Application Report

SBAA134 – June 2005

Thermocouple Measurements with ΔΣ ADCs

Russell Anderson, Thomas Kugelstadt

Data Acquisition Products

ABSTRACT

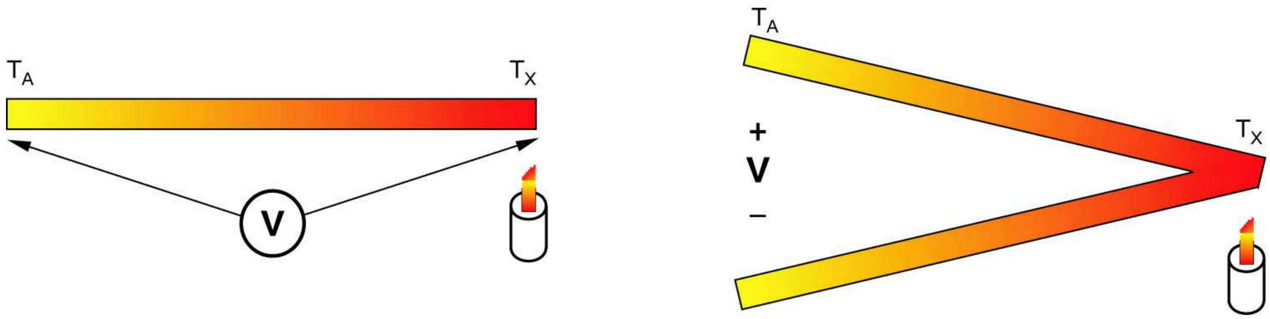
Thermocouples are widely used in a range of industrial and commercial settings, and the temperature/voltage response for thermocouples is fully characterized. The small voltage output is well-suited to the high resolution capability provided by ΔΣ (Delta-Sigma) analog-to-digital converters (ADCs) with a programmable gain amplifier (PGA). The specified NIST polynomials for calculating output voltages are not necessary for most embedded applications. This application note discusses how to use thermocouples and provides a PC program that generates C-code source that can be linked to embedded applications for converting the measured voltage into a temperature reading. A complete system will be demonstrated with the MSC12xx. Unless otherwise specified, all references to the MSC12xx apply to the MSC1200/01/02 and MSC1210/11/12/13 families of microsystem controllers.

Contents

List of Figures

1 Seebeck Voltage

In 1820, Thomas Johann Seebeck discovered that a heated metal bar would create a voltage across the length of the bar, as shown in Figure 1a. This voltage is therefore known as the *Seebeck voltage*, and it differs for various combinations of metals or metal alloys. It can be seen that if you measure two bars (or wires) of the same material with the same temperature differential, the wires would have the same voltage created; thus, there would be 0V between the open ends at T_A. If different metals are used, however, there is a voltage created between the wires that is proportional to the temperature differences (see Figure 1b).



a) Heat creates voltage

b) Voltage proportional to temperature

Figure 1. Seebeck Voltage

Thermocouples are widely used for temperature measurement in a range of settings, even though these devices are not always well understood. A thermocouple is made by connecting together two wires made of different materials. By choosing materials that have different Seebeck voltages, a measureable voltage will be created that depends on the temperature difference between T_X and T_{REF} as shown in Figure 2.

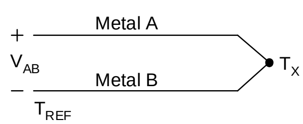


Figure 2. Thermocouple Voltage

All dissimilar metals exhibit this effect, but there are several specific combinations that are generally used in thermocouples. Table 1 lists some of the common thermocouple types.

Table 1. Common Thermocouple Types

THERMOCOUPLE TYPE	+ LEAD METAL A	– LEAD METAL B	TEMPERATURE RANGE (°C)	SEEBECK COEFFICIENT (μV/°C)
J	Iron	Constantan	–210 to 1200	50.37 at 0°C
K	Chromel	Alumel	–270 to 1370	39.48 at 0°C
T	Copper	Constantan	–200 to 400	38.74 at 0°C
E	Chromel	Constantan	–270 to 1000	58.70 at 0°C
S	P_T and 10% R_H	P_T	–50 to 1768	10.19 at 0°C

2 Converting Thermocouple Voltages to Temperature

One primary advantage of thermocouples is that they are simple to build; additionally, they work well over a wide range of temperatures. Thermocouple responses, on the other hand, have several problems. The greatest drawback is that these responses are non-linear. NIST (National Institute of Standards and Technology) has analyzed the output voltage versus temperature for the various types of thermocouples; several polynomial equations are defined that correlate the temperature and voltage output. This data can be found on the NIST web site at http://srdata.nist. gov/its90/main/.

Table 2 summarizes the polynomial orders and the respective temperature ranges for the types of thermocouples discussed in Table 1. Figure 3 illustrates the typical responses for these same thermocouple types.

Table 2. NIST Polynomials

THERMOCOUPLE TYPE	TEMPERATURE RANGE (°C) FOR POLYNOMIALS	POLYNOMIAL ORDER
J	–-210 to 760, 760 to 1200	8th, 5th
K	–270 to 0, 0 to 1370	10th, 9th, + a $e^{b(t - c)^2}$
T	–200 to 0, 0 to 400	7th, 6th
E	–270 to 0, 0 to 1000	13th, 10th
S	–50 to 1064.18, 1064.18 to 1664.5, 1664.5 to 1768	8th, 4th, 4th

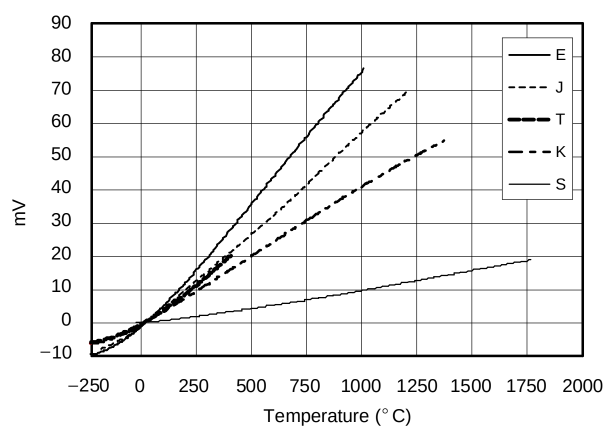


Figure 3. Thermocouple Responses

The mathematical operations required to calculate a 13th-order equation with no loss of precision can take a significant amount of computational processing with high resolution, floating-point numbers. This type of computation is not something that is very suitable for embedded processing or microprocessors.

