Implementing a Direct Thermocouple Interface With the MSP430F4xx and ADS1240

MSP430 Applications

ABSTRACT

This application report describes how to implement a direct thermocouple interface without using the signal conditioning circuitry normally required for thermocouples. The thermocouple interfaces directly to the ADS1240 24-bit analog-to-digital converter (ADC). An MSP430F413 ultra-low-power microcontroller (MCU) is used to communicate with the ADC, read the conversion values, convert them to temperature, and display them on an LCD. Although an MSP430F413 is used for this application report, any MSP430™ MCU could be used to implement this application.

This document includes a complete schematic and code listing. The code can also be downloaded from http://www.ti.com/lit/zip/slaa125.

Contents

Trademarks

MSP430 is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

1 Introduction

Thermocouples typically require signal conditioning to amplify the thermocouple voltage into the ADC range and to provide any offset that is required. This application report describes how a high-resolution ADC, such as the 24-bit ADS1240, can be used to implement a direct thermocouple interface without the need for signal conditioning. The thermocouple is connected directly to the inputs of the ADC. An MSP430F413 MCU communicates with the ADS1240 to read the ADC values, convert the ADC values into temperature, and display the temperature on the LCD. For this application report, a type K thermocouple was used, and the temperature range was limited to 0°C to 99.9°C.

2 Thermocouples

Thermocouples are constructed of two dissimilar metals welded at one end. They produce a voltage at the nonwelded end that is relative to the temperature difference between the two ends of the thermocouple. There are many types of thermocouples and much has been written on thermocouples and thermocouple usage. A simple search on the Internet can provide a host of useful information for anyone wanting to learn the intricate details of thermocouples and thermocouple usage that are not covered in this report.

The voltage produced by thermocouples depends on the temperature difference between the ends of the thermocouple. It is not enough to simply measure the voltage to determine the temperature. That measurement gives only the temperature difference between the two ends of the thermocouple. The temperature of the cold junction (the connection of the thermocouple to the measuring device) affects the voltage produced at the thermocouple end. As a result, some type of cold junction compensation is required. Often, circuits are employed to produce a voltage proportional to the cold junction temperature. This voltage is injected into the circuit and is part of the typical thermocouple signal conditioning circuitry.

Another technique of cold junction compensation involves measuring the temperature of the junction with a thermistor or some other type of temperature sensor. This is the technique employed in this report. In this technique, the temperature of the welded end of the thermocouple can be determined from knowing the cold junction temperature and measuring the thermocouple voltage. For accurate results, this technique requires the use of an isothermal block to assure the temperature of the cold junction temperature sensor is the same as the temperature of the cold junction.

The voltage that thermocouples produce is standardized by the National Institute of Standards and Technology. Data tables for thermocouple voltages are available from the NIST.

3 Application Circuit

Figure 1 shows the circuit diagram for the circuit used in this application report. Section 5 includes a complete schematic.

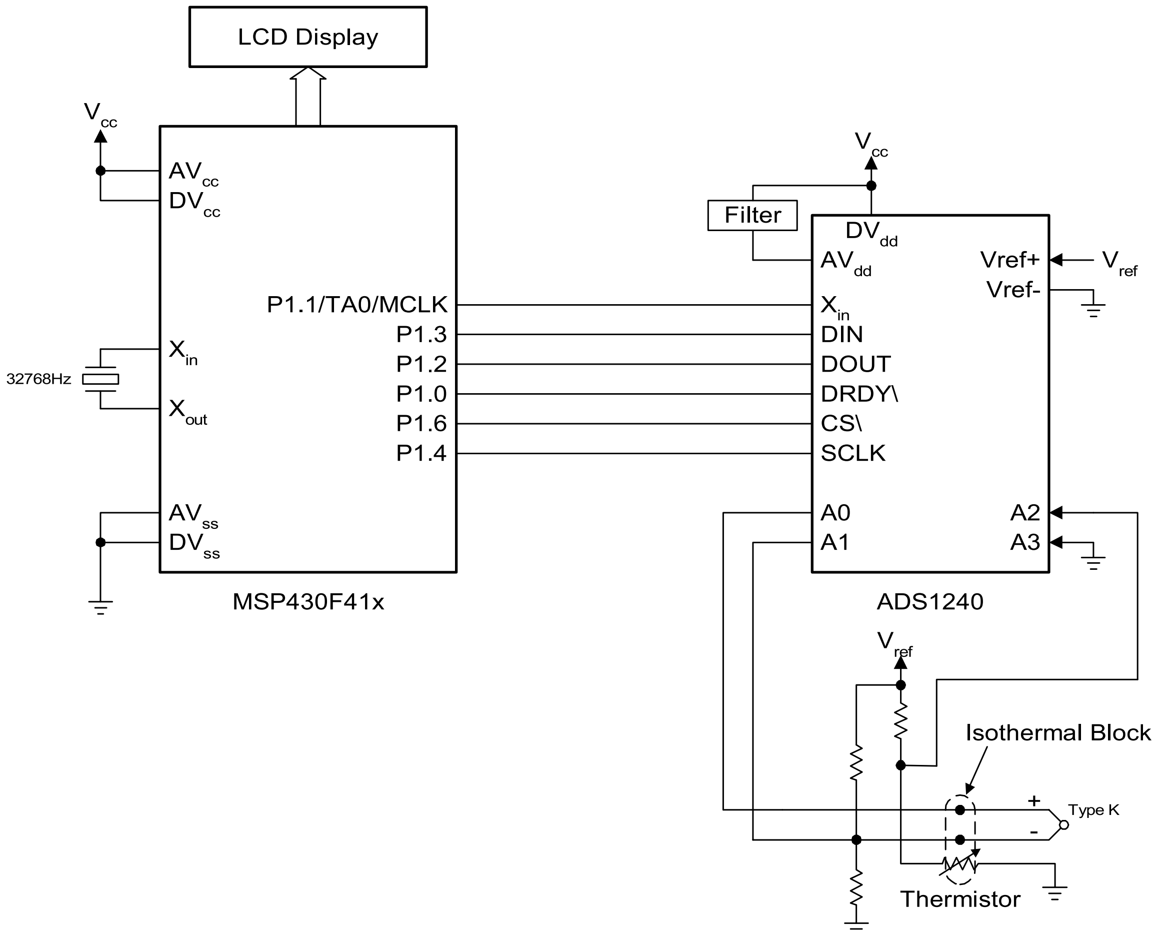

Figure 1. Circuit Diagram

- A thermistor is used to provide cold junction compensation.
- Several pins from port P1 on the MSP430F413 are used to implement serial communication with the ADS1240.
- A voltage divider is used to bias the negative terminal of the thermocouple.
- The ADS1240 is clocked from the MCLK output of the MSP430F413.
- The clock frequency is set at 1.5 MHz.

4 Implementation and Code Details

Figure 2 shows the program flow. The following sections describe this program flow.

