ZBasic

Application Note

AN-212 Getting Hot with the TPA81 Thermal Sensor

Introduction

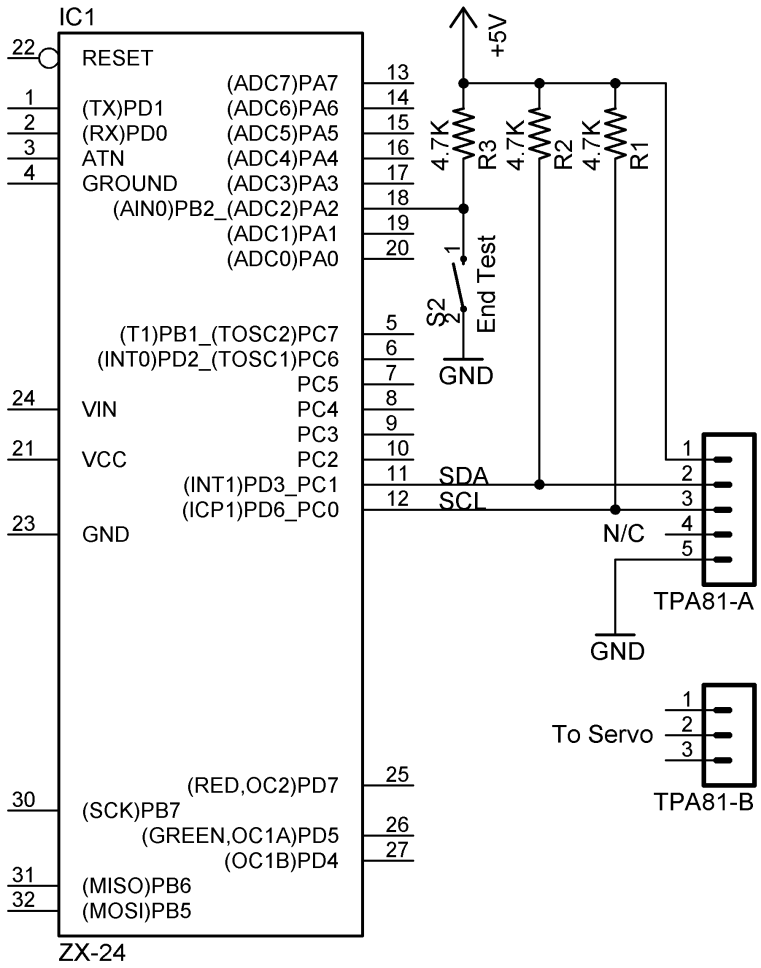
This application note describes how to connect and use the Devantech TPA81 Thermal Sensor (see http://www.robot-electronics.co.uk/htm/tpa81tech.htm). This device contains an array of eight infrared thermometers arranged in a row and can return the temperatures measured by each thermometer together with the ambient room temperature. The TPA81 can also control a servo to pan the module which can be used to build a thermal image. An on-board PIC microcontroller calculates the temperatures and provides an I2C control interface.

The TPA81 device can also be used to measure temperatures colder than room temperature but this was not verified in the production of this application note.

Hardware Hookup

The ZX chip is connected to the TPA81 Thermal Sensor 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 January 2006

Software

This application note also comes with some ZBasic software. The file TPA81.bas is an interface module for the TPA81 Thermal Sensor and the file AN212.bas is a test program for the TPA81.bas module. The public interface implemented by the TPA81.bas module consists of nine public routines named `InitTPA81()`, `TermTPA81()`, `GetTPA81AmbientTemperature()`, `GetTPA81Temperatures()`, `GetTPA81TemperatureFlags()`, `GetTPA81Version()`, `SetTPA81ServoPosition()`, `SetTPA81ServoRange()`, and `SetTPA81Address()`.

The `InitTPA81()` subroutine is used to initialize the I2C channel 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. The TPA81 device works at ZBasic I2C default speed of 100Khz but does not work at maximum supported speed of 410 KHz. The second parameter is used to set the temperature units of Centigrade or Fahrenheit that is used by the other routines.

```
Public Const TPA81_UNITS_CENTIGRADE as Byte         = 80
Public Const TPA81_UNITS_FAHRENHEIT as Byte         = 81

Public Sub InitTPA81(ByVal addr as Byte, ByVal units as Byte)
   ' save data for later use
   address = addr
   temperatureUnits = units
   ' open the I2C channel
   Call OpenI2C(channel, sdaPin, sclPin)
End Sub
```

The corresponding subroutine `TermTPA81()` is used to stop using the I2C channel for the TPA81 device. The source code for this subroutine can be found in the zip file associated with this application note.

