QS Series Master Development System

Software Documentation

Introduction

The software included with the MDEV-USB-QS Master Development Kit is a quick and easy way to test the features of the QS Series USB module. The Manual for the kit explains the board and hardware, and this document will explain how to use the software and explains some of the source code. Any questions not answered by this document can be referred to Linx.

The Program Interface

The development board connects the QS module to one of two sections, so when the software is first started, a screen will be displayed asking the user to choose which section of the board will be used. The **USB to RS232** section will activate the USB module and the serial port on the PC. This section will send data through the USB bus and receive it back through the serial port, and vice versa.

The **USB to PIC** section demonstrates how to interface the QS module with a popular microcontroller, the PIC16F88 from Microchip. This section will demonstrate how to have the PC instruct the PIC to turn on some LEDs and have the PIC tell the PC to light an indicator on the screen when a button on the board is pressed and move a slider when a potentiometer is turned. The user should decide which section will be used and click the appropriate button.

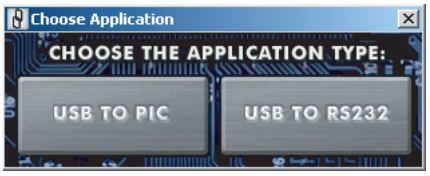

Figure 1: The Choose Application Screen

The PIC Control Screen

Selecting **USB TO PIC** will bring up the **PIC Control** screen as shown in Figure 2.

This screen is divided into several sections. At the top, the **CURRENT DEVICE** box will display the description and serial number of the device to which the computer is currently connected. The **CONNECTED USB DEVICES** box below that will list the description and serial number of all of the QS modules that are currently connected to the bus. Clicking on one of the listed devices and pressing the STATUS button will change the current device to the one that was highlighted in the box. A green light above the STATUS button indicates that the device is connected and

Figure 2: The PIC Control Screen

communicating normally. A red light indicates that an error has occurred or that no device is connected.

The section labeled **A/D VOLTAGE ADJUST** has a light bar and an indicator that will change based on the position of the **VOLTAGE ADJUST** potentiometer on the evaluation board. The PIC will take a voltage reading from its internal Analog-to-Digital Converter (ADC), will convert that value into a voltage, and will send that voltage to the computer. The computer will display the voltage in the indicator box and will activate the light bar based on the percentage of full voltage.

The section labeled **MODEM LINES** controls the QS module's UART handshaking lines. The outputs, **RTS** and **DTR** can be activated by pressing the buttons on the screen, which will turn on LEDs on the evaluation board. Pressing the Modem Line buttons on the evaluation board will cause the appropriate input indicator on the screen to turn on.

The buttons labeled **LED 1**, **LED 2** and **LED 3** will send a signal to the PIC that will turn on and off three LEDs on the evaluation board. The indicators above the buttons will also light up to match the state of the LEDs.

Pressing the button on the evaluation board will cause the PIC to send a signal to the PC that will cause the indicator labeled **USB COM** to turn on and off.

The RS232 Control Screen

Selecting **USB TO RS232** from the **CHOOSE APPLICATION** screen will bring up the **RS232 CONTROL** screen as shown in Figure 3.