3 Simplifying Temperature Measurement

There are usually ways to simplify the computations without a significant loss of precision. For small changes in temperature, a linear approximation provides a good correlation between voltage and temperature. For wider temperature ranges, an alternate method is to divide the entire range into several smaller ranges, and use a different linear approximation for each of the sub-ranges.

This application report provides a program that allows the user to select the number of segments and then generate the ***C***-code with the appropriate routines to implement the results.

4 Unwanted Thermocouples

Another of the significant problems when measuring a thermocouple voltage is that each connection creates a new thermocouple. This process is seen as the signal goes to the ADC integrated circuit. Each step along the path can encounter several additional thermocouples as one proceeds from wire connector, to solder, to copper trace, to IC pin, to bond wire, to chip contact. However, if the signal is differential, and each of the thermocouple pairs are at the same temperature, then the thermocouple voltages will cancel and have no net effect on the measurement. For high-precision applications, the user must be sure that these assumptions are correct. Measuring with differential inputs may have some thermocouple voltages that do not cancel if the thermocouples are not located close together on the device pins, or if there is a thermal gradient on the chip.

5 Converting Thermocouple Voltage to Temperature

As discussed earlier, the thermocouple generates a voltage that is related to the temperature difference between the two ends of the wires T_X and T_{REF} (see Figure 2). If the temperature at T_{REF} is known, then the tempature at T_X can be computed. The process of accounting for T_{REF} is called *cold junction compensation* since it is generally assumed that T_{REF} is the cold temperature.

One method of cold junction compensation is to provide a way to make isothermal connections at T_{REF}, then measure that temperature with a device such as a thermistor. Once the temperature of the thermistor (and T_{REF}) is known, the thermocouple voltage for that temperature (relative to 0°C) can be determined and added to the measured voltage on the thermocouple leads. This compensation gives the voltage that would have been developed if T_{REF} had been at 0°C. Note that this voltage is needed when referencing the NIST charts, since the chart values are specified relative to 0°C.

There might be a tendency to presume that the thermistor temperature can simply be added to the temperature that would be computed from the thermocouple voltage. As shown in Figure 3 and Figure 4, however, the thermocouple voltage is non-linear. Therefore, it can be seen that the same voltage differential would not yield the same temperature differential at different values of T_{REF}. The only way to compensate for those nonlinearities is to convert the T_{REF} temperature to its equivalent voltage, then compute the temperature for that combined voltage.

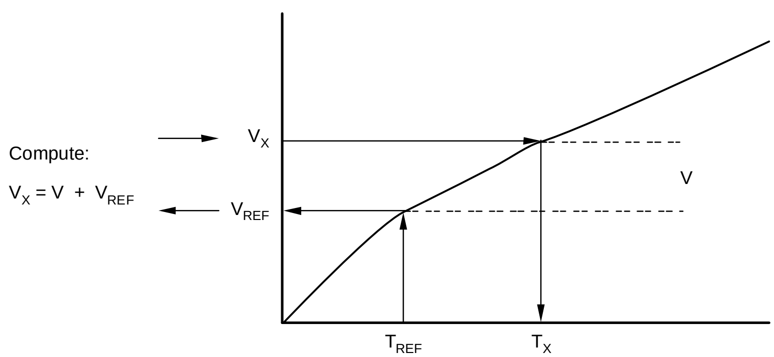


Figure 4. Temperature Algorithm

Figure 5 shows a simple circuit using an MSC1201. The four thermocouples are measured single-ended against the common ground potential at the AINCOM terminal.

