midon design

Last Updated 01/12/30

Frequently Asked Questions - TEMP05

---

1. What is Parasitic Power?

Parasitic Power is a term developed by Dallas Semiconductor to describe their method of powering their "One-Wire" devices. The DS18S20 sensor used by the TEMP05 is one of these One-Wire devices. As the term implies, there is only one wire (well, 2 really. A ground is also required) used to communicate on a bus with all devices. The power is derived from the communication line. Whenever a logic "1" is sent on the One-Wire bus, all devices on the bus use that logic "1" to charge up their internal circuitry. Providing that a logic "1" is sent often enough, that charge is sufficient to drive the individual devices. Fortunately, the idle state of the bus is a logic 1, so, statistically, a logic 1 is present very often.

2. Do I need a Dallas Weather Station to use the TEMP05?

No. TEMP05 can be used with any combination of the following devices on the One-Wire bus;

- Dallas One-Wire Weather Station (one only)
- DS18S20 sensors
- DS1820 sensors
- Dallas Rain Gauge (one only)
- Dallas Humidity Sensor
- None of the above (why would I do that? TEMP05 can also be used to drive a companion device called RELAY05 without needing any sensors)

3. How do I connect the temperature sensors to the TEMP05?

The sensors are connected via twisted pair wire, typically CAT-5 or better, to connector J2. The external wire can be "home run" (all sensors connected on their own wire and all end-points tied together), or daisy-chained (one, or more, wire runs are used and each sensor is "tapped" into the wire run where needed). The sensors themselves are then soldered on to the wire run wherever they are needed. If you don't like soldering these devices, some people have had success with crimping the sensors into RJ-11 plugs and running wires to RJ-11 sockets where the sensors are needed. Midon Design also provides small PCB mounts for attaching a DS18S20 sensor and either a RJ-11 connector or 2 terminal screw connector. E-mail us for information. mitch@midondesign.com

NOTE: if you are using parasitic power, please connect the ground wire to both outer pins of the DS18S20 to ensure reliable readings.

4. Can I daisy-chain temperature sensors?

Yes. They can be wired into anywhere along the wire run from TEMP05. A termination is not required at the end of the wire run when daisy-chaining, although care should be taken to avoid shorting out the end point.

5. How far away can the temperature sensors be run?

Using good quality CAT-5 cable, you should be able to run sensors up to 300 feet (100 meters) away from the TEMP05.

6. Where do I get a Weather Station?

The original manufacturer (Dallas Semiconductor) no longer makes this product. One manufacturer Tecnologia Aplicada has replicated the original design and is now shipping product (see also question 19). Another source of similar product is Texas Weather Instruments Inc. . Midon Design has verified functionality of Technoligia Aplicada and Dallas Semiconductors products with the TEMP05. Technoligia Aplicada: www.aag.com.mx/weather.html

7. How does TEMP05 connect to my PC?

A straight-through, not null modem, serial cable connects between the TEMP05 and your PC's serial port. The signal is 9600 bps, no parity, eight bits, with NO flow control. Hyperterminal can be used to monitor output from the TEMP05, or to configure it. See also #17. ____ Radio Shack has a serial cable that could be used to connect to your PC.

8. Can the sensors be used in water (pools, fish tanks, spas, etc.)?

Yes, although care must be taken to ensure that the connections to the sensor, and the sensor leads, are properly water-proofed. Silicon RTV or epoxy could be used to accomplish this.

9. How can I tell which sensor is in which area?

The recommended method is to connect one sensor at a time, run the INI command on TEMP05 and then record the newest serial number found in the list displayed by the DIS command. That serial number will appear in the sensor output if the SID ON command has been issued.

Alternatively, all sensors can be pre-configured. Attach each sensor one at a time. Note which sensor output value changes from "???" to a valid temperature reading after attaching a new one. That is your new sensor.

If you have already installed all your sensors, just disconnect each one individually and look for the sensor that changes value from a valid temperature reading to "???".

10. What do I use for power?

Just about any power adapter can be used with the TEMP05. If your are NOT using the companion RELAY05 module, then you can look through your discarded wall adapters from old electronic devices for something that is rated at least 9 Volts (AC or DC) and at least 100mA (milli-Amperes).

11. What do I need to read temperatures from a few different rooms?

For example, to get temperature readings from 3 different areas you will need;

1 TEMP05 (assembled and tested) @ $65 ea
3 DS18S20 Sensors @ $4 ea
1 12VAC or DC adapter (available from Radio Shack, Best Buy, etc. or from midon design @ $5 ea.

All items are on the ordering page on our web site. WWW.midondesign.com

The TEMP05 connects to your PC via a spare serial port. You will need a straight-through serial cable, not a null modem cable, to connect TEMP05 to your PC. An example cable would be Radio Shack's 26-117.

A sample HomeSeer script is available from the TEMP05 web page.

Note: if one of the temperatures you want to sense is in the same area as where you will place the TEMP05, then you will only need one extra sensor since there is one already built onto the TEMP05 board.

12. Do I have to use HomeSeer to read the temperatures?

No. Any PC application that reads the serial port can be used. HomeSeer provides some simple to use script interfaces to the serial port, however.

13. Will TEMP05 work with a serial to USB converter?

Yes. TEMP05 has been successfully tested with a Belkin F5U103 USB Serial Adapter.

See also this posting on the HomeSeer message board: http://www.keware.com/cgi-bin/bbs4/config.pl?read=8103

14. Will TEMP05 work with the Dallas Semiconductor Humidity Sensor?

Yes! V4.14 firmware and higher will read humidity from multiple Dallas Humidity sensors.

15. Is there an enclosure for the TEMP05?

Not yet. Stay tuned.

16. What are the differences between a DS1820 and a DS18S20?

The document here describes this best. Basically, the DS18S20 has superior temperature accuracy and a wider power supply tolerance.

17. How do I set up HyperTerm to talk to TEMP05?

Correct settings in HyperTerminal are:

- 9600 baud
- 8 bits
- no parity
- 1 stop bit
- NONE for flow control (***Important*** !)

18. How do I set-up HomeSeer to use the readings from TEMP05?

Follow the instructions here.

19. What is the difference between a Version 1 and a Version 3 One Wire Weather Station?

The Version 1 OWWS is the original unit developed and sold by Dallas Semiconductor. Is is electrically equivalent to the Version 2 OWWS sold by AAG, a Mexican company. AAG improved the design to replace the DS2401-based wind direction sensors with a new DS2450-based design and are now selling it as the Version 3 OWWS. TEMP05 firmware V4.16 or higher will work with the Version 3 or Version 1 or Version 2 OWWS.

20. I ordered DS18S20's, but the items I received are labeled DS1820. What gives?

Dallas/Maxim have chosen not to label the DS18S20's with the "S". This may be due to the fact that they have discontinued the original DS1820 package and assumed that there would be no confusion. The DS1820 stopped production around May 2000. The only visible difference is the package style. The DS1820 was in a PR-35 package, whereas the DS18S20 is in a standard TO-92 package. See FAQ #16 for the electrical differences.

21. Will TEMP05 work with DS18B20, DS1920 and DS1822 temperature sensors?

Absolutely! You will need V4.21 or higher software to read the DS18B20 and DS1822 sensors. DS1920 sensors are identical to DS1820 sensors and can be read with any version of TEMP05.

22. I am using V4.19 (or higher) and my weather station stopped providing the wind direction (gives ???). Why?

The ??? is used to display incorrect readings. While there are several possibilities for this, faulty sensors for example, there is one likelier problem with 4.19 or higher software. The NOR command, which is used to calibrate the wind direction sensor(s), stores the calibration results in one of your DS1820 sensor's EEPROM memory. If you have removed the DS1820 sensor that had that information stored, or did an ERA and INI command, then the software will read incorrect calibration results and error out on the direction. Try re-doing the NOR command and see if that helps.

23. I added a sensor and now my readings are coming back as "???". Why?

You might have added a the sensor via a new wire leg in a star configuration. This might have created reflections on the One Wire bus. Try adding a 100 ohm resistor (Radio Shack part number 271-1311) in series with each leg of the star (only on the DQ lead).

24. Where can I buy a Humidity Sensor?

Midon Design does not sell the One Wire Humidity Sensor. They are only available from Dallas/Maxim. Go to this link http://www.ibutton.com/weather/humidity.html to view specs and order.

midon design

Setting Up HomeSeer to Read From TEMP05

1. Make sure that TEMP05 is connected to your COM1 serial port on your HS PC. If it is not, rename the script below (COM1_EVENT.TXT) to whatever COM port that TEMP05 is connected to. Save that script in your HomeSeer Scripts folder.
2. Add the following lines to your "startup.txt" HS script (put them anywhere before the "end sub" line);

```
e=hs.OpenComPort(1,"9600,n,8,1",1,"com1_event.txt","main")
    if e<> "" then
         hs.writelog "Error opening COM1",e
    else
         hs.writelog "COM1", "Setup complete"
end if
```

3. Change the red numbers from 1 to whatever com port you are using.
4. Add the following lines to your "shutdown.txt" script (again, anywhere before the "end sub");

```
e=hs.CloseComPort("1")
    if e<> "" then
         hs.writelog "Error closing COM1",e
    else
         hs.writelog "COM1", "Port was closed"
    end if
```

5. Shutdown HS and then restart it. You should see the "COM1 setup complete" message in your log.
6. Add virtual X-10 devices R1 through Rx (where x is the number of sensors that you have). Define them as "status only" and create a new device type for them (doesn't matter what you call it).
7. If you have a humidity sensor connected, create a virtual device R20 for it. If you have multiple humidity sensors, create multiple virtual devices, starting at R20.
8. If you are using firmware with a version of 4.16 or lower, **OR, you are not using a V1 weather station**, add a virtual device R30. This will be used to record the readings for the "0" sensor. Remove the quote on the line in blue below.
9. If TEMP05 is connected to the proper COM port, then you should, within a minute (or whatever sampling time you set in TEMP05), start seeing at least one valid temperature in virtual devices R1-x. If you have all your sensors connected, then all of the virtual devices should be showing valid temperatures.
10. If you want to generate a web page using these virtual devices, go here.

