Test Solutions - Programming Manual

USB & RS232 <> SPI Converters

RS232 Series USB & RS232 <> SPI Converters

Mini-Circuits®
PO Box 350166, Brooklyn, NY 11235-0003
+1 718-934-4500 | testsolutions@minicircuits.com
www.minicircuits.com

Important Notice

Trademarks

Mini-Circuits
13 Neptune Avenue
Brooklyn, NY 11235, USA
Phone: +1-718-934-4500
Email: testsolutions@minicircuits.com
Web: www.minicircuits.com

1 - Overview

This Programming Manual is intended for customers wishing to create their own interface for Mini-Circuits' USB and RS232 controlled, SPI converters. The software package includes a GUI program, ActiveX and .NET DLL files, Linux support, project examples for third party software, and detailed user manuals. The latest package is available from:
https://www.minicircuits.com/softwaredownload/rs232_usb_spi.html

For details and specifications of individual models please see:
https://www.minicircuits.com/WebStore/PortableTestEquipment.html?sub_cat=USB%20/%20RS232%20/%20SPI%20Converters

Mini-Circuits has experience with a wide variety of environments including (but not limited to):

- Visual Basic®, Visual C#®, Visual C++®
- Delphi®
- Borland C++®
- CVI®
- LabVIEW®
- MATLAB®
- Python®
- Keysight VEE®

Files made available for download from the Mini-Circuits website are subject to Mini-Circuits' terms of use which are available on the website.

1.1 - Control Options

Communication with the device can use any of the following approaches:

1. Using the provided ActiveX or .Net API objects (DLL files) for USB control from a Windows operating system (see Operating in a Windows Environment via USB)
2. Using interrupt codes for USB control from Unix based operating systems (see Operating in a Linux Environment via USB)
3. Using RS232 serial communication (see Serial Control Using RS232 Communication)

2 - Operating in a Windows Environment via USB

When connected by USB, the computer will recognize the converter as a Human Interface Device (HID). In this mode of operation the DLL file provides the method of control. Alternatively, the device can be operated over a serial RS232 connection (see Serial Control Using RS232 Communication for details).

2.1 - The DLL (Dynamic Link Library) Concept

The Dynamic Link Library concept is Microsoft's implementation of the shared library concept in the Windows environment.

DLLs provide a mechanism for shared code and data, intended to allow a developer to distribute applications without requiring code to be re-linked or recompiled.

Mini-Circuits' software package provides DLL Objects designed to allow your own software application to interface with the functions of the SPI converter.

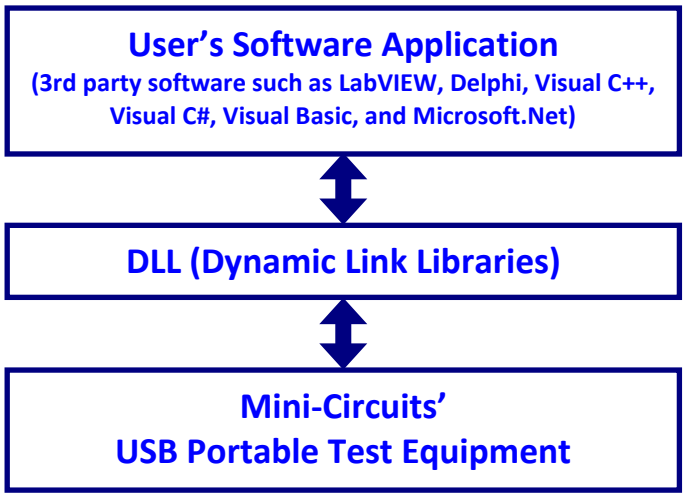

Fig 2.1-a: DLL Interface Concept

The software package provides two DLL files, the choice of which file to use is dictated by the user's operating system:

1. **ActiveX com object**
   Designed to be used in any programming environment that supports third party ActiveX COM (Component Object Model) compliant applications.
   The ActiveX file should be registered using RegSvr32 (see following sections for details).

2. **Microsoft.NET Class Library**
   A logical unit of functionality that runs under the control of the Microsoft.NET system.

2.1 (a) - ActiveX COM Object

ActiveX COM object DLL files are designed to be used with both 32-bit and 64-bit Windows operating systems. A 32-bit programming environment that is compatible with ActiveX is required. To develop 64-bit applications, the Microsoft.NET Class library should be used instead.

Supported Programming Environments

Mini-Circuits' devices have been tested in the following programming environments. This is not an exhaustive list and the DLL file is designed to operate in most environments that support ActiveX functionality. Please contact Mini-Circuits for support.

- Visual Studio® 6 (Visual C++ and Visual Basic)
- LabVIEW 8.0 or newer
- MATLAB 7 or newer
- Delphi
- Borland C++
- Agilent VEE
- Python

Installation

1. Copy the DLL file (MCL_RS232_USB_To_SPI.dll) to the correct directory:
   For 32-bit Windows operating systems this is C:\WINDOWS\System32
   For 64-bit Windows operating systems this is C:\WINDOWS\SysWOW64
2. Open the Command Prompt:
   a. For Windows XP® (see *Fig 2.1-b*):
      i. Select "All Programs" and then "Accessories" from the Start Menu
      ii. Click on "Command Prompt" to open
   b. For later versions of the Windows operating system you will need to have Administrator privileges in order to run the Command Prompt in "Elevated" mode (see *Fig 2.1-c* for Windows 7 and Windows 8):
      i. Open the Start Menu/Start Screen and type "Command Prompt"
      ii. Right-click on the shortcut for the Command Prompt
      iii. Select "Run as Administrator"
      iv. You may be prompted to enter the log in details for an Administrator account if the current user does not have Administrator privileges on the local PC
3. Use regsvr32 to register the DLL:
   For 32-bit Windows operating systems type (see Fig 2.1-d):

```
\WINDOWS\System32\Regsvr32 \WINDOWS\System32\MCL_RS232_USB_To_SPI.dll
```

   For 64-bit Windows operating systems type (see Fig 2.1-e):

```
\WINDOWS\SysWOW64\Regsvr32 \WINDOWS\SysWOW64\MCL_RS232_USB_To_SPI.dll
```

