Application Report
SLAA129C – May 2001 – Revised August 2018

Implementing An Ultra-Low-Power Thermostat With Slope Analog-to-Digital Conversion

ABSTRACT

This application report describes slope analog-to-digital measurement of a resistance and the ease with which it can be implemented with MSP430™ microcontrollers (MCUs). To demonstrate this technique, two examples of ultra-low-power thermostats are presented, one using the MSP430F1111A MCU and the other using the MSP430F413 MCU. Both examples implement the analog-to-digital conversion using the on-chip comparator and timer of the MCUs.

Code examples and additional information are available from www.ti.com/lit/zip/slaa129.

Contents

Trademarks

1 Introduction

Analog-to-digital conversion is critical to any number of microcontroller applications. Real-world signals are often analog, and they must therefore be converted to digital before they can be analyzed by a processor. To make this conversion, various types of analog-to-digital converters (ADCs) can be used. However, if an ADC module is not available on the MSP430 device in question, it is possible to do digitization with the integrated comparator and timer, using a technique known as slope analog-to-digital conversion.

This document describes two thermostat designs that demonstrate the slope analog-to-digital conversion technique using the on-chip comparator and timer of the MSP430 MCU. Each design uses the technique to measure the resistance of an external thermistor. When the software has obtained the digitized temperature value, it compares it to a target "setpoint" value and decides how to adjust the temperature in the room. One design prioritizes low cost, and the other incorporates additional features. This application report describes each design and includes a power analysis for each design.

2 Description of Slope Analog-to-Digital Technique Using MSP430 MCUs

2.1 Overview

Slope analog-to-digital conversion is an analog-to-digital conversion technique that can be implemented with a comparator rather than a standalone ADC module or device. The technique is based on the charging/discharging of a capacitor with a known value. The number of clock cycles necessary to discharge the capacitor is then counted. Longer discharge times indicate larger voltages. The voltage is derived from the discharge time using the standard equation for capacitor discharge.

In addition to digitizing voltages, a variation of the technique can be used to measure resistance. This is valuable in measuring any component that can have varying resistance, such as potentiometers and various types of transducers. Unlike voltage measurement, where the key relationship is between voltage and time while the resistance is constant, the key relationship in resistance measurement is between resistance and time, while the initial voltage remains constant. The R-t relationship is linear, which means the calculation is easier and less-costly to implement in a microcontroller than for the exponential V-t relationship.

Figure 1 shows the hardware configuration of a slope analog-to-digital resistance-measurement implementation using the MSP430 MCU. This circuit measures the resistance of R_{sens} by discharging capacitor C_m through it. The discharge through resistor R_{ref} is also measured and used in Section 2.2.

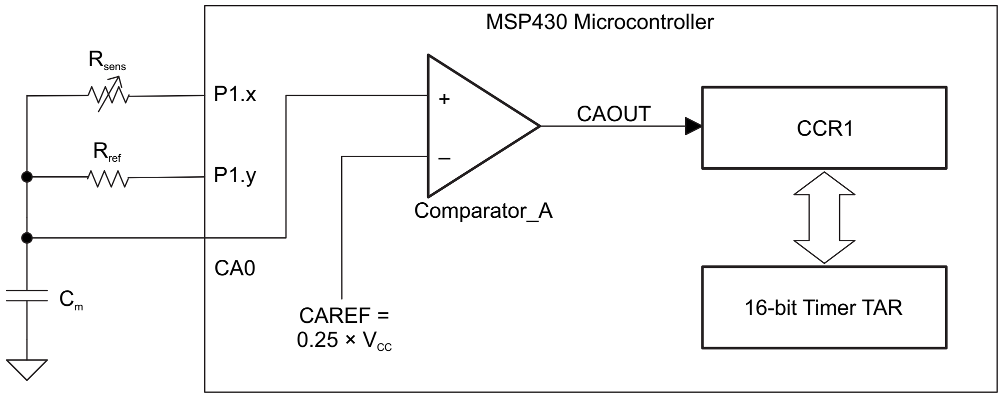


Figure 1. Measurement of Resistors

To measure the resistor value R_{sens}, capacitor C_m is first charged to the digital I/O high voltage ($V_{OH} \approx V_{CC}$) by outputting a high on either P1.x or P1.y. After configuring the timer, the capacitor is discharged through R_{sens} to P1.x by outputting a low-level voltage. At the start of capacitor discharge, register TAR is cleared, and the timer is started. When the voltage across capacitor C_m reaches a comparator reference value V_{CAREF} of $0.25 \times V_{CC}$, the negative edge of the comparator output CAOUT causes the TAR value to be captured in register CCR1. This value is the discharge time interval t_{sens}. The process is repeated for the reference resistor R_{ref}, which is used to translate t_{sens} into the resistor value R_{sens}. C_m is given time t_c to charge, where t_c is between 5τ (for 1%) and 7τ (for 0.1%), where $\tau = R_{ref} \times C_m$. The value within this range depends on the accuracy required.

Figure 2 shows the voltage V_{Cm} across capacitor C_m during the two measurements.