SCRIPT COM1_EVENT.TXT

```
' this script reads data from TEMP05 and stores received temperatures into HS virtual
devices
' Created by Mitch Matteau 00-01-15
' Modified 01-11-25: added DeviceValue saving of temperature results
' Modified 01-09-16 : added capture of relay status, rain counts, wind speed, wind
direction and TEMP05 version #
' Modified 01-08-25 : added conditional update of DeviceLastChange and removed restriction
for V4.16 or greater firmware
' Modified 01-08-21 : changed to add use of hs.SetDeviceLastChange to record changes
' Modified 01-07-04 : added support for OWWS (V1 or V2) users
' Modified 01-04-02 : added support for 1 humidity sensor
' Modified 01-05-06 : added SetDeviceValue output of temperatures
' Modified 01-05-27 : added support for multiple humidity sensors
'==============================================================
' HS virtual devices used;
' r1 to r19 for DS18S20 sensors
' r20-29 for humidity sensor
'=============================================================
sub main(data)
'=============================================================
dim i
dim result
dim virt_device
dim prev_result
'=============================================================
' Check if the received data is from a temperature reading
'=============================================================
if mid(data,1,6)="Sensor" then
    sensorid = mid(data,9,2)
    if left(sensorid,1) = "0" then
        sensorid=mid(sensorid,2,1)
    end if
' the following line is only required for V4.16 or lower firmware OR if you are not using a
V1 Weather Station
'   if sensorid="0" then sensorid = "30" ' needed for the zero'th sensor
'
    i = Instr(data,"=")
    result=trim(mid(data,i+1))
    if right(result,1)="F" or right(result,1)="C" then
        result = left(result,(len(result)-1))
    end if
    if CInt(sensorid)<=19 then
        virt_device="r"+sensorid '"r" is the house code for the virtual device used
        prev_result=hs.DeviceString(virt_device)
    end if
'=============================================================
' eliminate false readings
' false readings are replaced with the previous reading
'=============================================================
    if left(result,3) = "???" then
        result = prev_result
    end if
    if left(result,3) = "185" then '185 is an indicator that the device is not connected
        result = prev_result
    end if
'=============================================================
' Now save current readings into HomeSeer
'=============================================================
    hs.SetDeviceString virt_device,result
    if result<>prev_result then
       hs.SetDeviceLastChange virt_device, Now()
       hs.SetDeviceValue virt_device,result
    end if
end if
'=============================================================
' Humidity Sensor Capture v4.12 or v4.13 software only
'=============================================================
if left(data,11)="Humidity = " then
    prev_result=hs.DeviceString("r20")
    result = Trim(mid(data,11))
    if result="???" then
        result=prev_result
        hs.Writelog "Humidity","Sensor error"
    end if
    hs.SetDeviceString "r20",result
      if result<>prev_result then
        hs.SetDeviceLastChange "r20", Now()
    end if
end if

'=============================================================
' Humidity Sensor Capture v4.14, or higher, software only
'=============================================================
if left(data,10)="Humidity #" then
```

```
    i=Instr(data,"#")
    sensorid=mid(data,i+1,2)
    if left(sensorid,1) = "0" then
        sensorid=mid(sensorid,2,1)
    end if
    virt_device="r"+CStr(CInt(sensorid)+19)
    prev_result=hs.DeviceString(virt_device)
    i = Instr(data,"=")
    result = Trim(mid(data,i+1))
    if result="???" then
        result=prev_result
        hs.Writelog "Humidity","Sensor error on #"+sensorid
    end if
    hs.SetDeviceString virt_device,result
      if result<>prev_result then
        hs.SetDeviceLastChange virt_device, Now()
    end if
end if

'=============================================================
' Version capture
' virtual device z17 used to capture the data
'=============================================================
if mid(data,1,6)="TEMP05" then
version_id = "z17"
    hs.SetDeviceString version_id,data
    hs.SetDeviceLastChange version_id, Now()
end if
'=============================================================
' Relay Status capture
' virtual devices z31 through z38 used to capture the data
'=============================================================
if mid(data,1,5)="Relay" then
    i = Instr(data,"=")
    data = mid(data,i+1)
    for i=1 to 8
        virt_device= "z3"+trim(CStr(i))
        a = trim(hs.StringItem(data,i,","))
        if a<>hs.DeviceString(virt_device) then
            hs.SetDeviceString virt_device,a
            hs.SetDeviceLastChange virt_device,Now()
        end if
    next
end if
'=============================================================
' Wind and Rain Information capture
'=============================================================
if left(data,5)="=====" then
    dim wind_dirn
    dim wind_spd
    dim wind_gust
    dim high_wind
' virtual device list
    wind_dirn = "z7"
    wind_spd = "z4"
    wind_gust ="z1"
    high_wind = "v6"
    result = mid(data,6)
    a = trim(hs.StringItem(result,1,","))       ' wind direction
    if a <>"???" then
        hs.SetDeviceString wind_dirn,a
        hs.SetDeviceValue wind_dirn,0
    end if
    a = hs.StringItem(result,2,",")             'wind speed
    a = trim(left(a,len(a)-3))
    if left(a,2)<>"??" then
        hs.SetDeviceString wind_spd,a
        hs.SetDeviceValue wind_spd,CInt(a)
    else
```

```
        a=0
    end if

    b = hs.DeviceString(high_wind)
    if isnumeric(b) then
        b = CInt(hs.DeviceString(high_wind))
    else
        b = 0
        hs.SetDeviceString high_wind,"0"
        hs.SetDeviceLastChange high_wind, Now()
    end if

    if Cint(a) > CInt(b) then
        hs.SetDeviceString high_wind,CStr(a)
        hs.SetDeviceLastChange high_wind, Now()
        hs.WriteLog "HIGH WIND",a+" mph"
        hs.CreateVar("HWNDT")
        hs.SaveVar "HWNDT",Time()
    end if

    a = hs.StringItem(result,3,",") ' wind gust
    a = trim(left(a,len(a)-3))
    hs.SetDeviceString wind_gust,a
    hs.SetDeviceValue wind_gust,CInt(a)

'============================================================
' Rain Counts
'============================================================
    dim rain_history
    dim rain_today
    dim rain_absolute
    rain_history = "v8"
    rain_today = "v7"
    rain_absolute = "v9"

    b = hs.StringItem(result,4,",") ' rain count
    if len(b)>2 then
        b = trim(left(b,len(b)-2)) ' remove units
        c = hs.DeviceString (rain_history) ' History Info
        a = hs.StringItem(c,7,",") ' Yesterday's count

        if IsNumeric(b) then
        l = CSng(b) ' Value of Absolute count

            if Not IsNumeric(a) then
                k = l ' on error, assume no count
            else
                k = CSng(a) ' Value of Yesterday's count
            end if

            if (l-k) > 10 then
                a = "0.00" ' can't do more than 9 inches in 1 cycle
            else
                a = CStr(Round((l-k),2)) ' Result is today's count
            end if
        end if
    end if

    if IsNumeric(a) then
        hs.SetDeviceString rain_absolute,b ' absolute rain count
        hs.SetDeviceLastChange rain_absolute, Now()
        hs.SetDeviceString rain_today,a ' Today's count
        hs.SetDeviceValue rain_today,CInt(a)
        d = ""
        for i = 1 to 7
            d = d + hs.StringItem(c,i,",")+","
        next
        d = d + b
        hs.SetDeviceString rain_history,d ' Restore history info
        hs.SetDeviceLastChange rain_history, Now()
    end if
end if

end sub
```

Return to FAQ

Last Updated 01/11/25

midon design

Display TEMP05 readings on a web page

1. Setup TEMP05 to communicate with HomeSeer via the procedure here.
2. Add the gen_temp_page.txt script below to your HomeSeer Scripts folder.
3. Change any parameters you need to in the script (highlighted in red);

- Sensor_qty (currently set to 4)
- dimensions of hi(x,1), lo(x,1), Sensor_name(x) and Tnow(x) to match Sensor_qty
- sensor virtual device "house code" ("r" is used here)
- wind speed sensor (z4 used here)
- wind gust sensor (z1 used here)
- wind direction sensor (z7 used here)

3. Change the Sensor names to match your setup.
4. Add an event called "Refresh TEMP05" that calls this script on a recurring basis.
5. Add a link to your new web page "Temp05.html" somewhere.
6. Have fun!

