User's Guide

NTC Thermistor to TMP6 Linear Thermistor Replacement Guide

ABSTRACT

Linear thermistors use the same hardware and software design methods commonly used with linear positive temperature coefficient or negative temperature coefficient thermistors. The purpose of this document is to provide insight on hardware and software design methods for converting a NTC thermistor system to a linear thermistor.

Table of Contents

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

Thermistors can be used in temperature sensing applications instead of digital temperature sensors due to their reduced cost, lower footprint, and faster response time. More information on the benefits of thermistors over digital temperature sensors can be found in *Temperature Sensing using Thermistors*.

1.1 NTC Thermistor Versus TMP6 Linear Thermistor Family

The two main thermistors in the market are the NTC and linear thermistors. NTC thermistors operate by changing the resistance as temperature increases or decreases. Linear thermistors change their effective resistance while a current is flowing through them, depending on the temperature. The largest difference between the two types is that NTC thermistor's resistance will decrease logarithmically while a linear thermistor's effective resistance will increase linearly as the temperature increases. The graph below shows the difference in resistance to temperature characteristics of a typical 10-kΩ NTC thermistor versus TI's TMP6 Linear Thermistor family, specifically the TMP6131 DEC package.

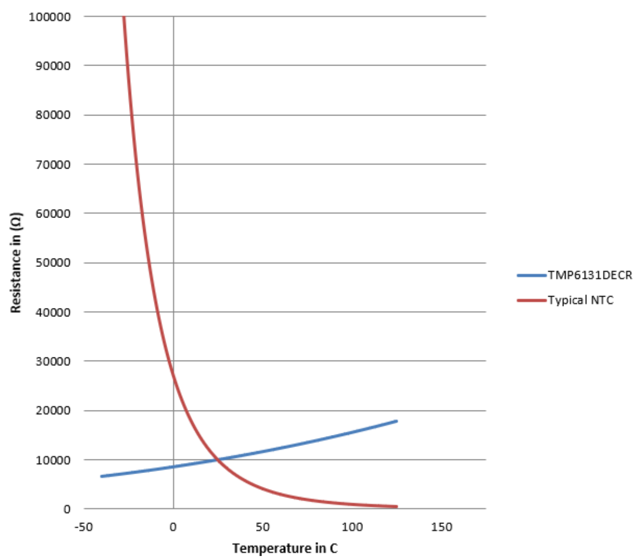

Figure 1-1. RT Curve of Typical NTC Thermistor vs. TMP6131DEC Thermistor

1.2 NTC/Linear Thermistor TCR

The temperature coefficient resistance (TCR) can be defined as the change in resistance as temperature changes for a device. Use Equation 1 to calculate the TCR measured in ppm/°C.

$$\text{Temperature Characteristic Resistance} = ((R2-R1)/R1\times(T2-T1))\times10^{\wedge}6 \quad (1)$$

Due to the linearity of the TMP61 thermistor, the device has a consistent TCR across a wide operating temperature range. Unlike an NTC thermistor, which is a purely resistive device, the TMP61 thermistor's effective resistance is affected by the current across the device and the effective resistance changes when the temperature changes. The TMP61 thermistor has a 6400 ppm/°C TCR (25°C) with a 0.2% typical TCR tolerance across the entire temperature range. However, this value does vary slightly depending on how the TMP61 thermistor is biased.

The TMP6 thermistor has many advantages while the NTC thermistor, at first glance, has an advantage of change in resistance at room temperature. The TMP6 thermistor can achieve the same or better accuracy with simple enhancements that TI provides.

1.3 NTC Versus Silicon-Based Linear Thermistor Trade-Offs

The configuration of the temperature sensing circuit in one's design will depend on any number of factors. While voltage-biased thermistors are simpler to construct, current-biased thermistors have a wider dynamic voltage range, greater stability, and higher accuracy across the temperature range for the output voltage V_{TEMP}.

The typical NTC thermistor has a tolerance range from (1% to 5%) between temp extremes, although this can be higher for some NTC thermistors, while the TMP61 thermistor has a tolerance of (0.5% to 1.5%) between extremes [–40°C, +150°C] which can be seen in Figure 1-2 below.

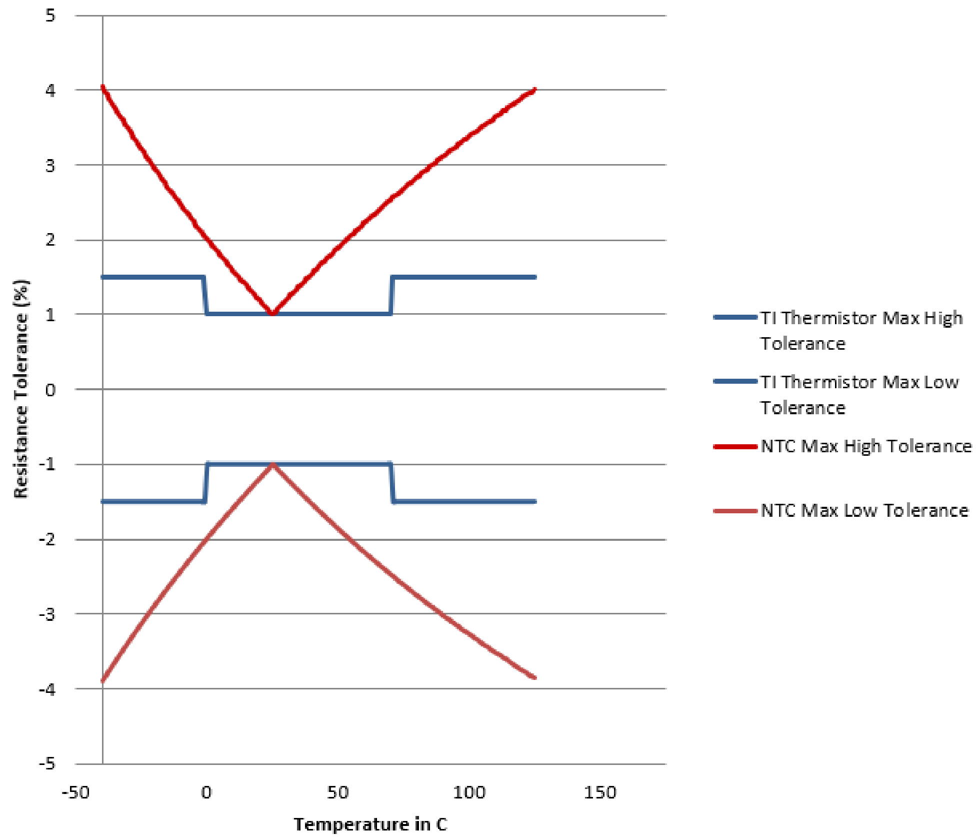

Figure 1-2. Resistance Tolerance of Typical NTC Thermistor vs. TMP6 Linear Thermistor

The thermistor's greatest strength is its simplicity in design. With a voltage-biased or current-biased network, a voltage drop across the thermistor or a current can be passed through the thermistor to be measured for sensing. The main configurations for a thermistor circuit are voltage-biased, as demonstrated in the voltage divider configuration in Figure 1-3, or current-biased, as shown in Figure 1-4. The output voltage, V_{TEMP}, can be fed to an ADC for digital processing of the temperature data in the MCU.

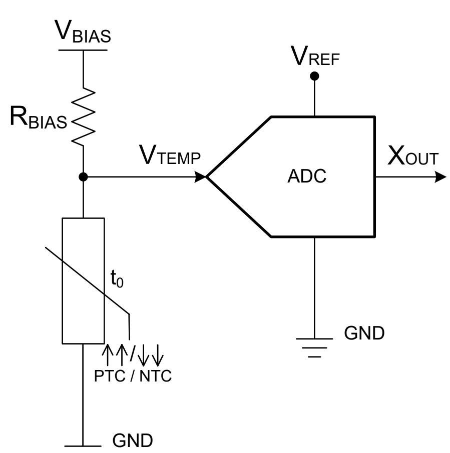

Figure 1-3. Thermistor Voltage Divider Circuit

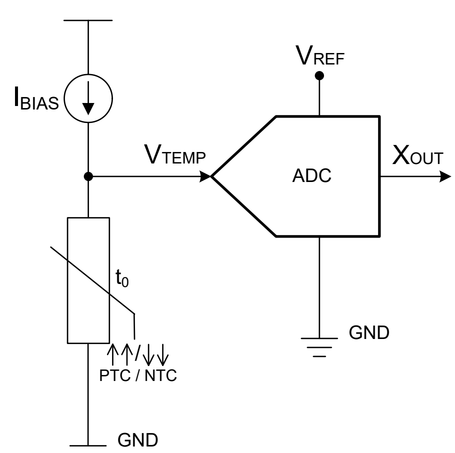

Figure 1-4. Thermistor Current Source Circuit

When looking at the characteristics of an NTC thermistor, it should be noted that when the ambient temperature is hot, the NTC thermistor presents a challenge to see what the temperature is at due to the low sensitivity at these higher temperatures. For easier software processing of incoming temperature data, one might need to linearize their NTC thermistor's R-T table. For NTC thermistors, this typically requires a fixed resistance value in parallel with the thermistor. Figure 1-5 shows a typical NTC thermistor voltage divider circuit with a parallel resistor of R_P and a bias resistor R_{BIAS}. A comparison of the voltage response of a typical NTC thermistor voltage divider, typical linear thermistor voltage divider and NTC thermistor voltage divider with parallel resistance is shown in Figure 1-6.