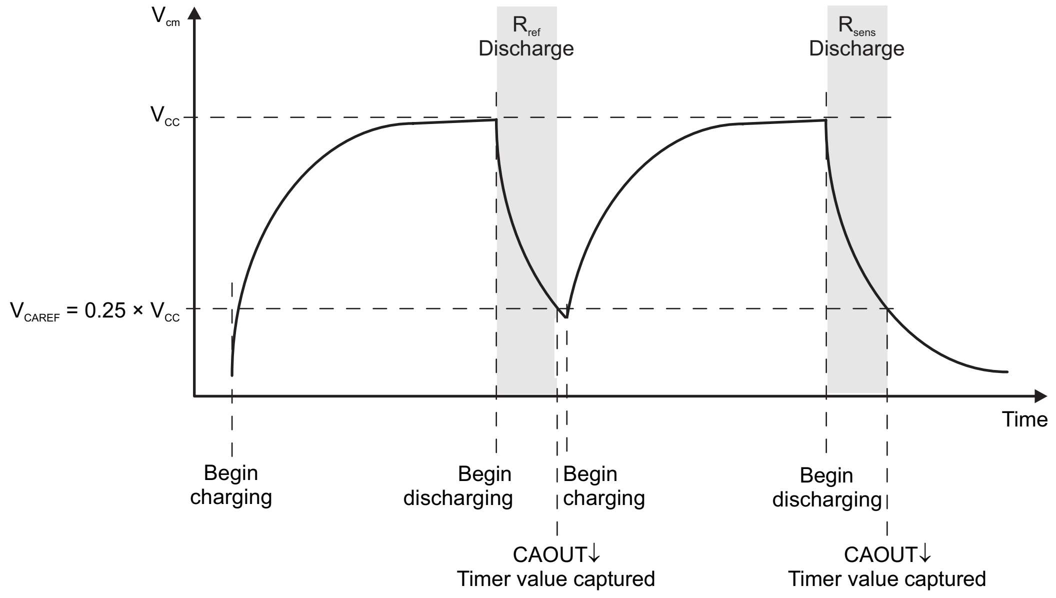


Figure 2. Voltage at C_m During Resistance Measurement

High accuracy can be obtained with this method, and it may broaden the number of MSP430 devices that fit a given application, because no ADC module is required. However, the tradeoff for this is that the sampling rate is limited due to the relatively long charge and discharge time, and there is additional code and execution cycles necessary to perform the measurement and calculation. Section 2.2 shows that the execution cost of the calculation (and therefore power consumption cost) comes in the form of one multiply and one divide per measurement.

2.2 *Calculation for Slope Measurement of a Resistance*

From basic circuit theory, the voltage across a capacitor discharging through a resistor is calculated as Equation 1.

$$V(t) = V_0 e^{-t/RC} \tag{1}$$

Given the following known values produces Equation 2.

$V(t) = V_{CAREF}$
$V_0 = V_{OH} \approx VCC$
$R = R_{sens}$
$C = C_m$

$$V_{CAREF} = V_{CC} e^{-t_{sens}/R_{sens}C_m} \tag{2}$$

The only unknown in Equation 2 is R_{sens}, which means R_{sens} can be calculated with this equation. However, it depends highly on the accuracy of C_m, which is a problem, because most capacitors have relatively wide tolerance. Another problem with Equation 2 is that it involves an exponential calculation, which is expensive either in execution cycles (if calculating) or memory (if using a lookup table).

One way to solve the C_m problem would be to use a very narrow-tolerance capacitor. Another way to solve the problem is to measure the discharge of a reference resistor R_{ref} attached to the same capacitor, and to use the resulting value to normalize R_{sens}. Not only does this provide an opportunity to remove C_m from the equation, it also removes the exponential component.

If C_m is discharged through a resistor R_{ref}, the equation would be identical to Equation 2 except with the new values of R_{ref} and t_{ref}. Because V_{CAREF} is equal in both cases, the right-handed portions of the R_{sens} and R_{ref} equations can be set equal to each other. When this compound equation is simplified, it produces a simple ratio:

$$\frac{R_{sens}}{t_{sens}} = \frac{R_{ref}}{t_{ref}} \tag{3}$$

Therefore, if the capacitor discharge timing sequence is performed on both R_{sens} and R_{ref}, with R_{ref} being a high-precision resistor of known value in the same range as R_{sens}, then the only unknown in equation 3 is R_{sens}. This allows a calculation of R_{sens} that is more accurate than Equation 2. The Equation 3 calculation also requires less power, because it can be performed with only a multiply and a divide. This is the calculation used in most slope analog-to-digital resistance-measurement applications.

3 Thermostat Implementation Using Slope Analog-to-Digital Conversion

Section 3.1 and Section 3.2 describe examples of using the method in Section 2 to build a thermostat. In both examples, the value of a thermistor is acquired using slope analog-to-digital conversion, which is then matched with a temperature value using a lookup table. Action is then taken to output the data to the user.

Section 3.1 describes a low-cost design that uses the MSP430F1111A. In this design, the user sets the desired temperature using a potentiometer. LEDs momentarily illuminate to show that the unit is cooling or heating. Slope analog-to-digital conversion is also used to determine the potentiometer setting.

Section 3.2 describes a design that offers advanced features and uses the MSP430F41x. The design includes an LCD to display the current temperature, setpoint temperature, or time, depending on the mode. Buttons are used to change modes and set the values.

3.1 Low-Cost Thermostat Implementation Using the MSP430F1111A

Figure 3 shows the block diagram of a simple thermostat using the MSP430F1111A. The software is designed around a main loop that executes once every five seconds, prompted by an interrupt.

Between the interrupts, the device stays in low power mode 3 (LPM3) and draws very low current (see the power analysis in Section 3.1.3).

