ZBasic

Application Note

AN-208 Using I2C with Devantech Ultrasonic Range Finders

Introduction

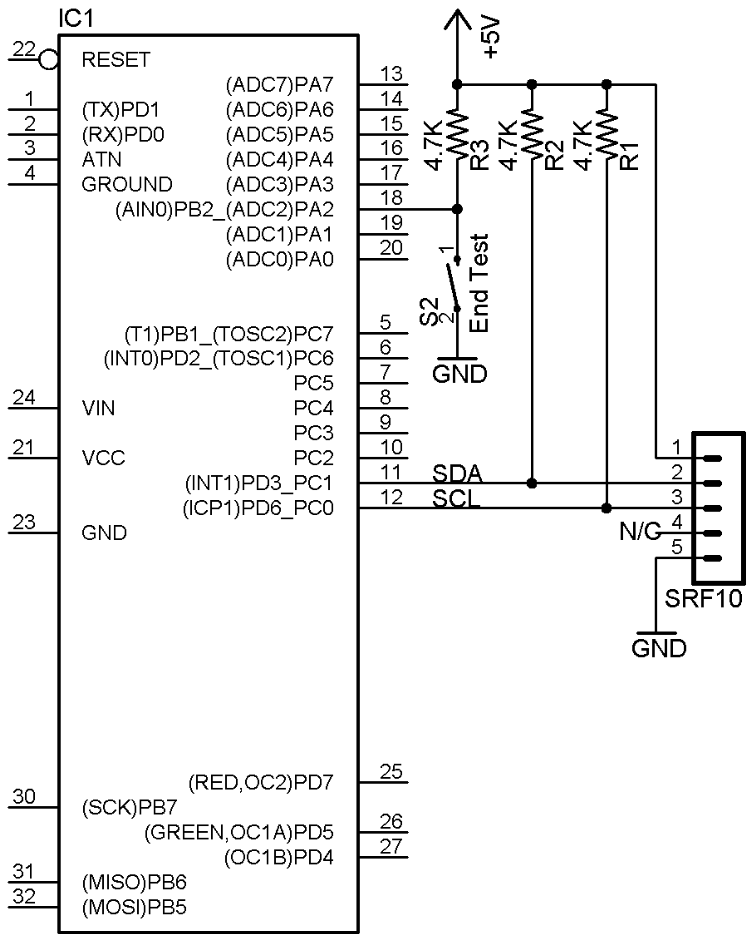
This application note describes how to connect and use the Devantech SRF10 Ultrasonic Range Finder (http://www.robot-electronics.co.uk/htm/srf10tech.htm). Although not tested the software in the zip file with this application note should also work with the SRF08 Ultrasonic Range Finder as well.

Ultrasonic range finders such as the SRF10 are typically used in robots for collision avoidance or object detection. The transmitter transducer sends out a 40 KHz ultrasonic burst that is then reflected off any nearby objects and picked up by the receiver transducer. The internal software uses the time delay and the speed of sound in air to calculate the distance of the object that reflected the ultrasonic burst. Connection to these devices is through 2 wires for power and 2 wires for I2C SDA (data) and SCL (clock). An embedded microcontroller handles the I2C slave interface and ultrasound calculations. These devices are somewhat expensive but they do save a lot of work.

Hardware Hookup

Connecting a ZX chip to the Devantech Ultrasonic Range Finder is very simple as shown below for the ZX-24. 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.



Software

This application note also comes with some ZBasic software. The file SRF.bas is an interface module for the SRF10 and SRF08 and the file AN208.bas is a test program for the SRF.bas module.

The public interface implemented by the SRF.bas module consists of constants for SRF10 and SRF08 configuration and some public routines named InitSRF(), TermSRF(), GetSRFRange(), GetSRFVersion(), and SetSRFAddress().

Because ultrasonic ranging takes a significant time (65 milliseconds), a fundamental design element of the SRF module is that the ranging is done in a separate task. This allows time-critical operations such as a robot control task to continue processing other sensor input and then request the distance from the SRF on demand. This is what the GetSRFRange() function does and is shown below.

```
Public Function GetSRFRange() as UnsignedInteger
   If range = 0 Then
       GetSRFRange = maximumRange
   Else
       GetSRFRange = range
   End If
End Function
```

The internal private variable named **range** holds the last range value received from the SRF10 or SRF08. Note that a range of zero (finger over the transducer) looks exactly the same as out of range and is returned as the maximum range.

