], the unit will turn on or off all outputs specified. All time delay outputs are cancelled. Inputs are not affected.

The delay time is set to Info(0) or [data] * 10 milliseconds. Delays of less than 200 milliseconds (except 0) result in a limit error and the command is not executed. A delay time of zero disables the digital watchdog function. If no delay time is sent, 0 is assumed.

The Optomux unit will respond to the first command after a watchdog timeout with a -7 (driver) or N06cr (Optomux protocol) error, and the command will NOT be executed (unless it is a PUC). This error code is sent as a warning to let the host know a watchdog timeout occurred.

Driver

Parameters:

Command:	71
Positions Array	Points to be turned on if the watchdog times out. Points not included in the array will be turned off
Info Array	Delay time (value * 10 milliseconds)

Example: Turn on output points 11, 9, 7, and 2 and turn off all other output points if there is no activity for 5 seconds. (Assumes that points were already configured as outputs.)

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 7
OMUX_Pos(2) = 9
OMUX_Pos(3) = 11
OMUX_Pos(4) = -1
OMUX_Info(0) = 500
```

```
result = Send_Receive_Optomux(OMUX_Handle, 71, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `m[positions][data]`

Outputs corresponding to 1 bits in [positions] are turned on; 0 bits are turned off.

Example: Turn on output points 11, 9, 7, and 2 and turn off all other output points on Optomux unit at address EC (hex), if there is no activity for 5 seconds. (This example assumes that the points were already configured as outputs.)

```
>ECm0A841F47Dcr
```

Set Analog Watchdog Timeout — Driver Command 78

Analog — Optomux Command m

Purpose: Sets the data that an analog Optomux unit writes to the specified output points when a watchdog timeout occurs.

Remarks: Only the points indicated are affected by the command. The delay time is set by command 45 (D), Set Analog Watchdog on page 37, which must be called before this command will work. When this command is being used, valid values for command 45 (D) are 20 to 65,535.

NOTE: The Optomux unit will respond to the first command after a watchdog timeout with a -7 (driver) or N06cr (Optomux protocol) error, and the command will not be executed (unless it is a PUC). This error code is sent as a warning to let the host know a watchdog timeout occurred.

Driver

Parameters:

Command:	78
Positions Array	Output points that are to have timeout data set for them. Other output points are not affected.
Info Array	Values to be written to the points. Values are passed in the Info array elements that correspond to the point: first element is value for point 0, second element is value for point 1, and so on.

Example: Sets output points 0, 5, and 7 to half-scale when a watchdog timeout occurs.

```
OMUX_Pos(0) = 0
OMUX_Pos(1) = 5
OMUX_Pos(2) = 7
OMUX_Pos(3) = -1
OMUX_Info(0) = 2048
OMUX_Info(5) = 2048
OMUX_Info(7) = 2048

result = Send_Receive_Optomux(OMUX_Handle, 78, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `m[positions][data]`

NOTE: [data] is set only for output points that correspond to a 1 in the [positions] bitmask. Three characters of data are expected for each output point specified. If enough data is not received (the number of data values must equal the number of bits set to 1 in the bitmask), Optomux will respond with a N05cr data field error message.

Example: On the Optomux unit at address AA (hex), output the value A20 at point 7 and the value 555 at point 0 when a serial watchdog timeout occurs.

```
>AAm0081A20555FAcr
```

Set Timer Resolution — Driver Command 75

Digital — Optomux Command n

Purpose: Sets a global timer value for all timing functions on the Optomux digital brain.

Remarks: If the value is 0, the timer resolution is 2.56 seconds. This command is a global command and affects the timing resolution for the following commands::

SET TIME DELAY INITIATE SQUARE WAVE HIGH RESOLUTION SQUARE WAVE RETRIGGER TIME DELAY GENERATE N PULSES	START ON PULSE START OFF PULSE READ PULSE COMPLETE BITS READ PULSE DURATION COUNTERS READ AND CLEAR DURATION COUNTERS

Driver

Parameters:

Command:	75
Info Array	First element specifies timer resolution (value * 10 milliseconds)

Example: Set the timer resolution to 100 ms.

```
OMUX_Info(0) = 10
```

```
result = Send_Receive_Optomux(OMUX_Handle, 75, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `n[data]`

Example: Set the timer resolution of the digital Optomux at address FE (hex) to 100 ms

```
>FEn0A6Acr
```

Set Temperature Probe Type — Driver Command 76

Analog — Optomux Command k

Purpose: Sets the probe type for points using temperature input modules (thermocouples, ICTDs, and RTDs), so that the READ TEMPERATURE INPUTS command can be used to read the temperature directly.

Remarks: Valid probe types are:

0	no temperature probe
1	ICTD probe
2	10 ohm RTD probe
3	100 ohm RTD probe
4	Type J thermocouple
5	Type K thermocouple
6	Type R thermocouple
7	Type S thermocouple
8	Type T thermocouple
9	Type E thermocouple

NOTE: Valid probe type may be 0-4 ASCII-hex characters. For example, ICTD probe type may be 1, 01, 001, or 0001.

This command can be sent multiple times with different values to set a variety of probes on one rack.

Driver

Parameters:

Command:	76
Positions Array	Points to be set to the desired probe type.
Info Array	First element specifies probe type.

Example: Set the temperature probe type for points 14, 6, and 5 to a type S thermocouple.

```
OMUX_Pos(0) = 5
OMUX_Pos(1) = 6
OMUX_Pos(2) = 14
OMUX_Pos(3) = -1
OMUX_Info(0) = 7

result = Send_Receive_Optomux(OMUX_Handle, 76, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `k[positions][data]`

Example: For Optomux address DC (hex), set the temperature probe type for points 14, 6, and 5 to a type S thermocouple:

```
>DCk40607F3cr
```

6: I/O Configuration Commands

Configure Positions — Driver Command 6

Digital, Analog — Optomux Command G

Purpose: Sets which points are inputs or outputs.

Remarks: Configures the points to function as either inputs or outputs. If the configuration for any point is changed by this command, then any time delay, latch, etc. is cleared. On power-up, all points are configured as inputs.

Driver

Parameters:

Command:	6
Positions Array:	Points to function as outputs. Points not specified here are configured as inputs.

Example: Configures points 2, 3, and 0 to function as outputs; all other points will be configured as inputs:

```
OMUX_Pos(0) = 0
OMUX_Pos(1) = 2
OMUX_Pos(2) = 3
OMUX_Pos(3) = -1

result = Send_Receive_Optomux(OMUX_Handle, 6, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `G[positions]`

Points corresponding to 1 bits are configured as outputs. Points corresponding to 0 bits are configured as inputs.

Example: Instructs Optomux unit at address 00 (hex) to configure points 0, 1, 4, 5, 8, and 12 to function as outputs. All other points will be configured to function as inputs:

```
>00G11336Fcr
```

Instructs the Optomux I/O unit at address 45 (hex) to configure points 0, 2, and 3 to function as inputs and point 1 to function as an output. All other points are unchanged:

```
>45G2E2cr
```

Configure as Inputs — Driver Command 7

Digital, Analog — Optomux Command H

Purpose: Configures the points to function as inputs.

Remarks: If the configuration for any point is changed by this command, then any time delay, latch, etc. is cleared. On power up, all points are configured as inputs.

Driver

Parameters:

Command:	7
Positions Array:	Points to function as inputs. Points not specified are left unchanged.

Example: Configure points 4, 11, and 15 to function as inputs:

```
OMUX_Pos(0) = 4
OMUX_Pos(1) = 11
OMUX_Pos(2) = 15
OMUX_Pos(3) = -1

result = Send_Receive_Optomux(OMUX_Handle, 7, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `H[positions]`

Example: Configure points 0 and 2 on Optomux 4B as inputs; all other points are left unchanged:
`>4BH5F3cr`

Configure as Outputs — Driver Command 8

Digital, Analog — Optomux Command I

Purpose: Configures the points to function as outputs.

Remarks: If the configuration for any point is changed by this command, then any time delay, latch, etc. is cleared. On power up, all points are configured as inputs.

Driver

Parameters:

Command:	8
Positions Array:	Points to function as outputs. Points not specified are left unchanged.

Example: Configure points 0, 2, 3, and 4 to function as outputs:

```
OMUX_Pos(0) = 0
OMUX_Pos(1) = 2
OMUX_Pos(2) = 3
OMUX_Pos(3) = 4
OMUX_Pos(4) = -1

result = Send_Receive_Optomux(OMUX_Handle, 8, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `I[positions]`

Example: Configure points 0 and 2 to function as outputs on Optomux unit 4B:
`>4BI5F4cr`

Read Configuration — Driver Command 70

Digital, Analog — Optomux Command j

Purpose: Returns the current input/output configuration for all 16 points.

Driver

Parameters:

Command:	70
Info Array	(Response) Non-zero = output; 0 = input. Values in the Info array elements indicate configuration for the corresponding point: first element is configuration for point 0, second element is for point 1, and so on.

Example: Request current configuration of all points on the unit:

```
result = Send_Receive_Optomux(OMUX_Handle, 70, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
for OMUX_Info = 0 to 15
if (OMUX_Info(0) = 1) then 'it’s an output
  else 'it’s an input
endif
```

Optomux Protocol

Format: `j`

Return message contains four ASCII-hex digits; each digital represents four points, just as in a [positions] field. Points corresponding to 1 bits are configured as outputs; points corresponding to 0 bits are inputs.

Example: Request the current configuration of all the points on Optomux unit FF hex (255 decimal). Notice how a `??` is used for the checksum. The `??` is a wildcard character for use when debugging.

```
>FFj??cr
```

Response from Optomux indicating that all points are configured as inputs:

```
A0000C0cr
```

7: Digital Read/Write Commands

Write Digital Outputs — Driver Command 9

Digital — Optomux Command J

Purpose: Turns output points on and off.

Remarks: Time delays, if set, are implemented when this command is executed. Points that have been configured to function as inputs are not affected by this command.

Driver

Parameters:

Command:	9
Positions Array:	Points to turn on. Points not included are turned off.

Example: Turn on points 2 and 3. Turn off all other output points:

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 3
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 9, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `J[positions]`

Points corresponding to 1 bits are turned on; points corresponding to 0 bits are turned off. If the [positions] field is left out, Optomux assumes a bitmask of FFFF.

Example: Turn ON all outputs at Optomux address 00 (hex):
`>00JFFFFC2cr`

Turn OFF points 0, 1, 2, 3 at Optomux address 45 (hex); points 4-15 are not affected.
`>45J0E3cr`

Write Binary Outputs — Driver Command 65

Digital

Purpose: Turns output points on and off.

Remarks: Time delays, if set, are implemented upon execution of this command. Points that have been configured to function as inputs are not affected by this command.

Driver

Parameters:

Command:	65
Info Array:	First element contains a bitmask indicating points to turn on or off. 1 = on; 0 = off. Bit 15, the most significant bit, corresponds to point 15; bit 0 corresponds to point 0.

Example: Turn on points 2 and 3. Turn off all other output points:

```
OMUX_Info(0) = 12 '1100 in binary

result = Send_Receive_Optomux(OMUX_Handle, 65, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

See Write Digital Outputs (Optomux command J).

Activate Digital Outputs — Driver Command 10

Digital — Optomux Command K

Purpose: Turns on the specified outputs.

Remarks: Time delays, if set, are implemented upon execution of this command. Points configured as inputs are not affected by this command.

Driver

Parameters:

Command:	10
Positions Array:	Points to turn on. All other points are unchanged.

Example: Turn on points 2 and 3, leaving all others unchanged:

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 3
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 10, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `K[positions]`

If the [positions] field is left out, Optomux assumes a bitmask of FFFF.

Example: Instruct Optomux at address 99 (hex) to activate outputs at points 2, 3, 6, 7, 8, 10, 12, and 14. All other points are not affected.

```
>99K55CCADcr
```

Deactivate Digital Outputs — Driver Command 11

Digital — Optomux Command L

Purpose: Turns off the specified outputs.

Remarks: Points configured as inputs are not affected by this command.

Driver

Parameters:

Command:	11
Positions Array:	Points to turn off. All other points are unchanged.

Example: Turn off points 2 and 3, leaving all others unchanged:

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 3
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 11, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `L[positions]`

If the [positions] field is left out, Optomux assumes a bitmask of FFFF.

Example: On the Optomux unit at address BC (hex), turn off the outputs at points 1 and 2:

```
>BCL607cr
```

At address 9A (hex), turn OFF all outputs (equivalent to a Write Digital Outputs command with [positions] = 0000):

```
>9ALFFFFDEcr
```

Read On/Off Status — Driver Command 12

Digital, Analog — Optomux Command M

Purpose: Returns the current on/off status of all 16 points.

Driver

Parameters:

Command:	12
Info Array:	(Response) Non-zero = on; 0 = off. Values in the Info array elements indicate status for the corresponding point: first element is status for point 0, second element is for point 1, and so on.

Example: Read on/off status of all 16 points:

```
result = Send_Receive_Optomux(OMUX_Handle, 12, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
'Loop through 16 points
For Index = 0 to 15
  If (OMUX_Info(Index) = 1) then
    'it's on
  Else
    'it's off
  End If
Next Index
```

Optomux Protocol

Format: `M[positions]`

Example: Request on/off status of all points on Optomux unit at address FF (hex):

`>FFMD9cr`

Response from Optomux indicating that all points are in the OFF state:

`A0000C0cr`

Response from Optomux indicating that points 1, 6, 7, 9, and 11 are ON, and that all others are OFF:

`A0AC2E6cr`

Read Binary On/Off Status — Driver Command 64

Digital

Purpose: Returns the on/off status of all 16 points in the form of a 16-bit binary number.

Driver

Parameters:

Command:	64
Info Array:	(Response) First element contains a 16-bit binary number indicating the on/off status of all points. 1 = on; 0 = off. Bit 15, the most significant bit, corresponds to point 15; bit 0 corresponds to point 0.

Example: Find out the on/off status of all 16 points:

```
result = Send_Receive_Optomux(OMUX_Handle, 64, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Status_Data = OMUX_Info(0)
```

Optomux Protocol

See Read On/Off Status (Optomux command J).

8: Digital Latch Commands

Set Latch Edges — Driver Command 13

Digital — Optomux Command N

Purpose: Sets points configured as inputs to latch on either ON-to-OFF or OFF-to-ON transitions.

Remarks: Points configured as outputs are not affected by this command. On power up, all points are set to latch on OFF-to-ON transitions.

Driver

Parameters:

Command:	13
Positions Array:	Points to latch on ON-to-OFF transitions. Input points not included are set to latch on OFF-to-ON transitions.

Example: Set points 2 and 14 to latch on ON-to-OFF transitions:

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 14
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 13, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `N[positions]`

Points corresponding to 1 bits are set to latch on ON-to-OFF transitions; points corresponding to 0 bits are set to latch on OFF-to-ON transitions.

Example: This command instructs the Optomux unit at address 55 (hex) to set point 0 to latch on OFF-to-ON transitions, all other points are set to latch on ON-to-OFF transitions.

```
>55NFFFECFcr
```

Set Off-to-On Latches — Driver Command 14

Digital — Optomux Command O

Purpose: Sets input points to latch on OFF-to-ON transitions.

Remarks: Points configured as outputs are not affected by this command. On power up, all points are set to latch on OFF-to-ON transitions.

Note that this command in the protocol is an uppercase O, not a zero.

Driver

Parameters:

Command:	14
Positions Array:	Input points to latch on OFF-to-ON transitions. All other points remain unchanged.

Example: Set points 14, 15, and 16 to latch on OFF-to-ON transitions:

```
OMUX_Pos(0) = 14
OMUX_Pos(1) = 15
OMUX_Pos(2) = 16
OMUX_Pos(3) = -1

result = Send_Receive_Optomux(OMUX_Handle, 14, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `O[positions]`

Points corresponding to 1 bits are set to latch on OFF-to-ON transitions; points corresponding to 0 bits are left unchanged.

Example: This command configures points 14 and 15 at Optomux address FF (hex) to latch on OFF-to-ON transitions; all other points are not affected.

```
>FFOC000AEcr
```

Purpose: Sets input points to latch on ON-to-OFF transitions.

Remarks: Points configured as outputs are not affected by this command. On power up, all points are set to latch on OFF-to-ON transitions.

Driver

Parameters:

Command:	15
Positions Array:	Input points to latch on ON-to-OFF transitions. All other points remain unchanged.

Example: Set points 14, 15, and 16 to latch on ON-to-OFF transitions:

```
OMUX_Pos(0) = 14
OMUX_Pos(1) = 15
OMUX_Pos(2) = 16
OMUX_Pos(3) = -1

result = Send_Receive_Optomux(OMUX_Handle, 15, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `P[positions]`

Points corresponding to 1 bits are set to latch on ON-to-OFF transitions; points corresponding to 0 bits are left unchanged.

Example: This command instructs Optomux at address FF (hex) to set points 14 and 15 to latch on ON-to-OFF transitions; all other points are not affected.