4. Hit enter to confirm and a message box will appear to advise of successful registration.

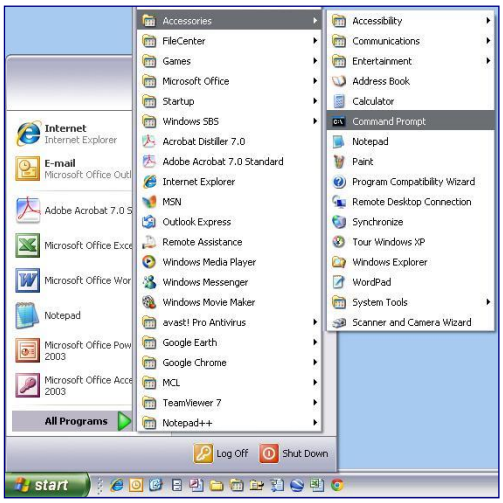

Fig 2.1-b: Opening the Command Prompt in Windows XP

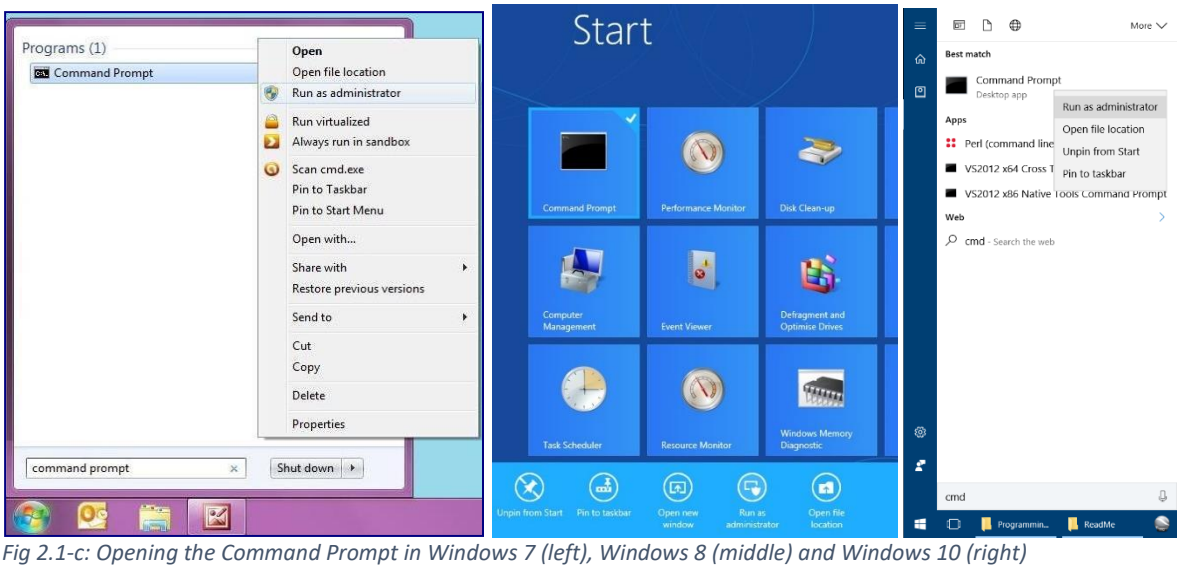

Fig 2.1-c: Opening the Command Prompt in Windows 7 (left), Windows 8 (middle) and Windows 10 (right)

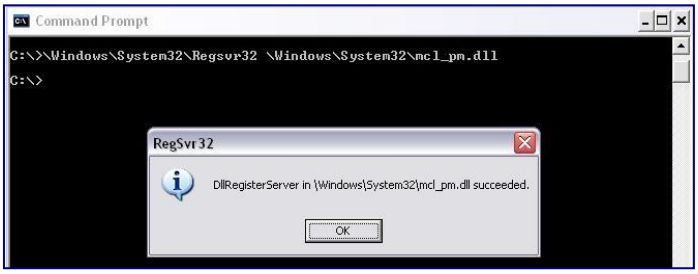

Fig 2.1-d: Registering the DLL in a 32-bit environment

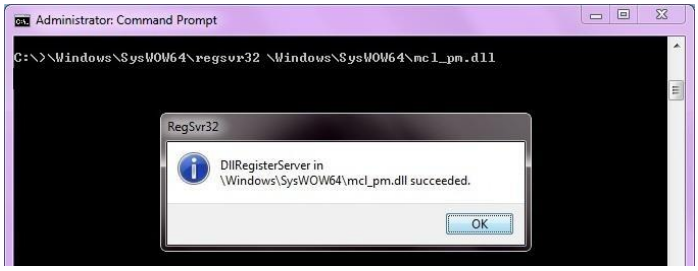

Fig 2.1-e: Registering the DLL in a 64-bit environment

2.1 (b) - Microsoft.NET Class Library

Microsoft.NET class libraries are designed to be used with both 32-bit and 64-bit Windows operating systems. To develop 64-bit applications the user must have both a 64-bit operating system and 64-bit programming environment. However, the Microsoft.NET class library is also compatible with 32-bit programming environments.

Supported Programming Environments

Mini-Circuits' devices have been tested in the following programming environments. This is not an exhaustive list and the DLL file is designed to operate in most environments that support Microsoft.NET functionality. Please contact Mini-Circuits for support.

- National Instruments CVI
- Microsoft.NET (Visual C++, Visual Basic.NET, Visual C# 2003 or newer)
- LabVIEW 2009 or newer
- MATLAB 2008 or newer
- Delphi
- Borland C++

Installation

1. Copy the DLL file (mcl_RS232_USB_To_SPI_64.dll) to the correct directory
    a. For 32 bit Windows operating systems this is C:\WINDOWS\System32
    b. For 64 bit Windows operating systems this is C:\WINDOWS\SysWOW64
2. **No registration is required**

2.2 - Referencing the DLL (Dynamic Linked Library)

The DLL file is installed in the host PC's system folders using the steps outlined above. Most programming environments will require a reference to be set to the DLL. Within the program, a new instance of the DLL's USB control class can be created for each converter to control. The details of this vary between programming environments and languages but Mini-Circuits can provide detailed support on request. In the following examples, MyPTE1 and MyPTE2 will be used as names of 2 declared converter objects.

2.2 (a) - Example Declarations using the ActiveX DLL (MCL_RS232_USB_To_SPI.dll)

Visual Basic

```
Public MyPTE1 As New MCL_RS232_USB_TO_SPI.USB_To_SPI
        ' Initialize new converter object, assign to MyPTE1
Public MyPTE2 As New MCL_RS232_USB_TO_SPI.USB_To_SPI
        ' Initialize new converter object, assign to MyPTE2
```

Visual C++

```
MCL_RS232_USB_To_SPI::USB_To_SPI ^MyPTE1 = gcnew
                                        _ MCL_RS232_USB_To_SPI::USB_To_SPI();
        // Initialize new converter instance, assign to MyPTE1
MCL_RS232_USB_To_SPI::USB_To_SPI ^MyPTE2 = gcnew
                                        _ MCL_RS232_USB_To_SPI::USB_To_SPI();
        // Initialize new converter instance, assign to MyPTE2
```

Visual C#

```
MCL_RS232_USB_To_SPI.USB_To_SPI MyPTE1 = new
                                        _ MCL_RS232_USB_To_SPI.USB_To_SPI();
        // Initialize new converter instance, assign to MyPTE1
MCL_RS232_USB_To_SPI.USB_To_SPI MyPTE2 = new
                                        _ MCL_RS232_USB_To_SPI.USB_To_SPI();
        // Initialize new converter instance, assign to MyPTE2
```

Matlab

```
MyPTE1 = actxserver('MCL_RS232_USB_TO_SPI.USB_To_SPI')
        % Initialize new converter instance, assign to MyPTE1
MyPTE2 = actxserver('MCL_RS232_USB_TO_SPI.USB_To_SPI')
        % Initialize new converter instance, assign to MyPTE2
```

2.2 (b) - Example Declarations using the .NET DLL (mcl_RS232_USB_To_SPI_64.dll)

Visual Basic

```
Public MyPTE1 As New MCL_RS232_USB_To_SPI.USB_To_SPI
        ' Initialize new object, assign to MyPTE1
Public MyPTE2 As New MCL_RS232_USB_To_SPI.USB_To_SPI
        ' Initialize new object, assign to MyPTE2
```

Visual C++

```
MCL RS232 USB To SPI::USB To SPI^MyPTE1 = gcnew MCL RS232 USB To SPI::USB To SPI();
        // Initialize new instance, assign to MyPTE1
MCL_RS232_USB_To_SPI::USB_To_SPI^MyPTE2 = gcnew MCL_RS232_USB_To_SPI::USB_To_SPI();
        // Initialize new instance, assign to MyPTE2
```

Visual C#

```
MCL_RS232_USB_To_SPI.USB_To_SPI MyPTE1 = new MCL_RS232_USB_To_SPI.USB_To_SPI();
        // Initialize new instance, assign to MyPTE1
MCL RS232 USB To SPI.USB To SPI MyPTE2 = new MCL RS232 USB To SPI.USB To SPI();
        // Initialize new instance, assign to MyPTE2
```

Matlab

```
MCL ATT=NET.addAssembly('C:\Windows\SysWOW64\MCL RS232 USB To SPI.dll')
MyPTE1=MCL_RS232_USB_To_SPI.USB_To_SPI       % Initialize new instance
MyPTE2=MCL_RS232_USB_To_SPI.USB_To_SPI       % Initialize new instance
```

2.3 - Summary of DLL Functions

The following functions are defined in both of the DLL files. Please see the following sections for a full description of their structure and implementation.

a) Short Connect (Option String SN)
b) Void Disconnect ()
c) Short Read_ModelName (String ModelName)
d) Short Read_SN (String SN)
e) Short Get_Available_SN_List (String SN_List)
f) Short Set_SPI_Mode (Short SPI_Mode)
g) Short Get_SPI_Mode ()
h) Short Send_SPI (Short NoOfBits, Int DataToSend)
i) Int Receive_SPI (Short NoOfBits)
j) Int Send_Receive_SPI (Short NoOfBits, Int DataToSend, Short CS, Short LE)
k) Int Set_PulseWidth(Int PulseWidth)
l) Short SetCS (Short BitVal)
m) Short SetLE (Short BitVal)
n) Short SetDO (Short BitVal)
o) Short SetCLK (Short BitVal)
p) Short GetCS ()
q) Short GetLE ()
r) Short GetDO ()
s) Short GetDI ()
t) Short GetCLK ()
u) Short GetStatus ()
v) Short GetExtFirmware (Short A0, Short A1, Short A2, Short A3, String Firmware)

2.4 - Detailed Description of DLL Functions

2.4 (a) - Connect to Converter

Declaration

```
Short Connect(Optional String SN)
```

Description

This function is called to initialize the connection to the USB to SPI converter. If multiple converters are connected to the same computer, then the serial number should be included, otherwise this can be omitted. The connection process can take a few milliseconds so it is recommended that the connection be made once at the beginning of the routine and left open until the converter is no longer needed. The converter should be disconnected on completion of the program using the Disconnect function.

