

INTRODUCTION

The MT Series transcoder is a remote control encoder and decoder in a single package. It is designed for bi-directional remote control applications, such as Remote Keyless Entry (RKE) with confirmation. However, the MT Series is capable of much more thanks to a Serial Interface Engine (SIE). While basic operation can be accomplished easily with the hardware control lines, the SIE enables full software control without using any of the hardware control lines and also enables several advanced features. This application note provides details on the advanced features and serial command set of the MT. Sample code is provided as an example and without any warranty.

SERIAL OUTPUT

Upon reception of every valid packet, the transcoder outputs a serial data stream consisting of a start byte, TX ID byte, status line state byte, custom data byte, and a stop byte. The start byte is always 0x00 and the stop byte is always 0xFF.

The bytes are output on the SER_IO line asynchronously, Least Significant Bit (LSB) first with one start bit, one stop bit, and no parity at the baud rate determined by the SEL_BAUD line. The line follows convention and is high when no data is being output (note that this line becomes an input when the transcoder goes to sleep).

The TX ID byte is a number that identifies which learned transcoder sent the transmission. The number normally corresponds to the order in which the transcoder was learned, so the first transcoder learned will get number '1' (binary 0000 0001), the second will get number '2' (binary 0000 0010), and so on. An exception arises when the memory is full, in which case the memory wraps around and the first numbers are overwritten, and if the SIE is used to write an address to a specific location.

The status line byte reflects the states of the status lines, '1' for on and '0' for off. This represents the current logic states of the outputs, not the command that was received, so that the states of latched lines are correctly represented. Line D0 corresponds to bit b0 in the byte, D1 corresponds to b1, and so forth. This allows applications that use an embedded microcontroller to read the transmitted commands without having to monitor eight hardware lines.

The custom data byte is a 1-byte value that is programmed on the transmitting side by the user through the SIE. If the option is disabled and no custom data is sent, this byte will be all high (binary 1111 1111).

Appendix A has sample code to read the bytes with a microcontroller and with a PC, using the USB module interface as described in the Hardware Interface section.

SERIAL INTERFACE ENGINE (SIE)

One of the most powerful features of the MT Series is its serial interface engine. The SIE allows the user to monitor and control the transcoder configuration settings through an automated system rather than manually through the hardware lines. While serial programming is not required for basic operation, it enables the advanced features offered by the MT, such as Targeted Device Addressing and Custom Data transmissions.

The SIE consists of twenty commands grouped into eight categories. The transcoder outputs an acknowledgement once it has received each command, and then a response of up to four additional bytes if required by the command. We will start with a description of each command and the features or options associated with them.

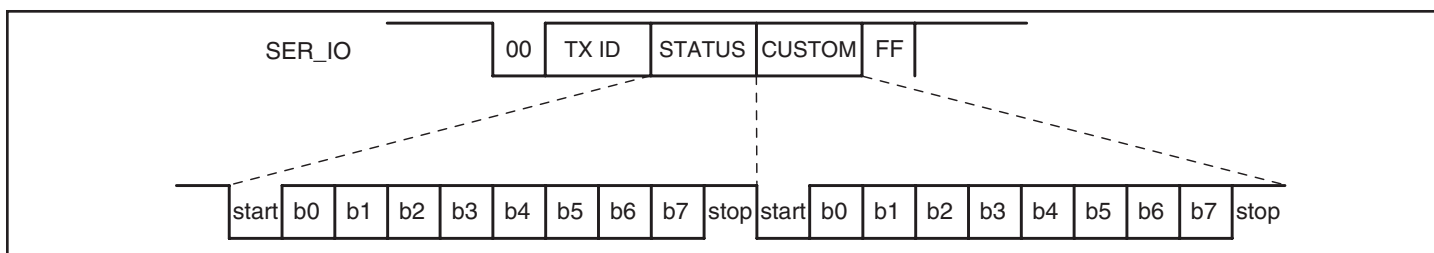


Figure 1: MT Series Transcoder Serial Output

LOCAL SETTINGS

This command reads and writes the transcoder's local 24-bit address and status line input/output configuration. This allows the option for the user to program all transcoders with the same address and status line configuration, or increment the address for each transcoder to utilize the full range of addresses.

The Read Local Settings command returns three bytes of the local address and one byte representing the input/output states of the status lines. Line D0 corresponds to bit b0 in the I/O state byte, D1 corresponds to b1, and so forth. A '0' means that the line represented by that bit is set to an output, a '1' means it is an input.

The Write Local Settings command allows the user to write a specific 3-byte address and 1-byte status line configuration to the transcoder.

NEXT USER ID

This command reads and writes the next available location that will be given to the next user that is manually learned by the transcoder. The Read Next User ID command returns the 1-byte ID that will be given to the next transcoder. The Write Next User ID command allows the user to assign the ID that will be given to the next transcoder that is learned. The transcoder will increment the ID from that point, overwriting the following IDs. For example, if there are 20 users learned and the Write Next User ID command is used to set the next user to 15, then then next transcoder learned manually will get ID 15. The one after that will get ID 16, overwriting the user that was already there.

SPECIFIC USER

This command reads and writes the address and Control Permissions of a specific device that has been learned in memory. This command can be used to change or remove a device that has been lost or stolen without having to re-learn all of the other devices in the system.

The Read Specific User command returns the three address bytes and one byte representing the status line permissions of the ID sent in the command. The Write Specific User command writes the address and permissions to the ID specified in the command. To remove an existing user, write 0xFF into the address and status line values. 0xFF is the default value for an empty ID, so it should not be used as a learned user.