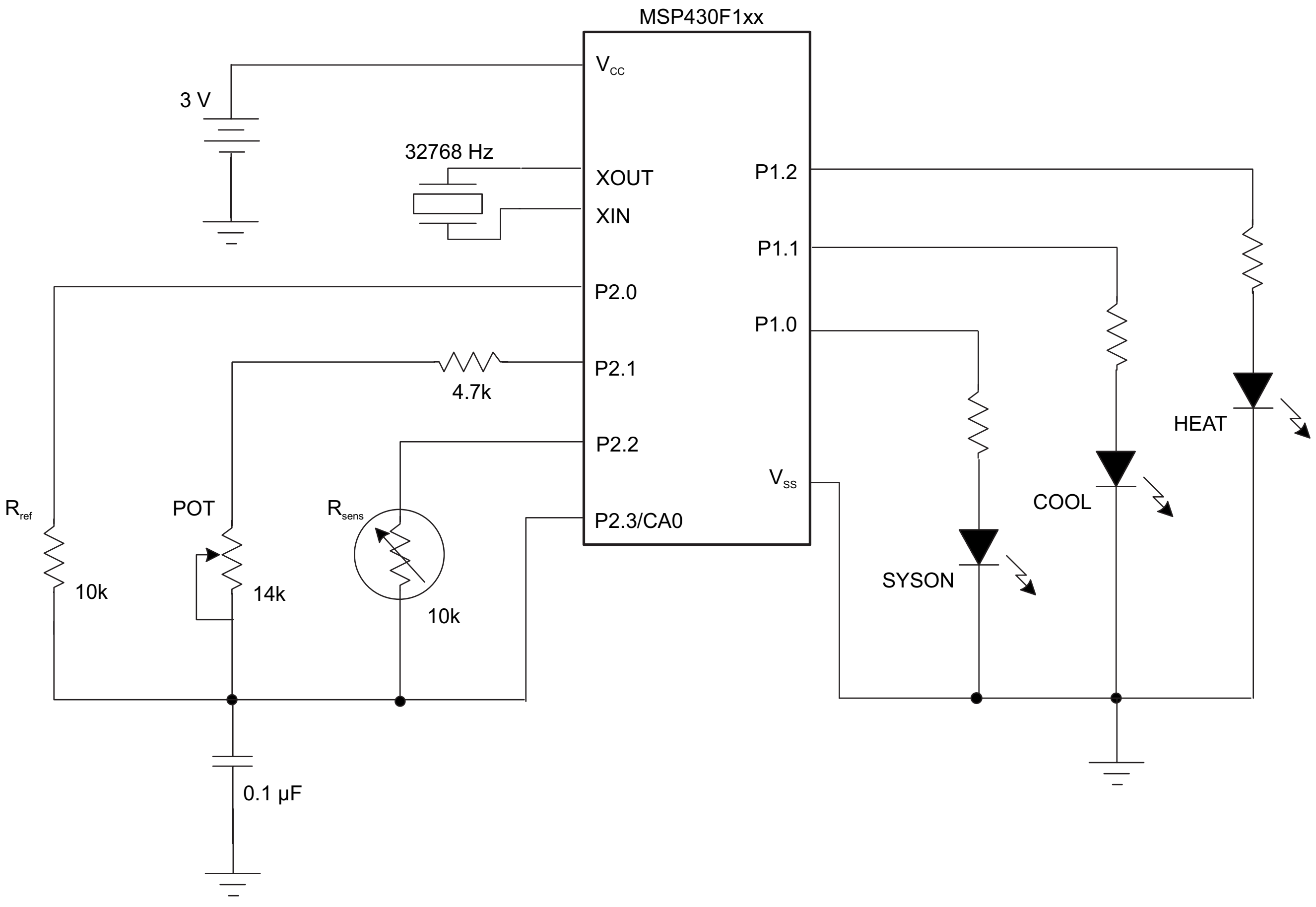


Figure 3. Low-Cost Thermostat Block Diagram

3.1.1 User Operation

The user selects the setpoint temperature using the potentiometer. The thermostat function then indicates whether the system needs to cool or heat the environment to match that temperature, using the "cool" and "heat" LEDs. At the same time, a "system on" LED is driven. If the current temperature matches the setpoint temperature, only the "system on" LED is driven. In theory, the "heat" and "cool" LEDs could be used to drive heating and cooling equipment. The LEDs illuminate for half a second.

In this design, the low end of the setpoint potentiometer is assigned to be 60°F, and the high end is assigned to 95°F. This can be user-defined in software.

There is no LCD display, so the only way to obtain the specific temperature value is to use a debugging environment to obtain the value from within the code. However, you can experiment with the device by locating the setpoint value (at which only the "system on" LED illuminates), then applying heat or cold to the thermistor and watching the resulting changes in the LEDs.

3.1.2 Design Description

A flowchart for the main loop, discharge measurement, and interrupt service routines for the low-cost implementation are shown in Figure 4, Figure 5, and Figure 6, respectively.

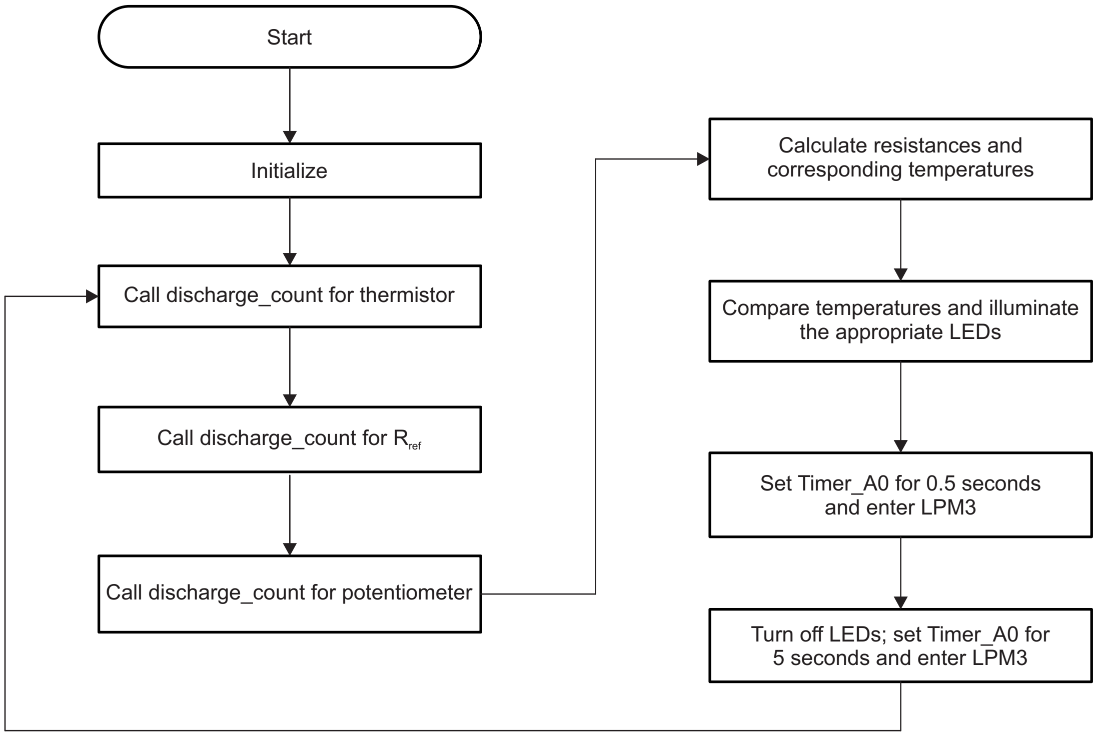

Start
Initialize
Call discharge_count for thermistor
Call discharge_count for R_{ref}
Call discharge_count for potentiometer
Calculate resistances and corresponding temperatures
Compare temperatures and illuminate the appropriate LEDs
Set Timer_A0 for 0.5 seconds and enter LPM3
Turn off LEDs; set Timer_A0 for 5 seconds and enter LPM3