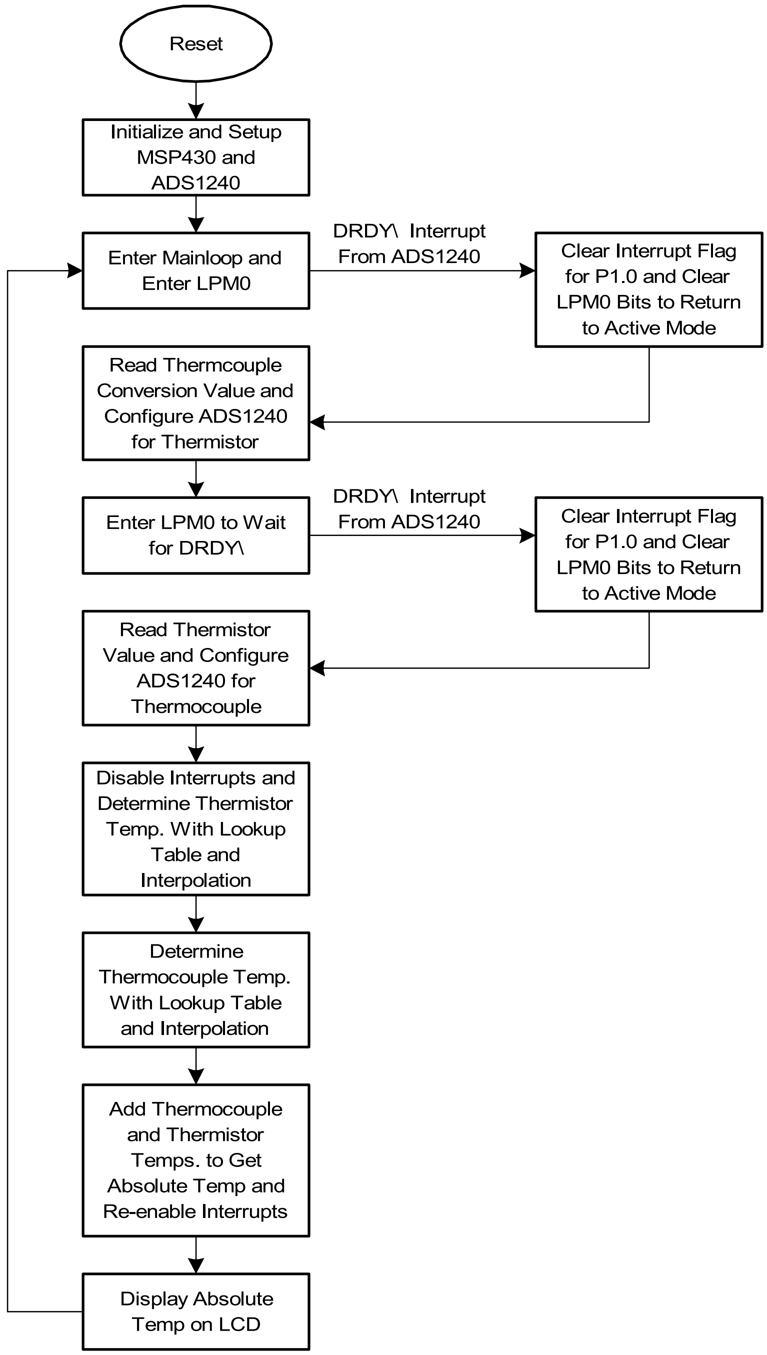

Figure 2. Program Flow

4.1 Initialization and Setup

The MSP430F413 and the ADS1240 are both initialized after a reset. On the MSP430F413, the watchdog timer is disabled. The FLL+ and DCO are configured to generate an MCLK of 1.507 MHz. This frequency is used for the CPU and is used externally to clock the ADS1240. This frequency was chosen for the ADS1240 because it is fast enough for the application but allows some power savings over the nominal 2.4576-MHz reference in the ADS1240 data sheet. The Basic Timer and LCD modules are configured for the LCD, and the LCD memory is cleared. The I/O ports are configured for desired function, direction and interrupt edge. The unused I/O ports are set to the output direction. A delay is implemented before calling the ADC setup routine. The delay allows the FLL+ time to adjust the DCO to the desired frequency of 1.507 MHz. For more information on the FLL+, DCO, or any other MSP430F413 peripheral, refer to the MSP430x4xx Family User's Guide. For more information on the MSP430F41x devices, see the MSP430x41x Mixed-Signal Microcontrollers data sheet.

To setup the ADS1240, a reset command is issued first. Then the input multiplexer is set for channel 0 and 1 for the plus and minus inputs, respectively. These are for the plus and minus sides of the thermocouple. The data rate is selected for the ADS1240 as the slowest it offers. This results in approximately 2.3 conversions per second, rather than the 3.75 listed in the data sheet, because the values in the data sheet assume a clock of 2.4576 MHz, but this application uses a clock of 1.5 MHz. When writing to successive registers of the ADS1240, it is not necessary to send the WREG command each time (see the ADS1240 24-Bit Analog-to-Digital Converters data sheet for more details).

Interrupts are enabled on the MSP430F413, and the SELFGCAL command is issued to the ADS1240. The SELFGCAL command works best when the PGA = 1, and it requires some time to complete. The $\overline{\text{DRDY}}$ line of the ADS1240 goes low to signal the completion of the SELFGCAL process. Interrupts are enabled on the MSP430F413 to allow the MSP430F413 to enter low-power mode 0 (LPM0) while the SELFGCAL process is being executed and to be interrupted by the ADS1240 when the process completes. Finally, the programmable gain amplifier (PGA) is set to a gain of 16 and the SELFOCAL command is issued. For more information on the ADS1240, refer to the ADS1240 24-Bit Analog-to-Digital Converters data sheet.

4.2 Mainloop

The mainloop is a succession of calls to other routines with a statements to enter LPM0. The first instruction of the mainloop puts the MSP430F413 into LPM0. The MSP430F413 stays in LPM0 until the $\overline{\text{DRDY}}$ line from the ADS1240 goes low, signaling that data is ready to be read. When the $\overline{\text{DRDY}}$ line goes low, the MSP430F413 receives the P1.0 interrupt and awakens to service it. The interrupt service routine then clears the appropriate status register bits so the MSP430F413 returns from the interrupt in active mode. Next, a call is made to the routine that reads the ADS1240. This value is for the thermocouple. Upon return from that routine, the MSP430F413 is again put into LPM0 to wait for the next $\overline{\text{DRDY}}$. After the next $\overline{\text{DRDY}}$, the ADS1240 is ready to be read again, so a call is made to the routine to read the thermistor value (see Figure 2). Upon return from reading the thermistor value, the MSP430F413 is ready to determine the temperature. Interrupts are disabled and calls are made to routines that convert the thermistor and thermocouple ADC values into their respective temperatures and then a call is made to determine the absolute temperature based on the thermistor and thermocouple temperatures. After the temperatures are determined, interrupts are re-enabled and a call is made to display the absolute temperature on the LCD. Then the mainloop starts over.

4.3 ReadTC and ReadTR Routines

The ReadTC and ReadTR routines read the thermocouple and thermistor values, respectively, from the ADS1240. Each routine sends the RDATA command and then clocks the data out of the ADS1240. Then, each routine sets up the PGA and multiplexer settings of the ADS1240 for the next conversion. The routine to read the thermocouple value sets up the ADS1240 to perform a conversion on the thermistor. The thermistor ADC value is then available at the next assertion of $\overline{\text{DRDY}}$. Likewise, the routine to read the thermistor value sets up the ADS1240 to perform a conversion on the thermocouple.