Parameters

Data Type	Variable	Description
String	SN	Optional. The serial number of the USB to SPI converter. Can be omitted if only one converter is connected.

Return Values

Data Type	Value	Description
Short	0	No connection was possible
	1	Connection successfully established
	2	Connection already established (Connect has been called more than once). The converter will continue to operate normally.

Examples

Visual Basic

```
status = MyPTE1.Connect(SN)
```

Visual C++

```
status = MyPTE1->Connect(SN);
```

Visual C#

```
status = MyPTE1.Connect(SN);
```

Matlab

```
status = MyPTE1.Connect(SN)
```

See Also

Disconnect from Converter

2.4 (b) - Disconnect from Converter

Declaration

```
Void Disconnect()
```

Description

This function is called to close the connection to the converter. It is strongly recommended that this function is used prior to ending the program. Failure to do so may result in a connection problem with the device. Should this occur, shut down the program and unplug the converter from the computer, then reconnect the converter before attempting to start again.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
None		

Examples

Visual Basic

```
MyPTE1.Disconnect()
```

Visual C++

```
MyPTE1->Disconnect();
```

Visual C#

```
MyPTE1.Disconnect();
```

Matlab

```
MyPTE1.Disconnect
```

See Also

Connect to Converter

2.4 (c) - Read Model Name of Converter

Declaration

```
Short Read_ModelName(String ModelName)
```

Description

This function is called to determine the Mini-Circuits part number of the connected converter. The user passes a string variable which is updated with the part number.

Parameters

Data Type	Variable	Description
String	ModelName	Required. A string variable that will be updated with the Mini-Circuits part number for the converter.

Return Values

Data Type	Value	Description
Short	0	Command failed
	1	Command completed successfully

Examples

Visual Basic

```
If MyPTE1.Read_ModelName(ModelName) > 0 Then
        MsgBox ("The connected converter is " & ModelName)
                ' Display a message stating the model name
End If
```

Visual C++

```
if (MyPTE1->Read_ModelName(ModelName) > 0 )
{
        MessageBox::Show("The connected converter is " + ModelName);
                // Display a message stating the model name
}
```

Visual C#

```
if (MyPTE1.Read_ModelName(ref(ModelName)) > 0 )
{
        MessageBox.Show("The connected converter is " + ModelName);
                // Display a message stating the model name
}
```

Matlab

```
[status, ModelName]= MyPTE1.Read_ModelName(ModelName)
if status > 0
        h = msgbox('The connected attenuator is ', ModelName)
                % Display a message stating the model name
end
```

See Also

Read Serial Number of Converter

2.4 (d) - Read Serial Number of Converter

Declaration

```
Short Read_SN(String SN)
```

Description

This function is called to determine the serial number of the connected converter. The user passes a string variable which is updated with the serial number.

Parameters

Data Type	Variable	Description
String	ModelName	Required. String variable that will be updated with the serial number for the converter.

Return Values

Data Type	Value	Description
Short	0	Command failed
	1	Command completed successfully

Examples

Visual Basic

```
If MyPTE1.Read_SN(SN) > 0 Then
        MsgBox ("The connected converter is " & SN)
                'Display a message stating the serial number
End If
```

Visual C++

```
if (MyPTE1->Read_SN(SN) > 0 )
{
        MessageBox::Show("The connected converter is " + SN);
                // Display a message stating the serial number
}
```

Visual C#

```
if (MyPTE1.Read_SN(ref(SN)) > 0 )
{
        MessageBox.Show("The connected converter is " + SN);
                // Display a message stating the serial number
}
```

Matlab

```
[status, SN]= MyPTE1.Read_SN(SN)
if status > 0
        h = msgbox('The connected generator is ', SN)
                % Display a message stating the serial number
end
```

See Also

Read Model Name of Converter
Get List of Connected Serial Numbers

2.4 (e) - Get List of Connected Serial Numbers

Declaration

```
Short Get_Available_SN_List(String SN_List)
```

Description

This function takes a user defined variable and updates it with a list of serial numbers for all available (currently connected) converters.

Parameters

Data Type	Variable	Description
String	SN_List	Required. String variable which the function will update with a list of all available serial numbers, separated by a single space, for example “11301210001 11301210002 11301210003”.

Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

Examples

Visual Basic

```
If MyPTE1.Get_Available_SN_List(SN_List) > 0 Then
        array_SN() = Split(SN_List, " ")
                ' Split the list into an array of serial numbers
        For i As Integer = 0 To array_SN.Length - 1
                ' Loop through the array and use each serial number
        Next
End If
```

Visual C++

```
if (MyPTE1 ->Get_Available_SN_List(SN_List) > 0)
{
        // split the List into array of SN's
}
```

Visual C#

```
if (MyPTE1.Get_Available_SN_List(ref(SN_List)) > 0)
{
        // split the List into array of SN's
}
```

Matlab

```
[status, SN_List]= MyPTE1.Get_Available_SN_List(SN_List)
if status > 0
        % split the List into array of SN's
end
```

See Also

Get Device Serial Number

2.4 (f) - Set SPI Mode

Declaration

```
Short Set_SPI_Mode(Short SPI_Mode)
```

Description

This function sets the required SPI (Serial Peripheral Interface) mode to specify how the data stream is to be sampled. This ensures compatibility with the device that the converter is to communicate with. The default is SPI_Mode = 0.

Note: This function is only available with firmware B0 or later.

Parameters

Data Type	Variable	Description
Short	SPI_Mode	Required. The options for the SPI mode setting are: • 0 - IDLE=0 and SAMPLE_RISE (default) • 1 - IDLE=0 and SAMPLE_FALL • 2 - IDLE=1 and SAMPLE_FALL • 3 - IDLE=1 and SAMPLE_RISE

Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

Examples

Visual Basic

```
Status = MyPTE1.Set_SPI_Mode(1)
```

Visual C++

```
Status = MyPTE1->Set_SPI_Mode(1);
```

Visual C#

```
Status = MyPTE1.Set_SPI_Mode(1);
```

Matlab

```
Status = MyPTE1.Set_SPI_Mode(1)
```

See Also

Get SPI Mode

2.4 (g) - Get SPI Mode

Declaration

```
Short Get_SPI_Mode()
```

Description

This function returns the current SPI (Serial Peripheral Interface) mode to specify how the data stream is being sampled.

Note: This function is only available with firmware B0 or later.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
Short	0	IDLE=0 and SAMPLE_RISE
Short	1	IDLE=0 and SAMPLE_FALL
Short	2	IDLE=1 and SAMPLE_FALL
Short	3	IDLE=1 and SAMPLE_RISE

Examples

Visual Basic

```
Mode = MyPTE1.Get_SPI_Mode()
```

Visual C++

```
Mode = MyPTE1->Get_SPI_Mode();
```

Visual C#

```
Mode = MyPTE1.Get_SPI_Mode();
```

Matlab

```
Mode = MyPTE1.Get_SPI_Mode()
```

See Also

Set SPI Mode

2.4 (h) - Send SPI Data

Declaration

```
Short Send_SPI(Short NoOfBits, Int DataToSend)
```

Description

This function sends a user specified number of SPI (Serial Peripheral Interface) data bits. The maximum number of data bits that can be sent is 16. The binary SPI data string is sent as a decimal value from 0 to 65,535 (if all 16 data bits are used).