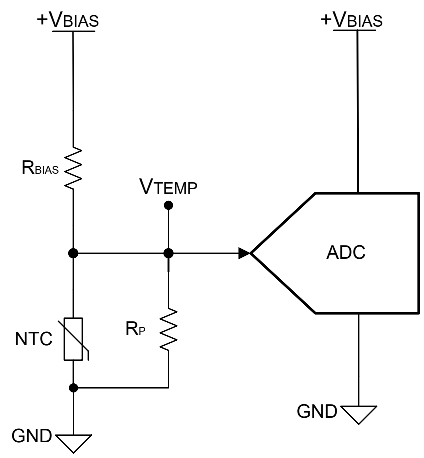

Figure 1-5. NTC Thermistor With Parallel Resistor

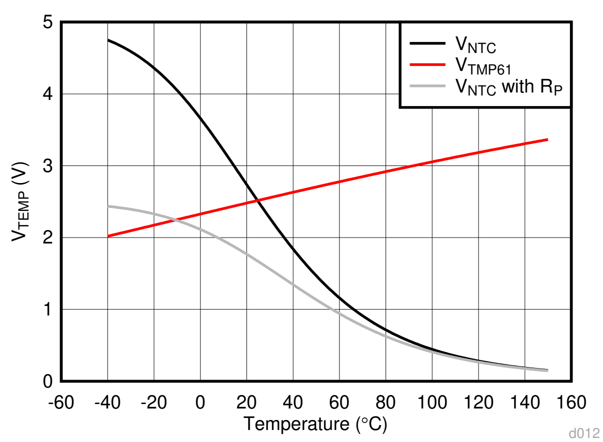

Figure 1-6. NTC Thermistor With and Without a Linearization Resistor vs. TMP61 Thermistor Temperature Voltages

Thiscan work for a limited temperature range, but it is more difficult to linearize an NTC thermistor over the entire temperature range and cannot be done with hardware only. Conversely, linear thermistors are fabricated with a linear R-T characteristic curve, which eliminates the requirement for a fixed-value resistor in parallel with the thermistor.

System designers may require a calibration on thermistors to ensure accuracy across the operating range of the device. To achieve higher accuracy across this range, NTC thermistors require multiple-point calibration at different temperature values such as -40°C, 25°C, and 125°C due to the non-linearity of an NTC thermistor. The same theory can be applied to a linear thermistor, but because of the linearity, only a single-point calibration (at 25°C, for example) is needed. Therefore, using a linear thermistor will save on manufacturing time and reduce memory demands on the MCU for temperature processing. Refer to the Thermistor Design Tool for additional information on calibration with the TMP6 linear thermistor family of devices.

1.4 TMP6 Accuracy

Following the design steps included in the document while taking various designs including Rbias tolerance/ PPM, Temperature Range of interest, Polynomial/LUT, Oversampling, Filtering and Calibration into account , the following table summarizes the accuracy in °C that can be achieved with the TMP6 linear thermistor.

Bias Resistor	Temperature Range	With room temperature calibration	
		Polynomial 12bit	Polynomial + (Oversampling or Filtering)
± 0.1%, 10 PPM	0 to 70°C	+/- 0.74°C	+/- 0.54°C
± 0.1%, 10 PPM	Full Range	+/- 0.92°C	+/- 0.72°C
± 0.5%, 25 PPM	0 to 70°C	+/- 1.30°C	+/- 1.10°C
± 0.5%, 25 PPM	Full Range	+/- 1.60°C	+/- 1.40°C
± 1%, 100 PPM	0 to 70°C	+/- 1.94°C	+/- 1.74°C
± 1%, 100 PPM	Full Range	+/- 2.50°C	+/- 2.30°C

Figure 1-7. TMP6 Thermistor Accuracy in °C

The accuracy numbers in the chart include assumptions of an ideal Vbias at 5 V and an ideal Vref with no ADC error.

2 Typical NTC Thermistor Design Considerations

Let us follow an example of this design process:

Table 2-1. System Requirements for Example Voltage-Biased Temperature Sensing Circuit

Temperature Range		Bias Voltage
T_{MIN}	T_{MAX}	V_{BIAS}
-40 C	125 C	5V

2.1 Voltage-Biased NTC Thermistor Network

The simplest construction of the NTC thermistor-based temperature sensing network is the one shown in Figure 2-1. For the ADC, we will use 12-bit resolution and a reference voltage of 5 VDC. The 12-bit resolution is an acceptable resolution for measurement while the 5 VDC reference voltage simplifies power rail requirements. These design decisions result in the voltage response shown in Figure 2-2.

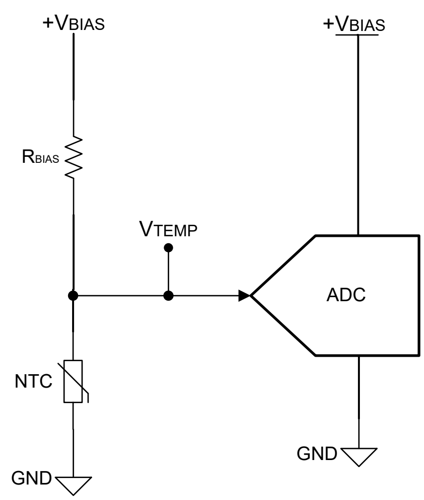

Figure 2-1. Simple Voltage Biased NTC Thermistor Schematic

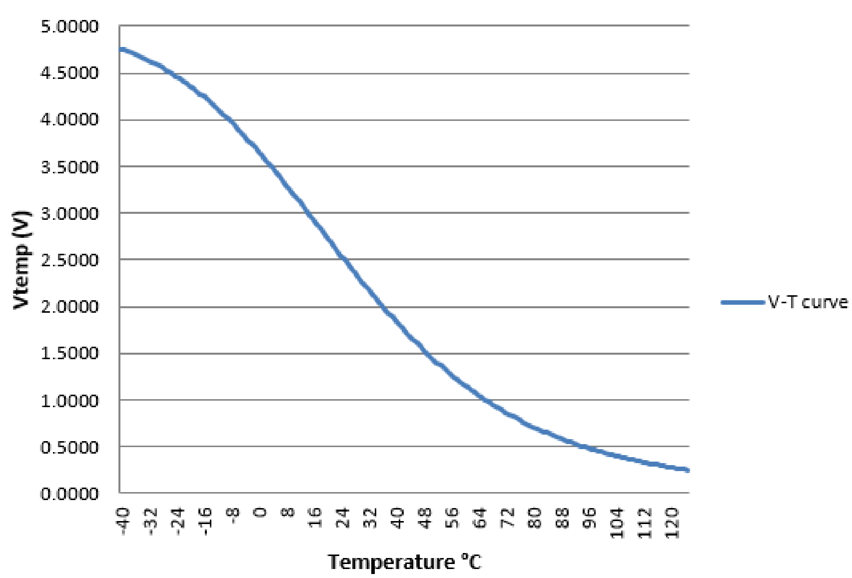

Figure 2-2. NTC Thermistor Voltage Response (5 V)

While this voltage-biased network is very simple, reducing the bill of materials cost required. The temperature output V_{TEMP} shows non-linear behavior at the temperature extremes.

2.2 Pinouts/Polarity

The TMP6 linear thermistor is a 2-pin device and therefore a pin-to-pin replacement of an NTC thermistor. These devices are also footprint-compatible as the TMP6 thermistor is offered in 0402 (1005-mm) and 0603 (1608-mm) packages. It should be noted that the 0603 part can also be used on an 0805 footprint. See Section 6.2 for more information.

It is important to note that the TMP6 thermistor has polarity (see Figure 2-3).

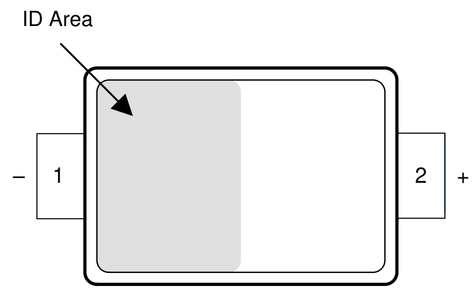

Figure 2-3. TMP6 Thermistor DYA Package 2-Pin SOT-5X3 Bottom View (Angled)

TI uses a special silicon process where the doping level and active region areas devices control the key characteristics (the TCR and nominal resistance). The device has an active area and a substrate due to the polarized terminals. Connect the positive terminal to the highest voltage potential and the negative terminal to the lowest voltage potential to ensure proper operation. If the voltage across the pads is reversed, the thermistor will appear properly until the p-n junction starts to conduct (approximately 0.6 V), then the thermistor I-V characteristics will break down. This will result in a measurement of 0.6 V across the device terminals if configured reversely.

2.3 Converting NTC Thermistor Hardware Design to TMP6 Linear Thermistor Design

For a simple NTC thermistor design, the converting from an NTC thermistor to the TMP6 Linear Thermistor is straightforward. The only hardware change is to swap the NTC thermistor and the TMP61 thermistor, while R_{BIAS} can be left as is. The resulting transformation can be seen in Figure 2-4 below.

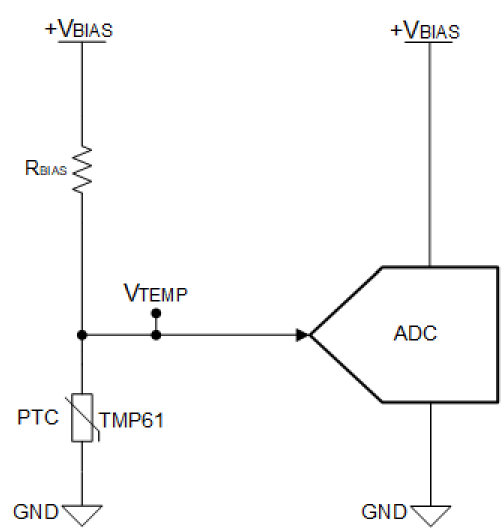

Figure 2-4. TMP6 Linear Thermistor Voltage Divider Circuit

The circuit above results in the following voltage response. Notice how the voltage response is positive.

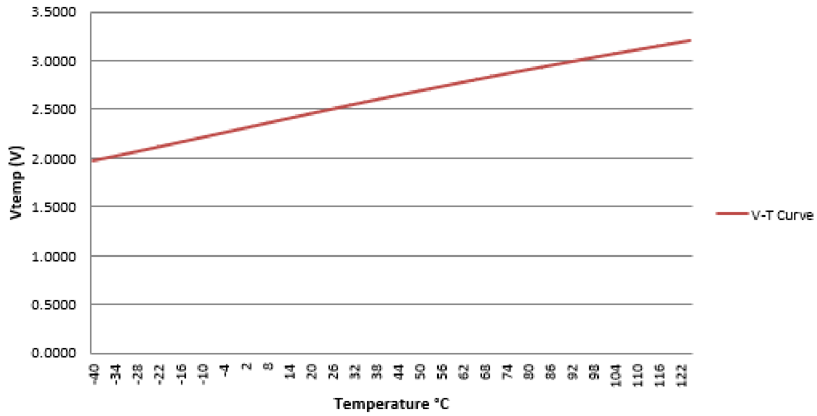

Figure 2-5. TMP6 Linear Thermistor Positive Voltage Response

A positive voltage response can be used for new designs, but matching a negative response that the original NTC thermistor circuit generated may be useful for old designs. To provide a negative voltage response with the TMP6 Linear Thermistor, the bias resistor and TMP6 Linear Thermistor must switch positions. After redesigning, you can see the resulting negative voltage response below.