4.4 Get_TR_Temp – Determining the Thermistor Temperature

To read the thermistor in this application a resistor divider is formed with a 10-kΩ resistor and 10-kΩ thermistor. The top of divider is connected to the reference and bottom to ground. This provides a voltage input to the ADC that varies with temperature. Thermistor values decrease as temperature increases, so the voltage from this divider decreases with temperature as well (see Figure 1). Perhaps a more common way of measuring a thermistor is with a slope analog-to-digital conversion. A slope conversion was not employed in this application, because a high-performance precision ADC was already available, and performing the slope analog-to-digital conversion would have unnecessarily complicated the application and the code. A complete application report with code examples are available to show how to perform a slope analog-to-digital conversion with the MSP430F4xx MCUs (see the MSP430 Based Digital Thermometer application report).

To determine the thermistor temperature from the ADC value, first a table lookup is performed to determine the temperature to the nearest degree. Then, an interpolation is done to determine the temperature to the nearest tenth of a degree. To do this, first the nearest whole degree temperature is multiplied by 10 and saved to RAM. Then, the tenths are determined from Equation 1.

$$\frac{(\text{higher} - \text{ADCvalue}) \times 10}{\text{higher} - \text{lower}}$$

where

- ADCvalue = ADC conversion value of thermistor voltage
- higher = the next higher value in the table
- lower = the next lower value in the table (1)

After the tenths are interpolated, they are added to the nearest whole temperature that was multiplied by ten previously. Now the thermistor temperature is stored in the form of XX.X in binary format.

The table of thermistor values shown in the code in Section 6 is specific to this application. This table uses measured values for the 10-kΩ resistor and for the ADC reference and uses only 16 of the 24 ADC bits. Equation 2 shows the general formula for computing the table.

$$\text{ADCvalue} = \text{hex}\left(2^N \times \frac{\text{voltage}}{2 \times \text{Vref}}\right)$$

where

- N = Desired resolution of the analog-to-digital conversion (the ADS1240 is a 24-bit ADC, but for this application only the upper 16 bits are used, so in this case, N is 16).
- voltage = Resulting voltage from the voltage divider
- Vref = Reference for the ADS1240 (2)

For this application, Equation 3 shows the equation for the voltage divider.

$$\text{voltage} = \frac{\text{V} \times \text{Rt}}{\text{Rt} + 10\text{k}}$$

where

- Rt = Thermistor resistance
- V = Voltage source for the divider – Vref in this application (3)

Combining the two equations results in Equation 4.

$$\text{ADCvalue} = \text{hex}\left[\left(\frac{2^N}{2 \times \text{Vref}}\right)\left(\frac{\text{Vref} \times \text{Rt}}{\text{Rt} + 10\text{k}}\right)\right] \quad (4)$$

Which reduces to Equation 5.

$$\text{ADCvalue} = \text{hex}\left(\frac{2^{N-1} \times \text{Rt}}{\text{Rt} + 10\text{k}}\right) \quad (5)$$

4.5 Get_TC_Temp – Determining the Thermocouple's Relative Temperature

Converting a thermocouple voltage measurement into temperature can sometimes be a difficult task, depending on the application, because of the nonlinearity of thermocouples. However, this application report simplifies the task by limiting the measurable temperature range and by using a table lookup rather than a mathematical calculation. A type K thermocouple was used for this application report.

Determining the thermocouple relative temperature is done with a lookup table and tenths interpolation, the same as was described in Section 4.4 for the thermistor temperature. If the temperature of the thermocouple tip is less than the cold junction temperature, the voltage from the thermocouple will be negative and the ADC value will be negative. At the completion of the routine, the thermocouple temperature is stored in RAM in the form of XXX.X and has the appropriate sign.

As with the thermistor, the table of values used for the thermocouple temperature lookup is specific to this application and incorporates measured values. Equation 6 shows the general formula for computing the table.

$$\text{ADCvalue} = \text{hex}\left(\frac{2^N \times \text{PGA} \times \text{Vtc}}{2 \times \text{Vref}}\right)$$

where

- N = Desired resolution of the analog-to-digital conversion (the ADS1240 is a 24-bit ADC, but for this application only the upper 16 bits are used, so in this case N is 16)
- PGA = Gain from the programmable gain amplifier
- Vref = Reference for the ADC
- Vtc = Thermocouple voltage (6)

4.6 Get_ABS_Temp – Determining the Absolute Temperature

The Get_ABS_Temp routine adds the thermistor (cold junction) temperature to the relative thermocouple measurement to produce the absolute temperature of the thermocouple tip. After adding, the result is checked to determines if it is negative. If so, it falls outside of the range of this application report (0°C to 99.9°C). However, the range checking is performed in the display routine, not in this routine. Therefore, if the absolute temperature is negative, then it is simply stored in RAM as a negative value. If the temperature is positive, it is converted to the BCD format and stored in RAM in BCD format.

4.7 DISPLCD – Display the Absolute Temperature on the LCD

The DISPLCD routine checks the absolute temperature to make sure it is within the specified range. If it is not, either an L or an H is displayed on the LCD to indicate the out-of-range temperature. If the measured temperature is within range, the bits are manipulated appropriately for display on the LCD. The code in this application report is written for demonstration purposes and must be modified for the LCD of your choice.

5 Circuit Schematic

Figure 3 shows a schematic of the circuit used in this application.