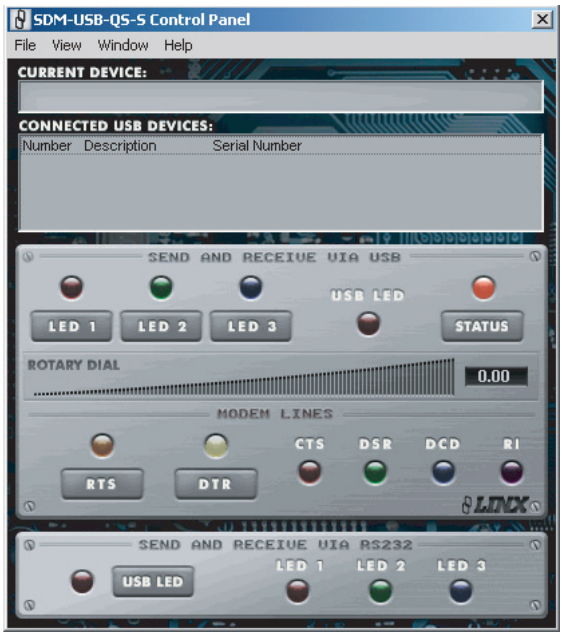

Figure 3: The RS232 Control Screen

Selecting this option causes the **SEND AND RECEIVE VIA RS232** section to become available. Pressing the **USB COM** button will send a signal out of the computer's serial port to the evaluation board, and back into the computer through the QS module. This signal will cause the **USB COM** indicator on the screen to turn on. Likewise, pressing the **LED1**, **LED2** and **LED3** buttons in the **SEND AND RECEIVE VIA USB** section will cause the appropriate lights in the **SEND AND RECEIVE VIA RS232** section to turn on.

The Menus

The **CONTROL** screen has four menu items at the top of the screen. The first menu, **FILE**, has only the **EXIT** option. This selection will close the windows and exit the program.

The **VIEW** menu has three options: **USB PROPERTIES**, **RS232 PROPERTIES** and **CONTACT US**. The **USB PROPERTIES** screen contains the communications settings for the USB port, as seen in Figure 4.

Figure 4: The USB Properties Screen

The **DEVICE DESCRIPTION** box will show the description of the current device. The **BAUD RATE**, **DATA BITS** and **PARITY** selection boxes allow the user to set the communication parameters. When in PIC mode, these options will be disabled and set for proper communication with the processor. The **ERROR DESCRIPTION** and **LOG REPORT** will list any communication errors that may occur. Click **OK** to apply any changes or CANCEL to exit out and return to the CONTROL screen with no changes.

The **RS232 PROPERTIES** screen contains the communications settings for the serial port, as seen in Figure 5.

Figure 5: The RS232 Properties Screen

The **PORT NUMBER** box will list all of the available COM ports on the PC. The software will default to COM1, so select the port that will be connected to the development board. The **BAUD RATE**, **DATA BITS**, STOP BITS, and PARITY selection boxes allow the user to set the communication parameters. Click OK to apply any changes or CANCEL to exit out and return to the CONTROL screen with no changes.

The **CONTACT US** screen has all of the contact information for Linx Technologies. Click on one of the web links to open the Linx Technologies web site in the default web browser and the email link to send an email in Microsoft Outlook. Click **OK** to exit out and return to the **CONTROL** screen.

Figure 6: The Contact Us Screen

The **WINDOW** menu item has two options: **PIC CONTROL** and **RS232 CONTROL**. These options will toggle the **CONTROL** screen between the **PIC CONTROL** screen and the **RS232 CONTROL** screen.

The **HELP** menu item has three options: **ABOUT**, **HELP FILE**, and **CONTACT US**. The **ABOUT** screen displays information about the

Figure 7: The About Screen

software, as shown in Figure 7.
Click on one of the web links to open the Linx Technologies web site in the default web browser and the email link to send an email in Microsoft Outlook. Click **OK** to exit out and return to the **CONTROL** screen. Click on **SYSTEM INFO** to open the Windows System folder.

The **HELP FILE** option will display this document in either .pdf or .htm format. The **CONTACT US** option will display the **CONTACT US** screen described above.

A QS Module System Example

The rest of the manual will give some of the source code used in the development software. The application software was written in Microsoft Visual Basic 6.0 and the Programmer's Guide has examples of these functions written in C. Error handling and many of the references to objects on the forms are not shown to conserve space and reduce the possibility of confusion, but comments have been added where the user should put their own code. The Programmer's Guide should be used in addition to this document to aid in software development.

> Note: This code is provided as an example to aid our customers in their development and may or may not be appropriate for an individual application. This code is provided "As Is" and Linx Technologies makes no warrantees and assumes no liability for the use of this code.

The functions listed in this software are contained in the FTD2XX.dll file, so these functions must be declared before they can be used. All of these functions are described in the Programmer's Guide in detail. The file "QS VB Header File.txt" on the CD with the Master Development Kit Software contains all of the necessary function and constant declarations used by the software. This text file should be copied into a module in the Visual Basic project. The header file is also included in the Programmer's Guide and can be downloaded from the Linx website as a text file.

