ZBasic

Application Note

AN-214 Interfacing with the LCD03 Text Display

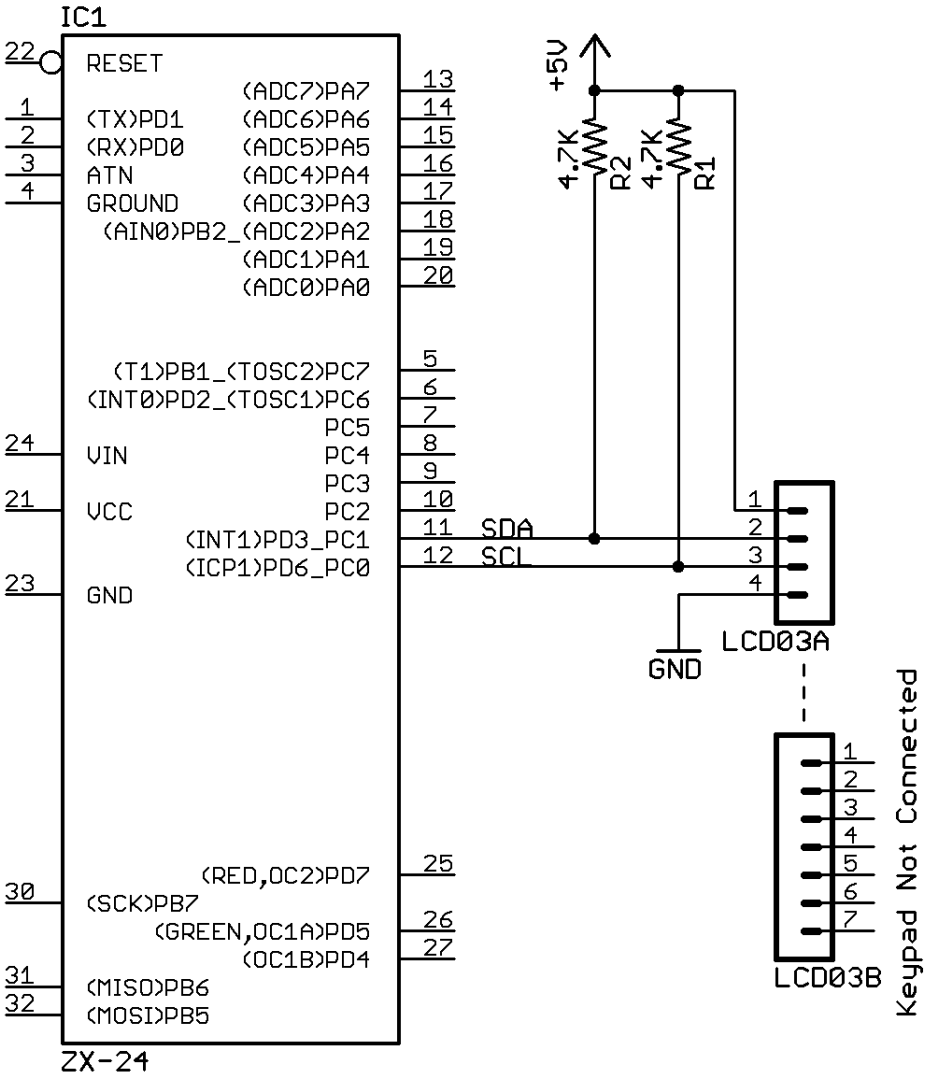
Introduction

This application note describes how to connect and use the Devantech LCD03 text display (see http://www.robot-electronics.co.uk/shop/I2C_Serial_Display_LCD032058.htm). This device is a standard 20*4 LCD text display with a PIC microcontroller chip that provides both a serial and I2C interface. The PIC provides cursor and text formatting functions, custom characters, keypad connectivity, and a 64 byte FIFO buffer to ensure a minimum of delay in writing to the display.

Hardware Hookup

For this application the ZX chip is connected to the LCD03 display using the I2C serial protocol. The I2C SDA and SCL pins are connected to the ZX hardware-based I2C on the ZX-24 pins 11 and 12 respectively. For a ZX-40 these are pins 23 and 22, and for a ZX-44 the corresponding pins are 20 and 19. Don't forget the pullup resistors on the SDA and SCL pins – any value between 1.8K and 6.8K should be sufficient.

Note that with a ZX-24, the standard SDA/SCL pins overlap with interrupt 1 and input capture. The other alternative of using the software-based I2C "uses up" the Timer1 resource. This resource issue is much less of a problem with ZX-40 and ZX-44.



Published March 2006

Software

This application note also comes with some ZBasic software. The file LCD03.bas is an interface module for the LCD03 display and the file AN214.bas is a test program for the display. Because the display supports a relatively large number of functions, this application notes first describes the public interface provided by the LCD03.bas module and then has additional sections to describe some of the special implementation features. All the source code can be found in the zip file associated with this application note.