Parameters

Data Type	Variable	Description
Short	NoOfBits	Required. The number of data bits (1 to 16) to be sent.
Int	DataToSend	Required. A decimal value representing the binary data to be sent. Values from 0 to 65,535 are possible if the full 16 data bits are used (the MSB will be sent first).

Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

Examples

Visual Basic

```
Status = MyPTE1.Send_SPI(8, 172)
        ' Send SPI data 10101100 (8 bit binary string, decimal value 172)
```

Visual C++

```
status = MyPTE1->Send_SPI(8, 172);
        // Send SPI data 10101100 (8 bit binary string, decimal value 172)
```

Visual C#

```
status = MyPTE1.Send_SPI(8, 172);
        // Send SPI data 10101100 (8 bit binary string, decimal value 172)
```

Matlab

```
Status = MyPTE1.Send_SPI(8, 172)
        % Send SPI data 10101100 (8 bit binary string, decimal value 172)
```

See Also

Receive SPI Data
Send/Receive SPI Data

2.4 (i) - Receive SPI Data

Declaration

```
Int Receive_SPI(Short NoOfBits)
```

Description

This function returns the user specified number of SPI (Serial Peripheral Interface) data bits received by the converter. The maximum number of data bits is 16. The binary SPI data string is received as a decimal value from 0 to 65,535 (if all 16 data bits are used).

Parameters

Data Type	Variable	Description
Short	NoOfBits	Required. The number of data bits (1 to 16) to be read.

Return Values

Data Type	Value	Description
Int	-1	Command failed
Int	SPI_Data	The decimal value of the SPI data received. This can be interpreted as a binary string to determine the value of each data bit.

Examples

Visual Basic

```
Data = MyPTE1.Receive_SPI(8)
If Data > -1 Then
        ' Process SPI data (8 bits)
End If
```

Visual C++

```
Data = MyPTE1->Receive_SPI(8);
If (Data > -1) {
        // Process SPI data (8 bits)
}
```

Visual C#

```
Data = MyPTE1.Receive_SPI(8);
If (Data > -1) {
        // Process SPI data (8 bits)
}
```

Matlab

```
Data = MyPTE1.Send_SPI(8)
if Data > -1
        % Process SPI data (8 bits)
end
```

See Also

Send SPI Data
Send/Receive SPI Data

2.4 (j) - Send/Receive SPI Data

Declaration

```
Int Send_Receive_SPI(Short NoOfBits, Int DataToSend, Short CS,
                                                    Short LE)
```

Description

This function sends and receives a user specified number of SPI (Serial Peripheral Interface) data bits. The maximum number of data bits is 16. The binary SPI data string is communicated as a decimal value from 0 to 65,535 (if all 16 data bits are used).

Note: This function is only available with firmware B0 or later.

Parameters

Data Type	Variable	Description
Short	NoOfBits	Required. The number of data bits (1 to 16) to be sent.
Int	DataToSend	Required. A decimal value representing the binary data to be sent. Values from 0 to 65,535 are possible if the full 16 data bits are used (the MSB will be sent first).
Short	CS	Required. Specifies how the Chip Select (CS) pin should be handled during communication: • 0 (CS is not used) • 1 (Clear CS before sending SPI data and set CS after the last bit is sent • 2 (Set CS before sending SPI data and clear CS after the last bit is sent)
Short	LE	Required. Specifies how the Latch Enable (LE) pin should be handled during communication: • 0 (LE is not used) • 1 (Toggle LE high then low after sending SPI data) • 2 (Toggle LE low then high after sending SPI data)

Return Values

Data Type	Value	Description
Int	-1	Command failed
Int	SPI_Data	The decimal value of the SPI data received. This can be interpreted as a binary string to determine the value of each data bit.

Examples

Visual Basic

```
Data = MyPTE1.Send_Receive_SPI(8, 122, 0, 0)
        ' Send SPI data 10011001 (8 bit binary string, decimal value 153)
If Data > -1 Then
        ' Process received SPI data (8 bits)
End If
```

Visual C++

```
Data = MyPTE1->Send_Receive_SPI(8, 122, 0, 0);
        // Send SPI data 10011001 (8 bit binary string, decimal value 153)
If (Data > -1) {
        // Process received SPI data (8 bits)
}
```

Visual C#

```
Data = MyPTE1.Send_Receive_SPI(8, 122, 0, 0);
        // Send SPI data 10011001 (8 bit binary string, decimal value 153)
If (Data > -1) {
        // Process received SPI data (8 bits)
}
```

Matlab

```
Data = MyPTE1.Send_Receive_SPI(8, 122, 0, 0)
        % Send SPI data 10011001 (8 bit binary string, decimal value 153)
if Data > -1
        % Process received SPI data (8 bits)
end
```

See Also

Send SPI Data
Receive SPI Data

2.4 (k) - Set SPI Pulse Width

Declaration

```
Int Set_PulseWidth(Int PulseWidth)
```

Description

Sets the pulse width for SPI communication, from 0 to 255 µs. Setting 0 µs will ensure minimal pulse width, around 100 ns in practice.

Note: This function is only available with firmware B0 or later.

Parameters

Data Type	Variable	Description
Int	PulseWidth	The pulse width in microseconds from 0 to 255

Return Values

Data Type	Value	Description
Int	0	Command failed
	1	Command completed successfully

Examples

Visual Basic

```
Status = MyPTE1.Set_PulseWidth(10)
```

Visual C++

```
Status = MyPTE1->Set_PulseWidth(10);
```

Visual C#

```
Status = MyPTE1.Set_PulseWidth(10);
```

Matlab

```
Status = MyPTE1.Set_PulseWidth(10)
```

2.4 (I) - Set Chip Select (CS)

Declaration

```
Short Set_CS(Short BitVal)
```

Description

This function sets the Chip Select (CS) pin to logic high or low.

Note: This function is only available with firmware B0 or later.

Parameters

Data Type	Variable	Description
Short	BitVal	Required. The logic value to set, 0 (logic low) or 1 (logic high).

Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

Examples

Visual Basic

```
Status = MyPTE1.Set_CS(1)
```

Visual C++

```
Status = MyPTE1->Set_CS(1);
```

Visual C#

```
Status = MyPTE1.Set_CS(1);
```

Matlab

```
Status = MyPTE1.Set_CS(1)
```

See Also

Set Latch Enable (LE)
Set Data Out (DO)
Set Clock (CLK)
Get Chip Select (CS)

2.4 (m) - Set Latch Enable (LE)

Declaration

```
Short Set_LE(Short BitVal)
```

Description

This function sets the Latch Enable (LE) pin to logic high or low.

Parameters

Data Type	Variable	Description
Short	BitVal	Required. The logic value to set, 0 (logic low) or 1 (logic high).

Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

Examples

Visual Basic

```
Status = MyPTE1.Set_LE(1)
```

Visual C++

```
Status = MyPTE1->Set_LE(1);
```

Visual C#

```
Status = MyPTE1.Set_LE(1);
```

Matlab

```
Status = MyPTE1.Set_LE(1)
```

See Also

Set Chip Select (CS)
Set Data Out (DO)
Set Clock (CLK)
Get Latch Enable (LE)

2.4 (n) - Set Data Out (DO)

Declaration

```
Short Set_DO(Short BitVal)
```

Description

This function sets the Data Out (DO) pin to logic high or low.

Parameters

Data Type	Variable	Description
Short	BitVal	Required. The logic value to set, 0 (logic low) or 1 (logic high).

Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

Examples

Visual Basic

```
Status = MyPTE1.Set_DO(1)
```

Visual C++

```
Status = MyPTE1->Set_DO(1);
```

Visual C#

```
Status = MyPTE1.Set_DO(1);
```

Matlab

```
Status = MyPTE1.Set_DO(1)
```

See Also

Set Chip Select (CS)
Set Latch Enable (LE)
Set Clock (CLK)
Get Data Out (DO)

2.4 (o) - Set Clock (CLK)

Declaration

```
Short Set_CLK(Short BitVal)
```

Description

This function sets the Clock (CLK) pin to logic high or low.