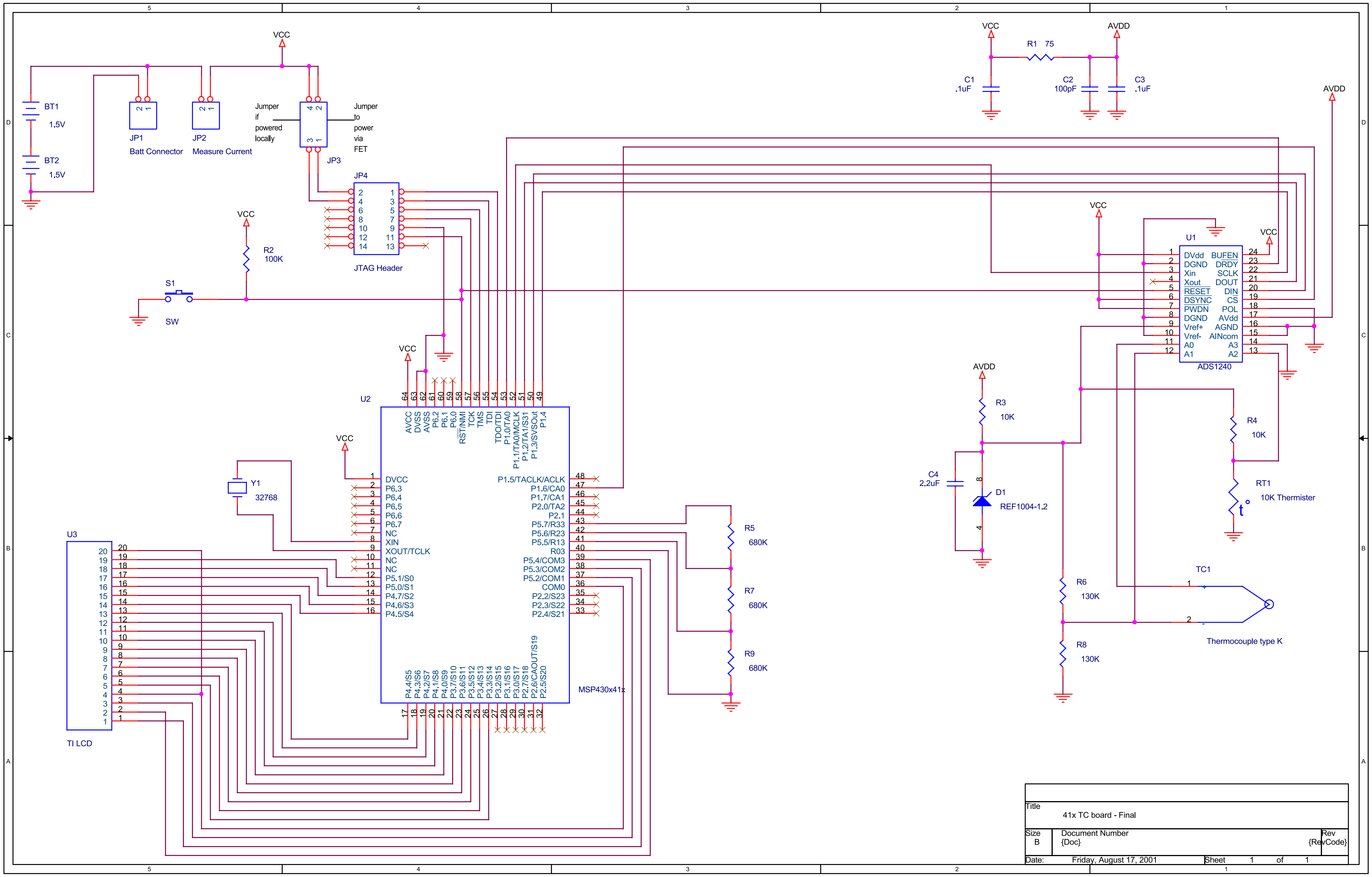

Figure 3. Circuit Schematic

6 Application Code

```
; THIS PROGRAM IS PROVIDED "AS IS". TI MAKES NO WARRANTIES OR
; REPRESENTATIONS, EITHER EXPRESS, IMPLIED OR STATUTORY,
; INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS
; FOR A PARTICULAR PURPOSE, LACK OF VIRUSES, ACCURACY OR
; COMPLETENESS OF RESPONSES, RESULTS AND LACK OF NEGLIGENCE.
; TI DISCLAIMS ANY WARRANTY OF TITLE, QUIET ENJOYMENT, QUIET
; POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY
; INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE PROGRAM OR
; YOUR USE OF THE PROGRAM.
;
; IN NO EVENT SHALL TI BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
; CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY
; THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED
; OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT
; OF THIS AGREEMENT, THE PROGRAM, OR YOUR USE OF THE PROGRAM.
; EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED TO, COST OF
; REMOVAL OR REINSTALLATION, COMPUTER TIME, LABOR COSTS, LOSS
; OF GOODWILL, LOSS OF PROFITS, LOSS OF SAVINGS, OR LOSS OF
; USE OR INTERRUPTION OF BUSINESS. IN NO EVENT WILL TI'S
; AGGREGATE LIABILITY UNDER THIS AGREEMENT OR ARISING OUT OF
; YOUR USE OF THE PROGRAM EXCEED FIVE HUNDRED DOLLARS
; (U.S.$500).
;
; Unless otherwise stated, the Program written and copyrighted
; by Texas Instruments is distributed as "freeware".  You may,
; only under TI's copyright in the Program, use and modify the
; Program without any charge or restriction.  You may
; distribute to third parties, provided that you transfer a
; copy of this license to the third party and the third party
; agrees to these terms by its first use of the Program. You
; must reproduce the copyright notice and any other legend of
; ownership on each copy or partial copy, of the Program.
;
; You acknowledge and agree that the Program contains
; copyrighted material, trade secrets and other TI proprietary
; information and is protected by copyright laws,
; international copyright treaties, and trade secret laws, as
; well as other intellectual property laws.  To protect TI's
; rights in the Program, you agree not to decompile, reverse
; engineer, disassemble or otherwise translate any object code
; versions of the Program to a human-readable form.  You agree
; that in no event will you alter, remove or destroy any
; copyright notice included in the Program.  TI reserves all
; rights not specifically granted under this license. Except
; as specifically provided herein, nothing in this agreement
; shall be construed as conferring by implication, estoppel,
; or otherwise, upon you, any license or other right under any
; TI patents, copyrights or trade secrets.
;
; You may not use the Program in non-TI devices.

#include  "msp430x41x.h"
;*****************************************************************************
;
;           Implement a direct thermocouple interface with the ADS1240 and
;           MSP430.
;
;           Mike Mitchell
;           MSP430 Applications
```

```
;           May 10, 2001
;
;*****************************************************************************

; Other Definitions

#define   ADC_HI  R14
#define   ADC_LO  R10
#define   TXBUF   R7
#define   RXBUF   R6
#define   BITCNT  R5

; LCD segment Definitions
a         EQU     001h
b         EQU     002h
c         EQU     010h
d         EQU     004h
e         EQU     080h
f         EQU     020h
g         EQU     008h
h         EQU     040h

; ADS1240 serial communication definitions
Dout      EQU     004h
Din       EQU     008h
SCLK      EQU     010h
CS        EQU     040h

; ADS1240 Register Definitions (Not complete list, see ADS1240 data sheet)
SETUP     EQU     000h
MUX       EQU     001h
ACR       EQU     002h

; ADS1240 Commands (Not complete list, see ADS1240 data sheet)
RDATA     EQU     001h
WREG      EQU     050h      ; The opcode for WREG is <0101 reg xxxx number>
RREG      EQU     010h      ; The opcode for RREG is <0001 reg xxxx number>
SELFCAL   EQU     0F0h
SELFOCAL  EQU     0F1h
SELFGCAL  EQU     0F2h
SYSOCAL   EQU     0F3H
SYSGCAL   EQU     0F4h
RST       EQU     0FEh


;*****************************************************************************
            RSEG    UDATA0      ; RAM Locations
;*****************************************************************************

TC_ADC_HI         DS      2     ; Thermocouple ADC reading, high word
TC_ADC_LO         DS      1     ; Thermocouple ADC reading, low byte
TR_ADC_LO         DS      1     ; Thermistor ADC reading, low byte
TR_ADC_HI         DS      2     ; Thermistor ADC reading, high word
TC_TMP            DS      2     ; Thermocouple temperature
TR_TMP            DS      2     ; Thermistor temperature
ABS_TMP           DS      2     ; Absolute temperature (TR_TMP+TC_TMP)
TC_NEG            DS      1     ; Used for keeping track of negative TC value

;*****************************************************************************
            RSEG    CSTACK
```

```
;*****************************************************************************

            DS      0

;*****************************************************************************
            RSEG    CODE
;*****************************************************************************

;  Setup uC and ADC
RESET       mov     #SFE(CSTACK),SP      ; define stackpointer
SetupWDT    mov     #WDTPW+WDTHOLD,&WDTCTL  ; Stop Watchdog Timer
SetupFLL    mov.b   #02Dh,&SCFQCTL       ; Set N=45. 46x32KHz = 1.5Mhz
                                         ; Set clock for ADS1240 to be above
                                         ; min of 1MHz for some margin.
SetupLCD    mov.b   #LCDON+LCD4MUX+LCDP2,&LCDCTL   ; LCD 4Mux, S0-S17
SetupBT     mov.b   #BTDIV+BTFRFQ1,&BTCTL  ; LCD freq.
ClearLCD    mov     #15,R15              ; Clear 15 bytes of LCD RAM
Clear1      clr.b   LCDMEM-1(R15)        ; Clear LCD RAM
            dec     R15                  ; All 15 bytes of LCD RAM?
            jnz     Clear1               ; Not done?

SetupIO     mov.b   #0ffh,&P5SEL         ; Select LCD functions
            mov.b   #BIT0+BIT1,&P5DIR    ; Set output direction
            mov.b   #BIT1,&P1SEL         ; Select MCLK output for pin
            mov.b   #BIT0,&P1IES         ; Select H/L transition for interrupt
                                         ; DRDY\ is connected pin P1.0.
            mov.b   #0,&P1IFG            ; The previous instruction could have
                                         ; set the P1.0 IFG, so clear it.
            mov.b   #BIT0,&P1IE          ; Enable interrupt on P1.0
            mov.b   #BIT1+Din+SCLK+CS,&P1DIR
                                         ; Set direction for serial comm. pins
                                         ; MCLK used as OSC input to ADS1240
            mov.b   #CS,&P1OUT           ; Set CS High, everything else low

SetupIO2  ; Set all unused IO ports to output direction
            bis.b   #BIT5+BIT6+BIT7,&P1DIR
            bis.b   #0FFh,&P2DIR
            bis.b   #BIT0+BIT1+BIT2,&P3DIR
            bis.b   #0FFh,&P6DIR

Delay1      mov    #0fffh,R15            ; Delay to allow FLL+ to
loop1       dec    R15                   ; sync up MCLK
            jnz    loop1

            Call   #SetupADC             ; Call setup routine for ADC

Mainloop    bis.w   #LPM0,SR             ; Enter LPM0
            call    #ReadTC              ; Read Thermocouple
            bis.w   #LPM0,SR             ; Enter LPM0
            call    #ReadTR              ; Read Thermistor
            call    #Get_TR_temp         ; Determine temperature with tables
                                         ; and interpolation
            call    #Get_TC_temp         ; Determine temperature with tables
                                         ; and interpolation