The first step in this system example is to determine if and how many devices are connected to the bus. This is done in C by using the **FT_List-Devices** function, though a difference in the way the Visual Basic variables are passed to the .dll has resulted in the creation of another function, **FT_GetNumDevices**. The code is as follows:

```
Dim lngStatus As Long
Dim lngNumDevices As Long'

' Get the number of devices attached to the bus
          lngStatus = FT_GetNumDevices (lngNumDevices, vbNullString, LIST_BY_NUMBER_ONLY)
          If lngStatus = OK Then
                    'The function was successful
          Else
                    'The function failed and the error code should be viewed to determine corrective action

          End If
```

LIST_BY_NUMBER_ONLY and **OK** are declared in the header file. If this function is successful, then the number of QS devices connected to the USB bus is stored in the variable **lngNumDevices**. If there is a problem, then the error code should be viewed so that the cause can be determined and corrected. This function can also be called periodically to check the bus for new devices or to see if old ones have been disconnected.

If devices are found, then the **FT_ListDevices** function can be used to get the device description and serial number.

```
Dim lngStatus As Long
Dim Index As Integer
Dim strDescription as String
Dim strSerialNumber as String

' Get the device description
          lngStatus = FT_ListDevices(Index, strDescription, LIST_BY_INDEX Or OPEN_BY_DESCRIPTION)
          If lngStatus = OK Then
                    'The function was successful and the device description is in strDescription
          Else
                    'The function failed and the error code should be viewed to determine corrective action
          End If

' Get the device serial number
          lngStatus = FT_ListDevices(Index, strSerialNumber, LIST_BY_INDEX Or OPEN_BY_SERIAL_NUMBER)
          If lngStatus = OK Then
                    'The function was successful and the device serial number is in strSerialNumber
          Else
                    'The function failed and the error code should be viewed to determine corrective action
          End If
```

Index is the number in which the bus located the device, starting with zero. If multiple devices are located on the bus then the above code can be used in a **For** loop to get the information for each device in turn, allowing the application software to decide which device to interface with. The **For** loop would be indexed by the number of devices returned by **FT_GetNumDevices**.

```
Dim arrDescription() As String * 256
Dim arrSerialNumber() As String * 256
Dim lngStatus As Long
Dim Index As Integer
Dim strDescription as String
Dim strSerialNumber as String
Dim lngNumDevices As Long

' Get the number of devices attached to the bus
          lngStatus = FT_GetNumDevices(lngNumDevices, vbNullString, LIST_BY_NUMBER_ONLY)
          If lngStatus = OK Then
                    ReDim arrDescription(lngNumDevices) As String * 256 'resize the arrays to the number of devices connected
                    ReDim arrSerialNumber(lngNumDevices) As String * 256
          End If

          For Index = 0 To (lngNumDevices - 1) Step 1
' Get the device description
          lngStatus = FT_ListDevices(Index, strDescription, LIST_BY_INDEX Or OPEN_BY_DESCRIPTION)
          If lngStatus = OK Then
                    arrDescription(Index) = strDescription
          End If

' Get the device serial number
          lngStatus = FT_ListDevices(Index, strSerialNumber, LIST_BY_INDEX Or OPEN_BY_SERIAL_NUMBER)
          If lngStatus = OK Then
                    arrNumber(Index) = strSerialNumber
          End If
Next Index
```

At this point, there are three ways of opening a communication channel with a device: by the index number, the description, or the serial number. The index number does not allow the application to open a specific named device, and if there are two devices with the same description on the bus then only the one with the lowest index number will be opened. The serial number is unique to each device and can be used to ensure that a specific device is opened. This example uses the serial number so refer to the Programmer's Guide for examples of the other two methods.

```
Dim lngStatus As Long
Dim strSerialNumber as String
Dim lngHandle As Long

' Open the device using the serial number
          lngStatus = FT_OpenEx(strSerialNumber, OPEN_BY_SERIAL_NUMBER, lngHandle)
          If lngStatus = OK Then
                    'The function was successful
          Else
                    'The function failed and the error code should be viewed to determine corrective action
          End If
```