Parameters

Data Type	Variable	Description
Short	BitVal	Required. The logic value to set, 0 (logic low) or 1 (logic high).

Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

Examples

Visual Basic

```
Status = MyPTE1.Set_CLK(1)
```

Visual C++

```
Status = MyPTE1->Set_CLK(1);
```

Visual C#

```
Status = MyPTE1.Set_CLK(1);
```

Matlab

```
Status = MyPTE1.Set_CLK(1)
```

See Also

Set Chip Select (CS)
Set Latch Enable (LE)
Set Data Out (DO)
Get Clock (CLK)

2.4 (p) - Get Chip Select (CS)

Declaration

```
Short Get_CS()
```

Description

This function returns the Chip Select (CS) pin logic state (high or low).

Note: This function is only available with firmware B0 or later.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
Short	0	Logic low
Short	1	Logic high

Examples

Visual Basic

```
Status = MyPTE1.Get_CS()
```

Visual C++

```
Status = MyPTE1->Get_CS();
```

Visual C#

```
Status = MyPTE1.Get_CS();
```

Matlab

```
Status = MyPTE1.Get_CS()
```

See Also

Set Chip Select (CS)
Get Latch Enable (LE)
Get Data Out (DO)
Get Data In (DI)
Get Clock (CLK)

2.4 (q) - Get Latch Enable (LE)

Declaration

```
Short Get_LE()
```

Description

This function returns the Latch Enable (LE) pin logic state (high or low).

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
Short	0	Logic low
Short	1	Logic high

Examples

Visual Basic
```
Status = MyPTE1.Get_LE()
```
Visual C++
```
Status = MyPTE1->Get_LE();
```
Visual C#
```
Status = MyPTE1.Get_LE();
```
Matlab
```
Status = MyPTE1.Get_LE()
```

See Also

Set Latch Enable (LE)
Get Chip Select (CS)
Get Data Out (DO)
Get Data In (DI)
Get Clock (CLK)

2.4 (r) - Get Data Out (DO)

Declaration

```
Short Get_DO()
```

Description

This function returns the Data Out (DO) pin logic state (high or low).

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
Short	0	Logic low
Short	1	Logic high

Examples

Visual Basic
```
Status = MyPTE1.Get_DO()
```
Visual C++
```
Status = MyPTE1->Get_DO();
```
Visual C#
```
Status = MyPTE1.Get_DO();
```
Matlab
```
Status = MyPTE1.Get_DO()
```

See Also

Set Data Out (DO)
Get Chip Select (CS)
Get Latch Enable (LE)
Get Data In (DI)
Get Clock (CLK)

2.4 (s) - Get Data In (DI)

Declaration

```
Short Get_DI()
```

Description

This function returns the Data In (DI) pin logic state (high or low).

Note: This function is only available with firmware B0 or later.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
Short	0	Logic low
Short	1	Logic high

Examples

Visual Basic

```
Status = MyPTE1.Get_DI()
```

Visual C++

```
Status = MyPTE1->Get_DI();
```

Visual C#

```
Status = MyPTE1.Get_DI();
```

Matlab

```
Status = MyPTE1.Get_DI()
```

See Also

Get Chip Select (CS)
Get Latch Enable (LE)
Get Data Out (DO)
Get Clock (CLK)

2.4 (t) - Get Clock (CLK)

Declaration

```
Short Get_CLK()
```

Description

This function returns the Clock (CLK) pin logic state (high or low).

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
Short	0	Logic low
Short	1	Logic high

Examples

Visual Basic

```
Status = MyPTE1.Get_CLK()
```

Visual C++

```
Status = MyPTE1->Get_CLK();
```

Visual C#

```
Status = MyPTE1.Get_CLK();
```

Matlab

```
Status = MyPTE1.Get_CLK()
```

See Also

Set Clock (CLK)
Get Chip Select (CS)
Get Latch Enable (LE)
Get Data Out (DO)
Get Data In (DI)

2.4 (u) - Get Status

Declaration

```
Short Get_Status()
```

Description

This function checks whether the USB connection to the converter is still active.

Parameters

Data Type	Variable	Description
None		

Return Values

Data Type	Value	Description
Short	0	No connection
Short	>0	USB connection is active

Examples

Visual Basic

```
Status = MyPTE1.Get_Status
```

Visual C++

```
Status= MyPTE1->Get_Status();
```

Visual C#

```
Status= MyPTE1.Get_Status();
```

Matlab

```
Status= MyPTE1.Get_Status
```

2.4 (v) - GetExtFirmware

Declaration

```
Short GetExtFirmware(Short A0, Short A1, Short A2, Short A3,
                                                    String Frimware)
```

Description

This function returns the internal firmware version of the converter along with four reserved variables (for factory use).

Parameters

Data Type	Variable	Description
Short	A0	Required. User defined variable for factory use only.
Short	A1	Required. User defined variable for factory use only.
Short	A2	Required. User defined variable for factory use only.
Short	A3	Required. User defined variable for factory use only.
String	Firmware	Required. User defined variable which will be updated with the current firmware version, for example “B3”.

Return Values

Data Type	Value	Description
Short	0	Command failed
Short	1	Command completed successfully

Examples

Visual Basic

```
If MyPTE1.GetExtFirmware(A0, A1, A2, A3, Firmware) > 0 Then
        MsgBox ("Firmware version is " & Firmware)
End If
```

Visual C++

```
if (MyPTE1->GetExtFirmware(A0, A1, A2, A3, Firmware) > 0 )
{
        MessageBox::Show("Firmware version is " + Firmware);
}
```

Visual C#

```
if (MyPTE1.GetExtFirmware(ref(A0, A1, A2, A3, Firmware)) > 0 )
{
        MessageBox.Show("Firmware version is " + Firmware);
}
```

Matlab

```
[status, A0, A1, A2, Firmware]=MyPTE1.GetExtFirmware(A0, A1, A2, Firmware)
if status > 0
        h = msgbox('Firmware version is ', Firmware)
end
```

3 - Operating in a Linux Environment via USB

For USB control, the computer will recognize the converter as a Human Interface Device (HID) when the USB connection is made. In this mode of operation the following command codes can be used.

For RS232 control, please see Serial Control Using RS232 Communication.

To open a connection to the USB to SPI converter, the Vendor ID and Product ID are required:

- Mini-Circuits Vendor ID: 0x20CE
- Converter Product ID: 0x25

Communication with the converter is carried out by way of USB Interrupt. The transmitted and received buffer sizes are 64 Bytes each:

- Transmit Array = [Byte 0][Byte1][Byte2]…[Byte 63]
- Returned Array = [Byte 0][Byte1][Byte2]…[Byte 63]

In most cases, the full 64 byte buffer size is not needed so any unused bytes become “don’t care” bytes; they can take on any value without affecting the operation of the converter.

Worked examples can be found in the Programming Examples & Troubleshooting Guide, downloadable from the Mini-Circuits website. The examples use the libhid and libusb libraries to interface with the programmable attenuator as a USB HID (Human Interface Device).

3.1 - Summary of Commands

The commands that can be sent to the converter are summarized in the table below and detailed on the following pages.

	Description	Command Code (Byte 0)	Comments
a	Get Device Model Name	40	
b	Get Device Serial Number	41	
c	Set SPI Mode	78	
d	Get SPI Mode	79	
e	Send SPI Data	65	
f	Receive SPI Data	66	
g	Send & Receive SPI Data	67	
h	Set Pulse Width	8	
i	Set Data State	68 69 71 72	CS (Chip Select) LE (Latch Enable) DO (Data Out) CLK (Clock)
j	Get Data State	73 74 75 76 77	CS (Chip Select) LE (Latch Enable) DI (Data In) DO (Data Out) CLK (Clock)
k	Get Firmware Revision	99	

3.2 - Detailed Description of Commands

3.2 (a) - Get Device Model Name

Description

This function determines the Mini-Circuits part number of the connected converter.