            call    #Get_ABS_temp        ; Determine absolute temperature
            call    #DISPLCD             ; Display
            jmp     Mainloop             ; Return to low power mode

;*****************************************************************************
SetupADC     ; Routine to setup the ADS1240 ADC
```

```
;*******************************************************************************
            bic.b   #CS,&P1OUT          ; Take CS\ low to select the ADC.

; Issue reset command
            mov.b   #RST,TXBUF          ; Move Reset command to TX buffer (R15)
            mov.b   #08h,BITCNT         ; Set bit count value
            call    #TXLoop             ; Shift out the Reset command
            call    #SCLKdlay           ; Delay for SCLK

; Set PGA to 16X to amplify thermocouple voltage
            mov.b   #WREG+SETUP,TXBUF   ; Move write register command plus reg
            mov.b   #08h,BITCNT         ; Set bit count value
            call    #TXLoop             ; Shift out command
            mov.b   #02h,TXBUF          ; Second part of write reg command
                                        ; Denotes going to write 3 registers total:
                                        ; Setup, Mux, and ACR
            mov.b   #08h,BITCNT         ; Set bit count value
            call    #TXLoop             ; Shift out command
            mov.b   #004h,TXBUF         ; Set PGA to 16X
            mov.b   #08h,BITCNT         ; Set bit count value
            call    #TXLoop             ; Shift out command
            call    #SCLKdlay           ; Delay for SCLK

; Select channels 0,1,common for +,- inputs.  This is for the thermocouple
            mov.b   #01h,TXBUF          ; Select analog channels for conversion
            mov.b   #08h,BITCNT         ; Set bit count value
            call    #TXLoop             ; Shift out command
            call    #SCLKdlay           ; Delay for SCLK

; Setup up Data rate with ACR register
            mov.b   #002h,TXBUF         ; Select slowest data rate for DRDY\.
            mov.b   #08h,BITCNT         ; Set bit count value
            call    #TXLoop             ; Shift out command
            call    #SCLKdlay           ; Delay for SCLK

; enable interrupts
            eint                        ; Enable interrupts
                                        ; Interrupts needed at this point to
                                        ; delay for SELFCAL command.

; Issue SELFCAL command
            mov.b   #SELFCAL,TXBUF      ; Move Self cal command to TX buffer
            mov.b   #08h,BITCNT         ; Set bit count value
            call    #TXLoop             ; Shift out the command
            bis     #LPM0,SR            ; Go to sleep until DRDY\ goes low.
                                        ; At that point the selfcal is done.
            bis.b   #CS,&P1OUT          ; Take CS\ back high again
            ret                         ; Done setting up ADC.


;*******************************************************************************
ReadTC     ; Routine to read thermocouple value
;*******************************************************************************
           bic.b   #CS,&P1OUT          ; Take CS\ low to select ADC.

; Transfer read data command
            mov     #RDATA,TXBUF        ; Move read command to TX buffer
            mov.b   #08h,BITCNT         ; Set Bit count value
            call    #TXLoop             ; Shift out the read command
            call    #SCLKdlay           ; Delay for SCLK
```

```
; Get the data from the ADC
            mov     #10h,BITCNT          ; Set bit counter to 16 bits to shift
                                         ; in the upper 16 bits of ADC value.
            call    #RXLoop              ; Call shift routine
            mov     RXBUF,TC_ADC_HI      ; Save off high word of data.
            mov     #08h,BITCNT          ; Set bit counter to 8 to shift in
                                         ; lower 8 bits of ADC data.
            call    #RXLoop              ; Call shift routine
            mov.b   RXBUF,TC_ADC_LO      ; Save off low byte of ADC data

; Now set mux and gain for thermistor for next ADC read.
            mov.b   #WREG+SETUP,TXBUF    ; Move write register command plus reg
            mov.b   #08h,BITCNT          ; Set bit count value
            call    #TXLoop              ; Shift out command
            mov.b   #01h,TXBUF           ; Second part of write reg command
            mov.b   #08h,BITCNT          ; Set bit count value
            call    #TXLoop              ; Shift out command
            call    #SCLKdlay            ; delay for SCLK

; Set PGA to 1X.  The Thermistor needs no amp.
            mov.b   #00h,TXBUF           ; Set PGA to 1X
            mov.b   #08h,BITCNT          ; Set bit count value
            call    #TXLoop              ; Shift out command
            call    #SCLKdlay            ; Delay for SCLK

; Select channels 2,common for +,- inputs.
            mov.b   #028h,TXBUF          ; Select analog channels for conversion
            mov.b   #08h,BITCNT          ; Set bit count value
            call    #TXLoop              ; Shift out command
            call    #SCLKdlay            ; Delay for SCLK

; Done reading values, set CS\ high and return
            bis.b   #CS,&P1OUT            ; Take CS\ back high.
            ret

;*******************************************************************************
ReadTR      ; Routine to read thermistor value
;*******************************************************************************
            bic.b   #CS,&P1OUT           ; Take CS\ low to select ADC.

; First read the data for the thermistor
; Transfer read data command
            mov     #RDATA,TXBUF         ; Move read command to TX buffer
            mov.b   #08h,BITCNT          ; Set Bit count value
            call    #TXLoop              ; Shift out the read command
            call    #SCLKdlay            ; Delay for SCLK;

; Get the data from the ADC
            mov     #10h,BITCNT          ; Set bit counter to 16 bits to shift
                                         ; in the upper 16 bits of ADC value.
            call    #RXLoop              ; Call shift routine
            mov     RXBUF,TR_ADC_HI      ; Save off high word of data.
            mov     #08h,BITCNT          ; Set bit counter to 8 to shift in
                                         ; lower 8 bits of ADC data.
            call    #RXLoop              ; Call shift routine
            mov.b   RXBUF,TR_ADC_LO      ; Save off low byte of ADC data

; Now set mux and gain for thermocouple for next ADC read.
            mov.b   #WREG+SETUP,TXBUF    ; Move write register command plus reg
```

```
            mov.b   #08h,BITCNT         ; Set bit count value
            call    #TXLoop             ; Shift out command
            mov.b   #01h,TXBUF          ; Second part of write reg command.
            mov.b   #08h,BITCNT         ; Set bit count value
            call    #TXLoop             ; Shift out command

; Set PGA to 16X to amplify thermocouple voltage.
            mov.b   #004h,TXBUF         ; Set PGA to 16X
            mov.b   #08h,BITCNT         ; Set bit count value
            call    #TXLoop             ; Shift out command
            call    #SCLKdlay           ; Delay for SCLK

; Select channels 0,1,common for +,- inputs.
            mov.b   #01h,TXBUF          ; Select analog channels for conversion
            mov.b   #08h,BITCNT         ; Set bit count value
            call    #TXLoop             ; Shift out command
            call    #SCLKdlay           ; Delay for SCLK

; Done reading values, set CS\ high and return
            bis.b   #CS,&P1OUT           ; Take CS\ back high.
            ret