TARGET ADDRESS

These commands enable, disable, read, and write the 24-bit address of the specific transcoder to which the transmission will be directed. All transcoders within range will get the transmission, but only the device with the address that matches the target address will take action and respond. This option requires that the addresses of all necessary system components are known in advance. There are many different ways of doing this, so it is up to the designer to determine the best method for their application.

The Read Device Targeting EN command returns the current state of the option, ('0' is disabled, '1' is enabled, disabled by default). The Write Device Targeting EN command allows the user to change the state of the option. The Read Target Address returns the address that is currently targeted. The Write Target Address allows the user to write the address of the device to be targeted.

CUSTOM DATA VALUE

These commands enable, disable, read, and write a custom data value that is sent with every packet. Care must be used if 0xFF (binary 1111 1111) is used as a legitimate value since this is the default output of the serial line when this option is disabled.

The Read Custom Data EN command returns the current state of the option, ('0' is disabled, '1' is enabled, disabled by default). The Write Custom Data EN command allows the user to change the state of the option. The Read Custom Data Value returns the byte that is currently in memory to be sent in the transmission. The Write Custom Data Value allows the user to write a new value to the transcoder.

LATCH MASK

The MT Series has the ability to make each status line that is set as an output be either latched or momentary. Momentary means that the line will only be high for as long as a valid signal is received. Once the signal stops and the transcoder times out, the lines are pulled low.

Latched means the transcoder will pull a data line high upon reception of a valid signal and hold it high until the signal is received a second time, at which point the transcoder will pull it low. The transcoder must see a break and time out between valid transmissions before it will toggle the outputs.

The Read Latch Mask Value command returns a byte corresponding to the current setting of each line, '0' for momentary or '1' for latched (bit b0 in the byte corresponds to line D0 and so forth). The Write Latch Mask command allows the user to write a byte to the transcoder to individually set the status lines as latched or momentary.

STATUS VALUE

This command reads the current state of the status line outputs and writes the state of the status line inputs for automatic transmission of a specified number of packets. The Read Status Outputs returns a byte that corresponds to the state of the status line outputs. The Write Status Line Inputs command writes a byte that corresponds to the desired state of the inputs and a byte that represents the number of packets to send. The transcoder will automatically send the specified number of packets as soon as the command is received.

CONFIRMATION

This command enables and disables the automatic confirmation. The confirmation is enabled by default, but some users may want to disable it to reduce the chance for interference in systems with multiple transcoders.

The Read Confirmation EN command returns the current state of the option, ('0' is disabled, '1' is enabled, disabled by default). The Write Confirmation EN command allows the user to change the state of the option.

SERIAL PROGRAMMING

The data structure sent into and coming out of the transcoder follows standard serial communication convention. Each byte is sent LSB first with one start bit, one stop bit, and no parity at the baud rate determined by the SEL_BAUD line. After the last command byte is received, there will be a 5mS pause while the transcoder processes the command, then it outputs the acknowledgement and a response if appropriate. Figure 2 shows the order and timing of the serial interface.

The MODE_IND line goes high for as long as the SER_IO line is an output, allowing it to be used with RS-232 style handshaking.

Appendix B has sample code for writing commands to the transcoder and receiving the acknowledgement and response.

HARDWARE INTERFACE

The serial interface on the MT Series can be connected to any device capable of serial communication, including microcontrollers, RS-232 drivers, and computers. Figure 3 gives an example of connecting the MT to the Linx QS Series USB module for connection to a computer.

The USB module follows the RS-232 convention of using separate lines for data input and data output while the transcoder has a single line for all data. This requires a switch to alternatively connect the transcoder's SER_IO line to the data lines on the module.

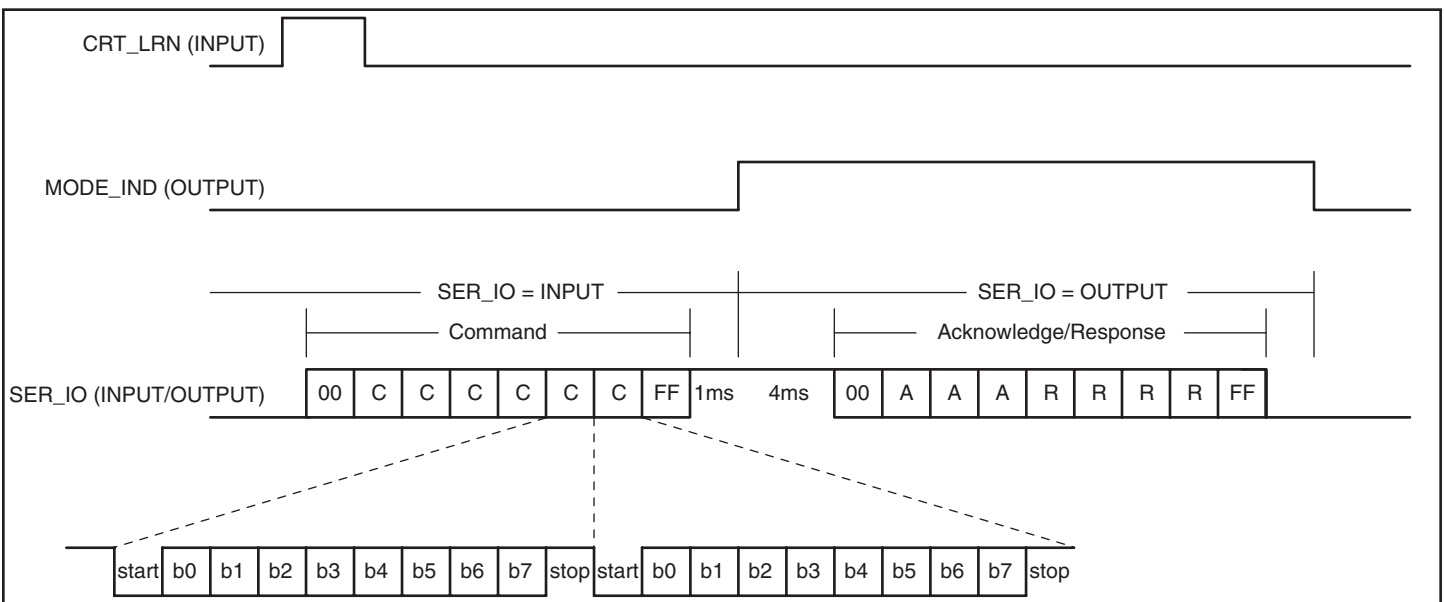


Figure 2: MT Series Transcoder Serial Programming

The RTS line is used to throw the switch as well as to activate the CRT/LRN line placing the transcoder into Serial Mode. This gives the PC the ability to control when communication is initiated.