The sixteen public functions implemented by LCD03.bas module are shown in the table below. Note that the LCD03 device does not support text scrolling or a block cursor.

Name	Description	Parameters and any Result
LCDBacklight	Turns the backlight on and off	backlight as Boolean
LCDBackspace()	Deletes the preceding character on the display	None
LCDClearScreen()	Erases the screen	None
LCDCursorHome()	Moves the cursor to top left home position	None
LCDDefineCustomCharacter()	Defines one of 6 custom characters	char As Byte bitMap() As Byte
LCDGetVersion()	Gets the version number for the PIC microcontroller software	returns byte
LCDInitialize()	Initializes the I2C interface and resets the LCD display	checkBuffer as Boolean
LCDMoveCursor()	Moves cursor to specified row and column	row as Byte col as Byte
LCDSetCursorStyle()	Sets the cursor style to be hidden, blinking, or underline	style As LCDCursorStyle (see below)
LCDSetPosition()	Sets the cursor absolute position on the display	pos as Byte
LCDSmartLinefeed()	Moves the cursor down one line in the same column	None
LCDTabCursor()	Moves the cursor right one tab	None
LCDVerticalTabCursor()	Moves the cursor down one line	None
LCDTabSize()	Sets the horizontal tab size in characters	size as Byte
LCDTerminate()	Stops using the LCD display	None
LCDWriteString	Writes a string at the current cursor position	text as String

The public interface provided by LCD03.bas also includes the following constants and an enumeration for LCDCursorStyle:

```
' Display is 20 columns x 4 rows.
Public Const LCDMaxColumn As Byte = 20
Public Const LCDMaxRow As Byte = 4

' Custom characters are in the range 128 to 133
Public Const LCDFirstCustomChar as Byte = 128
Public Const LCDLastCustomChar as Byte = 133

' Enumeration for SetLCDCursorStyle
Public Enum LCDCursorStyle
    HiddenCursor    = 4
    UnderlineCursor = 5
    BlinkingCursor  = 6
End Enum
```

Initializing and Terminating use of the LCD Display

The `LCDInitialize()` subroutine is used to initialize the I2C channel for the LCD display. The I2C channel is hardcoded to be channel 0 which uses the underlying hardware-based I2C support. This was a deliberate design decision so that the Timer1 resource is available for other uses. Note that the LCD03 device does not work reliably above a 100 KHz I2C clock. The initialization routine also configures the LCD display, clears the display, moves the cursor to the home position and turns off the backlight. In addition it defines two custom characters that are used for

"Writing Text to the Display". The corresponding `LCDTerminate()` subroutine is used to close the I2C channel and turn off the backlight.

Positioning the Cursor

A number of the routines in the public interface are used to position the cursor is different ways such as `LCDBackspace()`, `LCDCursorHome()`, `LCDMoveCursor()`, `LCDSetPosition()`, `LCDSmartLinefeed()`, `LCDTabCursor()`, and `LCDVerticalTabCursor()`. These routines reflect underlying command codes that can be sent to the PIC microcontroller on the LCD device.

A common private function called `sendCmd()` is used to send these one byte commands to the LCD display. Here is an example of its invocation and implementation:

```
Public Sub LCDNextLine()
   Call lcdCmd(NEXT_LINE)
End Sub

Private Sub lcdCmd(ByVal command as Byte)
   Dim cmd(1 to 2) as Byte
   Dim rc as Integer

   cmd(1) = REG_CMD
   cmd(2) = command
   rc = cmdWait(CByte(SizeOf(cmd)), cmd)
   If rc < 0 Then
       Debug.Print "Command "; CStr(command); " returned ";CStr(rc)
   End if
End Sub
```

The implementation of these functions is rather simple and could be simplified by making `lcdCmd()` a public routine. However the idea is to provide a level of information hiding so that the LCD interface or even LCD itself could be changed without affecting the calling application code. The private function `cmdWait()` is explained later.

Writing Text to the LCD Display

The subroutine `LCDWriteString()` is used to write text to the LCD display. The text sent to the display needs to be prefixed by the I2C command register. Rather than allocate a buffer of up to 65 bytes, a smaller buffer is chosen and the text string separated into chunks. The private constant `BUFFER_SIZE` is used to control the chunk size and is set to 8 in the supplied code.