;*******************************************************************************
TXLoop     ; Routine to transmit to the ADS1240 ADC
;*******************************************************************************
            bis.b   #SCLK,&P1OUT        ; Set SCLK high
            rla.b   TXBUF               ; Rotate TXBUF through C
            jc      Set_H               ; Jump if C is High
Set_L       bic.b   #Din,&P1OUT         ; Set Dout Low
            jmp     A
Set_H       bis.b   #Din,&P1OUT         ; Set Dout High
A           bic.b   #SCLK,&P1OUT        ; Set SCLK Low to latch data into ADC
            dec.b   BITCNT              ; Decrement Bit counter
            jnz     TXLoop              ; Continue if not done

            ret

;*******************************************************************************
RXLoop      ; Routine to shift in ADC data
;*******************************************************************************
            mov     #0,RXBUF            ; Clear buffer
L1          bis.b   #SCLK,&P1OUT        ; Set SCLK high.
            nop                         ; Short delay
            bic.b   #SCLK,&P1OUT        ; Set SCLK low
            bit.b   #Dout,&P1IN         ; Latch data into Carry bit
            rlc     RXBUF               ; C -> Receive buffer
            dec     BITCNT              ; Decrement bit counter
            jnz     L1                  ; Continue if not done
            ret

;*******************************************************************************
SCLKdlay     ; Delay loop to meet spec. for SCLK inactive
;*******************************************************************************
            mov     #0Fh,R7             ; load delay value
L2          dec     R7                  ; decrement
            jnz     L2                  ; repeat until zero
            ret

;*******************************************************************************
Get_TR_temp       ; Determine temperature of thermistor
```

```
;                  ; using tables and interpolation
;                  ;
;*******************************************************************************
            dint                            ; Disable interrupts.
            mov     TR_ADC_HI,R14           ; Move high word of TR data to R14
                                            ; Not using low byte in this app.
                                            ; Only using upper 16-bits.
            mov     #0h,R5                  ; Use R5 as table pointer
            mov     #0h,R13                 ; Use R13 for temperature value
            cmp     TR_Temps(R5),R14        ; Compare A/D high word to table
            jge     End_TR_Temp             ; If Thermistor value is greater than
                                            ; first value in table, then it's temp
                                            ; is too low, so end.  If equal, then
                                            ; then temp is = 0.
            incd    R5                      ; Point to second value in table
CMPloop1    cmp     TR_Temps(R5),R14        ; Compare again
            jeq     End_LU_1                ; If equal, need to add one to temp
                                            ; value then end.
            jge     End_TR_lookup           ; If greater, then end routine
            incd    R5                      ; Point to next value in table
            inc     R13                     ; Add one to temp value
            jmp     CMPloop1                ; Repeat loop

End_LU_1    inc     R13                     ; Add one to temp value
            incd    R5                      ; Also need to increment pointer

End_TR_lookup
; Resolved to nearest degree, now multiply by 10 and interpolate tenths.
; After tenths are determined, simply add it.  Then will have temp to nearest
; tenth in binary format.
            rla     R13                     ; rotate left
            mov     R13,R12                 ; copy
            rla     R12                     ; rotate the copy two more times
            rla     R12                     ;
            add     R12,R13                 ; then add.
            mov     R13,TR_TMP              ; Save off to RAM

; Interpolate to tenths. Tenths = ((higher-ADCvalue)x10)/(higher-lower)

            mov     TR_Temps(R5),R13        ; Move table value to R13
            decd    R5                      ; Point to previous value in table
            mov     TR_Temps(R5),R15        ; Now, the higher is in R15
                                            ; the ADC value is in R14
                                            ; and the lower is in R13
            sub     R13,R15                 ; Get delta between lower and higher
            mov     TR_Temps(R5),R13        ; Move higher table value to R13
            sub     R14,R13                 ; Get delta between ADC value and higher
                                            ;
; Multiply delta between ADC value and higher table value by 10
            rla     R13                     ; rotate left
            mov     R13,R12                 ; copy
            rla     R12                     ; rotate the copy two more times
            rla     R12                     ; then add.
            add     R12,R13                 ; Done with x10.  Now need to divide

; Divide (higher-ADCvalue)x10 by delta between lower and upper to get tenths
;    Dividend = (higher-ADCvalue)x10, which is in R13
;    Divisor  = (higher-lower), which is in R15
            mov     #0,R12                  ; Use R12 as counter for subtractions
            cmp     R15,R13                 ; Compare to see if R13>R15.
```

```
            jl      remainder               ; If not, no division, just remainder
Divide1     sub     R15,R13                 ; Dividend-Divisor -> dividend
            inc.b   R12
            cmp     R15,R13                 ; Is Dividend>Divisor?
            jge     Divide1                 ; Yes, subtract and increment again
remainder   rla     R13                     ; Multiply remainder by two
            cmp     R15,R13                 ; Is Dividend>Divisor?
            jl      EndDiv1                 ; If not, return
            inc.b   R12                     ; If so, incrment counter to round up

EndDiv1     ;    Division now done, so Add tenths (R12) to the whole number (TR_TMP)
            add    R12,TR_TMP               ; Now have xx.x (in binary) for the
                                            ; thermistor temperature.
            ret                             ; Done with TR temp. Note interrupts not
                                            ; re-enabled until end of Get_ABS_temp

End_TR_Temp
            mov    #0h,TR_TMP               ; If get here, TR temp is 0 or too low
                                            ; So store 00.0 as the temp.
            ret                             ; Return.  Note interrupts not re-
                                            ; enabled until end of Get_ABS_temp

;*******************************************************************************
Get_TC_temp       ; Determine temperature of thermocouple
;                 ; using tables and interpolation
;                 ;
;*******************************************************************************
            mov     TC_ADC_HI,R14           ; Move high word of TC data to R14
                                            ; Not using low byte in this app.
                                            ; Only using upper 16-bits.
            mov     #0h,R5                  ; Use R5 as table pointer
            mov     #0FFD7h,R13             ; Use R13 for temperature value
                                            ; Preload it with -41 deg C.
            cmp     TC_Temps(R5),R14        ; Compare to table
            jl      End_TC2                 ; Jump if A/D < 1st table value