SCRIPT gen_temp_page.txt

```
'========================================================================================
' Filename = gen_temp_page.txt v1.4
' Purpose: generate a Temperature status web page
' original author: Rich @ Keware (from gen_status_page.txt)
'
' Vintage Control
' ===============
' created 01-28-2001 by Mitch Matteau
' modified 09-30-2001 by Mitch Matteau - added wind sensor information
'========================================================================================
' this script will create a file named "temp05.html" in your HomeSeer "html" directory
' you can add this page as a link off of your HomeSeer personal home page
' the page displays the current temperature information as written to virtual devices
' by the Midon Design TEMP05 One-Wire Logger via script "com1_event.txt"
' For this to work properly you need to create an event named "Refresh TEMP05"
' and set the event to run this script (I set it to run every 10 minutes)
' To see the current outside temperature on HomeSeer's device status page, create a virtual
device r1
' Other virtual devices used include;
' r2 Sensor 2 Temperature
' r3 Sensor 3 Temperature
' r4 Sensor 4 Temperature
' r20 Humidity Sensor
' z4 Windspeed
' z1 Wind gust
' z7 wind direction
' note that you need to run this script at least once from within HomeSeer to create the
initial page
' to do this, right-click on the event and hit "execute now"
```

```
'=========================================================================================

sub main()
Dim Sensor_qty
Sensor_qty=4
Dim Sensor_name(4)
dim hi(4,1)
dim lo(4,1)
dim Tnow(4)
Dim virtual_device
virtual_device="r"

dim windspeed
dim wind_dirn
dim wind_gust

windspeed = "z4"
wind_dirn = "z7"
wind_gust = "z1"

Sensor_name(1)="Sensor 1"
Sensor_name(2)="Sensor 2"
Sensor_name(3)="Sensor 3"
Sensor_name(4)="Sensor 4"

Dim Folder, TextStream
dim i,p
dim x,y
dim ndate, ntime
dim temperature
dim S
dim File
dim hilo_date
dim a,b,c,d,e

dim filler
filler = "<TD>&nbsp</TD>"

hs.CreateVar("HILODATE")
hilo_date=hs.GetVar("HILODATE")

'=========================================================================================
' Get Hi and Lo temps from All Sensors
'=========================================================================================

for i = 1 to Sensor_qty
    x = "HI_R"+CStr(i)
    hs.CreateVar(x)
    hi(i,0)=hs.GetVar(x)
    x = "HIT_R"+CStr(i)
    hs.CreateVar(x)
    hi(i,1)=hs.getvar(x)
    x = "LO_R"+CStr(i)
    hs.CreateVar(x)
    lo(i,0)=hs.GetVar(x)
    x = "LOT_R"+CStr(i)
    hs.CreateVar(x)
    lo(i,1)=hs.GetVar(x)
next

'=========================================================================================
' Determine current time/date
'=========================================================================================

a = CStr(year(now))
b = CStr(month(now))
c = CStr(day(now))
d = CStr(time())
if Len(b) = 1 then
    b = "0"+b
end if
if Len(c) = 1 then
    c = "0"+c
end if
today = a+"-"+b+"-"+c+" "+d+" "
a = right(a,2)
ndate = a+"/"+b+"/"+c
ntime = d

'=========================================================================================
' Get current temps from the virtual devices
'=========================================================================================
For i = 1 to Sensor_qty
    x = virtual_device+CStr(i)
    Tnow(i) = hs.DeviceString(x)
next
'=========================================================================================
' Check if the variables need initialization
'=========================================================================================
for i = 1 to Sensor_qty
    if hi(i,0)="" then
        hi(i,0)=Tnow(i)
        hi(i,1)=ntime
    end if
    if lo(i,0)="" then
        lo(i,0)=Tnow(i)
        lo(i,1)=ntime
    end if
next
'=========================================================================================
' Check if today became tomorrow
'=========================================================================================
if hilo_date <> ndate then
    hs.CreateVar("HILODATE")
    hs.SaveVar "HILODATE",ndate
    For i = 1 to Sensor_qty
        x = "HI_R"+CStr(i)
        hs.SaveVar x,Tnow(i)
        x = "HIT_R"+CStr(i)
        hs.SaveVar x,ntime
        x = "LO_R"+CStr(i)
        hs.SaveVar x,Tnow(i)
        x = "LOT_R"+CStr(i)
        hs.SaveVar x,ntime
    Next
else
    For i = 1 to Sensor_qty
        if CSng(hi(i,0)) < CSng(Tnow(i)) then
            hi(i,0) = Tnow(i)
            hi(i,1) = ntime
        end if
        if CSng(lo(i,0)) > CSng(Tnow(i)) then
            lo(i,0) = Tnow(i)
            lo(i,1) = ntime
        end if
        x = "HI_R"+CStr(i)
        hs.SaveVar x,hi(i,0)
        x = "HIT_R"+CStr(i)
        hs.SaveVar x,hi(i,1)
        x = "LO_R"+CStr(i)
        hs.SaveVar x,lo(i,0)
        x = "LOT_R"+CStr(i)
        hs.SaveVar x,lo(i,1)
    Next
end if

'========================================================================================
' Now create the actual web page (overwrite any existing one)
'========================================================================================
p = hs.GetAppPath
Set Folder = CreateObject("Scripting.FileSystemObject")
Set TextStream = Folder.CreateTextFile(p+"\html\Temp05.html", True)
'========================================================================================
' create the header for the html page and include a button for refreshing
'========================================================================================
TextStream.Write "<html>"
TextStream.WriteBlankLines(1)
TextStream.Write "<Title>TEMP05 Results</Title>"
TextStream.WriteBlankLines(1)
TextStream.Write "<body BGCOLOR=""#D3D3D3"">"
TextStream.WriteBlankLines(1)
TextStream.Write "<H1><CENTER>Current Temperature Information</H1>"
TextStream.WriteBlankLines(1)
TextStream.Write "<form method=""POST"">"
TextStream.Write "<p><input type=""submit"" value=""Refresh TEMP05""
name=""run_event""></p>"
' NOTE: you must have an event created called Refresh TEMP05 that calls this script
TextStream.Write "<p>"
TextStream.Write "<center><p><FONT COLOR=""#0000b0"">"
TextStream.WriteBlankLines(1)
TextStream.Write "<b>The real-time temperature data displayed below is from a "
TextStream.WriteBlankLines(1)
TextStream.Write "<a href=""http://temp05.midondesign.com"">Midon Design Temp05</a>
One-Wire Sensor Serial Interface."
TextStream.WriteBlankLines(1)
TextStream.Write " They are connected to my home automation system, running "
TextStream.WriteBlankLines(1)
TextStream.Write "<a href=""http://www.homeseer.com""> HomeSeer</a>,"
TextStream.WriteBlankLines(1)
TextStream.Write " which also acts as a local web server. </b></p> </center>"
TextStream.WriteBlankLines(1)
TextStream.Write "<p></p>"
'========================================================================================
' now display the actual data
'========================================================================================
TextStream.Write "<B>Last reading= "+ndate+" "+ntime
TextStream.WriteBlankLines(1)
TextStream.Write "<P>Humidity = "+hs.DeviceString("r20")+"%</P>"
TextStream.WriteBlankLines(1)
TextStream.Write "<P><H2>Temperatures in °F</P>"
TextStream.Write "<TABLE BORDER>"
TextStream.Write "<TR><TH>Location</TH><TH>Current</TH><TH>High
```

```
Today</TH><TH>Time</TH><TH>Low Today</TH><TH>Time</TH></TR>"
TextStream.WriteBlankLines(1)
For i = 1 to Sensor_qty
    x = virtual_device+CStr(i)
    a = "<TR><TD>"+Sensor_name(i)+"</TD><TD ALIGN=CENTER>" + hs.DeviceString(x)+"</TD>"
    b = "<TD ALIGN=CENTER>" + hi(i,0) +"</TD>"
    c = "<TD ALIGN=CENTER>" + hi(i,1) +"</TD>"
    d = "<TD ALIGN=CENTER>" + lo(i,0) +"</TD>"
    e = "<TD ALIGN=CENTER>" + lo(i,1) +"</TD></TR>"
    TextStream.Write a+b+c+d+e
    TextStream.WriteBlankLines(1)
next

TextStream.Write "</TABLE>"
TextStream.WriteBlankLines(1)
TextStream.Write "<P>Wind Speed ="+hs.DeviceString(windspeed)+" MPH"
TextStream.WriteBlankLines(1)
TextStream.Write "<P>Wind Gust was "+hs.DeviceString(wind_gust)+" MPH"
TextStream.WriteBlankLines(1)
TextStream.Write "<P>Wind is from "+hs.DeviceString(wind_dirn)
TextStream.WriteBlankLines(2)

TextStream.Write "<P>Local Sunrise= "+ah.Sunrise
TextStream.Write "<P>Local Sunset= "+ah.Sunset
TextStream.WriteBlankLines(1)

TextStream.Write "</B></p>"
TextStream.WriteBlankLines(1)
TextStream.Write "<a href=""http://www.homeseer.com""><img align=center src=""powered.jpg""
border=0 alt=""HomeSeer logo"">"
TextStream.Write "</CENTER>"
TextStream.Write "</body>"
TextStream.Write "</html>"

TextStream.Close
end sub
```

Return to Setting Up Homeseer

Last Updated 01/09/30

□ midon design 2000

DALLAS SEMICONDUCTOR

03/07/00

Dear Weather Station experimenter:

The thermometer used in the Dallas Semiconductor WS-1 Weather Station Experimenters Kit is a Dallas DS1820, 1-Wire™ digital thermometer. Many issues about the DS1820 have been posted to the 1-Wire™ Weather Interest Group, and we feel it is important to try to explain the inaccuracies you may be encountering. Even more importantly, we would also like to introduce the next generation 1-Wire™ Digital Thermometer, the DS18S20, that addresses many of the issues with the DS1820 and offers smaller packaging options at a lower cost.

By design, the DS18S20 corrects for environmental stress-induced drift and a state machine glitch that has caused many of you confusion in the past months. DS18S20 accuracy and stability data will be presented to hopefully convince you the stability problems with the DS1820 have been eliminated. In addition to correcting stability problems, the DS18S20 design also allows for a wider power supply range and smaller packaging options. Incompatibilities between the DS18S20 and the DS1820 are minimal, but a few are discussed in detail. An incompatibility that will have a minor, yet noticeable effect on Weather Station users is the lower resolution of the DS18S20 data converter. We will very soon release a software patch to the WS-1 that performs waveform smoothing. Overall, we think you will be quite pleased with the performance of the newly redesigned and improved DS18S20.