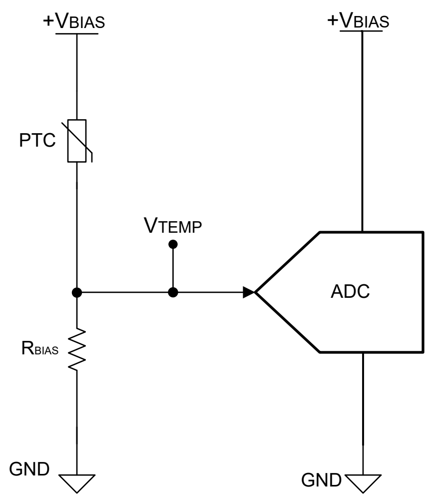

Figure 2-6. TMP6 Linear Thermistor Circuit for Negative Voltage Response

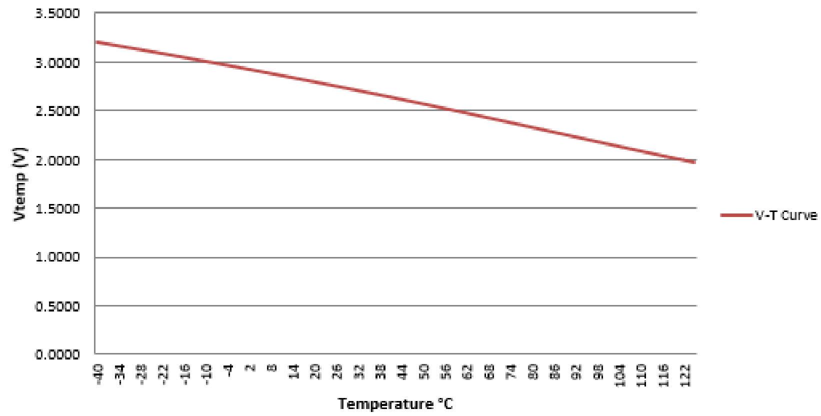

Figure 2-7. TMP6 Linear Thermistor Negative Voltage Response

2.4 Simple Look-Up Table

Now that we have the hardware design completed, the TI thermistor design tool can be used to generate software for internal resistance-temperature conversion within the MCU. Take a look at how to generate the code snippets from the Thermistor Design Tool.

We will use the following design parameters:

5-V, 10-kΩ bias resistor, 12-bit ADC, TMP6131DYA.

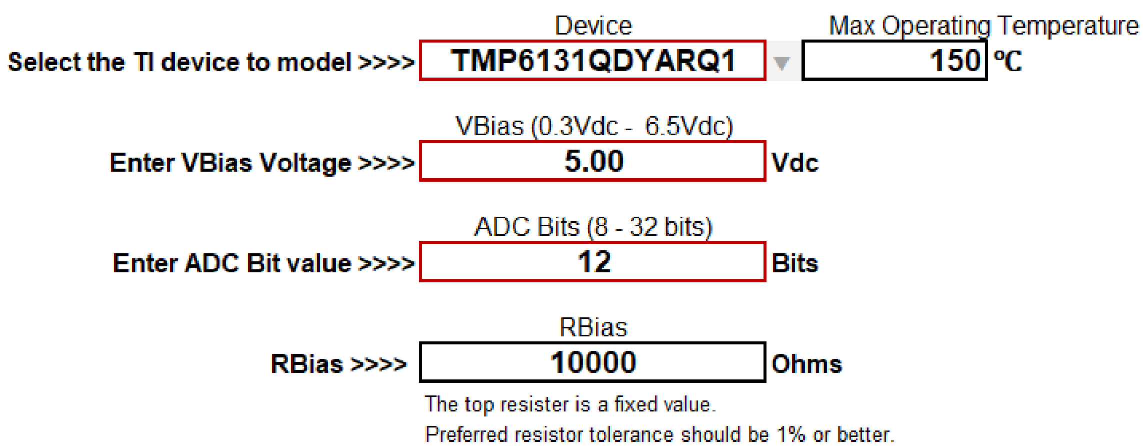

Figure 2-8. Thermistor Design Tool Parameters

Next we can move to the *Device Resistance Tables* tab. Here we can find both a 1°C and 5°C step look-up table. This page dynamically populates the resistance tables depending on the design parameters initially set. The 5°C Look-Up Table is shown below:

5 °C Steps Example Code

Line #	Temperature (°C)	Min Resistance (Ω)	Typical Resistance (Ω)	Max Resistance (Ω)
1	-40	6501	6600	6699
2	-35	6710	6812	6914
3	-30	6927	7032	7138
4	-25	7151	7260	7369
5	-20	7384	7496	7609
6	-15	7624	7740	7856
7	-10	7871	7991	8111
8	-5	8126	8250	8374
9	0	8431	8517	8602
10	5	8703	8791	8878
11	10	8981	9072	9163
12	15	9267	9361	9454
13	20	9560	9657	9754
14	25	9861	9961	10060
15	30	10169	10272	10375
16	35	10484	10590	10696
17	40	10807	10916	11025
18	45	11137	11250	11362
19	50	11475	11591	11707
20	55	11820	11940	12059
21	60	12173	12296	12419
22	65	12534	12661	12787
23	70	12902	13033	13163
24	75	13212	13413	13614
25	80	13595	13802	14009
26	85	13985	14198	14411
27	90	14385	14604	14823
28	95	14792	15018	15243
29	100	15209	15440	15672
30	105	15634	15872	16110
31	110	16068	16313	16558
32	115	16512	16764	17015
33	120	16965	17224	17482
34	125	17428	17694	17959
35	130	17901	18174	18446
36	135	18384	18664	18944
37	140	18878	19165	19453
38	145	19382	19677	19972
39	150	19897	20200	20503
40	155	#N/A	#N/A	#N/A
41	160	#N/A	#N/A	#N/A
42	165	#N/A	#N/A	#N/A
43	170	#N/A	#N/A	#N/A

Figure 2-9. 5°C LUT

On this page, we can find C code for the look-up table, as well.

Example of 5°C step C code **Copy the content of the box below, then paste into your C code**

```
int THRM_Res_S_Table[43][2] = {                // THRM short table
{ -40 , 6600 } ,
{ -35 , 6812 } ,
{ -30 , 7032 } ,
{ -25 , 7260 } ,
{ -20 , 7496 } ,
{ -15 , 7740 } ,
{ -10 , 7991 } ,
{ -5 , 8250 } ,
{ 0 , 8517 } ,
{ 5 , 8791 } ,
{ 10 , 9072 } ,
{ 15 , 9361 } ,
{ 20 , 9657 } ,
{ 25 , 9961 } ,
{ 30 , 10272 } ,
{ 35 , 10590 } ,
{ 40 , 10916 } ,
{ 45 , 11250 } ,
{ 50 , 11591 } ,
{ 55 , 11940 } ,
{ 60 , 12296 } ,
{ 65 , 12661 } ,
{ 70 , 13033 } ,
{ 75 , 13413 } ,
{ 80 , 13802 } ,
{ 85 , 14198 } ,
{ 90 , 14604 } ,
{ 95 , 15018 } ,
{ 100 , 15440 } ,
{ 105 , 15872 } ,
{ 110 , 16313 } ,
{ 115 , 16764 } ,
{ 120 , 17224 } ,
{ 125 , 17694 } ,
{ 130 , 18174 } ,
{ 135 , 18664 } ,
{ 140 , 19165 } ,
{ 145 , 19677 } ,
{ 150 , 20200 } ,
{ 155 , #N/A } ,
{ 160 , #N/A } ,
{ 165 , #N/A } ,
{ 170 , #N/A } ,
} // Remove all lines associated with unused temperatures and update the table length
```

Figure 2-10. 5°C LUT C Code

From simply implementing a look-up table, a comparison between the TMP6 thermistor accuracy and a typical NTC thermistor across the full operating temperature range is plotted below.

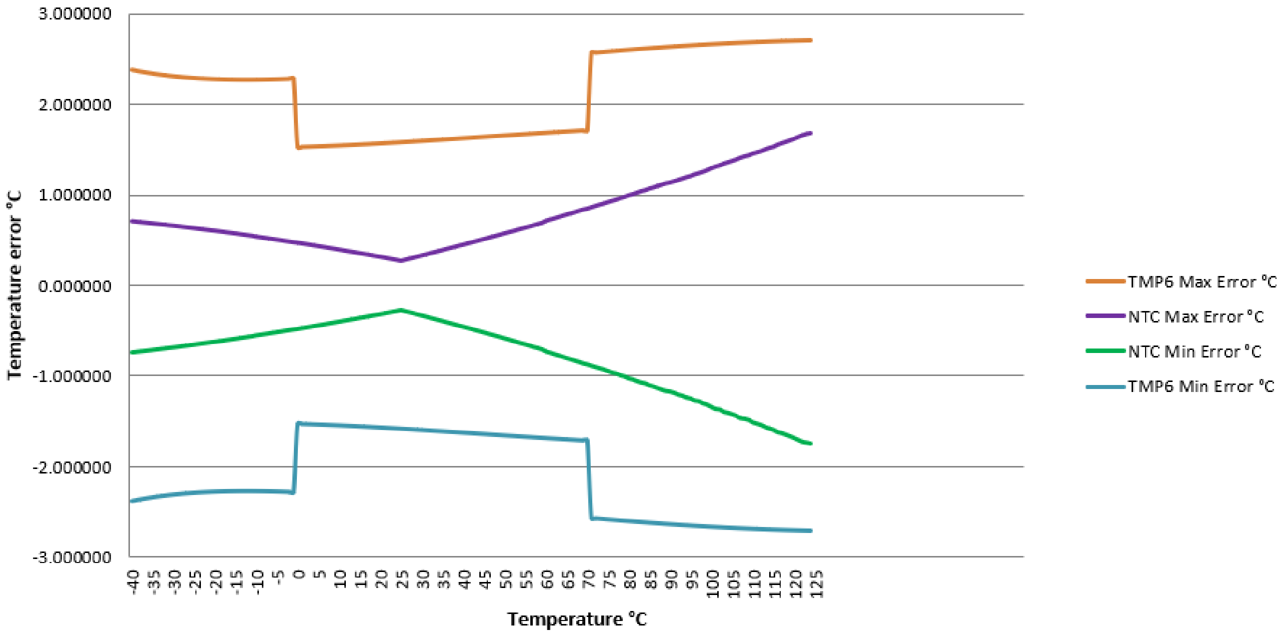

Figure 2-11. Un-Corrected Thermistor Accuracy Comparison

3 Software Changes

When the output of the thermistor circuit is sensed by an ADC and converted to digital information for the MCU to process, the output must be converted to a temperature value. One of the common software R-T conversion methods is using a look-up table. This involves pre-populating a table of resistances and the associated temperature value for those resistances. The code will determine which resistance value more closely aligns with the expected temperature value by interpolating between the points. This method results in a very simple setup for the R-T table but puts high demand on the flash memory requirements for the MCU and a lengthy array-parsing program. It can also be inaccurate due to system errors such as tolerance variations and temperature coefficients which may lead to divergence from the ideal R-T table.