            jeq     End_TC1                 ; Jump if A/D = 1st table value
CMPloop2    incd    R5                      ; Point to next value in table
            inc     R13                     ; Add one to temp value
            cmp     TC_Temps(R5),R14        ; Compare again
            jge     CMPloop2                ; If R14 greater or equal, loop

End_TC_lookup ;
; Resolved to nearest degree, now multiply by 10 and interpolate tenths.
; After tenths are determined, simply add it.  Then will have temp to nearest
; tenth in binary format.
            rla     R13                     ; rotate left
            mov     R13,R12                 ; copy
            rla     R12                     ; rotate the copy two more times
            rla     R12                     ;
            add     R12,R13                 ; then add.
            mov     R13,TC_TMP              ; Save off to RAM

; Interpolate to tenths. Tenths = ((ADCvalue-lower)x10)/(higher-lower)
            mov     TC_Temps(R5),R15        ; Move table value to register
            decd    R5                      ; decrement table pointer
            mov     TC_Temps(R5),R13        ; Now, the higher is in R15
                                            ; the ADC value is in R14
                                            ; and the lower is in R13
            sub     R13,R15                 ; Get delta between lower and higher
            sub     R13,R14                 ; Get delta between ADC value and lower
```

```
                                        ;
; Multiply delta between ADC value and lower table value (which is in R14) by 10
            rla     R14                 ; rotate left
            mov     R14,R12             ; copy
            rla     R12                 ; rotate the copy two more times
            rla     R12                 ; then add.
            add     R12,R14             ; Done with x10.  Now need to divide.

; Divide (ADC-lower)x10 by delta between lower and upper to get tenths
;    Dividend = (ADC-lower)x10, which is in R14
;    Divisor  = (higher-lower), which is in R15
            mov     #0,R12              ; Use R12 as counter for subtractions
            cmp     R15,R14             ; Compare R14, R15
            jl      remain              ; jump if R14<R15 - no division
Divide2     sub     R15,R14             ; Dividend-Divisor -> dividend
            inc.b   R12
            cmp     R15,R14             ; Is Dividend>Divisor?
            jge     Divide2             ; Yes, subtract and increment again
remain      rla     R14                 ; Multiply remainder by two
            cmp     R15,R14             ; Is Dividend>Divisor?
            jl      EndDiv2             ; If not, return
            inc.b   R12                 ; If so, increment counter to round up

EndDiv2     ;    Division now done, so Add tenths (R12) to the whole number (TC_TMP)
            add    R12,TC_TMP           ; Now have xx.x (in binary) in TC_TMP
            ret                         ; Done so return

End_TC1     inc     R13                 ; Increment R13 to -40 deg C.
End_TC2     mov     R13,TC_TMP          ; Move to RAM.
            ret                         ; Return

;End_TC_Temp ret                         ; Done

;*******************************************************************************
Get_ABS_temp      ; Calculate absolute temperature by adding the TC and TR
;                 ; temps.
;                 ;
;*******************************************************************************

; Add TR_TMP and TC_TMP to get ABS_TMP
Add_temps   mov     TC_TMP,R14          ; Put TC_TMP in R14
            add     TR_TMP,R14          ; Add TR_TMP to it
            bit     #08000h,R14         ; Test if result is negative
            jge     Bin_Dec             ; If not, proceed with conversion
            mov     R14,ABS_TMP         ; If so, store the negative temp
            jmp     End_Get_ABS_Temp    ; and jump to end

; Now convert to BCD format
Bin_Dec     rla      R14                ; Shift out upper nibble,
            rla      R14                ; not used for this
            rla      R14                ; conversion
            rla      R14                ;
            mov      #0Ch,R15           ; Loop counter
            clr      R12                ; Result goes into R12
L3          rla      R14                ; Shift MSB into C
            dadd     R12,R12            ; Add R14 to itself, plus C
            dec      R15
            jnz      L3                 ; Jump if not done

; Done converting to BCD
```

```
            mov     R12,ABS_TMP             ; Save absolute temp in BCD to RAM.

End_Get_ABS_Temp
            eint                            ; re-enable interrupts
            ret

;******************************************************************************
DISPLCD             ; Display temp on LCD.
;
;******************************************************************************
            mov     ABS_TMP,R14             ; Get temperature from RAM
            bit     #08000h,R14             ; Test if temp is negative
            jn      Disp_L                  ; If so, display "low" on LCD
            cmp     #01000h,R14             ; Test if temp >= 100
            jge     Disp_H                  ; If so, display "high" on LCD

; display temp on LCD
            mov.b   R14,R11                 ; Copy lower byte
            mov.b   R14,R12                 ; copy lower byte
            rra.b   R12                     ;
            rra.b   R12                     ;
            rra.b   R12                     ; Rotate right to expose nibble
            rra.b   R12                     ;
            swpb    R14                     ; Swap high and low bytes of data
            mov.b   R14,R13                 ; Copy again
            and     #0Fh,R13                ;
            and     #0Fh,R12                ;
            and     #0Fh,R11                ;
            mov.b   LCD_Tab(R11),&LCDM1
            mov.b   LCD_Tab(R12),&LCDM2
            bis.b   #040h,&LCDM2            ; Turn on decimal point
            mov.b   LCD_Tab(R13),&LCDM3
            mov.b   #0h,&LCDM4              ; Clear upper digit
            ret

; Display "L" on LCD
Disp_L      mov.b   #0h,&LCDM4
            mov.b   #0h,&LCDM3
            mov.b   #0h,&LCDM2
            mov.b   #d+e+f,&LCDM1
            ret

; Display "H" on LCD
Disp_H      mov.b   #0h,&LCDM4
            mov.b   #0h,&LCDM3
            mov.b   #0h,&LCDM2
            mov.b   #b+c+e+f+g,&LCDM1
            ret
;******************************************************************************
TR_Temps    DW      05DDEh   ; 0C
            DW      05CE1h   ; 1C
            DW      05BD5h   ; 2C
            DW      05AB9h   ; 3C
            DW      0598Ah   ; 4C
            DW      05848h   ; 5C
            DW      0573Dh   ; 6C
            DW      05624h   ; 7C
            DW      054FBh   ; 8C
            DW      053C1h   ; 9C
            DW      05274h   ; 10C
```

```
                DW      0515Ch    ; 11C
                DW      05035h    ; 12C
                DW      04EFFh    ; 13C
                DW      04DB9h    ; 14C
                DW      04C62h    ; 15C
                DW      04B46h    ; 16C
                DW      04A1Dh    ; 17C
                DW      048E7h    ; 18C
                DW      047A3h    ; 19C
                DW      0464Fh    ; 20C
                DW      04531h    ; 21C
                DW      04408h    ; 22C
                DW      042D3h    ; 23C
                DW      04191h    ; 24C
                DW      04042h    ; 25C
                DW      03F28h    ; 26C
                DW      03E05h    ; 27C
                DW      03CD8h    ; 28C
                DW      03BA0h    ; 29C
                DW      03A5Ch    ; 30C