Transmit Array

Send code 40 in BYTE0 of the transmit array. BYTE1 through to BYTE63 are don't care bytes and can be any value.

Byte	Byte 0
Description	Code
Value	40

Returned Array

The model name is represented as a series of ASCII characters in the returned array, starting from BYTE1. The final ASCII character is contained in the byte immediately preceding the first zero value byte. All subsequent bytes up to BYTE63 are "don't care" bytes and could be any value.

Byte	Byte 0	Byte 1	Byte 2	…	Byte (N-1)	Byte N
Description	Code	First Char	Second Char	…	Last Char	End Marker
Value	40	ASCII	ASCII	…	ASCII	0

Example

The following array would be returned for Mini-Circuits' RS232/USB-SPI converter. See Appendix A for conversions between decimal, binary and ASCII characters.

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Description	Code	Char 1	Char 2	Char 3	Char 4	Char 5	Char 6
Value	40	82	83	50	51	50	47
ASCII Character	N/A	R	S	2	3	2	/

Byte	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13
Description	Char 7	Char 8	Char 9	Char 10	Char 11	Char 12	Char 13
Value	85	83	66	45	83	80	73
ASCII Character	U	S	B	-	S	P	I

Byte	Byte 14
Description	End Marker
Value	0
ASCII Character	N/A

See Also

Get Device Serial Number

3.2 (b) - Get Device Serial Number

Description

This function determines the serial number of the connected converter.

Transmit Array

Send code 41 in BYTE0 of the transmit array. BYTE1 through to BYTE63 are “don’t care” bytes and can be any value.

Byte	Byte 0
Description	Code
Value	41

Returned Array

The serial number is represented as a series of ASCII characters in the returned array, starting from BYTE1. The final ASCII character is contained in the byte immediately preceding the first zero value byte. All subsequent bytes up to BYTE63 are “don’t care” bytes and could be any value.

Byte	Byte 0	Byte 1	Byte 2	…	Byte (N-1)	Byte N
Description	Code	First Char	Second Char	…	Last Char	End Marker
Value	41	ASCII	ASCII	…	ASCII	0

Example

The following example indicates that the current converter has serial number 11301210001. See Appendix A for conversions between decimal, binary and ASCII characters.

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Description	Code	Char 1	Char 2	Char 3	Char 4	Char 5	Char 6
Value	41	49	49	51	48	49	50
ASCII Character	N/A	1	1	3	0	1	2

Byte	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12
Description	Char 7	Char 8	Char 9	Char 10	Char 11	End Marker
Value	49	48	48	48	49	0
ASCII Character	1	0	0	0	1	N/A

See Also

Get Device Model Name

3.2 (c) - Set SPI Mode

Description

This function sets the required SPI (Serial Peripheral Interface) mode to specify how the data stream is to be sampled. This ensures compatibility with the device that the converter is to communicate with. The 4 available modes are:

- 0 - IDLE=0 and SAMPLE_RISE (default)
- 1 - IDLE=0 and SAMPLE_FALL
- 2 - IDLE=1 and SAMPLE_FALL
- 3 - IDLE=1 and SAMPLE_RISE

Transmit Array

The transmit array is made up of the following bytes:

- BYTE0
  - Code 78
- BYTE1
  - The mode to set (0 to 3)
- BYTE2 to BYTE63
  - Can be any value (“don’t care” bytes)

Byte	Byte 0	Byte 1
Description	Code	Mode
Value	78	0 to 3

Returned Array

Byte	Byte 0
Description	Code
Value	78

Example

The following transmit array would set SPI mode 3 (IDLE=1 and SAMPLE_RISE)

Byte	Byte 0	Byte 1
Description	Code	Mode
Value	78	3

See Also

Get SPI Mode

3.2 (d) - Get SPI Mode

Description

This function gets the current SPI (Serial Peripheral Interface) mode. This specifies how the data stream is to be sampled, ensuring compatibility with the device that the converter is to communicate with. The 4 available modes are:

- 0 - IDLE=0 and SAMPLE_RISE (default)
- 1 - IDLE=0 and SAMPLE_FALL
- 2 - IDLE=1 and SAMPLE_FALL
- 3 - IDLE=1 and SAMPLE_RISE

Transmit Array

Send code 79 in BYTE0 of the transmit array. BYTE1 to BYTE63 are "don't care" bytes and can be any value.

Byte	Byte 0
Description	Code
Value	79

Returned Array

The returned array is made up of the following bytes:

- BYTE0
  - Code 79
- BYTE1
  - The mode setting (0 to 3)
- BYTE2 to BYTE63
  - Could be any value ("don't care" bytes)

Byte	Byte 0	Byte 1
Description	Code	Mode
Value	79	0 to 3

Example

The following array would be returned to indicate that the converter is in the default SPI mode (0):

Byte	Byte 0	Byte 1
Description	Code	Mode
Value	79	0

See Also

Set Data State

3.2 (e) - Send SPI Data

Description

This function sends a SPI (Serial Peripheral Interface) data string.

Transmit Array

The transmit array is made up of the following bytes:

- BYTE0
  - Code 65
- BYTE1
  - The number of data bits (N) to send (1 to 16)
- BYTE2 to BYTE3
  - Decimal value of the SPI data to be sent, split into MSB (BYTE2) and LSB (BYTE3)
  - BYTE2 = INTEGER VALUE (DATA / 256)
  - BYTE3 = INTEGER VALUE (DATA - (BYTE2 * 256))
- BYTE4 to BYTE63
  - Can be any value (“don’t care” bytes)

Byte	Byte 0	Byte 1	Byte 2	Byte 3
Description	Code	N	Data MSB	Data LSB
Value	65	1-16	0-255	0-255

Returned Array

Byte	Byte 0
Description	Code
Value	65

Example

To send the 8 bit SPI data string “10010010”:

- Decimal value = 146
- BYTE2 = INT (146 / 256)
  = 0
- BYTE3 = INT (146 – (0 * 256))
  = 146

The transmit array is therefore:

Byte	Byte 0	Byte 1	Byte 2	Byte 3
Description	Code	N	Data MSB	Data LSB
Value	65	8	0	146

See Also

Receive SPI Data
Send and Receive SPI Data

3.2 (f) - Receive SPI Data

Description

This function is used to receive a SPI (Serial Peripheral Interface) data string.

Note: This function is only available with firmware B0 or later.

Transmit Array

The transmit array is made up of the following bytes:

- BYTE0
  - Code 66
- BYTE1
  - The number of data bits (N) to send (1 to 16)
- BYTE2 to BYTE63
  - Can be any value (“don’t care” bytes)

Byte	Byte 0	Byte 1
Description	Code	N
Value	66	1-16

Returned Array

The returned array is made up of the following bytes:

- BYTE0
  - Code 66
- BYTE1 to BYTE2
  - Decimal value of the SPI data received, split into MSB (BYTE1) and LSB (BYTE2)
  - DATA = BYTE1 * 256 + BYTE2
- BYTE3 to BYTE63
  - Could be any value (“don’t care” bytes)

Byte	Byte 0	Byte 1	Byte 2
Description	Code	Data MSB	Data LSB
Value	66	0-255	0-255

Example

To read 16 bits of data send:

Byte	Byte 0	Byte 1
Description	Code	N
Value	66	16

The following returned array indicates that the data value received is 33,820:

Byte	Byte 0	Byte 2	Byte 2
Description	Code	Data MSB	Data LSB
Value	66	132	28

Calculate the data value:

- Data = (132 * 256) + 28
  = 33,820

The binary value of 33,820 indicates that the received data string was:

- 1000 0000 0001 1100

See Also

Send SPI Data
Send and Receive SPI Data

3.2 (g) - Send and Receive SPI Data

Description

This function sends and receives a user specified number of SPI (Serial Peripheral Interface) data bits. The maximum number of data bits is 16. The binary SPI data string is communicated as a decimal value from 0 to 65,535 (if all 16 data bits are used).

Note: This function is only available with firmware B0 or later.