The MODE_IND line will go high when the transcoder is prepared to send data, so the CTS line on the USB module is used to monitor the MODE_IND line. This allows the computer to know when to throw the switch and look for data from the transcoder.

One point of note is that voltage translation may be necessary if the 5V USB module is used to communicate with a transcoder operating at 3V. There are many components and methods for implementing level shifting, so it is up to the designer to determine the best solution for the product.

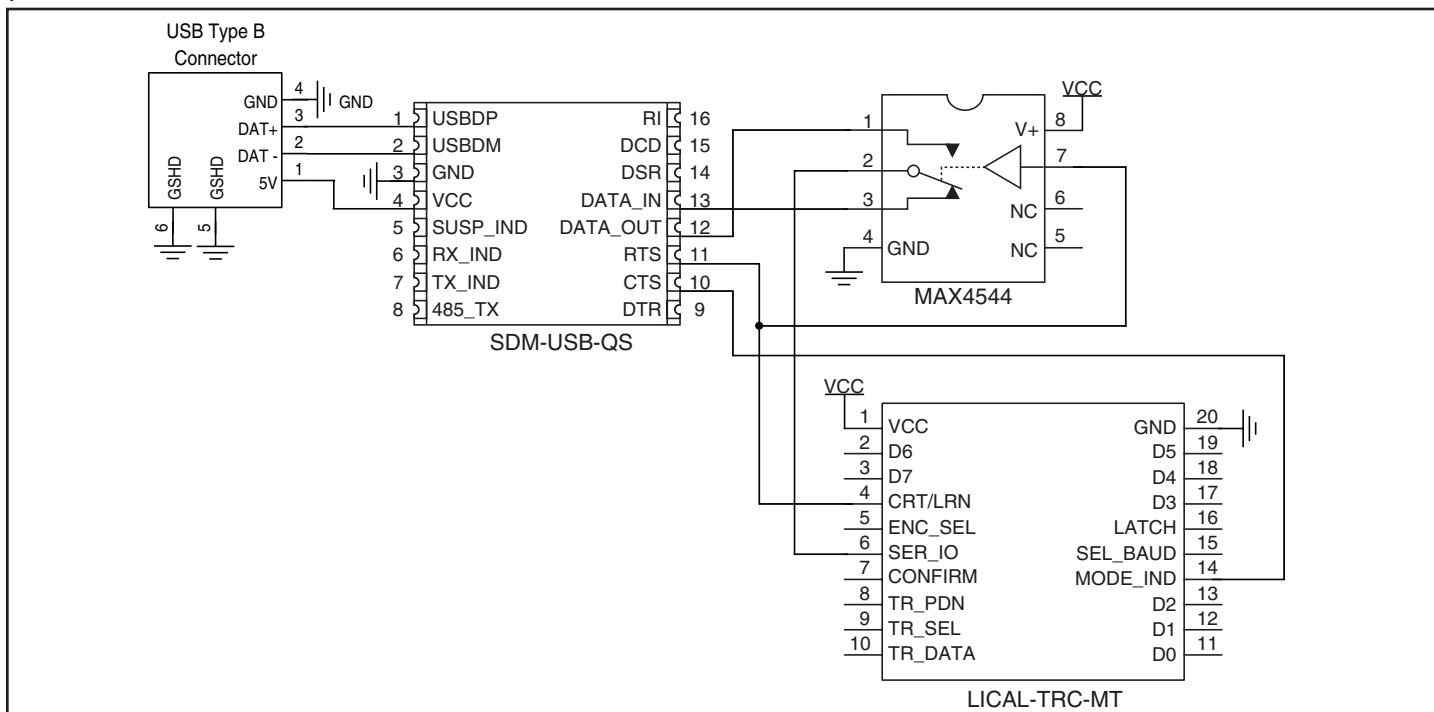


Figure 3: MT Series Transcoder Serial Interface to a PC

APPENDIX A

This software example is provided as a courtesy in "as is" condition. Linx Technologies makes no guarantee, representation, or warranty, whether express, implied, or statutory, regarding the suitability of the software for use in a specific application. The company shall not, in any circumstances, be liable for special, incidental, or consequential damages, for any reason whatsoever.

VISUAL BASIC CODE

This Visual Studio Visual Basic code uses the Linx SDM-USB-QS USB module to read the serial output of the transcoder with a PC.