This will open the device with the serial number that is contained in **strSerialNumber** and return a unique number in **lngHandle** that will be used by the other functions to communicate with the device. Once opened, the communications parameters should be set. The following function demonstrates the code to do this.

```
Dim lngStatus As Long
Dim lngHandle As Long
Dim sngUSB_Baud As Single
Dim intUSB_DataBits As Integer
Dim intUSB_Stops As Integer
Dim intUSB_Parity As Integer

sngUSB_Baud = 9600            '9600bps baud
intUSB_Stops = 0              'Stop Bits: 1 bit = 0, 1.5 bits = 1, 2 bits = 2
intUSB_DataBits = 8           'Data Bits: 7 or 8
intUSB_Parity = 0             'Parity: none = 0, odd = 1, even = 2, mark = 3, space = 4

Function SetUpUSB()
          ' Set baud rate
          lngStatus = FT_SetBaudRate(lngHandle, sngUSB_Baud)
          If lngStatus = OK Then
                    'The function was successful
          Else
                    'The function failed and the error code should be viewed to determine corrective action
          End If

          ' Set data bits, stop bits, and parity
          lngStatus = FT_SetDataCharacteristics(lngHandle, intUSB_DataBits, intUSB_Stops, intUSB_Parity)
          If lngStatus = OK Then
                    'The function was successful
          Else
                    'The function failed and the error code should be viewed to determine corrective action
          End If

          ' no flow control
          lngStatus = FT_SetFlowControl(lngHandle, FLOW_NONE, 0, 0)
          If lngStatus = OK Then
                    'The function was successful
          Else
                    'The function failed and the error code should be viewed to determine corrective action
          End If

          ' 5 second read timeout
          lngStatus = FT_SetTimeouts(lngHandle, 5000, 0)
          If lngStatus = OK Then
                    'The function was successful
          Else
                    'The function failed and the error code should be viewed to determine corrective action
          End If

End Function
```

Now that the device has been opened and the communication parameters have been set, the application can read and write data to the device. The code below will write the contents of the string **strWriteBuffer** to the device.

```
Dim lngStatus As Long
Dim lngHandle As Long
Dim strWriteBuffer As String
Dim lngTXAmt As Long
Dim lngBytesWritten As Long

' Write the data
          lngStatus = FT_Write(lngHandle, strWriteBuffer, lngTXAmt, lngBytesWritten)
          If lngStatus = OK Then
                    'The function was successful
          Else
                    'The function failed and the error code should be viewed to determine corrective action
          End If
```

The following code will read the data from the device and store it in **strReadBuffer**.

```
Dim lngStatus As Long
Dim lngHandle As Long
Dim strReadBuffer As String * 256
Dim lngRXBytes As Long
Dim lngBytesRead As Long

' Read the data
          lngStatus = FT_Read(lngHandle, strReadBuffer, lngRXBytes, lngBytesRead)
          If (lngStatus = OK) Then
                    'The function was successful
          Else
                    'The function failed and the error code should be viewed to determine corrective action
          End If
```

This will only read the data as soon as it is called, but will not be called automatically when there is data to be read. This means that the device must be periodically checked for data. The following code can be used with a timer to periodically check the receive buffer and the state of the modem control lines. In this example, the timer is set to check the device every 200mS, though a specific application may need more or less time.

```
Dim lngHandle As Long
Dim lngRXBytes As Long
Dim lngTXBytes As Long
Dim lngEvents As Long
' Set the timer to check the USB receive buffer
          tmrCheckRx.Interval = 200

Private Sub tmrCheckRx_Timer()
' Timer to periodically check the device for data
          If FT_GetStatus(lngHandle, lngRXBytes, lngTXBytes, lngEvents) = OK Then
                    If lngRXBytes > 0 Then
                              'The device has data, call the FT_Read function
                    End If
          End If

' Get the status of the modem lines
          lngStatus = FT_GetModemStatus(lngHandle, lngModemStatus)
          If (lngModemStatus And MODEM_STATUS_CTS) = MODEM_STATUS_CTS Then
                    'CTS is high
          Else
                    'CTS is low
          End If
          If (lngModemStatus And MODEM_STATUS_DSR) = MODEM_STATUS_DSR Then
                    'DSR is high
          Else
                    'DSR is low
          End If
          If (lngModemStatus And MODEM_STATUS_DCD) = MODEM_STATUS_DCD Then
                    'DCD is high
          Else
                    'DCD is low
          End If
          If (lngModemStatus And MODEM_STATUS_RI) = MODEM_STATUS_RI Then
                    'RI is high
          Else
                    'RI is low
          End If
End Sub
```