Transmit Array

The transmit array is made up of the following bytes:

- BYTE0
  - Code 67
- BYTE1
  - The number of data bits (N) to send (1 to 16)
- BYTE2 to BYTE3
  - Decimal value of the SPI data to be sent, split into MSB (BYTE2) and LSB (BYTE3)
  - BYTE2 = INTEGER VALUE (DATA / 256)
  - BYTE3 = INTEGER VALUE (DATA - (BYTE2 * 256))
- BYTE4
  - Chip Select (CS) setting from 0 to 2:
    - 0 - CS is not used
    - 1 - Clear CS before sending SPI data and set CS after last bit is sent
    - 2 - Set CS before sending SPI data and clear CS after last bit is sent
- BYTE5
  - Latch Enable (LE) setting from 0 to 2:
    - 0 - LE is not used
    - 1 - Toggle LE high then low after sending SPI data
    - 2 - Toggle LE low then high after sending SPI data
- BYTE6 to BYTE63
  - Can be any value (“don’t care” bytes)

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Description	Code	N	Data MSB	Data LSB	CS	LE
Value	67	1-16	0-255	0-255	0-2	0-2

Returned Array

The returned array is made up of the following bytes:

- BYTE0
  - Code 67
- BYTE1 to BYTE2
  - Decimal value of the SPI data received, split into MSB (BYTE1) and LSB (BYTE2)
  - DATA = BYTE1 * 256 + BYTE2
- BYTE3 to BYTE63
  - Could be any value ("don't care" bytes)

Byte	Byte 0	Byte 1	Byte 2
Description	Code	Data MSB	Data LSB
Value	67	0-255	0-255

Example

To set CS to logic low, send the 8 bit data string 00111000, set CS to logic high, then read the received SPI data string (containing 11000011):

- N = 8 (number of data bits)
- Data = 56 (decimal value of 00111000)
- Data MSB
  = INT (56 / 256)
  = 0
- Data LSB
  = INT (56 – (0 * 256)
  = 56
- CS = 1 (set CS low before sending and high after sending)
- LE = 0 (not needed in this application)

The complete transmit array is therefore:

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
Description	Code	N	Data MSB	Data LSB	CS	LE
Value	67	8	0	56	1	0

The following returned array indicates that the data value received is 175:

Byte	Byte 0	Byte 1	Byte 2
Description	Code	Data MSB	Data LSB
Value	67	0	175

Calculate the data value:

- Data = (0 * 256) + 175
  = 175

This binary value of 175 indicates that the received data string was 1100 0011.

See Also

Send SPI Data
Receive SPI Data

3.2 (h) - Set SPI Pulse Width

Description

Sets the pulse width for SPI communication, from 0 to 255 µs. Setting 0 µs will ensure minimal pulse width, around 100 ns in practice.

Transmit Array

The transmit array is made up of the following bytes:

- BYTE0
  - Code 8
- BYTE1
  - The pulse width to set (0 to 255)
- BYTE2 to BYTE63
  - Can be any value ("don't care" bytes)

Byte	Byte 0	Byte 1
Description	Code	Mode
Value	8	0 to 255

Returned Array

Byte	Byte 0
Description	Code
Value	8

Example

The following transmit array would set the pulse width to 10 µs

Byte	Byte 0	Byte 1
Description	Code	Mode
Value	8	10

3.2 (i) - Set Data State

Description

This function sets one of the Chip Select (CS), Latch Enable (LE), Data Out (DO), or Clock (CLK) data output pins to logic low (0) or logic high (1).

Transmit Array

The transmit array is made up of the following bytes:

- BYTE0
  - Code 68 for CS
  - Code 69 for LE
  - Code 71 for DO
  - Code 72 for CLK
- BYTE1
  - The logic state to set (1 for high or 0 for low)
- BYTE2 to BYTE63
  - Can be any value (“don’t care” bytes)

Byte	Byte 0	Byte 1
Description	Code	Logic State
Value	68-72	0-1

Returned Array

The returned array will repeat the code in BYTE0 on successful completion. BYTE1 to BYTE63 are “don’t care” bytes and could be any value.

Byte	Byte 0
Description	Code
Value	68-72

Example

Send the following transmit array to set the LE data pin to logic high:

Byte	Byte 0	Byte 1
Description	Code	Logic State
Value	69	1

See Also

Get Data State

3.2 (j) - Get Data State

Description

This function gets the logic state of one of the Chip Select (CS), Latch Enable (LE), Data In (DI), Data Out (DO), or Clock (CLK) data pins.

Note: This function is only available with firmware B0 or later.

Transmit Array

The transmit array is made up of the following bytes:

- BYTE0
  - Code 73 for CS
  - Code 74 for LE
  - Code 75 for DI
  - Code 76 for DO
  - Code 77 for CLK
- BYTE1 to BYTE63
  - Can be any value (“don’t care” bytes)

Byte	Byte 0	Byte 1
Description	Code	Logic State
Value	68-72	0-1

Returned Array

The returned array is made up of the following bytes:

- BYTE0
  - Code 73 for CS
  - Code 74 for LE
  - Code 75 for DI
  - Code 76 for DO
  - Code 77 for CLK
- BYTE1
  - The logic state of the requested byte (1 for high or 0 for low)
- BYTE2 to BYTE63
  - Could be any value (“don’t care” bytes)

Byte	Byte 0
Description	Code
Value	68-72

Example

Send the following transmit array to get the logic state of the DI pin:

Byte	Byte 0
Description	Code
Value	75

The following array would be returned if the DI pin is at logic high:

Byte	Byte 0	Byte 1
Description	Code	Logic State
Value	75	1

See Also

Set Data State

3.2 (k) - Get Firmware Revision

Description

This function returns the internal firmware version of the convertor.

Transmit Array

Send code 99 in BYTE0 of the transmit array (BYTE1 through BYTE63 can be any value).

Byte	Byte 0
Description	Code
Value	99

Returned Array

The firmware version will be returned in BYTE5 and BYTE6 as ASCII character codes representing the letter and number of the firmware. BYTE1 through to BYTE4 contain internal Mini-Circuits reference codes and BYTE7 through to BYTE63 are “don’t care” bytes and could be any value.

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Description	Code	Reserved	Reserved	Reserved	Reserved	Firmware Letter	Firmware Number
Value	99	ASCII	ASCII	ASCII	ASCII	ASCII	ASCII

Example

The following return array would indicate that the current firmware version is C3:

Byte	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Description	Code	Reserved	Reserved	Reserved	Reserved	Firmware Letter	Firmware Number
Value	99	ASCII	ASCII	ASCII	ASCII	ASCII	ASCII
ASCII Character	N/A	7	4	S	W	C	3

4 - Serial Control Using RS232 Communication

To create a serial RS232 connection to the converter, the following settings should be used:

- Baud = 9600
- Parity = E
- Data_Bits = 8

The 9 pin D-SUB connector of the converter should be connected to the computer's RS232 port. The device draws DC power through the USB type B connector; this can be connected to a computer or the AC mains adapter.

Communication with the converter is based on sending and receiving ASCII data over the RS232 port. Each command must be followed by a Carriage Return character.

A worked example is included in Appendix D.

4.1 - Summary of Commands

The commands that can be sent to the converter are summarized in the table below and detailed on the following pages.

	Description	Command	Comments
a	Get Device Model Name	M	
b	Get Device Serial Number	S	
c	Set/Get SPI Mode	D[m]	Set: m = mode (0-3) Get: m = ?
d	Send SPI Data	N[n]E[d]E	n = number of bits d = data to send
e	Receive SPI Data	R[n]E	n = number of bits
f	Send & Receive Data	A[n]E[d]E[CS][LE]	n = number of bits d = data to send CS = CS indicator LE = LE indicator
g	Set/Get CS	C[s]	Set: s = 0-1 Get: s = ?
h	Set/Get LE	L[s]	Set: s = 0-1 Get: s = ?
i	Set/Get DO	O[s]	Set: s = 0-1 Get: s = ?
j	Set/Get CLK	K[s]	Set: s = 0-1 Get: s = ?
k	Get DI	I?	