```
>FFPC000AFcr
```

Read Latches — Driver Command 16

Digital — Optomux Command Q

Purpose: Returns data indicating which of the inputs have latched.

Remarks: This command does not clear the latches. Subsequent Read Latches commands will return the same results.

Driver

Parameters:

Command:	16
Info Array:	(Response) Non-zero = latched; 0 - unlatched. Values in the Info array elements indicate latch status for the corresponding point: first element is latch status for point 0, second element is for point 1, and so on.

Example: Display latch states for Optomux unit:

```
result = Send_Receive_Optomux(OMUX_Handle, 16, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
'Loop through 16 points
For Index = 0 to 15
  If (OMUX_Info(Index) = 1) then
    'latch is set
  Else
    'latch is not set
  End If
Next Index
```

Optomux Protocol

Format: `Q`

Response is a bitmask in hex indicating latch status: 1 = latched; 0 = unlatched. Ignore bits corresponding to points configured as outputs.

Example: Read latches on Optomux at address 77 (hex):

```
>77QBFcr
```

Message returned from Optomux unit indicating that inputs at points 15, 11, 7, and 3 have latched.

```
A8888E0cr
```

Read and Clear Latches — Driver Command 17

Digital — Optomux Command R

Purpose: Returns data indicating which inputs have latched and then resets specified latches.

Remarks: This command returns the latch status for all points. It clears latches only for the specified points. All other latches remain unchanged.

Driver

Parameters:

Command:	17
Info Array:	(Response) Non-zero = latched; 0 - unlatched. Values in the Info array elements indicate latch status for the corresponding point: first element is latch status for point 0, second element is for point 1, and so on.

Example: Display latch states for Optomux unit and clear latches for points 4 and 5:

```
OMUX_Pos(0) = 4
OMUX_Pos(1) = 5

result = Send_Receive_Optomux(OMUX_Handle, 17, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
'Loop through 16 points
For Index = 0 to 15
  If (OMUX_Info(Index) = 1) then
    'latch is set
  Else
    'latch is not set
  End If
Next Index
```

Optomux Protocol

Format: `R[positions]`

Response is a bitmask in hex indicating latch status: 1 = latched; 0 = unlatched. Ignore bits corresponding to points configured as outputs.

Example: Command message to read all latches on Optomux at address 77 (hex), and to clear latches for points 4, 5, and 7.

```
>77RB032cr
```

Response from Optomux indicating that points 1, 3, 5, 9, 10, 12, 13, and 14 have latched.

```
A762AE0cr
```

Clear Latches — Driver Command 18

Digital — Optomux Command S

Purpose: Clears latches for specified input points to the unlatched state.

Remarks: Latches for points that are not specified will remain unchanged.

Driver

Parameters:

Command:	18
Positions Array:	Input points whose latches should be cleared. All other points remain unchanged.

Example: Clear latches for points 2 and 3:

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 3
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 18, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `S[positions]`

Latches for points corresponding to 1 bits are reset to unlatched; latches for points corresponding to 0 bits are left unchanged.

Example: This command instructs Optomux at address 77 (hex) to clear all latches.
`>77SC1cr`

Instructs Optomux at address 77 (hex) to clear the latch for point 0.
`>77S1F2cr`

Read Binary Latches

Driver Command 66

Digital

Purpose: Returns data indicating which of the inputs have latched.

Remarks: This command does not clear the latches. Subsequent Read Latches commands will return the same results.

Driver

Parameters:

Command:	66
Info Array:	(Response) First element contains a 16-bit binary number indicating latch state. 1 = latched; 0 - unlatched. Bit 15 (MSB) of the returned data corresponds to point 15; bit 0 corresponds to point 0.

Example: Display latch states for Optomux unit:

```
result = Send_Receive_Optomux(OMUX_Handle, 6, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Latch_Data = OMUX_Info(0)
```

Optomux Protocol

See Read Latches (Optomux command Q).

Read and Clear Binary Latches — Driver Command 67

Digital

Purpose: Returns data indicating which inputs have latched and then resets specified latches.

Remarks: This command returns the latch status for all points. It clears latches only for the specified points. All other latches remain unchanged.

Driver

Parameters:

Command:	67
Info Array:	(Command) In Element 0, a 16-bit number indicating the latch points to be cleared. (Response) First element contains a 16-bit binary number indicating latch state. 1 = latched; 0 - unlatched. Bit 15 (MSB) of the returned data corresponds to point 15; bit 0 corresponds to point 0.

Example: Display latch states for Optomux unit and clear latches for points 2 and 3:

```
OMUX_Info(0) = 12

result = Send_Receive_Optomux(OMUX_Handle, 67, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Latch_Data = OMUX_Info(0)
```

Optomux Protocol

See Read and Clear Latches (Optomux command R).

9: Digital Counting Commands

Start and Stop Counters — Driver Command 19

Digital **Optomux Command T**

Purpose: Starts and stops the counting of OFF-to-ON transitions at specified input points.

Remarks: This command has no effect on the stored count. Counting can start or resume at any time. The maximum count is 65,535; after that, the count resets to 0. Frequencies up to 400 Hz (50% duty cycle, minimum pulse width of 1.25 milliseconds) can be counted.

NOTE: Using the Generate N Pulses command 72 (i) will degrade the maximum counting frequency to about 350 Hz.

Driver

Parameters:

Command:	19
Positions Array:	Input points at which Optomux will start counting OFF-to-ON transitions. Counting is stopped at all other points.

Example: Start counting on points 7 and 8; stop counting at all other points:

```
OMUX_Pos(0) = 7
OMUX_Pos(1) = 8
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 19, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `T[positions]`

Counting starts at points corresponding to 1 bits; counting stops at points corresponding to 0 bits.

Example: Start counting at points 0, 1, 2, and 3 on Optomux unit 73 (hex); stop counting at points 4, 5, 6, and 7. All other points are left unchanged.

```
>73T0F34cr
```

Start Counters — Driver Command 20

Digital — Optomux Command U

Purpose: Starts or resumes counting of OFF-to-ON transitions at specified input points.

Remarks: This command has no effect on the stored count. Counting can start or resume at any time. The maximum count is 65,535; after that, the count resets to 0. Frequencies up to 400 Hz (50% duty cycle, minimum pulse width of 1.25 milliseconds) can be counted.

NOTE: Using the Generate N Pulses command 72 (i) will degrade the maximum counting frequency to about 350 Hz.

Driver

Parameters:

Command:	20
Positions Array:	Input points at which Optomux will start counting OFF-to-ON transitions. Other points are not affected.

Example: Start counting on points 7 and 8:

```
OMUX_Pos(0) = 7
OMUX_Pos(1) = 8
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 20, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `U[positions]`

Counting starts at points corresponding to 1 bits; other points are not affected.

Example: This command starts counting at points 4 and 5 on Optomux address EE (hex).

```
>EEU3042cr
```

Stop Counters

Driver Command 21

Digital

Optomux Command V

Purpose: Stops counting of OFF-to-ON transitions at specified input points.

Remarks: This command has no effect on the stored count. Counting can resume at any time.

Driver

Parameters:

Command:	21
Positions Array:	Input points at which Optomux will stop counting OFF-to-ON transitions. Other points are not affected.

Example: Stop counting on points 7 and 8:

```
OMUX_Pos(0) = 7
OMUX_Pos(1) = 8
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 21, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `V[positions]`

Counting stops at points corresponding to 1 bits; other points are not affected.

Example: This command message stops counting at point 11 on Optomux address ED (hex).

```
>EDV80077cr
```

Read Counters — Driver Command 22

Digital — Optomux Command W

Purpose: Returns the counter values for the specified input points.

Remarks: This command has no effect on the current values of the counters (stored count).

Driver

Parameters:

Command:	22
Positions Array:	Points for which counter values will be returned.
Info Array:	(Response) Values in the Info array elements contain counter values for the corresponding point: first element is counter value for point 0, second element is for point 1, and so on.

Example: Request counter values for points 2 and 5:

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 5
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 22, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_2_Count = OMUX_Info(2)
Pos_5_Count = OMUX_Info(5)
```

Optomux Protocol

Format: `W[positions]`

Counter values are returned as four ASCII-hex digits representing a 16-bit value (0–65,535 decimal). Values are returned in sequence from highest to lowest point. If an output is read, `????` is returned for that point. If the [positions] field is omitted, a field of FFFF is assumed (all counters are read).

Example: Read counters at Optomux address 23 (hex), points 1 and 2.

```
>23W6F2cr
```

Message returned by Optomux. Counter value is B000 for point 2 (45,056 decimal) and 0008 (8 decimal) for point 1.

```
AB00000089Acr
```

Read counters 0, 2, 4, 6, 8, and 10 at address 23 (hex).

```
>23W5555Bcr
```

Message returned:

```
A123405671111????ABCD000127cr
```

The return message can be interpreted as follows:

Point	Hex data	Count Value
0	`0001`	1
2	`ABCD`	43,981
4	`????`	Point 4 is an output.
6	`1111`	4,369
8	`0567`	1,383
10	`1234`	4,660

Read and Clear Counters — Driver Command 23

Digital — Optomux Command X

Purpose: Returns the counter values for the specified input points and then sets the counters for those points to 0.

Driver

Parameters

Command:	23
Positions Array:	Input points for which counter values will be returned and counters cleared.
Info Array:	(Response) Values in the Info array elements contain counter values for the corresponding point: first element is counter value for point 0, second element is for point 1, and so on.

Example: Request counter values and clear counters for points 2 and 5:

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 5
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 23, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_2_Count = OMUX_Info(2)
Pos_5_Count = OMUX_Info(5)
```

Optomux Protocol

Format: `X[positions]`

Counter values are returned as four ASCII-hex digits representing a 16-bit value (0–65,535 decimal). Values are returned in sequence from highest to lowest point. If an output is read, `????` is returned for that point. If the [positions] field is omitted, FFFF is assumed (all counters are read and cleared).

Example: Return a counter value for point 11 at address A8 (hex), then clear the counter to 0.

```
>A8X80069cr
```

Response shows the counter value of point 11 to be F00 hex or 3,840 decimal:

```
>A0F00D6cr
```

Clear Counters — Driver Command 24

Digital — Optomux Command Y

Purpose: Resets counters for specified input points to zero. Counters for unspecified points are left unchanged.

Driver

Parameters:

Command:	24
Positions Array:	Input points at which Optomux set counters to 0. Other points are not affected.

Example: Clear counters at points 0 through 7:

```
For Index = 0 to 7
  OMUX_Pos(Index) = Index
Next Index

result = Send_Receive_Optomux(OMUX_Handle, 24, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `Y[positions]`

If [positions] field is omitted, a field of FFFF is assumed and all counters are cleared.

Example: Clear the counters for point 11 at Optomux address A8 (hex):

```
>A8Y8006Acr
```

10: Digital Time Delay and Pulse Commands

Set Time Delay — Driver Command 25

Digital — Optomux Command Z

Purpose: Sets specified output points to be used in a pulsed or delayed mode.

Remarks: Valid delay descriptions are:

Driver	Optomux	Description
0	H	On Pulse - When instructed to go from OFF to ON, turn ON for the desired time, and then turn OFF.
1	I	On Delay - When instructed to go from OFF to ON, stay OFF for the desired time, and then turn ON.
2	J	Off Pulse - When instructed to go from ON to OFF, turn OFF for the desired time, and then turn ON.
3	K	Off Delay - When instructed to go from ON to OFF, stay ON for the desired time, and then turn OFF.

Before using this command, use the Set Timer Resolution command 75 (n) on page 43. Current timer resolution is multiplied by the delay length (set in the Info Array or [data] fields) to equal the desired time:

```
Desired time = delay length × timer resolution × 10 ms
```

Valid delay lengths are 0 through 65,535. A 0 value is equal to a delay length of 65,535 (FFFF).

Driver

Parameters:

Command:	25
Positions Array:	Output points to be used in pulsed or delayed mode. Other points are not affected.
Modifiers Array:	First element contains delay description (see above).
Info Array:	First element contains delay length (see above).

Example: Set points 2 and 3 to turn on for 1.2 seconds and then turn off when Optomux is instructed to turn them on (assumed timer resolution of 10 ms):

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 3
OMUX_Pos(2) = -1
OMUX_Mod(0) = 0
OMUX_Info(0) = 120

result = Send_Receive_Optomux(OMUX_Handle, 25, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `Z[positions][modifier][data]`

The [modifier] field contains the delay description (see above).

Also see the following related commands:

"Turn Off Time Delay/Square Wave" on page 76

"Initiate Square Wave" on page 75

"High Resolution Square Wave" on page 77

Example: NOTE: All examples assume a timer resolution value of 1 (10 ms).

Set points 2 and 3 at Optomux address 89 (hex) for delay type I with a pulse length of 1.02 seconds:

```
>89ZCI66C3cr
```

At Optomux address 10 (hex), set points 12, 8, 4, and 0 for a type K delay of 10 seconds:

```
>10Z1111K3E87Acr
```

Initiate Square Wave — Driver Command 26

Digital — Optomux Command Z

Purpose: Starts a continuous square wave at specified output points.

Remarks: The square wave continues until it is turned off using command 27 (Z) or modified with Set Time Delay command 25 (Z) on page 73 or High Resolution Square Wave command 68 (Z) on page 77. Write Digital Outputs, Activate Digital Outputs, and Deactivate Digital Outputs have no effect while the square wave continues.

Before using this command, use the Set Timer Resolution command 75 (n) on page 43. Current timer resolution and values in the Info Array or [data] fields are used to calculate on and off times of the square wave, as described below.

Driver

Parameters:

Command:	26
Positions Array:	Points to output square waves. Other points remain unchanged.
Info Array:	First two elements determine on and off times of the square wave as follows: On time = timer resolution x 256 x first element value Off time = timer resolution x 256 x second element value. Values can be between 0 and 255 (0 = 256). Maximum for on and off times is 2.56 seconds x 256 x 256 (2796.20 minutes or 46.6 hours).

Example: Output square wave at point 12 with an on time of 2.56 seconds and an off time of 5.12 seconds (assumed timer resolution of 10 ms):

```
OMUX_Pos(0) = 12
OMUX_Pos(1) = -1
OMUX_Info(0) = 1
OMUX_Info(1) = 2

result = Send_Receive_Optomux(OMUX_Handle, 26, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `Z[positions][modifier][data]`

The modifier is L for a continuous square wave. Square wave ON and OFF times are calculated as follows:

```
ON time = 2.56 seconds x timer resolution x first two digits in
[data] (leftmost characters)
OFF time = 2.56 x timer resolution x last two digits in [data]
```

Example: At address 11 (hex), set points 6, 5, 2, and 1 for a square wave — ON 2.56 seconds, OFF 3.75 minutes.

```
>11Z66L015842cr
```

Turn Off Time Delay/Square Wave — Driver Command 27

Digital — **Optomux Command Z**

Purpose: Turns off existing time delay or square wave at specified output points.

Driver

Parameters:

Command:	27
Positions Array:	Output points at which square wave or time delay will be turned off. Other points remain unchanged.

Example: Turn off any existing time delay or square wave at points 7 and 8:

```
OMUX_Pos(0) = 7
OMUX_Pos(1) = 8
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 27, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `Z[positions][modifier]`

Modifier for this command is G.

Example: Turn off existing delay at Optomux address 24 (hex), point 12:

```
>24Z1000GC8cr
```

High Resolution Square Wave — Driver Command 68

Digital — Optomux Command Z

Purpose: Starts a continuous square wave at specified output points.

Remarks: The square wave continues until it is turned off using command 27 (Z) or modified with Set Time Delay command 25 (Z) on page 73 or Initiate Square Wave command 26 (Z) on page 75. Write Digital Outputs, Activate Digital Outputs, and Deactivate Digital Outputs have no effect while the square wave continues.

This command operates the same as the standard square wave command except that it uses the current timer resolution instead of a resolution of 2.56 seconds. Before using this command, use the Set Timer Resolution command 75 (n) on page 43. Set Timer Resolution is very useful for altering the period of the square wave, especially for flashing lights at fast rates.

Driver

Parameters:

Command:	68
Positions Array:	Points to output square waves. Other points remain unchanged.
Info Array:	First two elements determine on and off times of the square wave as follows: On time = timer resolution x first element value Off time = timer resolution x second element value. Maximum for on and off times is 256 x 2.56 seconds (10.92 minutes).

Example: Output square wave at point 2 with an on time of 0.05 seconds and an off time of 0.12 seconds (assumed timer resolution of 10 ms):

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = -1
OMUX_Info(0) = 5
OMUX_Info(1) = 12

result = Send_Receive_Optomux(OMUX_Handle, 68, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `Z[positions][modifier][data]`

The modifier is M for a high resolution continuous square wave. Square wave ON and OFF times are calculated as follows:

```
ON time = timer resolution x first two digits in [data]
(leftmost characters)
OFF time = timer resolution x last two digits in [data]
```

Example: At address 1E (hex), set points 1 and 6 for a high resolution square wave; ON for 40 ms, and OFF for 310 ms.