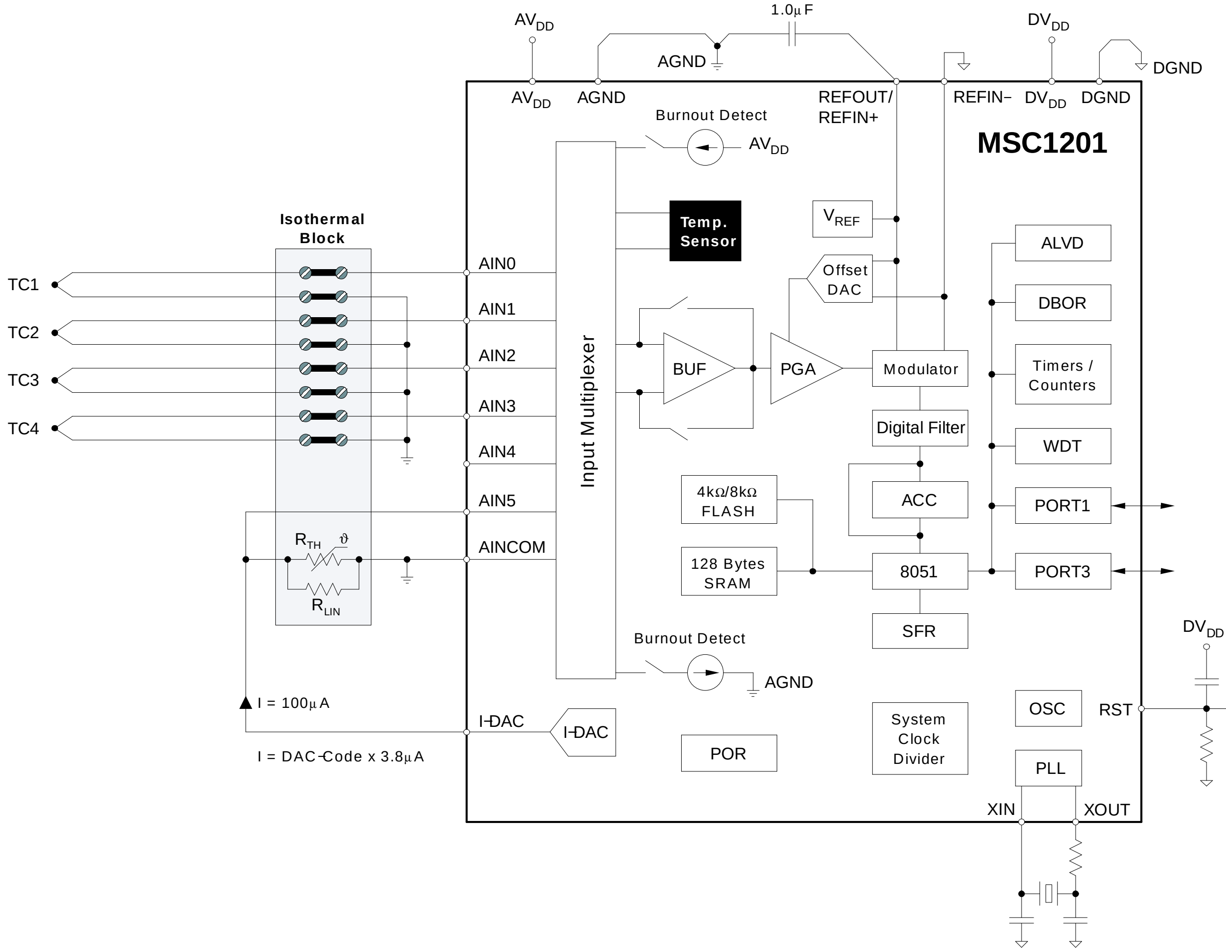


Figure 5. Simple Input Circuitry for Noise-Free Thermocouple Wires

The circuit design assumes a low-impedance ground at the isothermal block, short connecting wires to the thermocouples (without noise pickup), a bypassed input buffer, and a sufficiently high impedance at maximum gain.

The reference temperature can be measured via RTD or a precision thermistor, with the excitation current being provided by the on-chip IDAC. All measurements are performed against the common ground potential at AINCOM. Since this is a single-ended measurement, any temperature difference between the common ground and the input channel on the IC creates a thermocouple that cannot be compensated.

The thermocouple voltage causes current to flow in the thermocouple wires. Any resistance in those wires can cause an error in the measured voltage. In addtion, any other induced currents in the thermocouple wires could also cause an error in the measured voltage.

6 Differential Measurements with Filtering

The circuit in Figure 6 shows the addition of filters that could be necessary if long connecting wires are passing noise from factory equipment. Each thermocouple is measured differentially. This application suits the MSC12xx with its nine analog inputs. Measuring four differential thermocouples, however, means that the thermistor has to be measured relative to the REF voltage through the resistors to the analog inputs. That measurement would determine the current into the thermistor so that the thermistor resistance could be computed and therefore determine the temperature.