'Setup the QS module for use.

```

' Open the device
Call Open_USB_Device_By_Description("LINX SDM-USB-QS-S")
' Reset device
Call Reset_USB_Device()
' Set baud rate to 9600
Call Set_USB_Device_Baud_Rate(FT_BAUD_9600)
' 8 data bits, 1 stop bit, no parity
Call Set_USB_Device_Data_Characteristics(FT_DATA_BITS_8, FT_STOP_BITS_1, FT_PARITY_NONE)
' no flow control
Call Set_USB_Device_Flow_Control(FT_FLOW_NONE, 0, 0)
' 25mS read timeout, 25mS write timeout
Call Set_USB_Device_Timeouts(25, 25)
' RX-buffer size = 9 bytes and TX-buffer size = 8 bytes for max packets size
Call Set_USB_Device_Parameters(9, 8)
' Latency ranges 2-255mS and defaults to 16mS
Call Set_USB_Device_Latency_Timer(2)
'Purge the RX buffer
Call Purge_USB_Device_RxBfr()
'Purge the TX buffer
Call Purge_USB_Device_TxBfr()
'Clear SER_IO & CRT_LRN pin for RXD
Call Set_USB_Device_RTS() 'Output Low

```

'Function for monitoring SER_IO data during MT-Receive-Mode.

```

Public Function RxSerialOutput() As Boolean

```

```

    Dim n As Integer = 0
    Dim TX_ID As Byte = 0
    Dim StatusOutputs As Byte = 0
    Dim CustomData As Byte = 0
    Dim DataReceived As Boolean = False

```

```

'If the Mode_Ind is set(HIGH), receive data

```

```

If ((Get_USB_Device_Modem_Status() And FT_MODEM_STATUS_CTS) <> FT_MODEM_STATUS_CTS) Then

```

```

    'Perform Read on expected 5-bytes

```

```

    n = Read_Data_Bytes(5)

```

```

    'Make sure all 5 were received

```

```

    If (n = 5) Then

```

```

        'Test the Start and Stop bytes
    End If
End If
End Function

```

```

    If (FT_In_Buffer(0) = &H0) And (FT_In_Buffer(4) = &HFF) Then
        'Make sure expected UserID is in range
        If (FT_In_Buffer(1) > 0) And (FT_In_Buffer(1) < 61) Then
            'Assign RX values
            TX_ID = FT_In_Buffer(1)
            StatusOutputs = FT_In_Buffer(2)
            CustomData = FT_In_Buffer(3)
            DataReceived = True
        End If
    End If
    End If
    'Clear RX buffer for next receive
    Call Purge_USB_Device_RxBfr()
    'Return true if data was received
    RxSerialOutput = DataReceived
End If
End Function

```

MICROCONTROLLER CODE

This C code can be used by a microcontroller to monitor the serial output of the transcoder.

```

void RxSerialOut_2PC(void)
{
    int8 i = 0;
    char RxIn[5];

    // Get the 5 byte packet from MT RxSerialOut
    for(i=0; i<5; i++) RxIn[i] = Get_Byte();

    // If start and stop byte are good, send RX data to PC
    if((RxIn[0] == 0x00)&&(RxIn[4] == 0xFF)) {
        for(i=0; i<5; i++) putc(RxIn[i]);
    }
    // Clear RxIn buffer
    for(i=0; i<5; i++) RxIn[i] = 0;
}
//*****
void Put_Byte(int8 Data)
{
    // 100us for 9600baud or 30us for 28800baud
    int8 BitTm = 100;

    // Start bit
    output_low(Ser_IO); delay_us(BitTm);
    // 8 data bits
    output_bit(Ser_IO, bit_test(Data,0)); delay_us(BitTm);
    output_bit(Ser_IO, bit_test(Data,1)); delay_us(BitTm);
}

```

```
output_bit(Ser_IO, bit_test(Data,2)); delay_us(BitTm);
output_bit(Ser_IO, bit_test(Data,3)); delay_us(BitTm);
output_bit(Ser_IO, bit_test(Data,4)); delay_us(BitTm);
output_bit(Ser_IO, bit_test(Data,5)); delay_us(BitTm);
output_bit(Ser_IO, bit_test(Data,6)); delay_us(BitTm);
output_bit(Ser_IO, bit_test(Data,7)); delay_us(BitTm);
// Stop bit
output_high(Ser_IO); delay_us(BitTm);
}
//*****
int8 Get_Byte(void)
{
// 100us for 9600baud or 30us for 28800baud
int8 BitTm = 100;
int8 DataByte = 0;

// Wait for Ser_IO to drop for start bit
while(input(Ser_IO)) {}

// Start bit
delay_us(BitTm/2);
// 8 data bits
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,0);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,1);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,2);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,3);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,4);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,5);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,6);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,7);
// Stop bit
delay_us(BitTm);

// Return received value
return(DataByte);
}
//*****
```

APPENDIX B

This software example is provided as a courtesy in "as is" condition. Linx Technologies makes no guarantee, representation, or warranty, whether express, implied, or statutory, regarding the suitability of the software for use in a specific application. The company shall not, in any circumstances, be liable for special, incidental, or consequential damages, for any reason whatsoever.

VISUAL BASIC CODE

This Visual Studio Visual Basic code uses the Linx SDM-USB-QS USB module to write commands to the transcoder from a PC and to receive the response from the transcoder.

'Setup the QS module for use.

```

' Open the device
Call Open_USB_Device_By_Description(GetFTDeviceDescription(0))
' Reset device
Call Reset_USB_Device()
' Set baud rate to 9600
Call Set_USB_Device_Baud_Rate(FT_BAUD_9600)
' 8 data bits, 1 stop bit, no parity
Call Set_USB_Device_Data_Characteristics(FT_DATA_BITS_8, FT_STOP_BITS_1, FT_PARITY_NONE)
' no flow control
Call Set_USB_Device_Flow_Control(FT_FLOW_NONE, 0, 0)
' 25mS read timeout, 25mS write timeout
Call Set_USB_Device_Timeouts(25, 25)
' RX-buffer size = 9 bytes and TX-buffer size = 8 bytes for max packets size
Call Set_USB_Device_Parameters(9, 8)
' Latency ranges 2-255mS and defaults to 16mS
Call Set_USB_Device_Latency_Timer(2)
'Purge the RX buffer
Call Purge_USB_Device_RxBfr()
'Purge the TX buffer
Call Purge_USB_Device_TxBfr()
'Clear SER_IO & CRT_LRN pin for RXD
Call Set_USB_Device_RTS() 'Output Low