```
>1EZ42M041F5Ecr
```

Retrigger Time Delay — Driver Command 69

Digital — Optomux Command h

Purpose: Restarts or triggers an existing time delay.

Remarks: Use this command along with the Set Time Delay command 25 (Z) on page 73 to dynamically change an active time delay. This command overrides an existing time delayed output by setting the time delay counter to its original value, set in command 25 (Z). This command does not affect square wave generation.

Driver

Parameters:

Command:	69
Positions Array:	Output points to be retriggered.

Example: Retrigger time delays for points 2 and 3:

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 3
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 69, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `h[positions]`

If the [positions] field is omitted, a field of FFFF is assumed (all time delayed outputs are retriggered).

Example: This command instructs Optomux at address 2E (hex) to retrigger a time delay on point 5.

```
>2Eh2041cr
```

Generate N Pulses

Driver Command 72

Digital

Optomux Command i

Purpose: Instructs Optomux unit to output a counted string of pulses of a specified duration.

Driver

Parameters

Command:	72
Positions Array:	Points to output pulses.
Info Array:	First element determines on and off times of the pulses as follows: On time = timer resolution x first element value Off time = timer resolution x first element value 0 in first element cancels any existing time delay or pulse stream Second element contains the number of pulses.

Example: Generate 10 pulses of 10 ms resolution (10 ms on, 10 ms off) on points 2 and 3:

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 3
OMUX_Pos(2) = -1
OMUX_Info(1) = 10
OMUX_Info(2) = 10

result = Send_Receive_Optomux(OMUX_Handle, 72, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `i[positions][modifier][data]`

[modifier] is two digits (00 to FF) representing one-half of the period for the pulses in increments of the timer resolution setting. All pulses have a 50% duty cycle. A time of 0 cancels any existing time delay or pulse stream. Sending this command when a pulse stream is already active will retrigger it with the latest values.

[data] is the number of pulses to be output (0 to FFFF). A 0 in [data] generates 65,536 pulses.

Applicable only to B3000-series serial brains: [positions] must be either three or four digits (0000 to FFFF), with the most significant position's field first in sequence, and the least significant field last.

Example: This example instructs Optomux at address CC (hex) to output 100 pulses at point 6. Assuming the Timer Resolution Setting (TRS) is 1 (10 mSec.), the pulses will have a period of 1 second (500 mSec. OFF, 500 mSec. ON).

```
>CCi0040320064E2cr
```

Start On Pulse — Driver Command 73

Digital — Optomux Command k

Purpose: Turns on all specified outputs for a specific length of time and then turns them off.

Remarks: Before using this command, use the Set Timer Resolution command 75 (n) on page 43. Because this command is retriggerable, it can be used as a watchdog circuit by continuously sending this command at a rate faster than the pulse length. To cancel this command, set Info Array or [data] value to 1; the pulse will turn off and the outputs will be deactivated within one timer increment.

If the Info Array or [data] field contains a 0, this command does nothing.

Driver

Parameters:

Command:	73
Positions Array:	Outputs to pulse on.
Info Array:	First element contains delay length (range = 0 to 65,535 in units of the current timer resolution).

Example: Sets points 2 and 3 to turn on for 1.2 seconds and then turn off (timer resolution is assumed at 10 ms):

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 3
OMUX_Pos(2) = -1
OMUX_Info(0) = 120

result = Send_Receive_Optomux(OMUX_Handle, 73, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `k[positions][data]`

If no [data] value is given, a 0 is assumed (the command does nothing).

Example: Configures points 2 and 0 of the Optomux at address BB (hex) to generate an "ON" pulse for a duration of 20 times the timer resolution setting. All other points are not affected.

```
>BBk000514??cr
```

Start Off Pulse — Driver Command 74

Digital — Optomux Command l

Purpose: Turns off all specified outputs for a specific length of time and then turns them on.

Remarks: Before using this command, use the Set Timer Resolution command 75 (n) on page 43. Because this command is retriggerable, it can be used as a watchdog circuit by continuously sending this command at a rate faster than the pulse length. To cancel this command, set Info Array or [data] value to 1; the pulse will turn off and the outputs will be activated within one timer increment.

If the Info Array or [data] field contains a 0, this command does nothing.

Driver

Parameters:

Command:	74
Positions Array:	Outputs to pulse off.
Info Array:	First element contains delay length (range = 0 to 65,535 in units of the current timer resolution).

Example: Sets points 2 and 3 to turn off for 500 milliseconds and then turn on (timer resolution is assumed at 10 ms):

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 3
OMUX_Pos(2) = -1
OMUX_Info(0) = 50

result = Send_Receive_Optomux(OMUX_Handle, 74, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `l[positions][data]`

If no [data] value is given, a 0 is assumed (the command does nothing).

Example: This command configures point 0 of the Optomux at address 44 (hex) to generate an “OFF” pulse for a duration of six times the timer resolution setting (TRS). All other points are not affected.

```
>44l000106??cr
```

11: Digital Pulse Duration Measurement Commands

Set Pulse Trigger Polarity — Driver Command 28

Digital — Optomux Command a

Purpose: Measures ON and OFF pulses at specified input points.

Remarks: This command measures the duration of the first pulse of the appropriate level and stores the result to be recalled. As soon as a complete pulse is measured for a point, a *pulse complete bit* is set. To check whether measurements are finished, see "Read Pulse Complete Bits" on page 87.

The resolution for the duration counters is dependent upon the current timer resolution (see "Set Timer Resolution" on page 43). The default value of 10 ms allows you to measure a pulse of up to 10.92 minutes. At the lowest resolution (2.56 seconds as opposed to 0.01 seconds), you could measure a pulse of up to 2,796.16 minutes (46.6 hours).

This command does not clear preexisting duration counter values or pulse complete bits. If a point's pulse complete bit has been previously set, no measurements are made until that point's pulse complete bit and duration counter are cleared by either "Clear Duration Counters" on page 90 or "Read and Clear Duration Counters" on page 89.

Driver

Parameters:

Command:	28
Positions Array:	Input points to measure ON pulses. At all other points, OFF pulses will be measured.

Example: Measure ON pulses for points 2, 3, and 0; measure OFF pulses for all other input points. (assumed timer resolution of 10 ms):

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 3
OMUX_Pos(2) = 0
OMUX_Pos(3) = -1
OMUX_Mod(0) = 0

result = Send_Receive_Optomux(OMUX_Handle, 28, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `a[positions]`

Points corresponding to 1 bits are set to measure ON duration, while points corresponding to 0 bits are set to measure OFF duration. Maximum pulse duration is `FFFF (65,535) x timer resolution`

Example: Instruct Optomux at address 88 (hex) to configure points 2 and 0 to measure "ON" pulses; points 1 and 3 to measure "OFF" pulses; and leave all other points unchanged.

```
>88a506cr
```

Trigger on Positive Pulse — Driver Command 29

Digital — Optomux Command b

Purpose: Sets the specified input points to measure the duration of positive (ON) pulses.

Remarks: This command measures the duration of the first positive pulse and stores the result. As soon as a complete pulse is measured for a point, a *pulse complete bit* is set. To check whether measurements are finished, see "Read Pulse Complete Bits" on page 87.

The resolution for the duration counters is dependent upon the current timer resolution (see "Set Timer Resolution" on page 43). The default value of 10 ms allows you to measure a pulse of up to 10.92 minutes. At the lowest resolution (2.56 seconds as opposed to 0.01 seconds), you could measure a pulse of up to 2,796.16 minutes (46.6 hours).

This command does not clear preexisting duration counter values or pulse complete bits. If a point's pulse complete bit has been previously set, no measurements are made until that point's pulse complete bit and duration counter are cleared by either "Clear Duration Counters" on page 90 or "Read and Clear Duration Counters" on page 89.

Driver

Parameters:

Command:	29
Positions Array:	Input points to measure ON durations. Other points will remain unchanged.

Example: Measure ON duration at input points 5, 6, and 7:

```
OMUX_Pos(0) = 5
OMUX_Pos(1) = 6
OMUX_Pos(2) = 7
OMUX_Pos(3) = -1

result = Send_Receive_Optomux(OMUX_Handle, 29, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `b[positions]`

Points corresponding to 1 bits are set to measure ON durations. All other points are left unchanged.

Example: This command configures points 2 and 0 to measure ON pulses at Optomux address BB (hex). Other points are not affected.

```
>BBb51Bcr
```

Trigger on Negative Pulse — Driver Command 30

Digital — Optomux Command c

Purpose: Sets the specified input points to measure the duration of negative (OFF) pulses.

Remarks: This command measures the duration of the first negative pulse and stores the result. As soon as a complete pulse is measured for a point, a *pulse complete bit* is set. To check whether measurements are finished, see "Read Pulse Complete Bits" on page 87.

The resolution for the duration counters is dependent upon the current timer resolution (see "Set Timer Resolution" on page 43). The default value of 10 ms allows you to measure a pulse of up to 10.92 minutes. At the lowest resolution (2.56 seconds as opposed to 0.01 seconds), you could measure a pulse of up to 2,796.16 minutes (46.6 hours).

This command does not clear preexisting duration counter values or pulse complete bits. If a point's pulse complete bit has been previously set, no measurements are made until that point's pulse complete bit and duration counter are cleared by either "Clear Duration Counters" on page 90 or "Read and Clear Duration Counters" on page 89

Driver

Parameters:

Command:	30
Positions Array:	Input points to measure OFF durations. Other points will remain unchanged.

Example: Measure OFF duration at input points 5, 6, and 7:

```
OMUX_Pos(0) = 5
OMUX_Pos(1) = 6
OMUX_Pos(2) = 7
OMUX_Pos(3) = -1

result = Send_Receive_Optomux(OMUX_Handle, 30, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `c[positions]`

Points corresponding to 1 bits are set to measure OFF durations. All other points are left unchanged.

Example: This command configures points 2 and 0 to measure OFF pulses at Optomux address BB (hex). Other points are not affected.

```
>BBc51Ccr
```

Read Pulse Complete Bits — Driver Command 31

Digital — Optomux Command d

Purpose: Allows the host computer to determine which points have finished measuring pulse duration.

Driver

Parameters:

Command:	31
Info Array:	(Response) Values in the Info array elements indicate which measurements are complete (non-zero = complete; 0 = incomplete). First element is for point 0, second element is for point 1, and so on.

Example: Return the status of all pulse measurements:

```
result = Send_Receive_Optomux(OMUX_Handle, 31, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pulse_Complete = OMUX_Info(0)
```

Optomux Protocol

Format: d

Points corresponding to 1 bits have completed pulse measurement; points corresponding to 0 bits have not.

Example: Examples assume that points 0–7 have been configured as outputs. Request the pulse complete bits from Optomux address 76 (hex):

```
>76dD1cr
```

Return message from Optomux indicating the proper level pulse has been measured for points 12 and 8:

```
A1100C2cr
```

Read Pulse Duration Counters — Driver Command 32

Digital — **Optomux Command e**

Purpose: Returns the pulse duration counters for the specified points.

Remarks: Values are returned in the current timer resolution (see "Set Timer Resolution" on page 43). For example, a value of 2 equals a pulse length of: `2 x timer resolution`). Pulses up to 10.92 minutes can be timed with a resolution of 10 ms.

If this command is used before the pulse is finished, the current duration is returned. Use "Read Pulse Complete Bits" on page 87 first.

Driver

Parameters:

Command:	32
Positions Array:	Points to return duration counter values.
Info Array:	(Response) Values in the Info array elements contain the duration counter values for the corresponding point: first element is value for point 0, second element is for point 1, and so on. Reading an output returns a value of zero for the point.

Example: Return pulse duration counter values for points 2 and 5 (assumed timer resolution of 10 ms):

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 5
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 32, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_2_Duration = OMUX_Info(2)
Pos_5_Duration = OMUX_Info(5)
```

Optomux Protocol

Format: `e[positions]`

Reading an output returns a value of `????` for the point. Calculate the duration of the measured pulse as follows:

```
duration = [returned data] x timer resolution x 10 ms
```

Example: Request the counter value for point 9 on Optomux address FF (hex):

```
>FFe20083cr
```

Message returned from Optomux indicating that the duration of the pulse measured was 8.24 minutes. This example assumes a timer resolution value of 2.

```
A6090CFcr
6,090 hex = 24,720 decimal
24,720 x timer resolution x 10 ms = 8.24 minutes
```

Read and Clear Duration Counters — Driver Command 33

Digital — Optomux Command f

Purpose: Returns pulse duration counters for the specified points and then clears them.

Remarks: Reads the specified counters, and then clears the counters and the pulse complete bits. Values are returned in the current timer resolution (see "Set Timer Resolution" on page 43). For example, a value of 2 equals a pulse length of: `2 x timer resolution`). Pulses up to 10.92 minutes can be timed with a resolution of 10 ms.

If this command is used before the pulse is finished, the current duration is returned. Use "Read Pulse Complete Bits" on page 87 first.

Driver

Parameters:

Command:	33
Positions Array:	Points for which duration counter values will be returned and then reset.
Info Array:	(Response) Values in the Info array elements contain the duration counter values for the corresponding point: first element is value for point 0, second element is for point 1, and so on. Reading an output returns a value of zero for the point.

Example: Return and clear pulse duration counter values for points 8 and 9 (assumed timer resolution of 10 ms):

```
OMUX_Pos(0) = 8
OMUX_Pos(1) = 9
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 33, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_8_Duration = OMUX_Info(8)
Pos_9_Duration = OMUX_Info(9)
```

Optomux Protocol

Format: `f[positions]`

Reading an output returns a value of `????` for the point. Calculate the duration of the measured pulse as follows:

```
duration = [returned data] x timer resolution x 10 ms
```

Example: Request counter value for point 0 of Optomux at address 1 (hex); then clear counter:

```
>01f1F8cr
```

Message returned from Optomux indicating that the duration of the pulse measured was 8.24 minutes. This example assumes a timer resolution value of 2.

```
A6090CFcr
6,090 hex = 24,720 decimal
24,720 x timer resolution x 10 ms = 8.24 minutes
```

Clear Duration Counters — Driver Command 34

Digital — Optomux Command g

Purpose: Resets duration counters and pulse complete bits for the specified points to enable measurement of the next pulse.

Driver

Parameters

Command:	34
Positions Array:	Points for which duration counters and pulse complete bits will be cleared.

Example: Clear duration counters and pulse complete bits on points 5, 6, and 7:

```
OMUX_Pos(0) = 5
OMUX_Pos(1) = 6
OMUX_Pos(2) = 7
OMUX_Pos(3) = -1

result = Send_Receive_Optomux(OMUX_Handle, 34, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `g[positions]`

Example: Clear duration counters for points 5 and 11 on the Optomux I/O unit with address 22 (hex).

```
>22g82065cr
```

12: Analog Read/Write Commands

Write Analog Outputs — Driver Command 35

Analog — Optomux Command J

Purpose: Writes a specified value to one or more analog outputs.

Driver

Parameters:

Command:	35
Positions Array:	Points to write the value to.
Info Array:	First element contains value to be written to each point in the positions array.

Example: Write 30% of full scale to points 3 and 4.

```
OMUX_Pos(0) = 3
OMUX_Pos(1) = 4
OMUX_Pos(2) = -1
OMUX_Info(0) = 0.30 * 4095

result = Send_Receive_Optomux(OMUX_Handle, 35, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `J[positions][data]`

[data] contains a 12-bit ASCII-hex value that is written to each point with a 1 bit.

Example: Assume Optomux at address FF (hex) has DA4 (0-5 V) output points 0–3. We want to set points 1 and 3 to 1.25 volts.

```
(1.25 V-0 V)/5 V * 4095 = 1024 decimal = 400 hex
```

The command becomes:

```
>FFJ000A4003Bcr
```

Read Analog Outputs — Driver Command 36

Analog — Optomux Command K

Purpose: Returns the value of each analog output specified.

Driver

Parameters:

Command:	36
Positions Array:	Points to read.
Info Array:	(Response) Values in the Info array elements indicate values for the corresponding points: first element is value for point 0, second element is for point 1, and so on.

Example: Return values for analog outputs in points 7 and 8.

```
OMUX_Pos(0) = 7
OMUX_Pos(1) = 8
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 36, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_7_Value = OMUX_Info(7)
Pos_8_Value = OMUX_Info(8)
```

Optomux Protocol

Format: `K[positions]`

Returns three ASCII-hex characters for each point read. Returns `???` if the requested point is configured as an input. Since only three characters are returned for each point, do not subtract the offset of 1,000 hex from the value.