A second method of temperature conversion that can save on memory is the Steinhart-Hart equation below. The equation can be implemented into the temperature sensing code for mapping to the R-T curve of the thermistor:

$$1/T = A + B \times \ln(R) + C \times \ln^3(R) \quad (2)$$

where T = temperature (in Kelvin); R = measured resistance value; A,B,C are the calculated coefficients.

With the use of the TMP6 linear thermistor, however, a much better algorithm is available for converting is the 4th order polynomial regression model below:

$$T = A_4 \times R^4 + A_3 \times R^3 + A_2 \times R^2 + A_1 \times R + A_0 \quad (3)$$

where T = temperature (in Celsius); R = measured resistance value; $A_{0\text{-}4}$ are the calculated polynomial coefficients.

This approximation works well because of the linearity of the device and does not work for a non-linear NTC thermistor. The polynomial coefficients for the polynomial regression model can be generated in the Thermistor Design Tool.

3.1 Firmware Design Considerations

The recommended method for calculating the temperature values from TI's TMP6 Linear Thermistor portfolio is the 4th order polynomial regression. This is the most accurate and fastest method to calculate the temperature- with no look-up table needed. Moving on to the *4th Order Polynomial TMP vs. Res* tab, we find both the

Quartic Function and Regression model which provide the 4th order polynomial for the calculated temperature/ resistance of the device.

4th order polynomial

Quartic Function

R(Ω) = A4*(T^4) + A3*(T^3) + A2*(T^2) + A1*T + A0 — ***Resistance as a function of temperature***

	Coefficients	
	8.516625E+03	A0
	5.404381E+01	A1
	1.497209E-01	A2
	-5.699002E-05	A3
	7.918690E-07	A4
Temperature	25.0	°C	Enter temperature here to get the resistance
	77.0	°F	
	298.2	°K	
Resistance	9961	Ohms	Calculated resistance from the 4th order polynomial above

Regression

T°C = A4*(R^4) + A3*(R^3) + A2*(R^2) + A1*R + A0 — ***Temperature as a function of resistance***

	Coefficients	
	-2.694795E+02	A0
	5.094962E-02	A1
	-3.143945E-06	A2
	1.182160E-10	A3
	-1.810821E-15	A4
Resistance	*10052*	Ohms	Enter resistance to get the temperature
Temperature	26.6	°C	Calculated temperature from the 4th order polynomial regression above
	79.8	°F	
	299.7	°K	

Figure 3-1. 4th Order Polynomial

Here we provide the C code that can be readily implemented into a system designers software to calculate the temperature of the TI TMP6 linear thermistor chosen.

EXAMPLE OF C CODE   Copy the content of the box below, then paste into your C code

```c
// 4th order polynomial equations to calculate the temperature of the thermistor--------------------------------------------------------------
// C code examples only  (NOTE: this code example is based on floating point math)

    unsigned int RBias =           10000 ;    // set the value of the top resistor
    float VBias =                   3.30 ;    // set the VBIAS voltage
    unsigned int ADC_BITS =         4096 ;    // set the number of bits based on you ADC (2^# of ADC Bit Value)
    float VTEMP = 0;                          // set up the variable for the measured voltage
    float THRM_RES = 0;                       // setup the variable for the calculated resistance
    float THRM_TEMP = 0;                      // setup the variable for the calculated temperature


    float Thermistor(int raw_ADC)             // send the ADC bit value to the calculation function
    {
      // THRM calculations - 4th order polynomial regression
      VTEMP = 0;                              // reset these variables to zero in order to recalculate the new factors
      THRM_RES = 0;                           // reset these variables to zero in order to recalculate the new factors
      THRM_ADC = raw_ADC

      float THRM_A0 =        -2.694795E+02 ;
      float THRM_A1 =         5.094962E-02 ;
      float THRM_A2 =        -3.143945E-06 ;
      float THRM_A3 =         1.182160E-10 ;
      float THRM_A4 =        -1.810821E-15 ;

      VTEMP = (VBias/ADC_BITS) * THRM_ADC;    // calculate volts per bit then multiply that times the ADV value
      THRM_RES = VTEMP/((VBias - VTEMP)/RBias);    // calculate the resistance of the thermistor
      THRM_TEMP = (THRM_A4 * powf(THRM_RES,4)) + (THRM_A3 * powf(THRM_RES,3)) + (THRM_A2 * powf(THRM_RES,2)) + (THRM_A1 * THRM_RES) + THRM_A0;    // 4th order regression to get temperature
      return THRM_TEMP;

    }
```

Figure 3-2. 4th Order Polynomial C Code

3.2 Oversampling

Oversampling and averaging temperature measurements in a first-in-first-out (FIFO) sequence can improve measurement resolution and signal-to-noise ratio. This is recommended when using less than a 12-bit ADC, although it can applied to 12-bit or 14-bit ADCs, as well. After the ADC reads the bit value and your code calculates the temperature, you can store that value in an array. As a new value comes into the array, the oldest sample is dropped as all the other samples are shifted to the next corresponding cell, thus creating a FIFO. The averaging method can be applied to any of the values used in temperature conversion such as the temperature, the ADC bit value, the divider voltage, or even the calculated resistance. For every 8 oversamples the resolution will increase by 2 bits. 16 oversamples will increase a 10-bit ADC to 14 bits of resolution. The figures below demonstrate two methods on how the oversampling can be implemented. See the *Averaging* tab of the Thermistor Design Tool for more information.

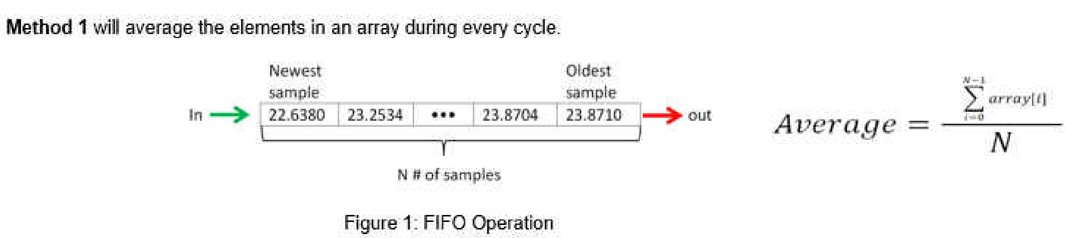

Figure 3-3. Oversampling Method 1

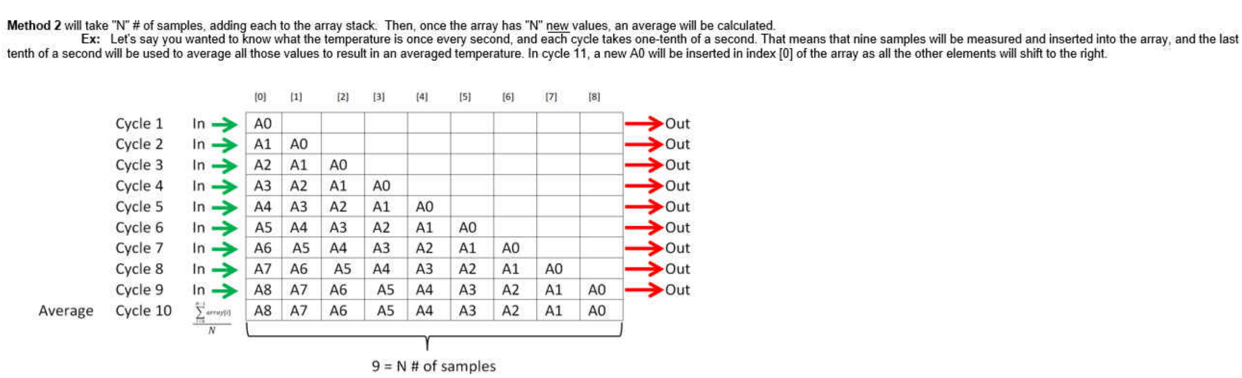

Figure 3-4. Oversampling Method 2

An example of the C code for Method 1 provided from the Thermistor Design Tool can be seen in Figure 3-5.

EXAMPLE OF C CODE **Copy the content of the box below, then paste into your C code**

```
// FIFO averaging of the data sample at predetermined intervals and averaged across the last (x) samples-----------------------------------------------
// C code examples only

// (1) Method one will read the ADC and average the last "N" values as set in the "#define Tmp_1_length"
//  FIFO setup of the temporary arrays and define the filter depth (samples to average)
#define Tmp_1_length 16                    // Sample length for the averaging filter for oversampling
float Tmp_1_array[Tmp_1_length];           // The FIFO arrays for averaging the ADC value

float ADC_AVG = 0;                         //  This is the averaged ADC value over (x) samples
float ADC_Value = 0;                       // this is the most recent ADC value captured
int i = 0;                                 //  set to 0
float sum_array_1 = 0;                     //  set to 0

void FIFO_AVG(void)
{
    // FIFO to average thermistor temperature //
    i = 0;                                 // reset to 0
    sum_array_1 = 0;                       // reset to 0
    for(i = 0; i < Tmp_1_length - 1; i++)  // shift the array as a FIFO and drop the last data value
    {
      Tmp_1_array[i] = Tmp_1_array[i + 1];           // Makes all the array indexes equal to the number after them
    }
    Tmp_1_array[Tmp_1_length - 1] = ADC_Value;       // add the new value to the beginning of the array
    for(i = 0; i < Tmp_1_length; i++)                // sum the array
    {
      sum_array_1 += Tmp_1_array[i];                 // add all of the array elements
    }
    ADC_AVG = sum_array_1 / Tmp_1_length;            // divide the sum of the array to get an average
}

//  Read the ADC and place the bit value into ADC_Value
// Call the ADC_AVG function to get the last ADC value added and averaged into the array
FIFO_AVG();
//  The ADC average value will be placed into ADC_AVG register
```

Figure 3-5. Oversampling Example C Code

Figure 3-6 and Figure 3-7 below you can see the impact of filtering the TMP6331 Thermistor's raw data using as 12-bit ADC. After a 32x oversample, the data aligns with the reference probe more closely.