When the communication session has ended, close the channel to the device using the following code.

```
Dim lngHandle As Long

' Close the current device
          lngStatus = FT_Close(lngHandle)
          If (lngStatus = OK) Then
                    'The function was successful
          Else
                    'The function failed and the error code should be viewed to determine corrective action
          End If
```

The following function can be used to display the type of error should one occur.

```
Public strUSB_Error As String

Public Function ErrorCode(lngErrorNum As Long)
          Select Case lngErrorNum
                    Case 0
                              strUSB_Error = "No Error"
                    Case 1
                              strUSB_Error = "Invalid Handle"
                    Case 2
                              strUSB_Error = "Device Not Found"
                    Case 3
                              strUSB_Error = "Device Not Opened"
                    Case 4
                              strUSB_Error = "IO Error"
                    Case 5
                              strUSB_Error = "Insufficient Resources"
                    Case 6
                              strUSB_Error = "Invalid Parameter"
                    Case 7
                              strUSB_Error = "Invalid Baud Rate"
                    Case 8
                              strUSB_Error = "Device Not Opened For Erase"
                    Case 9
                              strUSB_Error = "Device Not Opened For Write"
                    Case 10
                              strUSB_Error = "Failed To Write Device"
                    Case 11
                              strUSB_Error = "EEPROM Read Failed"
                    Case 12
                              strUSB_Error = "EEPROM Write Failed"
                    Case 13
                              strUSB_Error = "EEPROM Erase Failed"
                    Case 14
                              strUSB_Error = "EEPROM Not Present"
                    Case 15
                              strUSB_Error = "EEPROM Not Programmed"
                    Case 16
                              strUSB_Error = "Invalid Args"
                    Case 17
                              strUSB_Error = "Other Error"
          End Select
End Function
```

Should one of the functions fail, **lngStatus** can be passed to this function and **strUSB_Error** can be displayed on the form to show which error occurred. For example, suppose a text box named **txtStatus** is placed on the form, then the code to open the device could be modified to display the results of the function call in the text box.

```
' Open the device using the serial number
          lngStatus = FT_OpenEx(strSerialNumber, OPEN_BY_SERIAL_NUMBER, lngHandle)
          If lngStatus = OK Then
                    txtStatus.Text = "No Error"
          Else
                    Call ErrorCode(lngStatus)
                    txtStatus.Text = "Open Failed: " & strUSB_Error
          End If
```

The other functions can be modified in a similar manner to suit the needs of the application. Please refer to the Programmer's Guide for a complete list of functions and more sample code.

The Microprocessor Code

The processor used on the evaluation board is the Microchip PIC16F88. The source code for this chip is written in C and compiled using the CCS PIC C Compiler (www.ccsinfo.com). Interfacing the microprocessor to the QS module is very straightforward thanks to the UARTs built into both devices. The processor is initialized to use the default RX and TX lines as shown in the following code.

```
#include <16F88.h>
#device ADC=8
#include <stdio.h>
#fuses NOWDT,INTRC_IO,NOMCLR
#use delay(clock=4000000)
#use rs232(baud=9600,xmit=PIN_B5,rcv=PIN_B2)
```

When a specific event occurs, PIC processors can jump out of the main program into a program written for that event. These events are called interrupts and must be defined in the code. The types of interrupts that can be created depends on the processor. The first interrupt used by the processor is for received data and is coded below.

```
char RxData;                    // Holds the received data character

//Interrupts when received data available
#int_rda
          void rda_isr(void)
          {
          RxData = getc();      //Get the character
          Receive(RxData);      //Call the Receive function
          }
```