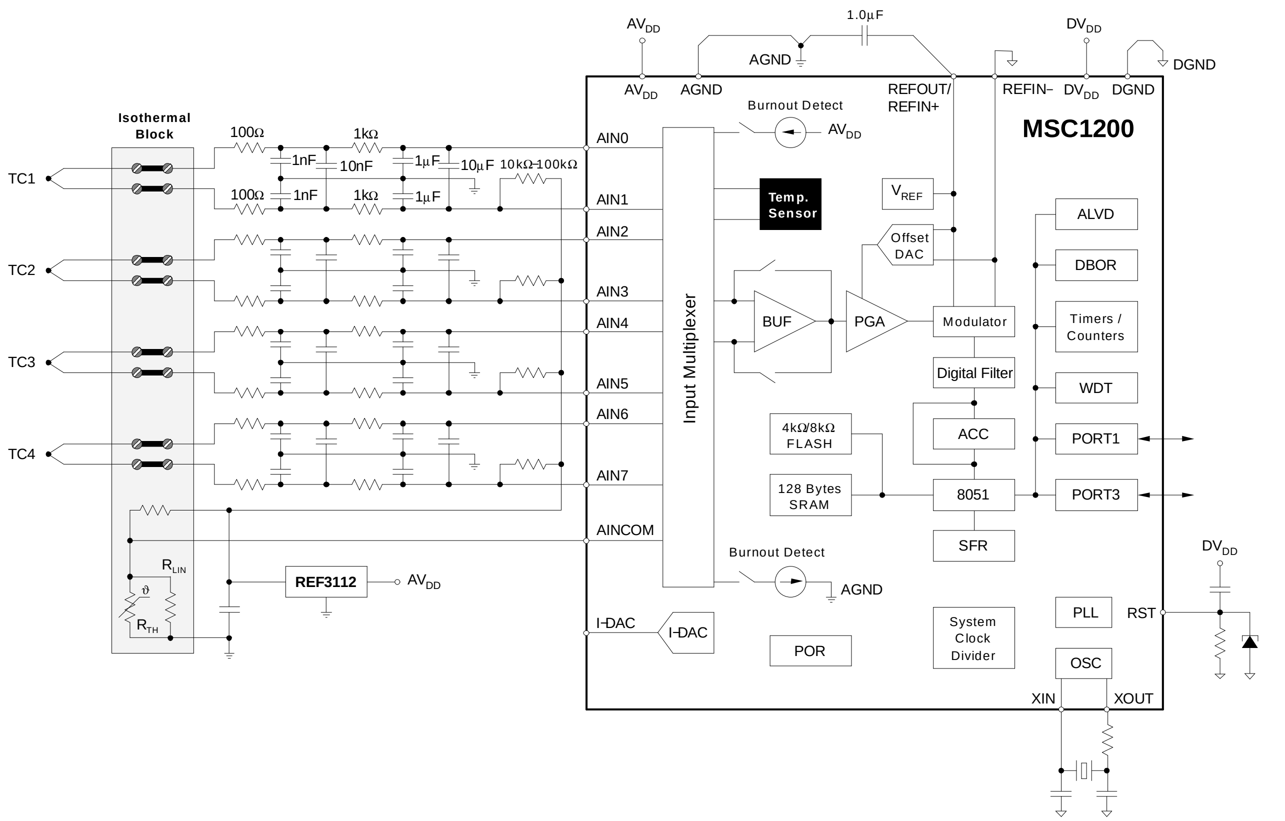


Figure 6. Filter Effort for Long Thermocouple Wires Passing Noisy Signals

Two RC filters remove common-mode and differential noise at both low and high frequencies. The input buffer is switched on for high input impedance. The precision voltage reference REF 3112 applies a common-mode voltage of 1.25V to each channel via the 10k – 100k bias resistors, thus shifting the differential input signals into the analog input voltage range of the buffer. When using this type of external R-C filter, care must be taken to protect the circuit from introducing an offset voltage, as discussed in SBAA111, *Understanding the ADC Input on the MSC12xx*.

The reference temperature is measured via RTD or thermistor, connected to the AINCOM input. As noted earlier, measuring four differential thermocouples requires that the thermistor be measured relative to either AIN1, AIN3, AIN5 or AIN7. That measurement would determine the current into the thermistor so that the thermistor resistance could be computed and the temperature determined.

7 Integrated Temperature Sensor

The ADS1216, 1217, and 1218 family of ADCs and the MSC121x / MSC120x systems have on-chip, diode temperature sensors. With a temperature coefficient of 345µV/°C or 375µV/°C, the sensor output is 115mV at 25°C. Applications with measurement electronics placed close to the isothermal block can further reduce the external component count by using the on-chip sensor for reference temperature measurement. Care must be observed, though, because on-chip heating could shift the on-chip sensor from the temperature of the isothermal block.

8 Related Application Notes

Table 3 lists additional application notes available from Texas Instruments. These important resources can be downloaded from www.ti.com by searching for the respective literature number or clicking the active hyperlink for each document.

Table 3. Related Application Notes

TITLE	LITERATURE NUMBER
Input Currents for High-Resolution ADCs	SBAA090
Understanding the ADC Input on the MSC12xx	SBAA111
The Offset DAC	SBAA077
Using The Delta-Sigma ADC on the MSC12xx	SBAA101A
Understanding Ratiometric Conversions	SBAA110
Measuring Temperature with the ADS1216, ADS1217, or ADS1218	SBAA073A
ADC Gain Calibration – Extending the ADC Input Range in MSC12xx Devices	SBAA107
ADC Offset in MSC12xx Devices	SBAA097B
Using the MSC121x as a High-Precision Intelligent Temperature Sensor	SBAA100
Incorporating the MSC1210 into Electronic Weight Scale Systems	SBAA092A

Appendix A Generation of Thermocouple Routines

The Tempgen program (Figure A-1, available for download with this application note) creates the ***C*** routines that can be added to a thermocouple project for five different thermocouple types (E, J, K, S, and T). The program starts by asking the user to enter the type of thermocouple; low and high temperature limits; and the number of sections. If a value of '1' is entered for the number of sections, the program simply uses a linear interpolation. Up to 255 linear sections can be selected. If more than '1' is selected, the program breaks up the temperature range into sub-sections, each with its own linear interpolation. The program generates the ***C*** code and computes the error from the NIST calibration coefficients. The ***C*** code is stored in a file with the name **ThermL_x.c** for the single linear interpolation, where *x* is the thermocouple type. For code generated with multiple linear sections, the file name will be **ThermP_x.c**.