Figure 4. Main Loop Flowchart for Low-Cost Implementation

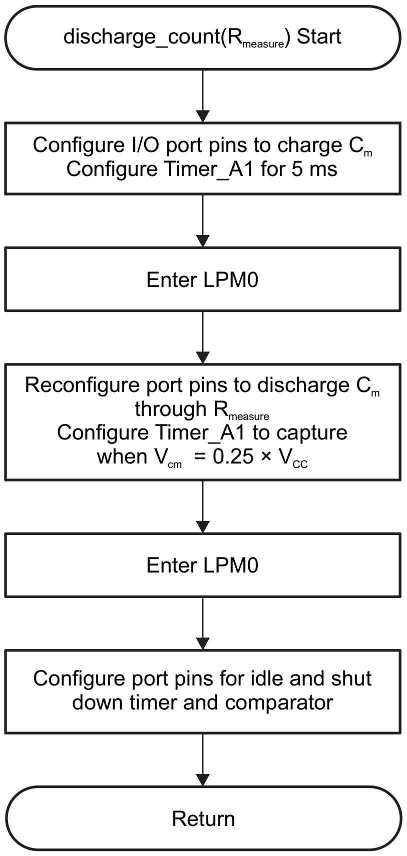

discharge_count($R_{measure}$) Start
Configure I/O port pins to charge C_m
Configure Timer_A1 for 5 ms
Enter LPM0
Reconfigure port pins to discharge C_m through $R_{measure}$
Configure Timer_A1 to capture when V_{cm} = 0.25 × V_{CC}
Enter LPM0
Configure port pins for idle and shut down timer and comparator
Return

Figure 5. Discharge Measurement Flowchart for Low-Cost Implementation

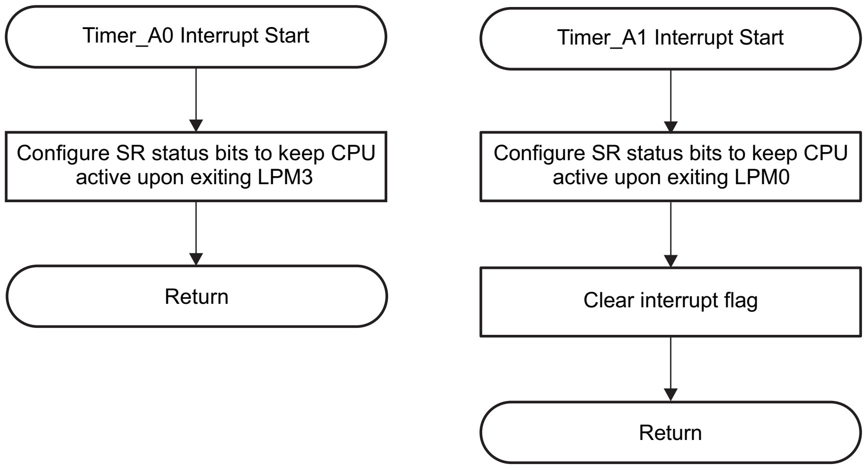

Timer_A0 Interrupt Start
Configure SR status bits to keep CPU active upon exiting LPM3
Return
Timer_A1 Interrupt Start
Configure SR status bits to keep CPU active upon exiting LPM0
Clear interrupt flag
Return

Figure 6. ISR Flowcharts for Low-Cost Implementation

The MSP430 MCU enters sleep mode several times:

- For five seconds between main loop runs
- For half a second while the LEDs illuminate
- While the capacitor is charging or discharging

In each case, the Timer_A module is used. For the first two cases, ACLK is the source clock, and therefore the sleep mode is LPM3. For the last case, SMCLK is used (derived from the DCO) because of the higher frequency and greater slope ADC accuracy. Therefore, LPM0 is used in this case. A different CCR register is used in each case: CCR0 for LPM3 and CCR1 for LPM0. In the interrupt service routines, the respective low-power mode bits are cleared, so that when the ISR is exited, the device remains active and returns to operation at the code location where it was put to sleep.

Every time the main loop runs, the MSP430 MCU calculates the resistance value of the thermistor and potentiometer using the discharge times of R_{sens} and R_{ref} as described in Section 2. These resistance values are then converted to temperatures, using two different methods described in the following paragraphs. All three resistor values are in the range of 10 kΩ: the reference resistor is a fixed precise 10-kΩ resistor, and the possible resistance values of both the thermistor and the setpoint resistor center around 10 kΩ. Because all three resistors use the same capacitor in the RC discharge measurement, assigning similar resistance values results in discharge times within the same range. This in turn allows use of the same timer clock setup for each measurement and simple comparison of timer values.

The thermistor in this design is a BC Components 2322-640-54103, a negative temperature coefficient (NTC) thermistor device.

Conversion of the thermistor and setpoint potentiometer resistance values to temperature values use different approaches. For the thermistor, data is provided by the manufacturer to correlate resistance and temperature. For most NTC thermistors, the data is provided in 5°C increments. For the purposes of a residential thermostat, the resistance values for the temperatures between these points have been interpolated linearly in one-degree increments. This data is represented in the software as a table, indexed by one-degree values. The software determines the index at which the resistance value falls, and adds this to the minimum temperature value (60°F) to produce the current temperature.

In contrast, the setpoint potentiometer resistance is converted with an equation that seeks to span the temperature range (65°F to 90°F) across the resistance range of the circuit (4.7 kΩ to 14.7 kΩ). The resistance range is evenly distributed among the available number of degrees (26). With 65° being represented by the bottom of the range, this leaves 25 segments to be represented by 10 kΩ. There are 25 400-Ω segments in this range. Keeping in mind that all resistances within the program code are represented as actual resistance divided by 100, a temperature value between 65°F to 90°F can be reached by subtracting 47 (4.7 kΩ) from the potentiometer resistance and dividing by 4 (400 Ω). For example, a potentiometer resistance of 14.7 kΩ is represented within the program as 147; subtracting 47 and dividing by 4 results in 90°F. Note that the potentiometer value was carefully chosen so that the divisor would be a value of 2^n. This allows the MCU to perform division by right-shift instructions, significantly reducing execution cycles and power draw.

To avoid the potential for rapid alteration between heat and cool when the current temperature is near the setpoint, a ±1° error is allowed when determining whether to "heat" or "cool". If the current temperature is not within this range of the setpoint temperature, the appropriate heat or cool LED is driven.