                DW      0394Bh    ; 31C
                DW      03832h    ; 32C
                DW      03711h    ; 33C
                DW      035E6h    ; 34C
                DW      034B1h    ; 35C
                DW      033AFh    ; 36C
                DW      032A5h    ; 37C
                DW      03195h    ; 38C
                DW      0307Ch    ; 39C
                DW      02F5Ch    ; 40C
                DW      00000h    ; Too High

;*******************************************************************************

TC_Temps        DW      0FD78h    ; -40C
                DW      0FD87h    ; -39C
                DW      0FD97h    ; -38C
                DW      0FDA6h    ; -37C
                DW      0FDB6h    ; -36C
                DW      0FDC6h    ; -35C
                DW      0FDD6h    ; -34C
                DW      0FDE6h    ; -33C
                DW      0FDF5h    ; -32C
                DW      0FE05h    ; -31C
                DW      0FE15h    ; -30C
                DW      0FE25h    ; -29C
                DW      0FE35h    ; -28C
                DW      0FE45h    ; -27C
                DW      0FE55h    ; -26C
                DW      0FE65h    ; -25C
                DW      0FE75h    ; -24C
                DW      0FE85h    ; -23C
                DW      0FE95h    ; -22C
                DW      0FEA6h    ; -21C
                DW      0FEB6h    ; -20C
                DW      0FEC6h    ; -19C
                DW      0FED6h    ; -18C
                DW      0FEE7h    ; -17C
                DW      0FEF7h    ; -16C
                DW      0FF07h    ; -15C
```

```
DW      0FF18h   ; -14C
DW      0FF28h   ; -13C
DW      0FF38h   ; -12C
DW      0FF49h   ; -11C
DW      0FF5Ah   ; -10C
DW      0FF6Ah   ; -9C
DW      0FF7Bh   ; -8C
DW      0FF8Bh   ; -7C
DW      0FF9Ch   ; -6C
DW      0FFACh   ; -5C
DW      0FFBDh   ; -4C
DW      0FFCEh   ; -3C
DW      0FFDEh   ; -2C
DW      0FFEFh   ; -1C
DW      00000h   ;  0C
DW      00011h   ;  1C
DW      00022h   ;  2C
DW      00033h   ;  3C
DW      00043h   ;  4C
DW      00054h   ;  5C
DW      00065h   ;  6C
DW      00076h   ;  7C
DW      00087h   ;  8C
DW      00098h   ;  9C
DW      000A9h   ; 10C
DW      000BAh   ; 11C
DW      000CAh   ; 12C
DW      000DBh   ; 13C
DW      000ECh   ; 14C
DW      000FDh   ; 15C
DW      0010Eh   ; 16C
DW      0011Fh   ; 17C
DW      00131h   ; 18C
DW      00142h   ; 19C
DW      00153h   ; 20C
DW      00164h   ; 21C
DW      00175h   ; 22C
DW      00186h   ; 23C
DW      00198h   ; 24C
DW      001A9h   ; 25C
DW      001BAh   ; 26C
DW      001CBh   ; 27C
DW      001DCh   ; 28C
DW      001EEh   ; 29C
DW      001FFh   ; 30C
DW      00210h   ; 31C
DW      00222h   ; 32C
DW      00233h   ; 33C
DW      00244h   ; 34C
DW      00255h   ; 35C
DW      00267h   ; 36C
DW      00278h   ; 37C
DW      0028Ah   ; 38C
DW      0029Bh   ; 39C
DW      002ACh   ; 40C
DW      002BEh   ; 41C
DW      002CFh   ; 42C
DW      002E1h   ; 43C
DW      002F2h   ; 44C
DW      00303h   ; 45C
```

```
                DW      00315h   ; 46C
                DW      00326h   ; 47C
                DW      00338h   ; 48C
                DW      00349h   ; 49C
                DW      0035Bh   ; 50C
                DW      0036Ch   ; 51C
                DW      0037Eh   ; 52C
                DW      0038Fh   ; 53C
                DW      003A1h   ; 54C
                DW      003B3h   ; 55C
                DW      003C4h   ; 56C
                DW      003D5h   ; 57C
                DW      003E7h   ; 58C
                DW      003F9h   ; 59C
                DW      0040Ah   ; 60C
                DW      0041Ch   ; 61C
                DW      0042Dh   ; 62C
                DW      0043Fh   ; 63C
                DW      00451h   ; 64C
                DW      00462h   ; 65C
                DW      00474h   ; 66C
                DW      00486h   ; 67C
                DW      00497h   ; 68C
                DW      004A9h   ; 69C
                DW      004BAh   ; 70C
                DW      004CCh   ; 71C
                DW      004DEh   ; 72C
                DW      004EFh   ; 73C
                DW      00501h   ; 74C
                DW      00513h   ; 75C
                DW      00524h   ; 76C
                DW      00536h   ; 77C
                DW      00548h   ; 78C
                DW      00559h   ; 79C
                DW      0056Bh   ; 80C
                DW      0057Ch   ; 81C
                DW      0058Eh   ; 82C
                DW      005A0h   ; 83C
                DW      005B1h   ; 84C
                DW      005C3h   ; 85C
                DW      005D5h   ; 86C
                DW      005E6h   ; 87C
                DW      005F8h   ; 88C
                DW      00609h   ; 89C
                DW      0061Bh   ; 90C
                DW      0062Ch   ; 91C
                DW      0063Eh   ; 92C
                DW      00650h   ; 93C
                DW      00662h   ; 94C
                DW      00673h   ; 95C
                DW      00685h   ; 96C
                DW      00696h   ; 97C
                DW      006A8h   ; 98C
                DW      006B9h   ; 99C
                DW      006CBh   ; 100C
                DW      07FFFh   ; Too high

;*******************************************************************************
;  LCD Table of definitions
;  These use the bit definitions defined above
```

```
;  for each segment to turn on the appropriate
;  segments to form each numeral
;******************************************************************************
LCD_Tab      DB   a+b+c+d+e+f          ; displays "0"
             DB   b+c                  ; displays "1"
             DB   a+b+d+e+g            ; displays "2"
             DB   a+b+c+d+g            ; displays "3"
             DB   b+c+f+g              ; displays "4"
             DB   a+c+d+f+g            ; displays "5"
             DB   a+c+d+e+f+g          ; displays "6"
             DB   a+b+c                ; displays "7"
             DB   a+b+c+d+e+f+g        ; displays "8"
             DB   a+b+c+d+f+g          ; displays "9"
             DB   a+b+c+e+f+g          ; displays "A"
             DB   c+d+e+f+g            ; displays "b"
             DB   a+d+e+f              ; displays "C"
             DB   b+c+d+e+g            ; displays "d"
             DB   a+d+e+f+g            ; displays "E"
             DB   a+e+f+g              ; displays "F"

;******************************************************************************
P1_ISR;      Exit    LPM0
;******************************************************************************
             bic.b   #BIT0,&P1IFG        ; Clear P1.0 IFG
             bic.w   #LPM0,0(SP)         ; Clear LPM0 bits to exit LPM0
             reti                        ; on return from interrupt

;******************************************************************************
             COMMON    INTVEC              ; MSP430x41x Interrupt vectors
;******************************************************************************

             ORG     PORT1_VECTOR
             DW      P1_ISR              ; P1 ISR
             ORG     RESET_VECTOR
             DW      RESET               ; POR, ext. Reset, Watchdog
             END
;******************************************************************************
```

7 Conclusion

While traditional thermocouple circuits can be complex to design and implement, this report shows how new high-resolution ADCs and microcontrollers can simplify the task.

8 References

1. MSP430x4xx Family User's Guide
2. MSP430x41x Mixed-Signal Microcontrollers data sheet
3. ADS1240 24-Bit Analog-to-Digital Converters data sheet
4. MSP430 Based Digital Thermometer

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from October 4, 2001 to May 29, 2018	Page
• Editorial changes and format updates throughout document	1