The LCD displays the incorrect symbols for backslash (\) and tilde (~) so `LCDWriteString()` remaps these character codes to two custom character codes so that the correct symbol is displayed for backslash and tilde.

```
Public Sub LCDWriteString(ByVal text as String)
   Dim buffer(1 to BUFFER_SIZE+1) as Byte
   Dim i as Integer, rc as Integer
   Dim length As Byte, char as Byte

   ' initialize standard part of buffer
   buffer(1) = REG_CMD

   ' break up string into parts rather than send one long command
   ' each part is BUFFER_SIZE characters long
   i = 0
   length = 1
   Do While i < len(text)
       ' next character
       i = i + 1
       length = length + 1

       ' translate character for two special cases \ and ~
       char = asc(text, i)
       If char = ASCIITilde Then
           char = LCDtilde
```

```
        ElseIf char = ASCIIbackslash Then
            char = LCDbackslash
        End If

        ' add character to buffer and see if it is time to output the string
        buffer(length) = char
        If (length = BUFFER_SIZE+1) Or (i = len(text)) Then
            rc = sendCmd(length, buffer)
            If rc < 0 Then
                Debug.Print "LCDWriteString i2cmd returned ";CStr(rc)
            End if

            ' reset the length for the next buffer of characters
            length = 1
        End If
    Loop
End Sub
```

Defining Custom Symbols

The public routine `LCDDefineCustomCharacter()` supports defining up to 6 custom symbols (characters). See the online documentation (http://www.robot-electronics.co.uk/shop/I2C_Serial_Display_LCD032058.htm) for an explanation of how to define custom characters.

The `LCDDefineCustomCharacter()` routine shown below checks for a valid character code and then calls an internal routine to define the character. The reason for the internal routine is so that it can be used by `LCDInitialize()` without going through the check for valid character codes.

```
Public Sub LCDDefineCustomCharacter(ByVal code As Byte, ByRef bitMap() As Byte)
    ' validate the custom character code
    If (code < LCDFirstCustomChar) Or (code > LCDLastCustomChar) Then
        Debug.Print "LCDDefineCustomCharacter invalid character code"
        Exit Sub
    End If

    ' call the internal routine that does the work
    Call defineCharacter(code, bitmap)
End Sub
```

The code for private `defineCharacter()` subroutine is not described further. See the attached zip file for details.

Overflowing the Command Buffer

All of routines that send commands to the LCD03 device use a common internal function called `sendCmd()` shown below. This function can optionally check the size of the LCD FIFO buffer and wait until there is enough space before sending the command. The Boolean parameter for `LCDInitialize()` is used to control if the buffer check is performed by `sendCmd()`.

```
Private Function sendCmd(ByVal writeCnt as Byte, ByRef writeData() as Byte) as Integer
    Dim rc as Integer
    Dim reg as Byte, freeSpace as Byte
    reg = REG_FREE

    ' check if buffer too big to even fit
    If writeCnt > MAX_FIFO_SIZE Then
        Debug.Print "Too much data for LCD03"
        sendCmd = -1
        Exit Function
    End If

    ' do buffer check if needed
    If needBufferCheck Then
        ' check to see if there is enough room in the FIFO buffer for the command
        Do
            rc = I2CCmd(channel, ADDRESS, CByte(SizeOf(reg)), reg, 1, freeSpace)
```

```
            If rc < 1 Then
                Debug.Print "Get FIFO buffer bytes free count i2cmd returned ";CStr(rc)
                freeSpace = 0
            End If
            'Debug.Print "Sizeof free buffer is ";CStr(freeSpace)

            If freeSpace < writeCnt Then
                Call Sleep(delayTime)
            End If
        Loop Until freeSpace >= writeCnt
    End If

    ' now send the command to the LCD display
    sendCmd = I2CCmd(channel, ADDRESS, writeCnt, writeData, 0, 0)
End Function
```

So far in my testing I have not ever had to wait for the LCD03 device. It seems to be able to keep up with the I2C datastream. However I decided to keep this code to show how it could be done. If desired you could remove it in your own implementation.

Using the LCD Display Interface

Here is some code extracted from AN214.bas that demonstrates how to use the public interface implemented by LCD03.bas:

```
Sub Main()
    ' start test
    Call LCDInitialize(false)

    ' write test string and demonstrate tilde and backslash character remapping
    Call LCDTabSize(4)
    Call LCDMoveCursor(2, 5)
    Call LCDVerticalTabCursor()
    Call LCDTabCursor()
    Call LCDSmartLineFeed()
    Call LCDBackSpace()
    Call LCDWriteString("~Test\String")
    Call LCDSetPosition(61)
    Call LCDWriteString("Software Version: " & CStr(LCDGetVersion()))
    Call LCDSetCursorStyle(UnderlineCursor)
End Sub
```

Other Considerations

This application note uses I2C to communicate with the LCD03 display. I have not tested the serial connection but I believe it should be possible to replace the implementation of the public interface with one that uses the serial port. I have not implemented or tested the use of a keypad attached to the LCD03 device.

Author

Mike Perks is a professional software engineer who became interested in microcontrollers a few years ago. Mike has written a number of articles, projects and application notes related to ZBasic, BasicX and AVR microcontrollers. Mike is also the owner of Oak Micros which specializes in AVR-based devices including his own ZX-based products. You may contact Mike at mikep@oakmicros.com or visit his website http://oakmicros.com.