The separate task routine called doRanging() that does all the work to set the private variable **range** is listed below. This routine is a never-ending loop that starts a ranging command, waits for 65 milliseconds and then retrieves the ranging result in an unsigned integer, a byte at a time.

```
Private Sub doRanging()
   Dim cmd(1 to 2) as Byte
   Dim reg as Byte
   Dim rc as Integer

   Do
       ' start ranging
       cmd(1) = REG_CMD
       cmd(2) = rangeMode
       rc = I2CCmd(channel, address, CByte(SizeOf(cmd)), cmd, 0, 0)
       If rc < 0 Then
           Debug.Print "GetRange start i2cmd returned ";CStr(rc)
       End if

       ' sleep while waiting for the ranging to complete
       Call Sleep(RANGING_WAIT_TIME)

       reg = REG_RANGE_HIGH
       rc = I2CCmd(channel, address, 1, reg, 2, cmd)
       If rc = 2 Then
           ' the MS byte is returned in the first byte, LS byte in the second
           range = MakeWord(cmd(2), cmd(1))
       Else
           Debug.Print "GetRange i2cmd returned ";CStr(rc)
       End If
   Loop
End Sub
```

The InitSRF() subroutine is used to initialize the I2C channel, configure the SRF and start the doRanging task as shown in the source code below. 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. Although not shown, the highest bit rate of 410KHz works with the SRF10. The I2C slave address parameter is stored for use by the doRanging() task.

Because it is not possible to configure the gain and maximum ranging distance while the SRF10 is ranging, this is done by the InitSRF() subroutine using the additional 3 parameters. The required range (parameter 2) is given as a value in the units of parameter 3 (inches or centimeters). Public constants are defined for units (inches or centimeters) and different gain values. The two private routines (not shown here) of setSRFGain() and setSRFMaxRange() set the internal configuration of the SRF10 or SRF08 before the ranging task is started.

```
' ***************************************************************
' Sub InitSRF
'
' Initialize the SRF10/SRF08 by configuring it and creating a new task
' Parm1 is the I2C address of the SRF
' Parm2 is the required maximum range in inches or cmd
' Parm3 is the units for parm2 (inches or cm)
' Parm4 is the gain setting for the SRF - see SRFxx_GAIN_xxx constants
'***************************************************************
Public Sub InitSRF(ByVal addr as Byte, ByVal reqRange as UnsignedInteger, _
                   ByVal units as Byte, ByVal gain as Byte)
   ' save data for later use
   address = addr

   ' open the I2C channel and configure the gain
   Call OpenI2C(channel, sdaPin, sclPin)
   Call setSRFGain(addr, gain)

   ' initialize the maximum range and start the range task
   maximumRange = setSRFMaxRange(addr, reqRange, units)
   range = 0
   CallTask "doRanging", rangeStack
   Call Sleep(0.1)
End Sub
```

Use of the ultrasonic range finder can be terminated using the TermSRF() subroutine shown below. This subroutine terminates the task and closes the I2C channel.

```
Public Sub TermSRF()
   Call ExitTask(rangeStack)
   Call CloseI2C(channel)
End Sub
```

The public constants and two other public routines are not shown here but are in the associated zip file. The public function GetSRFVersion() returns the software revision for the SRF10 or SRF08 device. The public function SetSRFAddress() is used to set a new I2C slave address for the SRF10 or SRF08 device and is normally only used once for each device. A potential enhancement to this SRF module is to support multiple SRF devices using arrays for the slave addresses and ranges. This enhancement would require changes to InitSRF(), GetSRFRange(), and doRanging().

Using the Range Finder

The example test program (AN208.bas) below shows how to invoke the interface to the SRF module described previously.

```
Private Const addr As Byte = &HE0    ' default I2C Address
Private Const stopPin as Byte = A.2 ' just used for testing purposes

Sub Main()
   ' start test
   Debug.Print "Start of SRF test"
   Call Sleep(0.25) ' wait for SRF to wakeup

   ' initialize SRF and get software version number
   Call InitSRF(addr, 48, SRF_UNITS_INCHES, SRF10_DEFAULT_GAIN)
   Debug.Print "Software Version: ";CStr(GetSRFVersion(addr))

   ' main test loop which gets the range in inches every second
   ' until the stop button is pressed
   Do While GetPin(stopPin) = 1
       Debug.print "Range ";CStr(GetSRFRange(addr)); " inches"
       Call Sleep(1.0)
   Loop

   ' stop using the SRF
   Call TermSRF()
   Debug.Print "SRF test finished"
End Sub
```

Here is the console output from the above program showing how the range changes as a hand is moved over the SRF10 device until the stop test button is pressed. Although the range is only displayed every second, the LED on the SRF device is flashing approximately 15 times a second, once for each ranging request from the doRanging() task.

```
ZBasic v1.1
Start of SRF test
Software Version: 5
Range 49 inches
Range 2 inches
Range 6 inches
Range 13 inches
Range 49 inches
Range 8 inches
SRF test finished
```

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.

e-mail: support@zbasic.net **Web Site:** http://www.zbasic.net