```

'Function for transmitting packets and receiving confirmation.

```

Public Function TransferData(ByVal Wr_Cnt As Integer, ByVal Rd_Cnt As Integer) As Boolean
    Dim i As Integer = 0
    Dim n As Integer = 0
    Dim blnACK_Rcvd As Boolean = False

    'Purge the QS RX/TX buffers
    Call Purge_USB_Device_RxBfr()
    Call Purge_USB_Device_TxBfr()

    'Set SER_IO & CRT_LRN pin to trigger MT for serial data and set switch to QS-TX-Data pin
    Call Clear_USB_Device_RTS() 'Output HIGH

```

'Allow time(ms) for MT to finish possible RX or PDN modes

```
Sleep(50)
```

```
'If Mode_Ind is clear, send the data
```

```
If (Get_USB_Device_Modem_Status() And FT_MODEM_STATUS_CTS) Then
```

```
    Call Write_Data_Bytes(Wr_Cnt)
```

```
    'Wait for Mode_Ind to go HIGH indicating MT command reception
```

```
    Do While ((Get_USB_Device_Modem_Status() And FT_MODEM_STATUS_CTS) And (i < 10000))
```

```
        i += 1
```

```
    Loop
```

```
End If
```

```
'Clear SER_IO & CRT_LRN pin to flip switch for QS-RX-Data pin
```

```
Call Set_USB_Device_RTS() 'Output LOW
```

```
'Make sure Write-Wait loop didn't timeout
```

```
If (i < 10000) Then
```

```
    'Read the QS RX buffer
```

```
    n = Read_Data_Bytes(Rd_Cnt)
```

```
    If (n > 4) Then
```

```
        'Test Start and Stop bytes
```

```
        If (FT_In_Buffer(0) = 0 And FT_In_Buffer(n - 1) = 255) Then
```

```
            blnACK_Rcvd = True
```

```
        End If
```

```
    End If
```

```
End If
```

```
'Return true if ACK message was received from MT device
```

```
TransferData = blnACK_Rcvd
```

```
End Function
```

```
'Functions used to fill the QS TX buffers with each command.
```

```
Public Function DoRdLocalSettings() As Boolean
```

```
    FT_Out_Buffer(0) = &H0
```

```
    FT_Out_Buffer(1) = &H1
```

```
    FT_Out_Buffer(2) = &H0
```

```
    FT_Out_Buffer(3) = &H0
```

```
    FT_Out_Buffer(4) = &H0
```

```
    FT_Out_Buffer(5) = &H0
```

```
    FT_Out_Buffer(6) = &H0
```

```
    FT_Out_Buffer(7) = &HFF
```

```
If TransferData(8, 9) = True Then
```

```
    DoRdLocalSettings = True
```

```
Else
```

```
    DoRdLocalSettings = False
```

```
End If
```

```
End Function
```

```
Public Function DoWrLocalSettings(ByVal A1 As Byte, ByVal A2 As Byte, ByVal A3 As Byte, ByVal IO As Byte) As Boolean
```

```
    FT_Out_Buffer(0) = &H0
```

```
    FT_Out_Buffer(1) = &H2
```

```
    FT_Out_Buffer(2) = A1
```

```
    FT_Out_Buffer(3) = A2
```

```
    FT_Out_Buffer(4) = A3
```

```
    FT_Out_Buffer(5) = IO
```

```
    FT_Out_Buffer(6) = &H0
```

```
    FT_Out_Buffer(7) = &HFF
```

```
    If TransferData(8, 5) = True Then
```

```
        DoWrLocalSettings = True
```

```
    Else
```

```
        DoWrLocalSettings = False
```

```
    End If
```

```
End Function
```

```
Public Function DoRdNextUserID() As Boolean
```

```
    FT_Out_Buffer(0) = &H0
```

```
    FT_Out_Buffer(1) = &H11
```

```
    FT_Out_Buffer(2) = &H0
```

```
    FT_Out_Buffer(3) = &H0
```

```
    FT_Out_Buffer(4) = &H0
```

```
    FT_Out_Buffer(5) = &H0
```

```
    FT_Out_Buffer(6) = &H0
```

```
    FT_Out_Buffer(7) = &HFF
```

```
    If TransferData(8, 6) = True Then
```

```
        DoRdNextUserID = True
```

```
    Else
```

```
        DoRdNextUserID = False
```

```
    End If
```

```
End Function
```

```
Public Function DoWrNextUserID(ByVal ID As Byte) As Boolean
```

```
    FT_Out_Buffer(0) = &H0
```

```
    FT_Out_Buffer(1) = &H12
```

```
    FT_Out_Buffer(2) = ID
```

```
    FT_Out_Buffer(3) = &H0
```

```
    FT_Out_Buffer(4) = &H0
```

```
    FT_Out_Buffer(5) = &H0
```

```
    FT_Out_Buffer(6) = &H0
```

```
    FT_Out_Buffer(7) = &HFF
```

```
    If TransferData(8, 5) = True Then
```

```
        DoWrNextUserID = True
```

```
Else