Example: Points 0–7 are configured as inputs. Return the current value of point 8 at address 86 (hex):
`>86K1004Acr`

NOTE: A 0 value is assumed for leading points which are omitted within the [positions] field.
`>86K01007Acr` is a functionally identical command.

This is the response if point 8 had a hex value of B2E.
`AB2EB9cr`

The following command requests the values for points 7, 8, and 9:
`>86K38054cr`

This response indicates that point 9 is an output with a value of 000 hex, point 8 is an output with a value of BE2 hex, and point 7 is configured as an input.
`A000BE2???06cr`

Update Analog Outputs — Driver Command 46

Analog — Optomux Command S

Purpose: Writes values to one or more analog outputs.

Remarks: Use this command to write different values to multiple outputs. Use Write Analog Outputs (page 91) to write the same value to multiple outputs. Attempts to write values to points configured as inputs will be ignored.

Driver

Parameters:

Command:	46
Positions Array:	Points to write values to. Other points are not affected.
Info Array:	Info array elements contain the value to write to each corresponding point: first element is value to write to point 0, second element to point 1, and so on.

Example: Write 61% of full scale to position 3 and 84% of full scale to point 6:

```
OMUX_Pos(0) = 3
OMUX_Pos(1) = 6
OMUX_Pos(2) = -1
OMUX_Info(3) = 4095 * 0.61
OMUX_Info(6) = 4095 * 0.84

result = Send_Receive_Optomux(OMUX_Handle, 46, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `S[positions][data]`

[data] must contain three ASCII-hex digits for each point specified. Place the 12-bit values in order starting with the highest numbered point and ending with the lowest numbered point.

Example: Instruct Optomux at address D0 (hex) to write the value 1F0 to point 2, 0C0 to point 5, and FFF to point 9.

```
>D0S0224FFF0C01F0ABcr
```

Read Analog Inputs — Driver Command 37

Analog — Optomux Command L

Purpose: Returns the value of each specified analog input.

Driver

Parameters:

Command:	37
Positions Array:	Points to read.
Info Array:	(Response) Values in the Info array elements indicate values for the corresponding points: first element is value for point 0, second element is for point 1, and so on. Reading an output point returns a -4,096 in that element.

Example: Read the values at analog input points 7 and 10.

```
OMUX_Pos(0) = 7
OMUX_Pos(1) = 10
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 37, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_7_Value = OMUX_Info(7)
Pos_10_Value = OMUX_Info(10)
```

Optomux Protocol

Format: `L[positions]`

The normal range of values for inputs is 1000h to 1FFFh. If the most significant character is any character other than 1, it is an indication of over or under range. 0 = under range; >1 = over range.

Example: Read the values of points 0 and 2 from an Optomux unit at address 90 (hex).

```
>90L5EAcr
```

The response indicates a value of 1,000 hex for point 2 and a value of 1,888 hex for point 0. Both values are valid readings in the range of 1,000 to 1FFF hex.

```
A1000188889cr
```

Request the values for points 10, 8, and 0:

```
>90L5014Bcr
```

Response (8 and 10 are outputs):

```
A????????1FEEF9cr
```

NOTE: If an output point is configured as an input, and that value is requested, Optomux will return 0000. A value of 0000 will result in a decimal value of -4,096 after the 1,000 hex offset is subtracted.

Average and Read Input — Driver Command 38

Analog — Optomux Command M

Purpose: Averages the value of a single point over a specified number of samples and returns the result.

Remarks: This command returns a response only when it is finished averaging. If the number of samples is very large and the system is communicating serially, it can tie up the bus while waiting for an acknowledgment message. You can use Start Averaging Inputs on page 96, Read Average Complete Bits on page 97, and Read Averaged Inputs on page 98 commands instead of this command.

Averaging is done using a continuous running average with a sample rate of 100 milliseconds. After the number of samples has been reached, the value is returned to the host. The following equation shows how the average is calculated:

```
Average = ((N-1) (Old Average) + (New Reading))/N
```

Driver

Parameters:

Command:	38
Positions Array:	First element contains the point to average.
Info Array:	(Command) First element contains the number of samples to average (maximum 255). (Response) Element corresponding to point contains averaged value.

Example: Average point 4's values over 8 samples:

```
OMUX_Pos(0) = 4
OMUX_Pos(1) = -1
OMUX_Info(0) = 8

result = Send_Receive_Optomux(OMUX_Handle, 38, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_4_Avg_Val = OMUX_Info(4)
```

Optomux Protocol

Format: `M[position][data]`

[position] is a single ASCII-hex character.

[data] contains two ASCII-hex digits specifying the number of samples (01–FF hex; 1–255 decimal).

Example: Return the average of 10 samples of point 7 on Optomux at address 03 (hex).

```
>03M70A58cr
```

Start Averaging Inputs — Driver Command 47

Analog — Optomux Command T

Purpose: Instructs the Optomux unit to start averaging the value of input points over a specified number of samples.

Remarks: Averaging is done using a continuous running average with a sample rate of 100 milliseconds. The following equation shows how the average is calculated:

```
Average = ((N-1) (Old Average) + (New Reading))/N
```

To find out whether sampling is finished, use Read Average Complete Bits (page 97). To return the averaged value, use Read Averaged Inputs (page 98) or Read Average Temperature Inputs (page 101).

Driver

Parameters:

Command:	47
Positions Array:	Points at which the unit will start averaging. Other points are not affected.
Info Array:	First element contains the number of samples to average (maximum 255).

Example: Start averaging values at input points 7 and 8; average 15 samples:

```
OMUX_Pos(0) = 7
OMUX_Pos(1) = 8
OMUX_Pos(2) = -1
OMUX_Info(0) = 15

result = Send_Receive_Optomux(OMUX_Handle, 47, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `T[positions][data]`

[data] contains 0 to 4 ASCII-hex digits specifying the number of samples.

Example: Instruct Optomux at address 1A (hex) to start calculating the average of 1F samples for points 3, 4, and 10.

```
>1AT04181F0Acr
```

Read Average Complete Bits — Driver Command 48

Analog — Optomux Command i

Purpose: Allows the host to determine which points have completed averaging.

Remarks: Use Start Averaging Inputs (page 96) before using this command. A 1 bit indicates that input averaging has been completed; a 0 bit indicates that it has not.

Ignore response bits corresponding to points configured as outputs or where averaging has not been started.

Driver

Parameters:

Command:	48
Info Array:	(Response) Values in the Info array elements indicate whether averaging is complete for the corresponding points. Non-zero = complete; 0 = incomplete.

Example: Read average complete bits. Assume averaging has been started on points 0–7.

```
result = Send_Receive_Optomux(OMUX_Handle, 48, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
'Loop through 8 points
For Index = 0 to 7
  If (OMUX_Info(Index) = 1) then
    'averaging completed
  Else
    'wait and check again
  End If
Next Index
```

Optomux Protocol

Format: `i`.

Example: Assume that commands have been sent instructing the unit at address A1 (hex) to start averaging at point 9, 10, and 11. Return average complete bits:

```
>A1iDBcr
```

Response from Optomux indicating that averaging has been completed for points 9 and 11. Optomux has not completed averaging point 10.

```
A0A00D1cr
```

Read Averaged Inputs — Driver Command 49

Analog — Optomux Command U

Purpose: Returns the results of averaging input values at specified input points.

Remarks: Input averaging must have already been started using Start Averaging Inputs on page 96. Check whether averaging is completed by using Read Average Complete Bits (page 97). If averaging has not been completed, Read Averaged Inputs returns the current value of the average.

Driver

Parameters:

Command:	49
Positions Array:	Contains the input points where averages are to be returned.
Info Array:	(Response) Values in the Info array elements show averaged values for the corresponding point: first element is average value for point 0, second element is for point 1, and so on.

Example: Read averaged values for points 7 and 10. Assume that averaging has already been started on them:

```
OMUX_Pos(0) = 7
OMUX_Pos(1) = 10
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 49, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_7_Avg_Value = OMUX_Info(7)
Pos_10_Avg_Value = OMUX_Info(10)
```

Optomux Protocol

Format: `U[positions]`

If the input goes out of range, a value of `0000` is returned for the point until the out-of-range condition is resolved and input averaging is restarted.

Example: This message instructs Optomux at address FF (hex) to return the averages for points 2 and 5. Averaging has been started already.

```
>FFU2447cr
```

Response from Optomux indicating the average for point 2 is 1F0C and the average for point 5 is 1A0D:

```
A1A0D1F0CD0cr
```

Read Temperature Inputs — Driver Command 77

Analog — Optomux Command l

Purpose: Returns the temperature at the specified input points.

Remarks: Before using this command, set the temperature probe type (see page 44). For additional information on reading temperatures, see Appendix , "C: Reading Negative Numbers and Temperature."

Driver

Parameters:

Command:	77
Positions Array:	Contains the input points whose temperatures are to be returned.
Info Array:	(Response) Returns temperatures in 16ths of a degree Celsius. Values in the Info array elements show temperature for the corresponding point: first element is temperature for point 0, and so on. Reading a temperature from a point that has not had its probe type set returns a value of –4096.

Example: Read the temperature at input points 7 and 10:

```
OMUX_Pos(0) = 7
OMUX_Pos(1) = 10
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 77, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Temp_Pos_7 = OMUX_Info(7) / 16
Temp_Pos_10 = OMUX_Info(10) / 16
```

Optomux Protocol

Format: `l[positions]`

NOTE: This command is a lower-case `L`, not an upper-case `I`.

Optomux returns four ASCII-hex characters representing a signed 16-bit number for each point corresponding to a 1 in the [positions] bitmask. The first three characters represent whole degrees Celsius, while the last character represents the fractional part (sixteenths—remember, it's hex) of degrees Celsius. In other words, the data returned is 16 times the temperature in degrees Celsius.

If the point contains an output or its temperature probe has not been sent, a `????` is returned.

Points that read below the scale of the set probe type return a value of `-273` ºC. Points that read above the scale for the set probe type return `2047` ºC.

Example: This command instructs Optomux to return the temperature values for inputs at points 9 and 11 (assumes probe type was set for these points):

```
>FFl0A00C9cr
```

The response indicates that point 11 has a value of 08A8 hex and point 9 has a value of 046B hex. Converting the hex values to decimal, we get point 11 equal to 2,216, and point 9 equal to 1,131:

```
A08A8046BBDcr
```

Since these returned values are 16 times greater than the actual temperature, divide the values by 16:

```
Point 11 Temperature = 2,216/16 = 138.5º C
Point 9 Temperature = 1,131/16 = 70.68º C
```

The next example reads the temperature value from point 15:

```
>FFl8000C0cr
```

The response shows that the value for point 15 is F956:

```
AF956EAcr
```

The sign bit is set; therefore this value is a negative number. Converting the signed value of F956 to decimal results in a value of -1,705. Dividing -1,705 by 16 results in a temperature of -106.56º C. See "Converting Negative Numbers" on page 149 for more information.

Read Average Temperature Inputs — Driver Command 79

Analog — Optomux Command o

Purpose: Returns the averaged temperature at the specified input points.

Remarks: Before using this command, set the temperature probe type (see page 44). Start averaging using the Start Averaging Input command (page 96). To find out whether sampling is finished, use Read Average Complete Bits (page 97). For additional information on reading temperatures, see Appendix , "C: Reading Negative Numbers and Temperature."

Driver

Parameters:

Command:	79
Positions Array:	Contains the input points whose averaged temperatures are to be returned.
Info Array:	(Response) Returns averaged temperatures in 16ths of a degree Celsius. Values in the Info array elements show temperature for the corresponding point: first element is temperature for point 0, and so on. Reading a temperature from a point that has not had its probe type set returns a value of –4096.

Example: Read the averaged temperature at input points 7 and 10:

```
OMUX_Pos(0) = 7
OMUX_Pos(1) = 10
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 79, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Avg_Temp_Pos_7 = OMUX_Info(7) / 16
Avg_Temp_Pos_10 = OMUX_Info(10) / 16
```

Optomux Protocol

Format: `o[positions]`

Optomux returns four ASCII-hex characters representing a signed 16-bit number for each point corresponding to a 1 in [positions]. The first three characters are whole degrees Celsius, while the last character is the fractional part (sixteenths) of degrees Celsius. The data returned is 16 times the temperature in degrees Celsius.

If the point contains an output or its temperature probe has not been sent, a `????` is returned.

Points that read below the scale of the set probe type return a value of `-273` ºC. Points that read above the scale for the set probe type return `2047` ºC.

Example: Return the averaged temperature values for inputs at points 9 and 11 (assumes that the probe type has been set and input averaging has been started):

```
>FFo0A00CCcr
```

The response indicates that point 11 has a value of 08A8 hex and point 9 has a value of 046B hex. Converting the hex values to decimal, we get point 11 equal to 2,216, and point 9 equal to 1,131:

```
A08A8046BBDcr
```

Since the returned values are 16 times greater than the actual temperature, divide them by 16:

```
Point 11 Temperature = 2,216/16 = 138.5 °C
Point 9 Temperature = 1,131/16 = 70.68 °C
```

The next example reads the average temperature value from point 15:

```
>FFo8000C3cr
```

The response shows that the value for point 15 is F956.

```
AF956EAcr
```

Since the sign bit is set, this value is a negative number. Converting the signed value of F956 to decimal results in a value of -1,705. Dividing -1,705 by 16 results in a temperature of -106.56º C. See "Converting Negative Numbers" on page 149 for more information.

13: Analog Input Range Commands

Set Input Range — Driver Command 39

Analog — Optomux Command N

Purpose: Defines the high and low limits for the specified input points.

Remarks: This command defines a range for the specified inputs. If an input is out of the specified range, one of two latches is set: high or low limit exceeded. To read these latches, use the Read Out-of-Range Latches command 40 (O) on page 104.

Driver

Parameters:

Command:	39
Positions Array:	Points to be set for high and low limits.
Info Array:	First element contains the high limit, second element contains the low limit.

Example: Set the input range of points 0 and 3 to 25% to 50% of full scale:

```
OMUX_Pos(0) = 0
OMUX_Pos(1) = 3
OMUX_Pos(2) = -1
OMUX_Info(0) = 4095 * 0.50
OMUX_Info(1) = 4095 * 0.25

result = Send_Receive_Optomux(OMUX_Handle, 39, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `N[positions][data]`

[data] contains six ASCII-hex characters. three representing the high limit (12-bits), followed by three representing the low limit.

Example: For points 0 and 1 on Optomux at address 09 (hex), set the upper limit at 760 hex and the lower limit at 710 hex. The defined range of values as read back by the Read Analog Inputs command will be from 1,710 to 1,760.

```
>09N0003760710AFcr
```

Read Out-of-Range Latches — Driver Command 40

Analog — Optomux Command O

Purpose: Returns the high and low out-of-range latches for all points.

Remarks: Before using this command, use Set Input Range (page 103) to set high and low limits for the points. This command does not clear the latches.

Driver

Parameters:

Command:	40
Info Array:	(Response) The Info array elements indicate values for their corresponding points: 0 = point has remained within limits or is an output. 1 = low-limit latch has been set 2 = high-limit latch has been set. 3 = both low- and high-limit latches have been set.

Example: Read current state of out-of-range latches:

```
result = Send_Receive_Optomux(OMUX_Handle, 40, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
'Loop through 16 points
For Index = 0 to 15
  If (OMUX_Info(Index) = 0) then
    'stayed within limits or configured as output
    Else (OMUX_Info(Index) = 1) then
      'went below limit
    Else (OMUX_Info(Index) = 2) then
      'went above limit
    Else (OMUX_Info(Index) = 3) then
      'went both below and above limits
  End If
Next Index
```

Optomux Protocol

Format: `O   (hex 4F)`

The first four characters show the status of high-limit latches: the remaining four show low-limit latches (`HHHHLLLL`). Points corresponding to 1 bits in the first 16-bit value have exceeded their high limits. Points corresponding to 1 bits in the second 16-bit value have fallen below their low limits.

Example: Read Out-Of-Range latches at Optomux address 71 (hex):

```
>71OB7cr
```

Response shows that point 0 has exceeded the high limit since the latches were last cleared, and point 4 has exceeded the low limit:

```
A0001001062cr
```

Read and Clear Out-of-Range Latches — Driver Command 41

Analog — Optomux Command P

Purpose: Returns the high and low out-of-range latches for all points and then clears latches for specified points.

Remarks: Before using this command, use Set Input Range (page 103) to set high and low limits for the points.

Driver

Parameters:

Command:	41
Positions Array:	Points whose out-of-range latches will be cleared.
Info Array:	(Response) The Info array elements indicate values for their corresponding points: 0 = point has remained within limits or is an output. 1 = low-limit latch has been set 2 = high-limit latch has been set. 3 = both low- and high-limit latches have been set.

Example: Read current state of out-of-range latches and clears latches at points 2 and 5:

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 5
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 41, OMUX_Pos(0), _ OMUX_Mod(0),
          OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
'Loop through 16 points
For Index = 0 to 15
  If (OMUX_Info(Index) = 0) then
    'stayed within limits or configured as output
    Else (OMUX_Info(Index) = 1) then
      'went below limit
    Else (OMUX_Info(Index) = 2) then
      'went above limit
    Else (OMUX_Info(Index) = 3) then
      'went both below and above limits
  End If
Next Index
```

Optomux Protocol

Format: `P[positions]`

The first four characters show the status of high-limit latches: the remaining four show low-limit latches (`HHHHLLLL`). Points corresponding to 1 bits in the first 16-bit value have exceeded their high limits. Points corresponding to 1 bits in the second 16-bit value have fallen below their low limits.