Before getting into detail of the DS18S20, let us attempt to explain some of the peculiarities you may have experienced with the DS1820 in the past. The DS1820 was the first of its kind to offer an integrated solid state temperature sensor, an "analog-to-digital" converter, and a 1-Wire™ Network interface. Before describing the cause for some of the inaccuracies of the DS1820, a brief tutorial on the temperature-sensing algorithm used by the DS1820 would be beneficial. Refer to Fig.1 below:

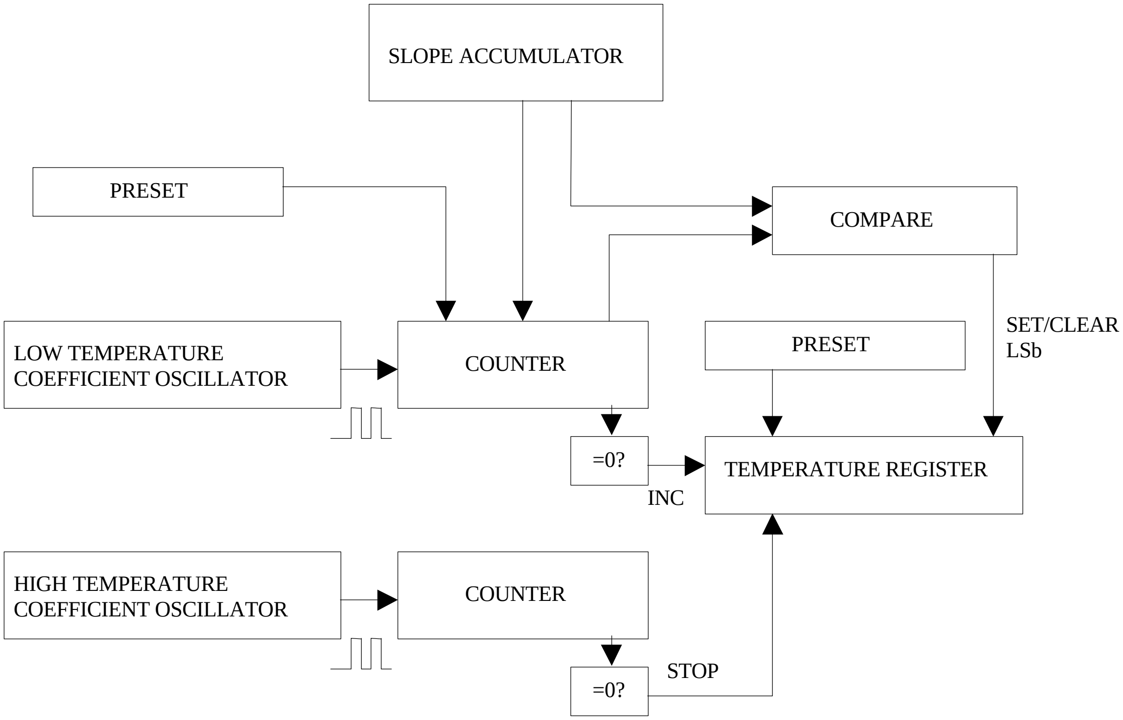

Fig.1. Temperature Sensing Block Diagram

The DS1820 measures temperature by counting the number of clock cycles that an oscillator with a low temperature coefficient goes through during a gate period determined by a high temperature coefficient oscillator. The counter is preset with a base count that corresponds to -55°C. If the counter reaches zero before the gate period is over, the temperature register, which is also preset to the -55°C value, is incremented, indicating that the temperature is higher than -55°C.

At the same time, the counter is then preset with a value determined by the slope accumulator circuitry. This circuitry is needed to compensate for the parabolic behavior of the oscillators over temperature. The counter is then clocked again until it reaches zero. If the gate period is still not finished, then this process repeats.

The slope accumulator is used to compensate for the nonlinear behavior of the oscillators over temperature, yielding a high-resolution temperature measurement. This is done by changing the number of counts necessary for the counter to go through for each incremental degree in temperature. To obtain the desired resolution, therefore, both the value of the counter (COUNT_REMAIN) and the value of the slope accumulator (COUNT_PER_C) at a given temperature must be known. Therefore, attainable resolution (1/COUNT_PER_C) is a function of temperature, and will vary from part to part. A typical COUNT_PER_C ranges from 80 –120 over the DS1820 temperature range, translating to a 14-15-bit resolution.

There are two issues with this algorithm. The first manifests itself as a drift in accuracy, and explains many complaints about the DS1820's temperature error. The solid state oscillators in Fig.1 contain polysilicon resistors, which are extremely difficult to control over temperature and mechanical stress. The temperature coefficient of the resistors is not a problem because we calibrate out their effect in the liquid bath. Mechanical stress, however, is something we cannot control or predict so to account for its effect in the slope accumulator. As the mechanical stress on the DS1820 die changes, so too does the value of the poly resistors in the oscillators. Resistance changes cause oscillator frequency changes, which will ultimately cause changes in the temperature register value. We assemble the DS1820 in a non-hermetic plastic package and calibrate the parts in a liquid bath. Liquid is free to enter the cavity above the die in the non-hermetic package, putting a nominal stress on the die (and resistors). The device is calibrated under this very controlled environment. As the device is removed from the liquid and allowed to dry out, the stress on the die changes and the temperature readout changes. This is why many of you note temperature readout variations in extremely humid environments. Another stress that will cause a drift is a high temperature process such as a vapor phase. Expansion/contraction of the plastic package/leadframe will cause the stress on the die to shift, and again a readout drift will result. A typical drift following a vapor phase is 2.0°C (approximately 4.0°F), and generally the part will read low following the stress. A typical drift following an Autoclave process (device is subjected to pressurized steam (2atm) at 121°C for 168 hours) is over 3.0°C (6.0°F); Autoclave has proven to be the worst case stress insofar as drift is concerned.

This is obviously an extremely frustrating problem. We have tried various layout implementations of the resistors (if the resistance shift in the high tempco oscillator matches that of the resistor in the low tempco oscillator, the effect would cancel), different plastic mold compounds, and the drift could never be completely eliminated.

Many of you have tried numerous experiments in an attempt to minimize the DS1820 thermometer error. Although some of the experiments posted may offer a minimal effect, the dominating factor in the error is the mechanical stress-induced drift described above, and very likely not related to anything you are doing wrong. The cause of the drift is a physical phenomenon that cannot easily be removed from the DS1820. It has been shown that the drift diminishes as the device is baked at 125°C for several hundred hours. After the bake, the user can compare the DS1820 reading against an accurate reference at any temperature (recalibrating the device), and add this offset to future readings. Generally speaking, the drift magnitude is consistent over the entire temperature range, thus allowing for a single-temperature post-bake offset correction.

This however requires access to an oven, a controlled temperature chamber, an accurate reference sensor, and the ability to modify the WS-1 software to add the resulting offset to the DS1820's readout. This solution is understandably a severe restriction for most users. The most feasible solution is for Dallas to redesign the DS1820 thermal algorithm so that mechanical stress will have a negligible effect on stability. The aforementioned DS18S20 does just that by measuring temperature with a completely different algorithm, one that virtually eliminates the effect mechanical stress has on stability. The DS18S20 will be discussed in detail later in this document.

The second problem many of you have noted with the DS1820 is also an artifact of the dual-oscillator algorithm and is also eliminated with the DS18S20. It manifests itself as a "glitch" in the readout of a maximum of 1.0°C. This issue is described in detail in an attached document. This document makes reference to the DS1620, but is directly applicable to the DS1820, as both Dallas sensors use the same thermal-sensing algorithm.

Solving these two issues was of primary concern in redesigning the DS1820 1-Wire™ Digital Thermometer. Technology has improved since the DS1820 was designed, allowing us fabricate the redesign with a 0.6µm minimum geometry process. So, not only are the drift and glitch problems eliminated, but also the new die is significantly smaller which allows for smaller packages and a lower cost sensor. The redesigned thermometer, the DS18S20, will be assembled in a TO-92 package (replaces the DS1820 PR-35) and a 150mil 8-lead SOIC (replaces the DS1820 SSOP).

The block diagram of the DS18S20 thermal sensing algorithm is illustrated below in Fig. 2.

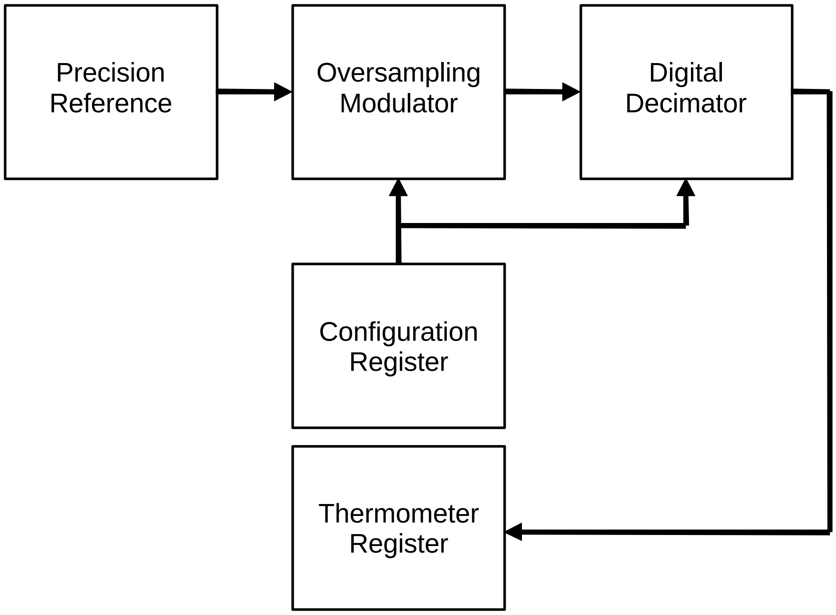

Fig. 2. Bandgap / Sigma-delta modulator temperature sensor

A very precise, on-chip bandgap reference is the first stage of the signal chain. Bandgap voltage references are inherently very stable and are commonly used in solid state temperature sensors, power monitoring products, etc. The sigma delta modulator serves as the data converter in addition to providing chopping which eliminates stress-induced errors in the differential Vbe bandgap. Dallas has this algorithm in production with several digital thermometers – DS75, DS1721, DS1722, DS1775, DS1780, DS1615, and the DS1921 Thermochron. Many of you have tested the DS1921 against the DS1820 and have noted the superior performance of the DS1921.