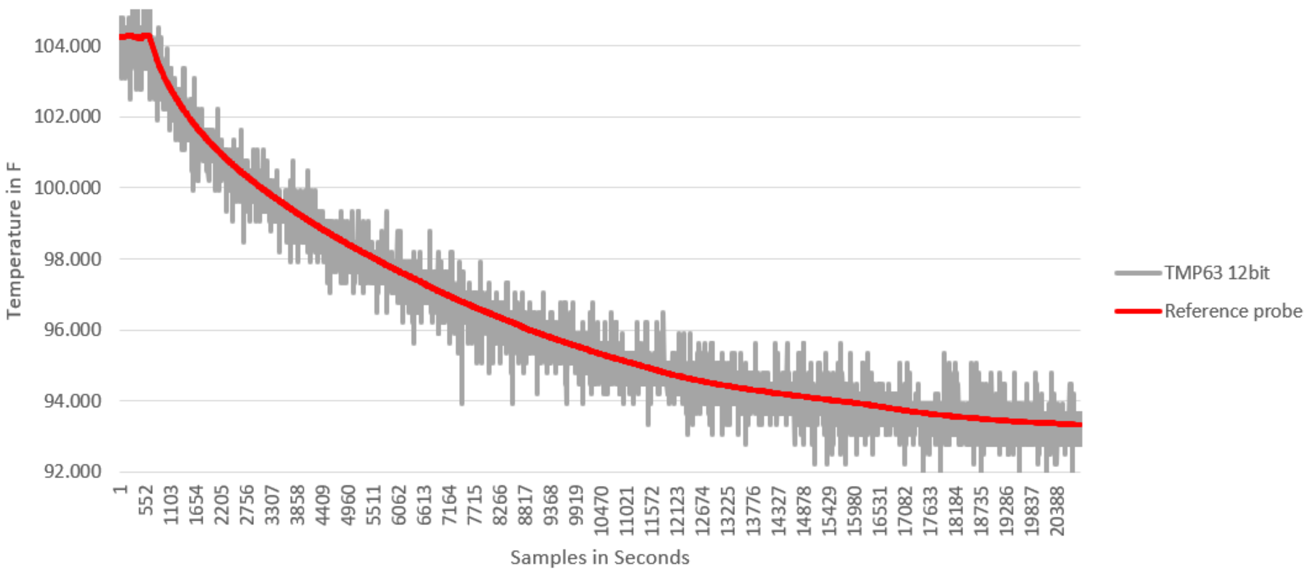

Figure 3-6. TMP6331 Thermistor Data With No Oversampling

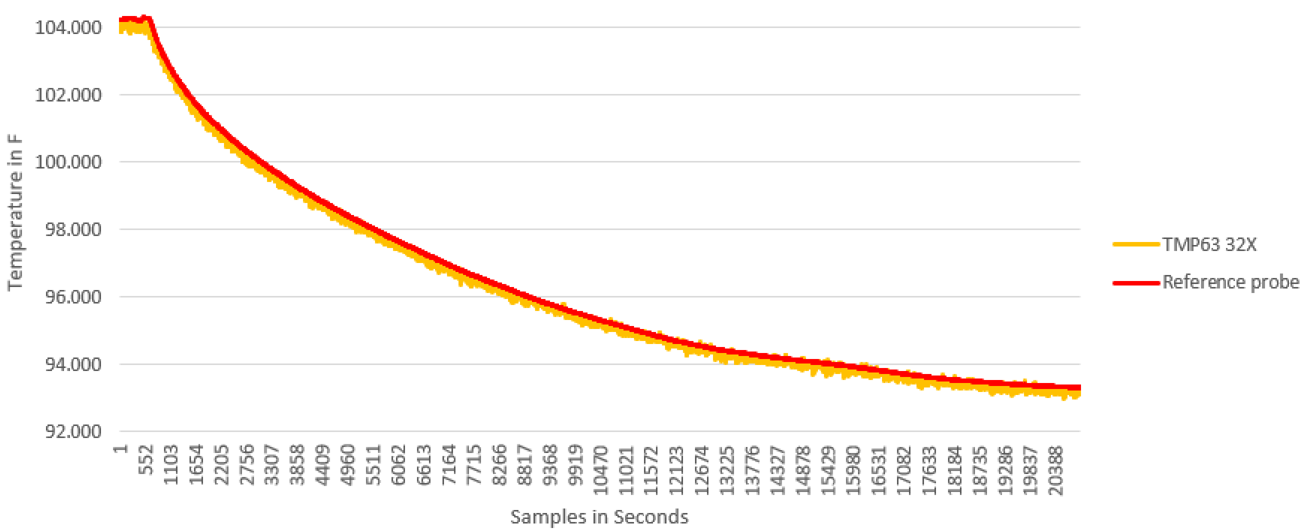

Figure 3-7. TMP6331 Thermistor Data With 32x Oversampling Applied

3.3 Low-Pass Filtering in HW Versus SW

Noise can cause erroneous temperature measurements, which is why many designers choose to add an RC filter in hardware to filter out noise coming from the system. But instead of filtering in hardware, you can use this method to eliminate the need for the extra resistor and capacitor, enabling greater board and cost savings. Implementing your filter in software gives you more control over the filter's response by changing the alpha value in real time. Additionally, having the ability to set the filtered temperature minimizes your start-up time.

There are three variables needed for the firmware based low-pass filter:

1. Alpha
2. Measure Temp
3. Filtered Temp

Alpha: This variable controls how much noise is filtered out.

Measured Temp: This variable stores your calculated, pre-filtered temperature reading.

Filtered Temp: This variable stores the resulting temperature after having passed the temperature value through the filter.

The firmware low-pass filtering is executed by the following equation:

Low pass filter equation:

$$Y(n) = (1 - \alpha) \times Y(n-1) + (\alpha \times X(n)) \tag{4}$$

where

- Y = Filtered Temp
- α = Alpha
- X = Measured Temp

Simplifying...

$$Y(n) = Y(n-1) - (\alpha \times (Y(n-1) - X(n))) \tag{5}$$

Simplifying further...

$Y(n) = Y - (\alpha \times (Y - X))$ meaning Filtered_Temp = Previous_Filtered_Temp - (Alpha * (Previous_Filtered_Temp - Meas_Temp)) (6)

On the *Low-Pass Filter* tab of the Thermistor Design Tool, you can adjust the alpha and samples per second values to change the filter. In Figure 3-8, you can see that the alpha was set to 0.8. The result seen in Figure 3-9 is that the resulting temperature data after implementing the low pass filter does not change much from the raw data.

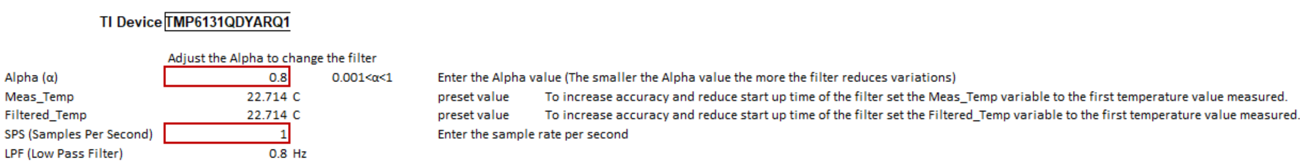

Figure 3-8. Low-Pass Filter Setting With 0.8 Alpha Value

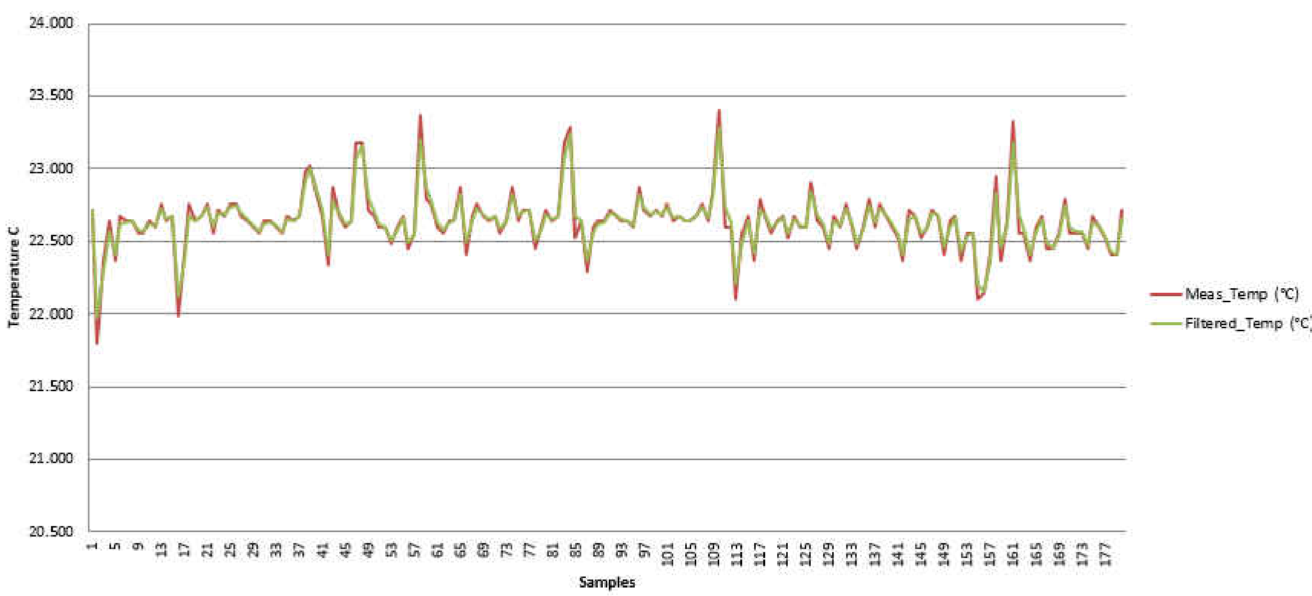

Figure 3-9. Low-Pass Filter Response With 0.8 Alpha Value

After adjusting the alpha value, a filtered response for an alpha value of 0.2 is shown below:

Figure 3-10. Low-Pass Filter Response With 0.2 Alpha Value

Here you can see that the filtered temperature data comes out to be much smoother than the raw data.

An example of the C code for the low pass filter from the Thermistor Design Tool can be seen in Figure 3-11.