Example: Read out-of-range latches and reset latch at point 2 at Optomux address 70 (hex):

```
>70P4EBcr
```

Point 0 has exceeded the upper limit, point 2 has exceeded the lower limit; only point 2 will have the latch reset:

```
A0001000465cr
```

Clear Out-of-Range Latches — Driver Command 42

Analog — Optomux Command Q

Purpose: Resets out-of-range latches for specified points.

Remarks: To read and clear latches at the same time, see page 105.

Driver

Parameters:

Command:	42
Positions Array:	Points whose out-of-range latches should be cleared.

Example: Clear out-of-range latches for point 6.

```
OMUX_Pos(0) = 6
OMUX_Pos(1) = -1

result = Send_Receive_Optomux(OMUX_Handle, 42, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `Q[positions]`

Out-of-range latches are reset for points corresponding to 1 bits in [positions]. If no [positions] field is included in the command, all latches are reset.

Example: Clear latches at points 0 and 2 at Optomux address 00 (hex).

```
>00Q5E6cr
```

Clear all latches at Optomux address 99 (hex).

```
>99QC3cr
```

Read Lowest Values — Driver Command 58

Analog — Optomux Command a

Purpose: Returns the lowest readings at specified input points.

Remarks: The reading will be the lowest the unit has encountered since last receiving a Read and Clear Lowest Values (page 110) or Clear Lowest Values (page 109) command. Units set all low values to an extreme over range of 2000 hex (for the driver) or 3000 hex (for the protocol) upon power-up.

Driver

Parameters:

Command:	58
Positions Array:	Points whose lowest values should be read.
Info Array:	(Response) Values in the Info array elements show lowest values for the corresponding point: first element is lowest value for point 0, second element is for point 1, and so on.

Example: Read lowest value for points 7 and 10 (assume that points 7 and 10 are 0–5 V inputs):

```
OMUX_Pos(0) = 7
OMUX_Pos(1) = 10
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 58, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_7_Low_Val = OMUX_Info(7) / 4095 * 5
Pos_10_Low_Val = OMUX_Info(10) / 4095 * 5
```

Optomux Protocol

Format: `a[positions]`

Four ASCII-hex digits are returned for each point corresponding to 1 bits in the [positions] field. Values are returned in sequence from the highest to lowest point.

Example: Return the low values for points 0, 5, 7, and 10 on the unit at address 2A (hex):

```
>2Aa4A17Acr
```

Response:

```
A130110010FFA100D59cr
```

This response indicates:

```
Lowest Value for point 0 = 100D
Lowest Value for point 5 = 0FFA (under range)
Lowest Value for point 7 = 1,001
Lowest Value for point 10 = 1,301
```

Clear Lowest Values — Driver Command 59

Analog — Optomux Command b

Purpose: Clears the lowest reading for specified input points.

Remarks: Units clear the lowest readings by setting the lowest value to an extreme over range of 2000 hex (for the driver) or 3000 hex (for the protocol). This allows the unit to store the lowest value encountered in subsequent readings.

Driver

Parameters:

Command:	59
Positions Array:	Points whose lowest readings should be cleared.

Example: Clear lowest readings at points 0 through 6:

```
For Index = 0 to 6
  OMUX_Pos(Index) = Index
Next Index

result = Send_Receive_Optomux(OMUX_Handle, 59, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `b[positions]`

The unit clears the lowest reading for each input point corresponding to 1 bits in the [positions] field. If [positions] is omitted, a bitmask of FFFF is assumed and all lowest values are cleared.

Example: Clear the lowest values for points 0, 5, 7, and 10 at address 2A (hex):

```
>2Ab4A17Bcr
```

Read and Clear Lowest Values — Driver Command 60

Analog — Optomux Command c

Purpose: Returns the lowest readings for specified input points and then clears the lowest values.

Remarks: The reading will be the lowest the unit has encountered since last receiving a Read and Clear Lowest Values (page 110) or Clear Lowest Values (page 109) command. Units clear the lowest values by setting the lowest value to an extreme over range of 2000 hex (for the driver) or 3000 hex (for the protocol). This allows the unit to store the lowest value encountered in subsequent readings.

Driver

Parameters:

Command:	60
Positions Array:	Points whose lowest values should be read and then cleared.
Info Array:	(Response) Values in the Info array elements contain lowest readings for their corresponding points.

Example: Read and clear lowest values for points 4 and 5.

```
OMUX_Pos(0) = 4
OMUX_Pos(1) = 5
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 60, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_4_Low_Value = OMUX_Info(4)
Pos_5_Low_Value = OMUX_Info(5)
```

Optomux Protocol

Format: `c[positions]`

The unit returns the lowest reading for each point corresponding to 1 bits in the [positions] field. Four ASCII-hex digits are returned for each point. Values are returned in sequence from the highest to lowest point.

Example: Return lowest values for points 0, 5, 7, and 10 at address 2A (hex) and then clear them.

```
>2Ac4A17Ccr
```

Response:

```
A130110010FFA100D59cr
```

This response indicates:

```
Lowest Value for point 0 = 100D
Lowest Value for point 5 = 0FFA (under-range)
Lowest Value for point 7 = 1,001
Lowest Value for point 10 = 1,301
```

Read Peak Values — Driver Command 61

Analog — Optomux Command d

Purpose: Returns the highest (peak) readings at specified input points.

Remarks: The reading will be the highest the unit has encountered since last receiving a Read and Clear Peak Values (page 113) or Clear Peak Values (page 112) command. Units set all low values to 0000 hex (under range) upon power-up.

Driver

Parameters:

Command:	61
Positions Array:	Points whose peak values should be read.
Info Array:	(Response) Values in the Info array elements contain peak readings for their corresponding points.

Example: Read peak values for points 5 and 6:

```
OMUX_Pos(0) = 5
OMUX_Pos(1) = 6
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 61, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_5_Peak_Value = OMUX_Info(5)
Pos_6_Peak_Value = OMUX_Info(6)
```

Optomux Protocol

Format: `d[positions]`

The unit returns the peak reading for each point corresponding to 1 bits in the [positions] field. Four ASCII-hex digits are returned for each point. Values are returned in sequence from the highest to lowest point.

Example: Return the peak values for points 5, 7, 9, and 11 at address 2A (hex):

```
>2AdAA089cr
```

Response:

```
A1DFF10C11FAC1FB0BAcr
```

The response indicates:

```
peak Value for point 5 = 1FB0
peak Value for point 7 = 1FAC
peak Value for point 9 = 10C1
peak Value for point 11 = 1DFF
```

Clear Peak Values — Driver Command 62

Analog — Optomux Command e

Purpose: Clears the highest (peak) reading for specified input points.

Remarks: Units clear the peak readings by setting the highest value to 0000 hex (under range). This allows the unit to store the highest value encountered in subsequent readings.

Driver

Parameters:

Command:	62
Positions Array:	Points whose peak readings should be cleared.

Example: Clear peak readings at points 0 through 15:

```
For Index = 0 to 15
  OMUX_Pos(Index) = Index
Next Index

result = Send_Receive_Optomux(OMUX_Handle, 62, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `e[positions]`

The unit will clear the peak values for each input point corresponding to 1 bits in the [positions] field. If [positions] is omitted, a bitmask of FFFF is assumed and all peak values are cleared.

Example: Instructs Optomux at address 2A (hex) to clear the peak values for points 6, 7, 8, and 9:
`>2Ae3C07Ecr`

Read and Clear Peak Values — Driver Command 63

Analog — Optomux Command f

Purpose: Returns the highest (peak) readings for specified input points and then clears the highest values.

Remarks: The reading will be the highest the unit has encountered since last receiving a Read and Clear Peak Values (page 113) or Clear Peak Values (page 112) command. Units clear the peak readings by setting the highest value to 0000 hex (under range). This allows the unit to store the highest value encountered in subsequent readings.

Driver

Parameters:

Command:	63
Positions Array:	Points whose peak values should be read and then cleared.
Info Array:	(Response) Values in the Info array elements contain peak readings for their corresponding points.

Example: Read and clear peak values for point 4:

```
OMUX_Pos(0) = 4
OMUX_Pos(1) = -1

result = Send_Receive_Optomux(OMUX_Handle, 63, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_4_Peak_Value = OMUX_Info(4)
```

Optomux Protocol

Format: `f[positions]`

The unit returns the highest reading for each point corresponding to 1 bits in the [positions] field. If [positions] is omitted, Optomux assumes a bitmask of FFFF, and peak values at all points are cleared. Four ASCII-hex digits are returned for each point. Values are returned in sequence from the highest to lowest point.

Example: Return and then clear the peak values for points 14 and 15 at address 2A (hex):

```
>2AfC000ACcr
```

Response:

```
A1DFA1C1CE4
```

This response indicates:

```
Peak Value for point 14 = 1C1C
Peak Value for point 15 = 1DFA
```

14: Analog Offset/Gain Commands

Calculate Offsets — Driver Command 52

Analog — Optomux Command g

Purpose: Calculates and returns offsets for specified input points.

Remarks: Because offset values are calculated using the current values of the inputs, use this command when the specified points are receiving the value you wish to consider zero scale. This is usually done during system installation and calibration, when known inputs (zero scale) can be applied to the points.

Use the offset values obtained from this command to set offset values (see page 117) during Optomux initialization.

Always set offsets before calculating gain coefficients (see page 121).

Driver

Parameters:

Command:	52
Positions Array:	Points where offsets should be calculated.
Info Array:	(Response) Elements contain offsets for the corresponding points.

Example: Calculate offsets for points 4 and 13.

```
OMUX_Pos(0) = 4
OMUX_Pos(1) = 13
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 52, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_4_Offset = OMUX_Info(4)
Pos_13_Offset = OMUX_Info(13)
```

Optomux Protocol

Format: `g[positions]`

Four ASCII-hex digits are returned for each point specified in the [positions] field. Values are returned in sequence from the highest to lowest point in the same form required by the Set Offsets command. A negative offset is represented as a two's complement number. (See more information on negative numbers in Appendix .)

```
0001 = +1 Offset
FFFF = -1 Offset
```

If the point is an output, `????` is returned.

Example: Calculate and return offset values for points 0, 1, 2 and 3 at address DF (hex):

```
>DFgF37cr
```

Response from Optomux:

```
A000AFFFE000100036Ccr
Offset for point 0 = 0003
Offset for point 1 = 0001
Offset for point 2 = FFFE
Offset for point 3 = 000A
```

Calculate and return offset values for points 0, 2, 4, 6, 8, 10, 12, and 14 at address AF (hex):

```
>AFg5555C2cr
```

Response from Optomux -

```
A0020FFE9000EFFFD????????001A000D55cr
Offset for point 0 = 000D
Offset for point 2 = 001A
Offset for point 4 = ???? (point 4 is an output)
Offset for point 6 = ???? (point 6 is an output)
Offset for point 8 = FFFD
Offset for point 10 = 000E
Offset for point 12 = FFE9
Offset for point 14 = 0020
```

Set Offsets — Driver Command 53

Analog — Optomux Command W

Purpose: Sets offset values for specified input points.

Remarks: This command uses offset values in the same form in which they are returned by the Calculate Offsets command (see page 115). If a specified point is an output, the command is ignored for that point.

Always set offsets before calculating gain coefficients (see page 121).

Driver

Parameters:

Command:	53
Positions Array:	Points where offset values should be set.
Info Array:	Values in the Info array elements contain offset values to be written to the corresponding points.

Example: Set offsets for all 16 points (assumes that offset values were already placed in an array named Offsets):

```
For Index = 0 to 15
  OMUX_Pos(Index) = Index
  OMUX_Info(Index) = Offsets(Index)
Next Index

result = Send_Receive_Optomux(OMUX_Handle, 53, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `W[positions][data]`

[data] contains four ASCII-hex digits for each point specified in the [positions] field. The values are placed in the data field in order starting with the highest numbered point and ending with the lowest numbered point. If the command does not include four digits of data for each bit specified in the [positions] field, a data field error will be returned.

Offsets are placed in the [data] field in the same form as they are returned by the Calculate Offsets command (page 115). A negative offset is represented as a two's complement number. (See more information on negative numbers in Appendix .)

```
0001 = +1 Offset
FFFF = -1 Offset
```

Example: Set the following offsets at address DF (hex):

```
Offset for point 0 = 0003
Offset for point 1 = 0001
Offset for point 2 = FFFE
Offset for point 3 = 000A

>DFW000F000AFFFE0001000323cr
```

Set the following offset values for points 0, 2, 4, 6, 8, 10, 12, and 14 at address DF (hex):

```
Offset for point 0 = 000D
Offset for point 2 = 001A
Offset for point 4 = 0000
Offset for point 6 = 0000
Offset for point 8 = FFFD
Offset for point 10 = 000E
Offset for point 12 = FFE9
Offset for point 14 = 0020

>DFW55550020FFE9000EFFFD00000000001A000D92cr
```

Calculate and Set Offsets — Driver Command 54

Analog — Optomux Command h

Purpose: Calculates and sets offsets for specified input points, and then returns calculated offsets to the host.

Remarks: Because offset values are calculated using the current values of the inputs, use this command when the specified points are receiving the value you wish to consider zero scale. This is usually done during system installation and calibration, when known inputs (zero scale) can be applied to the points. The offset values returned to the host can be saved and used during system initialization (Set Offsets command, page 117).

Attempts to calculate and set offsets on an output will have no effect.

Always set offsets before calculating gain coefficients (see page 121).

Driver

Parameters:

Command:	54
Positions Array:	Points where offsets should be calculated and set.
Info Array:	(Response) Elements contain offsets for the corresponding points.

Example: Calculate and set offsets for points 2 and 3, and save the offset values:

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = 3
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 54, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)

Pos_2_Offset = OMUX_Info(2)
Pos_3_Offset = OMUX_Info(3)
```

Optomux Protocol

Format: `h[positions]`

Four ASCII-hex digits are returned for each point specified in the [positions] field. Values are returned in sequence from the highest to lowest point in the same form required by the Set Offsets command. A negative offset is represented as a two's complement number. (See more information on negative numbers in Appendix .)

```
0001 = +1 Offset
FFFF = -1 Offset
```

If the point is an output, `????` is returned.

Example: Calculate and set offsets for points 0, 2, 5, 12, and 13 at address 00 (hex):

```
>00h302592cr
```

Response from Optomux indicating that the following offsets have been set:

```
Offset for point 0 = 0003
Offset for point 2 = 0001
Offset for point 5 = FFFE
Offset for point 12 = 0010
Offset for point 13 = 000B

A000B0010FFFE000100032Ecr
```

Calculate and set offset values for points 0, 2, 4, 6, 8, 10, 12, and 14 at address DF (hex):

```
>DFh5555C6cr
```

Response from Optomux indicating that the following offset values have been set:

```
Offset for point 0 = 000D
Offset for point 2 = 001A
Offset for point 4 = ???? Is an Output
Offset for point 6 = ???? Is an Output
Offset for point 8 = FFFD
Offset for point 10 = 000E
Offset for point 12 = FFE9
Offset for point 14 = 0020

A0020FFE9000EFFFD????????001A000D55cr
```

Calculate Gain Coefficients — Driver Command 55

Analog — Optomux Command X

Purpose: Calculates and returns gain coefficients for specified input points.

Remarks: Before using this command, set offsets for points. See Set Offsets (page 117) or Calculate and Set Offsets (page 119).

Because gain values are calculated using the current values of the inputs, use this command when the specified points are receiving the value you wish to consider full scale. This is usually done during system installation and calibration, when known inputs (full scale) can be applied to the points. Use the values obtained from this command to set gain coefficient values (see page 123) during Optomux initialization.

Driver

Parameters:

Command:	55
Positions Array:	Points where gain coefficients should be calculated.
Info Array:	(Response) Values in the Info array elements indicate gain coefficient values for the corresponding points. Values returned are 10,000 times the actual gain coefficients (a value of 14,000 represents a gain coefficient of 1.40). This is the same format used by other gain commands.

Example: Calculate and return the gain coefficients for points 4 and 5:

```
OMUX_Pos(0) = 4
OMUX_Pos(1) = 5
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 55, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Handling the returned message:

```
Pos_4_Gain = OMUX_Info(4)
Pos_5_gain = OMUX_Info(5)
```

Optomux Protocol

Format: `X[positions]`

Four ASCII-hex digits are returned for each point specified. Values are returned in sequence from the highest to lowest point in the same form required by the Set Gain Coefficients command. A decimal point is assumed to be between the first digit and the last three. The first ASCII-hex digit represents the whole part of the coefficient and the last three digits represent the fractional part. On power-up, all gain coefficients are set to 1,000 hex.