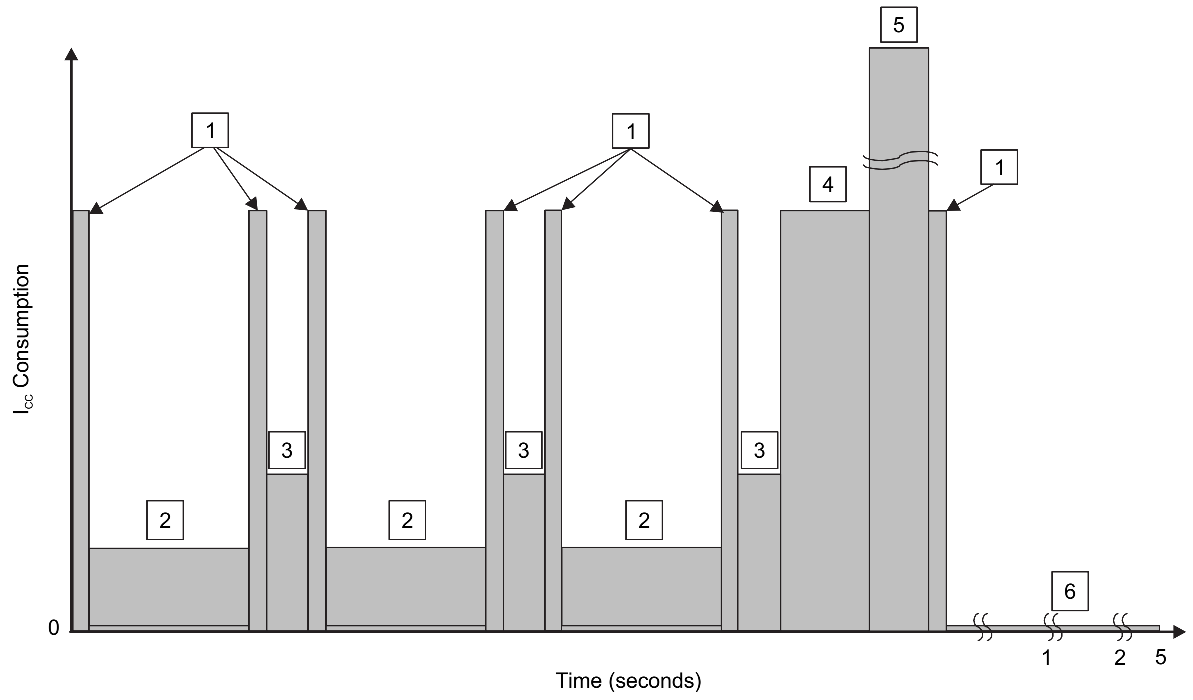
An interrupt period of five seconds is sufficient because room temperature does not change quickly. This has a significant positive impact on power consumption; for example, the battery lasts approximately twice as long as it would if the measurement were performed every 2.5 seconds. A tradeoff to consider is that the user must wait longer for changes to take effect. This is not a problem when temperature is input through a switch (as in the advanced implementation in Section 3.2), because the switch drives an interrupt that wakes the device. In contrast, a rotation applied to the potentiometer does not generate an interrupt.

When the resistors are not being measured, P2.1 is driven high, which also drives the other resistor port pins (set as inputs) high. Floating inputs can cause the device to draw current in the hundreds of µA. By setting all port pins as either outputs or inputs that are pulled high or low, this situation is prevented.

3.1.3 Power Analysis

Figure 7 shows the power draw of the low-cost implementation. Time periods of similar lengths and current levels are given common labels. Table 1 lists the values associated with those labels, and Table 2 lists the final power calculation. Leakage current through the capacitors, diode, and digital I/Os are assumed to be negligible. The values shown represent a typical run through a main loop cycle; each run depends on the jumps and conditions taken, and in particular the varying values of the resistors, which can affect the multiply and divide functions.

The values in the tables are for the assembly version of the code. The assembly code is more power-efficient, mostly because of optimizations in the multiply and divide functions.



NOTE: Drawing is not to scale.

Figure 7. Power Draw Over Time on the Low-Cost Implementation

Table 1. Label Definitions for Low-Cost Implementation

Label	Activity Description	Duration	Components Drawing Power	Current Draw (TYP)
1	Active-mode main loop execution	51 µs[(1)] (each)	CPU active	300 µA
2	Capacitor charging	5.000 ms (each)	CPU in LPM0	55 µA
3	Capacitor discharging	1.400 ms (each)	Comp_A	45 µA
			CPU in LPM0	55 µA
			Total	100 µA
4	Main loop processing discharge data	1.414 ms	CPU active	300 µA
5	Drive LEDs	500 ms	LED outputs	18036 µA
			CPU in LPM0	55 µA
			Total	18091 µA
6	Sleep	5000 ms	CPU in LPM3	1.6 µA

(1) Values of each occurrence vary slightly; this is the average value of each segment.

Table 2. Power Draw Calculation

Label	Instances	Current (TYP)	Duration per Instance	Total Time	µA-ms
1	7	300 µA	0.051 ms	0.357 ms	107.100
2	3	55 µA	5.000 ms	15.000 ms	825.000
3	3	100 µA	1.400 ms	4.200 ms	420.000
4	1	300 µA	1.414 ms	1.414 ms	424.200
5	1	18091 µA	500.000 ms	500.000 ms	9045500.000
6	1	1.6 µA	5000.000 ms	5000.000 ms	8000.000
With LED					
				5.521 s	9.055 mA-s
					1.640 mA/s
Without LED					
				5.021 s	9.776 µA-s
					1.947 µA/s

The LED is the most significant source of current draw and has a major affect on the overall average. To show the low-power performance of the system with and without the LED, the data is calculated in both ways in Table 2. To obtain the "without LED" calculation, omit segment 5.

3.2 Advanced Thermostat Implementation Using the MSP430F413

The circuit in Figure 8 is based on the MSP430F413 and features an LCD display, a real-time clock, and pushbutton setpoint input. The main loop for this implementation runs once per second, prompted by an interrupt. Between interrupts, the devices is in LPM3 mode, resulting in very low power operation (see the power analysis in Section 3.2.3).