The DS18S20 will actually go one step further than all these sensors in that it can be bath calibrated (because of its on-chip EEPROM). The most accurate of the non-bath calibrated bandgap-based sensors above are the DS1721, DS1615, and DS1921 at ±1.0°C max error. Bath calibration drastically reduces the error at the calibration temperature; the maximum error over the -10°C to +85°C range for the DS18S20 is ±0.5°C, before, during and after environmental stress.

Fig 3 shows thermometer accuracy data for 4 different samples of DS18S20. Each sample of approximately 50 parts was subjected to one of 4 stresses: High Voltage Life (1000 hours at 6.0V at T=125°C), 3 consecutive passes through a vapor phase, Autoclave (unbiased at 2atm steam T=121°C for 168 hours), and Temperature Cycle (unbiased under 1000cycles from -55°C to +125°C). The data was taken before and after the respective stress and the plots show the mean and ±3σ error (relative to platinum RTD reference in liquid bath) after stress. Mean drift (average difference between post stress and pre stress error) is also plotted. This is a significant improvement over the DS1820. The average part drifts less than 0.1°C (roughly 1 LSb), with insignificant dependence on the type or duration of stress it was subjected to. Recall from earlier in the document, the average drift of the DS1820 following the Autoclave stress (the worst case stress) was over 3.0°C, representing better than a 99% reduction in stress-induced drift.

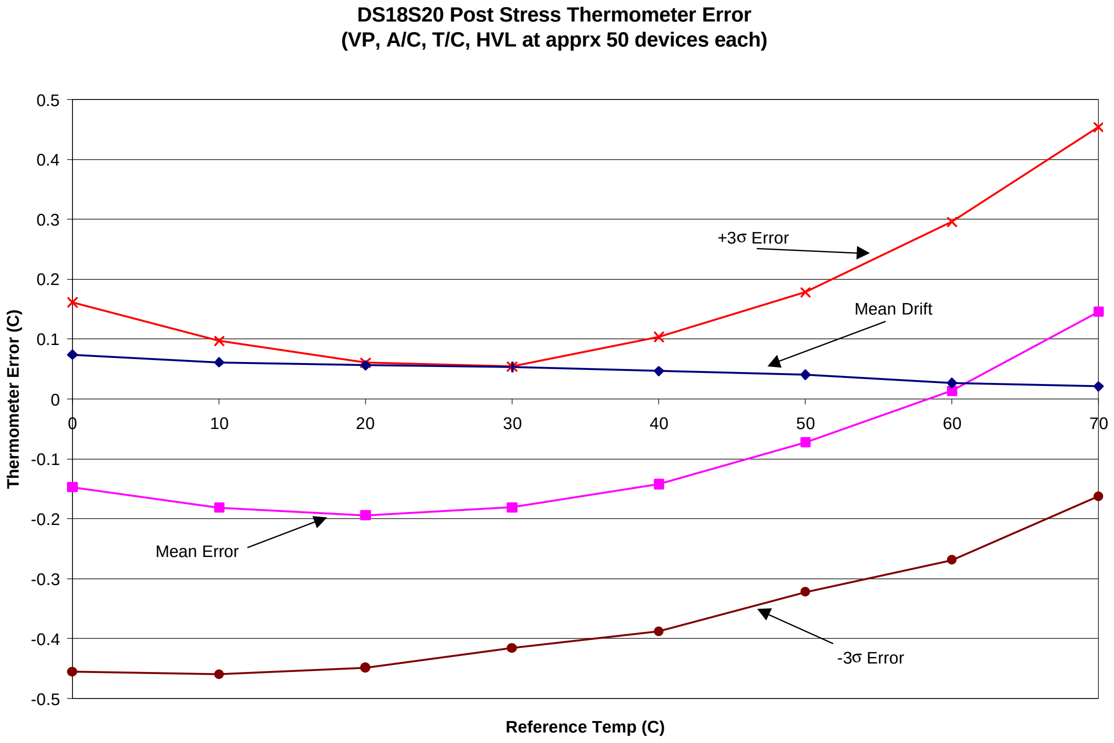

Fig. 3. DS18S20 Performance

Clearly, the DS18S20 has solved the inaccuracies many of you in the interest group have noted in the past. The other problem with the DS1820, the readout glitch that was described in the attached document, is totally an artifact of the dual-oscillator thermal sensing algorithm. The bandgap / Sigma-delta ADC sensing core of the DS18S20 by design will not be subject to such glitches.

The design goals of the DS18S20 were numerous:

1.) Eliminate stress-induced drift.
2.) Eliminate the state-machine glitch.
3.) Expand the operating voltage from 4.5V-5.5V of the DS1820 to 2.7V-5.5V.
4.) Significantly shrink the die so that it can fit in smaller, faster responding plastic and flipchip packages.
5.) Satisfy 1-4 while allowing the DS18S20 to drop into every DS1820 application, with no noticeable adverse differences.

All 5 of these goals are practically impossible to meet for such a drastic redesign of a complex mixed-signal IC. This document has described how we have met goals 1 through 4, but there are some noteworthy differences between the DS1820 and the DS18S20 that may not allow for a seamless transition into 100% of applications.

1.) Packaging. Smaller/lower cost packages was one of the design goals, and we met that. However, package differences may have an effect on some customers. The PR-35 of the DS1820 will no longer be offered with the DS18S20; it will be replaced by the TO-92 package. The pinout and lead spacing is identical between the two packages, so most customers will not have a problem. Customers who depend on the height of the PR-35 package in their application will be forced to make mechanical changes to their design. The surface mount option is more drastic. The huge DS1820 die was packaged in a 16-lead SSOP (with 13 no-connects); that was the smallest SMT package we could fit the DS1820 in. The SSOP is large and expensive. The SSOP will not be offered with the DS18S20, and the replacement SMT package is the 8-lead SOIC. Obviously, all SMT DS1820 customers will have a change. Dallas is offering the DS1820 in both the PR-35 and SSOP packages through 2000 on last time buy bases (but again, neither package will be offered with the redesigned DS18S20). Refer to the DS18S20 datasheet on the website for pinouts and mechanical specs of the packages.

2.) Thermometer Resolution. Recall the discussion about the temperature-to-digital conversion technique we use in the DS1820. Due to the nonlinear characteristic of the oscillators over temperature, we designed the slope accumulator with a high resolution (COUNT_PER_C generally over 100, yielding a sub-millidegree C resolution) so that the nonlinearity can be trimmed out. A conversion would always start with the counter preset to -55°C, and would count up to the temperature of the environment. Thus, conversion time depends significantly on the absolute temperature with the fastest at very low temperatures. This algorithm allowed for a conversion time of less than a half second, over the entire temperature range. The resolution attainable considering the conversion time and active current was outstanding with the DS1820, yet it was the dual oscillator technique that caused the drift. Solving the drift meant the dual-oscillator algorithm had to be designed out, while trying to minimize adversely affecting other specs. Sigma-delta modulators like that used in the DS18S20 are by design, slow data converters. There is a tradeoff between supply current, resolution, and conversion time. High resolution can be obtained in a relatively short amount of time if enough supply current is available. If supply current is crucial, high resolution requires long conversion times. You see the dependence.

    If the DS18S20 met the resolution (14-15bit) and active current (1.5mA) spec of the DS1820, the conversion time required would be between 4 and 8 seconds! That was obviously not an option. A tradeoff was made in the DS18S20 design. We kept the active current spec constant at 1.5mA, and allowed for a max conversion time increase from 500ms (DS1820) to 750ms (DS18S20). This will yield 12-bits of resolution (LSb weight of 0.0625°C). In the time/temperature display of the Weather Station, you will certainly notice a difference in the resolution. In a relatively stable temperature environment, the DS1820 curve would look smooth compared to that of the DS18S20. The conversion time increase will be transparent to the Weather Station because the WS-1 software polls the sensor for a “conversion complete” flag. If any of you are using the DS1820 in other designs, be aware of the resolution and conversion time changes with the DS18S20.

Although your display may appear more “jagged” with the lower resolution DS18S20, do not confuse that for inaccuracy. Data has been presented on the superior accuracy and stability of the DS18S20. The DS18S20 is accurate to ±0.5°C. That means the 9 most significant bits of the thermometer have zero error. Anything less significant than the ninth bit does not contribute to the accuracy of the sensor, but they are quite useful for monitoring minute temperature changes on a relative basis (relative to the last measurement, for example). For example, if bit 12 of the readout increases by 1 from one readout to the next, that does not necessarily mean the absolute temperature increased by the LSb weight of 0.0625°C. The part is not accurate to the 12th bit. It does imply however, that there was an increase of some absolute magnitude less than 0.5°C (the rated accuracy) from the last reading. Also, do not confuse the lower resolution graph of the DS18S20 with the state machine glitching of the DS1820.

Some of you in the interest group may choose to modify the Weather Station code to add algorithms that essentially smooth the temperature vs. time graph for the DS18S20. Smoothing algorithms will have a negligible effect on the accuracy or time constant of the thermal sensing system. The WS-1 has a large amount of thermal mass (and thus a very long thermal time constant) and the sensor is not exposed to moving medium (i.e. wind). Dallas will also release a patch to the WS-1 software that will perform curve smoothing.

3.) Error notification. The DS18S20 relies on strict protocol for predictable operation. Unfortunately, in the real world, less than ideal activity on the 1-Wire™ interface due to noise, lightening, etc may cause the DS18S20 to enter an ambiguous state. The absence of an error flag in memory left us to decide to notify the user of an error condition by reserving 1 state of the 12-bit (4,096 possible) thermometer readout for that purpose. The error state for the DS18S20 results in a temperature of +85.0000°C, which corresponds to a value of AAh in byte0 (TEMPERATURE LSB), 00h in byte1 (TEMPERATURE MSB), 0Ch in byte6 (COUNT_REMAIN), and 10h in byte7 (COUNT_PER_C). This reading indicates an error state and the reading should be discarded. The patch to the WS-1 software will ignore such readings and display that last completed conversion.