Attempts to calculate the gain coefficient for an output return `????`.

Example: Calculate and return gain coefficients for points 0, 1, 2, and 3 at address DF (hex):

```
>DFXF28cr
```

Response from unit:
```
A1400101E1019114835cr
```

	Hex	Decimal
Coefficient for point 0 =	1148	1.0800
Coefficient for point 1 =	1019	1.0060
Coefficient for point 2 =	101E	1.0073
Coefficient for point 3 =	1400	1.2500

Calculate and return gain coefficients for points 0, 12, and 14 at address AF (hex):
```
>AFX5001A5cr
```

Response from unit:
```
A130A????126EAFcr
```

	Hex	Decimal
Coefficient for point 0 =	126E	1.1520
Coefficient for point 12 =	????	(not an input)
Coefficient for point 14 =	130A	1.1900

Set Gain Coefficients — Driver Command 56

Analog — Optomux Command Y

Purpose: Sets the gain coefficients for specified input points.

Remarks: Before using this command, set offsets for points. See Set Offsets (page 117) or Calculate and Set Offsets (page 119).

This command uses gain values in the same form in which they are returned by the Calculate Gain Coefficients command (see page 121). If a specified point is an output, the command is ignored for that point.

Driver

Parameters:

Command:	56
Positions Array:	Points where gain coefficient values should be set.
Info Array:	Values in the Info array elements contain gain coefficient values to be written to the corresponding points.

Example: Set the gain coefficients for points 0 through 8:

```
For Index = 0 to 8
  OMUX_Pos(Index) = Index
  OMUX_Info(Index) = Offsets(Index)
Next Index

result = Send_Receive_Optomux(OMUX_Handle, 56, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `Y[positions][data]`

[data] contains four ASCII-hex digits for each point specified in the [positions] field. The values are placed in the data field in order starting with the highest numbered point and ending with the lowest numbered point. If the command does not include four digits of data for each bit specified in the [positions] field, a data field error will be returned.

Gain coefficients are placed in the [data] field in the same form as they are returned by the Calculate Gain Coefficients command (page 121).

Example: Set the following gain coefficients for points 0, 1, 2, and 3 at address DF (hex):

	Hex	Decimal
Coefficient for point 0 =	1148	1.0803
Coefficient for point 1 =	1019	1.0063

Coefficient for point 2 =	101E	1.0075
Coefficient for point 3 =	1400	1.2500

```
>DFY000F1400101E10191148EEcr
```

Set the following gain coefficients at address AF (hex):

	Hex	**Decimal**
Coefficient for point 0 =	126E	1.1520
Coefficient for point 12 =	1333	1.2000
Coefficient for point 14 =	1308	1.1900

```
>AFY500113081333126E1Acr
```

Calculate and Set Gain Coefficients — Driver Command 57

Analog — Optomux Command Z

Purpose: Calculates and sets gain coefficients for specified input points and then returns the calculated values to the host.

Remarks: Before using this command, set offsets for points. See Set Offsets (page 117) or Calculate and Set Offsets (page 119).

Because gain values are calculated using the current values of the inputs, use this command when the specified points are receiving the value you wish to consider full scale. This is usually done during system installation and calibration, when known inputs (full scale) can be applied to the points. The gain values returned to the host can be saved and used during system initialization (Set Gain Coefficients command, page 123).

Driver

Parameters:

Command:	57
Positions Array:	Points where gain coefficients should be calculated and set.
Info Array:	(Response) Values in the Info array elements indicate gain coefficient values for the corresponding points. Values returned are 10,000 times the actual gain coefficients (a value of 14,000 represents a gain coefficient of 1.40). This is the same format used by other gain commands.

Example: Calculate and set the gain coefficients for points 4 and 5, and save the values:

```
OMUX_Pos(0) = 4
OMUX_Pos(1) = 5
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 57, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)

Pos_4_Gain = OMUX_Info(4)
Pos_5_Gain = OMUX_Info(5)
```

Optomux Protocol

Format: `Z[positions]`

Four ASCII-hex digits are returned for each point specified. Values are returned in sequence from the highest to lowest point in the same form required by the Set Gain Coefficients command. A decimal point is assumed to be between the first digit and the last three. The first ASCII-hex digit represents the whole part of the coefficient and the last three digits represent the fractional part.

Attempts to calculate and set the gain coefficient for an output return `????`.

Example: Calculate and set gain coefficients for points 0, 1, 2, and 3 at address DF (hex):
`>DFZF2Acr`

Response from Optomux indicating that the following gain coefficients have been set:

	Hex	Decimal
Coefficient for point 0 =	1368	1.213
Coefficient for point 1 =	127E	1.156
Coefficient for point 2 =	11C2	1.110
Coefficient for point 3 =	1031	1.012

```
A103111C2127E13684Dcr
```

Calculate and set gain coefficients for points 0, 12, and 14 at address AF (hex):

```
>AFZ5001A7cr
```

Response from Optomux indicating that the following gain coefficients have been set.:

	Hex	Decimal
Coefficient for point 0 =	126E	1.152
Coefficient for point 12 =	1333	1.200
Coefficient for point 14 =	1308	1.190

```
A13081333126E74cr
```

15: Analog Waveform Commands

Set Output Waveform — Driver Command 43

Analog — Optomux Command R

Purpose: Starts a constant waveform at specified output points.

Remarks: Use Enhanced Output Waveform on page 131 instead of this command for greater flexibility in setting the period of the waveform.

Valid rates (period of a ramp waveform or one-half the period of a triangle or square wave) are shown below. These rates assume a full-scale change.

Driver	Optomux Protocol	Rate
(See page 130)	0	Disable waveform
0	1	2.18 minutes
1	2	3.28 minutes
2	3	4.37 minutes
3	4	5.46 minutes
4	5	6.56 minutes
5	6	7.65 minutes
6	7	8.74 minutes
7	8	1.09 minutes
8	9	32.8 seconds
9	A	21.8 seconds
10	B	16.4 seconds
11	C	13.1 seconds
12	D	10.9 seconds
13	E	9.4 seconds
14	F	8.2 seconds

Valid waveform types are:

Driver	Optomux Protocol	Waveform Type
0	0	No waveform
1	1	Triangle wave with positive initial slope
2	2	Ramp up—waveform terminates upon reaching the upper limit
3	3	Continuous ramp up
4	4	Square wave (50 % duty cycle)
5	5	Triangle wave with negative initial slope
6	6	Ramp down—waveform terminates at lower limit
7	7	Continuous ramp down

Driver

Parameters:

Command:	43
Positions Array:	Points where the waveform should be started.
Modifiers Array:	First element contains the waveform rate (see table on page 127). Second element contains waveform type (see table above).
Info Array:	First element contains high limit; second element contains low limit. If a lower value is passed in the first element, a -8 error is returned. Range limit is 0 to 255 (0 = zero scale; 255 = full scale).

Example: Start a continuous square wave at output points 4, 13, and 14. High limit is 170 (66.4% of full scale at 8.74 minutes for full-scale change x 2).

```
OMUX_Pos(0) = 4
OMUX_Pos(1) = 13
OMUX_Pos(2) = 14
OMUX_Pos(3) = -1
OMUX_Mod(0) = 6 'period is 8.74 mins
OMUX_Mod(1) = 4 'square wave
OMUX_Info(0) = 170 'high limit
OMUX_Info(1) = 0 'low limit

result = Send_Receive_Optomux(OMUX_Handle, 43, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `R[positions][modifiers][data]`

[modifiers] contains two ASCII-hex characters. The first character specifies the rate of the waveform (see table on page 127). The second character specifies the type of waveform (see table above).

NOTE: A "0" rate terminates the current waveform; no other data is required.

[data] contains four ASCII-hex digits. The first two represent the upper 8 bits of the 12-bit high limit of the waveform. The lower 4 bits are 0000. The second two characters represent the upper 8 bits of the 12-bit low limit of the waveform. The lower 4 bits are 0000.

The period of the entire ramp can be from 00 to FF. To calculate the actual period, determine what percentage of a full 0 to FF transition your limits represent. Multiply this fraction by the full-scale period. This result is the actual period of 1/2 of a triangle or square wave, or the entire period of a ramp.

Example: At Optomux 00, point 0, initiate a continuous ramp up with an upper limit of A00, a lower limit of 360, and a period of 1.09 minutes:

```
>00R000183A036B8cr
```

At Optomux AA, points 4, 5, and 6, initiate a square wave with an upper limit of AA0, a lower limit of 000, and a period of (AA / FF) x 8.74 min) = (0.66 x 8.74 min) = 11.6 min

```
>AAR007074AA00E8cr
```

Turn Off Existing Waveforms — Driver Command 44

Analog

Purpose: Turns off existing waveforms that were started with Set Output Waveforms (page 127).

Remarks: This command cancels waveforms that were started using command 43 (R). If the Enhanced Output Waveform command 50 (V) was used (page 131), then the Cancel Enhanced Waveforms command 51 (page 133) must be used to cancel.

Driver

Parameters:

Command:	44
Positions Array:	Points where the waveform should be cancelled.

Example: Cancel the waveform at point 2:

```
OMUX_Pos(0) = 2
OMUX_Pos(1) = -1

result = Send_Receive_Optomux(OMUX_Handle, 44, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

See Set Output Waveforms on page 127. A rate of 0 terminates the current waveform; no other data is required.

Enhanced Output Waveform — Driver Command 50

Analog — Optomux Command V

Purpose: Starts a constant waveform at specified output points.

Remarks: This command offers greater flexibility in setting the period of the waveform than the older command Set Output Waveform on page 127.

Valid waveform types are:

Driver	Optomux Protocol	Waveform Type
(see page 133)	0	Turn waveform off.
0	1	Triangle wave with positive initial slope
1	2	Ramp up—waveform terminates upon reaching the upper limit
2	3	Sawtooth, continuous ramp up
3	4	Square wave (50 % duty cycle)
4	5	Triangle wave with negative initial slope
5	6	Ramp down—waveform terminates at lower limit
6	7	Sawtooth, continuous ramp down

Driver

Parameters:

Command:	50
Positions Array:	Points where the waveform should be started.
Modifiers Array:	First element contains waveform type (see table above).
Info Array:	First element contains the high limit of the waveform (0–4095). Second element contains the low limit (also 0–4095). Third element contains the period of the waveform (1–32,767 in 100 ms units). This value represents half the period of a triangle or square waveform, or the entire period of a sawtooth or ramp waveform.

Example: Start a ramp up at output point 12 with a low limit of 10% of full scale and a high limit of 65% of full scale for a period of five seconds:

```
OMUX_Pos(0) = 12
OMUX_Pos(1) = -1
OMUX_Mod(0) = 2 'ramp up
OMUX_Info(0) = 4095 * 0.65 'high limit
OMUX_Info(1) = 4095 * 0.10 'low limit

result = Send_Receive_Optomux(OMUX_Handle, 50, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: `V[positions][modifiers][data]` [modifiers] is a single ASCII-hex character that specifies the type of waveform (see table above).

NOTE: A type of 0 terminates the current waveform.

[data] contains a 3-digit value representing the high limit of the waveform, a 3-digit value representing the low limit, and a 4-digit value representing the period of the waveform. The order is important.

High limit is three digits (000 = zero scale, FFF = full scale)

Low limit is three ASCII-hex digits (000 = zero scale, FFF = full scale)

Period is four ASCII-hex digits. The period can range from 0001 to 7FFF, each count representing 100 milliseconds (0.1 second to 54.61 minutes). The value represents the period of one-half a triangle or square wave. It is the entire period of sawtooths and ramps.

NOTE: The waveform is updated by Optomux every 50 milliseconds, therefore, waveforms with short periods may not provide a smooth output.

Example: At address FE (hex), output a triangle wave at point 3 with a high limit of FF0, a low limit of 00C, and a period of 12 seconds:

```
>FEV00081FF000C007808cr
```

At address 01 (hex), output a continuous ramp up (sawtooth) at points 2 and 5 with a high limit of F0A, a low limit of 00C, and a period of 12 seconds:

```
>01V00243F0A00C0078D9cr
```

Cancel Enhanced Waveforms — Driver Command 51

Analog

Purpose: Turns off existing waveforms that were started with Enhanced Output Waveform (page 131).

Remarks: This command cancels waveforms that were started using command 50 (V). If the Set Output Waveform command 43 (R) was used (page 127), then the Turn Off Existing Waveforms command 44 (page 130) must be used to cancel.

Driver

Parameters:

Command:	51
Positions Array:	Points where the waveform should be cancelled.

Example: Cancel the waveform at points 7 and 8:

```
OMUX_Pos(0) = 7
OMUX_Pos(1) = 8
OMUX_Pos(2) = -1

result = Send_Receive_Optomux(OMUX_Handle, 51, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

See Enhanced Output Waveforms on page 131. A type of 0 terminates the current waveform.

16: Revision Identification Command

Date of Firmware

Driver Command 80

Digital, Analog

Optomux Command ‘

Purpose: Identifies brain firmware revision by date of release.

Remarks: ‘ is the single quote under the tilde sign on the computer keyboard and is an ASCII 60 hex.

Response for a B1, B2, E1, or E2 is a string containing the date of firmware release in the format MM/DD/YY.

The response for a B3000 brain also contains the date.

Very old brain boards may not understand this command and return an error.

Driver

Parameters:

Command:	80

Example: Identify brain firmware by release date:

```
result = Send_Receive_Optomux(OMUX_Handle, 80, OMUX_Pos(0), _
OMUX_Mod(0), OMUX_Info(0), OMUX_TimeOut)
```

Optomux Protocol

Format: ‘

Example: Identify firmware revision at address 00 by date of release.