    DoWrNextUserID = False
End If
End Function
```

```
Public Function DoRdSpecificUser(ByVal ID As Byte) As Boolean
```

```
    FT_Out_Buffer(0) = &H0
    FT_Out_Buffer(1) = &H21
    FT_Out_Buffer(2) = ID
    FT_Out_Buffer(3) = &H0
    FT_Out_Buffer(4) = &H0
    FT_Out_Buffer(5) = &H0
    FT_Out_Buffer(6) = &H0
    FT_Out_Buffer(7) = &HFF
```

```
    If TransferData(8, 9) = True Then
        DoRdSpecificUser = True
    Else
        DoRdSpecificUser = False
    End If
```

```
End Function
```

```
Public Function DoWrSpecificUser(ByVal A1 As Byte, ByVal A2 As Byte, ByVal A3 As Byte, ByVal IO As Byte, ByVal ID As Byte) As Boolean
```

```
    FT_Out_Buffer(0) = &H0
    FT_Out_Buffer(1) = &H22
    FT_Out_Buffer(2) = A1
    FT_Out_Buffer(3) = A2
    FT_Out_Buffer(4) = A3
    FT_Out_Buffer(5) = IO
    FT_Out_Buffer(6) = ID
    FT_Out_Buffer(7) = &HFF
```

```
    If TransferData(8, 5) = True Then
        DoWrSpecificUser = True
    Else
        DoWrSpecificUser = False
    End If
```

```
End Function
```

```
Public Function DoRdTargetAddr() As Boolean
```

```
    FT_Out_Buffer(0) = &H0
    FT_Out_Buffer(1) = &H31
    FT_Out_Buffer(2) = &H0
    FT_Out_Buffer(3) = &H0
    FT_Out_Buffer(4) = &H0
    FT_Out_Buffer(5) = &H0
```

```
FT_Out_Buffer(6) = &H0
FT_Out_Buffer(7) = &HFF

If TransferData(8, 8) = True Then
    DoRdTargetAddr = True
Else
    DoRdTargetAddr = False
End If
End Function

Public Function DoWrTargetAddr(ByVal A1 As Byte, ByVal A2 As Byte, ByVal A3 As Byte) As Boolean
    FT_Out_Buffer(0) = &H0
    FT_Out_Buffer(1) = &H32
    FT_Out_Buffer(2) = A1
    FT_Out_Buffer(3) = A2
    FT_Out_Buffer(4) = A3
    FT_Out_Buffer(5) = &H0
    FT_Out_Buffer(6) = &H0
    FT_Out_Buffer(7) = &HFF

    If TransferData(8, 5) = True Then
        DoWrTargetAddr = True
    Else
        DoWrTargetAddr = False
    End If
End Function

Public Function DoRdCustomData() As Boolean
    FT_Out_Buffer(0) = &H0
    FT_Out_Buffer(1) = &H41
    FT_Out_Buffer(2) = &H0
    FT_Out_Buffer(3) = &H0
    FT_Out_Buffer(4) = &H0
    FT_Out_Buffer(5) = &H0
    FT_Out_Buffer(6) = &H0
    FT_Out_Buffer(7) = &HFF

    If TransferData(8, 6) = True Then
        DoRdCustomData = True
    Else
        DoRdCustomData = False
    End If
End Function

Public Function DoWrCustomData(ByVal CB As Byte) As Boolean
    FT_Out_Buffer(0) = &H0
    FT_Out_Buffer(1) = &H42
```

```
FT_Out_Buffer(2) = CB
FT_Out_Buffer(3) = &H0
FT_Out_Buffer(4) = &H0
FT_Out_Buffer(5) = &H0
FT_Out_Buffer(6) = &H0
FT_Out_Buffer(7) = &HFF
```

```
If TransferData(8, 5) = True Then
```

```
    DoWrCustomData = True
```

```
Else
```

```
    DoWrCustomData = False
```

```
End If
```

```
End Function
```

```
Public Function DoRdLatchMask() As Boolean
```

```
    FT_Out_Buffer(0) = &H0
```

```
    FT_Out_Buffer(1) = &H51
```

```
    FT_Out_Buffer(2) = &H0
```

```
    FT_Out_Buffer(3) = &H0
```

```
    FT_Out_Buffer(4) = &H0
```

```
    FT_Out_Buffer(5) = &H0
```

```
    FT_Out_Buffer(6) = &H0
```

```
    FT_Out_Buffer(7) = &HFF
```

```
If TransferData(8, 6) = True Then
```

```
    DoRdLatchMask = True
```

```
Else
```

```
    DoRdLatchMask = False
```

```
End If
```

```
End Function
```

```
Public Function DoWrLatchMask(ByVal LM As Byte) As Boolean
```

```
    FT_Out_Buffer(0) = &H0
```

```
    FT_Out_Buffer(1) = &H52
```

```
    FT_Out_Buffer(2) = LM
```

```
    FT_Out_Buffer(3) = &H0
```

```
    FT_Out_Buffer(4) = &H0
```

```
    FT_Out_Buffer(5) = &H0
```

```
    FT_Out_Buffer(6) = &H0
```

```
    FT_Out_Buffer(7) = &HFF
```

```
If TransferData(8, 5) = True Then
```

```
    DoWrLatchMask = True
```

```
Else
```

```
    DoWrLatchMask = False
```

```
End If
```

```
End Function
```

```
Public Function DoRdStatusOuputs() As Boolean
```

```
    FT_Out_Buffer(0) = &H0
```

```