When the processor receives a character, the interrupt is generated and the code above is called. The **getc()** function gets the data from the UART and the next line passes it to function **Receive**. This function would then take an action based on the received data, such as taking a line high to activate a LED. The second interrupt is for a timer that is used to generate a value from the ADC to move the **A/D Voltage Adjust** slider on the computer screen and also demonstrates how to send data to the QS module.

```
int CurrentCount;                               // Holds the current timer count
float OldValue;                                 // Holds the last voltage value sent

// Initialize the variables
CurrentCount = 10;
OldValue = 0;

// Interrupts when the clock overflows
#INT_RTCC
          void clock_isr(void)
                    {
                    if(--CurrentCount==0)         // Subtract 1 from count and if it is 0
                              {
                              GetADC(OldValue);   // Call the GetADC function
                              CurrentCount = 10;  // Reset the counter
                              }
                    }
```

When the internal clock reaches its maximum value then this code will be called. With a 4MHz clock this occurs about every 13mS and the counter will wait for the overrun to happen ten times before reading the ADC value, meaning the **GetADC** function is called about every 130mS. This function is shown below.

```
int ADCValue;                                   // Holds the value read by the ADC
float Ratio;                                    // Holds MaxVolt/MaxADC
float Voltage;                                  // Holds the voltage measured by the ADC

Ratio = 0.019608;                               // = 5/255

void GetADC(float LastValue)
          {
          ADCValue = Read_ADC();                // Get the ADC value
          Voltage = (ADCValue * Ratio);         // Multiply by ratio to convert to voltage
          if (Voltage != LastValue)             // if it is not equal to the last value
                    {
                    printf("%01.2f", Voltage);  // Send the new value
                    OldValue = Voltage;         // Save the new value
                    }
          }
```

The ADC will return a value of 0 for a voltage of 0V and a value of 255 for a voltage of Vcc, in this case 5V. The ratio of 5/255 will convert the ADC value into the actual voltage measured on the pin. This value is compared to the previous reading from the ADC and, if different, output to the QS module with the printf() statement. This comparison helps to keep the bus quiet by only sending the value when it has changed rather than sending the same value repeatedly every 130mS.

```
char Data;                                      // Holds the last character sent by Button 1
Data1 = 'h';                                    // Initialize the button to off

void CheckButtons(void)
          {
          if(INPUT(PIN_B6))                     //If the button was pressed
                    {
                    if(Data=='h')               //If the last character sent was h (Button off)
                              {
                              putc('g');        //send g (Button on)
                              Data='g';         //update the last character sent
                              }
                    else if(Data=='g') //If the last character sent was g (Button on)
                              {
                              putc('h');        //send h (Button off)
                              Data='h';         //update the last character sent
                              }
                    while(INPUT(PIN_B6))        //Wait for the line to go low again
                              {
                              delay_ms(1);
                              }
                              delay_ms(100);    //Debounce wait
                    }
                                        }
```

The button press is detected by looking for a change on the button line. The button on the board is momentary so the software is used to latch the indicator in the application software. When a button press is detected, the software check what was sent the last time and sends the opposite command. It then waits for the line to go low to prevent the light from constantly switching while the button is held down. Finally, a 100mS debounce wait will prevent any noise from the button from causing the PIC to recognize multiple presses.

The opening of the main function sets the clock speed, enables the ADC and interrupts, and initializes the outputs.

```
void main(void)
          {
          //Setup chip and initialize interrupts
          setup_oscillator(OSC_4MHZ);
          enable_interrupts(global);
          enable_interrupts(int_rda);
          setup_adc_ports(sAN0);
          SETUP_ADC(ADC_CLOCK_INTERNAL);
          set_adc_channel(0);
          set_rtcc(0);
          setup_counters(RTCC_INTERNAL, RTCC_DIV_256);
          enable_interrupts(INT_RTCC);

          //Initialize LED outputs to low
          OUTPUT_LOW(PIN_A2);
          OUTPUT_LOW(PIN_A3);
          OUTPUT_LOW(PIN_A4);
          }
```


159 Ort Lane, Merlin, OR, US 97532
Phone: +1 541 471 6256
Fax: +1 541 471 6251
www.linxtechnologies.com