The function `GetTPA81AmbientTemperature()` can be used to retrieve the ambient temperature around the TPA81 device and is used to determine if a significant heat source is being detected by the TPA81. An extract of the source code is shown below including the conversion to Fahrenheit.

```
Public Function GetTPA81AmbientTemperature() as Byte
   Dim rc as Integer
   Dim reg as Byte
   reg = REG_AMBIENT
   rc = I2CCmd(channel, address, CByte(SizeOf(reg)), reg, 1, GetTPA81AmbientTemperature)

   If temperatureUnits = TPA81_UNITS_FAHRENHEIT Then
       GetTPA81AmbientTemperature = CByte(CSng(GetTPA81AmbientTemperature) * 1.8 + 32.0)
   End If
End Function
```

The subroutine `GetTPA81Temperatures()` is the heart of the TPA81 interface. This subroutine (not function) takes a single parameter which is a reference to an array of 8 bytes to hold the temperature detected by the TPA81 device. The source code including the conversion to Fahrenheit is shown below. The I2CCmd ZBasic system function handles all the work of retrieving the 8 bytes from the TPA81 device.

```
Public Sub GetTPA81Temperatures(ByRef temps() as Byte)
   Dim rc as Integer
   Dim reg as Byte

   reg = REG_TEMP
   rc = I2CCmd(channel, address, CByte(SizeOf(reg)), reg, 8, temps)

   If temperatureUnits = TPA81_UNITS_FAHRENHEIT Then
       Dim i as Byte
       For i=1 to 8
           temps(i) = CByte(CSng(temps(i)) * 1.8 + 32.0)
       Next
   End If
End Sub
```


Published January 2006

Sometimes the actual temperatures are not needed and it is useful just to find out if there is a hot (or cold) source nearby. This is done using the `GetTPA81TemperatureFlags()` function and can be used to detect a human, animal or other heat source. This function takes a delta temperature parameter in the units used by InitTPA81 and sets bits in a single byte for each thermometer when the temperature difference is greater than the delta. The source code below shows how this works using the *bit array* and *alias* features of the ZBasic language.

```
Public Function GetTPA81TemperatureFlags(ByVal Delta as Byte) as Byte
   Dim t(1 to 8) as Byte, i as Byte, ambient as Byte

   ' setup an array of 8 bits to alias the returned byte
   Dim b(1 to 8) as Bit Alias GetTPA81TemperatureFlags

   ' retrieve the temperatures and ambient temperature
   Call GetTPA81Temperatures(t)
   ambient = GetTPA81AmbientTemperature()

   GetTPA81TemperatureFlags = 0
   For i=1 to 8
       ' set the bit flag if the temperature difference exceeds the delta
       If Abs(CInt(t(i)) - CInt(ambient)) > CInt(Delta) Then
           b(i) = 1
       End If
   Next
End Function
```

Taking the Temperature

An extract of the example test program (AN212.bas) below shows how to invoke the interface to the TPA81 device as described previously. The `GetTPA81Version()` function is used to get the software version for the TPA81 device and source can be found in the associated zip file.

```
Private Const addr As Byte = &HD0   ' default I2C Address
Private Const stopPin as Byte = A.2 ' just used for testing purposes
Private temps(1 to 8) as Byte

Sub Main()
   Dim i as Byte

   ' start test
   Debug.Print "Start of TPA81 test"
   Call Sleep(0.25) ' wait for TPA81 to wakeup

   ' initialize TPA81 and get software version number
   Call InitTPA81(addr, TPA81_UNITS_FAHRENHEIT)
   Debug.Print "Software Version: ";CStr(GetTPA81Version(addr))

   ' main test loop which gets the temperature readings every second
   ' until the stop button is pressed
   Do While GetPin(stopPin) = 1
       ' Print out the ambient temperature and hex value of the temperature flags
       Debug.Print "Ambient Temperature: ";CStr(GetTPA81AmbientTemperature());
       Debug.Print " Flags: "; CStrHex(GetTPA81TemperatureFlags(25));     ' delta is 25F

       ' Print out the temperature array in a row
       Call GetTPA81Temperatures(temps)
       Debug.Print " Temperatures:";
       For i = 1 to 8
           Debug.Print " ";CStr(temps(i));
       Next
       Debug.Print
       Call Sleep(1.0)
   Loop
   ' stop using the TPA81
   Call TermTPA81()
   Debug.Print "TPA81 test finished"
End Sub
```