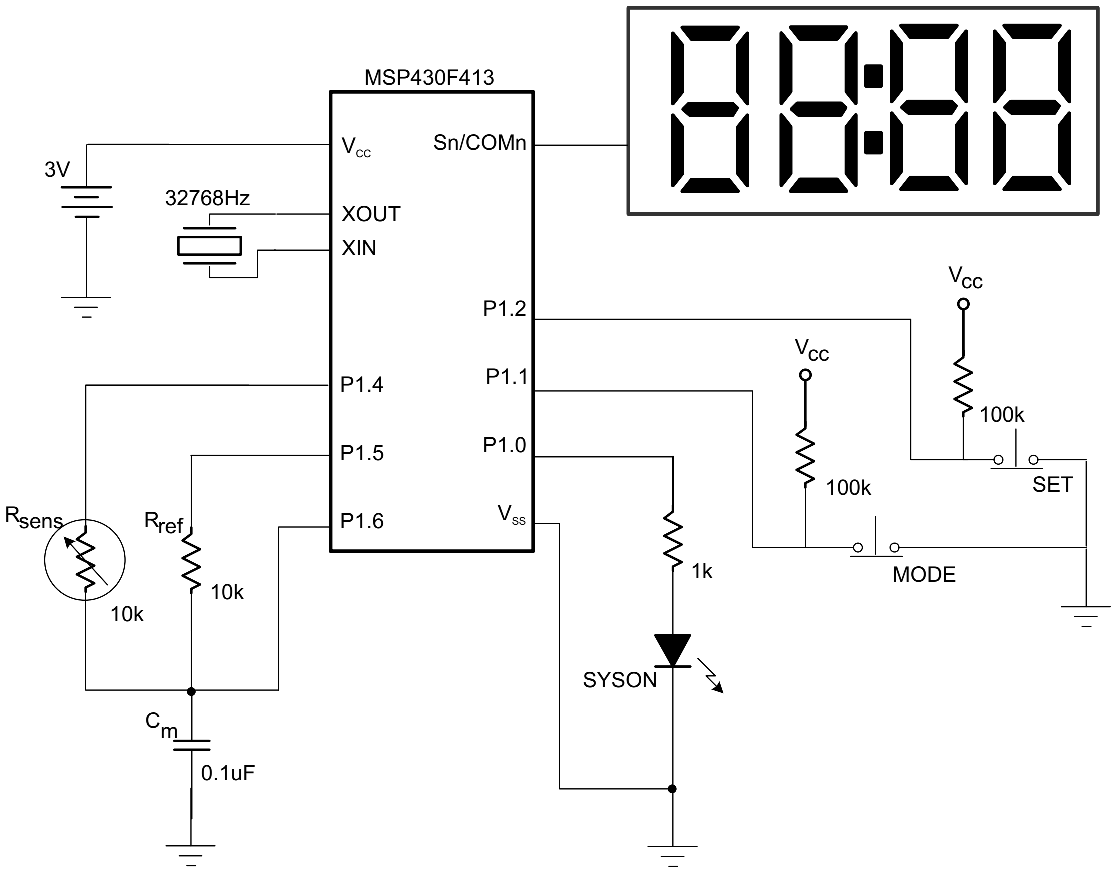


Figure 8. Advanced Thermostat Block Diagram

3.2.1 User Operation

The device can operate in 4 modes, selected by the "mode" switch:

- *Thermostat*. Measures the current temperature and displays it on the LCD. If the setpoint temperature is less than the current temperature, the LED is driven to indicate the activation of "cool".
- *Setpoint*. Allows the user to select the target temperature using the LCD and the "set" switch. Press the "set" switch repeatedly to raise the setpoint temperature. The setpoint temperature rolls back to 60 when 95 has been reached. If the setpoint upon leaving this mode is different than the one in place when the mode was entered, then the new value is written to flash memory between 0x1000 and 0x10FF (information memory).
- *Time*. Displays the real-time clock on the LCD. The time can be changed by pressing the "set" switch; each press of the switch increments the time by one minute. After ten seconds of inactivity, the mode automatically reverts to "thermostat" mode.
- *Seconds*. Counts seconds and displays them on the LCD.

Press the "mode" switch again to rotate through the sequence.

3.2.2 Design Description

Figure 9 shows a flowchart for the main loop. Figure 10 shows the interrupt service routines. The discharge_count function is the same as for the low-cost implementation in Figure 5.