To summarize, Dallas is continually trying to make improvements to its thermal management product line. The DS1820 provided a solution to users who wanted a digital I/O thermometer with the minimum number of interface signals. Although it did fill a niche quite well, its algorithm contained some inherent inaccuracies and instabilities. Removal of those inaccuracies, in addition to widening the voltage range, and offering new/smaller/lower cost packages was of primary concern in the design of the DS1820’s replacement, the DS18S20.

The B5 revision of the DS18S20 will be sampled to customers soon, and we would like for you to be among the first to “try them in the field”. We will replace your DS1820 inventory with DS18S20 on a “2-for-1” basis. The DS18S20 design has been submitted for reliability testing, which we expect to be completed in May2000; at that time, the DS18S20 will be released to mass production. We do expect that not all applications will be able to transition to the new device without some system modification, but we expect these cases will be the exception and not the rule. The only noticeable adverse difference you should see with the DS18S20 in a Weather Station with unmodified code is lower thermometer resolution and a possible error notification reading of 85.0000°C. Dallas will very soon release a patch to the WS-1 software that will make both issues transparent to the Weather Station display. Thank you for your involvement in the Weather Station interest group and in the Dallas temperature sensor product line. If you have any questions or comments, please contact Dan Awtrey at dan.awtrey@dalsemi.com.

DALLAS SEMICONDUCTOR

Dear Weather Station experimenter:

An anomaly has been detected in the temperature conversion algorithm of Dallas Semiconductor's thermal and battery management products that may affect some applications. The anomaly manifests itself as a random erroneous temperature measurement that can be as high as 1.0°C. The subsequent explanation of the problem and the suggested work-around assumes knowledge of the high resolution algorithm explained in Application Note 105, which is available on the Dallas website or faxback.

EXPLANATION OF THE PROBLEM

A digital state machine is used to control the temperature conversion algorithm. In normal operation, the state machine cycles between two states, watching for a control signal that signals the end of the conversion. Once this signal is detected, the next clock of the state machine sends it to the final state, suspending the conversion. As the state machine cycles between these two intermediate states, it increments the temperature register and reloads the slope counter, with the updated value in the slope accumulator.

Occasionally, through a mechanism still under investigation, the temperature conversion algorithm causes the state machine to cycle between the two intermediate states one more time after the end of conversion control signal is asserted. And, as is expected, this additional cycle increments the temperature register and reloads the counter from the slope accumulator for that new temperature. On the additional clock that sends the state machine to its final state, the slope counter is decremented. The result from the sensor is a temperature conversion result with a positive error in the range of:

$$+\frac{1}{COUNT\_PER\_C} \le Error(\deg C) \le +1.0$$

The magnitude of the error is dependent on the absolute temperature. Because the slope counter is decremented only once by the additional clock that sends the state machine to its final state, COUNT_PER_C - COUNT-REMAIN will be 1 when this error occurs. Therefore, the high resolution measurement will yield a result of:

$$Temp_{HIGHRES} = xxx.75 + \frac{1}{COUNT\_PER\_C}.$$

The error will be smallest for absolute temperatures slightly less than xxx.75°C (where COUNT_PER_C and COUNT_REMAIN approach their maximum difference), and approach 1°C for absolute temperatures slightly above xxx.75°C (when COUNT_PER_C and COUNT_REMAIN are nearly equal).

This behavior is illustrated in the following example. This example illustrates the result of a "good" measurement and that for a "bad" measurement in a simulated temperature sweep from 21°C to 24°C. The first figure, Fig. 1 gives the 9-bit direct reading for both a "good" and "bad" measurement. Note that the error, the difference between the two curves, ranges from 0.5°C to 1.0°C, depending upon the absolute temperature. This will be illustrated in Fig. 6.

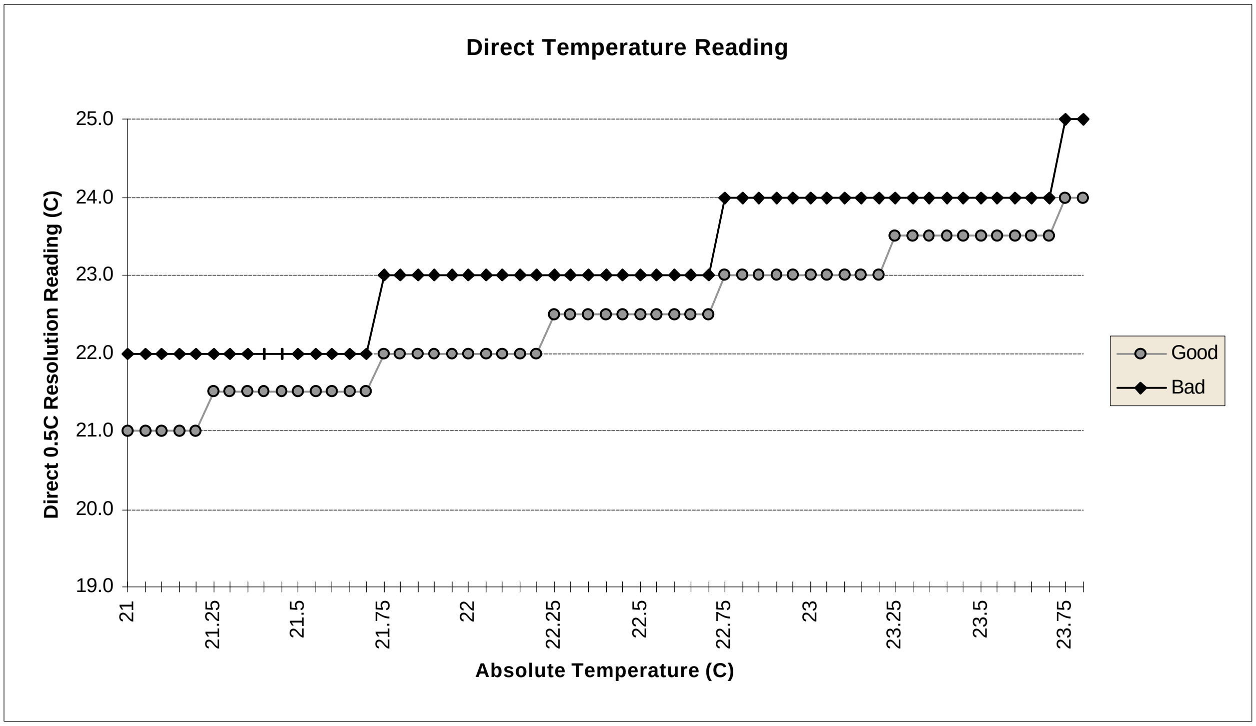

Fig. 1. Direct 9-bit Temperature Reading

The high resolution algorithm requires the truncation of the LSb from the direct 9-bit reading. Fig. 2 illustrates "good" and "bad" measurements after this truncation.

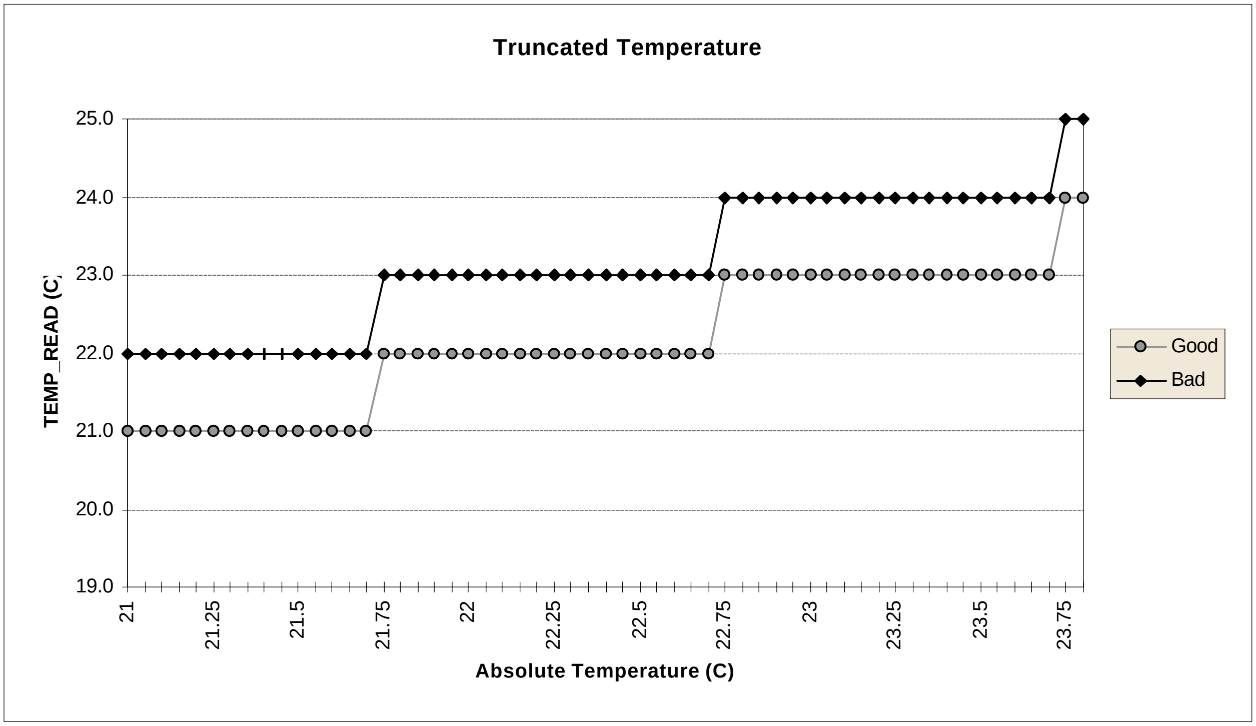

Fig. 2. Temperature Reading after Truncation (TEMP_READ)

The COUNT_PER_C register contains data that is programmed into the device at calibration. It sets the value of the slope accumulator at a given temperature. For a given temperature, this value is not the same for all devices. The absolute value depends upon the thermal characteristics of the oscillators. The values for this particular example were extracted from measured data on a DS1620, but may not be representative of the values in other DS1620 parts or other temperature sensors, in general. Nevertheless, it does illustrate the comment made earlier in that when the error occurs, the temperature register is incremented and the appropriate slope accumulator value is loaded. In this case, but not necessarily in all, the COUNT_PER_C value is 1 higher when the error occurs. This would not be true in a part which is calibrated such that two adjacent temperature values point to the same value for the slope accumulator.