EXAMPLE OF C CODE **Copy the content of the box below, then paste into your C code**

```
// Low Pass Filter using an Alpha and preset values to create a filtered or smoothed temperature response ------------------------------------------------
// C code examples only

//Floating Point

float Meas_Temp = 2! // It's assumed that you have read the ADC and calculated your temperature
float Filtered_Temp = // Set a default filtered temperature value equal to the first measured value to reduce the ramp time
float Alpha = 0.1;      // .001<α<1, adjust as required

// Insert this line of code after the temperature calculation (remember to add the offset to your temperature values)
Filtered_Temp = Filtered_Temp - (Alpha * (Filtered_Temp - Meas_Temp));
```

Figure 3-11. Low-Pass Filter Example C Code

After oversampling and implementing the low-pass filter algorithm the TMP6 thermistor can show improved performance. A comparison between the corrected TMP6 thermistor and a typical NTC thermistor can be shown below:

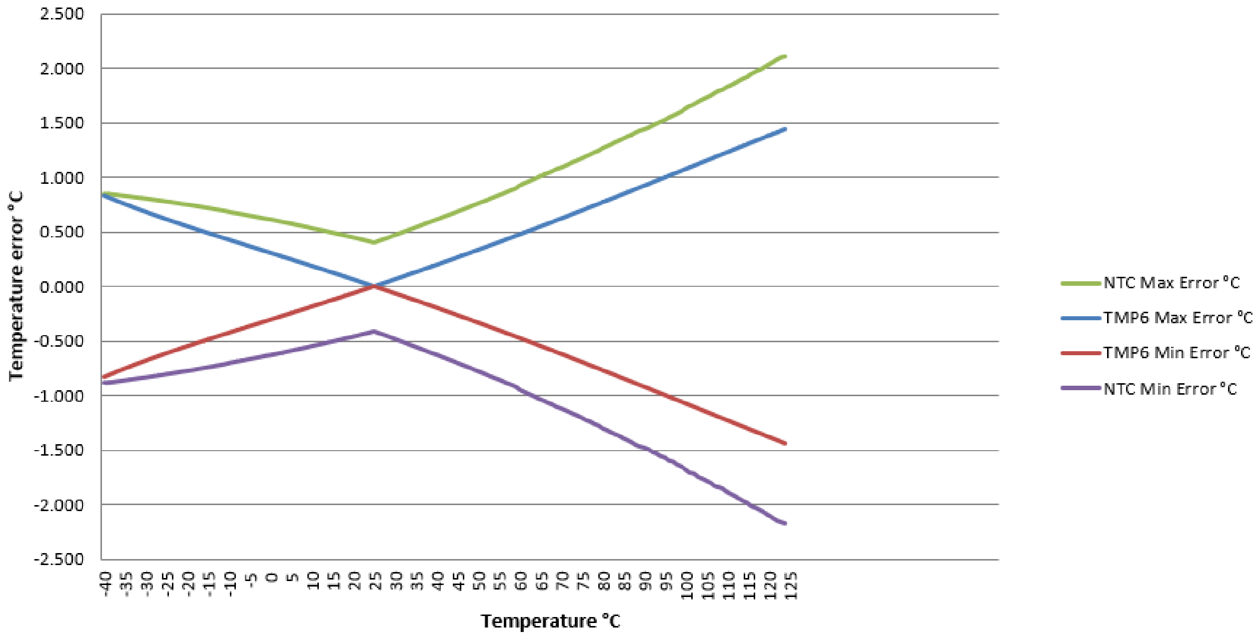

Figure 3-12. Corrected Thermistor Accuracy Comparison

See the *Low-Pass Filter* tab of the Thermistor Design Tool for more information.

3.4 Calibration

Figure 3-13 shows the accuracy across the temperature range of multiple TMP6 thermistor units. As can be seen in the figure, each units accuracy varies, but shows linearity. Thanks to the high linearity of the TMP6x thermistors and their similarity across units, we can remove the tolerance errors from the thermistor, VCC, VREF, ADC LSB and bias resistor and get a very consistent accuracy across the whole temperature range without additional costs. You will need a high accuracy temperature reference. The TMP117 is a high-accuracy, low-power, digital temperature sensor with a NIST traceable accuracy of ±0.3°C (maximum) from –55°C to +150°C. You will need to add some amount of automation to the process after the firmware has been programmed into the UUT (Unit Under Test). After adding in the calibration, the units are now lined up within ±0.3°C across the temperature range (see Figure 3-14).

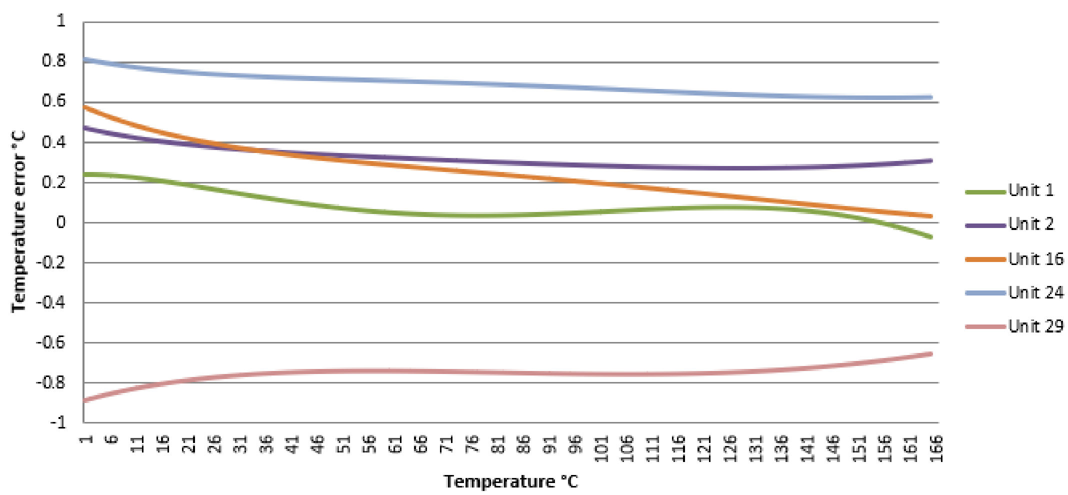

Figure 3-13. TMP6 Thermistor Potential Temperature Error

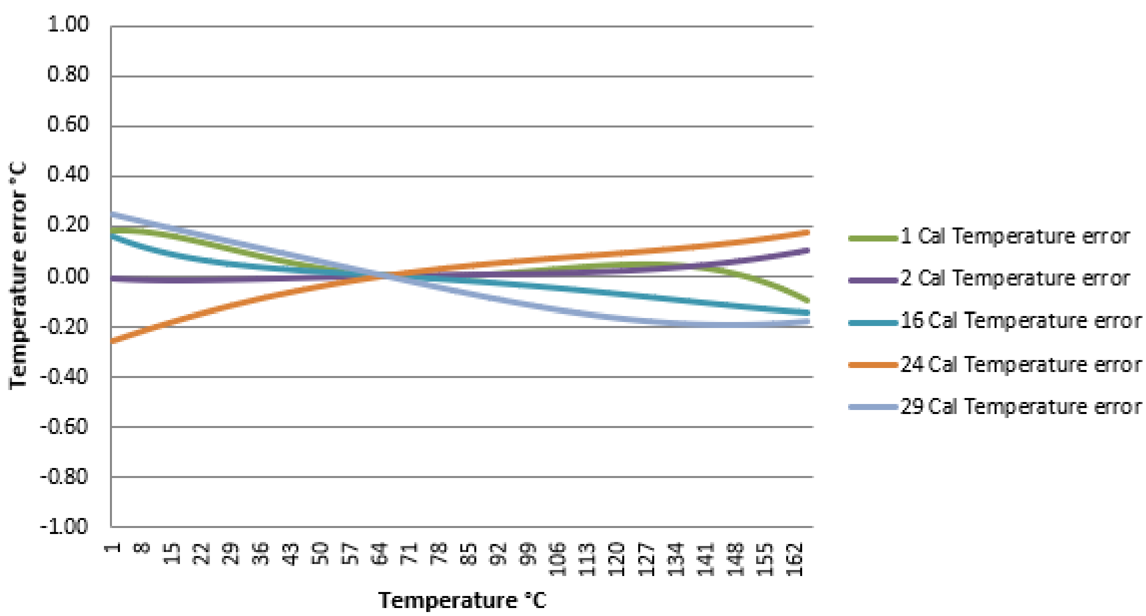

Figure 3-14. TMP6 Thermistor Potential Temperature Error With Correction

Process: The production programming device will program the firmware into the UUT. Once the UUT powers up for the first time, the UUT will use its ADC to measure the VSensevoltage from the TMP6x thermistor on the PCB, calculate the temperature, and write the temperature into the temperature register. At this point either the production programming device or the UUT will read the temperature register in the UUT and read the external temperature reference, subtract the measured temperature from the external temperature reference, and write this value into the offset register. For all future temperature measurements the UUT will add the measured TMP6 thermistor temperature value to the offset register value for the final corrected temperature.

Assumptions: The UUT and temperature reference is at ambient temperature. The UUT will be operating at low power during firmware programming. The TMP6x thermistor's temperature will be measured immediately upon power up. The offset will be calculated based on the first temperature measured upon power up and the temperature reference.

After implementing a single point calibration, a comparison between the accuracy of an NTC thermistor and the TMP61 Linear Thermistor is shown in Figure 3-15.