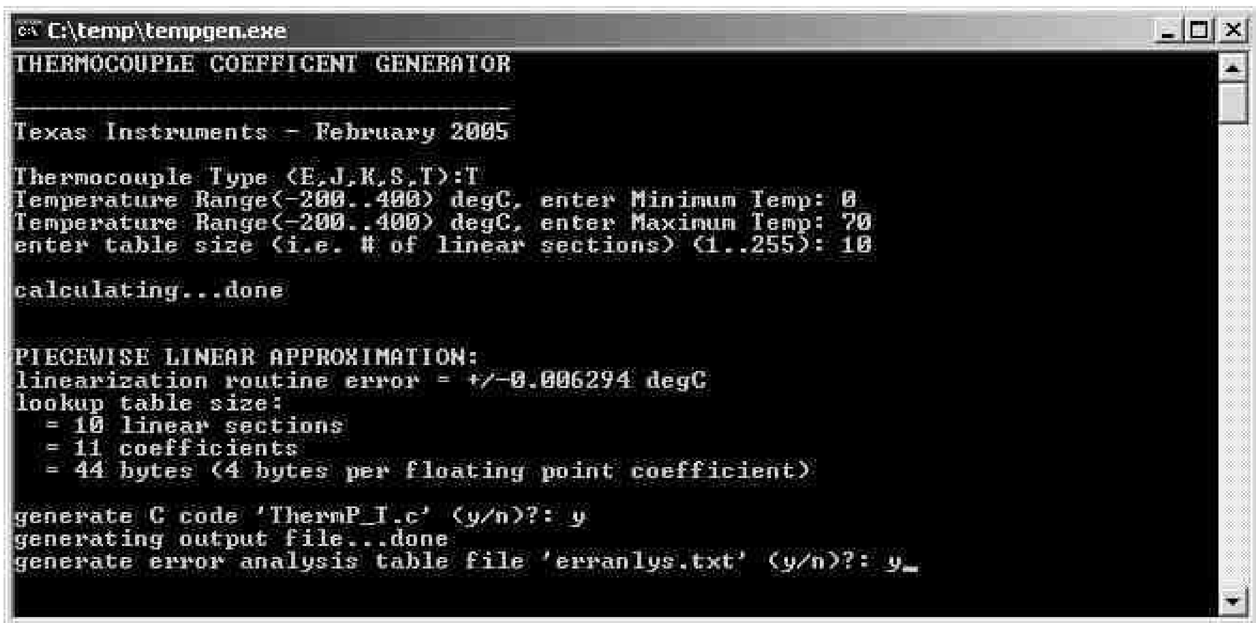


Figure A-1. Thermocouple Coefficient Generator Software

The ***C*** code includes the following routines:

```
Ttype ()     // The Thermocouple type
MinTemp ()        // Returns the minimum temperature selected for these routines
MaxTemp ()        // Returns the maximum temperature selected for these routines
T_volt (float mV) // The temperature difference for this mV and thermocouple type
V_temp (float t)  // The voltage difference (mV) for this temperature difference
```

Appendix B MSC Thermocouple Measurement Program

A complete thermocouple program can be easily generated by using the routines described in this appendix. This example program demonstrates measurement of the thermistor resistance to determine the thermistor temperature. This circuit (shown in Figure B-1) does not use a linearization circuit for the thermistor, as shown in Figure 5 and Figure 6. Rather, it simply uses a general-purpose equation to convert the resistance into a temperature. That temperature is then used to calculate the voltage for a type-T thermocouple at that same temperature. This procedure calculates the voltage from 0°C to T_{REF}. The voltage is then added to the voltage measured from the thermocouple. The total voltage is then used to calculate the temperature at the end of the thermocouple.

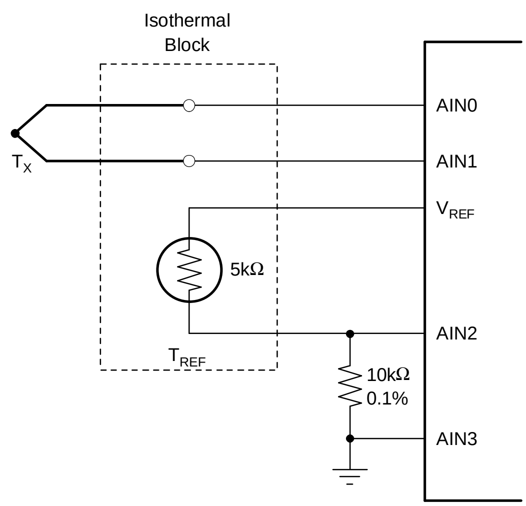


Figure B-1. Test Circuit

B.1 Thermocouple Measurement Program Routines

Table B-1 lists the measurement program routines.

Table B-1. Thermocouple Measurement Program Routines

ROUTINE	DESCRIPTION
`autobaud()`	Accepts carriage routine to set communications baud rate
`bipolar()`	Reads the 24-bit ADC value and returns a 32-bit integer result
`Ttype()`	Returns the thermocouple type for the thermocouple routines
`MinTemp()`	The minimum temperature for the thermocouple conversion routines
`MaxTemp()`	The maximum temperature for the thermocouple conversion routines
`V_Temp()`	Returns the thermocouple voltage for the Celsius (°C) temperature
`T_Volt()`	Returns the temperature for the thermocouple voltage