The output of the test program is shown below. The temperature flags were triggered at 25 degrees Fahrenheit above the ambient temperature by placing a desk lamp light bulb near the TPA81 device. In real application this could be a heat source such as a fire for a fire-fighting robot.

```
Start of TPA81 test
Software Version: 7
Ambient Temperature: 72 Flags: 00 Temperatures: 97 97 95 97 95 97 95 97
Ambient Temperature: 72 Flags: 00 Temperatures: 97 97 95 95 93 95 95 97
Ambient Temperature: 72 Flags: 8f Temperatures: 100 100 99 99 97 97 97 99
Ambient Temperature: 72 Flags: ff Temperatures: 108 106 104 102 102 102 100 102
Ambient Temperature: 73 Flags: ff Temperatures: 115 115 113 109 108 108 106 111
Ambient Temperature: 73 Flags: ff Temperatures: 120 118 117 115 111 111 109 113
Ambient Temperature: 73 Flags: ff Temperatures: 124 122 118 117 115 113 115 118
Ambient Temperature: 73 Flags: ff Temperatures: 117 117 113 113 109 109 109 109
Ambient Temperature: 73 Flags: 7f Temperatures: 111 111 109 108 106 104 99 93
TPA81 test finished
```

Using the TPA81 Servo Support

The TPA81 device also supports driving a servo which can be used to rotate the TPA81 into position to take temperatures. The `SetTPA81ServoPosition()` subroutine as shown below is used to position the servo in one of 32 different positions. The `sendCmd()` function is a private helper routine which also appears in some other I2C application notes.

```
Public Sub SetTPA81ServoPosition(ByVal pos as Byte)
   If pos < 0 or pos > 31 Then
       Debug.Print "Invalid position parameter for SetTPA81ServoPosition"
   Else
       If sendCmd(address, REG_CMD, pos) < 0 Then
           Debug.Print "SetTPA81ServoPosition i2cmd returned error"
       End If
   End If
End Sub
```

The range of the servo stepping in microseconds can also be changed using the `SetTPA81ServoRange()` subroutine. The two parameters to the subroutine specify the servo range in microseconds around the center point of 1500us. The `SetTPA81ServoRange()` routine then calculates and sets the required servo range value for the TPA81 device as shown in the code below.

```
Public Sub SetTPA81ServoRange(ByVal range1 as Integer, ByVal range2 as Integer)
   Dim range as Byte

   range = CByte(min(abs(range1-RANGE_MIDPOINT), abs(range2-RANGE_MIDPOINT)) * 4 \ 31)
   If sendCmd(address, REG_RANGE, range) < 0 Then
       Debug.Print "SetTPA81ServoRange i2cmd returned error"
   End If
End Sub
```

Here is an extract of the AN212.bas that illustrates how to use these servo routines. The test function `findHotSpot()` is used to move the servo to each position and calculate which one is the hottest.

```
Sub Main()
   …
   ' scan area using servo to get hottest spot
   ' and move servo to that position for rest of measurements
   Debug.Print "Hottest position is "; CStr(findHotSpot())
   …
End Sub


Private Function findHotSpot() as Byte
   Dim maxHeat as Integer, heat as Integer
   Dim i as Byte, p as Byte

   ' set the servo range - this is particular to a servo
```

```
    SetTPA81ServoRange(50)

    ' get the temperature for each of the 32 servo positions
    maxHeat = 0
    findHotSpot = 0
    For p=0 to 31
        ' set the position and wait 40ms for the TPA81 sensor
        Call SetTPA81ServoPosition(p)
        Call Sleep(0.04)

        ' get the temperature and calculate how hot it is by simply
        ' aggregating the eight temperatures
        Call GetTPA81Temperatures(temps)
        heat = 0
        For i= 1 to 8
            heat = CInt(temps(i)) + heat
        Next

        ' see if this is the "hottest" position so far
        If heat > maxHeat Then
            maxHeat = heat
            findHotSpot = p
        End If
    Next

    ' move the servo to the hottest position
    Call SetTPA81ServoPosition(findHotSpot)
End Function
```

When the test program is run, the servo moves to each position, calculates the hottest spot and then moves the servo to that position. From that point on the test program continues to measure temperatures until the stop button is pressed. The resulting output from the test program is the single additional print line that may look like this:

```
Hottest position is 29
```

Multiple TPA81 Device Support

An alternative to moving the sensor on a servo is to measure temperatures using more than one TPA81 device. Up to 8 are supported, each with a different I2C address. The `SetTPA81Address()` function is used to change the address of a particular TPA81 device and only needs to be done once per device. The code for `SetTPA81Address()` is the same as that for `SetSRFAddress()` in application note AN208 and is not repeated here for brevity.

Note that the implementation of the TPA81 interface routines presented in this application note will need to be updated to support multiple TPA81 devices. This could be done by supporting the I2C address parameter for every routine or by gathering the temperature data for all the devices at once.

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.