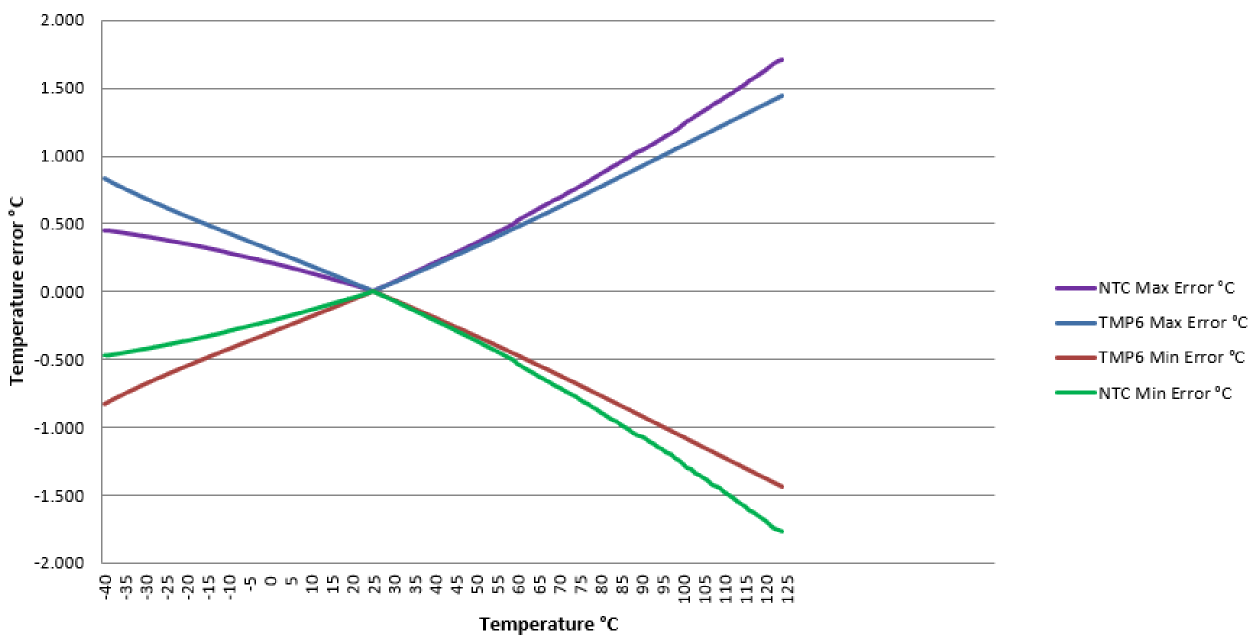

Figure 3-15. Offset Corrected Total Thermistor Temperature Error Compared

4 Design considerations for Full-Scale Range Voltage Output

While the voltage divider circuit with the TMP61 thermistor is simple to convert to, it does not take advantage of the full-scale range of the ADC input. There are two recommended design approaches to increase the full scale range of the thermistor divider circuit. One approach is to use a current source and the other is to design your circuit with the addition of an operation amplifier. The following sections will help to design these circuits.

4.1 Simple Current-Biased

A superior configuration of the TMP61 linear thermistor is to use a current-biased network. Similar to the design steps above, the simplest model for a current-biased TMP61 linear thermistor network is the one shown in Figure 4-1 below. For the ADC, we will use 12-bit resolution and a reference voltage V_{BIAS} of 5 V.

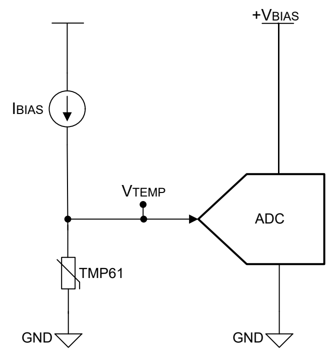

Figure 4-1. TMP6 Linear Thermistor Current Source Circuit

With an ADC reference voltage of 5 V, using a 200-µA current source to bias the TMP61 thermistor is a good option. The resulting Vtemp voltage swing from -40°C to 125°C is 1.3226 V to 3.58 V as shown in the simulation below.

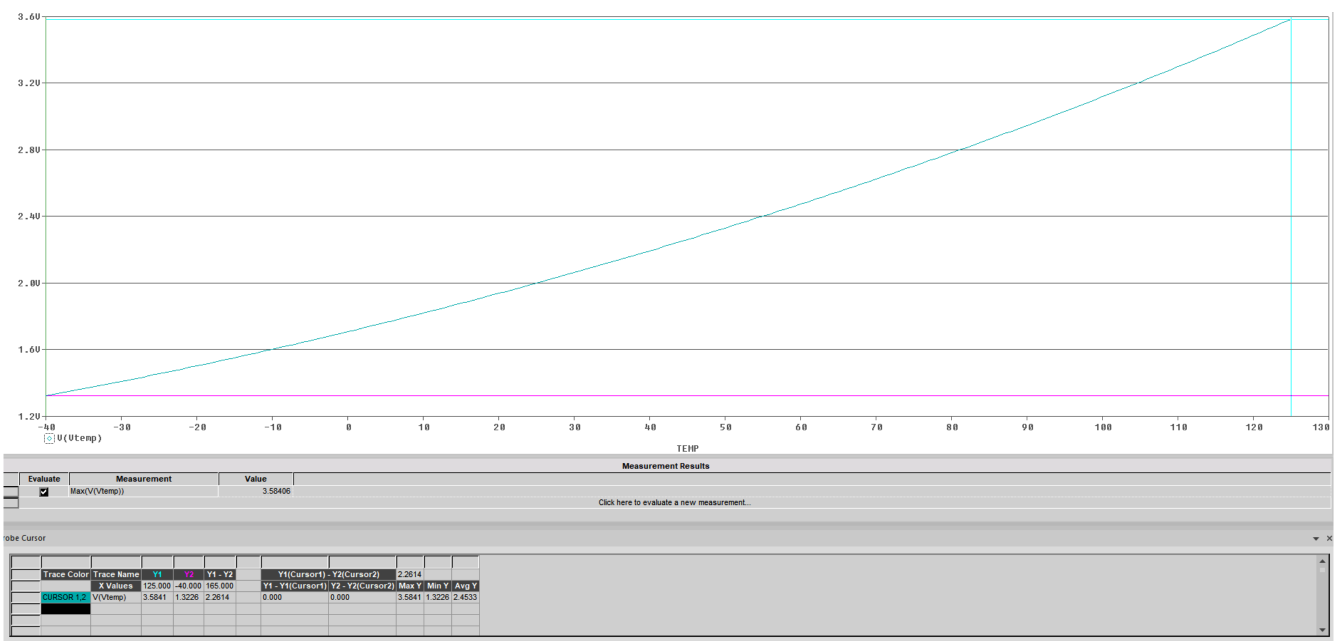

Figure 4-2. TMP61 Thermistor Voltage Swing With 200-µA Current Source

Varying the current when implemented with the TMP61 thermistor will increase the dynamic range of the output response. When biasing the TMP61 thermistor with currents between 50 µA and 400 µA, the output responses can be shown below:

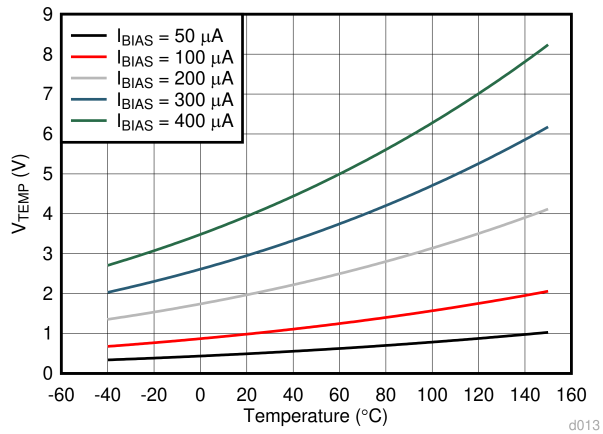

Figure 4-3. TMP61 Thermistor Temperature Voltage With Varying Current Sources

The proper value of the bias current is dependent on the reference voltage of the ADC in your system. You will want to choose the value so that the dynamic range is optimized with the full-scale range of the ADC input. In most cases, 200 µA is recommended. For lower system current draw a TI TMP6 linear thermistor of a higher nominal resistance such as the TMP64 (47 kΩ) and TMP63 (100 kΩ) thermistors can be used. The best current source implementation for the TMP64 (47 kΩ) and TMP63 (100 kΩ) thermistors are 42.533 µA and 20 µA, respectively.

4.2 Active Voltage-Biased

The hardware change for the active thermistor network involves a few more steps. Similar to the simple design above, the first hardware change is to swap the NTC and TMP61 thermistors, while R_{BIAS} can remain as is.

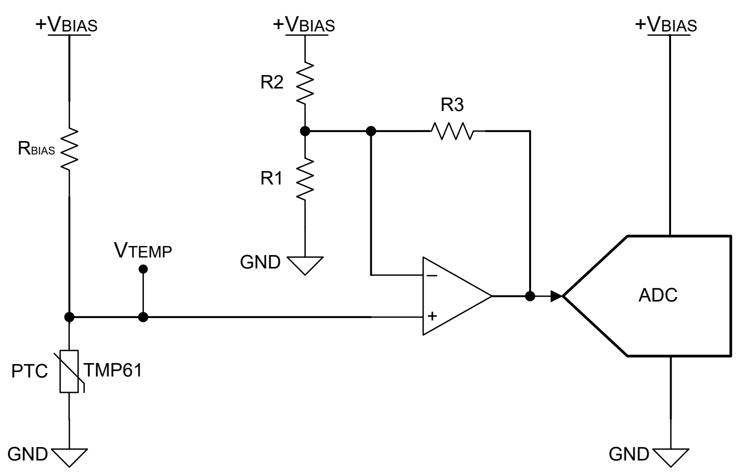

Figure 4-4. TMP6 Linear Thermistor Schematic With Op Amp

Following the design steps of the guide, we end up with resistance values of R_{BIAS} = 10 kΩ, R1 = 6.84 kΩ, R2 = 6.25 kΩ and R3 = 10 kΩ. The resulting V_OUT has range of 0.129 V to 4.86 V, which is within the linear operating range of the op amp and provides better resolution. These design decisions result in the voltage response shown in Figure 4-5 below.