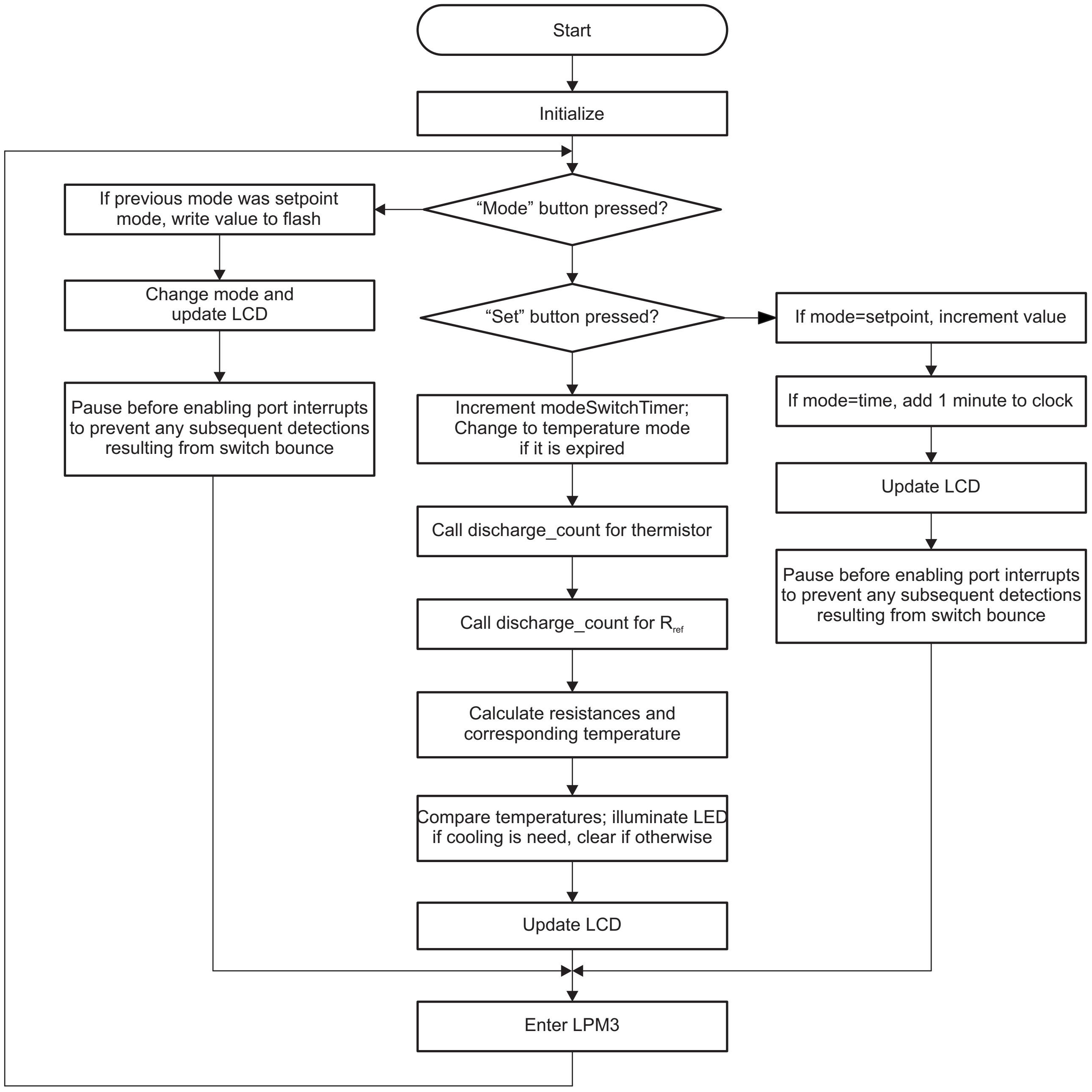

Start
Initialize
"Mode" button pressed?
If previous mode was setpoint mode, write value to flash
Change mode and update LCD
Pause before enabling port interrupts to prevent any subsequent detections resulting from switch bounce
"Set" button pressed?
If mode=setpoint, increment value
If mode=time, add 1 minute to clock
Update LCD
Pause before enabling port interrupts to prevent any subsequent detections resulting from switch bounce
Increment modeSwitchTimer; Change to temperature mode if it is expired
Call discharge_count for thermistor
Call discharge_count for R_ref
Calculate resistances and corresponding temperature
Compare temperatures; illuminate LED if cooling is need, clear if otherwise
Update LCD
Enter LPM3

Figure 9. Main Loop Flowchart for Advanced Implementation

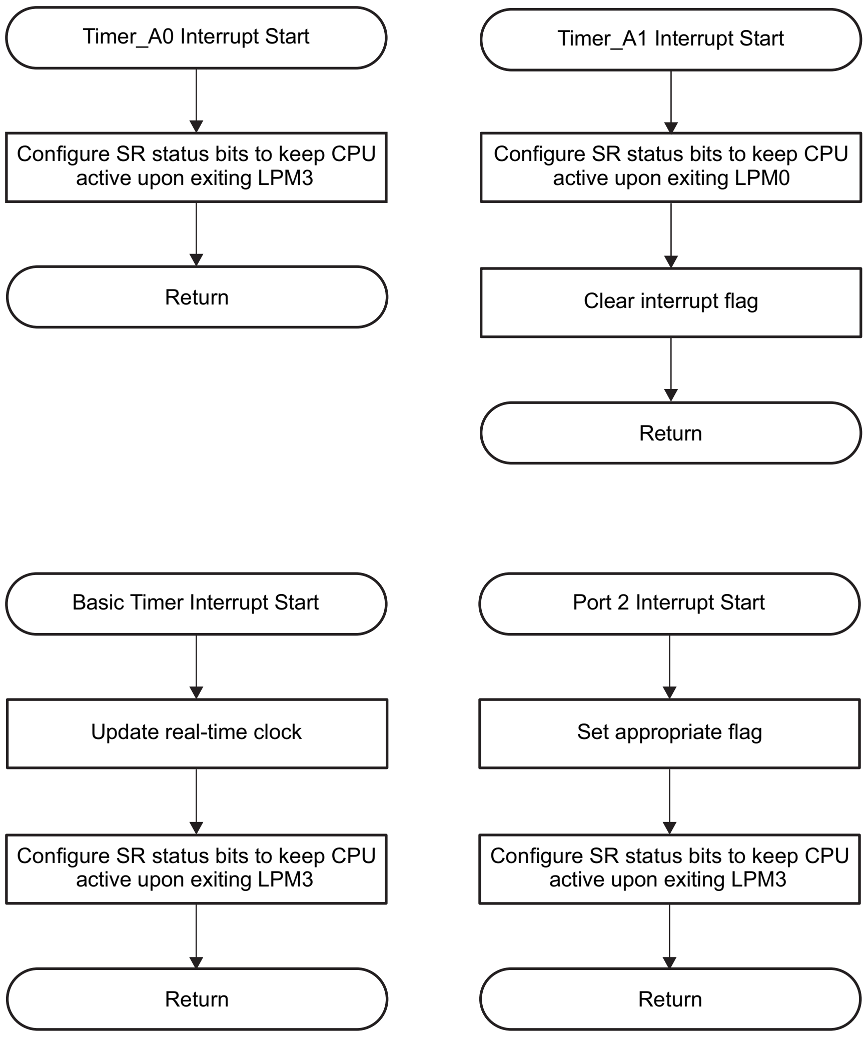

Timer_A0 Interrupt Start
Configure SR status bits to keep CPU active upon exiting LPM3
Return
Timer_A1 Interrupt Start
Configure SR status bits to keep CPU active upon exiting LPM0
Clear interrupt flag
Return
Basic Timer Interrupt Start
Update real-time clock
Configure SR status bits to keep CPU active upon exiting LPM3
Return
Port 2 Interrupt Start
Set appropriate flag
Configure SR status bits to keep CPU active upon exiting LPM3
Return

Figure 10. ISR Flowcharts for Advanced Implementation

Unlike the low-cost thermostat example, which wakes up every five seconds, this example wakes up every second. The main reason for this is that the real-time clock needs to be updated every second. The device also wakes between the 1-second intervals if the user presses either of the push-button switches.

The 1-second interrupt is generated by one of the counters in the Basic Timer module, which is powered by the 32-kHz watch crystal and continues to count while the device is in LPM3. This differs from the MSP430F1111A implementation in Section 3.1, because the F1xx family does not contain the Basic Timer. Using the Basic Timer frees Timer_A or Timer_B, if present, for functions that are more complex.

The Basic Timer interrupt service routine clears the low-power mode bits to that when the ISR is exited, the device remains active and starts the main loop. Because the loop is executed for multiple purposes (to update the clock, to update the display, to measure the temperature, to handle the push buttons) it is necessary to check the state of the device within the main loop to determine what action needs to be taken. When all appropriate actions are completed, the device enters LPM3 until the next 1-second interrupt or press of a button.