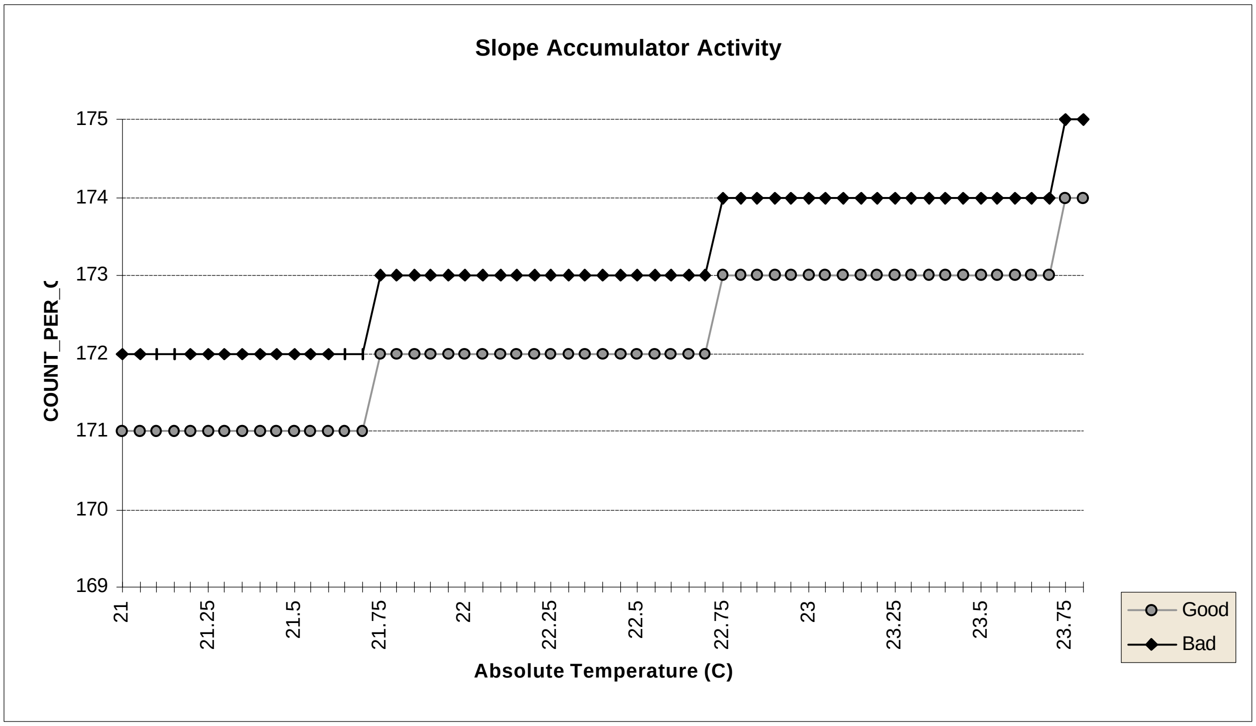

Fig. 3. Slope Accumulator Contents

Fig. 4 shows the value of the COUNT_REMAIN register in this temperature sweep example. It is the value left in the counter that was initialized to COUNT_PER_C for that given temperature. Recall that one additional clock cycle occurs in the event of an erroneous measurement; therefore, COUNT_REMAIN will be 1 less than COUNT_PER_C at a given temperature.

Using the calculation highlighted in Application Note 105 to obtain the high-resolution measurement, the measurement of Fig. 5 results. Note that the temperature always reads approximately xxx.75°C for an erroneous measurement. The final plot, Fig. 6, shows the error that one can expect when this anomaly occurs. As stated in the inequalities, the error varies from 1/COUNT_PER_C to 1°C in the high resolution mode. If only the 9-bit temperature is used, i.e., no high-resolution calculations are performed, the error will either be 0.5°C or 1.0°C, depending on the fractional value of the absolute temperature.