B.2 Main Program

```
//*********************************************************************
// File name: Therm.c
//
// Copyright 2005 Texas Instruments Inc as an  unpublished work.
//
// Version 1.0
//
// Compiler Version (Keil V2.38), (Raisonance V6.10.13)
//
// Module Description:
//  Thermocouple Measurement example, using a thermistor to measure the cold junction.
//  Designed to measure a 5K ohm Thermistor connected from Vref to ground through a
10K resistor
//  Vref -- Thermistor -- (AIN+) -- 10K --  (AINCOM)AGND
//  A voltage equivalent to the cold reference junction is added to the measured
thermocouple
//  voltage. This total voltage is then used to compute the thermocouple temperature.
//
//*********************************************************************
#include "legal.c"          //Texas Instruments, Inc. copyright and liability
#include <REG1210.H>        // The header file with the MSC register definitions
#include <stdio.h>          // Standard I/O so we can use the printf function
#include <math.h>
extern signed long bipolar(void);
extern void autobaud(void);
extern char Ttype(void);
extern float MinTemp(void);
extern float MaxTemp(void);
extern float V_temp(float);
extern float T_volt (float);
#define LSB 298e-9          // LSB in V
#define mVLSB 298e-6        // LSB in mV
#ifndef XTAL                // if no XTAL compiler variable defined use:
   #define XTAL 11059200    // XTAL frequency 11.0592 MHz
#endif
void main(void) {
   char i, j, samples;
   float result, volts, temp, resistance, ratio, lr, ave, Vref, Vx, Tx;

   PDCON = 0x75;                  // Turn on the A/D
   ACLK = XTAL/1000000;           // ACLK freq. = XTAL Freq./(ACLK +1) = 0.9216 MHz
                                  // 0.9216 Mhz/64 = 14,400 Hz
   DECIMATION = 1440;             // Data Rate = 14,400/1,440 = 10 Hz
   ADMUX = 0x23;                  // AINP = AIN2, AINN = AIN3
   ADCON0 = 0x30;                 // Vref On, 2.5V, Buffer Off, PGA=1
   ADCON1 = 0x01;                 // bipolar, auto, self calibration, offset, gain
    CKCON = 0x10;                    // MSC1200 Timer1 div 4
    TCON = 0;                        // MSC1200 Stop TR1
   autobaud();
   printf("Thermister Test using a Type %c Thermocouple\n",Ttype());
   printf("Measuring from %5.2f to %5.2f degrees C\n", MinTemp(), MaxTemp() );
   printf("Min Temp Voltage = %8.6fmV, Max Temp Voltage =
%8.6fmV\n",V_temp(MinTemp()), V_temp(MaxTemp()) );
    samples = 10;
    while(1)
    {
      // SET MUX FOR THERMISTOR AND AVERAGE samples number of MEASUREMENTs
      ADMUX = 0x78;               // AINP = AIN7, AINN = AINCOM, Thermistor
      // wait for the calibration to take place
```

```
      for (i=0;i<4;i++){         // dump 3+1 conversions
         while(!(AIE&0x20)) {}
         j=ADRESL;
      }
      ave = 0.0;
      for (i = 0; i < samples; i++)
      {
         // Wait for new next result
         while (!(AIE & 0x20));
         ave += bipolar() * LSB; // This read clears ADCIRQ
      }
      volts = ave/samples;       // Thermistor Voltage
      resistance = 25/volts - 10;
      ratio = resistance/5.0;
      lr = log(ratio);
      temp = 1/(3.3540180e-3 + 2.5415585e-4*lr + 3.7354242e-6*lr*lr - 7.8037673e-
8*lr*lr*lr)-273.0;

      Vref = V_temp(temp);
      printf("Tmistr T=%5.2fC, V Tmocoupl=%6.3fmV, ",temp, Vref );
      // SET MUX FOR THERMOCOUPLE AND AVERAGE samples number of MEASUREMENTs
      ADMUX = 0x67;              // AINP = AIN6, AINN = AIN7, Thermocouple
      for (i=0;i<4;i++){         // dump 3+1 conversions
         while(!(AIE&0x20)) {}
         j=ADRESL;
      }
      ave = 0;
      for (i = 0; i < samples; i++)
      {
         // Wait for new next result
         while (!(AIE & 0x20));
         ave += bipolar() * mVLSB; // This read clears ADC Interrupt flag
      }
      volts = ave/samples;       // Thermocouple Voltage
      Vx = Vref + volts;         // Add mV to mV
      Tx = T_volt(Vx);
      printf ("Thermocouple Voltage=%5.2fmV,  Temp=%6.3f\n", volts, Tx);
   }// while
}
```

File: Themp_T.c Thermocouple Routines generated by PC program

```
/******************************************************************************
 * File : ThermP_T.c
 *
 * Source       : Automatically generated using 'TempGen.exe'
 *
 * Compiler     : Intended for most C compilers
 * Description  : Subroutines for thermocouple linearization
 *                (for type T thermocouples)
 *                using piecewise linear approximation method.
 * More Info    : Details in application note available at....
 *                http://www.ti.com/msc
 ******************************************************************************/
/*  Defines section */
#define TMIN (0.000000)  // = minimum temperature in degC
#define TMAX (70.000000)  // = maximum temperature in degC
#define NSEG 10  // = number of sections in table
#define TSEG 7.000000  // = (TMAX-TMIN)/NSEG = temperature span in degC of each segment
/********************* Lookup Table ****************************************
 * lookup table size:
 *   = 10 linear sections
 *   = 11 coefficients
```

```
 *   = 44 bytes (4 bytes per floating point coefficient)  *******************/
const float code C_volt[] = {-
0.000216,0.272698,0.549237,0.829661,1.114142,1.402784,1.695630,1.992682,2.293910,2.599256,2.908646
};
/* linearization routine error = +/-0.006294 degC
 * specified over measurement range 0.000000 degC to 70.000000 degC  */
/*****************************************************************************
 * Function name     : Ttype
 *   Returns         : Thermocouple Type
 *   Arguments       : None
 * Description       : Returns the letter designation for the thermocouple type
 *****************************************************************************/
char Ttype () {
  return ('T');
}
/*****************************************************************************
 * Function name     : MinTemp
 *   Returns         : Minimum Temperature of Temperature Range
 *   Arguments       : None
 * Description       : Returns minimum temperature specified by lookup table.
 *****************************************************************************/
float MinTemp () {
  return (TMIN);
}
/****************************************************************************
 * Function Name     : MaxTemp
 *   Returns         : Maximum Temperature of Temperature Range
 *   Arguments       : None
 * Description       : Returns maximum temperature specified by lookup table.
 ****************************************************************************/
float MaxTemp () {
  return (TMAX);
}
/*****************************************************************************
 * Function Name     : T_volt(mV)
 *   Returns         : Temperature in degC (for type T thermocouples)
 *   Argument        : Voltage in mV
 * Description       : Calculates temperature of thermocouple as a function of
 *                     voltage via a piecewise linear approximation method.
 ******************************************************************************/
  float T_volt (float mV) {
  float t;
  int i, Add;
                               // set up initial values
  i = NSEG/2;                  // starting value for 'i' index
  Add = (i+1)/2;               // Add value used in do loop
  // determine if input v is within range
  if (mV<C_volt[0])            // if input is under-range..
    i=0;                       // ..then use lowest coefficients
  else if (mV>C_volt[NSEG])    // if input is over-range..
    i=NSEG-1;                  // ..then use highest coefficients
  // if input within range, determine which coefficients to use
  else do {
    if (C_volt[i]>mV)   i-=Add; // either decrease i by Add..
    if (C_volt[i+1]<mV) i+=Add; // ..or increase i by Add
    if (i<0)       i=0;        // make sure i is >=0..
    if (i>NSEG-1)  i=NSEG-1;   // ..and <=NSEG-1
    Add = (Add+1)/2;           // divide Add by two (rounded)
  } while ((C_volt[i]>mV)||(C_volt[i+1]<mV));  // repeat 'til done
  // compute final result
  t = TMIN+TSEG*i + (mV-C_volt[i])*TSEG/(C_volt[i+1]-C_volt[i]);
  return (t);
}
/***************************************************************************
 * Function Name    : V_temp(t)
 *   Returns        : Voltage (mV) as a function of temperature
```

```
 *                   : (for type T thermocouples)
 *   Argument        : Temperature of thermocouple in degC
 * Description       : Calculates voltage of thermocouple as a function of
 *                     temperature via a piecewise linear approximation method.
 ***************************************************************************/
float V_temp (float t) {
  float v;
  int i;
  i=(t-TMIN)/TSEG;      // determine which coefficients to use
  if (i<0)              // if input is under-range..
    i=0;                // ..then use lowest coefficients
  else if (i>NSEG-1)    // if input is over-range..
    i=NSEG-1;           // ..then use highest coefficients
  // compute final result
  v = C_volt[i]+(t-(TMIN+TSEG*i))*(C_volt[i+1]-C_volt[i])/TSEG;
//  printf("cvolt[%d]=%6.3f, cvolt[%d]=%6.3f\n",i, C_volt[i], i+1, C_volt[i+1]);
  return (v);
}
```