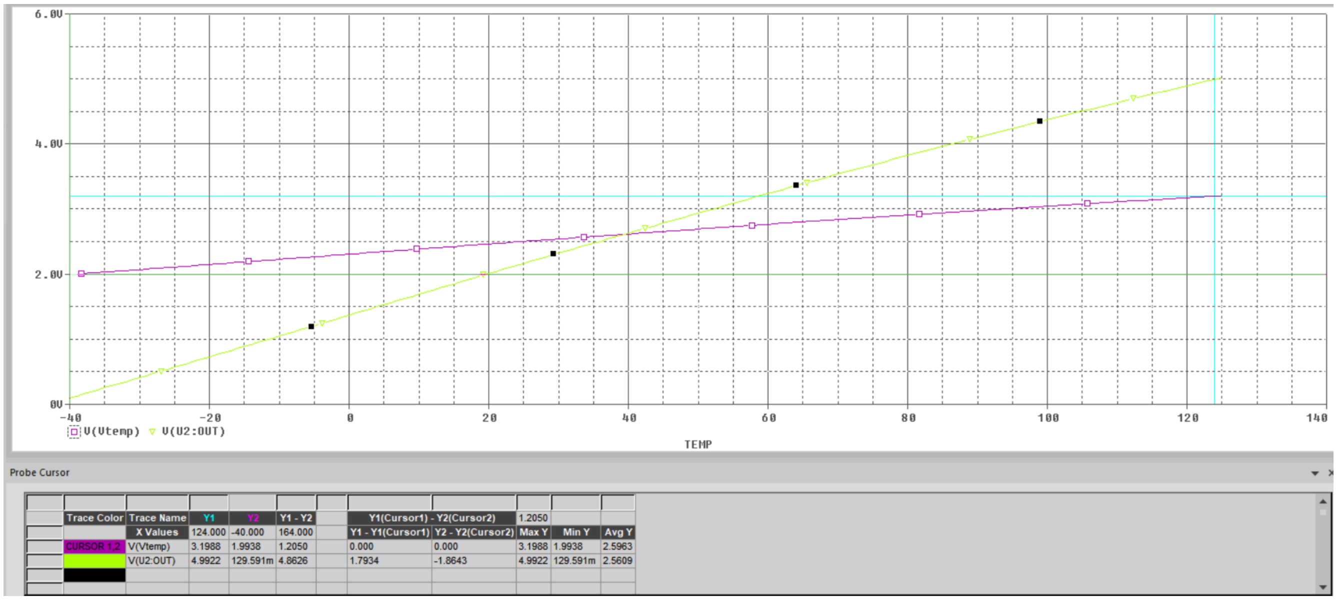

Figure 4-5. TMP6 Linear Thermistor With Op Amp Voltage Response

5 Conclusion

In conclusion, when comparing an NTC thermistor and a linear thermistor, an NTC thermistor may seem to have a benefit of resolution around room temperature. However, when we take a deeper look we can see that there are many added benefits to using a linear thermistor, such as TI's TMP61 linear thermistor family, rather than an NTC thermistor. Switching out the component of an NTC thermistor and TI's TMP61 linear thermistor can be done due to the parts being a pin to pin replacement. When considerations of oversampling, low pass filtering, calibration are also put in to place a much higher accuracy over the entire temperature range can be achieved with TI's TMP6 family of thermistors.

6 Additional Resources/Considerations

6.1 Constant-Current Source Design

Using the TI thermistor design tool, we can design a constant-current source appropriate for temperature sensing with thermistors.

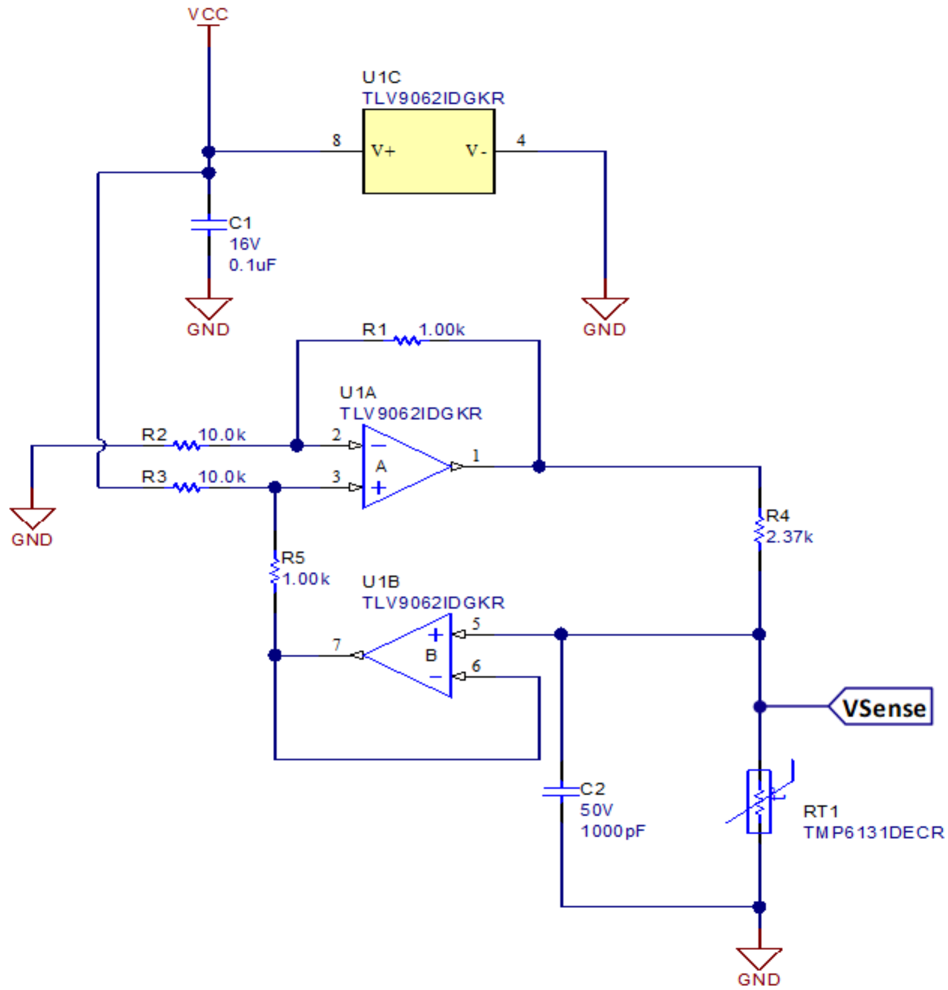

Figure 6-1. Constant Current Source Design

The TLV9062 is a dual-channel operation amplifier with rail-to-rail input and output-swing capabilities.

Spice models for the TMP6131 thermistor can be found on the product folder here.

6.2 TMP6 Thermistor Standard Component Footprints

The TMP6 thermistors are currently available in a X1SON (DEC), SOT-5X3 (DYA) and TO-92S (LPG) packages. The DYA package sits on an IPC-782A 0603 footprint with no issues to fit or solder quality. TI recommends to reduce the pads if possible to minimize excess copper to achieve the maximum sensitivity of the part. However it's not necessary to change the foot print at all. The size of the footprint has no bearing on the accuracy of the part. Also note that the IPC recommends increasing the land pad size to increase robustness in wave soldering. This is good for this device when using a standard 0603 pad size.

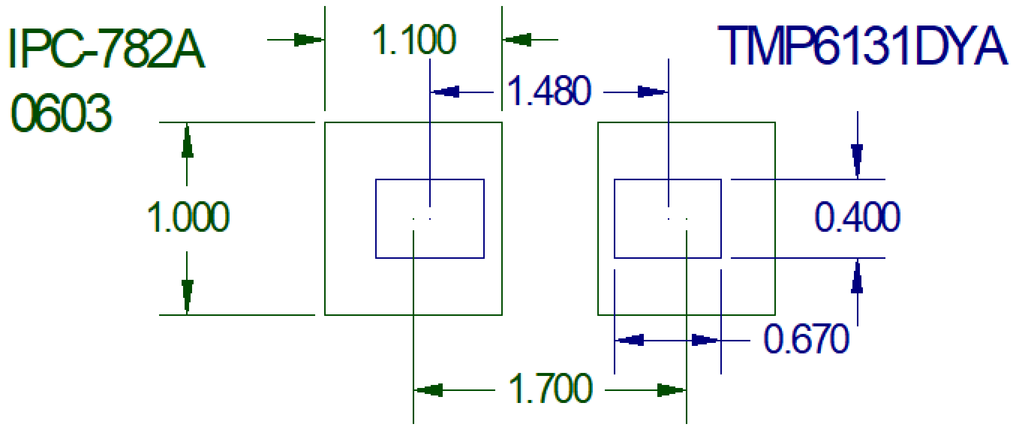

Figure 6-2. TMP6 DYA Package on IPC-782A 0603 Footprint

While the data sheet highlights the SOT-5X3/DYA package being 0603/1608 footprint-compatible, it is also compatible with the 0805/2012 footprint due to its unique lead frame dimensions.

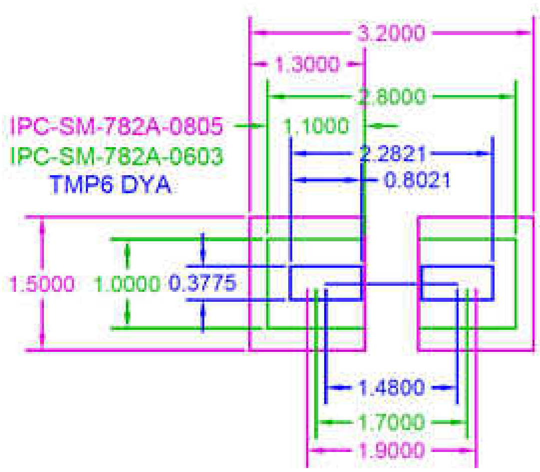

Figure 6-3. TMP6 Thermistor DYA on IPC-782A-0805 Footprint

When the DYA, IPC 0603 and IPC 0805 PCB footprints are overlaid, as shown in image below, it can be seen that the recommended PCB foot print for the DYA has a slightly larger space recommendation than the 0603/1608 or the 0805/2012. The important part to observe in this comparison is to ensure the heel of the TMP6 thermistor pins, as landed on the 0805/2012 pads will allow for the required heel fillet according to the IPC-A-610G standards (for solder quality). The toe and side fillets will easily meet the same IPC-A-610G requirements with the larger pad size that the 0805/2012 footprint presents as compared to either the 0603/1608 or the DYA footprint.

Additional details can be found on the e2e forum in the TMP61 Thermistor FAQ here.

6.3 Dual-Sourcing Approach for TMP6 and NTC Thermistors

A common system requirement is the ability to multi-source components on a BOM. This section provides a method for multi-sourcing with the TMP6 and an NTC thermistors. The key adjective here is to determine which device is onboard based on its initial change in voltage (ΔV) to use the correct temperature conversion code from then on.

The first step is to pre-determine the direction of the initial change in temperature (ΔT) during start-up. During assembly, the board can heat up by about 5°C on its own due to self heating of power supplies, the processor, etc. If you'd like, you can induce a greater change in temperature by using a heat lamp (+ΔT) or freeze spray (-ΔT).

The second step is to have your software determine if the initial change in voltage (ΔV) during start-up is positive (+) or negative (-). Using the table below as a reference, your software can determine which thermistor type is onboard and use the correct temperature conversion code.

	Increase in temperature (+ΔT)	Decrease in temperature (−ΔT)
Change in voltage (ΔV) for TMP6	+	−
Change in voltage (ΔV) for NTC	−	+

Figure 6-4. Dual Sourcing for TMP6 and NTC Thermistors