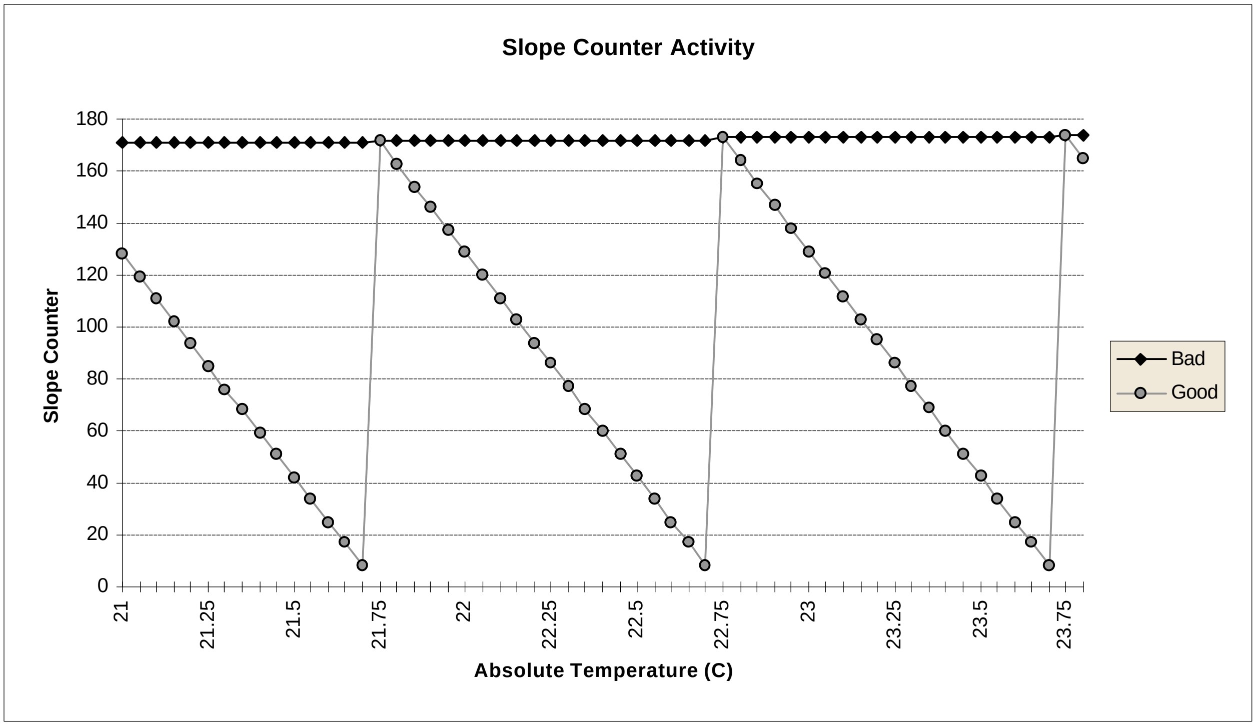

Fig. 4. Slope Counter Activity

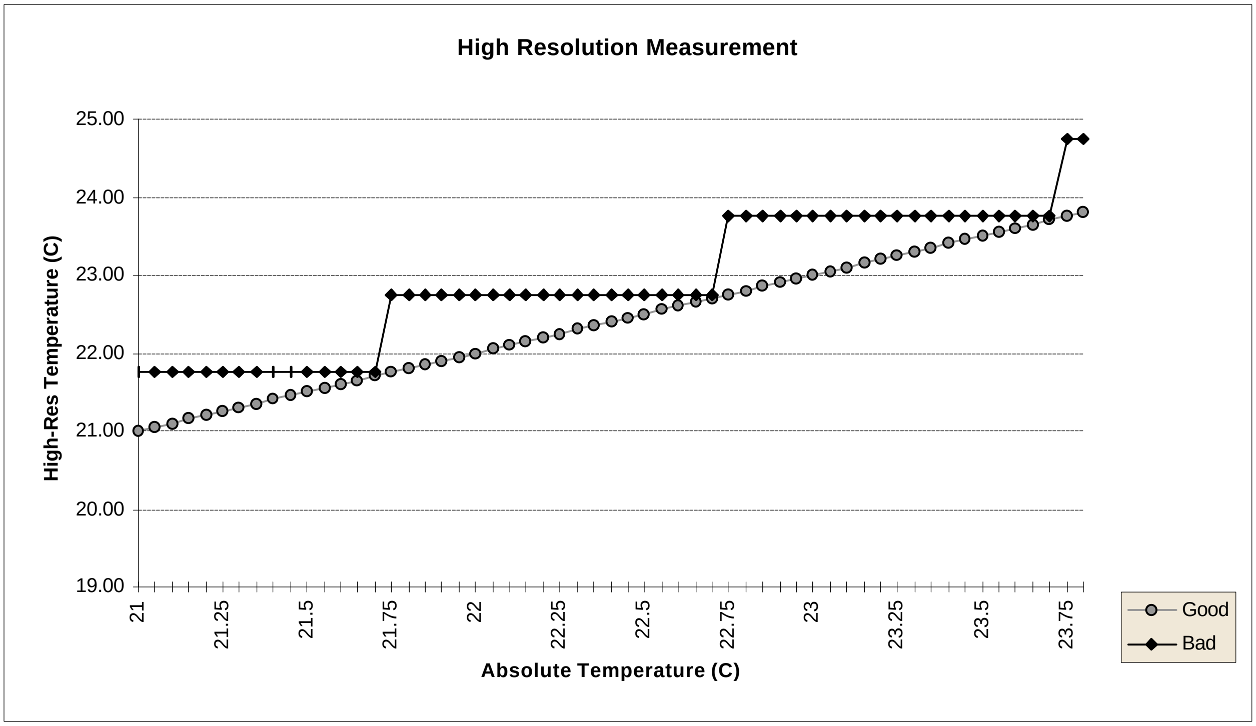

Fig. 5. Temperature Measurement after High-Resolution Calculation

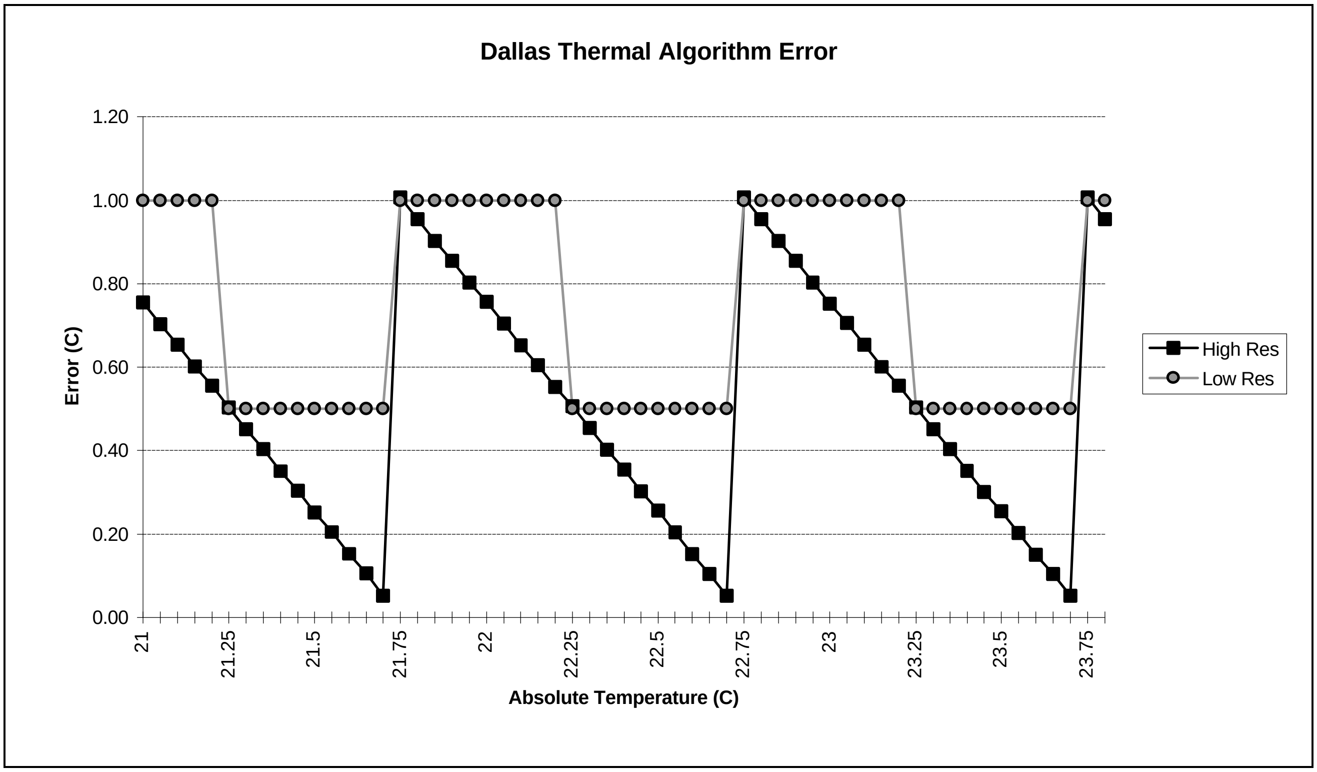

Fig. 6. Temperature Error Encountered when Anomaly Occurs

The occurrence of erroneous readings appear to be random in measurements we have performed thus far. Close examination of the example given, however, gives the user a means of understanding erroneous measurements based on the values of the registers read. Recall that only one additional clock cycle occurs to halt the state machine cycle; thus, COUNT_REMAIN will **ALWAYS** be one less than COUNT_PER_C for an erroneous measurement.

SUGGESTED ERROR DETECTION

The occurrence of the error is always associated with the condition COUNT_PER_C - COUNT_REMAIN = 1. Therefore, one could simply check for this condition before performing any further high-resolution calculations, and discard any measurements associated with this condition. However, COUNT_PER_C - COUNT_REMAIN = 1 does not necessarily constitute an error condition, and you could be discarding a valid measurement if you use the corrective action described above. This could be an acute problem in applications where the temperature is extremely stable, and the sensor returns a valid measurement such that COUNT_PER_C - COUNT_REMAIN = 1. If the method described above were implemented in this environment, several valid measurements in a row could be discarded. In measurements we have performed thus far, the error occurs very infrequently, about once per 5000 conversions. Never have we observed the erroneous condition occurring in two adjacent measurements.

Therefore, we suggest the algorithm depicted in the flowchart in Fig. 7 to detect an erroneous measurement and subsequently discard it. Basically, if the condition COUNT_PER_C - COUNT_REMAIN = 1 occurs on two measurements an a row, there is an extremely high likelihood that the measurement is valid. Similarly, if a conversion returns the above condition and the next returns something different, there is an extremely high likelihood that the measurement is invalid, and should be discarded.

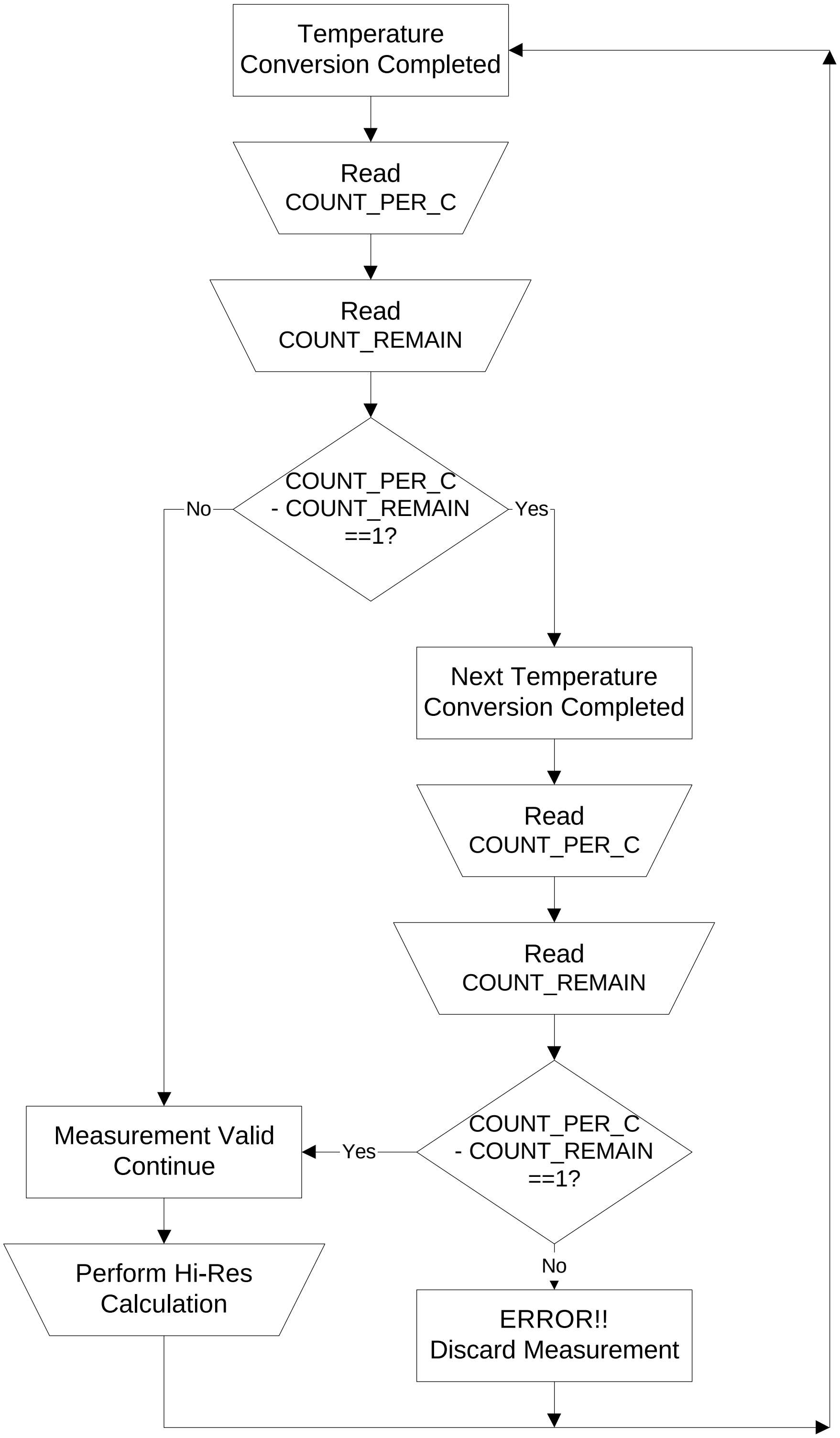
Temperature Conversion Completed
Read COUNT_PER_C
Read COUNT_REMAIN
COUNT_PER_C - COUNT_REMAIN ==1?
No
Yes
Next Temperature Conversion Completed
Read COUNT_PER_C
Read COUNT_REMAIN
COUNT_PER_C - COUNT_REMAIN ==1?
Yes
Measurement Valid Continue
Perform Hi-Res Calculation
No
ERROR!! Discard Measurement

Fig. 7. Suggested Method of Error Detection RETURN TO FAQ