Appendix C Delta-Sigma Inputs

C.1 ADC Input Stage

Figure C-1 shows an analog-to-digital converter (ADC) input stage, which is identical to all ADCs discussed in this application note:

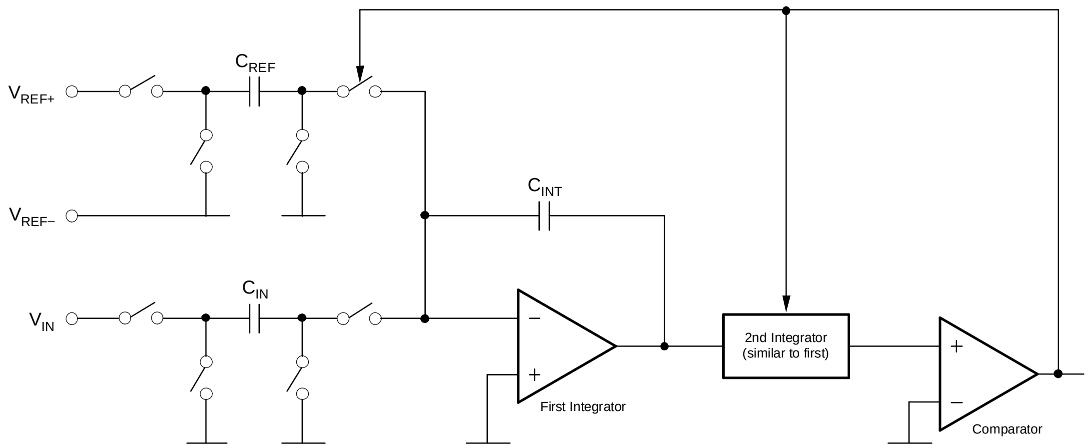


Figure C-1. Principle of Input Voltage Measurement of a ΔΣ Converter

The analog input signal is sampled and then compared with a reference signal. Figure C-2 shows a simplified structure of a differential input.

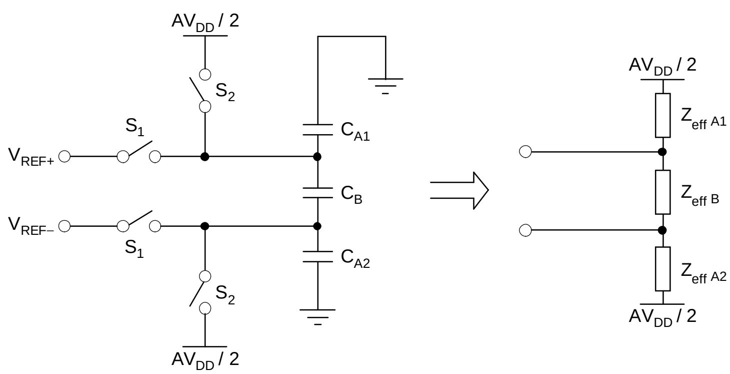


Figure C-2. Switched Capacitors Yield Effective Input Impedances

Here, the input impedance depends on the capacitor value and the sampling frequency derived from Equation C-1:

$$Z_{eff} = \frac{1}{C \cdot f_{SAMP}} \quad \text{(C-1)}$$

When programming the PGA, the values of the input capacitor and the reference capacitor change. Therefore, different gain factors cause different input impedances (see Table C-1).