    FT_Out_Buffer(1) = &H61
```

```
    FT_Out_Buffer(2) = &H0
```

```
    FT_Out_Buffer(3) = &H0
```

```
    FT_Out_Buffer(4) = &H0
```

```
    FT_Out_Buffer(5) = &H0
```

```
    FT_Out_Buffer(6) = &H0
```

```
    FT_Out_Buffer(7) = &HFF
```

```
    If TransferData(8, 6) = True Then
```

```
        DoRdStatusOuputs = True
```

```
    Else
```

```
        DoRdStatusOuputs = False
```

```
    End If
```

```
End Function
```

```
Public Function DoWrStatusInputs(ByVal Data As Byte, ByVal Num As Byte) As Boolean
```

```
    FT_Out_Buffer(0) = &H0
```

```
    FT_Out_Buffer(1) = &H62
```

```
    FT_Out_Buffer(2) = Data
```

```
    FT_Out_Buffer(3) = Num
```

```
    FT_Out_Buffer(4) = &H0
```

```
    FT_Out_Buffer(5) = &H0
```

```
    FT_Out_Buffer(6) = &H0
```

```
    FT_Out_Buffer(7) = &HFF
```

```
    If TransferData(8, 5) = True Then
```

```
        DoWrStatusInputs = True
```

```
    Else
```

```
        DoWrStatusInputs = False
```

```
    End If
```

```
End Function
```

```
Public Function DoRdConfirmEn() As Boolean
```

```
    FT_Out_Buffer(0) = &H0
```

```
    FT_Out_Buffer(1) = &H71
```

```
    FT_Out_Buffer(2) = &H0
```

```
    FT_Out_Buffer(3) = &H0
```

```
    FT_Out_Buffer(4) = &H0
```

```
    FT_Out_Buffer(5) = &H0
```

```
    FT_Out_Buffer(6) = &H0
```

```
    FT_Out_Buffer(7) = &HFF
```

```
    If TransferData(8, 6) = True Then
```

```
        DoRdConfirmEn = True
```

```
Else
    DoRdConfirmEn = False
End If
End Function

Public Function DoWrConfirmEn(ByVal CfEn As Byte) As Boolean
    FT_Out_Buffer(0) = &H0
    FT_Out_Buffer(1) = &H72
    FT_Out_Buffer(2) = CfEn
    FT_Out_Buffer(3) = &H0
    FT_Out_Buffer(4) = &H0
    FT_Out_Buffer(5) = &H0
    FT_Out_Buffer(6) = &H0
    FT_Out_Buffer(7) = &HFF

    If TransferData(8, 5) = True Then
        DoWrConfirmEn = True
    Else
        DoWrConfirmEn = False
    End If
End Function

Public Function DoRdTargetEn() As Boolean
    FT_Out_Buffer(0) = &H0
    FT_Out_Buffer(1) = &H81
    FT_Out_Buffer(2) = &H0
    FT_Out_Buffer(3) = &H0
    FT_Out_Buffer(4) = &H0
    FT_Out_Buffer(5) = &H0
    FT_Out_Buffer(6) = &H0
    FT_Out_Buffer(7) = &HFF

    If TransferData(8, 6) = True Then
        DoRdTargetEn = True
    Else
        DoRdTargetEn = False
    End If
End Function

Public Function DoWrTargetEn(ByVal TgEn As Byte) As Boolean
    FT_Out_Buffer(0) = &H0
    FT_Out_Buffer(1) = &H82
    FT_Out_Buffer(2) = TgEn
    FT_Out_Buffer(3) = &H0
    FT_Out_Buffer(4) = &H0
    FT_Out_Buffer(5) = &H0
    FT_Out_Buffer(6) = &H0
```

```
FT_Out_Buffer(7) = &HFF
```

```
If TransferData(8, 5) = True Then
```

```
    DoWrTargetEn = True
```

```
Else
```

```
    DoWrTargetEn = False
```

```
End If
```

```
End Function
```

```
Public Function DoRdCustomDataEn() As Boolean
```

```
    FT_Out_Buffer(0) = &H0
```

```
    FT_Out_Buffer(1) = &H91
```

```
    FT_Out_Buffer(2) = &H0
```

```
    FT_Out_Buffer(3) = &H0
```

```
    FT_Out_Buffer(4) = &H0
```

```
    FT_Out_Buffer(5) = &H0
```

```
    FT_Out_Buffer(6) = &H0
```

```
    FT_Out_Buffer(7) = &HFF
```

```
If TransferData(8, 6) = True Then
```

```
    DoRdCustomDataEn = True
```

```
Else
```

```
    DoRdCustomDataEn = False
```

```
End If
```

```
End Function
```

```
Public Function DoWrCustomDataEn(ByVal CdEn As Byte) As Boolean
```

```
    FT_Out_Buffer(0) = &H0
```

```
    FT_Out_Buffer(1) = &H92
```

```
    FT_Out_Buffer(2) = CdEn
```

```
    FT_Out_Buffer(3) = &H0
```

```
    FT_Out_Buffer(4) = &H0
```

```
    FT_Out_Buffer(5) = &H0
```

```
    FT_Out_Buffer(6) = &H0
```

```
    FT_Out_Buffer(7) = &HFF
```

```
If TransferData(8, 5) = True Then
```

```
    DoWrCustomDataEn = True
```

```
Else
```

```
    DoWrCustomDataEn = False
```

```
End If
```

```
End Function
```