4.2 - Detailed Description of Commands

4.2 (a) - Get Device Model Name

This function returns the Mini-Circuits model name of the connected converter.

Command

```
M
```

Return Value

```
[mn]
```

Where:
[mn] = model name of the converter

Example

Send the text command "M", followed by Carriage Return.

The response will be of the format “RS232/USB-SPI” for model RS232/USB-SPI.

See Also

Get Device Serial Number

4.2 (b) - Get Device Serial Number

This function returns the serial number of the connected converter.

Command

`s`

Return Value

`[sn]`

Where:
[sn] = serial number of the converter

Example

Send the text command "S", followed by Carriage Return.

The response will be of the format “11301050025”.

See Also

Get Device Model Name

4.2 (c) - Set/Get SPI Mode

This function sets or gets the SPI (Serial Peripheral Interface) mode which specifies how the data stream is to be sampled. This ensures compatibility with the device that the converter is to communicate with. The default is mode 0 (IDLE=0 and SAMPLE_RISE).

Note: This function is only available with firmware B0 or later.

Command

D [m]

Where:
[m] = SPI mode from 0 to 3:
0 - IDLE=0 and SAMPLE_RISE (default)
1 - IDLE=0 and SAMPLE_FALL
2 - IDLE=1 and SAMPLE_FALL
3 - IDLE=1 and SAMPLE_RISE
Or:
[m] = "?" to read the current SPI mode

Return Value (Set SPI Mode)

1 (to indicate success)

Return Value (Get SPI Mode)

[m]

Where:
[m] = current SPI mode from 0 to 3

Example

To set mode 3 (IDLE=1 and SAMPLE_FALL):
Send the text command "D3", followed by Carriage Return
The response will be "1"

To read the current SPI mode:
Send the text command "D?", followed by Carriage Return
The response will be "3" if the converter is set to mode 3

4.2 (d) - Send SPI Data

This function sends a SPI (Serial Peripheral Interface) data string up to a maximum of 16 data bits.

Command

N[n]E[d]E

Where:
[n] = number of data bits to send (1 to 16)
[d] = decimal value of the binary data string to send

Return Value

ACK (to indicate success)

Example

To send the 8 bit binary data string “10011001”:

- [n] = 8 (number of data bits)
- [d] = 153 (decimal value of 10011001)

Send the text command “N8E153E”, followed by Carriage Return.

See Also

Receive SPI Data
Send and Receive SPI Data

4.2 (e) - Receive SPI Data

This function reads a user specified SPI (Serial Peripheral Interface) data string up to a maximum of 16 data bits.

Note: This function is only available with firmware B0 or later.

Command

R[n]E

Where:
[n] = number of data bits to read (1 to 16)

Return Value

ACK[b]

Where:
[b] = binary data string received (MSB first)

Example

To receive a 12 bit SPI data string (containing 001100101010):

Send the text command "R8E", followed by Carriage Return

The response will be "ACK001100101010"

See Also

Send SPI Data
Send and Receive SPI Data

4.2 (f) - Send and Receive SPI Data

This function sends a SPI (Serial Peripheral Interface) data string up to a maximum of 16 data bits, and reads a data string of the same length.

Note: This function is only available with firmware B0 or later.

Command

A[n]E[d]E[CS][LE]

Where:

[n] = number of data bits to send and receive (1 to 16)
[d] = decimal value of the binary data string to send
[CS]= Chip Select (CS) setting from 0 to 2:
0 - CS is not used
1 - Clear CS before sending SPI data and set CS after last bit is sent
2 - Set CS before sending SPI data and clear CS after last bit is sent
[LE] = Latch Enable (LE) setting from 0 to 2:
0 - LE is not used
1 - Toggle LE high then low after sending SPI data
2 - Toggle LE low then high after sending SPI data

Return Value

ACK[b]

Where:

[b] = binary data string received (MSB first)

Example

To set CS to logic low, send the 8 bit data string 00111000, set CS to logic high, then read the received SPI data string (containing 11000011):

- [n] = 8 (number of data bits)
- [d] = 56 (decimal value of 00111000)
- [CS] = 1 (set CS low before sending and high after sending)
- [LE] = 0 (not needed in this application)

Send the text command “A8E56E10”, followed by Carriage Return

The response will be “ACK11000011”

See Also

Send SPI Data
Receive SPI Data

4.2 (g) - Set/Get Chip Select (CS)

This function sets or gets the Chip Select (CS) pin logic state, either logic high or logic low.

Note: This function is only available with firmware B0 or later.

Command

`C[s]`

Where:
[s] = logic state to set (1 = high, 0 = low)
Or:
[s] = "?" to read the current state

Return Value

`[s]`

Where:
[s] = 1 to acknowledge success of the set function
Or:
[s] = current logic level (0 or 1) following the get function

Example

To set CS to logic 1:
Send the text command "C1", followed by Carriage Return
The response will be "1"

To read the current CS logic state:
Send the text command "C?", followed by Carriage Return
The response will be "1" if the pin is at logic high or "0" if the pin is at logic low

See Also

Set/Get Latch Enable (LE)
Set/Get Data Out (DO)
Set/Get Clock (CLK)
Get Data In (DI)

4.2 (h) - Set/Get Latch Enable (LE)

This function sets or gets the Latch Enable (LE) pin logic state, either logic high or logic low.

Command

L[s]

Where:
[s] = logic state to set (1 = high, 0 = low)
Or:
[s] = "?" to read the current state

Return Value

[s]

Where:
[s] = 1 to acknowledge success of the set function
Or:
[s] = current logic level (0 or 1) following the get function

Example

To set LE to logic 1:
Send the text command "L1", followed by Carriage Return
The response will be "1"

To read the current LE logic state:
Send the text command "L?", followed by Carriage Return
The response will be "1" if the pin is at logic high or "0" if the pin is at logic low

See Also

Set/Get Chip Select (CS)
Set/Get Data Out (DO)
Set/Get Clock (CLK)
Get Data In (DI)

4.2 (i) - Set/Get Data Out (DO)

This function sets or gets the Data Out (DO) pin logic state, either logic high or logic low.

Command

O[s]

Where:
[s] = logic state to set (1 = high, 0 = low)
Or:
[s] = "?" to read the current state

Return Value

[s]

Where:
[s] = 1 to acknowledge success of the set function
Or:
[s] = current logic level (0 or 1) following the get function

Example

To set DO to logic 1:
Send the text command "O1", followed by Carriage Return
The response will be "1"

To read the current DO logic state:
Send the text command "O?", followed by Carriage Return
The response will be "1" if the pin is at logic high or "0" if the pin is at logic low

See Also

Set/Get Chip Select (CS)
Set/Get Latch Enable (LE)
Set/Get Clock (CLK)
Get Data In (DI)

4.2 (j) - Set/Get Clock (CLK)

This function sets or gets the Clock (CLK) pin logic state, either logic high or logic low.

Command

```
K[s]
```

Where:
[s] = logic state to set (1 = high, 0 = low)
Or:
[s] = "?" to read the current state

Return Value

```
[s]
```

Where:
[s] = 1 to acknowledge success of the set function
Or:
[s] = current logic level (0 or 1) following the get function

Example

To set CLK to logic 1:
Send the text command "K1", followed by Carriage Return
The response will be "1"

To read the current CLK logic state:
Send the text command "K?", followed by Carriage Return
The response will be "1" if the pin is at logic high or "0" if the pin is at logic low

See Also

Set/Get Chip Select (CS)
Set/Get Latch Enable (LE)
Set/Get Data Out (DO)
Get Data In (DI)

4.2 (k) - Get Data In (DI)

This function gets the Data In (DI) pin logic state, either logic high or logic low.

Note: This function is only available with firmware B0 or later.

Command

```
I?
```

Return Value

```
[s]
```

Where:
[s] = current logic level (0 or 1)

Example

To read the current DO logic state:

Send the text command “I?”, followed by Carriage Return

The response will be “1” if the pin is at logic high or “0” if the pin is at logic low

See Also

Set/Get Chip Select (CS)
Set/Get Latch Enable (LE)
Set/Get Data Out (DO)
Set/Get Clock (CLK)