Table C-1. Input Impedance vs f_{CLK} , C_{IN} and C_{REF}

PGA	FSR Input	Z_{effA} (Ω)	Z_{effB} (Ω)	C_{IN} (pF)	C_{REF} (pF)	f_{SAMP}
1	$\pm V_{REF}$	13M	6.1M	9	12	f_{MOD}
2	$\pm V_{REF}$ / 2	6.5M	3.1M	18	12	f_{MOD}
4	$\pm V_{REF}$ / 4	3.3M	1.5M	36	12	f_{MOD}
8	$\pm V_{REF}$ / 8	1.6M	760k	36	6	f_{MOD} x 2
16	$\pm V_{REF}$ / 16	820k	380k	36	3	f_{MOD} x 4
32	$\pm V_{REF}$ / 32	410k	190k	36	1.5	f_{MOD} x 8
64	$\pm V_{REF}$ / 64	200k	90k	36	0.75	f_{MOD} x 16
128	$\pm V_{REF}$ / 128	200k	90k	36	0.375	f_{MOD} x 16

For the previous discussion, the input buffer was considered to be switched off (bypassed). However, for high-gain applications that require high input impedances, it is recommended to switch the input buffer on. As shown in Figure C-3, the active buffer uses a chopper stage that removes DC-offset and 1/f-noise.

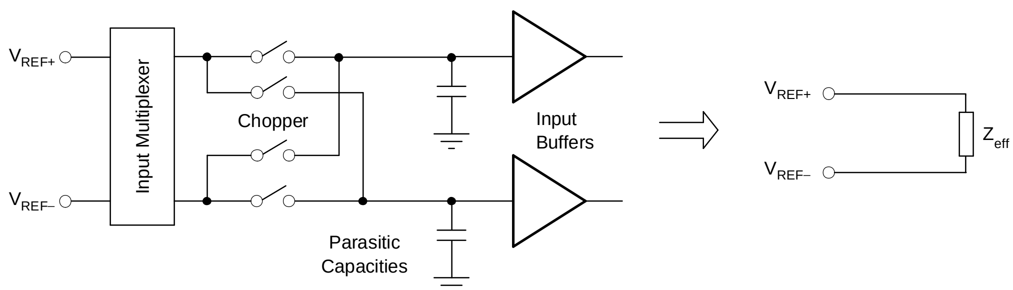


Figure C-3. Input Impedance of Active Buffer Depends on Parasitic Capacitance Only

Parasitic capacitance at the buffer input determines the input impedance via the oscillator frequency, f_{OSC}. Table C-2 shows Z_{eff} decreasing with increasing f_{SAMP}.

Table C-2. Z_{eff} as a Function of f_{SAMP}

f_{SAMP} (MHz)	Z_{eff} (GΩ)
1	12.0
2	6.0
2.45	4.9
4	3.0
4.91	2.4
8	1.5

At a gain of PGA = 128, the minimum impedance is still 1.5GΩ. When measuring in *Buffer-On* mode, the current consumption increases by 0.5nA, while the effective number of bits (ENOB) drops by approximately 0.8bits-rms, as a result of slightly increased noise (see Figure C-4).

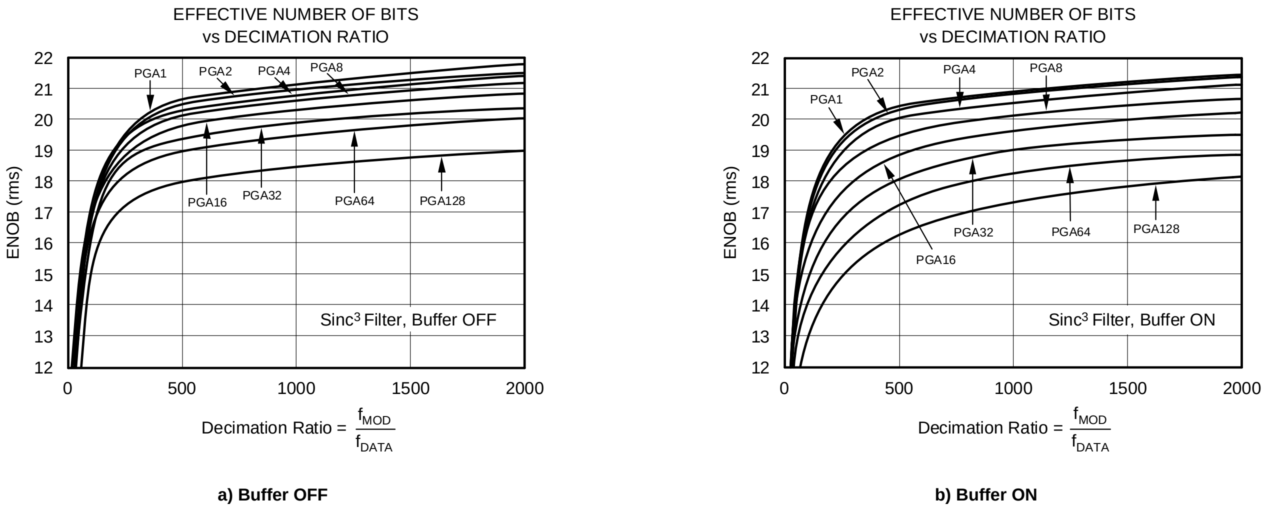


Figure C-4. ENOB Comparison: Buffer Off and Buffer On

The input voltage range also changes, from:

$$\text{AINMIN} = \text{AGND} - 0.1\text{V } \mathit{TO} \text{ AINMAX} = \text{AVDD} + 0.1\text{V} \quad \text{(C-2)}$$

without the buffer, to:

$$\text{AINMIN} = \text{AGND} + 50\text{mV } \mathit{TO} \text{ AINMAX} = \text{AVDD} - 1.5\text{V} \quad \text{(C-3)}$$

with the buffer.

It will also be noticed that for the *Buffer-Off* case, switching from a gain of 64 to 128 does not give an improvement in signal/noise. In that case, the noise is increased by one bit or a power of two, which is the same improvement in the signal.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265