MICROCONTROLLER CODE

This C code can be used by a microcontroller to write commands to the transcoder and receive the response.

```
// Constant Arrays
const char RdLocSttns[8] = {0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char WrLocSttns[8] = {0x00, 0x02, 0x11, 0x22, 0x33, 0x0F, 0x00, 0xFF};
const char RdNxtUser[8]  = {0x00, 0x11, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char WrNxtUser[8]  = {0x00, 0x12, 0x09, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char RdSpecUser[8] = {0x00, 0x21, 0x3C, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char WrSpecUser[8] = {0x00, 0x22, 0x12, 0x34, 0x56, 0x78, 0x3C, 0xFF};
const char RdTrgtAddr[8] = {0x00, 0x31, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char WrTrgtAddr[8] = {0x00, 0x32, 0x11, 0x22, 0x33, 0x00, 0x00, 0xFF};
const char RdCstmData[8] = {0x00, 0x41, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char WrCstmData[8] = {0x00, 0x42, 0xAA, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char RdLatchMsk[8] = {0x00, 0x51, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char WrLatchMsk[8] = {0x00, 0x52, 0x55, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char RdStatus_Out[8] = {0x00, 0x61, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char WrStatus_In[8]  = {0x00, 0x62, 0xFF, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char RdCnfrmEN[8]   = {0x00, 0x71, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char WrCnfrmEN[8]   = {0x00, 0x72, 0x01, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char RdTrgtngEN[8]  = {0x00, 0x81, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char WrTrgtngEN[8]  = {0x00, 0x82, 0x01, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char RdCstmDtaEN[8] = {0x00, 0x91, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF};
const char WrCstmDtaEN[8] = {0x00, 0x92, 0x01, 0x00, 0x00, 0x00, 0x00, 0xFF};

// Variable Arrays
char TxCmdDta[8];
char RxCmdDta[9];

// Constants
const int8 SerIO_IN = 0b11000111;
const int8 SerIO_OUT = 0b11000011;

//*****
void ProcessCmd(void)
{
    int8 i = 0;

    if(input(Cmd_RW)) {
        // Determine which write command to send
        if (input(Cmd_1)) { for(i=0; i<8; i++) TxCmdDta[i] = WrLocSttns[i]; }
        else if(input(Cmd_2)) { for(i=0; i<8; i++) TxCmdDta[i] = WrSpecUser[i]; }
        else if(input(Cmd_3)) { for(i=0; i<8; i++) TxCmdDta[i] = WrTrgtAddr[i]; }
        else if(input(Cmd_4)) { for(i=0; i<8; i++) TxCmdDta[i] = WrCstmData[i]; }
        else if(input(Cmd_5)) { for(i=0; i<8; i++) TxCmdDta[i] = WrLatchMsk[i]; }
        else if(input(Cmd_6)) { for(i=0; i<8; i++) TxCmdDta[i] = WrCnfrmEN[i]; }
    }
}
```

```

    else if(input(Cmd_7)) { for(i=0; i<8; i++) TxCmdDta[i] = WrTrgtngEN[i]; }
    else if(input(Cmd_8)) { for(i=0; i<8; i++) TxCmdDta[i] = WrCstmDtaEN[i]; }
    else
        { for(i=0; i<8; i++) TxCmdDta[i] = WrNxtUser[i]; }
}
else {
    // Determine which read command to send
    if (input(Cmd_1)) { for(i=0; i<8; i++) TxCmdDta[i] = RdLocSttns[i]; }
    else if(input(Cmd_2)) { for(i=0; i<8; i++) TxCmdDta[i] = RdSpecUser[i]; }
    else if(input(Cmd_3)) { for(i=0; i<8; i++) TxCmdDta[i] = RdTrgtAddr[i]; }
    else if(input(Cmd_4)) { for(i=0; i<8; i++) TxCmdDta[i] = RdCstmData[i]; }
    else if(input(Cmd_5)) { for(i=0; i<8; i++) TxCmdDta[i] = RdLatchMsk[i]; }
    else if(input(Cmd_6)) { for(i=0; i<8; i++) TxCmdDta[i] = RdCnfrmEN[i]; }
    else if(input(Cmd_7)) { for(i=0; i<8; i++) TxCmdDta[i] = RdTrgtngEN[i]; }
    else if(input(Cmd_8)) { for(i=0; i<8; i++) TxCmdDta[i] = RdCstmDtaEN[i]; }
    else
        { for(i=0; i<8; i++) TxCmdDta[i] = RdNxtUser[i]; }
}
// Make SerIO pin an output
set_tris_a(SerIO_OUT);
// Set both lines HIGH at the same time
output_high(CrtLrn);
output_high(Ser_IO);
// Allow MT time to enter SIE command mode
delay_ms(50);
output_low(CrtLrn);
// Send the command to the MT
for(i=0; i<8; i++) { Put_Byte(TxCmdDta[i]); }
delay_ms(1);
// Make SerIO an input
set_tris_a(SerIO_IN);
delay_ms(4);
// Get the ACK/reply from the MT
for(i=0; i<9; i++) RxCmdDta[i] = Get_Byte();
}
//*****
void Put_Byte(int8 Data)
{
    // 100us for 9600baud or 30us for 28800baud
    int8 BitTm = 100;

    // Start bit
    output_low(Ser_IO); delay_us(BitTm);
    // 8 data bits
    output_bit(Ser_IO, bit_test(Data,0)); delay_us(BitTm);
    output_bit(Ser_IO, bit_test(Data,1)); delay_us(BitTm);
    output_bit(Ser_IO, bit_test(Data,2)); delay_us(BitTm);
    output_bit(Ser_IO, bit_test(Data,3)); delay_us(BitTm);
    output_bit(Ser_IO, bit_test(Data,4)); delay_us(BitTm);

```

```
output_bit(Ser_IO, bit_test(Data,5)); delay_us(BitTm);
output_bit(Ser_IO, bit_test(Data,6)); delay_us(BitTm);
output_bit(Ser_IO, bit_test(Data,7)); delay_us(BitTm);
// Stop bit
output_high(Ser_IO); delay_us(BitTm);
}
//*****
int8 Get_Byte(void)
{
// 100us for 9600baud or 30us for 28800baud
int8 BitTm = 100;
int8 DataByte = 0;

// Wait for Ser_IO to drop for start bit
while(input(Ser_IO)) {}

// Start bit
delay_us(BitTm/2);
// 8 data bits
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,0);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,1);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,2);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,3);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,4);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,5);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,6);
delay_us(BitTm); if(input(Ser_IO)) bit_set(DataByte,7);
// Stop bit
delay_us(BitTm);

// Return received value
return(DataByte);
}
//*****
```