```
>00‘C0cr
```

Response from a B1, B2, E1, or E2:

```
A07/05/05*B9
```

Response from a B3000 (the portion in boldface is the date in the format MMDDYY):

A8116**090199**11050100300000B7

A: Troubleshooting Optomux

INTRODUCTION

Communication problems—If you are receiving serial communication errors, see the following section. If you are using Optomux over Ethernet, see the troubleshooting chapter in the *E1 and E2 User's Guide*.

Errors—For descriptions of common Optomux errors and how to respond to them, see page 140.

TROUBLESHOOTING SERIAL COMMUNICATIONS

Use this section to diagnose the most common causes of communication problems:

- 5 VDC power adjusted too low at brain boards
- Incorrect jumper settings
- Wiring problems

Power Tips

1. Ensure that the 5 VDC power supply wiring *connections* are secure.
2. Ensure the *voltage* is 5.00-5.20 VDC as measured on the brain board. Check voltage at all the Optomux brain boards by measuring across a capacitor on the brain board itself, not across the power supply terminals. For more information, see *Using Power Supplies with Opto 22 Systems* (form 1271).
3. Ensure the power supply has no ripple. If possible, inspect the power supply voltage with an oscilloscope, instead of using only a multimeter. Multimeters do not catch fast AC signals and transients.
4. Make certain that the power supplies are "floating"; that is, they do *not* connect the power supply common (5 VDC "return") to chassis/earth ground.
5. Make sure that the 5 VDC power supply common is *not* connected to the +/- 15 VDC common terminal on analog mounting racks; this can cause instable analog readings.

Jumper Tips

1. Make sure the jumpers are intact. The red jumpers provided with Optomux brain boards consist of a metal jumper on the inside covered by red plastic. Sometimes the metal portion of the jumper separates from the red plastic piece, causing incorrect jumper settings even though the jumpers appear to be correct.

   For example, if a jumper is removed but the metal internal piece stays attached to the brain board, the jumper will look like it is removed, but the brain will detect that it is installed.

On the other hand, the jumper can appear to be installed when it is not, if the red plastic part is installed but is missing the metal internal piece. In this case, the brain board will detect that the jumper is removed.

2. Make certain that the addresses are set correctly. No two brain boards on the same network can have the same address. If two or more brain boards on the same network have the same address, they will respond at the same time, causing a garbled response to be received at the host.
3. Make certain that the baud rates are set correctly. The host device and all brain boards must be operating at the same baud rate. A brain board set for the wrong baud rate will not respond.
4. Make certain the last Optomux brain board on the communications link has the correct termination jumpers installed (jumpers A0 and A6 for brain boards set up in multi-drop mode).
5. Make sure the host RS-422/485 device is configured to provide the correct termination and biasing.
   - The host receive pair (FO) must provide both termination and biasing, because this part of the link is RS-485.
   - The host transmit pair (TO) must provide termination and possibly biasing, depending on the interface device. Check the following to ensure it is set up correctly.
     - The RS-422 transmit pair must provide end-of-line termination resistor.
     - If the transmitter is actually an RS-485 transmitter, then biasing resistors are also necessary.

   NOTE: An RS-485 transmitter is disabled ("tri-stated") when not transmitting, so the transmit pair "floats" and may allow fluctuations to be interpreted as data or errors. However, an RS-422 transmitter remains enabled in either high or low state when not actively transmitting data, so fluctuations are not possible and biasing is not needed.

   NOTE: Some non-Opto 22 RS-422/485 interface products do not provide an option for termination and/or biasing. If the interface product does not provide the necessary options, then it will have to be replaced with one that does. Termination resistors can be applied externally, but biasing resistors typically cannot.

Communication Wiring Tips

1. Make sure to use a shielded, twisted pair RS-422/485 communication cable, and make sure the cable has at least three twisted pairs as detailed below. If you use a cable with four twisted pairs, you will have one extra pair.
   - two pairs for data (a transmit pair and a receive pair), and
   - one pair to use for the signal common
     - The signal common only requires one insulated conductor, but it is hard to find a cable with two pairs plus an extra insulated wire that is separate from the shield drain wire.
     - For the signal common, you can use both wires from the pair tied together or just one of the wires.
2. Ensure that the signal common is connected from one brain board to the next and also to the host RS-422/485 device. This connection must be made with an insulated wire, which is typically one wire in the overall communication cable.
3. Make sure that the COM terminal (signal common) on the brain boards is *not* connected to chassis/earth ground.
4. Ensure that the overall cable shield drain wire is connected to chassis/earth ground at one location only. Do *not* connect the shield drain wire to the COM (signal common) terminal on any brain board.
5. Make sure the communications cable is daisy-chained from brain to brain; no Ts are allowed under the RS-422/485 specification.

6. Make sure the communication pairs are not crossed. In some types of cable, one wire from each pair is the same color. This makes it very easy to get those wires mixed up. If you strip back the jacket and shield far enough, it will become clear which wire belongs to which pair.
7. Check the polarity of the + and – communication wires throughout each twisted pair. If you have one or more receive LEDs stuck on, this may be the cause. (Incorrect biasing might also cause LEDs to stick.)
8. Check the resistance of each twisted pair. One basic check of the communication wiring in a multi-drop system is to measure the resistance across the + and – lines of each twisted pair. There should be a 220 ohm resistor across the + and – wires of each data pair (transmit and receive) at each end of the cable. If both ends are terminated properly, you should see about 110 ohms, because there should be termination (220 ohms) at both ends. Make certain that in a multi-drop system, only the last physical brain board on the communication link has jumpers A0 and A6 installed.
9. If using an RS-232 to RS-422/485 converter at the host end of the system, make sure that CTS handshaking is disabled on the host. Otherwise, jumper RTS to CTS at the host end. If RTS/CTS handshaking is enabled (and not jumpered), the host will not transmit; no data will be allowed to go out the computer's RS-232 COM port.

Other Tips

1. **Diagnostic/Test Utilities**: Test the system using one of the Optomux utilities such as OptoScan, which can trap errors. Often, the errors received can help determine the nature of the problem. If OptoScan works properly and there are no errors, the cause of the problem might be the host software.
2. **Baud Rate**: Try running the system at a lower baud rate. Lower baud rates are more forgiving in a *noisy environment*.
3. **LED Indicators**: If you see receive LEDs blinking on the brain boards but not transmit LEDs, the cause of the problem could be incorrect address or baud rate settings on one or more brain boards. It can also be caused by wiring problems, low voltage, or noisy power.
4. **Computer Issues**: If you are using an ISA bus RS-422/485 card in the host computer, make sure that there are no I/O port or IRQ conflicts.
5. **Loopback test**: If the host device is a PC, it is possible to verify the operation of the RS-422/485 port by jumping TX+ to RX+, and TX- to RX- (these are TO and FO on Opto 22 brain boards) at the computer. A communication program such as Windows Terminal or HyperTerminal can be used to test communication with the port.

   Make sure the terminal test utility is set up with flow control set to None and that the local echo is turned off. Once started, anything typed on the keyboard should be displayed on the screen. If this does *not* happen, there may be a hardware or configuration problem with the communication port. Also remember that Opto 22 ISA bus serial adapter cards do not use standard Windows COM port settings (I/O port base address and IRQ) for COM3 and COM4. Make sure that the host software is using the correct I/O port base address and IRQ.
6. If you cannot solve the problem, contact Opto 22 Product Support. Contact information is on page 3.

COMMON OPTOMUX ERRORS

The following tables describe the most common Optomux errors. For a complete list of errors, see the `iolib_error_codes.h` file. If you need to convert these errors to older driver equivalents, see "Handling Errors in Migrated Applications" on page 13.

Errors Originating from the Brain

Driver Return Code	Driver Error Code	Protocol	Error Description and Troubleshooting Comments
-2126197482	-13034	00	O22_RESULT_OPTOMUX_PROTO_PUC_EXPECTED A command other than 0 (Power-Up Clear) was attempted after powerup or power failure. Command Ignored. Once the error is received, it is not necessary to execute a Power-Up Clear command. The next command will be executed normally. **IMPORTANT:** If this error message is received, it means that the Optomux unit has gone through its power-up sequence and has reset all characteristics to defaults. The unit needs to be reinitialized. See page 143. The brain contains a voltage watchdog circuit. If the 5 VDC logic supply voltage falls below the reset voltage, the brain will automatically reset. Use a high-quality DVM (Digital Volt Memory) or a high-precision oscilloscope to examine the brain's supply voltage. Power supply selection or design, poor power distribution techniques, or loose power wiring may cause spurious power-up-clear alerts. For more information, see *Using Power Supplies with Opto 22 Systems* (form 1271).
-2126197483	-13035	01	O22_RESULT_OPTOMUX_PROTO_UNDEF_COMMAND The brain cannot recognize the command. Most likely, this error is caused when an improper command is sent to the brain. It may also be created when using very old Optomux hardware.
-2126197484	-13036	02	O22_RESULT_OPTOMUX_PROTO_CHECKSUM_ERROR The checksum received by the brain board did not match the checksum calculated by the brain board. The command message was corrupted while being transferred over the wire. Data corruption can occur due to improper wiring, termination, and/or radio frequency or electromagnetic noise.
-2126197485	-13037	03	O22_RESULT_OPTOMUX_PROTO_INPUT_BUFFER_OVER The brain's input buffer was overrun: a command or a string of characters was more than 71 characters for an analog brain or 16 characters for a digital brain. This error is probably the result of a command sent to the wrong type of brain (for example, sending a write analog outputs command to a digital brain).
-2126197486	-13038	04	O22_RESULT_OPTOMUX_PROTO_NON_PRINTABLE_ASCII Only characters from 21 Hex to 7F Hex are permitted within Optomux messages. The command is ignored.
-2126197487	-13039	05	O22_RESULT_OPTOMUX_PROTO_DATA_FIELD_ERROR A data field is out of range or not enough data was sent. This error could also be caused by some kind of data corruption. Data corruption can occur due to improper wiring, termination, and/or spurious radio frequency or electromagnetic noise.
-2126197488	-13040	06	O22_RESULT_OPTOMUX_PROTO_WATCHDOG_TIMEOUT A communications link watchdog error has occurred. The brain performed a watchdog action due to the absence of communication. See the Set Watchdog commands in Chapter 5: Setup Commands.
-2126197489	-13041	07	O22_RESULT_OPTOMUX_PROTO_INVALID_LIMITS_SET Limits in the command are out of range.

Errors Reported by the Driver

Driver Return Code	Driver Error Code	Error Description and Troubleshooting Comments
-2124153252	-420	O22_RESULT_INVALID_HANDLE Most likely causes: either the call to Open_Optomux did not succeed or the handle used is closed. Inspect application code to see if the handle was not opened or was inadvertently closed.
-2128412673	-1	O22_RESULT_UNDEFINED_COMMAND The command requested is not a valid command number. Use constants or defines as provided by Opto 22 Visual Basic modules or C/C++ headers. See Chapter 2: Using the Optomux Driver.
-2128412674	-2	O22_RESULT_DVF_MISMATCH The response message's data validation failed (checksum error). The checksum of the message received by the Optomux driver does not match the checksum calculated by the driver. Data corruption can occur due to improper wiring, termination, and/or radio frequency or electro-magnetic noise.
-2128412678	-6	O22_RESULT_INVALID_DATA_FIELD A data field is out of range for the command. Inspect application code for an invalid data value.
-2128412681	-9	O22_RESULT_TIMEOUT The brain board did not respond within the specified time interval or did not respond at all. Timeouts occur if the timeout interval is too short or the brain board is not available. Possible causes of a brain board not being available include power/voltage problems, the brain board's jumper settings being incor-rect (address, baud rate, 2/4-pass mode, termination, etc.), or a problem with the communication wiring. For Ethernet, try pinging the IP address to make sure it is accessible.
-2128412712	-40	O22_RESULT_NOT_ENOUGH_DATA_RETURNED The response message is too short for the response parser to interpret what the response is. Data corruption may be the cause. Data corruption can occur due to improper wiring, termination, and/or radio frequency or electromagnetic noise.
-2128412719	-47	O22_RESULT_ALREADY_OPEN Application tried to open a handle to an object that is already open.
-2128412728	-56	O22_RESULT_INVALID_ADDRESS The address specified is out of range. Valid Optomux addresses range from zero to 255.
-2128412875	-203	O22_RESULT_DRIVER_NOT_FOUND The driver specified in the method string (the name before the first | char-acter) passed to the Open_Optomux command was not recognized or found. The driver name may be misspelled or contain a leading or trailing space. The driver dll may also have been deleted or moved. Driver name options are "serial" or "ip" and their corresponding dlls are iolibif_serial.dll and iolibif_ip.dll.
-2128425692	-13020	O22_RESULT_MISCONFIGURED_POINT A module configuration is inappropriate for the type of module. Compare application code and module configuration to make sure they match.
-2128425700	-13028	O22_RESULT_OBJECT_NOT_OPEN The object used is not open. Inspect the application code to verify whether the object is properly opened. If an Open is performed, there may be an error code the application isn't checking.

Driver Return Code	Driver Error Code	Error Description and Troubleshooting Comments
-2128425705	-13033	O22_RESULT_COMMAND_VALIDATION_FAILED The command echo does not match the command sent. The addressed brain board is probably in 2-pass mode. This error can only occur when the driver is configured for 4-pass mode. This mode was originally intended for diagnostic purposes, but it is generally not used even for that purpose. It is best to always use 2-pass mode.
-2132681120	-8608	O22_RESULT_PORT_INITIALIZE_FAILED Port initialization failed. Most likely to occur if the serial port is in use by another application.
-2133721147	-59	O22_RESULT_RECEIVE_PROBLEM A problem occurred while trying to receive data on a stream. This error is usually associated with IP sockets. It may indicate that the destination is no longer accepting or listening for requests, or that the connection-oriented session was abruptly closed by the remote host. Try pinging the IP address to make sure it is accessible.
-2133721499	-411	O22_RESULT_INVALID_SOCKET (Ethernet error) The driver could not create a socket.
-2133721500	-412	O22_RESULT_SOCKET_CONNECT_ERROR An attempt to connect to a host failed. The destination host may not be listening or accepting sessions on the specified port. Inspect the method parameters to make sure the proper host and port are selected.
-2133721503	-415	O22_RESULT_CANNOT_CREATE_FILE Could not open a communication device because it is already used by another application (or by another instance of this application).
-2133721502 (IP) or -2132672926 (serial)	-414	O22_RESULT_OUT_OF_HANDLES The driver ran out of handles. Creating new handles without closing others may cause this error. Also, a loop that opens handles may have improper loop limits.
-2133721528	-440	O22_RESULT_ERROR_ON_SOCKET_BIND (Ethernet error) Could not bind socket
-2133721530	-442	O22_RESULT_ERROR_ON_SOCKET_ACCEPT (Ethernet error) Could not accept socket
-2133734091	-13003	O22_RESULT_INVALID_METHOD_ARGUMENT An argument in the method string passed to the Open_Optomux function is incorrect.
-2133734092 (IP) or -2132685516 (serial)	-13004	O22_RESULT_DATA_SEND_ERROR The Optomux driver cannot send the message.
-2133734106	-13018	O22_RESULT_ERROR_ON_SOCKET_OPT_BROADCAST (Ethernet error) Failed to set the socket broadcast option.
-2141061173 (stream dll) or -2132672565 (serial dll)	-53	O22_RESULT_INVALID_STREAM_HANDLE The stream handle used for communication is closed. Inspect application code to see if the handle was not opened or was inadvertently closed.
-2141074122 (stream) or -2132685514 (serial dll)	-13002	O22_RESULT_BAD_STREAM_METHOD_STRING An attempt to configure a data stream failed. A stream method argument may be invalid, or the argument may be out of range. A streams driver may also be missing in the system (iolib_streams.dll).
-2141074125	-13005	O22_RESULT_INVALID_DRIVER_FUNCTION A driver function was not found. This error involves the iolib_streams.dll.

GENERAL TROUBLESHOOTING

Reinitializing the I/O Unit

An Optomux unit responds with a Power-Up Clear error whenever power was cycled. The purpose of this error is to warn you that the unit has lost its configuration and needs to be initialized.

When an Optomux unit is first powered up or after a Reset command, any prior initialization is lost, and the unit initializes itself to a set of default characteristics:

- Turnaround delay is set to 0.
- Message protocol is set according to jumper B10.
- Watchdog timer is disabled.
- All analog points are configured as inputs with gain set to 1.000 and offsets set to zero.
- All digital points are configured to function as inputs.
- All counters are set to 0.
- All time delays are set to 0.
- All latches are cleared and trigger set for OFF-to-ON.
- Pulse width measurement is set for ON pulses.

Always reinitialize the Optomux network when an Optomux unit goes through a reset condition. The reset condition can occur whenever a Reset command is sent OR when power is lost and returns. Note that a power-up condition can also be caused by a momentary dip on the + 5 VDC line, which causes the Optomux unit's processor to reset.

Q and A

I send a command to turn on output point 5 of a digital brain board, and output point 4 goes on.

Points are numbered from 0 to 15; therefore the fifth bit in the bitmask is for point 4. Following are the bitmask values for each point, in hex:

Point	Bitmask (Hex)
0	0001
1	0002
2	0004
3	0008
4	0010
5	0020
6	0040
7	0080

Point	Bitmask (Hex)
8	0100
9	0200
10	0400
11	0800
12	1000
13	2000
14	4000
15	8000

I send a command to activate several outputs. The brain board responds with no error, but none of my outputs come on.

If an output point does *not* turn on, check the following:

- Make sure you configured the output points correctly. The unit defaults to all points configured as inputs.
- If power went out and came back on, the unit will have lost its configuration.

- Make sure you are using output modules with a 5 VDC logic voltage (OAC5, ODC5, etc.). Sometimes modules with a 15 VDC or 24 VDC logic voltage (ODC15, OAC24, etc.) are used by mistake. The LED may turn on or be dim with the 15 VDC and 24 VDC modules, but there will be no output on the field side.
- If the output LED is on, but the load does not turn on, check the field voltage, wiring, and fuse. This could result in a field device *not* turning on.

When I read point 3 on an Optomux unit, I receive 0000 hex, which converts to a decimal -4096 value when I subtract the 1000 hex offset.

A -4096 decimal reading may indicate one of the following:

- You are reading a point where no input module is installed.
- You are reading a point on a thermocouple module that has no thermocouple installed or the thermocouple probe is open.
- An ICTD module has the ICTD wired in reverse.
- A 4–20 mA module is wired with reverse polarity.
- The field connections are made to the wrong terminals. Field connections vary with each module. Refer to the module data sheet for information on wiring each module. In general, field connections made to the terminals on the rack are made to the terminals labeled UPPER (closest to module), and if the module has terminals on the top of the module, connections would be made there. Analog racks have the module positions labeled from 1 to 10. These channels correspond to module positions 0 to 15 respectively.
- If all inputs on that brain board have the same -4096 reading, then check to make sure the unit has +15 and -15 VDC at the corresponding terminals (with reference to the terminal that is marked COMMON).

Does Opto 22 have any troubleshooting software for the Optomux system?

Yes. Opto 22 has OptoScan (oswin32.exe) and OmuxUser (omuxuser.exe) utilities that may be used to troubleshoot an Optomux system attached to a PC. These are Visual Basic 6 applications and are included with the Optomux Protocol Driver. Source code is included as an example of Opto 22's Optomux driver dlls. These applications include the ability to manually configure and poll a digital or analog brain board. The driver is available on Opto 22's website, www.opto22.com.

Serial Communication Questions and Answers

I send a command message to a brain board, and I get no response. However, the unit's LED flashes.

The receive LEDs on all the brain boards wired in a multi-drop mode should blink whenever a command is sent from the host. Only the unit at the address that matches the command message address should respond. When a brain board responds, the XMT (transmit) LED blinks. At high baud rates, and for messages with only a short response, the blink will be so brief that it may be missed visually.

If the host receives no response, and no blink of the XMT light is visible (even at slower baud rates), then check the following:

- Make sure the address in the command message matches the unit's address. Check the jumpers. A common mistake is a reversal of jumpers. All address jumpers (B0 through B7) installed corresponds to address 0, and all address jumpers (B0 through B7) removed corresponds to address 255. See the brain board user's guide for jumper settings.
- Make sure the baud rate is correct between the host and the unit. All brain boards on the same link should be configured to the same baud rate.
- Make sure you have a solid 5.0–5.2 VDC (as measured on the brain) powering the brain board. If the voltage is too low, the RCV light may flash, but the unit will not be able to respond. Measure the voltage

across one of the yellow capacitors on the brain board. See *Using Power Supplies with Opto 22 Systems* (form 1271).

- Check to make sure the communications link is wired with the correct polarity and that jumper group A is configured correctly.

I receive a large number of checksum errors when I send commands to Optomux brain boards.

Make sure that you are using twisted pair cable. See the cables recommended in the brain board user's guide. The RS-422/485 network is only reliable when the communications cable is twisted (+ and – lines of EACH pair twisted together), with at least 1/2 twist per inch. Sometimes, the wire is twisted but one of the connections of a pair is actually used as the mate to the opposite pair (– of one pair used as – of opposite pair). This cross-twist condition is usually due to jacketed twisted pair cables that have a wire of each pair with the same color code. In this case, strip the jacket back far enough to properly identify the individual pairs.

You should also check the Group A jumpers to make sure the termination and bias jumpers are configured correctly.

B: ASCII-Hex Tables

ASCII CHARACTER TABLE

Most Significant Byte

Least Significant Byte

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0 (1) ^@ NUL	16 ► ^P DLE	32 (1) space	48 0	64 @	80 P	96 `	112 p	128 Ç	144 É	160 á	176 ░	192 └	208 ╨	224 α	240 ≡
1	1 ☺ ^A SOH	17 ◄ ^Q DC1	33 !	49 1	65 A	81 Q	97 a	113 q	129 ü	145 æ	161 í	177 ▒	193 ┴	209 ╤	225 ß	241 ±
2	2 ☻ ^B STX	18 ↕ ^R DC2	34 “	50 2	66 B	82 R	98 b	114 r	130 é	146 Æ	162 ó	178 ▓	194 ┬	210 ╥	226 Γ	242 ≥
3	3 ♥ ^C ETX	19 ‼ ^S DC3	35 #	51 3	67 C	83 S	99 c	115 s	131 â	147 ô	163 ú	179 │	195 ├	211 ╙	227 π	243 ≤
4	4 ♦ ^D EOT	20 π ^T DC4	36 $	52 4	68 D	84 T	100 d	116 t	132 ä	148 ö	164 ñ	180 ┤	196 ─	212 ╘	228 Σ	244 ⌠
5	5 ♣ ^E ENQ	21 § ^U NAK	37 %	53 5	69 E	85 U	101 e	117 u	133 à	149 ò	165 Ñ	181 ╡	197 ┼	213 ╒	229 σ	245 ⌡
6	6 ♠ ^F ACK	22 ▬ ^V SYN	38 &	54 6	70 F	86 V	102 f	118 v	134 å	150 û	166 ª	182 ╢	198 ╞	214 ╓	230 µ	246 ÷
7	7 • ^G BEL	23 ↨ ^W ETB	39 ‘	55 7	71 G	87 W	103 g	119 w	135 ç	151 ù	167 º	183 ╖	199 ╟	215 ╫	231 τ	247 ≈
8	8 ◘ ^H BS	24 ↑ ^X CAN	40 (	56 8	72 H	88 X	104 h	120 x	136 ê	152 ÿ	168 ¿	184 ╕	200 ╚	216 ╪	232 Φ	248 °
9	9 ○ ^I HT	25 ↓ ^Y EM	41)	57 9	73 I	89 Y	105 i	121 y	137 ë	153 Ö	169 ⌐	185 ╣	201 ╔	217 ┘	233 Θ	249 •
A	10 ◙ ^J LF	26 → ^Z SUB	42 *	58 :	74 J	90 Z	106 j	122 z	138 è	154 Ü	170 ¬	186 ║	202 ╩	218 ┌	234 Ω	250 ·
B	11 ♂ ^K VT	27 ← ^[ESC	43 +	59 ;	75 K	91 [	107 k	123 {	139 ï	155 ¢	171 ½	187 ╗	203 ╦	219 █	235 δ	251 √
C	12 ♀ ^L FF	28 ∟ ^/ FS	44 ,	60 <	76 L	92 \	108 l	124 ¦	140 î	156 £	172 ¼	188 ╝	204 ╠	220 ▄	236 ∞	252 ⁿ
D	13 ¶ ^M CR	29 ↔ ^] GS	45 -	61 =	77 M	93]	109 m	125 }	141 ì	157 ¥	173 ¡	189 ╜	205 ═	221 ▌	237 ø	253 ²
E	14 ^N SO	30 ▲ ^^ RS	46 .	62 >	78 N	94 ^	110 n	126 ~	142 Ä	158 ₧	174 «	190 ╛	206 ╬	222 ▐	238 ε	254 ■
F	15 ¤ ^O SI	31 ▼ ^ US	47 /	63 ?	79 O	95 —	111 o	127 ⌂	143 Å	159 ƒ	175 »	191 ┐	207 ╧	223 ▀	239 ∩	255 (1)

HEX TO BINARY CONVERSION TABLE

Hex	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

C: Reading Negative Numbers and Temperature

INTRODUCTION

This appendix covers two issues that may arise when reading temperatures: converting negative numbers and converting temperature readings using thermocouple and RTD linearization.

Negative Numbers—Negative numbers may be returned when using Read Temperature commands and also a few other commands, such as "Calculate Offsets" on page 115. When negative numbers are returned, they are represented as a two's complement number. See "Converting Negative Numbers" on page 149.

Temperature Readings—Some Optomux commands are specifically designed for temperature measurement ("Read Temperature Inputs" on page 99 and "Read Average Temperature Inputs" on page 101). When you use a Read Temperature command, the brain board normally takes care of thermocouple linearization and returns temperature in degrees C. However, the commands are valid only when the temperature is within the nominal range for the module. If temperature is outside the module's nominal range (or if your software does not support the Read Temperature commands), then you will need to read counts from the module, linearize the counts, and convert them to temperature. More information and equations showing how to linearize and convert readings from temperature modules are in "Converting Temperature Readings" on page 150.

CONVERTING NEGATIVE NUMBERS

A negative number is returned as a two's complement number. It is a 16-bit ASCII-hex number with the most significant bit set to indicate that it is negative (for example, F956). To convert this hex number to a negative integer number, do the following:

1. Subtract the hex value from 0001 0000.
2. Convert to decimal.
3. Multiply by negative 1 (–1) to get the negative value.

For example:

```
0001  0000  hex
   –  F956  hex
-----------------
       6AA  hex
   =  1706  decimal
   x    –1
-----------------
   = –1706  decimal
```

To convert the value to temperature, divide by 16, since the first three hex digits are the whole part of the number, and the last hex digit is the fractional part:

	–1706	decimal
/	16	
=	–106.625	decimal

CONVERTING TEMPERATURE READINGS

The following sections use the term VALUE%. If you are using the Optomux protocol rather than the Optomux Driver, VALUE% represents the value returned by the brain board after it has been converted to decimal and the 4096 offset has been subtracted. If you are using the driver, the subtraction has already been done by the driver.

- `VALUE%` is < 0 when module is below zero scale.
- `VALUE%` is 0 when module is at zero scale.
- `VALUE%` is 4095 when module is at full scale.
- `VALUE%` is > 4095 when module is above full scale.

For specific equations and the constants to use in them, see the section for your module type.

ICTD Input Module—AD4

The ICTD Input module is used with an Opto 22 ICTD probe. To determine temperature in degrees Centigrade, use the following linear equation:

```
TEMP = (0.08262 * VALUE%) - 188.4
```

Type J Thermocouple—AD5, AD5T

The Type J thermocouple is a non-linear temperature transducer. AD5 and AD5T modules are linear and do not compensate for the non-linearity of the thermocouple. The modules have built-in cold junction compensation. Conversion to degrees Centigrade can be performed using piecewise linearization.

The formula for conversion is shown below:

```
TEMP = (VALUE% - A) * B + C
```

The values of A, B, and C are dependent upon `VALUE%` and can be obtained from the following table. (Accurate to ±0.2 °C from -20 to 700 °C.)

VALUE%		
	<	1
1	to	161
162	to	354
355	to	551
552	to	867
868	to	1766
1767	to	2546
2547	to	2895
2896	to	3191
3192	to	3464
3465	to	3743
3744	to	3901
	>	3901

A	B	C
104	0.1923076	20.15
0	0.1863354	0.10
161	0.1813472	30.19
354	0.1776649	65.10
551	0.1740506	100.15
867	0.1724137	155.10
1766	0.1730769	310.00
2546	0.1719197	445.05
2895	0.1689189	505.10
3191	0.1654411	555.10
3464	0.1612903	600.20
3743	0.1572327	645.17
3901	0.1546391	670.10

Temperature Range		
-20	to	-1
0	to	30
31	to	65
66	to	100
101	to	155
156	to	310
311	to	445
446	to	505
506	to	555
556	to	600
601	to	645
646	to	670
	>	670

Type K Thermocouple—AD8, AD8T

The Type K thermocouple is a non-linear temperature transducer. AD8 and AD8T modules are linear and do not compensate for the non-linearity of the thermocouple. The modules have built-in cold junction compensation. Conversion to degrees Centigrade can be performed using piecewise linearization. The formula for conversion is shown below:

```
TEMP = (VALUE% - A) * B + C
```

The values of A, B, and C are dependent upon `VALUE%` and can be obtained from the following table. (Accurate to ±0.4 °C from -100 to 1250 °C.)

VALUE%		
	<	96
96	to	198
199	to	348
349	to	578
579	to	909
910	to	1365
1366	to	1870
1871	to	3084
3085	to	3621
3622	to	4095
4096	to	4524
4525	to	4874
4875	to	5170
5171	to	5422
	>	5422

A	B	C
0	0.3167899	-99.6
95	0.2892961	-69.6
198	0.2675585	-39.8
348	0.2518454	0.3
478	0.2478090	57.9
909	0.2541073	140.2
1365	0.2456905	256.2
1870	0.2405867	380.1
3084	0.2456271	671.8
3621	0.2532714	803.8
4095	0.2611331	924.0
4524	0.2685714	1035.9
4874	0.2770270	1130.0
5170	0.2858277	1211.9
5422	0.2959973	1283.9

Temperature Range		
	<	-70
-70	to	-40
-39	to	0
1	to	58
59	to	140
141	to	256
257	to	380
381	to	672
673	to	804
805	to	924
925	to	1036
1037	to	1130
1131	to	1212
1213	to	1284
	>	1284

100 Ohm RTD Input Module—AD10T2

A resistive temperature detector (RTD) is a non-linear temperature transducer. The AD10T2 module is linear and does not compensate for the non-linearity of the RTD. Conversion to degrees Centigrade can be performed using piecewise linearization.

A 100 OHM Platinum RTD (ref DIN 43760, alpha = 0.00385) must be used with the AD10T2 in order for the conversion routine to be accurate.

The formula is shown below:

```
A = (VALUE% - A0) * A1 – A2
Temperature (degrees C) = A – (A3 * A^2) – (A4 * A) + A5
```

Use the constants shown in the following table:

For VALUE% of	Use these constants					
	A0	A1	A2	A3	A4	A5
<=2110	0	9.474182E-02	50	-0.000156	0.0156	-1.11
2111 to 4094	2111	0.1008065	100	-0.000156	0	248.45
4095 to 6218	4095	0.1082863	115	-0.00017	0	462.76
>6218	6219	0.118525	135	-0.000188	0	711.56

Accuracy:

- From –50 to 150 °C, ± 0.15 °C
- From 150 to 350 °C, ± 0.15 °C
- From 350 to 580 °C, ± 0.15 °C
- From 580 to 850 °C, ± 0.2 °C

Type R Thermocouple—AD17T

The Type R thermocouples are non-linear temperature transducers. AD17T modules are linear and do not compensate for the non-linearity of the thermocouples. The modules have built-in cold junction compensation. Conversion to degrees Centigrade can be performed using piecewise linearization.

The formula is shown below:

```
TV0 = (VALUE% * 2.43663)
Temperature (degrees C) = A0 + (A1 * TV0) + (A2 * (TV0^2)) + (A3 * (TV0^3))
+ (A4 * (TV0^4))
```

Use the constants shown in the following table:

For VALUE% of	Use these constants				
	A0	A1	A2	A3	A4
<=739	0	0.1625144	-2.045438E-05	2.540494E-09	-1.7679E-13
>739	46.67453	0.1117991	-2.565926E-06	5.347317E-11	0

Accuracy:

- From –50 to 0 °C, ± 12.1 °C
- From 0 to 350 °C, ± 3.6 °C
- From 350 to 1710 °C, ± 0.2 °C

Type S Thermocouple—AD17T

The Type S thermocouples are non-linear temperature transducers. AD17T modules are linear and do not compensate for the non-linearity of the thermocouples. The modules have built-in cold junction compensation. Conversion to degrees Centigrade can be performed using piecewise linearization.

The formula is shown below:

```
TV0 = (VALUE% * 2.43663)
Temperature (degrees C) = A0 + (A1 * TV0) + (A2 * (TV0^2)) + (A3 * (TV0^3))
+ (A4 * (TV0^4))
```

Use the constants shown in the following table:

For VALUE% of	Use these constants				
	A0	A1	A2	A3	A4
<=478	0	0.1641405	-2.024176E-05	2.784973E-09	-1.41721E-13
>478	30.1319	0.1215561	-2.752449E-06	6.475822E-11	0

Accuracy:

- From –50 to 0 °C, ± 10.2 °C
- From 0 to 400 °C, ± 2.7 °C
- From 400 to 1680 °C, ± 0.6 °C

Type T Thermocouple—AD18T

The Type T thermocouples are non-linear temperature transducers. AD18T modules are linear and do not compensate for the non-linearity of the thermocouples. The modules have built-in cold junction compensation. Conversion to degrees Centigrade can be performed using piecewise linearization.

The formula is shown below:

```
TV0 = (VALUE% * 3.951286) – 5602.92
Temperature (degrees C) = A0 + (A1 * TV0) + (A2 * (TV0^2)) + (A3 * (TV0^3))
+ (A4 * (TV0^4))
```

Use the constants shown in the following table:

For VALUE% of	Use these constants				
	A0	A1	A2	A3	A4
<=1418	0	2.383709E-02	-2.987884E-06	-7.194581E-10	-1.004194E-13
>1418	0	0.0256613	-6.195487E-07	2.218164E-11	-3.55009E-16

Accuracy:

- From –200 to -32 °C, ± 0.3 °C
- From -32 to 0 °C, ± 0.5 °C
- From 0 to 400 °C, ± 0.2 °C

Type E Thermocouple—AD19T

The Type E thermocouples are non-linear temperature transducers. AD19T modules are linear and do not compensate for the non-linearity of the thermocouples. The modules have built-in cold junction compensation. Conversion to degrees Centigrade can be performed using piecewise linearization.

The formula is shown below:

```
TV0 = (VALUE% * (36988.85 / 4095)) - 5236.65
Temperature (degrees C) = A0 + (A1 * TV0) + (A2 * (TV0^2)) + (A3 * (TV0^3))
+ (A4 * (TV0^4))
```

Use the constants shown in the following table:

For VALUE% of	Use these constants				
	A0	A1	A2	A3	A4
<=367	0	1.572665E-02	-1.210215E-06	-1.95778E-10	-1.66963E-14
367 to 3783	0	1.702253E-02	-2.209724E-07	5.480931E-12	-5.766989E-17
>3783	19.66945	1.420774E-02	-5.184451E-08	5.636137E-13	-1.564634E-18

Accuracy:

- From –150 to 0 °C, ± 0.3 °C
- From 0 to 1000 °C, ± 0.1 °C

Index