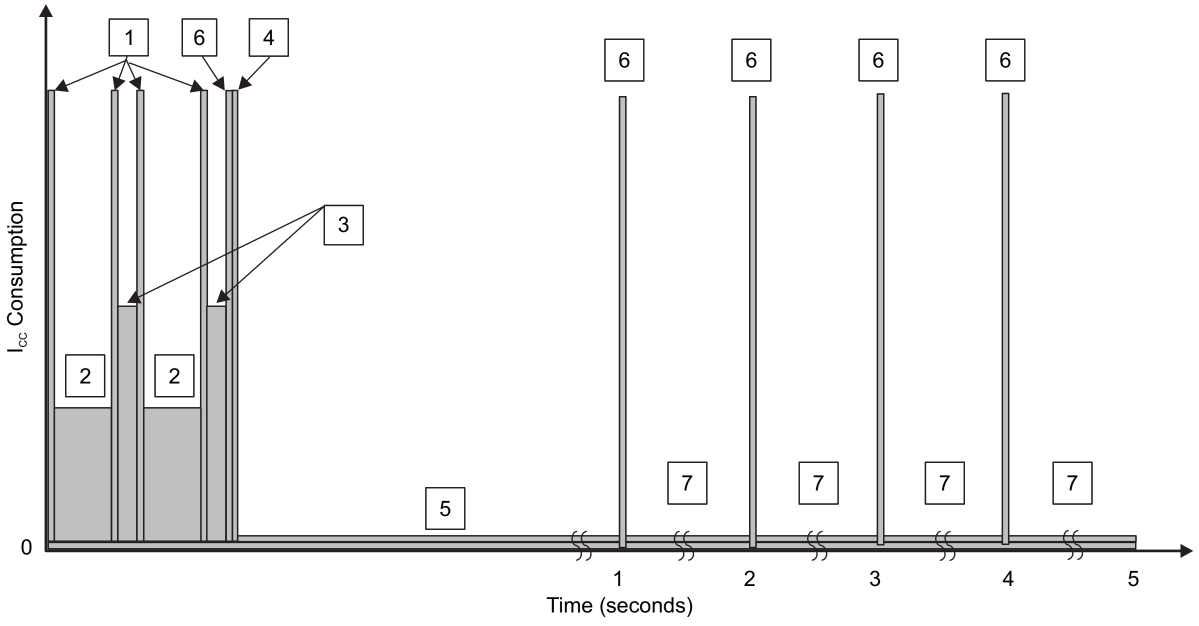
While the circuit is in setpoint mode, the symbol "-|" is displayed on the LCD followed by the two-digit temperature setpoint. The symbol "-|" was chosen because the 3.5-digit LCD does not have space to display "SP" next to the 2-digit value of the setpoint.

When the application exits the setpoint mode, it writes the new setpoint value to the information memory segment of flash. The application does not write to a single location each time, it increments across the 256 bytes of flash and, when information memory is full, erases and starts at the beginning. This procedure extends the operating life of the flash. The MSP430 flash memory can be erased 100000 times at 25°C. This means that the setpoint could be changed reliably up to 25.6 million times.

3.2.3 Power Analysis

Figure 11 shows the power draw of the advanced implementation. Time periods of similar lengths and current levels have common labels. Table 3 lists the values associated with those labels, and Table 4 lists the final power calculation. These calculations assume that leakage current in the capacitors and digital I/Os is negligible. The values represent a typical run through a main loop cycle. The power consumption of each run depends on the jumps and conditions taken, and in particular the varying values of the resistors, which can affect the multiply and divide functions.

The values in the tables are for the assembly version of the code. The assembly code is more power efficient, mostly because of optimizations in the multiply and divide functions.



NOTE: Drawing is not to scale.

Figure 11. Power Draw Over Time on the Advanced Implementation

Table 3. Label Definitions for Advanced Implementation

Label	Activity Description	Duration	Components Drawing Power	Current Draw (TYP)
1	Active-mode main loop execution	67 µs[(1)] (each)	LCD display	1 µA
			CPU active	300 µA
			Total	301 µA
2	Capacitor charging	5.000 ms (each)	LCD display	1 µA
			CPU in LPM0	92 µA
			Total	93 µA
3	Capacitor discharging	1.400 ms (each)	Comp_A	45 µA
			LCD display	1 µA
			CPU in LPM0	92 µA
			Total	138 µA

(1) Values of each occurrence vary slightly; this is the average value of each segment.

Table 3. Label Definitions for Advanced Implementation (continued)

Label	Activity Description	Duration	Components Drawing Power	Current Draw (TYP)
4	Main loop processing discharge data	730 µs	LCD display	1 µA
			CPU active	300 µA
			Total	301 µA
5	Sleep	985.955 ms	LCD display	1 µA
			CPU in LPM3	0.9 µA
			Total	1.9 µA
6	Update clock and display every second	247 µs (each)	LCD display	1 µA
			CPU active	300 µA
			Total	301 µA
7	Sleep	999.753 ms (each)	LCD display	1 µA
			CPU in LPM3	0.9 µA
			Total	1.9 µA

Table 4. Power Draw Calculation

Label	Instances	Current (TYP)	Duration per Instance	Total Time	µA-ms
1	4	301 µA	0.067 ms	0.268 ms	80.668
2	2	93 µA	5.000 ms	10.000 ms	930.000
3	2	138 µA	1.400 ms	2.800 ms	386.400
4	1	301 µA	0.305 ms	0.305 ms	91.805
5	1	1.9 µA	985.955 ms	985.955 ms	1873.315
6	5	301 µA	0.672 ms	3.360 ms	1011.360
7	4	1.9 µA	999.328 ms	3997.312 ms	7594.893
				5 s	11.978 µA-s
					2.394 µA/s

In the advanced implementation, the LED is either on continuously or not at all. Therefore, the power calculation in Table 4 remains unchanged when the LED is off. If the LED is on, the current draw in every segment type is increased by 3 mA, the per-second average.

4 Additional Reading

The TI Tech Note Single-Slope Analog-to-Digital Conversion Technique Using MSP430 MCUs is based on this application report and describes the implementation of a slope ADC on a low-cost 0.5KB MSP430FR2000 microcontroller. The code provided with the Tech Note can easily be ported to other MSP430 FRAM microcontrollers for other thermostat implementations.

5 References

1. MSP430x41x Mixed Signal Microcontroller
2. MSP430x11x1, MSP430F11x1A Mixed Signal Microcontrollers
3. MSP430x1xx Family User's Guide
4. MSP430x4xx Family User’s Guide
5. Economic Voltage Measurement with the MSP430 Family
6. MSP430FR21xx, MSP430FR2000 Mixed-Signal Microcontrollers
7. MSP430FR4xx and MSP430FR2xx User's Guide
8. Single-Slope Analog-to-Digital Conversion Technique Using MSP430 MCUs

Revision History

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

Changes from January 30, 2006 to August 3, 2018	**Page**
• Editorial and formatting changes throughout document	1
• Added Section 4, *Additional Reading*	15
• Added items in Section 5, *References*	15